**1. VIDEO (Temel Bilgiler)**

Veri yapilari ile veritabani birbiri ile iliskilidir ;)

Veritananin asil amaci veriyi verimli tutmaktir.

Big data ile birlikte veritabanin asil amaci veriye hizlica ulasmak oldu.

VERI TABANI PROBLEMLERI

Redundancy = Gereksiz veri tutmamak

Inconsistency = Butun verilerimiz ayni ozellige sahip olacak

- Veritabanlari aslinda su an isletim sistemi gibi calisiyor. Kendi ram kontrolu, disk kontrolu tarzi isletim sistemi ozelliklerini barindirir.

Integrity = Kosullarin baglanmasi (Tek bir kez kosul giriyorsun tum veriler ona uymak zorunda)

Atomicity = Para transferi yapilirken bu atomik olmak zorunda mesela. (A hesabinda 100 lira artacak ve B hesabinda para ondan sonra azalacak yoksa para havada ucar gider)

Concurrency = Birden fazla is ayni anda yuruyorsa bu islerin senkronize olmasi gerekiyor. Hangisinin once hangisinin sonra olacaginin bilinmesi lazim.

Security

Abstraction (Soyutlama)

Katmanlar:

1) Physical = Disk (Veriler diskte nasil tutulur tarzi karisik isler halledilir)

2) Logical = Relationships (Insanin anlayacagi sekilde) (Mesela bir sayiyi integer diye belirttigimiz asama)

3) Viev = Hides datatypes, grants (Belirttigimiz sayinin integer oldugunun bilinmesine gerek duyulmayan asama yani musteri bakiyor mesela o integer mi degil mi o kisi icin cok onemli degil bu mesele)

Instance & Schema

- Physical (Iste alt tarafta veriler diskte nasil tutuluyor nasil iliski kuruyorlar tarzi karisik bilgiler semasi)

- Logical (Veri tabanindaki tablolarin nasil birbiriyle iliskilendirildigi semasi mesela)

Instance = Veritabaninin anlik goruntusu. (Semalarin gercek verilerle doldurulmus hali)

Physical Data Independence (Ust taraftaki veriyi sabit tutup alt taraftaki sistemi degistirebilir miyiz? Yani mesela T-SQL kullanirken ustundeki verileri sistemi hic bozmadan alttaki teknolojiyi PL-SQL ile degistirebilir miyiz? Logical katmani etkilemeden physical katmani degistirmek - ODBMS)

DATABASE LANGUAGES

- DML (Data Manipulation Language) = Sorgular, SQL. Veri uzerinde degisiklik yapabilmek icin kullanilir.

- DDL (Data Definition Language) = Create Table (tablo olusturmak icin kullanilan dil), Data Dictionary (Veri Sozlukleri)

\*\*\* Ayni data dictionary de konusmuyoruz diye sohbet sirasinda bir arkadasin sana soyleyebilir. :D

Data Dictionary = Veri tabanindaki tum islemlerin tanimlandigi sozluklerdir. (Primary key vs…) (Bu bir teknolojidir !)

Silindir varsa veri veya veritabani vardir orda :P :D :D

Butun bilgisayar bilimleri matematikten gelir.

Sorgu Islemesi

Query 🡪 parser and translator 🡪 relational-algebra expression 🡪 optimizer 🡪 execution plan 🡪 evaluation engine 🡪 query output

Bir veritabanina birden fazla kullanici ayni zamanda farkli kanallardan erisiyor olabilir, bu mumkun.

Veritabaninin dertlerinden birisi de bu kullanicilari dogru sekilde yonetmektir.

Database Anatomisi

Alt tarafta neler oluyor? (Biraz karisikti cok yazmadim)

Istatistikler bilgiler tutuluyor. (Kim eristi, neye eristi vs..)

Database Mimarileri

Centralized = Kendi bilgisayarimiza kuracaz, kendi bilgisayarimizdan sorgulayacaz.

Client-server (Two – tier architecture) = Veritabani sunucusu (serveri) uzakta bir yerde biz ona erisiyoruz bir client (web sunucusu, console vs..) uzerinden. (user 🡪 application 🡪 network 🡪 database system) (masaustu uygulamalar, mobil vs..) (Bir kullanici bir de sunucu var)

Three-tier architecture = Bir kullanici, application server (web sunucusu vs..) ve bu sunucunun baglandigi veritabani var. Araci bir sunucu var. Veritabani daha guvenli oluyor.

Parallel (multi-processor) = Cok islemcili bir sunucu uzerinde veritabani calistiriliyor, bu sunucuda islemlerin bu islemcilere dagilmasi isteniyor. (Bugun cogu isletme bunu kullaniyor) (Veritabani saticilari ucretlemeyi islemci sayisi uzerinden yapiyorlar) (Daha onceki videolardaki gibi ayni sistem var ogrenmek icin al kullan ucretsiz ama ticaret icin kullanmaya baslarsan islemci sayisi uzerinden hemen ucretlendirilir)

Distributed = Veritabanin parcalara bolunmesi, ve bu parcalarin farkli yerlerde hatta farkli cografyalarda durmasi islemidir.

**2. VIDEO (Relational Algebra – Iliskisel Cebir)**

SQL dilinin altinda yatan temel seyler bu videoda yer alacak.

Iliski = Birden fazla tablo uzerinde tanimli olan herhangi birseydir. (Tablolar arasi iliski)

Sizin tablolariniz var onlarin uzerinde bir iliski tanimlayip verileri harmanliyorsunuz ve onlarin uzerinden bir veri elde ediyorsunuz.

