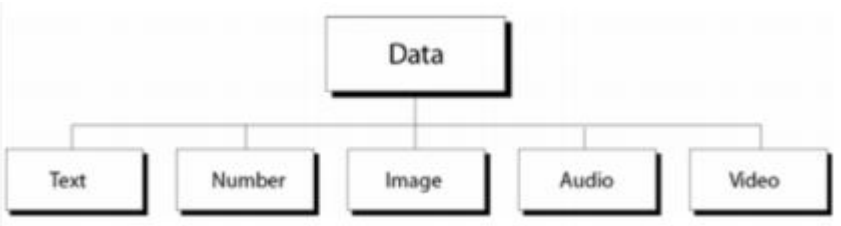
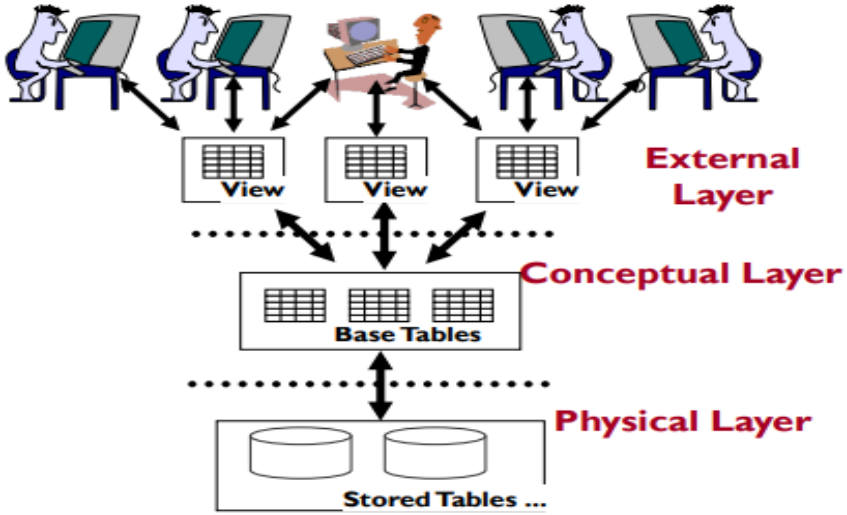
1) *Ders01 - Introduction:*



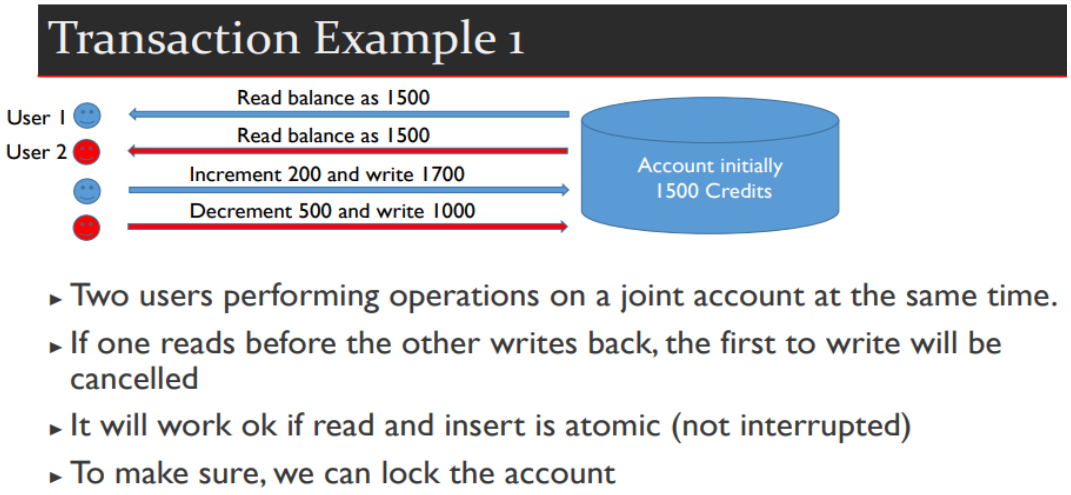
- **Data,** hemen hemen her türlü örgütlenmemiş gerçek(ler). Ör: Bilgisayar Dosyası.
- **Signal,** iletim için gerekli olan verilerin kodlanmasıdır.
- Veri işlendiğinde ve organize edildiğinde bilgi olur ve böylece yararlı olur.
- Komplex Verileri; ***relational (ilişkisel), graph (grafik)*** veya ***Structured (Yapısal)*** olarak ifade edilebilir.
- **VTYS,** bir yazılım paketidir, veritabanlarını depolamak ve yönetmek için tasarlanmış. Çok büyük, entegre bir veri koleksiyonu.
- **Neden VTYS kullanırız?**
  - ✓ Veri bağımsızlığı ve verimli erişim,
  - ✓ Veri bütünlüğü ve güvenliği,
  - ✓ Eşzamanlı erişim,
  - ✓ Azaltılmış uygulama ve geliştirme süresi,
  - ✓ Tekdüzen veri yönetimi,
  - ✓ Çökmelerden kurtarma.
- **Veri Modelleri,** verileri açıklamak için oluşturulan bir kavramlar topluluğudur.

