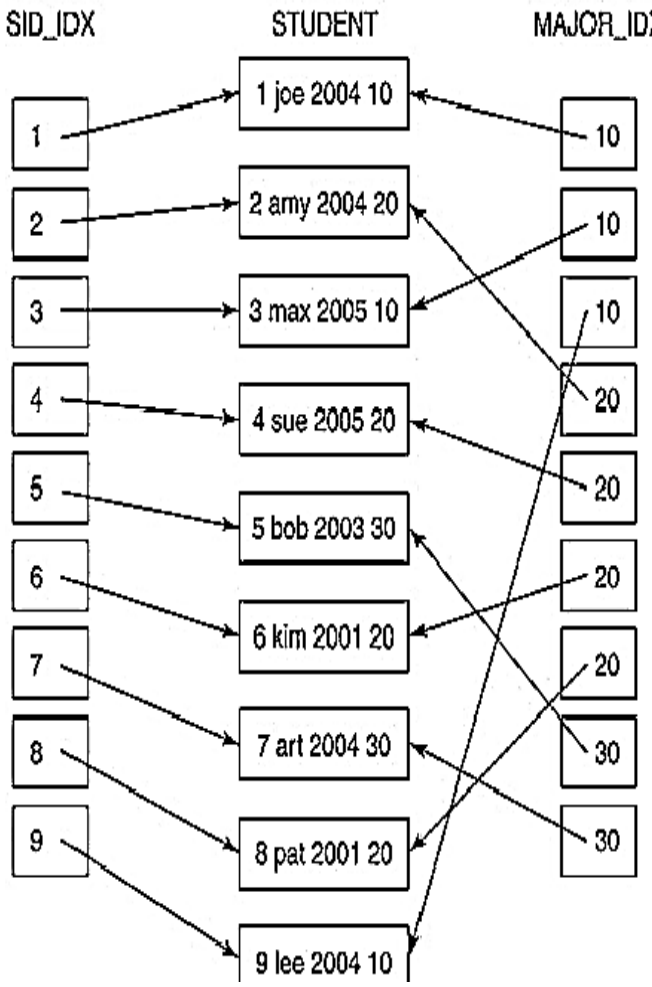
VT Sistem Gerçeklemesi Ders Notları- #11

- Dosya düzenleme ve amacı
- Adrese Dayalı indeksleme (*hashing*)
  - Statik hashing
  - Genişletilebilir hashing
- B-tree Indeksleme
- SimpleDB'de Indeks-duyarlı operatör gerçeklemeleri

Dosya Düzenleme

- Dosya Düzenleme:
  - Adrese dayalı düzenleme (hashing)
  - indeksleme
- Düzenlemede Amaç: Ana dosya üzerinde sıralı erişim yapmadan isenilen kayıt(lar)a direk olarak erişim (*random access*).
- En basit düzenleme : Dosyanın herhangi bir niteliğine göre sıralanması.
  - Farklı aramalar için ana dosyanın arama anahtarına göre sıralı kopyası
    - yer kaybı
    - Senkronizasyon
    - Veri tutarlılığı tehlikesi
- Ana dosya kopyası yerine ➔ INDEX
- Indeks, sadece arama anahtarının bir kopyası sıralı olarak tutan dosyadır.
- indeks kaydı = arama anahtarı (**dataval**) + RID (**datarid**)
- Indeks organizasyonu, ana dosya iç organizasyonundan bağımsız..

```
create index MAJOR_IDX on STUDENT(MajorId);
create unique index SID_IDX on STUDENT(SId)
```



***RULE**: The usefulness of the index on field A is proportional to the number of different A-values in the table.*

- 45.000 STUDENT records, 10 STUDENTS/disk_block
- 40 departments, 20 DEPTs / block
- divided to majors evenly.(1125 students/major)
- Indeks'te bir anahtar bulma: 2 disk blok erişimi olsun.

```
select MajorId
from STUDENT
where SId=8
```

(a) An SQL query to find the major of student #8

1. For each record in STUDENT:
   If the *SId*-value of this record is 8 then:
   Return the *MajorId*-value of that record.

4500 *(en kötü durum)*

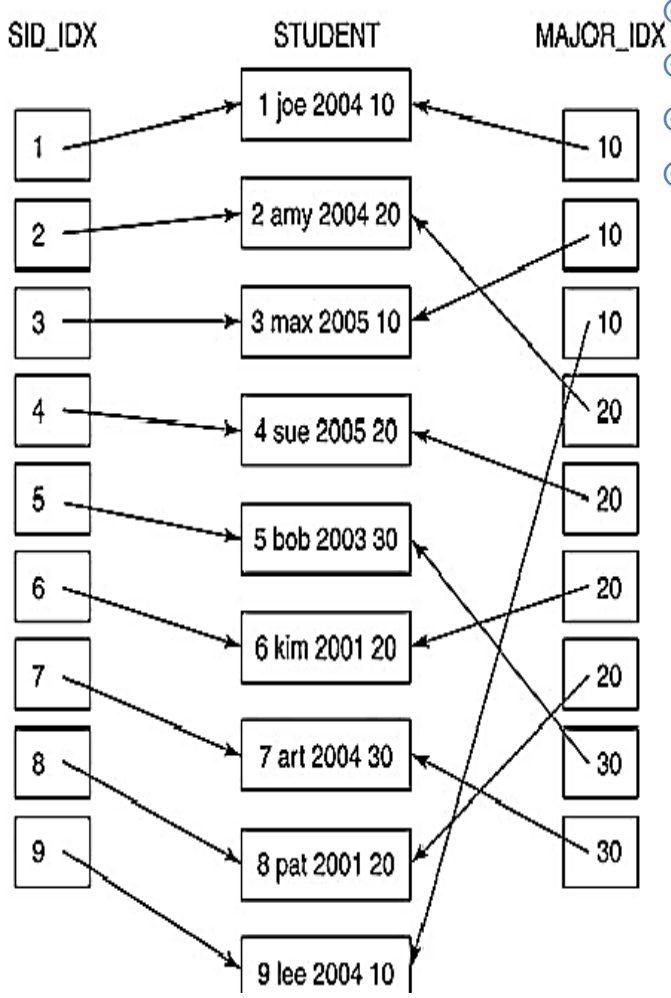
(b) Implementing the query without an index

1. Use the SID_IDX index to find the index record whose *SId*-value is 8.
2. Obtain the value of its *RID* field.
3. Move directly to the STUDENT record having that RID.
4. Return the *MajorId*-value of that record.

3

(c) Implementing the query using an index

```
create index MAJOR_IDX on STUDENT(MajorId);
create unique index SID_IDX on STUDENT(SId)
```



- 45.000 STUDENT records, 10 STUDENTS/disk_block
- 40 departments, 20 DEPTs / block
- divided to majors evenly.(1125 students/major)
- Indeks'te bir anahtar bulma: 2 disk blok erişimi olsun.

```
select SId
from STUDENT
where MajorId=10
```

(a) An SQL query to find the IDs of students having major #10

1. For each record in STUDENT:
   If the *MajorId*-value of this record is 10 then:
   Add its *SId*-value to the output list.
2. Return the output list.

4500 *(her zaman)*

(b) Implementing the query without an index

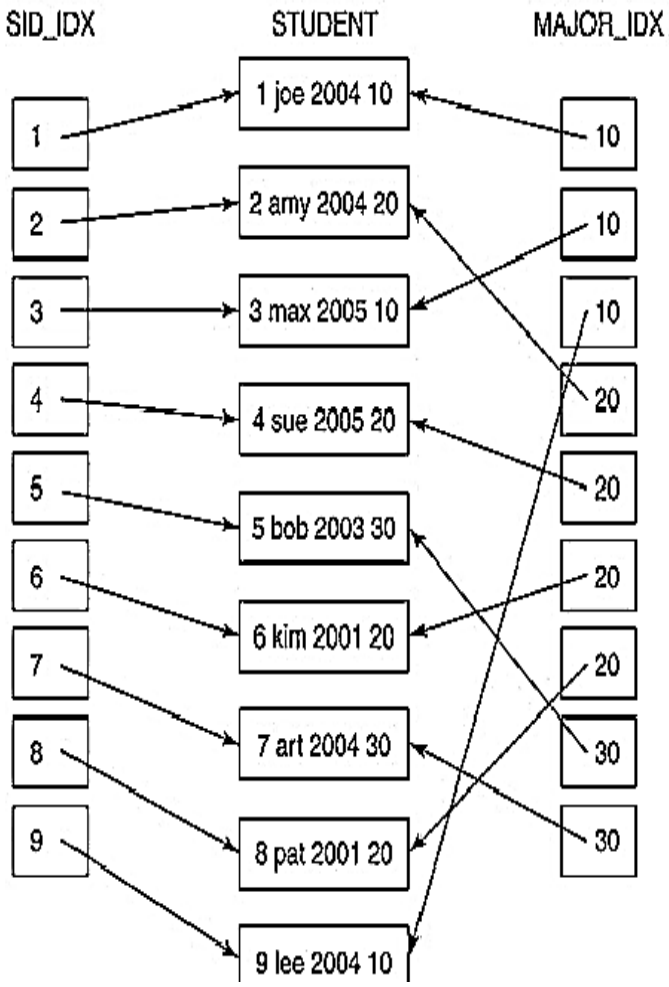
1. Use the MAJOR_IDX index to find the index records whose *MajorId*-value is 10.
2. For each such index record:
   a) Obtain the value of its *RID* field.
   b) Move directly to the STUDENT record having that RID.
   c) Add the *SId*-value of that record to the output list.
3. Return the output list.

1125+2

(c) Implementing the query using an index

***RULE**: The usefulness of the index on field A is proportional to the number of different A-values in the table.*

```
create index MAJOR_IDX on STUDENT(MajorId);
create unique index SID_IDX on STUDENT(SId)
```



- 45.000 STUDENT records, 10 STUDENTS/disk_block
- 40 departments, 20 DEPTs / block
- divided to majors evenly.(1125 students/major)
- Indeks'te bir anahtar bulma: 2 disk blok erişimi olsun.

```
select SName
from STUDENT
where SId > 8
```

(a) An SQL query to find the names of students having ID > 8

1. For each record in STUDENT:
   If the *SId*-value of this record is > 8 then:
   Add the *SName*-value of that record to the output list.
2. Return the output list.

