**VIDEO (7 Dakikada NOSQL Nedir?)**

Not only SQL

Iliskisel veritabanlari (RDBMS) = Oracle, Sybase, MS SQL Server, PostgreSQL, MySQL

Iliskisel veritabanlariyla NoSQL in karsilastirilmasini inceleyecegiz.

1) Iliskisel Veritabanlari 1) NoSQL

1) Scale up – Veri buyudukce verinin tutuldugu kutunun buyumesi gereklidir.

2) Scale Out – Verilerin tutulmasi gereken kutular burda da veriyle birlikte buyur ama burda kutular sinirsizdir iliskisel veritabaninda ise kutular sinirlidir.

1) Structured Data

2) Semi Structured, UnStructured

1) Atomic Transactions (ACID) – Bir islem bitmeden oteki baslayamaz. (mesela bankada para transferi)

2) Eventual Consistency (BASE) – Tam tersi olarak calisir. Bir islem durdurulup oteki yapilabilir. Veriler RAM de durdugu icin elektrik kesintisinde veriler kaybolabilir.

1) Schema on Write (Object Relatioanal Impedance Mismatch) – Programlama dilinde kullandiginiz classlari falan tanimlayan sema ile veritabanini tanimlamak icin kullanilan semanin uymamasi durumu.

2) Schema in Read (OOP) – OOP sevenlerin daha sevdigi bir ortam vardir. Semalar birbiriyle uyusur.

**8.2 VIDEO (Dosya Yapisi (File Organization)**

Su anda aslinda tum veritabanlari isletim sistemi gibi calisiyorlar yani veri nasil kaydedilecekten tutun RAM de nasil bulunduracagina kadar tum herseyi veritabani zaten bilir ve yapar.

Record size = Kayit boyutu biz bunu konuyu kolay anlayalim diye sabit alacaz sonra nasil degistigini gorecez zaten.

Fixed-Length Records

Recordun konumuna erisebiliyoruz. Her bir recordun kac byte oldugunu biliyorsak onunla carpip yerini buluruz. Diyelim ki 1 recordumuz 1 mB ‘n \* (i-1)’ formulunden faydalanarak ulasmak istedigimiz veriyi biliriz. (i = kac record oldugu sayisi, n = byte sayisi)

Peki mesela aradan bir kayit silindi bu durumda erismek istedigimiz veriler, recordlarimizin sayisi falan hersey oynayacak yani bunu nasil yapabiliriz kolayca tekrar bulmak icin recordumuzu.

1) Silinen recorddan sonra gelen tum recordlar bir basamak asagiya kaydirilir. (Kotu bir yaklasim cunku tum diski neredeyse kaydiriyorsunuz)

2) Silinen recordun yerine son recordu koyma. Bu da 2. yontemdir. (Bu da biraz kotu cunku biz genelde diskte bir islem yapmak istemeyiz cunku diskin uzerinde tasima islemleri RAM’I mesgul eder ve vakit alici bir islemdir, maliyetlidir – Mesela o sirada cok daha onemli bir is yapilacak ama disk recordlarinin kaydirilmasi sistemi mesgul ediyor)

3) Bos alanlarin linklerini tutmak (Link List gibi pointerlarla mesela) Bu yontemle birlikte atiyorum yeni bir kayit geldiginde iste ilk silinen yerin konumu mesela 3 oraya yazilacak iste 2. Silinen recorundun konumu 15 atiyorum 2 gelen yeni record da 15 e yazilacak boyle boyle gidecek.

Variable-Length Record (Degisen boyutlu recordlar)

Recordun nerden basladigini ve boyutunu bilirsek bu sayede ne kadar alan kapladigini bilir ve sonraki verinin de nerden baslayacagini bilip yine boyutuna gore o uzunlukta diske yerlestirilir.

Ilk basta verilerin nerden basladigi ve ne kadar uzunluga sahip olduklari tutulur daha sonra bir isaretle bu baslik kisminin bittigini duyurulur ve sonra verilerin tutuldugu kisim baslar.

