

VT Gerçeklenmesi Ders Notları-

#2

Remote: Kullanıcıdan gelen JDBC isteklerini karşılar.

Planner: SQL ifadesi için işleme planı oluşturur ve karşılık gelen ilişkisel 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.

- ◆ Veri tabanı sunucusunun iç işleyişi
- ◆ Eğitimsel Veri Tabanı: **SimpleDB** mimarisi
- ◆ Dosya Yönetimi

ÖZET

- VT dersine genel bakış
- Yeni bir açılım: VT iç işleyişi/gerçekleme
- Mevcut çalışmalar
- SimpleDB
 - Ana modülleri (Kullanıcı, Sunucu, Eklentiler)
- SimpleDB üzerinde proje konuları

Veri Tabanı dersi

- Veri tabanı kullanımı

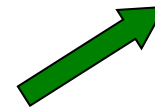
- Tasarım
- İlişkisel algebra
- SQL
- Veri tabanı uygulamaları

Web-tabanlı uygulamalar

Veri madenciliği ...



Kilitleme, çok-
versiyonlu
eşzamanlılık...



Spanned kayıt
organizasyonu, değişken
uzunluklu kayıt org.



B+tree, hashing,
Çok-boyutlu veri
indeksleme



- Veri tabanı sisteminin iç işleyişi

- Eşzamanlılık kontrolü (*concurrency*)
- Kurtarma (*recovery*)
- Kayıt organizasyonu (*record management*)
- İndeksleme (*indexing*)
- Sorgu işleme (*query processing*)



En iyi sorgu ağacının bulunması

Veri tabanı sisteminin iç işleyişinin anlaşılması

- “Basit” bir veri tabanı sistemi uygulamasının yazılması
 - Çok basit
- Açık-kaynak ticari veri tabanı sistemlerinin anlaşılması ve yeni kod eklenmesi
 - Ailamaki, A., and Hellerstein, J. “*Exposing Undergraduate Students to Database System Internals.*” *ACM SIGMOD* (2003),
- Eğitimsel amaçlı bir veri tabanı sisteminin anlaşılması ve yeni kod eklemesi
 - Swart, G. “*MinSQL: A Simple Componentized Database for the Classroom.*” *ACM 2003*
 - *MiniBase* : Ramakrishnan, R. and Gehrke, J. *Database Management Systems (Third Edition)*. McGraw-Hill, Boston, 2003.
 - “*SimpleDB: A Simple Java-Based Multiuser System for Teaching Database Internals*”. *Proc. ACM SIGCSE Conference on Computer Science Education*, March 2007. (<http://www.cs.bc.edu/~sciore/simplydb/intro.html>)

SimpleDB

- Çok-kullanıcılı, işlemsel (*transactional*) Java tabanlı eğitimsel bir veri tabanı
- Hedef:
 - Okunabilir,
 - Kullanımı kolay,
 - Kolayca değiştirilebilir olması
- Küçük olarak bilinen *DerbyDB*'nin yaklaşık 1/100'ü
- Yaklaşık 3500 satır kod, 85 class, 12 adet paket
 - Kullanıcı-tarafı : JDBC arayüzleri ve sürücünün gerçekleşmesi
 - Temel sunucu-tarafı: iskelet yapı
 - Eklentiler: sorgu işleme verimini arttıran algoritmalar.

SimpleDB: Kullanıcı-tarafı

```
String qry = “select sal from EMP where dept = ‘sales’ ”;
```

```
Driver d = new SimpleDriver();
```

```
Connection c =
```

```
    d.connect(“jdbc:simpledb://ce.yildiz.edu.tr/MYDB”);
```

```
Statement s = c.createStatement();
```

```
ResultSet r = s.executeQuery(qry);
```

```
while (r.next())
```

```
    System.out.println(r.getInt(“sal”));
```

```
r.close();
```

```
c.commit();
```

SimpleDB: Temel sunucu-tarafı:

Remote: Kullanıcıdan gelen JDBC isteklerini karşılar.

Planner: SQL ifadesi için işleme planı oluşturur ve karşılık gelen ilişkisel 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.