Yaklaşık 4500 blok

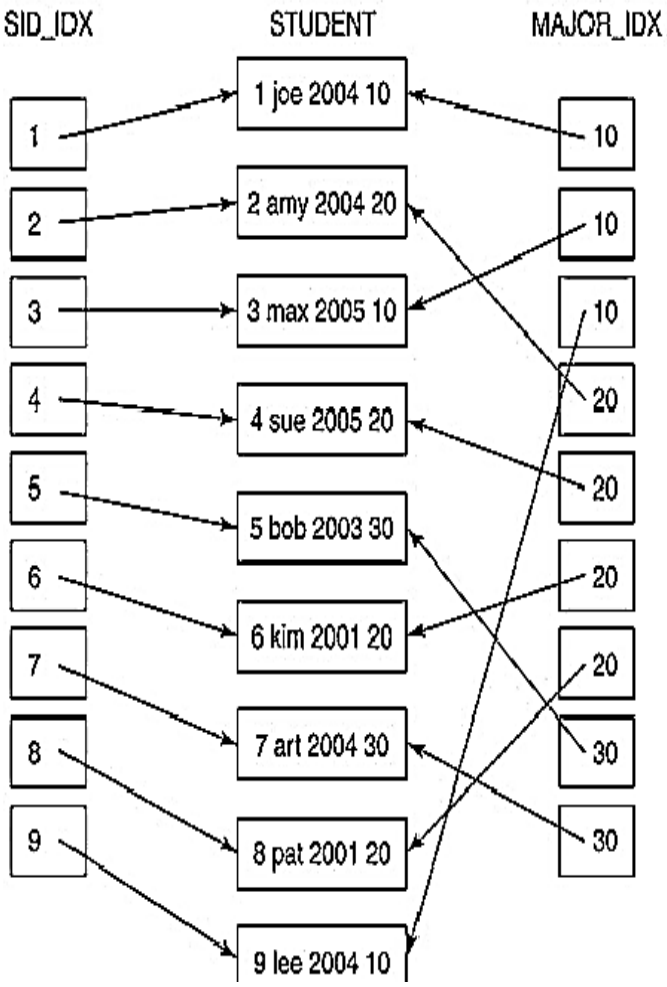
(b) Implementing the query without an index

1. Use the SID_IDX index to find the last index record whose *SId*-value is 8.
2. For the remaining records in the index (assuming that they are in sorted order):
   a) Obtain the value of its *RID* field.
   b) Move directly to the STUDENT record having that RID.
   c) Add the *SName*-value of that record to the output list.
3. Return the output list.

Ana dosyanın sıralı omadığını varsaymış. O zaman belki 4500'den daha fazla..

Eğer ID>8 olan çok sayıda STUDENT var ise; STUDENT tablosunda sıralı arama tercih edilir...

```
create index MAJOR_IDX on STUDENT(MajorId);
create unique index SID_IDX on STUDENT(SId)
```



- 45.000 STUDENT records, 10 STUDENTS/disk_block
- 40 departments, 20 DEPTs / block
- divided to majors evenly.(1125 students/major)
- Indeks'te bir anahtar bulma: 2 disk blok erişimi olsun.

```
select s.SName, e.Grade
from STUDENT s, ENROLL e
where s.SId=e.StudentId
```

(a) An SQL query to find the names of students and their grades

1. For each record in ENROLL:
   For each record in STUDENT:
   If *SId* = *StudentId* then:
   Add the values for *SName* and *Grade* to the output list.
2. Return the output list.

1,5 milyon * 4500

(b) Implementing the query without an index

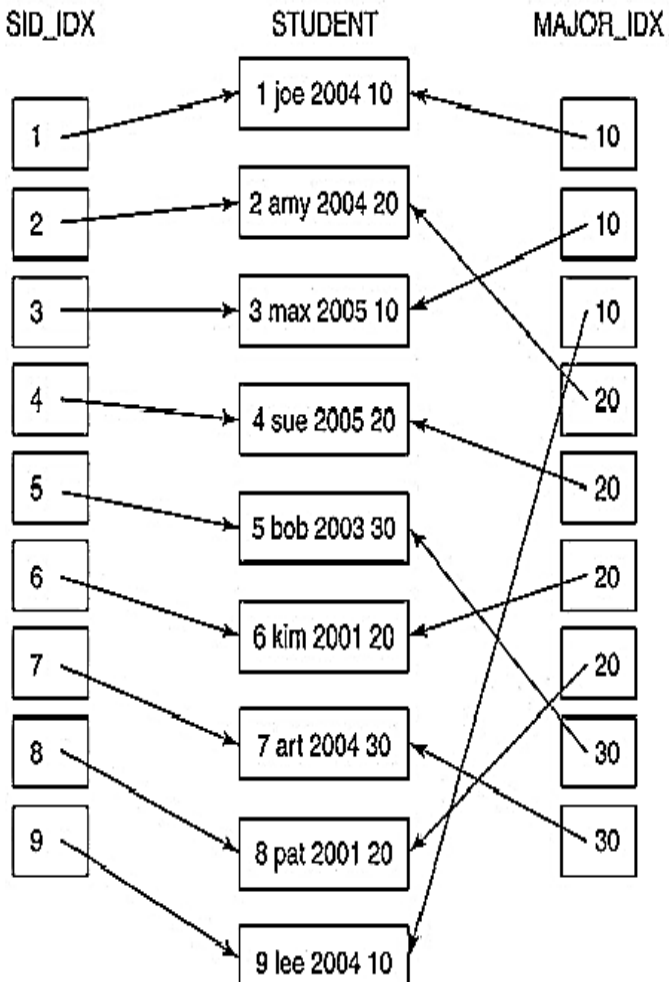
1. For each record in ENROLL:
   a) Let x be the *StudentId*-value of that record.
   b) Use the SID_IDX index to find the index record whose *SId*-value = x.
   c) Obtain the value of its *RID* field.
   d) Move directly to the STUDENT record having that RID.
   e) Add the values for *SName* and *Grade* to the output list.
2. Return the output list.

1,5 milyon * 3

(c) Implementing the query using an index

Her zaman olmasa da idx kullanmak, table scan'dan çoğu zaman daha iyi performans veriyor..

```
create index MAJOR_IDX on STUDENT(MajorId);
create unique index SID_IDX on STUDENT(SId)
```



- 45.000 STUDENT records, 10 STUDENTS/disk_block
- 40 departments, 20 DEPTs / block
- divided to majors evenly.(1125 students/major)
- Indeks'te bir anahtar bulma: 2 disk blok erişimi olsun.

```
select d.DName
from STUDENT s, DEPT d
where s.MajorId=d.DId and s.SId=8
```

(a) An SQL query to find the name of student #8's major

1. Use the SID_IDX index to find the index record having *SId*=8.
2. Obtain the value of its *RID* field.
3. Move directly to the STUDENT record having that RID.
   Let x be the *MajorId*-value of that record.
4. For each record in DEPT:
   If the *DId*-value of that record = x then:
   Return the value of *DName*.

3 + 2 = 5 (en kötü)

3+1=4 (en iyi)

(b) Implementing the query using the index SID_IDX

1. For each record in DEPT:
   a) Let y be the *DId*-value of that record.
   b) Use the MAJOR_IDX index to find the index records whose *MajorId*-value = y.
   c) For each such index record:
      - Obtain the value of its *RID* field.
      - Move directly to the STUDENT record having that RID.
      - If the *SId*-value of that record = 8 then:
        Return the value of *DName*.

*40 *3 =120*

(c) Implementing the query using the index MAJOR_IDX

Sorgu Yükleminde çok sayıda index varsa; herbirinin kullanımı farklı performans değerleri veriyor. İsabetli kararı vermek Planner(*optimizer*)'ın vazifesi..

Index bakımı ve seçimi

- Index yer kaplıyor.
- Veri bütünlüğü ve senkronizasyon sağlanmalı.
- Ana dosyadaki durum değişiklikleri:
  - Kayıt ekleme
  - Kayıt silme
  - Bazı kayıtlara ait bir niteliğin değişmesi
- **Cost of index= "disk space required to hold it" * "time to maintain it" * "the frequency of update to base table"**
- **Benefit of an index= "cost savings that the index provides for each query" * "the frequency of occurance of each query using the index"**
- Maliyet hesabı için: B(T),R(T), V(T,F) değerlerini saklamak gerekiyor.