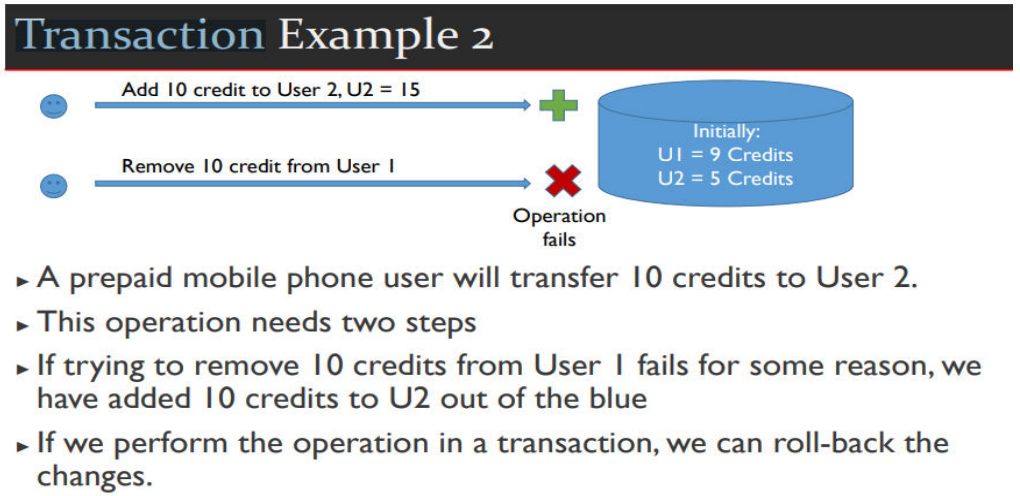


  - ✓ Bir şema belirli bir veri koleksiyonunun, verilen veri modelini kullanarak tanımlanmasıdır.
  - ✓ İlişkisel (relational) veri modeli bugün en çok kullanılan modeldir.
  - ✓ Ana konsept ilişki, temelde satır ve sütun içeren bir tablo.
  - ✓ Her ilişkide sütunları veya alanları tanımlayan bir şema vardır.
  - ✓ Şema şu şekilde tanımlanır: şema adı, her alanın adı (veya niteliği veya sütunu) ve her alanın türü. Ör: Öğrenci(numara: integer, isim: string)
- **Veri Bağımsızlığı,** verilerin nasıl yapılandırıldığından ve depolandığından yalıtılmış uygulamalar;

✓ ***Mantıksal Veri Bağımsızlığı:*** Verilerin mantıksal yapısındaki değişikliklerden korunma. (Güzel Açıklama :başparmak: :D)

✓ ***Fiziksel Veri Bağımsızlığı:*** Verilerin fiziksel yapısındaki değişikliklerden korunma.





2) Ders02 - Storage Devices:

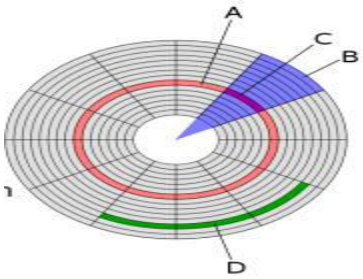
❖ **Disk Organizasyonu;**

✓ ***Platters:*** Verileri depolamak için her iki tarafı kullanabilir.

✓ ***Track:*** Verilerin manyetik olarak saklandığı dairesel bir yoldur (A şeklinde gösterilir).

✓ ***Sector:*** Parçanın bir alt bölümü. (C)Normal olarak sektör, iki yarıçap ve yay ile çevrelenmiş disk kısmı anlamına gelir.(B)

✓ ***Cluster:*** Aktarılan veri birimi, çoklu sektörler (örn. 3 sektör) cinsinden tanımlanır (D olarak gösterilir). Bu genellikle bir dosya sistemi ile ilgili ayardır.



✓ ***Sector:*** Veri depolama için minimum parça bölümü.

✓ ***Cluster:*** Sektörler grubu

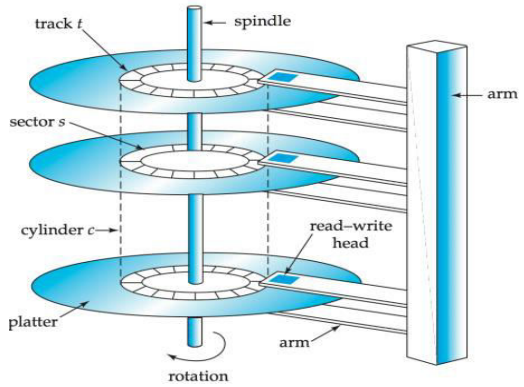
✓ ***Track:*** Platter'da bir döngü.

✓ ***Cylinder:*** Plakalardaki aynı seviye izlerden oluşan semantik bir şekil.

✓ ***Platters,*** istenen sektör için pozisyona dönüyor.

✓ ***Kol tertibatı,*** kafayı istenilen bir parça üzerine konumlandırmak için içeri veya dışarı doğru hareket ettirilir.

- ✓ Sadece bir kafa herhangi bir zamanda okur / yazar.
- ✓ ***Blok boyutu,*** sektör büyüklüğünün (sabit olan) bir katıdır.



❖ **Bloklama (Blocking);**

- ✓ ***Fiziksel Blok,*** Birincil ve ikincil hafıza arasında aktarılan minimum veri
- ✓ ***Mantıksal Blok,*** Birincil ve ikincil hafıza arasında transfer edilen minimum kayıt seti
- ✓ ***Engelleme Faktörü,*** Mantıksal blok başına mantıksal kayıt sayısı
- ✓ Varsayalım, bir küme = bir blok.