Mesela dynamic allocation gibi dusunursek anlayabiliriz. Atiyorum ben veritabaninda varchar(8) ile bir column tanimladik bu gidip hafizadan 8 byte yer ayiracak evet ama atiyorum ben daha dusuk karakterli birsey girersem 8 byte’in hepsini tutmasina gerek yok ayni dynamic bellekte oldugu gibi veri ne kadarsa o kadar tutacak aksi taktirde oraya char(8) yazmamizdan bir farki kalmaz iste bu durumlar icin biz varchar yani variable char kullaniriz bu sekilde degisken veri tiplerine ihtiyac duyariz.

Bu yaklasimda ayni sekilde yukarida anlattigim gibi degisken boyutlarda islem yapabilmemize olanak saglar ;)

Slotted page yaklasimi var.

Bunlarin disinda sunlar kullanilabilir. (Alt baslikta)

Organization of Records in Files

Heap – Agac seklinde erisime yarayan veri yapilarindan hatirlayabilirsin.

Sequential

Hashing – Index olusturmak yerine verilere hizli erisim icin hash’lerin tutulmasidir.

Multitable Clustering File Organization

Data Dictionary Storage

Sistem katalogu olarak gecer. Metadatalari icerir. Ust bilgileri icerir yani. Iste mesela atiyorum bir tabloyla otekini tablonun iste ID’sini birlestir ama birlestirecegimiz 2. tabloda ID numarasi yok e biz bunu nasil ogrenecez tablonun icine girip var mi yok mu diye kontrol mu edecez? Hayir gidip iste Data Dictionary’den bunun bir ID’sinin olmadigini ogrenecegiz. Bu tarz ornekler iste…

Ve bu data dictionary cok basvurulan bir yapi oldugu icin cabuk erisilen bir yerde olmasi gerekir genelde veritabanlari data dictionary’i RAM de tutarlar.

Storage Access

Bu sekilde veriyi RAM de tutarak daha hizli erisebiliyoruz veriye bu aslinda su siralar populer olan NoSQL ve Big Data kavramlarinin da felsefesini olusturur. Bu iki kavramda veriyi hic diskte tutmayip herseyi RAM de tutup ve RAM’i verimli kullanmayi amaclarlar ve buna calisirlar. Klasik veritabanlarinda ise malum RAM daha pahali oldugu icin diskten (disk cok daha ucuz) verileri diskte tutmayi ve bu sekil gerekli seylerin RAM’de tutulmasi saglanir. Iste bu amacla hem verileri diskte tutup maliyeti kisip hem de diskteki verilere erisirken RAM’i kullanarak hizli erisim gibi yaklasimlar gelistirilmistir. Bunlardan birisi buffer kullanmaktir.

Buffer – Tampon bellek (ara bellek). Yani veriler diske yazilacaksa once buraya yaziliyor sonra diske yaziliyor. Verileri bu sekilde arada tutmak da belki veri hic diske gitmeden direkt ordan hizli bir sekilde diske gitmeden isi cozecegiz.

Buffer-Replacement Policies

– Yani butun verileri buffer’da tutamiyoruz zaten tutsak diske ihtiyacimiz olmaz ;) Belli bir sinirda veriyi dolduruyoruz e sonra buffer doldu. Iste burda veriyi sondan silersek eger yani son kullanilmis yani en eski veriyi silerek ‘least recently used (LRU strategy)’ bos yer aciyoruz yeni gelen veriye. Verilere RAM’de hizlica erisim islemlerimizi yapmaya isletim sistemlerinde cash deniyor. Burada da ayni sekilde cash’lerin degisim kurallari falan var (cash-replacement policies).

Pinned block – Data dictionary’ler boyle kullanilabilir yani soyleki bu veriler onemli bunlari silme demek gibi bisey oluyor. Data dictionary surekli RAM’de kalacak yani silinmeyecek. Veya iste cok erisilen veriler yine boyle onemli veri gibi tutulup hic bufferdan silinmeyebilir.

Toss-immediate – Aninda degistir. Yani biris geldi hemen onu degistir iste oteki geldi hemen onu degistir seklinde calisir.

Most recently used (MRU) strategy – En son islenen veriyi tutmak uzerine calisir.