SimpleDB'de indeks kullanımı

```
public interface Index {
   public void    beforeFirst(Constant searchkey);
   public boolean next();
   public RID     getDataRid();
   public void    insert(Constant dataval, RID datarid);
   public void    delete(Constant dataval, RID datarid);
   public void    close();
}
```

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

// Open a scan on the data table.
Plan studentplan = new TablePlan("student", tx);
TableScan studentscan = (TableScan) studentplan.open();

// Open the index on MajorId, as in Figure 16-13.
MetadataMgr mdmgr = SimpleDB.mdMgr();
Map<String,IndexInfo> indexes =
                        mdmgr.getIndexInfo("student", tx);
IndexInfo ii = indexes.get("majorid");
Index idx = ii.open();

// Retrieve all index records having a dataval of 10.
idx.beforeFirst(new IntConstant(10));
while (idx.next()) {
   // Use the datarid to move to a STUDENT record.
   RID datarid = idx.getDataRid();
   studentscan.moveToRid(datarid);
   System.out.println(studentscan.getString("sname"));
}

// Close the index and the data table.
idx.close();
studentscan.close();
tx.commit();
```

SimpleDB'de indeks kullanımı

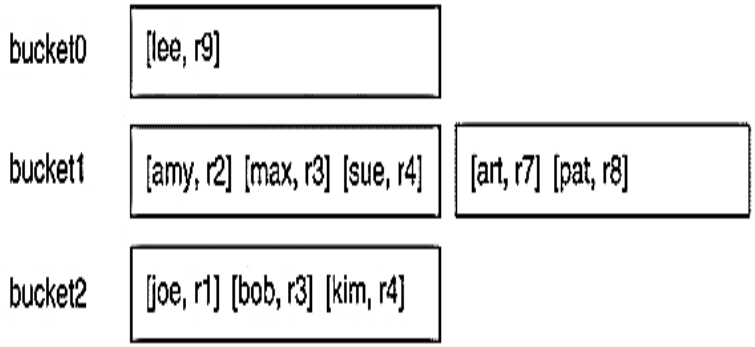
```
SimpleDB.init("studentdb");
Transaction tx = new Transaction();
Plan studentplan = new TablePlan("student", tx);
TableScan studentscan = (TableScan) studentplan.open();

// Create a map containing all indexes for STUDENT.
MetadataMgr mdmgr = SimpleDB.mdMgr();
Map<String,Index> indexes = new HashMap<String,Index>();
Map<String,IndexInfo> idxinfo =
                          mdmgr.getIndexInfo("student", tx)
for (String fldname : idxinfo.keySet()) {
   Index idx = idxinfo.get(fldname).open();
   indexes.put(fldname, idx);
}

// Task 1: insert a new STUDENT record for Sam
//    First, insert the record into STUDENT.
studentscan.insert();
studentscan.setInt("sid", 11);
studentscan.setString("sname", "sam");
studentscan.setInt("gradyear", 2010);
studentscan.setInt("majorid", 30);

//    Then insert a record into each of the indexes.
RID datarid = studentscan.getRid();
for (String fldname : indexes.keySet()) {
   Constant dataval = studentscan.getVal(fldname);
   Index idx = indexes.get(fldname);
   idx.insert(dataval, datarid);
}
```

```
// Task 2: find and delete Joe's record
studentscan.beforeFirst();
while (studentscan.next()) {
   if (studentscan.getString("sname").equals("joe")) {

      // First, delete the index records for Joe.
      RID joeRid = studentscan.getRid();
      for (String fldname : indexes.keySet()) {
         Constant dataval = studentscan.getVal(fldname);
         Index idx = indexes.get(fldname);
         idx.delete(dataval, datarid);
      }
      // Then delete Joe's record in STUDENT.
      studentscan.delete();
      break;
   }
}
studentscan.close();
for (Index idx : indexes.values())
   idx.close();
tx.commit();
```

Statik Hash indeksleme

- H(*dataval*) = bucket_number
- Bucket içindeki indeks kayıtları : (dataval,datarid)
- Bucket dosyası B bloktan ve N bucket, oluşuyorsa:
  - Bir indeks kaydına erişim disk sayısı: **B/N** olur.
- Örnek: N= 3, H(*dataval*) = (*dataval* içerisinde, 'm' harfinden küçük harflerin sayısı) mod N,
- 3 indeks kaydı /block

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



Sname *varchar(10)* ➔ 14 B + 4B + 4B +1B =23 B *(idx_record büyüklüğü)*
4KB /23B = 178 idx_record / block
N=B=1024 ➔ 178*1024 = 182.272 idx_record'a **1** disk erişimi ile erişebiliyoruz.
N=1024, B=2048 ➔ 178*2048 = 364.544 idx_record'a **2** disk erişimi ile erişebiliyoruz.

Statik Hash gerçekleme

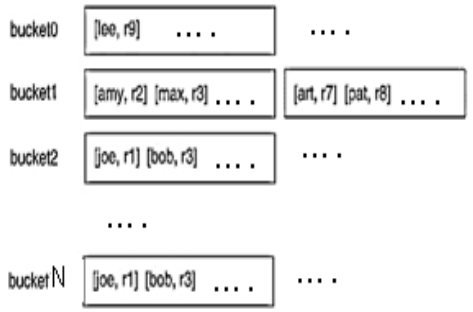
```
public class HashIndex implements Index {
   public static int NUM_BUCKETS = 100;
   private String idxname;
   private Schema sch;
   private Transaction tx;
   private Constant searchkey = null;
   private TableScan ts = null;

   public HashIndex(String idxname, Schema sch,
                    Transaction tx) {
      this.idxname = idxname;
      this.sch = sch;
      this.tx = tx;
   }

   public void beforeFirst(Constant val) {
      close();
      searchkey = val;
      int bucket = val.hashCode() % NUM_BUCKETS;
      String tblname = idxname + bucket;
      TableInfo ti = new TableInfo(tblname, sch);
      ts = new TableScan(ti, tx);
   }

   public boolean next() {
      while (ts.next())
         if (ts.getVal("dataval").equals(searchkey))
            return true;
      return false;
   }

   public RID getDataRid() {
      int blknum = ts.getInt("block");
      int id = ts.getInt("id");
      return new RID(blknum, id);
   }
```

```
   public void insert(Constant val, RID rid) {
      beforeFirst(val);
      ts.insert();
      ts.setInt("block", rid.blockNumber());
      ts.setInt("id", rid.id());
      ts.setVal("dataval", val);
   }

   public void delete(Constant val, RID rid) {
      beforeFirst(val);
      while(next())
         if (getDataRid().equals(rid)) {
            ts.delete();
            return;
         }
   }

   public void close() {
      if (ts != null)
         ts.close();
   }

   public static int searchCost(int numblocks, int rpb) {
      return numblocks / HashIndex.NUM_BUCKETS;
   }
}
```

si= üzerinde idx oluşturulan tabloya ait istatistiksel bilgiler

Statik Hash *searchCost()*

SId	SName	GradYear	MajorId
1	joe	2004	10
2	amy	2004	20
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
	pat	2001	20
	lee	2004	10



SearchCost = B/N

B.blok

```
public int blocksAccessed() {
   int rpb = BLOCK_SIZE / ti.recordLength();
   int numblocks = si.recordsOutput() / rpb;
   return BTreeIndex.searchCost(numblocks, rpb);
}
```

```
public int blockAccessed(){
     TableInfo idxti= new TableInfo(«»,schema());
     int rpb = BLOCK_SIZE / idxti.recordLength();
     int numblocks = si.recordsOutput() / rpb;
     return HashIndex.searchCost(numblocks,rpb);
```

- Yukarıda KatalogYonetimi/Sunu:13'deki IndexInfo.blockAccessed() fonk. gerçeklemesinin düzeltilmiş hali görünüyor.