❖ **Disk Kapasitesinin Tahmini;**

- ✓ ***Track Capacity = Number Of Sectors Per Track * Bytes Per Sector***
- ✓ ***Cylinder Capacity = Number Of Tracks Per Cylinder * Track Capacity***
- ✓ ***Driver Capacity = Number Of Cylinder * Cylinder Capacity***
- ✓ ***Access Time = Seek Time + Rotational Delay***
- ✓ ***Time To Read = Seek Time + Rotational Delay + Transfer Time***
- ✓ ***Transfer Time = (Number Of Sectors Transferred / Number Of Sectors On a Track) * Rotation Time***

▸ 20 plate, 800 track/plate, 25 sector/track, 512 byte/sector, 3600 rpm
▸ 7 ms from track to track, 28 ms avg. seek time, 50 msec max seek time

Avg. Rotational time	=1/2 * (60000/3600) = 16.7/2 = 8.3msec
Disk capacity	= 25 * 512 * 800 * 20 = 204.8 MB
Time to read track	= 1 rotation = 16.7 msec
Time to read cylinder	= 20 * 16.7 = 334msec
Time to read whole disc	= 800 * time to read cylinder + 799* time to pass from one cylinder to another =800*334 + 799*7msec =267 sec + 5.59 sec = 272.59 sec

▸ Disc specs: avg. seek time 8 msec, 10.000 rpm, 170 sector/track, 512 byte/sector

Time to read one sector = avg. seek time + rotational delay + time to transfer one sector
= 8 + (0.5 * 60.000 / 10.000) + (6* 1/170) = 8 + 3+ 0.035 = 11.035 msec

Time to read 10 sequential blocks = avg. seek time + rotational delay + 10 * time to transfer one sector = 8 + 3 + 10 * 0.035 = 11.35 msec

Time to read random 10 block= 10 * (avg. seek time + rotational delay + time to transfer one sector)= 10 * (8 + 3 + 0.035) = 113.5 msec

You have total 200.000 records stored in heap file and length of each record is 250 byte. Here are the specifications of disk used for storage: 512 byte/sector, 20 sector/track, 5 sector/cluster, 3000 track/platter, one sided total 10 plate, rotational time 7200 rpm, seek time 8 msec. Records do not span between sectors

- ▸ Blocking factor of heap file: 10
- ▸ Number of blocks to store whole file: 20.000 blocks
- ▸ Number of blocks to estimate disk capacity: 120.000 blocks
- ▸ Cylinder number to store the data with cylinder based approach : 500
- ▸ Time to read one block : 14,225 ms
- ▸ Time to read file from beginning to end (in worst case): 284.500sec, 4741,67 min, 79,02 hour

❖ **RAID;**

- ✓ ***RAID Seviye 1:*** Her diski, bir veri diski, diğeri de yedek disk olarak yansıtırız.
- ✓ ***RAID Seviye 2:*** Parity bloklarını depolayan sadece bir yedek disk kullanın.
- ✓ Aynı anda iki veya daha fazla disk çökmesi varsa ne olur?
- ✓ ***RAID Seviye 5:*** Her disk, bazı bloklar için yedek bir disktir.
- ✓ Örnek: 4 disk varsa, disk 0, 4, 8, 12, vb. Bloklar için yedeklidir; disk 1; 1, 5, 9 vb. için gereksizdir.
- ✓ ***RAID Seviye 6:*** Birden çok disk çökmesi için tasarlanmıştır. Hamming kodunu kullanır.

3) Ders03 - File Concepts:

❖ **Pin_count,** sayfayı kullanan işlemlerin sayısını takip etmek için kullanılır. "0" kimse kullanmadığı anlamına gelir.

❖ **Dirty,** diskten okunduğundan beri bir sayfanın değiştirildiğini belirtmek için bayrak (kirli bit) olarak kullanılır. Sayfa havuzdan çıkarılacaksa, diski yıkamanız gerekir.

Sample Buffer Pool

Page_no = 1 Pin_count = 3 Dirty = 1 Last Used: 12:34:05	Page_no = 2 Pin_count = 0 Dirty = 1 Last Used: 12:35:05	Page_no = 3 Pin_count = 1 Dirty = 0 Last Used: 12:36:05	Page_no = 4 Pin_count = 2 Dirty = 0 Last Used: 12:37:05	Page_no = 5 Pin_count = 0 Dirty = 0 Last Used: 12:38:05
Page_no = 6 Pin_count = 0 Dirty = 0 Last Used: 12:29:05	Page_no = 7 Pin_count = 1 Dirty = 1 Last Used: 12:20:05	Page_no = 8 Pin_count = 0 Dirty = 1 Last Used: 12:40:05	Page_no = 9 Pin_count = 2 Dirty = 0 Last Used: 12:27:05	Page_no = 10 Pin_count = 0 Dirty = 1 Last Used: 12:39:05

Which page should be removed if LRU is used as the policy:..... 6

Which page should be removed if MRU is used as the policy :..... 8

Which pages do not need to be written to disc, if it is removed:..... 5, 6

Which pages could not be removed in this situation:..... P.c != 0

❖ Genellikle, (bf is blocking factor. N is the size of the file in terms of the number of records) :

❖ ***At least 1 block is accessed (I/O cost : 1),*** ***At most N/bf blocks are accessed.*** *On average N/2bf.*

