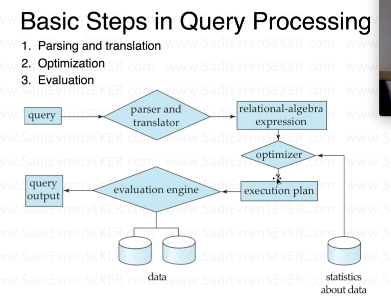
**10.1 VIDEO (Sorgularin Calistirilmasi – Query Processing)**



- query

- parser and translator – yazilan sorgu parcalara bolunup karsilik geldigi anlama cevriliyor. Hatalar burada farkediliyor.

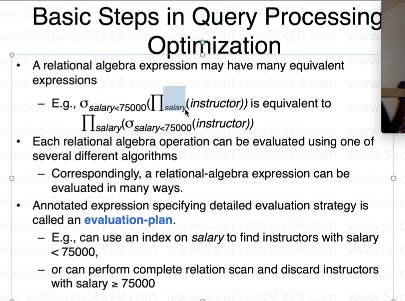
- relational-algebra expression – algoritmayla relational algebraya ceviriliyor sorgu.

- optimizer – ve daha sonra optimizasyon yapiliyor yani yazdigimiz sorgunun nasil en iyi sekilde en hizli sekilde bize gosterilecegine bu yapi karar verir. Ve bu sekilde bir execution plan olusturulur.

- execution plan – su sirayla su islemleri yapacaksin planlar dizisidir.

- evaluation engine – daha sonra evaluation plana gore bunlar bu mekanizma sayesinde calistirilir.

- query output



Query optimization – hangi yaklasimin, islemlerin hangi sirayla gerceklesirse daha iyi olacagina karar verebilecegimiz bir mekanizma olmalidir yani bunu olcebilmemiz lazim ki karar verelim. Burdaki en buyuk sey sorgu maliyedir olcmeyi yani neyin iyi oldugunu neyin kotu oldugunu sorgu maliyetine gore olceriz.

- Zaman maliyeti (disk access, CPU, or even network communication)

- Number of seeks (kac kaydin aranacagi) \* Average-seek-cost

- Number of blocks read (kac blockun okunacagi) \* Average-block-read-cost

- Number of blocks written \* Average-block-write-cost

tT – time to transfer one block

tS – time for one seek

b \* tT + S \* tS (b = blockun kendi maliyeti, S = seekin kendi maliyeti)

Normalde CPU maliyetleri de vardir ama biz bunlari bu videoda almayacagiz.

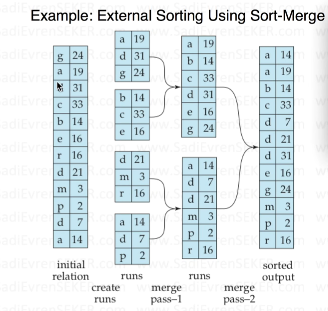
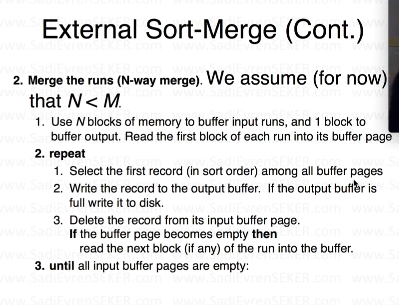
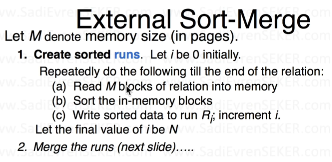
Mesela bazi alanlar sifreli tutuluyor olabilir veya resim, video gibi ozel verilerimiz oldugunda ozel durumlarimiz oldugunda CPU cok fazla derecede mesgul olur ve bu durumda CPU maliyeti herseyin onune gecer.

Sorting

Siralama isleminde tum veriye erismemiz lazimdir yani 1000 verimiz varsa siralama yapabilmemiz icin 1000 veriye de erismemiz lazim.

Ayni anda veritabanimizi direkt siralayamayiz o yuzden ‘Divide and Conquer’ methodu uygulanir.

Externeal sort-merge (merge sort)



N tane eleman siralamak icin su kadar islem yapilir algoritma karmasikligi.

20 blok boyutu, 500 kayit var

500 / 20 = 25 tane block varmis.

Log\_2(25) =~ 5 Demekki 5 birlestirme islemi ile bitecek bu islem ( n.log(n) )

- RAM’i sinirli kullandigi icin iyi bir algoritmadir.

- Amacimiz hafizayi doldurmadan siralama islemini gerceklestirmektir.

Join Operation

En basit join operasyonu kartezyen carpimidir hatirla ;)

- Nested-loop join

- Block nested-loop join

- Indexed nested-loop join

- Merge-join

- Hash-join

Example:

Number of records of student: 5,000 takes: 10,000

Number of blocks of student: 100 takes: 400

Bu bilgiler kullanilarak algoritmalarin ne kadar hizli islem yapildigini gorecegiz simdi;

- Nested-loop join

R den alindan bir tuple tutuluyor ve S deki butun tuple’larin yanina yaziliyor. R den 2. Bir tuple aliniyor yine ayni sekilde S’nin yanina yaziliyor ve islem bu sekilde devam ediyor. Her eleman her elemanla bu sayede eslesmis oluyor.