```
public static int searchCost(int numblocks, int rpb) {
   return numblocks / HashIndex.NUM_BUCKETS;
}
```

- HashIndex.searchCost(), rpb'yi kullanmıyor, numblocks değeri ise; bütün idx_record'ları içeren dense indeks'deki blokların sayısıdır. Bu Btree için yaprak düğümlerin toplam sayısı olacaktır; HashIndex için ise HashIndex dosylarının (SID_IDX1,SID_IDX2, …) toplam blok sayısı olacaktır. *(Not: Dense index, ana dosyadaki her bir kayıt için idx_record içeren indekstir.)*

Genişletilebilir Hash indeksleme

- N bucket, B blok ➔ N=B (*en iyi durum*)
- Dinamik dosyalar için:
  - B sayısı küçülürse ➔ ?
  - B sayısı büyürse ➔ ?
- Genişletilebilir Hash indekslemede **N büyük fakat** :
  - Bir disk sayfası çok sayıda bucket tarafından **ortak** kullanılıyor...
  - Bucket dizini (*drectory*), "bucket ➔➔ disk sayfası" eşlemesini gerçekleştiriyor.
- Örnek: 3 idx_records / block. N=8, H(*dataval*) = (*dataval*) mod **8**

  Dataval değerleri: 1,2,4,5,7,8,12

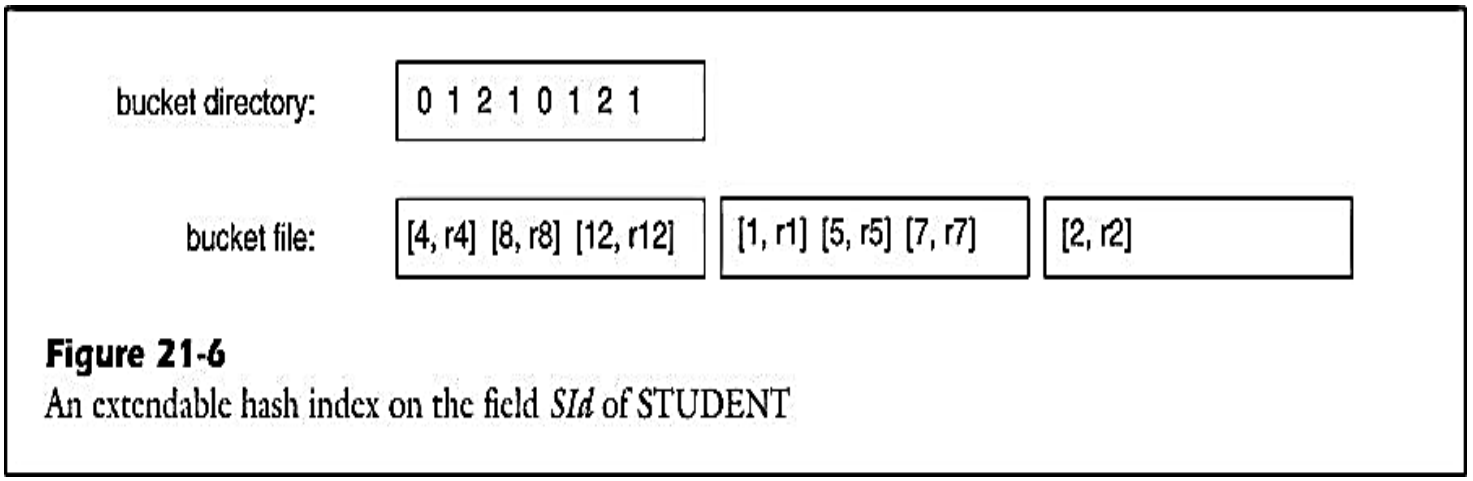
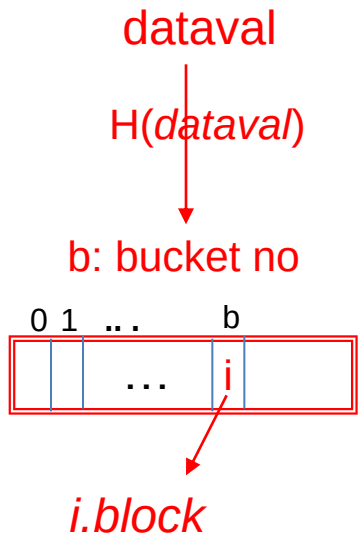
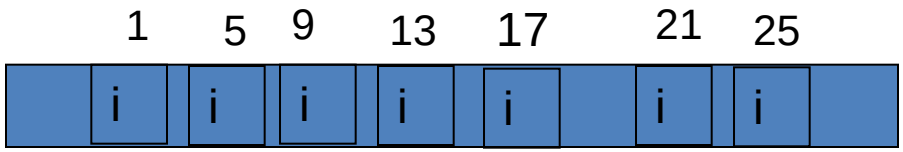


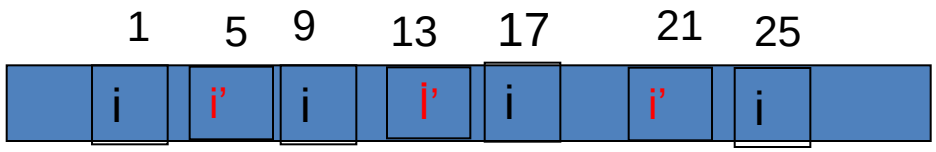
Figure 21-6
An extendable hash index on the field *SId* of STUDENT



Tanımlar

- **M** : maksimum derinlik: Üretilen hash adreslerinin bit sayısı. M=32 ➔ 2^{32} buckets
- **H(x)** = *f(x)* mod 2^{M}
- **L:** lokal derinlik (*L<=M*) : Blok için geçerlidir. Block içersindeki indeks kayıtlarının H(dataval) değerlerinin ortak bitlerinin sayısıdır.
- **D:** Global derinlik. D=max(L_0, L_1,……L_i, …)
- **SPLIT=** Bir **i** bloğu split ile ikiye bölündüğünde,
  - **Bloklardaki değişiklik: i.blok** ve yeni oluşan bloğun **(i'.blok)** derinliği **L+1** olur. **i.bloktaki** deki kayıtlar yeniden hash edildiğinde;
    - hash adresi **0**b_L....b_1 olan kayıtlar **i bloğuna**,
    - hash adresi **1**b_L....b_1 olanlar yeni bloğa, **i' bloğuna** yerleşir. *(Bu dizindeki eşleşmede yapılacak aşağıdaki değişiklik ile mümkündür)*
  - **Dizindeki değişiklik**: yeni eklenen kaydın hash adresi(bucket no): **b** olsun: **b'nin** en sağ L biti (b_L....b_1) ile aynı olan, dizindeki diğer tüm bucketlar **i.bloğa** işaret ediyor. SPLIT'den sonra,
    - *L+1. biti **0** olan, yani bucket no: **0**b_L....b_1 olan her bucket eski bloğa,* **i.bloğuna;**
    - *L+1. biti **1** olan, yani bucket no: **1**b_L....b_1 olan her bucket yeni oluşan bloğa,* **i' bloğuna** işaret edecek.





L=2

L=3

L=3

1. Hash the record's dataval to get bucket b.
2. Find $B = Dir[b]$. Let L be the local depth of block B.

3a. If the record fits into B, insert it and return.

3b. If the record does not fit into B:

- Allocate a new block B' in the bucket file.
- Set the local depth of both B and B' to be $L+1$.
- Adjust the bucket directory so that all buckets having the rightmost $L+1$ bits $1b_L...b_2b_1$ will point to B'.