**9.1 VIDEO (Indexing)**

Buradaki hashing (ozetleme diyebiliriz – veriyle ilgili maksimum bilgi iceren ama kucuk bir parcasini cikartmaya yarayan olaydir yani veriden kucuk ama en cok bilgiyi de o bulunduruyor) ve indeks kavramlarini veritabanina gore inceleyecegiz. Yani mesela cryptology’de de hashing diye bir kavram var ama farkli bir sey yani aslinda ayni hashing ayni islemi yapiyor ama farkli amaclar icin kullaniliyor. Veya farkli yerlerde de bu terimler gecebilir ama biz veritabani icin konusacaz.

Basic Concepts

Soyle dusunelim zaten veritabanimizda veriler duruyor bu verilere daha hizli erismek icin ne yapabiliriz.

Kullanilan taktiklerden birisi verilerin her birinin indexini olusturmaktir ayni fihrist gibi hani iste kitabin uzerinde A harfi var mesela hemen orayi aciyorsun A ile ilgili seyler gozukuyor mesela iste A ile baslayan isimler buna gore indexleniyor ve hemen A ile ilgili islemleri gormek istedigimizde hizlica goruyoruz falan tarzi.

Su anki veritabanlari daha cok okuma yapmak icin tasarlanmislardir daha az yazma. (yani update islemleri daha az olur select islemleri cok daha fazla olur. Aslinda dusundugumuzde veri yazmak icin de okumaya ihtiyacimiz var nasil ki mesela veriyi yazacakken atiyorum banka hesap transferinde bizim once bu kisinin banka hesabina erismemiz lazim ki oraya sonra veri yazabilelim yani once okuyacaz ki bi sonra yazabilelim.

Su anki veritabanlari okumayi hizli yapmaya calisir yani veriye hizli ulasmayi amaclar aksi takdirde yazma hizli olup okuma yavas olsa mesela bu durumda ATM’ye kartimizi taktigimizda bizim kartimizi tanimlayip okumasi 1 saat alabilir ki bu da hic istenmeyen bir durumdur takdir edersiniz ki.

Iste simdi veritabaninda nasil index olusturuyoruz, nasil hizli erisiyoruz yani bu nasil calisiyor ona bakacagiz.

Btree binary tree degildir.

Mesela dusunecek olursak hani ayni binary treede oldugu gibi kucuk deger sola buyuk deger kokun sagina yazilir seklindeki yaklasim indexte uygulanir ama atiyorum adam simdi cv’sini yazmis hayat hikayesini anlatmis bunu oteki biriyle karsilastirdigimizda hangisi daha buyuk hangisi kucuk diye bir karsilastirma yapamayacagimiz icin bu gibi durumlarda index yerine hash kullanilmasi daha mantiklidir.

Yani ozet olarak bizim iki amacimiz var. Bir aranan deger var yani neye gore arama yaptigimiz (search key) bir de bu aranan degerin bulunmasi durumunda veritabaninda, diskte nereye gidecegimiz var (pointer).

Index Evaluation Metrics

Access time – veriye erisme zamani

Insertion time – veriyi ekleme zamani

Deletion time – veriyi silme zamani

Space overhead – bir veriyi tutmak icin ne kadar fazla yer kapladigimiz bilgisidir. Mesela veri yapilarinda counting sort vardir cok hizli calisir cok iyidir ama cok fazla yer kaplar. Dolayisiyla hem hizi hem de hafizayi birlikte dusunmemiz lazim.

Ordered Indices

- Yani soyle dusunursek atiyorum elimizde milyonlarca tablo var biz yine bu tablolara ulasmak icin bir index olusturabiliriz ama atiyorum bir tabloya bu indexle hizlica ulastiktan sonra bir de o tablonun icinde milyarlarca veri oldugunu dusunursek orda o veriye hizlica erismek bir daha sorun hale geliyor. Ve burada en cok aranan kelimeler yani birincil kelimeler veya iste en cok aranan istenen yapilar yani her neyse bunlarin uzerine bir index olusturulur. Genelde primary key uzerinden olusturulur bu indexler (primary index – verinin nasil tutuldugunu da iceren basic islemleri gerceklestiren indextir – clustering index de denir)

