VT Gerçeklenmesi Ders Notları- #5

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.

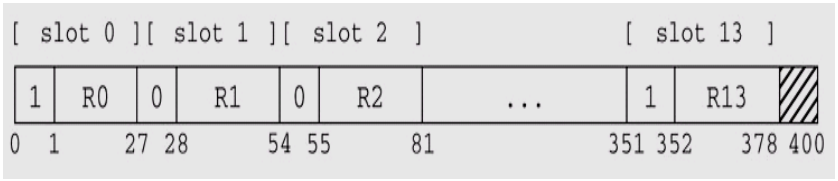
Kayıt (record) yönetimi

- Kayıt içindeki nitelikler..
- Blok içindeki kayıtlar...
- Dosya içindeki kayıtlar...

KAYITLAR (*records*)

- Erişim soyutlaması:
  - Dosya → blok → byte
  - Dosya → kayıt
- Şimdiye kadar; "bir dosyanın bir disk bloğunun X.byte"ına *int* veya *string* tipinde bir değer yazma işlemi gerçekleştirildi.
- Kayıt yönetici dosyaya kayıt ekleyebilir, silebilir, erişebilir, kaydın bir niteliğini değiştirebilir..
- KAYIT ORGANİZASYONU:
  - Kayıt içi nitelik organizasyonu: SCHEMA, TABLEINFO
    - Sabit uzunluklu nitelikler & değişken uzunluklu nitelikler.
    - Niteliklerin kayıt içi fiziksel yerleşimi
  - Blok içi kayıt organizasyonu: RECORDPAGE
    - Zincirli & zincirli olmayan (spanned vs. unspanned)
  - Dosya içi kayıt organizasyonu: RECORDFILE
    - Homojen & homojen olmayan dosya
- SimpleDB Kayıt Yönetimi:
  - Homojen,
  - Zincirsiz (unspanned)
  - Sabit uzunluklu nitelikler.

Kayıt içi nitelik organizasyonu: sabit uzunluklu gerçekleme



Record Length: 26
Field Information:

Name	Type	Length	Offset
SId	int	4	0
SName	varchar	14	4
GradYear	int	4	18
MajorId	int	4	22

- Kayıt ID: slot numarası, sabit. Değişmek zorunda kalmıyor.
- Kayıt okuma/yazmada pozisyon hesaplama:
  - bloktaki k. kayda erişim: *offset:* (RL+1)*k
  - bloktaki k. kaydın herhangi bir niteliğine erişim: *offset:*(RL+1)*k + 1 + "kayıt içi offset adresi".
  - Örneğin: yukarıdaki blokta 11. kayıttaki GradYear niteliğine erişmek istiyoruz:
  - *offset:* (26+1)*11 + 1+18 = 316.byte