- Reinsert each record from B into the index. (These records will hash either to B or to B'.)
- Try again to insert the new record into the index.

Figure 21-7
The algorithm for inserting a record into an extend

Maksimum derinlik :10

H(x) = x%1024

eklemeler: 4, 8,12,1, 5,7,2

H(x), düzenli dağılım olması koşulu altında sorun yok. Fakat tek bloğa yığılma olduğunda?

→ L<= **M** koşulu sağlanıncaya kadar split devam eder...

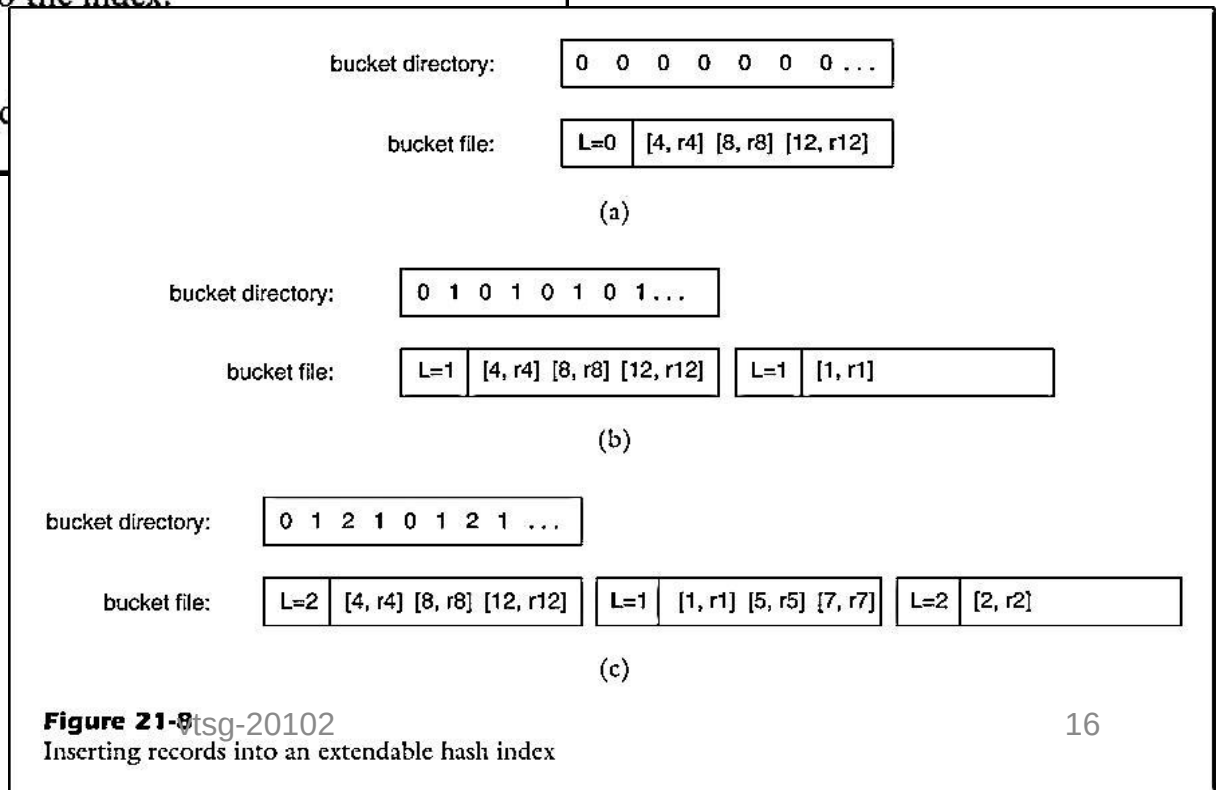
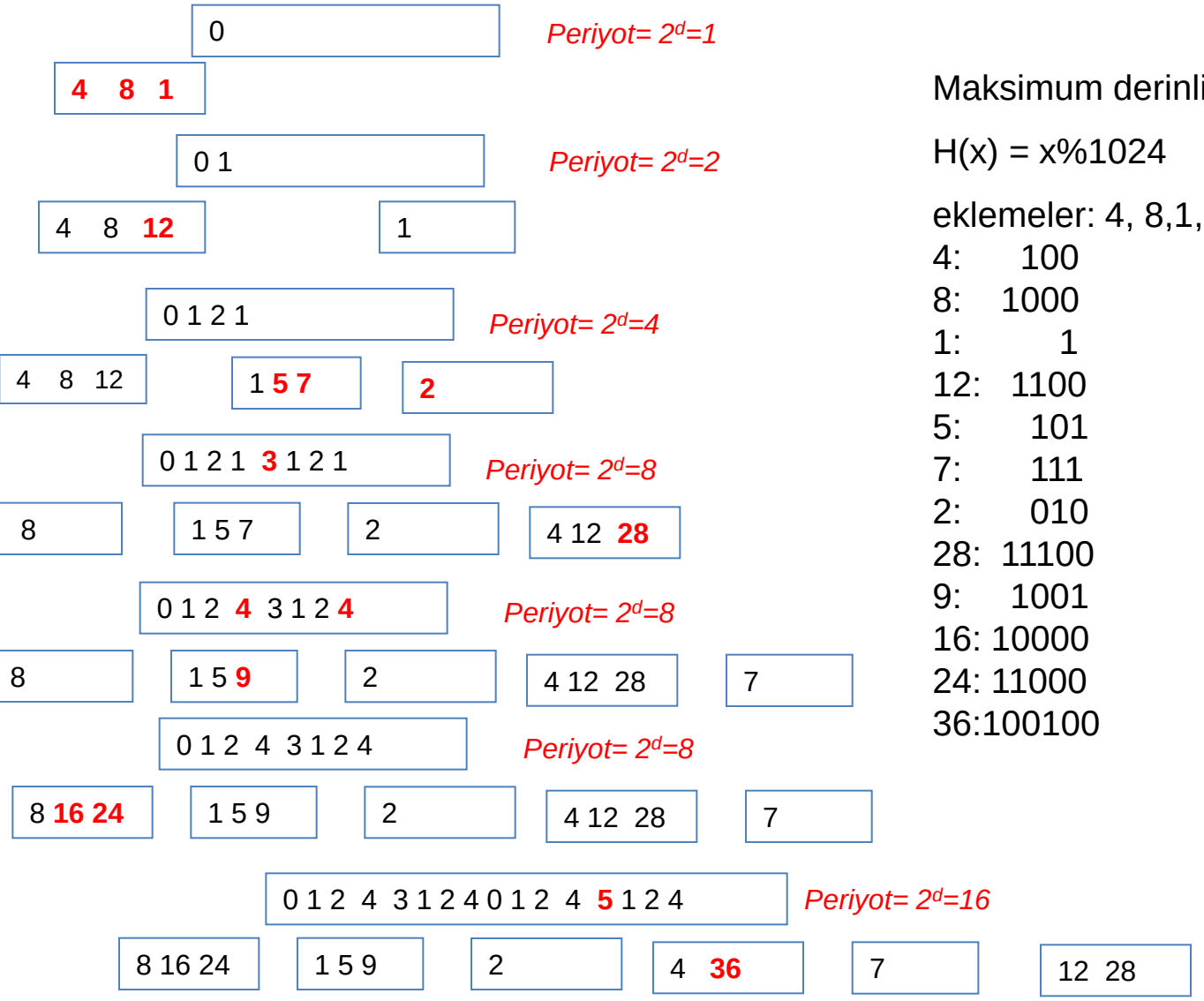


Figure 21-8
Inserting records into an extendable hash index

Genişletilebilir Hash indeksleme-Dizin düzenlemesi

- M=10 ➔ 2^{10} bucket
  - ➔ dizin büyüklüğü: 1 blok (B=4KB)
  - ➔ idx dosyası: 4MB
- M=20 ➔ 2^{20} bucket
  - ➔ dizin büyüklüğü: 1024 blok (B=4KB**): 4MB**
  - ➔idx dosyası: **4GB**
- Oysa; bucket dizin içinde belirli bir periyotta bilgi tekrar ediyor.
  - L=1 ➔ periyot: 2
  - L=2➔ periyot: 4
  - L=3 ➔ periyot: 8
- D *(D≤M)* = global derinlik = Maksimum L ➔ periyot: 2^D ile bilgi tekrar ediyor. O zaman; **dizin büyüklüğü= 2^D slot** yeterli.
  - Herhangi bir dataval için; H(dataval) değerinin sadece **D** bitine bakmak yeterli.
  - Yeni kayıt ekleme ile; split oluyorsa ve split olan blok global derinlik (**D**) değerini 1 arttırıyorsa; bu dizin değerini 2'ye katlayacak.

Genişletilebilir Hash indeksleme-Dizin düzenlemesi, ÖRNEK



Maksimum derinlik :10

H(x) = x%1024

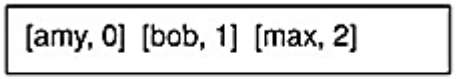
eklemeler: 4, 8,1,12, 5,7,2, 28, 9, 16, 24, 36

4: 100
8: 1000
1: 1
12: 1100
5: 101
7: 111
2: 010
28: 11100
9: 1001
16: 10000
24: 11000
36:100100

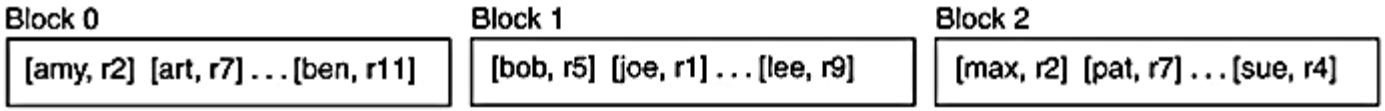
B-tree indeksleme *(literatürde B+-tree)*

- Düzenleme: Sıralanmış indeks dosyası

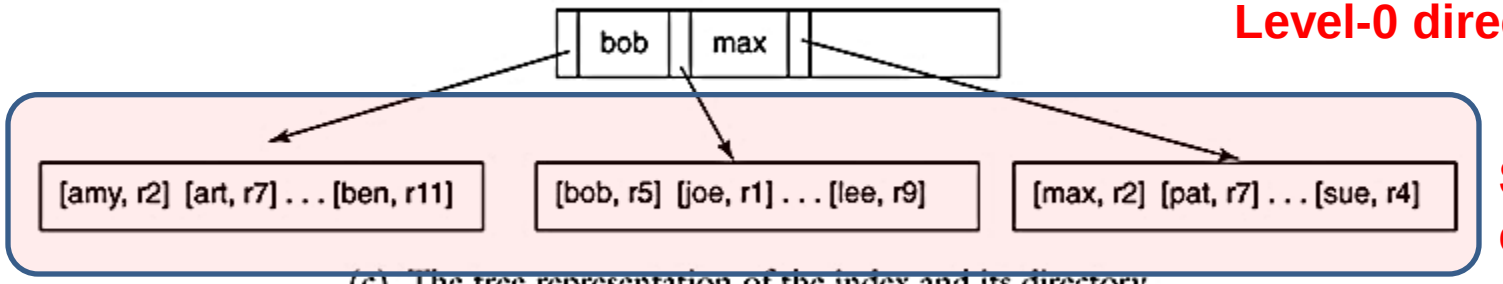
SId	SName	GradYear	MajorId
1	joe	2004	10
2	amy	2004	20
3	max	2005	10
4	sue	2005	20
5	bob	2003	30
6	kim	2001	20
7	art	2004	30
8	pat	2001	20
9	lee	2004	10



(b) The sorted level-0 directory



(a) The sorted index file

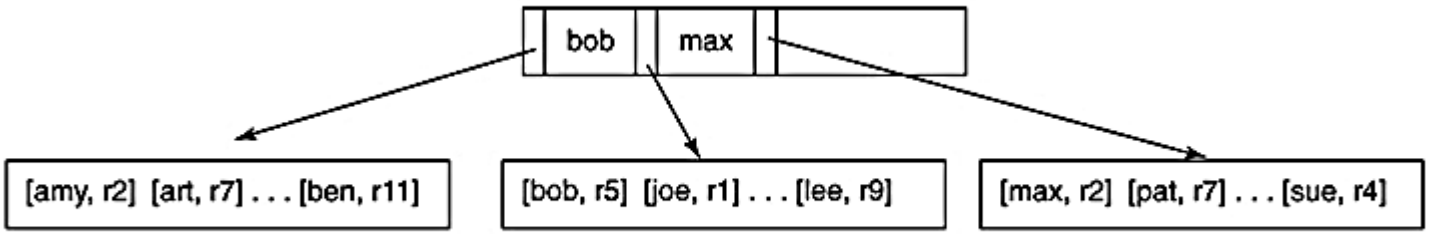


(c) The tree representation of the index and its directory

Level-0 directory

Simple İndeks dosyası

Kayıt bulma ve ekleme algoritması (*eksik*)



(c) The tree representation of the index and its directory

1. Search the directory block to find the directory record whose range of datavals contains *v*.
2. Read the index block pointed to by that directory record.
3. Examine the contents of this block to find the desired index records.

(a) Finding the index records having a specified dataval *v*

1. Search the directory block to find the directory record whose range of datavals contains *v*.
2. Read the index block pointed to by that directory record.
3. Insert the new index record into this block.

(b) Inserting a new index record having a specified dataval *v*

Figure 21-13
Algorithms to find and insert index records into the tree of Figure 21-12

B-tree dizin düzenleme