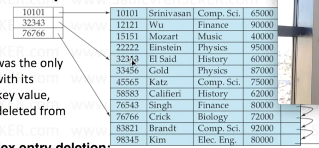
- Secondary index – Bu baglamda ise atiyorum iste biz bir ID numarasina index olusturduk ama bir de iste atiyorum bankanin sube adi cok kullaniliyor veritabanimizda ve biz bu veriye de hizli erismek istedigimizden buna da bir index olusturuyoruz bu da secondary index oluyor. (non-clustering)

(Dense Index – Veritabaninda bulunan verilerin buyuk bir kisminin indexlenmesi mesela ID numarasinin indexlenmesi. Yogun olmayan indexse mesela atiyorum iste ID’si 3’ten kucuk olanlarin indexlenmesi boyle bir durumda veritabanindaki cogu veriyi kapsayamadigimiz icin bu yogun olmaz – sparse index)

Multilevel index

Yani bir index olusturduk o indexin icinde bir index daha olusturuyoruz ve en icteki veriye erisirken mesela ilk index direkt ikinci indexe atliyor iletisim kuruyorlar yani aralarinda veri veri bulunuyor hizli bir sekilde.

Index Update: Deletion



Mesela bu sekilde sparse indexleme vardir ve atiyorum biz iste Katz kisisini silmek istedik yani ID si 45565 olan kisiyi simdi indeximize baktigimizda iste 4 ile baslayan ID nerde benim indeximde 3 ile baslayan ve 7 ile baslayanin arasinda o zaman 3 ile baslayanin oldugu yere hizlica gider sonra ilerler silmek istedigimiz veriyi bulup sileriz bu durumda cok bir sikintimiz yok cunku sildigimiz veri indeximizde yer almiyor zaten ;) ama eger indexte olan veri silinirse bu durumda yeni eklenen veriyi silinen indexin yerine yazmakta yani indeximizie yeni gelen veriyi eklemekte fayda vardir cunku eger yazmazsak bu durumda mesela biz 32343 u sildik veritabanimizdan ama bu veri hala indeximizde duruyor bu durumda indeximizde aslinda tabloda olmayan bir veri bulundurmus oluruz ki bu kotu biseydir tabiki.

Dense indexlerde ise zaten verinin buyuk bir kismi kapsandigi icin veri her silindiginde ya da degistirildiginde indexin de ayni sekilde bu islemlere uygun olarak guncellenmesi gereklidir. Bu yuzden sparse indexlerde bu islemler daha rahattir.

Mesela multilevel index olmasi durumunda ise icteki indexten distaki indexe gidecek sekilde yine verilerin degismesi durumunda indexlerin guncellenmesi degismesi gerekir.

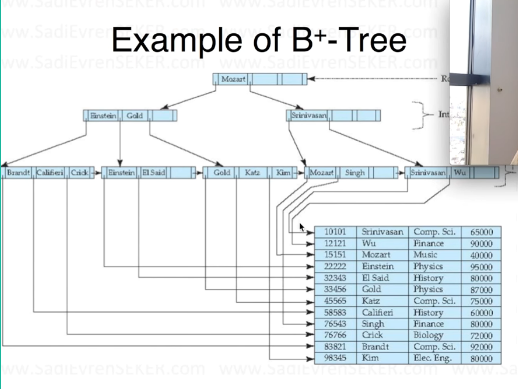
B+- Tree

Bir dugumun boyutunu iceren bilgi (node size) vardir. Atiyorum bir dugumde 3 tane indis durabilir seklinde bu 3 o bilgidir iste bu sayede her dugumde 3 tane indis saklanabilir 4 tane yazilamaz.

Root node – Internal nodes – Leaf nodes

Leaf node larin yapisi bagli listedir. (Node lar birbirine pointerlarla baglanmistir yani)

Ve bu leaf nodelardan sonra da zaten veriye erisilir. Veri katmanina gelinir yani. Mesela binary tree yi dusunecek olursak veriler dugumlerde saklanir ama burda boyle degildir leaf nodelardan yani agacin en son seviyesinden verilere erisilir. (Agacin son dugumleri)



Ayni binary tree gibi calisir mesela iste yeni bir veri insert edecegiz iste atiyorum sadi ismini ekleyecegiz S harfi M den buyuk oldugu icin agacin sagina gidecegiz daha sonra leaf node a ulasip Singh’in yanina Sadiyi ekleyecegiz bos bir dugum var cunku orda ve rahat ama mesela leylayi ekleyecek olursak bu durumda once sola sonra saga gidip leaf node a erisecegiz ama bos yer olmadigi icin ekleyemeyecez iste bu durumda hemen yeni 2 dugum acilacak ve bos yer olmayan leaf node’dan bu yeni acilan dugumlere bir anahtar olusturulacak bu anahtarin olusmasi zaten ilerleyen zamanda anlatilacaktir.