Outer join (iki taraftan birinde veri yoksa oraya null degeri ataniyor) veya inner join (iki tarafra illa ayni veri tipinden eleman bulunmak zorunda) de kullanilabilir. R den bir tuple alinacak karsilastirilacak karsi taraftan onunla bir veri iliskisi kurulabiliyorsa yani bir join kurulabiliyorsa kuruluyor yoksa o veri atlanip bir sonrakine geciliyor. Eger R den alinan tuple S’nin sonuna gidip hic bir elemanla join kuramazsa atilir ;)

Performans ve hafiza acisindan problemli bir durum.

Block Nested-loop join

Burda da blocklar aliniyor ve alinan blocklardaki tuple’ler karsilastiriliyor. (Her bir blockta 500 tane bilgi var 5,000 / 100 = 500)

Indexed Nested-Loop Join

Iki taraftada da index oldugunu kabul ediyoruz. Indexler baz alinarak 1. Indexler join olusturulacak daha sonra 2. Daha sonra 3 boyle boyle gidecek.

Digerlerinden bir asama daha iyidir diyebiliriz.

Hash-Join

Elimizdeki bir hash yapisindan islemler yapiliyor ve iliski kuruluyor.

Simdi biz R’den bir tuple alip karsilastirmaya basladigimizda S’dekilerle iste varsa eslesecek yoksa atilacak tarzi islemler yapilacak iste bu baglamda bizim verinin karsiliginin olup olmadigini bilmemiz cok onemlidir. Ve burda devreye veriye erisim hizi giriyor yani tum verileri bir bir karsilastirmak tabiki maliyetli ve zaman alici olacaktir. Bu acidan mesela hash baya avantajlidir.

Pipelining

Islemi parcalara boluyor. Mesela 4 adimdan olusan bir isleminiz var diyelim birinci islemi ilk block icin yapiyorken ikinci isleme gectiginde o 1. Islemi 2. Block icin yapabilirsiniz. Boyle boyle gidiyor iste…

- Demand driven or lazy (cekme yapisi vardir - ceker)

- Proceducer-driven or eager (arkadan itilerek ilerleyen - iter)

**VIDEO (Veri Tabaninda Zamansallik)**

Kredi karti odemesi vardir mesela her ay iste su kadar cek falan tarzi veya iste rezervasyon islemleri vs…

Zaman aslinda tamamen bizim kabulumuzdur. Birseyler degisiyor ve bunun bir duzleme oturtulmasina zaman denmis. Olaylarin zamani vardir varliklarin zamani yoktur. Zamanla ilgili hersey aslinda bir olaya referanstir yani su olaydan once, sirasinda, sonra vs…

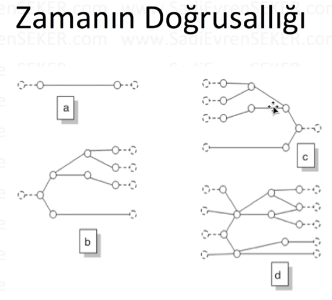
Reichenbach Zamansal Mantigi – 2. Dunya savasinda cok sayida yahudinin Turkiye’ye geldigi donemde o da gelmis ve Istanbul Universitesinde ogretim gorevlisi olmustur.



Birisi gitti marketten bir ekmek aldi program olarak dusunursek bir olayin gerceklestigi zaman var, bir bu olayin veritabanina kaydedildigi zaman var ve bir de bu olayin raporunun olusturulma zamani var.

Once veritabanimiza yazip sonra olay da gerceklesebilir mesela iste su tarihte kartimdan su kadar para cekilecek tarzi vs…

Zamanin Dogrusalligi



Allen Zamansal Mantigi

- Before(x,y) or After(y,x)

- Overlaps(x,y) or Overlapped(y,x)

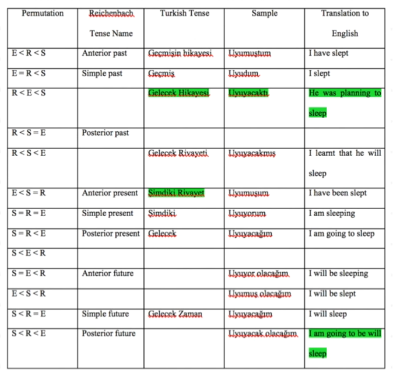
- Meets(x,y) or Met by(y,x)

- Contains(x,y) or During(x,y)

- Starts(x,y) or Started by(y,x)

- Ends(x,y) or Ended by(y,x)

- Equals(x,y)



Uygulamalardaki Durumlar

- Kesik olaylar ve olay zamanlari (olay belli bir sure icinde oluyor ve bitiyor, bir randevu alinmasi mesela)

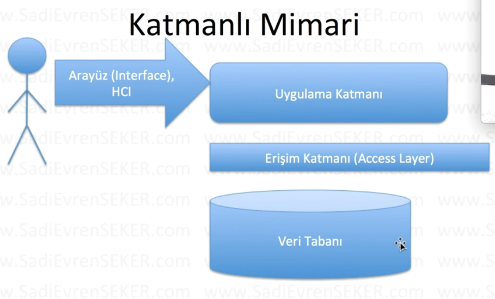
- Surekli olaylar (Randevunun suresi mesela 1 saat surer 30 dk surer vs..)