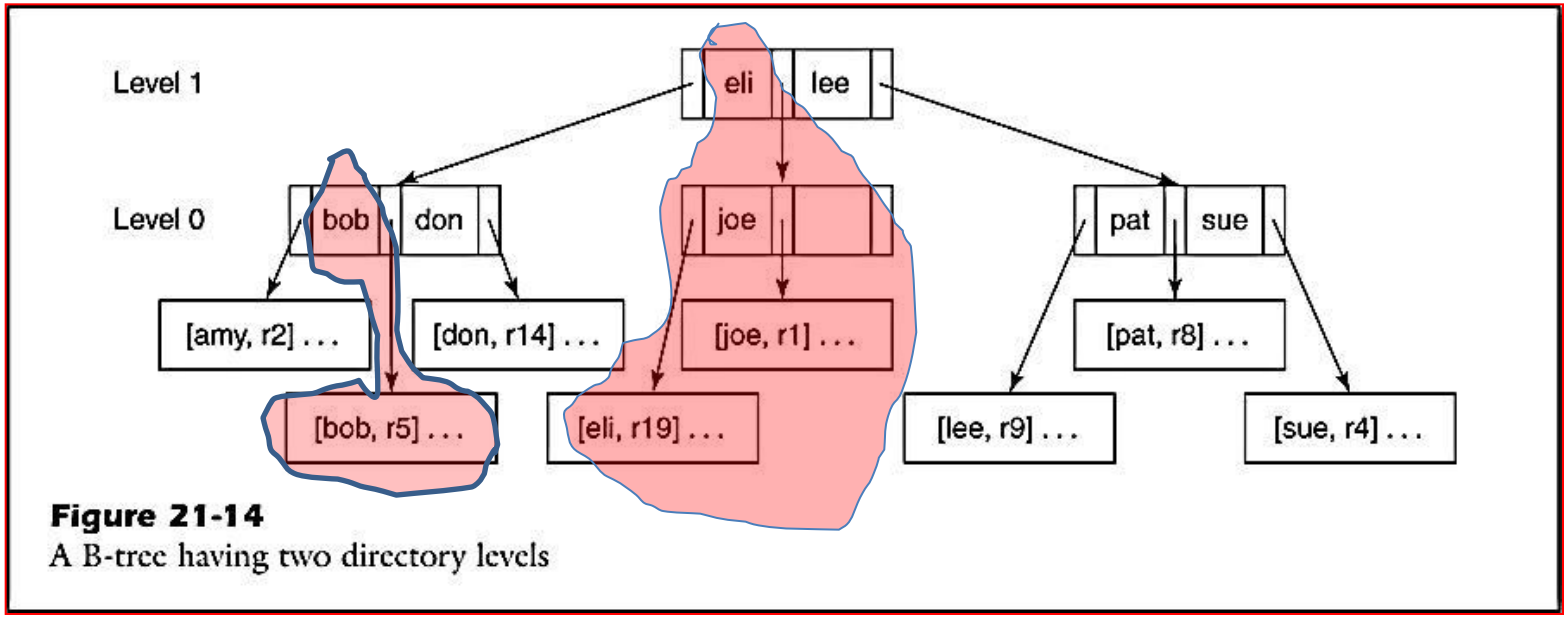


Figure 21-14
A B-tree having two directory levels

- Level-i'de >1 sayıda blok oluştuğu zaman, level-(i+1)'de yeni bir dizin bloğuna ihtiyacımız olur.
- Arama maaliyeti= toplam dizin seviyelerinin sayısı +1
- Level-i directory kayıtları: (dataval, blok_id@level-(i-1))
- Indeks kayıtları: (dataval, rid)
- Örneğin: 178 indeks_kaydı/blok, 227 dizin_kaydı/blok olsun.
  - Level-0 dizini: 227*178 adet indeks kaydı tutar.
  - Level-1 dizini: 227*227*178 adet indeks kaydı tutar.
  - Level-2 dizini: 227*227*227*178 = 2.082.080.774 adet indeks kaydı tutar.

B-tree kayıt eklemede split

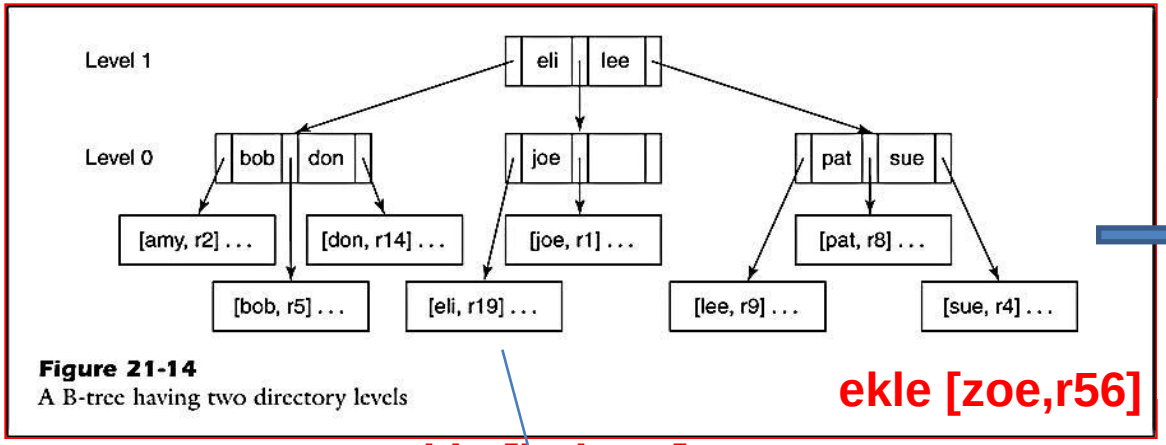
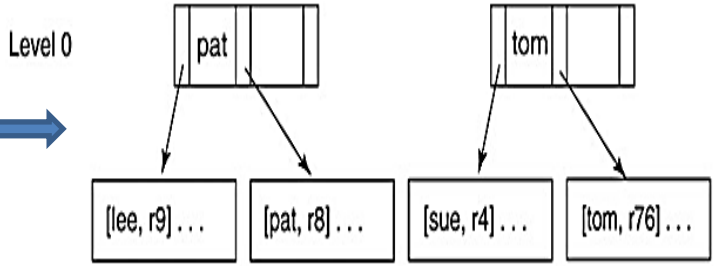
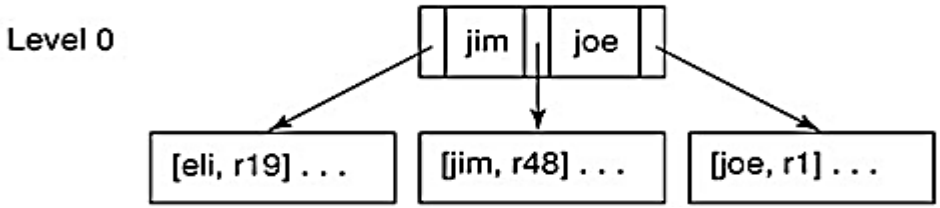


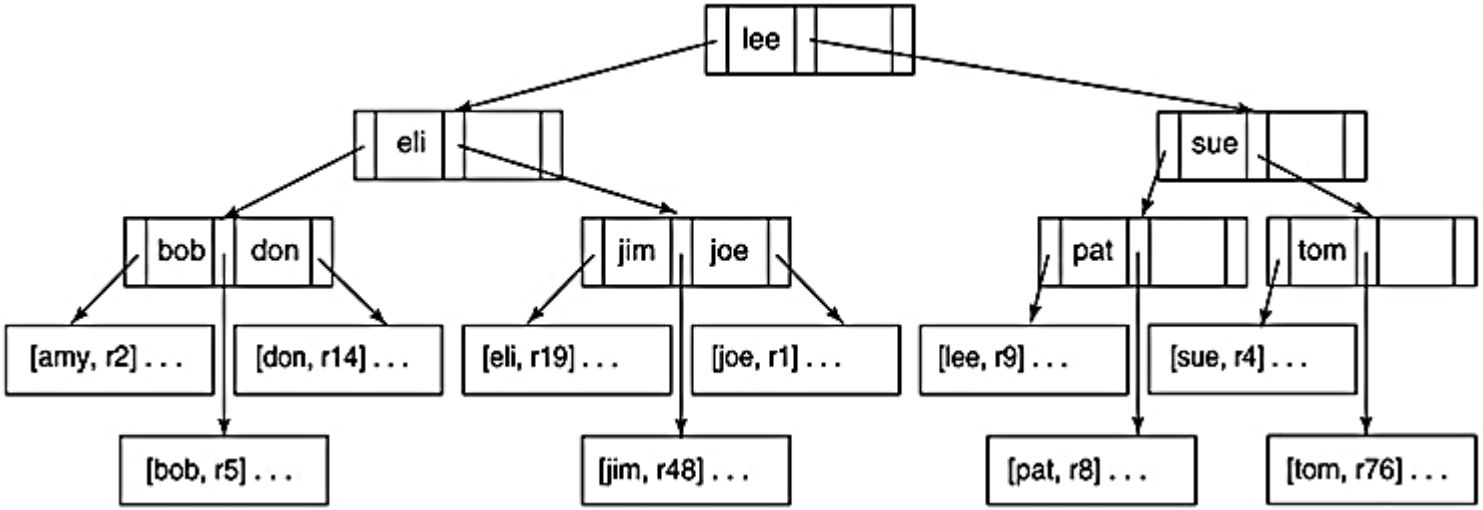
Figure 21-14
A B-tree having two directory levels

ekle [zoe,r56]



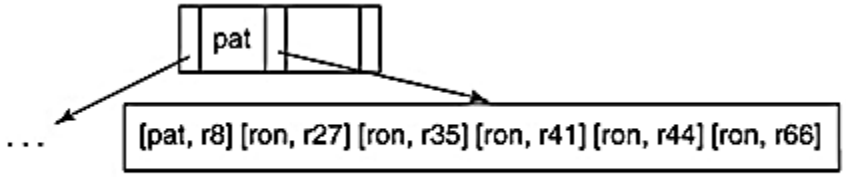
ekle [bal,r55]





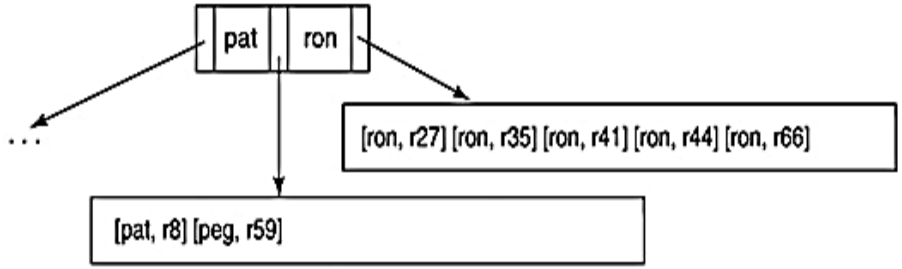
Tekrarlı kayıt eklemeleri

- Kural: Aynı dataval değerine sahip kayıtlar aynı indeks blok'da yer almalı.
- Ne zaman ki bu kural sağlanarak split mümkün olmadı; o zaman taşan blok (*overflow*) kullanılabilir.
- Directory kayıtları için tekrarlı yapı sorunu var mı?
- Tekrar sayısı arttıkça, idx seçiciliği azalıyor ➔ idx kullanılmaz hale geliyor.

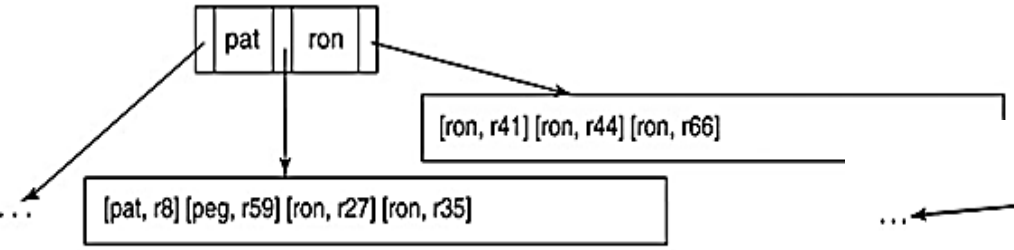


(a) The original leaf block and its parent

ekle [peg,r59]

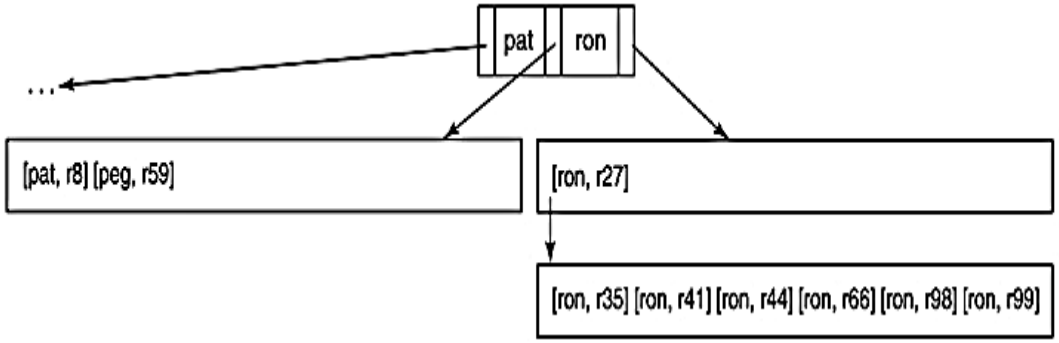


(c) The correct way to split the block



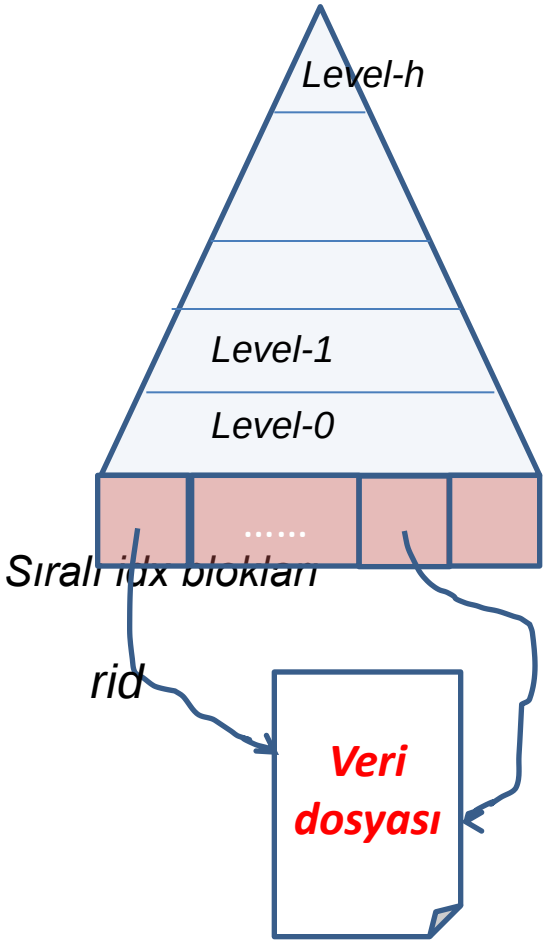
(b) An incorrect way to split the block

ekle [ron,r98]



B-tree gerçeklemesi

- Simpledb.index.btree paketi içerisinde:
  - BtreePage:
    - **Leaf ve Dir sayfalarının içeriğini tutar** ve ortak operasyonları gerçekler.
    - Kayıtlar sıralı *(Örneğin: yeni gelen kayıt nereye yerleşmeli findSortBefore(searchkey) ile bulunuyor..)*
    - Kayıt-ID sabit olmak zorunda değil..
    - Split ile komşu bloğa veri aktarımı
    - En başta bir bayrak: Dir ise «hangi seviyede olduğu» veya Leaf ise «taşan blok var mı bilgisini tutar»
  - **BtreeLeaf *(sıralı idx blokları)***
    - «Delete», «insert», «overflow» ile ilgili operasyonlar…
  - **BtreeDir**
    - Ağacın kökünden başlayarak «search», «insert», yeni kök oluşumunun geçekleyen operasyon
  - BTreeIndex :
    - Index arayüzünü gerçekler.
- Örneğin; SID_IDX'i, toplamda 2 dosya ile gerçekleniyor:
  - SID_IDXleaf
  - SID_IDXdir

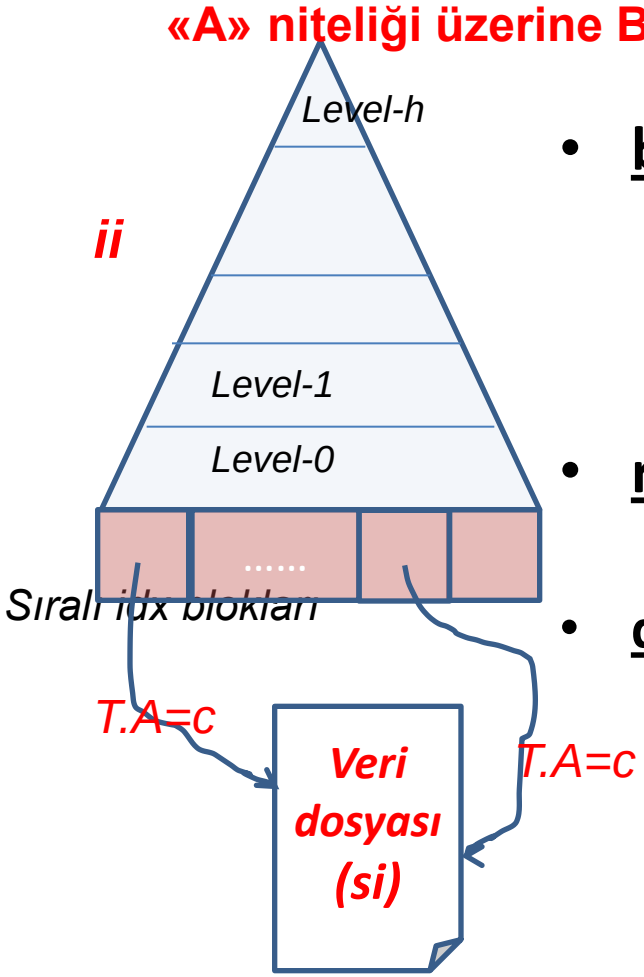


Indeks-duyarlı operatör gerçeklemeleri

- Planlayıcının görevleri:
  - En iyi ağaci seçmek
  - Ağaç içerisindeki operatörler için planı belirlemek
    - ProjectPlan
    - TablePlan
    - **SelectPlan → IndexSelectPlan**
    - **ProductPlan → IndexJoinPlan *(HashIndex, Merge-Join,BNL)***
  - **BasicUpdatePlanner → IndexUpdatePlanner**

IndexSelectPlan Maaliyeti

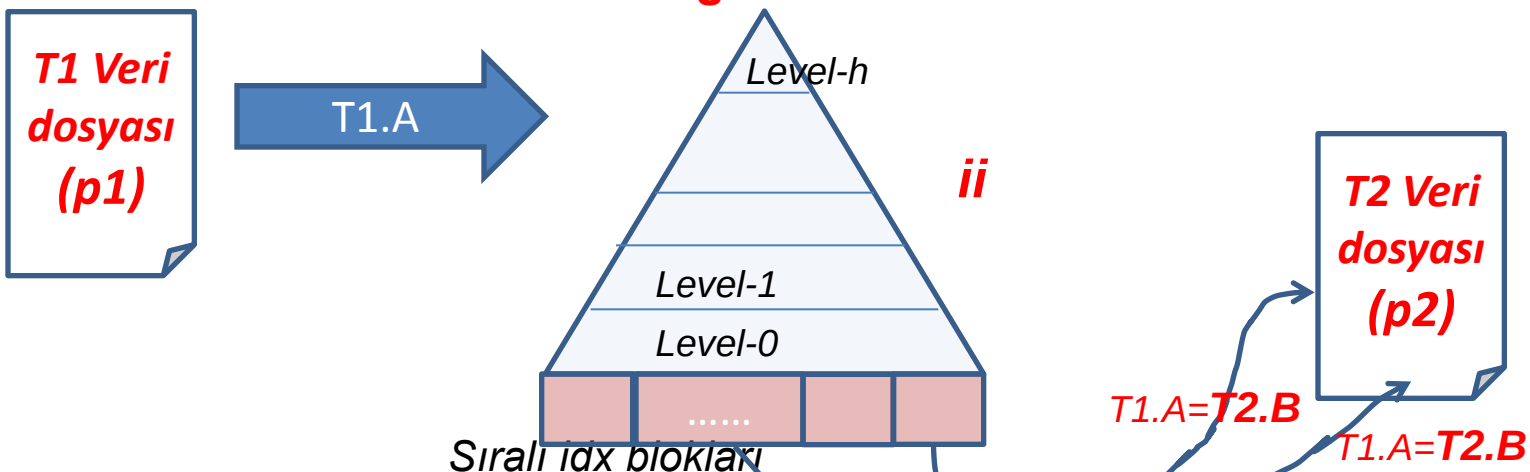