Temel sunucu-tarafı: “Remote”

- 2 tür veri tabanı kullanımı:



Doğrudan erişim (gömülü veritabanı)	
SimpleDB	←
Transaction	←
Planner	←
Scan	←
Schema	←

Sunucu üzerinden (<i>sunucu vt</i>)	
Driver	←
Connection	←
Statement	←
ResultSet	←
ResultSetMetaData	←

Karşılaştırma *(gömülü erişim, istemci sunucu erişimi)*

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

Planner planner = SimpleDB.planner();
String qry = "select sname, gradyear from student";
Plan p = planner.createQueryPlan(qry, tx);
Scan s = p.open();

while (s.next())
    System.out.println(s.getString("sname") + " " +
                      s.getInt("gradyear"));

s.close();
tx.commit();
```

(a) Using an embedded database system

```
Driver d = new SimpleDriver();
String url = "jdbc:simpledb://localhost";
Connection conn = d.connect(url, null);

Statement stmt = conn.createStatement();
String qry = "select sname, gradyear from student";
ResultSet rs = stmt.executeQuery(qry);

while (rs.next())
    System.out.println(rs.getString("sname") + " " +
                      rs.getInt("gradyear"));

rs.close();
conn.commit();
```

(b) Using JDBC code

Figure 20-1

Two equivalent application program fragments

Figure 20-1 *(Continued)*

Ödev ve Proje Örnekleri

- Hafıza Yönetimi:
 - Tampon yöneticisi ve farklı sayfa seçim algoritmaları
- Kayıt Yönetimi:
 - Değişken uzunluklu kayıtların organizasyonu
 - Spanned organizasyonlar
 - Homojen olmayan organizasyonlar
- İndeks oluşturma:
 - Adrese dayalı indeks
 - Çok seviyeli indeks
 - Bileşik nitelikler(composite keys)üzerindeki indeksler
- Sıralama:
 - External sorting
- Kilitlenme denetimi ve çözümü
- Yeni ilişkisel algebra operatörlerinin tasarımı ve gerçekleştirilmesi
- Mevcut SQL'in genişletilmesi.
- “PreparedStatement” gerçekleştirilmesi
- Toplanma (*aggragation*), *mergesort* ve tekrarların ayıklanması (*duplicate removal*) gibi Sıralamaya dayalı işlemler için verimli harici sıralama algoritmaları..
- Sorgu eniyileme yöntemleri

Disk erişim arayüzü 1: Blok

- İşletim sistemi, disk detaylarını (*sektor adresleme, sektör büyüklüğü, veri transferi, veri yerleşimi*) uygulama tarafından saklamak için **blok** arayüzü kullanır.
- **Blok**, disk üzerinde “bir veya daha çok bitişik sektörün oluşturduğu byte dizisi” olup;
 - Büyüklüğü işletim sistemi tarafından belirlenir
 - Bütün disk üzerinde sabittir
 - Ana hafıza ile disk arasındaki minimum transfer edilebilen veri miktarıdır.
 - Disk üzerindeki blok numaraları 0’dan başlar.
- Blok içeriğine erişmek disk üzerinde gerçekleşmiyor. Öncelikle bu bloğun ana hafızaya iletilmesi gerektir. İşte ana hafızada 1 blok için ayrılan bölgeye ise **sayfa(page)** adı verilir.
- İşletim sisteminde disk bloklarını kullanan fonksiyonlardan bazıları:
 - *readblock(n,p)*
 - *writeblock(n,p)*
 - *allocate(k,n)*
 - *deallocate(k,n)*

Disk erişim arayüzü 2: **Dosya**

- Kullanıcının diske bakışı: dosyalar ve dosyadaki herhangi bir byte erişmek/değiştirmek
- İşletim sistemindeki **dosya sistemi** (*file system*) bu arayüzü gerçekleştirir.
- Aşağıdaki kod parçası **dosya sistemi arayüzü** ile bilgi erişimi ve değiştirmeyi içeriyor..