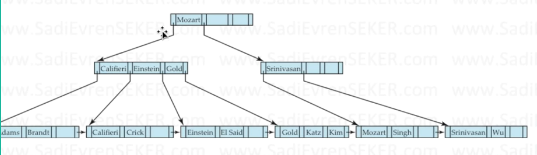
B+- Tree Ozellikleri

Mesela her leaf node’a giden yol esit uzakliktadir.

Her dugumun n/2 ile n tane arasinda degisen cocugu vardir sayet internal node ise o dugum tabiki yani root ve leaf degilse. (yani agacin dengeli oldugunu soyluyor aslinda)

Arama islemleri basittir ama bizim amacimiz hizli ulasmaktir veriye yani hizli arama yapmaktir.

Handling Duplicates – Veri eklemeden once verinin eklenecegi yere onceden ulasilip o verinin daha once eklenip eklenmediginin kontrol edilmesi islemidir.



Ekleme algoritmasi karmasikligi – Mesela soyle dusunelim Mozarttan iki tane pointer cikmis bu sayede otomatik olarak bir yolu sectigimizde verinin %50 sini egale etmis oluyoruz. Daha sonra alta inince burada 4 tane pointer oldugunu goruyoruz bu durumda da elimizde kalan verinin %75’ini egale edip sadece %25’ine odaklandigimizi gosterir yani aslinda algoritma karmasikligi node size’a baglidir diyebilir burada node size’i pointer larin sayisi olarak dusunulebilir. Ve bu agacimizda olan verilerde max 5 node size olabilir ve minimum da 2 pointer oldugu icin root node dan 2 olabilir bu durumda log(n) algoritma karmasikligi uzerinden istatiksel bir deger alinarak karmasikligimizin log2 ile log5 arasinda oldugunu soyleyebiliriz. (log2(n) ve log5(n) yani) Ornek verecek olursak mesela elimizde 1 milyon tane search key values olsun ve n = 100 olsun. Bu durumda 100’un ortalamasini aliyoruz mesela 2 ye boluyoruz daha sonra log50(1.000.000) diyoruz ve algoritma karmasikligimiz bu oluyor.

Index Definition in SQL

Create index <index-name> on <relation-name> (<attribute-list>)

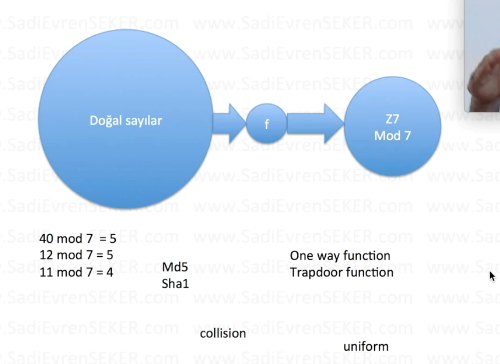
**Create index b-index on branch (branch\_name)**

Eger unique index kullanirsak yani **create unique index** dersek bu durumda her veriden bir tane olmasi sarti araniyor. Ve bu sayede tabi unique ozelligini sagladigi icinde ayni zamanda bir candidate key de oluyor.

Drop index <index-name> (indexi silmek icin kullanilir)

**9.2 VIDEO (Hashing)**

What is Hashing?



Daha buyuk bir dunyayi daha kucuk bir dunyaya indirgeme islemidir aslinda. Yani mesela elimizde sonsuz sayida dogal sayilar kumesi varken bunlarin hepsinin mod 7 si alinarak sadece 7 sayiya indirgenmis olur.