❖ ***Time To Fetch One Record = (N/2bf) * Time To Read One Block***

❖ ***Time To Read One Block = Seek Time + Rotational Delay + Block Transfer Time***

❖ ***Time To Read All Records = N/bf * Time To Read Per Block***

❖ ***Time To Add New Record = Time To Read One Block (For Last Block) + Time To Write One Block (For Last Block)***

❖ ***If the last block is full; Time To Add New Record= Time To Read One Block (For Last Block) + Time To Write New One Block (For New Last Block)***

❖ ***Time To Update One Fixed Length Record = Time To Fetch One Record + Time To Write One Block***

❖ ***Time To Update One Variable Length Record = Time To Delete One Record + Time To Add New Record***

- FileA: 10000 records , BF = 100, 4 extents
- File B: 5000 records, BF = 150, 3 extents

- Time to find the number of common records of FileA and B

  Time to read FileA= 4 * (seek time + rotational delay) + (10000/100) * block transfer time

  Time to read FileB = 3 * (seek time + rotational delay) + (5000/150) * block transfer time

  = Time to read FileA + 100 * Time to read FileB

  (imagine you've got only two frames in the buffer pool.)
- Read FileA and compare each record of FileA with whole records in FileB

- We can do binary search (assuming fixed-length records) in the sorted part.

- Worst case to fetch a record :

$$T_F = \log_2 (m/bf) * \text{time to read per block.}$$

- If the record is not found, search the overflow area too. Thus total time is:

$$T_F = \log_2 (m/bf) * \text{time to read per block} + k/bf * \text{time to read per block}$$

4) *Ders04 - Index Files:*

❖ **Birkaç Terim;**

✓ Bir ***veri dosyası*** tüm kayıtları saklar.

✓ Veri dosyası bir veya daha fazla ***dizin dosyasına*** sahip olabilir.

✓ Dizin dosyaları, ***arama anahtarları*** ve ***işaretçileri*** veri dosyası kayıtlarına kaydeder.

✓ Dizinler ***birincil*** veya ***ikincil*** olabilir.

✓ ***Birincil Dizin:*** Veri dosyası, arama anahtarına göre yapılandırılmıştır. Örneğin. Masanın birincil anahtarı.

✓ ***İkincil İndeks:*** Tablonun diğer bazı özelliklerine dayanarak, veri dosyasındaki kayıtların sırası arama anahtarından bağımsızdır.

✓ ***Sıralı dosya:*** Kayıtlar birincil anahtara göre sıralanır.

❖ **Dense Index:** [https://www.youtube.com/watch?v=lg8S2s_yTh4]

✓ Bir dosyadır.

✓ Ilk satırı işaret eder.

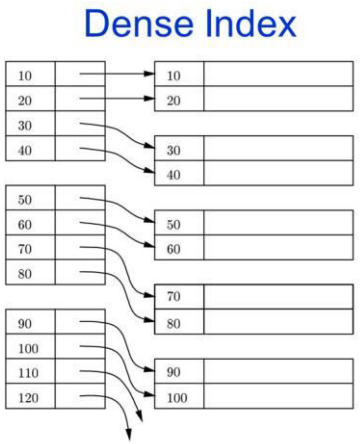
✓ Key - Pointer (Her kayıt için)

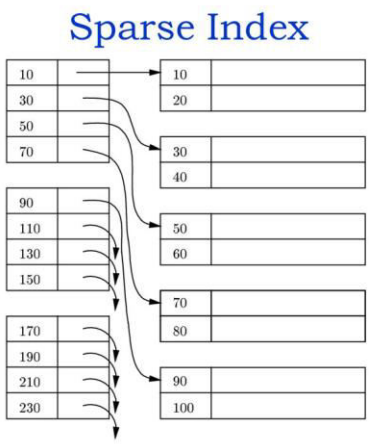
❖ **Sparse Index:**

✓ Bir dosyadır.

✓ En düşük arama anahtarını işaret eder.

✓ Key - Pointer (Her blok için)





- Suppose there is a data file of 4 GB (2^{32}) in a system with blocks of 1KB and fixed length records of 256Bytes.
- The records are stored in sorted order with respect to the key Student ID.
- Index stores a search key of 4 Bytes and a 4 Bytes of pointer. So, an index entry is 8 bytes.
- How many disk accesses do we need to find a record with a given Student ID:
  - Using sorted data file
  - Using dense index
  - Using sparse index

- $2^{32} / 2^{10} = 2^{22}$ blocks are in the data file.
- Blocking Factor of data file = $2^{10}/2^{8} = 2^{2}$
- 2^{22} x BF of data file = 2^{24} records in the file
- With binary search we can have 22 disk accesses to find the record we are searching for in the worst case.
- If an index entry is 8 bytes we can fit into a block $2^{10}/2^{3}=2^{7}$ entries
  - Dense Index should be: $2^{24}/ 2^{7}=2^{17}$ blocks = 2^{17} x 2^{10} = 2^{27} = 128MB index file. So, a binary search is 17 disk accesses.
  - Sparse Index should be: $2^{22}/2^{7}=2^{15}$ blocks = 2^{15} x 2^{10} = 2^{25} = 32MB index file. So, a search is 15 disk accesses.
- What would happen if we had a two-level sparse index?

5) Ders05 - Cost:

Cost Model for Our Analysis

We ignore CPU costs, for simplicity:

- B: The number of data pages
- R: Number of records per page
- D: (Average) time to read or write disk page
- Measuring number of page I/O's ignores gains of pre-fetching a sequence of pages; thus, even I/O cost is only approximated.
- Average-case analysis; based on several simplistic assumptions.