```
RandomAccessFile f = new RandomAccessFile("junk", "rws");  
f.seek(7992);  
int n = f.readInt();  
f.seek(7992);  
f.writeInt(n+1);  
f.close();
```

Figure 12-7

Using the file-system interface to the disk

- Yukarıdaki kod parçası ne yapıyor?
- işletim sistemi/**dosya sistemi** arayüzü sayesinde hangi detaylar kullanıcıdan saklanıyor?
 - Dosya erişim izni
 - **Tampon kullanımı**
 - **Adresleme**: Byte adres → mantıksal blok adres → fiziksel blok adres
 - ...



Dosya sisteminde yapılan Extent-based yerleşim, indeksli yerleşim gibi yöntemler....

Bir VTYS hangi erişim arayüzünü kullanmalı?

Blok mu? / Dosya mı?

	Avantajları	Dezavantajları
Blok	<ul style="list-style-type: none">◆ Disk blokları tamamiyle VTYS kontrolünde (Örneğin, sık kullanılan bloklar diskin ortasına yerleştirilebilir, çoğu zaman beraber erişilen bloklar diskte yakın yerlere yerleştirilebilir)◆ Dosya sisteminin bazı kısıtlamalarından kurtulmuş oluyoruz (Örneğin, tablo büyüklüğü İşletim sisteminin destekleyebildiği dosya büyüklüğü ile kısıtlı değil!)	<ul style="list-style-type: none">◆ Sistem gerçekleştirme karmaşıklığı yüksek (örneğin blok erişimi nasıl olacak?, hangi bloklar hangi tablolara ait?)◆ sistem performansının ince ayarı (<i>fine-tuning</i>) oldukça zor.
Dosya	<ul style="list-style-type: none">◆ Gerçeklenmesi oldukça kolay! (her tablo için bir dosya ve dosyadaki kayıtlara erişim dosya sistemi fonksiyonları ile kolayca gerçekleşir)◆ erişim karmaşıklığı çok az	<ul style="list-style-type: none">◆ VTYS, veri yerleşim ve erişim organizasyonlarının verimliliği için blok sınırlarını bilmesi gerekiyor (değişken uzunluklu kayıtları veya spanned organizasyonları düşünün!)◆ VTYS'nin tampon bölgeyi kullanımı işletim sisteminden çok farklı (ileride anlatılacak...)

- ◆ Çözüm: orta yol!
- ◆ **Veri tabanı tabloları bir veya daha çok dosyada saklanıyor; dosya içeriğine erişim blok seviyesinde (byte değil!)**
- ◆ mantıksal → fiziksel blok dönüşümleri önceki sunudaki gibi İşletim sistemi tarafından gerçekleştiriliyor.
(yani, VTYS'nin veri yerleşiminde bütün diske hakimiyeti yok!)
- ◆ Sistem gerçekleştirmedeki bu orta yol bir çok VTYS'de kullanılmaktadır (MSAccess, Oracle ve **SimpleDB**)

SimpleDB: File Manager

- ◆ SimpleDB dosya yöneticisi (*simplifiedb.file package*),
 - işletim sistemi dosya yöneticisinin hizmetlerini kullanır (*dosya oluşturma/açma/kapama, rastgele erişim, mantıksal/fiziksel blok adres dönüşümü, diskteki blokların bulunması gibi...*)
 - Açılan bir dosyadaki herhangi bir bloğa erişim, bloğun ana hafızaya getirilmesi, içeriğinin okunup değiştirilmesi ve geriye yazılması...

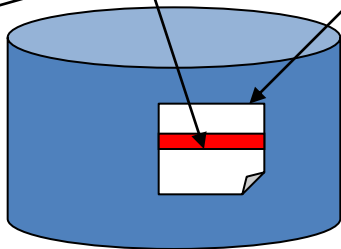
Remote
Planner
Parser
Query
Metadata
Record
Transaction & Recovery
Buffer
Log
File manager

SimpleDB

İşletim sistemi

....
Multiprogramming
Buffer Manager
File Manager

- ◆ Bir **SimpleDB** veri tabanı birden çok dosyadan oluşuyor:
 - Her tablo için 1 dosya
 - Her index için 1 dosya
 - 1 adet log dosyası
 - Birden çok katalog dosyaları



SimpleDB File Manager API ve örnek kullanımı

Block

Blok: bilgi içermiyor! sadece referans,

```
public Block(String filename, int blknum);  
public String  fileName();  
public int     number();
```

Mantıksal blok no

Page

Sayfa: bilgi içeriyor!

```
public Page();  
public int   getInt(int offset);  
public void  setInt(int offset, int val);  
public String getString(int offset);  
public void  setString(int offset, String val);  
  
public void  read(Block blk);  
public void  write(Block blk);  
public Block append(String filename);
```

FileMgr **İşletim sistemi ile bağlantıya geçen birim!**

```
public FileMgr(String dirname);  
public boolean isNew();  
public int     size(String filename);
```

Figure 12-11

The API for the SimpleDB file manager

```
SimpleDB.initFileMgr("studentdb");  
FileMgr fm = SimpleDB.fileMgr();  
  
Block blk1 = new Block("student.tbl", 0);  
Page p1 = new Page();  
p1.read(blk1);  
String sname = p1.getString(46);  
int gy = p1.getInt(38);  
System.out.println(sname + " has gradyr " + gy);  
  
Page p2 = new Page();  
p2.setString(16, "tim");  
p2.setInt(8, 2012);  
Block blk2 = p2.append("student.tbl");
```

Figure 12-12

Using the SimpleDB file manager

Blok, Sayfa sınıflarının gerçeklenmesi:

```
public class Block {
    private String filename;
    private int blknum;

    public Block(String filename, int blknum) {
        this.filename = filename;
        this.blknum = blknum;
    }

    public String fileName() {
        return filename;
    }

    public int number() {
        return blknum;
    }

    public boolean equals(Object obj) {
        Block blk = (Block) obj;
        return filename.equals(blk.filename) &&
            blknum == blk.blknum;
    }

    public String toString() {
        return "[file " + filename + ", block " + blknum + "]";
    }

    public int hashCode() {
        return toString().hashCode();
    }
}
```

Figure 12-13

The code for the SimpleDB class *Block*