Bunlara ayni zamanda one way function da denir veya trapdoor function. (Trapdoor kavrami ikinci dunya savasindan kalan bir terimdir – Bir kapiyi acinca bir daha acamama anlamini tasir cunku arkasinda bomba vardir duzenekle ve kapi acildiginda patlar) Yani soyleki bu dogal sayilari mesela mod 7 ile 7 sayiya indirdikten sonra atiyorum bu 7 rakamdan 5 rakamini ele aldigimizda hangi sayinin 7 ile modu alinip da bu 5 sayisinin elde edildigini bilemeyiz yani bir kere indirgeme yapildiktan sonra artik geri donusu yoktur tek yonlu fonksiyon ve trapdoor fonksiyon denmesi bu sebeptendir.

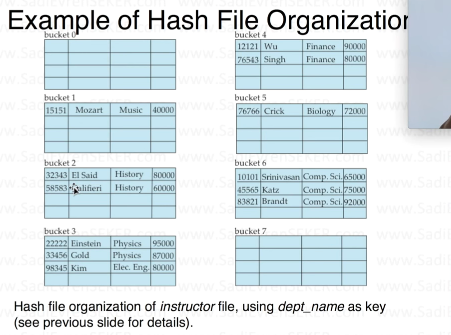
Mesela hashing islemi genelde Authentication (kimlik dogrulama - yetkilendirme) islemlerinde kullanilir. Yani atiyorum bir sistemin kullanicisinin parolasini acik bir sekilde veritabaninda saklarsak sayet o veritabanina erisim izni olan okuma izni olan biri ordan sifreyi alabilir yetkili olmadigi halde yani yetkilendirilmemistir ama okuma izni vardir atiyorum bu durumda sifreyi alabilir ordan cunku okuma izni var. Dolayisiyla ne yapiyoruz bu veriyi hashleyip veritabanimiza koyuyoruz yani atiyorum diyoruz ki 123’un mod 7 karsiligi iste sudur deyip yaziyorsunuz veritabanina sonra diyorsunuz ki sen sifreni soyle bakalim sonra iste adam yaziyor sifresini ve siz de sifresini alip mod 7 yapiyorsunuz ve esit mi degil mi bakiyorsunuz. Esitse diyoruz ki tamam sen sifreni dogru yazdin. Esit degilse de yanlis girdin diyoruz tabi mod 7 cok iyi bir hash fonksiyonu degil. (Hash fonksiyonu ne kadar buyuk bir uzay dondurebiliyorsa o kadar iyi bir fonksiyondur diyoruz) Yani soyle dusunursek mod 7 ile yapilan sifreleme isleminde atiyorum adam baska bir sayi girdi yani sifreyi aslinda yanlis girdi ama mod 7’si gercek sifrenin mod 7’si ile tuttugu icin sifreyi dogru girdi olarak kabul edecegiz bu ise cok sikintili bir durum yani adam yanlis bile girse en fazla 7 deneme sonunda dogru sifreyi bulacak cok basit yani iste o yuzden ne kadar genis bir uzayimiz olursa o kadar iyi. ( Md5, Sha1 gibi hash fonksiyonlarimiz var mesela sifreleme icin ;) ) Sifreleme de bu sekilde ise yariyor ama veritabaninda veriye daha hizli erismeyi sagliyor hashing ;)

Mesela atiyorum bir sistemimiz var ve yuz tanimayla iste kapi aciyoruz simdi biz gidip veritabanimizdaki tum resimlere bakip karsilastirma yaparak yuzunu gosteren kisi var mi yok mu diye arama yapamayiz bu cok uzun surer. Ne yapalim o zaman adamin resminin veritabaninda hash’ini tutalim daha sonra o adam tekrar geldiginde adamin resminin hash’i ile veritabanimizdaki hash’ini karsilastiralim ve boylece buyuk verilerde hizli bir erisim saglamis oluruz.

Static Hashing

Verilerin her birinin kendi hash’i olmasi durumudur. Yani veriyi veriyorsunuz f fonksiyonuna ve boyle her verinin bir hash’i olusturuluyor ve hep o sonucu donderiyor ve veritabanimizda biz hep bunu kullaniyoruz.

Bazen mesela hashing direkt veriye ulastirmaz atiyorum derki senin aradigin veri su 10 verinin (bucket) arasinda bir yerdedir. Veriye ulasma hizini dusurur. Bucket – Buket gibi dusunulebilir.