Operations to Compare

- Scan: Fetch all records from disk
- Equality search
- Range selection
- Insert a record
- Delete a record

Cost of Operations

	Heap File	Sorted File	Hashed File
Scan all recs	BD	BD	1.25 BD
Equality Search	0.5 BD	D $\log_2$B	D
Range Search	BD	D ($\log_2$B + # of pages with matches)	1.25 BD
Insert	2D	Search + BD	2D
Delete	Search + D	Search + BD	2D

6) *Ders06 & Ders07 - Hast Tables:*

Load Factor

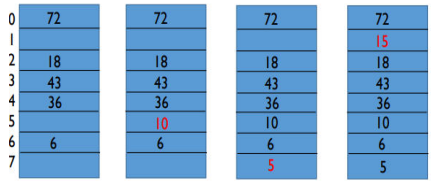
- Loading factor (LF), α = n / m
  n: number of keys
  m: number of slots
- If uniform distribution (1/m) to get mapped to a slot, a slot will have an expectation of α elements.
- If m increases
  - Collision decreases
  - LF decreases
  - 0.5 > LF > 0.8 is unacceptable
  - Storage requirements increases.
- Reduce collisions while keeping storage requirements low.

Linear Probing

$$h(k,i) = (h'(k) + i) \bmod m$$

- Always check the next index
- Increments index linearly with respect to i.
- Clustering problem

hash(10) = 2
hash(5) = 5
hash(15) = 7

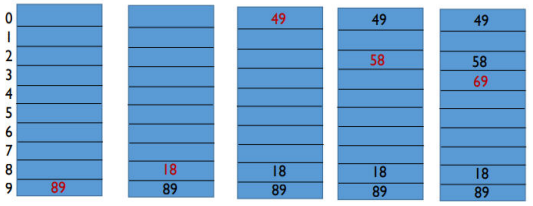


Open Addressing – Quadratic Probing

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

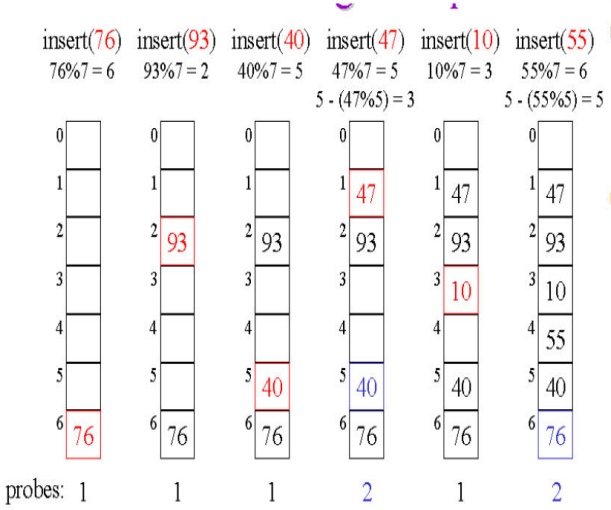
- Instead of moving by one, move i^2

c_1=0, c_2=1
hash(89)=9
hash(18)=8
hash(49)=9
hash(49, 1) = 0
hash(58) = 8
hash(58, 1) = 9
hash(58,2) = 2
hash(69) = 9
hash(69,1) = 0
hash(69,2) = 3



Closed Hashing >>> https://www.cs.usfca.edu/~galles/visualization/ClosedHash.html **Open Hashing >>>** https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

Example 2 – Double Hashing

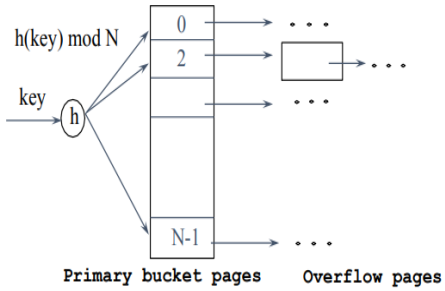


Dynamic Hashing Methods

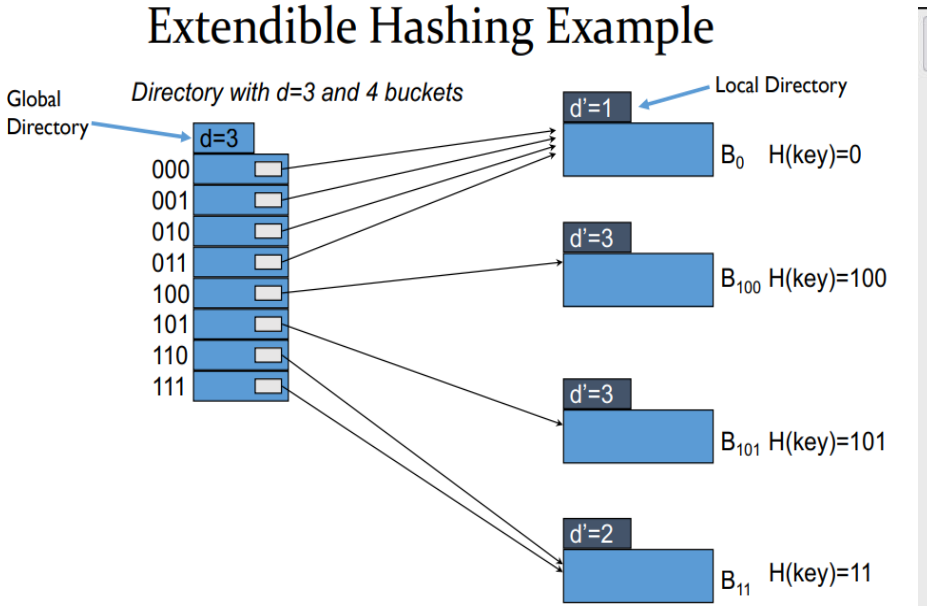
- As for any index, 2 alternatives for data entries **k***:
  - <**k**, rid of data record with search key value **k**>
  - <**k**, list of rids of data records with search key **k**>
  - Choice orthogonal to the *indexing technique*
- *Hash-based* indexes are best for *equality selections*. ***Cannot*** support range searches.