Kayıt içi nitelik organizasyonu: değişik uzunluklu gerçekleme

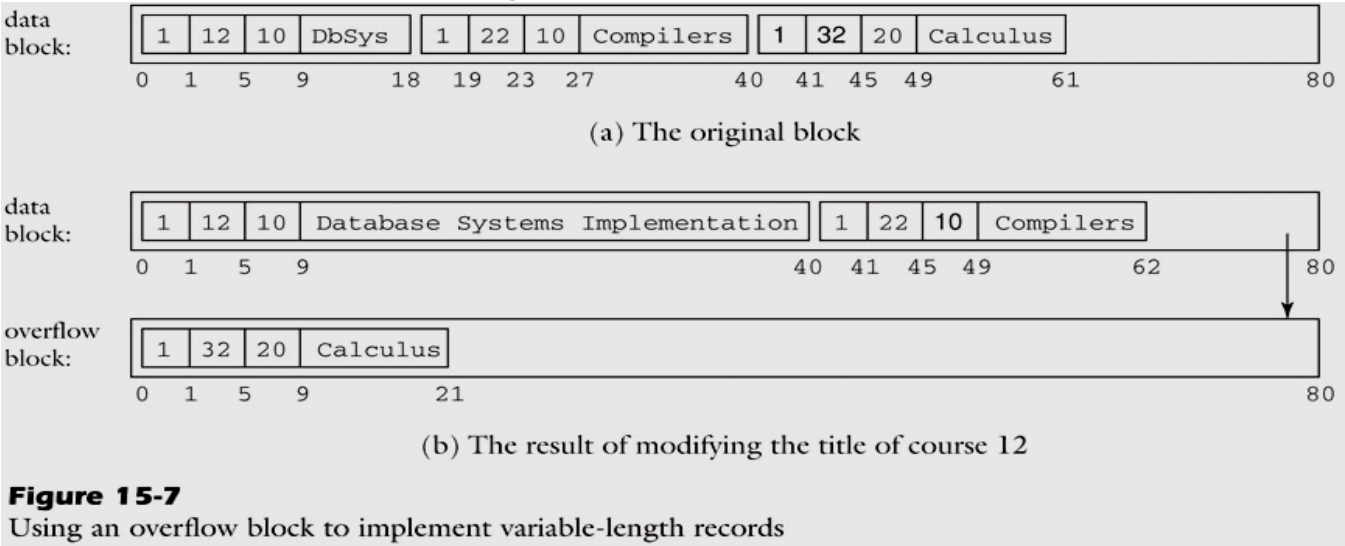


Figure 15-7
Using an overflow block to implement variable-length records

• Aralarında değişken uzunluklu nitelikler olduğu için; Kayıt içi nitelik ofsetlerinin hepsini sabit tutamayız. Ancak, Sabit uzunluklu nitelikleri değişken uzunluklu niteliklerden önce yerleştirirsek, 2. karşılaşılan değişken uzunluklu niteliğe kadar olan kayıt içi ofset'ler sabit olabilir.Bundan sonraki değişken uzunluklu niteliklere erişmek ancak, öncesindeki niteliklere erişime ile mümkün olabilir. *(Şekilde görülüyor)*
• Herhangi bir niteliğin değişmesi kayıtların yerini değiştirir; taşan (*overflow*) bloğa ihtiyaç olabilir..Yeni gelen diğer kayıtlar taşan bloğa yerleşmemeli. *(Şekilde görülüyor)*
• Şekil 15.8b'deki gibi kayıtların yerinin değişmesi gerekebileceği için; kayıt ID slot numarası olmamalı.

Kayıt içi nitelik organizasyonu: değişik uzunluklu gerçekleme

- ID-table gerçeklemesi: dizinin indisi kayıt-ID; içerdiği değer ise o kaydın blok içindeki offset değeri olur.
- Böylece kayıtların blok içerisinde yerleri kolayca değiştirilebilir.