select(T,A=***c***) : T alt-ağaç ➔ T.A=***c***
Algo: Her bir T kaydı için
T.A =***c*** olan T kayıtlarını T.A indeksi üzerinden bul.



- **blocksAccessed()** =ii.blocksAccessed() +recordsOutput();
  - *ii.blocksAccessed() = log $_{rpb}$ (sıralı idx bloklarının sayısı)*
  - *rpb = (sıralı idx kayıt/block)*
- **recordsOutput()** = ii. recordsOutput();
  - ii.recordsOutput()= si. recordsOutput()/si.distinctValues(A)
- **distinctValues(F)** = ii.distinctValues ();
  - 1, eğer F=A
  - min(si.distinctValues(F), ii.recordsOutput()), F≠A

IndexJoinPlan Maaliyeti

Join(T1,T2,p) : T1 alt-ağaç, T2 saklı tablo, T2.B idx'i var, p ➔ T1.A=T2.B
Algo: Her bir T1 kaydı için
T1.A =T2.B olan T2 kaydını T2.B idx üzerinden bul



- **blocksAccessed()** =p1.blocksAccessed() +(p1.recordsOutput() * ii.blocksAccessed())
  + recordsOutput();
  - *ii.blocksAccessed() = log $_{rpb}$ (sıralı idx bloklarının sayısı)*
  - *rpb = (sıralı idx kayıt/block)*
- **recordsOutput()** = p1. recordsOutput()*ii. recordsOutput(); *(eğer JSF $_{T1.A}$=1 ise)*
  - ii.recordsOutput()= p2. recordsOutput()/p2.distinctValues(**B**)
- **distinctValues(F)** = p1.distinctValues (F), *eğer F, p1.schema'sı içerisindeyse* ***VE JSF $_{T1.A}$=1***
  p2.distincValues(F), *eğer F, p2.schema'sı içerisindeyse* ***VE JSF $_{T2.B}$=1***

Eğer ***JSF $_{T1.A}$*** ≠ 1 veya ***JSF $_{T2.B}$ ≠ 1*** *ise SorguIsleme/Sunu:12'deki tarama maaliyetleri kullanılır.*

Örnek1 *(tablo büyüklüğünün IndexJoin'e etkisi)*

- STUDENT(Sname,**MajorID**)
- DEPT (**DId**, DName)

S Sname,Dname

F STUDENT,DEPT

W MajorId = DId

- 45.000 STUDENT records,
  - 10 STUDENTS/disk_block
- 40 departments,
  - 20 DEPTs / block

- **MajorID_IDX kullanalım:**

B(DEPT) + R(DEPT)*idx_maliyeti
+ (eşleşen toplam STU. kayıtları)

- **DId_IDX kullanalım:**

B(STUDENT) +R(STUDENT)*idx_maliyeti
+ (eşleşen toplam DEPT kayıtları)

- «eşleşen toplam STU. Kayıtları» = «eşleşen toplam DEPT kayıtları»
- idx_maliyeti ihmal edersek, MajorID_IDX kullanmak daha iyi.

KURAL: IndexJoin kullanırken; küçük olan tablo sol tarafta *(outer loop)* olması daha iyi.

Örnek 2 *(JoinSelectionFactor'nin IndexJoin'e etkisi)*

- STUDENT(**SID**, MajorID)
- DEPT (DId, Dname, **Dtemsilci**)

S SId

F STUDENT,DEPT

W SId = Dtemsilci

$JSF_{STU.SID} = 0,5$

$JSF_{DEPT.Dtemsilci} = 1$

- 45.000 STUDENT records,
  - 10 STUDENTS/disk_block
- 40 departments,
  - 20 DEPTs / block

- **SID_IDX kullanalım:**

B(DEPT) + R(DEPT)*idx_maliyeti
+ (eşleşen toplam STU. kayıtları)

- **Dtemsilci_IDX kullanalım:**

B(STUDENT) +R(STUDENT)*idx_maliyeti
+ (eşleşen toplam DEPT kayıtları)

- «eşleşen toplam STU. Kayıtları» = «eşleşen toplam DEPT kayıtları»
- Bütün DEPT kayıtları SID_IDX'den bir STU kaydı bulacak. Fakat, çoğu STU kaydı Dtemsilci_IDX'den eşleşen kayıt bulmayacak.

KURAL: IndexJoin kullanırken; JSF değeri büyük olan tablonun sol tarafta *(outer loop)* olması daha iyi.