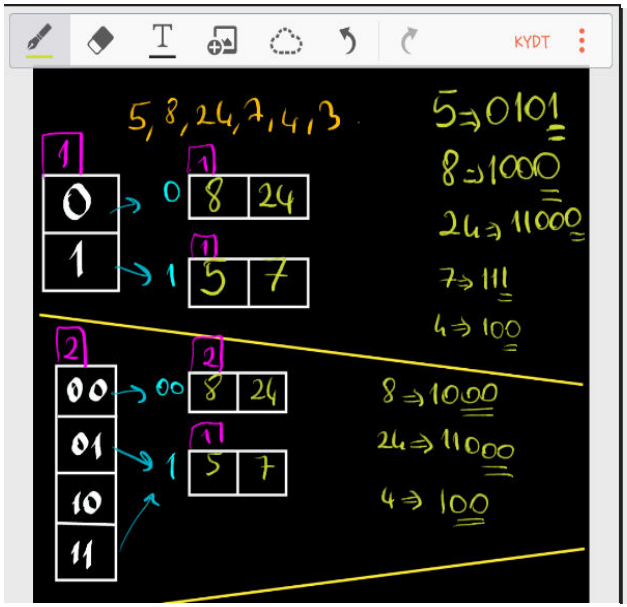
Static Hashing

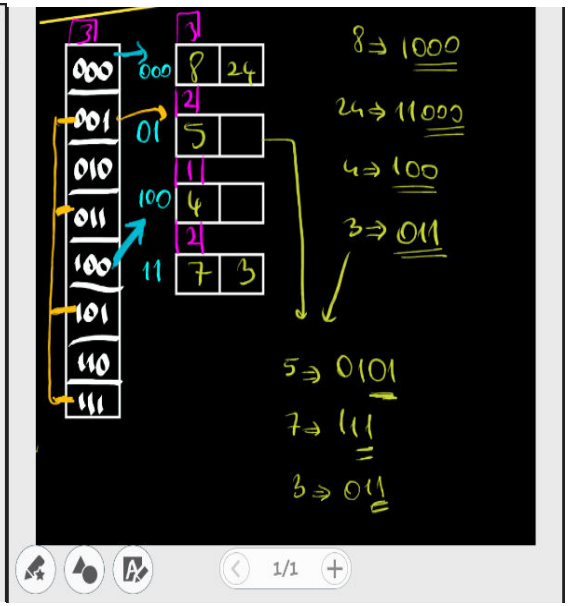
- # primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- **h**(*k*) mod M = bucket to which data entry with key *k* belongs. (M = # of buckets)



Extendible Hashing >>> https://www.youtube.com/watch?v=TtkN2xRAgv4&t=519s







Linear Hashing With Next Pointer >>> https://www.youtube.com/watch?v=h37Jhr21ByQ

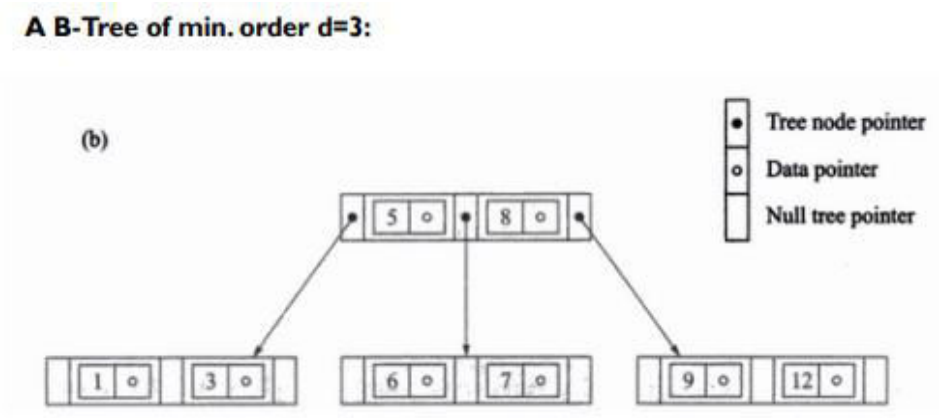
7) Ders08 - Tree Based Indexing:

❖ **Hash temelli endekslemenin** eşitlik araması (linear search) için kullanıldığını ve aralık araması (range search) için çalışmadığını unutmayın.

❖ Aralık tabanlı arama (range search) için **ağaç tabanlı indeksleme (tree based indexing)** (ve eşitlik araması için de) kullanılır.