\* Baslangici ve bitisi zamana bagli olan olaylar

\* Baslangici ve bitisi olaya bagli olan olaylar

\* Baslangici veya bitisi bilinmeyen olaylar (genelde veritabaninda bunlarla ilgilenmeyiz)

Katmanli Mimari



Uygulama katmaninda bizim yazdigimiz program calisiyor artik hangi dille yazmissak (java, c,c c# vs…) daha sonra bu kodlar gidip erisim katmanindaki fonksiyonlari calistiriyor ve bu sayede veritabanina erisebiliyoruz. Bunun amaci sudur atiyorum ilerde bir gun veritabaninda bir degisiklik yapilirsa iste tum programimizi dolanip kodlari degistirmektense sadece erisim katmanindaki fonksiyonlari degistirelim ve teknolojiden bagimsiz olsun. Ve ayni zamanda atiyorum uygulama katmanimiz degisecek olsun yani programimizi degistirmek isteyelim bu durumda da veritabanimiza hic dokunmadan yine erisim katmani fonksiyonlarini degistirelim. Kullanicimiz da arayuzden bu yapiya baglanir.

Biz tarihi veritabaninda tuttuk adam bu tarihi gorecegi zaman veya bu tarihle ilgili bir hesaplama yapilacagi zaman atiyorum 01.01.2016 yerine biz bu Cuma gunuydu diye bir gosterim yapmak istiyorsak goruntusel bir degisiklik yapmak istiyorsak arayuzde bu degisikligi yapmak en mantiklisidir.

Veya iste ayda bir olacak haftada bir olacak tarzi seyler yapmak istiyorsak da uygulama katmanina gecmekte fayda var.

Veri Tabani Cozumu

\* Kesik olaylar ve olay zamanlari 🡪 Tek bir kolon

\* Surekli olaylar

- Baslangici ve bitisi zamana bagli olan olaylar 🡪 iki ilave kolon

- Baslangici ve bitisi olaya bagli olan olaylar 🡪 bir veya iki ilave kolon

- Baslangici veya bitisi bilinmeyen olaylar 🡪 uygulama seviyesi – veritabaninda yapilacak bisey yok.

Uygulama Katmani

\* Hesaplamalar

- Haftada bir gun

- Her ayin ilk carsambasi

- 1. Ve 3. Haftalar …

\* Iliski cozumlemeleri veritabaninda yapilabilir. (once / sonra iliskisi)

Uygulama katmaninda iliskileri basitlestirip veritabaninda cozuyoruz gibi birsey.

**VIDEO (Veri Tabani ve Uygulama Katmani Karmasikligi)**

Soru su data layer da mi yapalim yoksa daha ust katmanlara mi birakalim guvenlik islemlerini.

Yapilan is data ile ilgili ise data layer da yapmak daha iyidir tabiki.

Eger gorunumle ilgili birseyse mesela presentation layer a birakilabilir.

Verinin formatlanmasiysa verinin tiplerine ayrilmasiysa belki servers layer a birakilabilir.

Verilerin siralamasi atiyorum uygulama katmaninda da yapilabilir, veri tabaninda da yapilabilir hangisi daha iyi?

Mesela iste calisanlarimiz var ve siz 30 yas ustundeki calisanlari goruntulemek istiyorsunuz bunun 2 cozumu var.

1 – Veri tabanindan tum verileri uygulama katmanina cekip uygulama katmaninda 30 yas ustunu filtreleyip gosterebilirsiniz.

2 – Direkt veri tabaninda 30 yas altini hic almayip uygulama katmanina sadece 30 yas ustu calisanlari gonderip gosterebilirsiniz.

Burdaki cozumlerde gozle gorulur bir sekilde 2. Cozum daha iyidir cunku veritabani veriye daha yakindir ve veriyi optimize ettigi icin de daha hizli ve daha iyi calisacaktir.

Ama mesela uzerinde bir hesaplama yapacaksiniz atiyorum calisan maaslarina %10 zam yapacaksaniz bu durumda application layer da islem yapmak daha mantikli olabilir.

Veya veritabaninda cok kullanilan tablolar veriler vs… uygulama katmanina tasinabilir veya kucuk bir veritabaninda tutulabilir veya RAM de tutulabilir bu sekilde islem yapmak da mantikli olabilir.

Bu sorularin genel bir cevabi yoktur aslinda.

SAP – Herseyi veritabaninda tutar ve application layer’i free tutar veritabani cok komplekstir burda.

Bazi yerlerde ise uygulama katmani cok karmasiktir veritabani cok free’dir.

Bu kararlar genel olarak stratejik kararlardir stratejiye gore degisir.