```
public class IndexSelectPlan implements Plan {
   private Plan p;
   private IndexInfo ii;
   private Constant val;

   public IndexSelectPlan(Plan p, IndexInfo ii,
                          Constant val, Transaction tx) {
      this.p = p;
      this.ii = ii;
      this.val = val;
   }

   public Scan open() {
      TableScan ts = (TableScan) p.open();
      Index idx = ii.open();
      return new IndexSelectScan(idx, val, ts);
   }

   public int blocksAccessed() {
      return ii.blocksAccessed() + recordsOutput();
   }

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

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

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

```
public class IndexSelectScan implements Scan {
   private Index idx;
   private Constant val;
   private TableScan ts;

   public IndexSelectScan(Index idx, Constant val,
                          TableScan ts) {
      this.idx = idx;
      this.val = val;
      this.ts  = ts;
      beforeFirst();
   }

   public void beforeFirst() {
      idx.beforeFirst(val);
   }

   public boolean next() {
      boolean ok = idx.next();
      if (ok) {
         RID rid = idx.getDataRid();
         ts.moveToRid(rid);
      }
      return ok;
   }

   public void close() {
      idx.close();
      ts.close();
   }

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

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

   public String getString(String fldname) {
      return ts.getString(fldname);
   }
   public boolean hasField(String fldname) {
      return ts.hasField(fldname);
   }
}
```

Figure 21-24
The code for the SimpleDB class *IndexSelectPlan*

```
public class IndexJoinPlan implements Plan {
   private Plan p1, p2;
   private IndexInfo ii;
   private String joinfield;
   private Schema sch = new Schema();

   public IndexJoinPlan(Plan p1, Plan p2, IndexInfo ii,
                        String joinfield, Transaction tx) {
      this.p1 = p1;
      this.p2 = p2;
      this.ii = ii;
      this.joinfield = joinfield;
      sch.addAll(p1.schema());
      sch.addAll(p2.schema());
   }
  public Scan open() {
     Scan s = p1.open();
     // assume that p2 is a table plan
     TableScan ts = (TableScan) p2.open();
     Index idx = ii.open();
     return new IndexJoinScan(s, idx, joinfield, ts);
  }

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

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

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

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

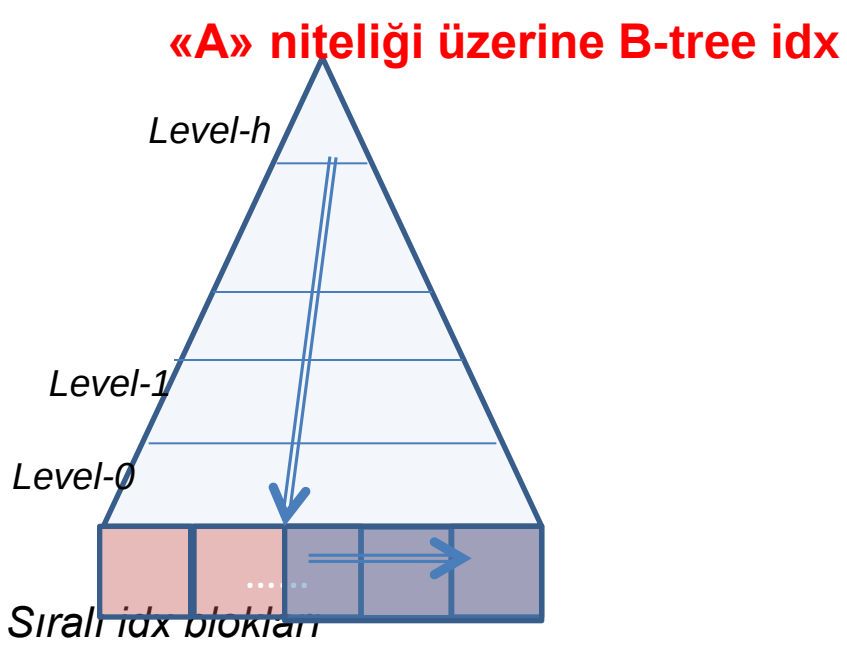
```
public class IndexJoinScan implements Scan {
   private Scan s;
   private TableScan ts;  // the data table
   private Index idx;
   private String joinfield;

   public IndexJoinScan(Scan s, Index idx,
                        String joinfield, TableScan ts) {
      this.s = s;
      this.idx  = idx;
      this.joinfield = joinfield;
      this.ts = ts;
      beforeFirst();
   }

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

   public boolean next() {
      while (true) {
         if (idx.next()) {
            ts.moveToRid(idx.getDataRid());
            return true;
         }
         if (!s.next())
            return false;
         resetIndex();
      }
   }

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

   public Constant getVal(String fldname) {
      if (ts.hasField(fldname))
         return ts.getVal(fldname);
      else
         return s.getVal(fldname);
   }
```

```
public class IndexUpdatePlanner implements UpdatePlanner {

   public int executeInsert(InsertData data,
                            Transaction tx) {
      String tblname = data.tableName();
      Plan p = new TablePlan(tblname, tx);

      // first, insert the record
      UpdateScan s = (UpdateScan) p.open();
      s.insert();
      RID rid = s.getRid();

      // then modify each field of the record,
      // inserting an index record when appropriate
      Map<String,IndexInfo> indexes =
               SimpleDB.mdMgr().getIndexInfo(tblname, tx);
      Iterator<Constant> valIter = data.vals().iterator();
      for (String fldname : data.fields()) {
         Constant val = valIter.next();
         s.setVal(fldname, val);

         IndexInfo ii = indexes.get(fldname);
         if (ii != null) {
            Index idx = ii.open();
            idx.insert(val, rid);
            idx.close();
         }
      }
      s.close();
      return 1;
   }
```

```
public int executeDelete(DeleteData data,
                         Transaction tx) {
   String tblname = data.tableName();
   Plan p = new TablePlan(tblname, tx);
   p = new SelectPlan(p, data.pred());
   Map<String,IndexInfo> indexes =
            SimpleDB.mdMgr().getIndexInfo(tblname, tx);

   UpdateScan s = (UpdateScan) p.open();
   int count = 0;
   while(s.next()) {
      // first, delete the record's RID from every index
      RID rid = s.getRid();
      for (String fldname : indexes.keySet()) {
         Constant val = s.getVal(fldname);
         Index idx = indexes.get(fldname).open();
         idx.delete(val, rid);
         idx.close();
      }
      // then delete the record
      s.delete();
      count++;
   }
   s.close();
   return count;
}
```

Örnek

- aralık sorgusu *(range query):* SELECT (T, A>***c***)



- Sıralı idx blokları; fakat «A_IDXleaf» dosyası sıralı değil.
  - **BTreeLeaf kodunda değişiklik**: Her **split** için iki bloğu birbirine eklemek gerekiyor ki; aralık sorguları index üzerinden işlenebilsin.
  - **IndexSelectPlan/Scanda değişiklik**:
    - Maliyet hesabı ve next()fonksiyonu değişmeli…

- **NOT: aralık sorgularında sadece B-tree veya benzeri sıralı index yapıları kullanılır.**