❖ **B TREE;**

- ✓ Bir B-ağacında, her bir düğüm, çok sayıda anahtar içerebilir.
- ✓ B-ağacı, çok sayıda yöne dallanmak ve her düğümde çok sayıda anahtar bulundurmak için tasarlanmıştır, böylece ağacın yüksekliği nispeten küçüktür.
- ✓ Ağaç her zaman dengelidir.



❖ **B Ağacı Unsurları;**

- ✓ Minimum "d+1" order'a sahip B Ağacı'nın özellikleri:

1. İç düğümler (kök hariç) en az d, en fazla 2d anahtarlarına sahiptir.
2. Kök (bir yaprak değilse) en az 2 çocuğa sahiptir.
3. Tüm yaprak düğümleri aynı seviyededir (dengeli ağaç).
4. Seviye artışı ve azalışlar yukarı doğru (aşağı değil) yapılır.
5. İç düğümde en az d ve d + 1 çocuk var.

❖ **B TREE Visualization >>> https://www.cs.usfca.edu/~galles/visualization/BTree.html**

❖ **B+TREE;**

- ✓ B + ağacı her zaman dengeli.
- ✓ Kök düğüm hariç her düğüm için en az yüzde 50'lik bir kullanım garantisi verilir.
- ✓ Bir kaydın aranması, kökten uygun kanala geçiş yapılmasını gerektirir.
- ✓ Sadece yaprak düğümleri veri girişleri içerir.
- ✓ İç düğümler dizin içerir.
- ✓ Her veri girişi bir yaprak sayfasında görünmelidir.
- ✓ Yaprak düğümleri iki kez birbirine bağlanır. (Doubly Linked List)

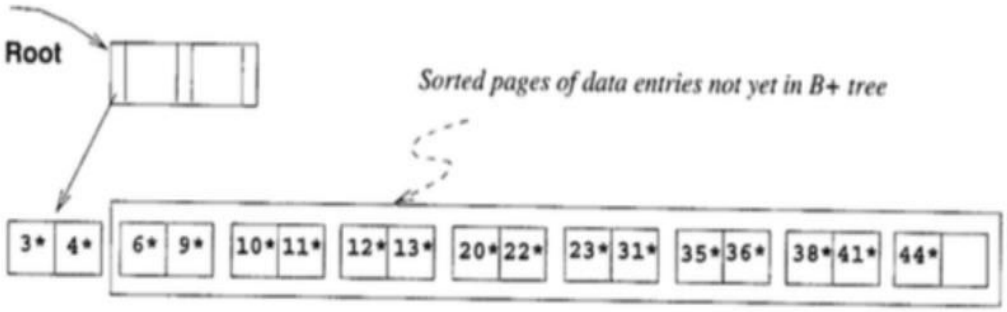
The insert algorithm for B+ Tree

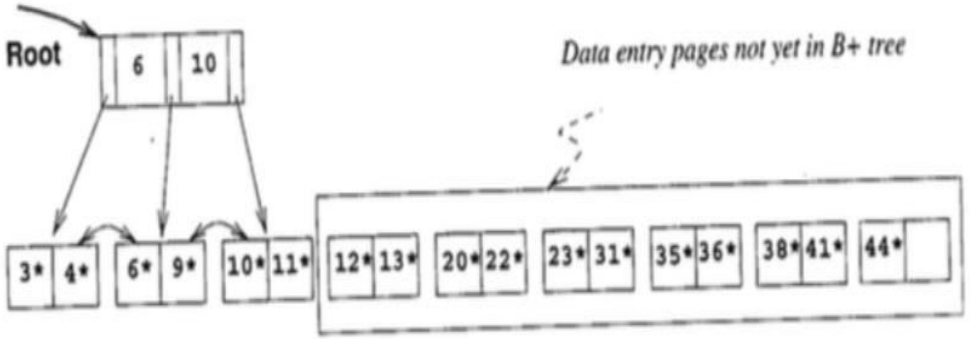
Data Page Full	Index Page Full	Action
NO	NO	Place the record in sorted position in the appropriate leaf page
YES	NO	1. Split the leaf page 2. Place Middle Key in the index page in sorted order. 3. Left leaf page contains records with keys below the middle key. 4. Right leaf page contains records with keys equal to or greater than the middle key.
YES	YES	1. Split the leaf page. 2. Records with keys < middle key go to the left leaf page. 3. Records with keys >= middle key go to the right leaf page. Split the index page. 4. Keys < middle key go to the left index page. 5. Keys > middle key go to the right index page. 6. The middle key goes to the next (higher level) index. IF the next level index page is full, continue splitting the index pages.