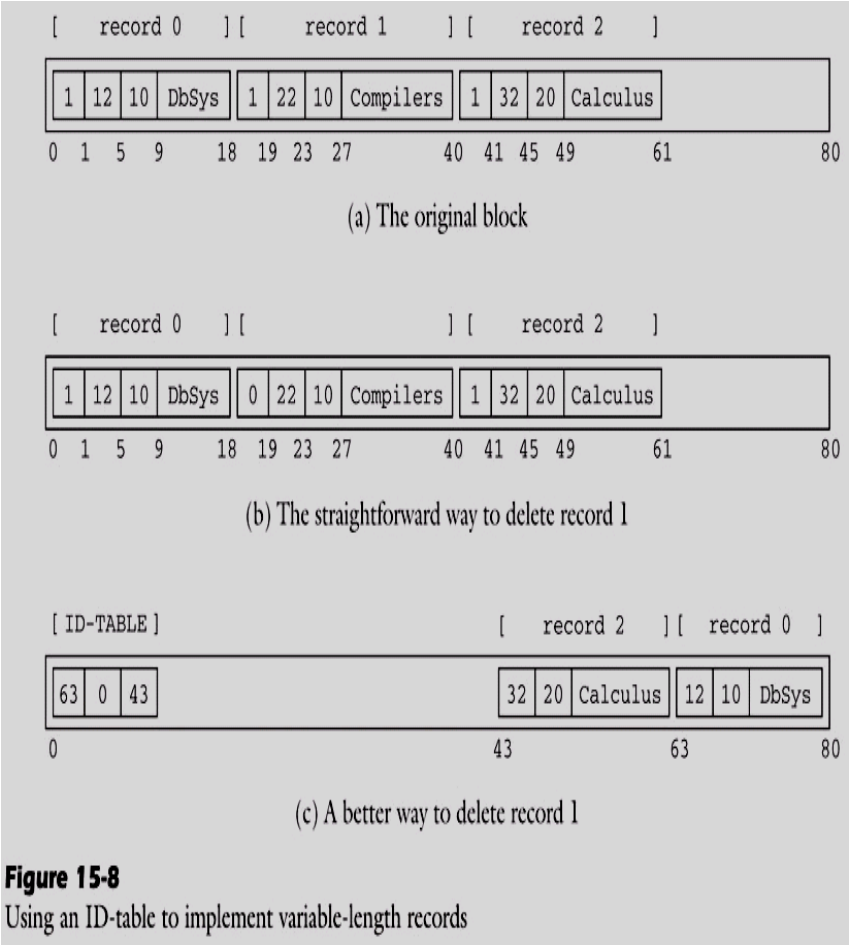


Figure 15-8
Using an ID-table to implement variable-length records

Örnek

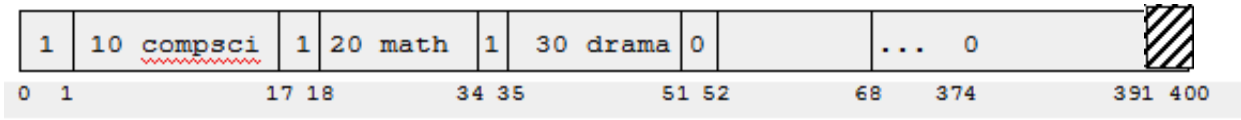
DEPT: Record Length: 16

Name	Type	Length	Offset
DId	int	4	0
DName	varchar	12	4

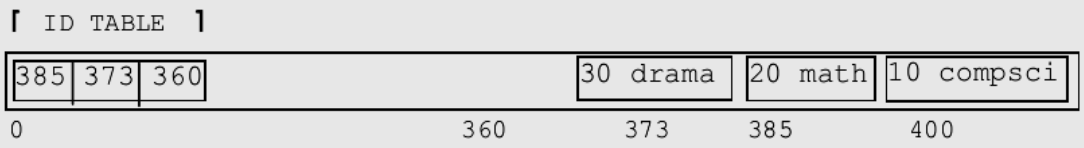
DEPT	DId	DName
	10	compsci
	20	math
	30	drama

DEPT: Each slot is 17 bytes. There are 23 slots that fill the block with 9 bytes left over.

[slot 0] [slot 1][slot 2] [slot 3] [slot 22]



DEPT:



SQL: Sabit & değişken uzunluklu nitelikler

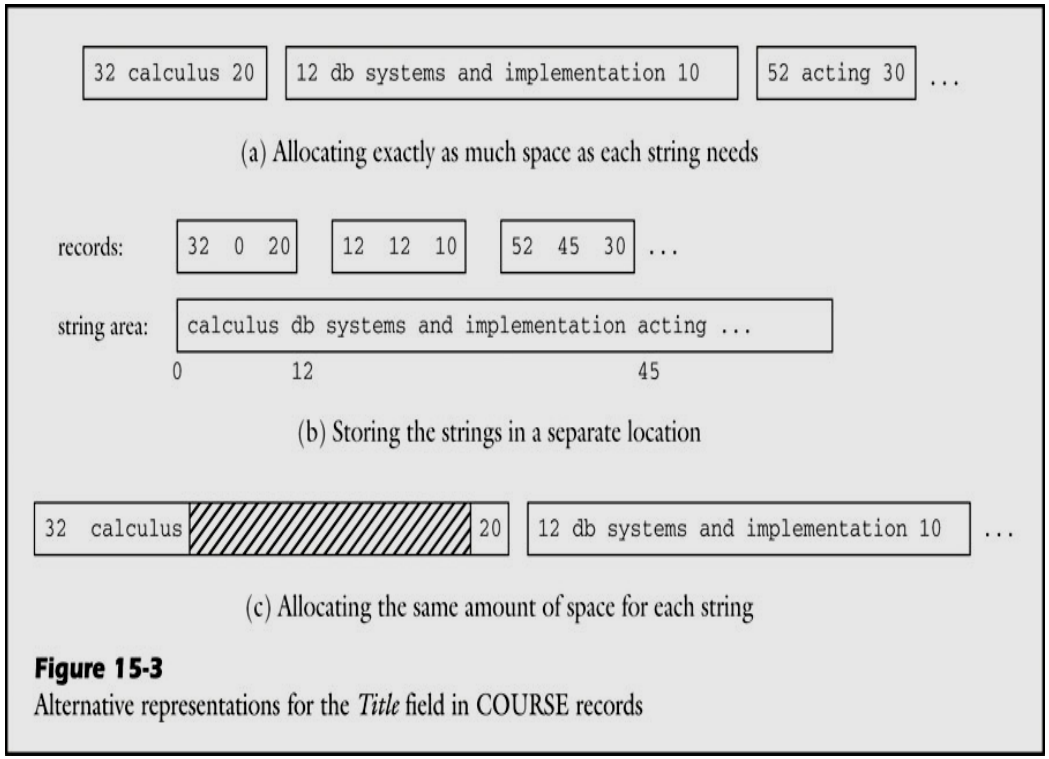


Figure 15-3
Alternative representations for the *Title* field in COURSE records

- SQL
  - char(n): şekilde c) gerçeklemesi
  - varchar(n): şekilde a) veya c) gerçeklemesi
  - clob(n): şekilde b) gerçeklemesi

Kayıt yönetimi stratejilerinin gerçeklenmesi-2

- Zincirli yapıda kayıt yerleşiminin gerçeklenmesi

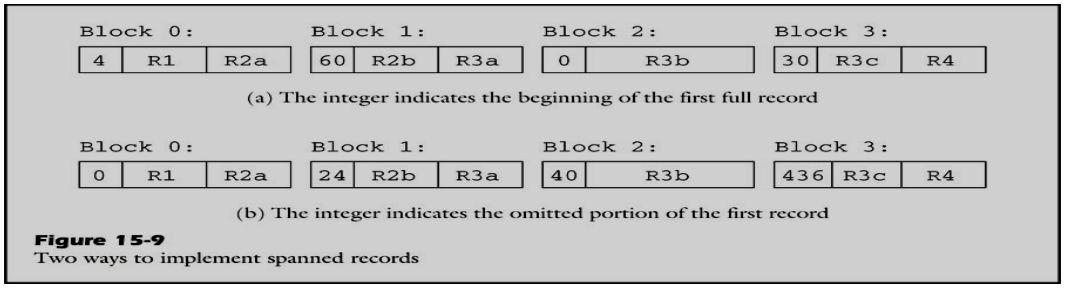


Figure 15-9
Two ways to implement spanned records

- Homojen olmayan dosya gerçeklenmesi.
  - Her kayıt tipine ait tablo bilgisi saklanmalı
  - Okunan bir kaydın hangi tabloya ait olduğu bilgisi saklanmalı
  - SimpleDB'de bu tip bir dosyaya örnek var. O da ..

Örnek SimpleDB uygulamaları

```
Schema sch = new Schema();
sch.addIntField("cid");
sch.addStringField("title", 20);
sch.addIntField("deptid");
TableInfo ti = new TableInfo("course", sch);

for (String fldname : ti.schema().fields()) {
   int offset = ti.offset(fldname);
   System.out.println(fldname + " has offset " + offset);
}
```

En fazla 20 karakter

Figure 15-11
Specifying the structure of COURSE records

```
SimpleDB.init("studentdb");
//get COURSE table info, as in Figure 15-11
TableInfo ti = ...

Transaction tx = new Transaction();
String filename = ti.fileName();
Block blk = new Block(filename, 337);
RecordPage rp = new RecordPage(blk, ti, tx);

// Part 1
while (rp.next()) {
   int dept = rp.getInt("deptid");
   if (dept == 30)
      rp.delete();
   else if (rp.getString("title").equals("Algebra"))
      rp.setString("title", "Group Theory");
}

// Part 2
boolean ok = rp.insert();
if (ok) {
   rp.setInt("cid", 82);
   rp.setString("title", "OO Design");
   rp.setInt("deptid", 20);
}
rp.close();
tx.commit();
```

```
SimpleDB.init("studentdb");
// get the COURSE table info, as in Figure 15-11
TableInfo ti = ...

Transaction tx = new Transaction();
RecordFormatter fmtr = new RecordFormatter(ti);
Block blk = tx.append(ti.fileName(), fmtr);
RecordPage rp = new RecordPage(blk, ti, tx);
rp.insert();
rp.close();
tx.commit();
```

```
SimpleDB.init("studentdb");
Transaction tx = new Transaction();
Schema sch = new Schema();
sch.addIntField("A");
TableInfo ti = new TableInfo("junk", sch);
RecordFile rf = new RecordFile(ti, tx);
for (int i=0; i<10000; i++) {
   rf.insert();
   int n = (int) Math.round(Math.random() * 200);
   rf.setInt("A", n);
}

int count = 0;
rf.beforeFirst();
while (rf.next()) {
   if (rf.getInt("A") < 100) {
      count++;
      rf.delete();
   }
}
System.out.println(count + " values were deleted");
rf.close();
tx.commit();
```