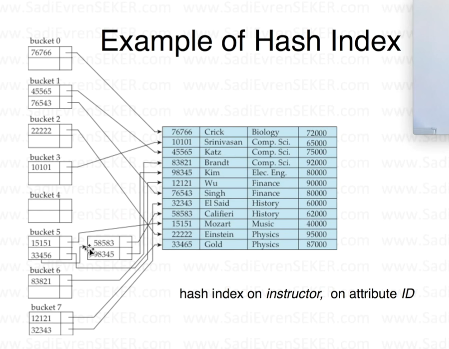
Mesela burdaki hashing de iste verilerin ikilik tabandaki karsiliklarinin sayilarinin toplaminin 10 ile modu alinip bulunmus bir bucketler tablosudur. Mesela bucket 3 de bir collision (carpisma) vardir. Fizik ve elektrik muhendisligi ayni bucket’e dusmustur diger bolumler ise farkli bucketlere yayilmistir.

Atiyorum bucket doldu bu durumda yeni veri eklenemiyecek mi o bucket’e evet static hashing de bu bir sorundur. (overflow) Mesela bir bucket cok veri geldi ve bucket tasti static hashingde bu durumda o bucket’in devami olan bucket’ler ekliyoruz tasan bucketimizin yanina ve tasan verileri buraya geciriyoruz. Bir pointer ile obur yedek bucket isaret ediliyor. Bunlar arttikca verilerin erisilme hizi dusuyor normal olarak. (closed hashing – open hashing)

Uniform dagilma – verilerin esit dagilmasi

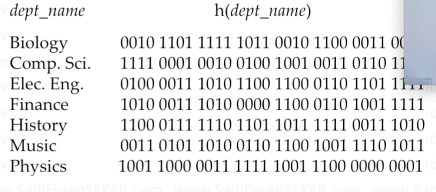
Hash Indices

Olusturulan hash’ler indexlenir bu sayede hash’lere erisim hizimir artirilir.



Dynamic Hashing (Extendable Hashing)

Veriler arttikca collision olma ihtimali artacak e veriler az oldukca da bosu bosuna bir suru yer kaplayan bucket tutmus olacaksin sen en iyis verin kadar tut alan verin geldikce biz genisletiriz :D (dynamic allocation gibi)



Burda iste degisik bir hashleme ile degisik bir algoritma ile kelimeler 32 bitlik sayilara donusturulmus ve buna gore hashlenecekler simdi. Once ilk karakterlerine, ilk digitlerine bak 0 ile baslayanlari 0 bucketine 1 ile baslayanlari da 1 bucketine koy mesela. Mesela diyelim bir bucket de 2 tane veri alabilir en fazla bu durumda veriler yerlestirilmeye baslandigi sirada bucketlere History’e gelindiginde 1 bucket’i dolmus olacak ve bir cakisma olacak hah iste bu durumda o zaman hashing diyor ki git o zaman ilk iki haneye, digite bak. Simdi mesele iki tane buketimiz varken simdi 11, 10, 01, 00 ile baslayanlar olarak 4 tane bucketimiz olmus oldu yani duruma gore dynamic olarak bucketlerimizi genislettik. Bir daha olursa bir daha genislet ayni seyi yap. Ha mesela veri silinirse de ayni islemleri geri dogru yap yani bucketleri daralt iste en son mesela elinde sadece carpisma olmadan ayirt edebilecegin veriler kaldiginda atiyorum iste 2 tane 1 ile baslayan 2 tane de 0 ile baslayan verin kaldi bu durumda iste iki bucket olacak sekilde azalt bucketleri ve verileri oyle sakla ;) Bosu bosuna bucketler yer kaplamasin.

Bitmap Indices

Tree indexler var, hashlerin kullanildigi indexler var bir de bitmap indexler var.

Bitlerden olusturulan harita gibi dusunulebilir. Veriyi aliyorsunuz ve gruplara boluyorsunuz ve sonra bu gruplara ulasmaya yarayan bitmapler olusturuyorsunuz. Sagladigi en buyuk avantaj 2 lik tabanda islemlerin yapilabilmesidir (mesela xor, not, or, and vs…)