```
public class Page {
    public static final int BLOCK_SIZE = 400;
    public static final int INT_SIZE = Integer.SIZE /
        Byte.SIZE;

    public static final int STR_SIZE(int n) {
        float bytesPerChar =
            Charset.defaultCharset().newEncoder().maxBytesPerChar();
        return INT_SIZE + (n * (int)bytesPerChar);
    }

    private ByteBuffer contents =
        ByteBuffer.allocateDirect(BLOCK_SIZE);
    private FileMgr filemgr = SimpleDB.fileMgr();

    public synchronized void read(Block blk) {
        filemgr.read(blk, contents);
    }

    public synchronized void write(Block blk) {
        filemgr.write(blk, contents);
    }

    public synchronized Block append(String filename) {
        return filemgr.append(filename, contents);
    }

    public synchronized int getInt(int offset) {
        contents.position(offset);
        return contents.getInt();
    }

    public synchronized void setInt(int offset, int val) {
        contents.position(offset);
        contents.putInt(val);
    }

    public synchronized String getString(int offset) {
        contents.position(offset);
        int len = contents.getInt();
        byte[] byteval = new byte[len];
        contents.get(byteval);
        return new String(byteval);
    }

    public synchronized void setString(int offset,
        String val) {
        contents.position(offset);
        byte[] byteval = val.getBytes();
        contents.putInt(byteval.length);
        contents.put(byteval);
    }
}
```

Contents, OS tampon bölgesi içinde

*Sayfanın sınırını
taşma kontrolünü
yapmadık?*

**Neden “sync” kullanılmış? Kullanılmasaydı neler
olabilirdi?**

Figure 12-14 (Continued)

FileMgr sınıfının gerçekleştirilmesi

Bir SimpleDB instance,
sadece 1 FileMgr'a sahiptir.
Bu FileMgr'ı
server.SimpleDB oluşturur.

```
public class FileMgr {
    private File dbDirectory;
    private boolean isNew;
    private Map<String,FileChannel> openFiles =
        new HashMap<String,FileChannel>();

    public FileMgr(String dbname) {
        String homedir = System.getProperty("user.home");
        dbDirectory = new File(homedir, dbname);
        isNew = !dbDirectory.exists();

        // create the directory if the database is new
        if (isNew && !dbDirectory.mkdir())
            throw new RuntimeException();

        // remove any leftover temporary tables
        for (String filename : dbDirectory.list())
            if (filename.startsWith("temp"))
                new File(dbDirectory, filename).delete();
    }

    synchronized void read(Block blk, ByteBuffer bb) {
        try {
            bb.clear();
            FileChannel fc = getFile(blk.fileName());
            fc.read(bb, blk.number() * bb.capacity());
        }
        catch (IOException e) {
            throw new RuntimeException();
        }
    }

    synchronized void write(Block blk, ByteBuffer bb) {
        try {
            bb.rewind();
            FileChannel fc = getFile(blk.fileName());
            fc.write(bb, blk.number() * bb.capacity());
        }
        catch (IOException e) {
            throw new RuntimeException();
        }
    }
}
```

Figure 12-15

The code for the SimpleDB class *FileMgr*

```
synchronized Block append(String filename,
                           ByteBuffer bb) {
    try {
        FileChannel fc = getFile(filename);
        int newblknum = size(filename);
        Block blk = new Block(filename, newblknum);
        write(blk, bb);
        return blk;
    }
    catch (IOException e) {
        throw new RuntimeException();
    }
}

public synchronized int size(String filename) {
    try {
        FileChannel fc = getFile(filename);
        return (int)(fc.size() / Page.BLOCK_SIZE);
    }
    catch (IOException e) {
        throw new RuntimeException();
    }
}

public boolean isNew() {
    return isNew;
}

private FileChannel getFile(String filename)
    throws IOException {
    FileChannel fc = openFiles.get(filename);
    if (fc == null) {
        File dbTable = new File(dbDirectory, filename);
        RandomAccessFile f =
            new RandomAccessFile(dbTable, "rws");
        fc = f.getChannel();
        openFiles.put(filename, fc);
    }
    return fc;
}
```

Figure 12-15 (Continued)

's', OS'nin kendi tampon
bölgesindeki sayfayı
hemen diske yazmasını
sağlıyor

Read, write ve append fonksiyonlarının her biri sadece 1 disk erişimine neden olur..

Birim test (*unit test*)

- Servis sağlayıcının kodunu değiştirip JDBC istemci ile kodu test etmek hem zor hem masraflı
- Birim test ile sadece ilgili paket içerisinde test senaryosu yazıp, sadece o paketi test etmek mümkün. (*Örneğin, Sunu:15, fig.12.12 bir birim test olarak düşünülebilir.*)