SimpleDB'de Kayıt Yönetimi

Schema

```
public void addField(String fldname, int type,
                     int length);
public void addIntField(String fldname);
public void addStringField(String fldname,
                           int length);
public void add(String fldname, Schema sch);
public void addAll(Schema sch);

public Collection<String> fields();
public boolean hasField(String fldname);
public int type(String fldname);
public int length(String fldname);
```

TableInfo

```
public TableInfo(String tblname, Schema schema);
public TableInfo(String tblname, Schema schema,
                 Map<String,Integer> offsets,
                 int recordlen);

public String fileName();
public Schema schema();
public int offset(String fldname);
public int recordLength();
```

Figure 15-10
The API for SimpleDB table information

- Bir tabloya ait "Schema", [nitelik, tip,uzunluk] bilgisi tutuyor.
- "TableInfo" ise şema ile birlikte fiziksel özellikleri tutuyor (tablonun saklandığı dosya ismi, niteliklerin offset adreslerini, kayıt özellikleri)

Schema ve TableInfo Gerçeklenmesi

```
public class Schema {
   private Map<String,FieldInfo> info =
                          new HashMap<String,FieldInfo>();

   public void addField(String fldname, int type,
                        int length) {
      info.put(fldname, new FieldInfo(type, length));
   }
   public void addIntField(String fldname) {
      addField(fldname, INTEGER, 0);
   }

   public void addStringField(String fldname,
                              int length) {
      addField(fldname, VARCHAR, length);
   }

   public void add(String fldname, Schema sch) {
      int type = sch.type(fldname);
      int length = sch.length(fldname);
      addField(fldname, type, length);
   }

   public void addAll(Schema sch) {
      info.putAll(sch.info);
   }

   public Collection<String> fields() {
      return info.keySet();
   }

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

   public int type(String fldname) {
      return info.get(fldname).type;
   }

   public int length(String fldname) {
      return info.get(fldname).length;
   }

   class FieldInfo {
      int type, length;
      public FieldInfo(int type, int length) {
         this.type = type;
         this.length = length;
      }
```

```
public class TableInfo {
   private Schema schema;
   private Map<String,Integer> offsets;
   private int recordlen;
   private String tblname;

   public TableInfo(String tblname, Schema schema) {
      this.schema = schema;
      this.tblname = tblname;
      offsets = new HashMap<String,Integer>();
      int pos = 0;
      for (String fldname : schema.fields()) {
         offsets.put(fldname, pos);
         pos += lengthInBytes(fldname);
      }
      recordlen = pos;
   }

   public TableInfo(String tblname, Schema schema,
     Map<String,Integer> offsets, int recordlen) {
      this.tblname = tblname;
      this.schema = schema;
public Schema schema() {
   return schema;
}

public int offset(String fldname) {
   return offsets.get(fldname);
}

public int recordLength() {
   return recordlen;
}

private int lengthInBytes(String fldname) {
   int fldtype = schema.type(fldname);
   if (fldtype == INTEGER)
      return INT_SIZE;
   else
      return STR_SIZE(schema.length(fldname));
}
```

Blok(Sayfa) içi Kayıt Yönetimi: RecordPage

```
public RecordPage(Block blk, TableInfo ti, Transaction tx);

public boolean next();
public int     getInt(String fldname);
public String  getString(String fldname);
public void    setInt(String fldname, int val);
public void    setString(String fldname, String val);
public void    close();

public void    delete();
public boolean insert();

public void    moveToId(int id);
public int     currentId();
```

Figure 15-14
The API for the SimpleDB class *RecordPage*

- Kayıt ID, sabit olup, indeks tarafından kullanılır.