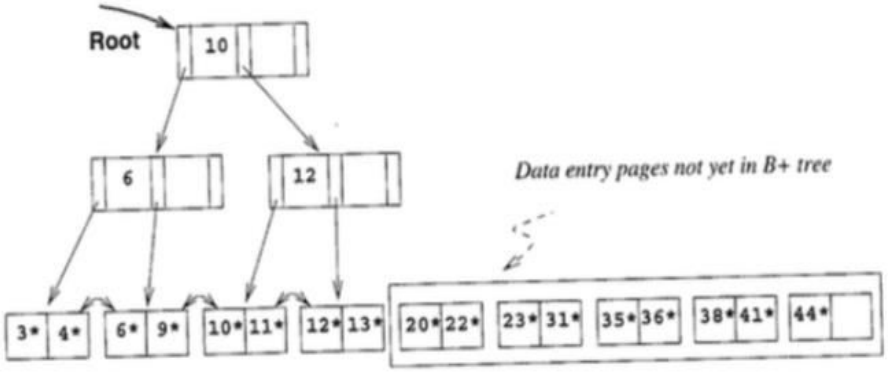
▸ Delete algorithm for B+ trees

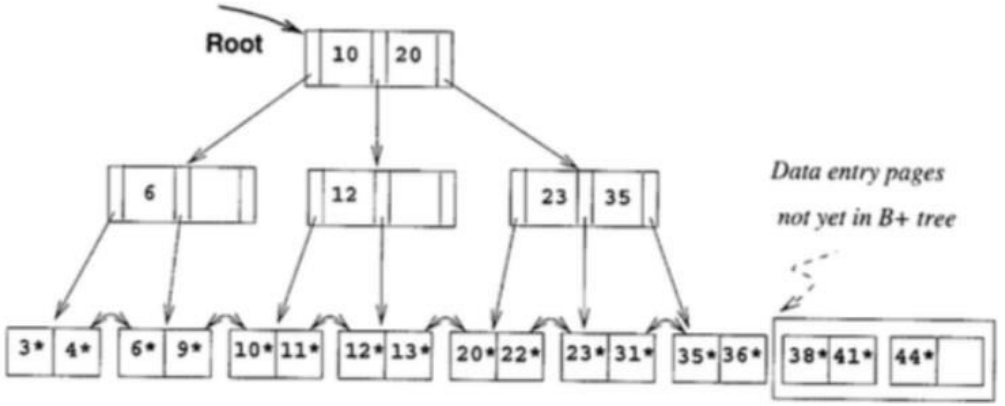
Data Page Below Fill Factor	Index Page Below Fill Factor	Action
NO	NO	Delete the record from the leaf page. Arrange keys in ascending order to fill void. If the key of the deleted record appears in the index page, use the next key to replace it.
YES	NO	Combine the leaf page and its sibling. Change the index page to reflect the change.
YES	YES	1. Combine the leaf page and its sibling. 2. Adjust the index page to reflect the change. 3. Combine the index page with its sibling. Continue combining index pages until you reach a page with the correct fill factor or you reach the root page.

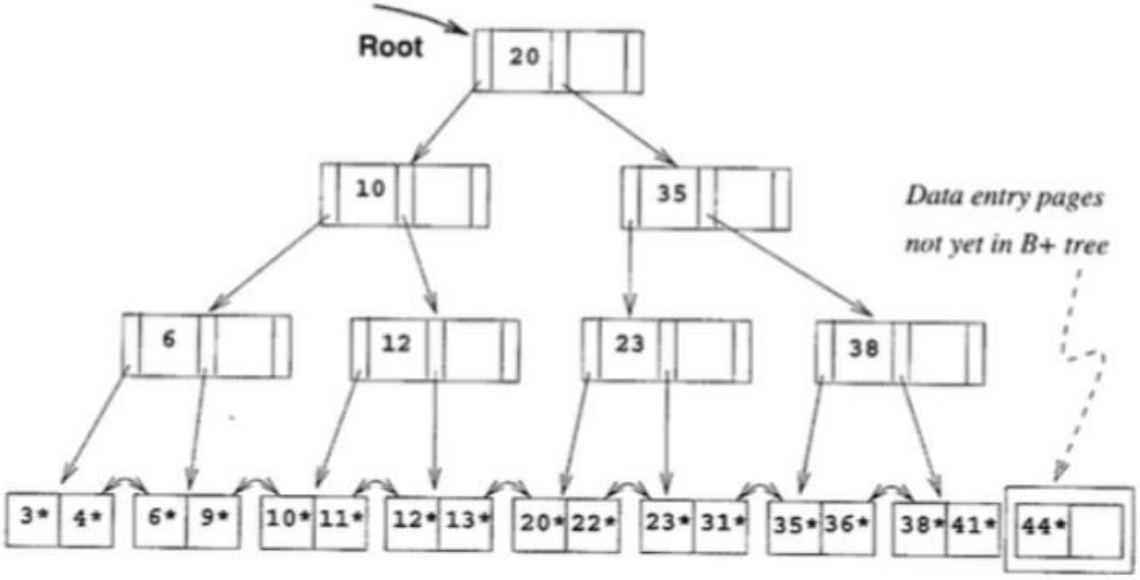
- **B+TREE Visualization >>> https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html**
- Bulk Loading in B+TREE;
  - ✓ Girişlerin birer birer eklenmesi pahalıdır, çünkü her bir eklemede yaprak düğümüne kadar kökten arama yapılmasını gerektirir.
  - ✓ Bunun yerine, eklenecek tüm veriler bir veri girişleri koleksiyonu olarak sıralanır ve eklenir.






Root
10
6
12
Data entry pages not yet in B+ tree
3* 4*
6* 9*
10* 11*
12* 13*
20* 22*
23* 31*
35* 36*
38* 41*
44*


Root
10 20
6
12
23 35
Data entry pages
not yet in B+ tree
3* 4*
6* 9*
10* 11*
12* 13*
20* 22*
23* 31*
35* 36*
38* 41*
44*


Root
20
10
35
Data entry pages
not yet in B+ tree
6
12
23
38
3* 4*
6* 9*
10* 11*
12* 13*
20* 22*
23* 31*
35* 36*
38* 41*
44*