RecordPage sınıfının gerçeklenmesi

```
public class RecordPage {
   public static final int EMPTY = 0, INUSE = 1;

   private Block blk;
   private TableInfo ti;
   private Transaction tx;
   private int slotsize;
   private int currentslot = -1;
  public RecordPage(Block blk, TableInfo ti,
                    Transaction tx) {
     this.blk = blk;
     this.ti = ti;
     this.tx = tx;
     tx.pin(blk);
     slotsize = ti.recordLength() + INT_SIZE;
  }

  public void close() {
     if (blk != null)
       tx.unpin(blk);
     blk = null;
  }

  public boolean next() {
     return searchFor(INUSE);
  }

  public int getInt(String fldname) {
     int position = fieldpos(fldname);
     return tx.getInt(blk, position);
  }

  public String getString(String fldname) {
     int position = fieldpos(fldname);
     return tx.getString(blk, position);
  }

  public void setInt(String fldname, int val) {
     int position = fieldpos(fldname);
     tx.setInt(blk, position, val);
  }

  public void setString(String fldname, String val) {
     int position = fieldpos(fldname);
     tx.setString(blk, position, val);
  }

  public void delete() {
     int position = currentpos();
     tx.setInt(blk, position, EMPTY);
  }

  public boolean insert() {
     currentslot = -1;
     boolean found = searchFor(EMPTY);
     if (found) {
        int position = currentpos();
        tx.setInt(blk, position, INUSE);
     }
     return found;
  }

  public void moveToId(int id) {
     currentslot = id;
  }

  public int currentId() {
     return currentslot;
  }

  private int currentpos() {
     return currentslot * slotsize;
  }

  private int fieldpos(String fldname) {
     int offset = INT_SIZE + ti.offset(fldname);
     return currentpos() + offset;
  }

  private boolean isValidSlot() {
     return currentpos() + slotsize <= BLOCK_SIZE;
  }

  private boolean searchFor(int flag) {
     currentslot++;
     while (isValidSlot()) {
        int position = currentpos();
        if (tx.getInt(blk, position) == flag)
           return true;
        currentslot++;
     }
     return false;
  }
}
```

Figure 15-16 *(Continued)*

RecordFormatter sınıfının gerçeklenmesi

```
class RecordFormatter implements PageFormatter {
   private TableInfo ti;

   public RecordFormatter(TableInfo ti) {
      this.ti = ti;
   }

   public void format(Page page) {
      int recsize = ti.recordLength() + INT_SIZE;
      for (int pos=0; pos+recsize <= BLOCK_SIZE;
            pos += recsize) {
         page.setInt(pos, EMPTY);
         makeDefaultRecord(page, pos);
      }
   }

   private void makeDefaultRecord(Page page, int pos) {
      for (String fldname : ti.schema().fields()) {
         int offset = ti.offset(fldname);
         if (ti.schema().type(fldname) == INTEGER)
            page.setInt(pos + INT_SIZE + offset, 0);
         else
            page.setString(pos + INT_SIZE + offset, "");
      }
   }
}
```

Figure 15-17
The code for the SimpleDB class *RecordFormatter*

- TableInfo'daki fiziksel konum bilgilerine göre yeni eklenen sayfayı)bloğu biçimlendirir.
- Tx.append ➔ BuffMgr.pinNew ➔ Buffer.assignToNew ➔ RecordFormatter.format()

Dosya içi Kayıt Yönetimi: RecordFile

«Dosya➔Blok» erişimini kullanıcıdan gizler. Kullanıcı dosyaya «Dosya➔Kayıt» soyutlaması ile erişir..

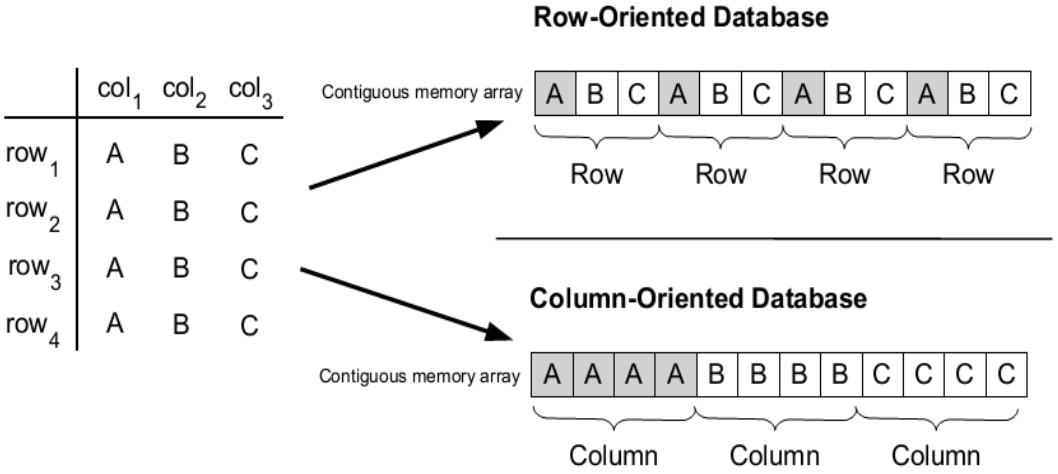
```
public RecordFile(TableInfo ti, Transaction tx);

// methods that establish the current record
public void    beforeFirst();
public boolean next();
public void    moveToRid(RID r);
public void    insert();
public void    close();

// methods that access the current record
public int     getInt(String fldname);
public String  getString(String fldname);
public void    setInt(String fldname, int val);
public void    setString(String fldname, String val);
public RID     currentRid();
public void    delete();
```

Figure 15-19
The API for the SimpleDB class *RecordFile*

```
public class RID {
   private int blknum;
   private int id;

   public RID(int blknum, int id) {
      this.blknum = blknum;
      this.id = id;
   }

   public int blockNumber() {
      return blknum;
   }

   public int id() {
      return id;
   }

   public boolean equals(Object obj) {
      RID r = (RID) obj;
      return blknum == r.blknum && id==r.id;
   }
}
```

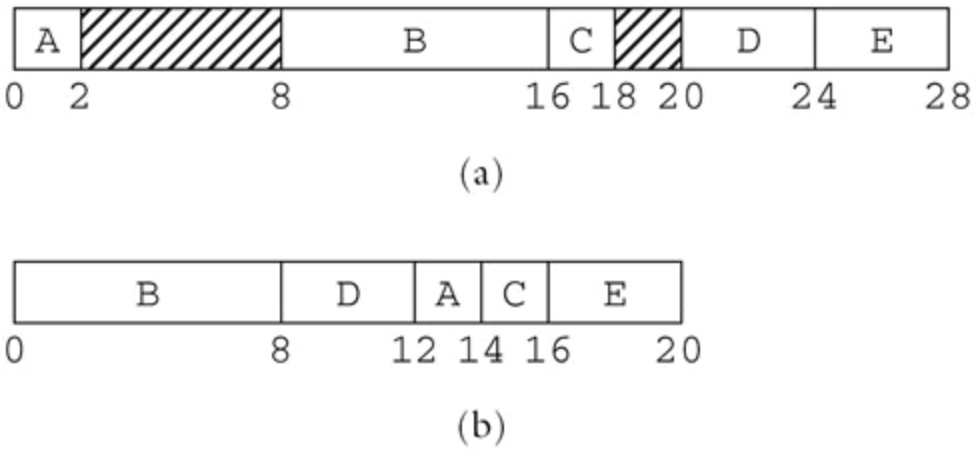
Figure 15-22
The code for the SimpleDB class *RID*

Kayıt org'da Farklı yaklaşımlar



- Satır-yönelimli(*row-based*) / nitelik yönelimli (*column-based*) yerleşimler.
  - küçük boyutlu kayıtlarda satır-yönelimli, uzun kayıtlarda nitelik yönelimli tercih edilir.
  - OLTP için satır yönelimli saklama tercih edilir.
  - "OLAP / enterprice uygulamaları" için nitelik yönelimli tercih edilir.
- clob gerçeklenmesi:
  - Büyük verinin kendisi yerine; Büyük karakter dizilerinin, veya multimedia verilerinin tutulduğu dosya isminin veri tabanında saklanması. En önemli dezavantaj: veri kurtarma ve eşzamanlılıktan hizmet alamamak.
- veri sıkıştırma

Farklı sistemlerde Niteliklerin Kayıt içi fiziksel yerleşiminde "alignment"



- Create table T
  (A smallint,
  B double,
  C smallint,
  D int,
  E int)

- JAVA, üzerinde çalıştığı sistemin makine kodunu doğrudan çalıştırmıyor. JVM, byte dizisinden topladığı 4 B bilgiyi toplayıp integer veri oluşturuyor. int, double ...gibi veriyi doğrudan makine kodu ile okutmadığı için ise; alignment'a ihtiyaç olmuyor.