Cebir = Yani her veriyi alma da iste su verileri al veya su kolonu al sadece tarzi seyler.

(Kolon – Attribute, Satir – Tuple)

Farkli farkli tablolardan cesitli attributes lerin alinmasi islemi diyebiliriz kisaca.

Anahtar = Bir tablodaki herhangi bir degeri, satiri ifade eden ozellik.

SuperKey = Bir tabloyu anlamak icin yani mesela bir kayit arattiginizda o isimle birden fazla kisi olabilir ama ayni kimlik numarasiyla birden fazla kisi olamaz iste atiyoum isim ve kimlik numarasi birlikte kullanildiginda bu tabloyu tanimlama yeter bir SuperKey’dir. (Ozellik kombinasyonlari)

CandidateKey = Oyle SuperKeyler alicaz ki bunun bir alt parcasi SuperKey olmayacak. Mesela ad, adres ve kimlik numarasi superkey olabilir ama candidatekey olamaz cunku bir alt parcasi mesela ad ve kimlik numarasi da bir superkey. Yani alt parcasi superkey olmayan superkeylere CandidateKey diyoruz.

PrimaryKey = PrimaryKey’i biz tanimliyoruz. SuperKey, CandidateKey tarzi seyleri veritabanina baktiginizda zaten siz goruyorsunuz ve bunlardan birini secip primarykey atiyoruz. (primarykey olmasi icin candidatekey olmak zorundadir)

Iliskisel Cebir

Select, project, union, set difference, Cartesian product, rename

Set intersection (kesisim) , natural join, division, assignment

Bunlar SQL syntaxinin altta ne is yaptiginin, bunlarin nasil oldugunun matematiksel kolay bir aciklamasi.

(Inner join, outer join, iste komutlar select vs.. tarzi seyler anlatiyor yazmadim bakarsin ;)

DB’nin degistirilmesi

Silme (Deletion), Ekleme (Insertion), Degistirme (Update)

**3. VIDEO (SQL - 1)**

Veri Tipleri = char, varchar, int, smallint, numeric, real/double precision, float vs…

Iliskisel veritabanlari tablolar uzerine kuruludur.

Create table <r> (A1 D1, A2 D2, … An Dn

<Integrity-contraint1>…<integrity-contraint>) 🡪 Not null veya PrimaryKey tarzi seyler.

**Create table branch**

**(branch\_name char(15) not null,**

**branch\_city char(30),**

**assets integer)**

Integrity Contraint

- not null

- primary key (birden fazla ayni sey olamaz) (unique olacak)

**Create table branch**

**(branch\_name char(15) not null,**

**branch\_city char(30),**

**assets integer,**

**primary key(branch\_name) )**

Drop table = Databaseden siler tabloyu.

Alter table ile de tabloyu degistirebiliriz.

- alter table <r> add A D

- alter table <r> drop A

SORGULAR

Select A1, A2, …, An from r1, r2, …, rm where P (P = kosul)

Distinct ve All

**Select distinct branc\_name from loan** (Distinch = Tekrarlari at sonucu ozet seklinde goster demektir)

**Select all branc\_name from loan** (All = Tekrarli veri varsa da gozukuyor distinch gibi atmiyor)

\* sembolu = **select \* from loan** (tum verileri getirir)

\* carpi anlaminda da kullanilabilir elbetteki burdaki anlami hepsini sec falan degildir o farkli bir anlamdir.

Kosullar

**Select loan\_number from loan where branch\_name = ‘Perryridge’ and amount > 1200**

And, or ve not gibi baglaclar kullanilabilir.

**Select loan\_number from loan where amoun between 90000 and 100000**

JOIN (Iki kumenin kartezyen carpimidir aslinda)

Bir tablonun satirlariyla obur tablonun satirlarini tek tek kartezyen carpim yapip yeni bir tablo olusturuyor gibi bisey.

Select \* from borrower, loan

Yani diyelim ki birisi borc almis ve borc (loan) tablosunda da bunun detaylari var. Kisinin bilgileri ise borrower tablomuzda yer aliyor olsun. Yani mesela dersem ki bu borc alanin hesap numarasiyla sadece borrower tablosundaki hesap numarasi esit olan kisileri getir diye bir sart yazarsam o adamin aldigi borcun detaylari bana gelecek. Obur turlu bir adam bir borc, bir adam bir borc bu adamla bu borcun alakasi var mi yok tarzi sacma seyler olur o yuzden kosul bu etapta cok onemlidir.

Relational Database de en onemli sey Joinlerdir onlar olmazsa Relation kalmaz.

**Select customer\_name, borrower.loan\_number, amount from borrower, loan where borrower.loan\_number = loan.loan\_number and branch\_name = ‘Perryidge’**

Isim Degistirme

Old-name as new-name (hani mesela iste ahmet as a diyoruz sonra ahmetinkileri a.ornek falan diye cagiriyoruz mesela) (Herseye bu sekilde isim verebiliriz. Attributeler de degisebilir, tablolar da vs…)

Bazen mesela as kullanmadan da isim degisir direkt atiyorum tablo ismi sonra A boyle de olur yani.

Satir Degiskenleri

**- Select customer\_name, T.loan\_number, S.amount from borrower as T, loan as S where T.loan\_number = S.loan\_number**

- **select distinct T.branch\_name from branch T, branch S here T.assets > S.assets and S.branch\_city = ‘Brooklyn’**

Dizgi Islemleri (String)

Biraz sorunlu islerdir. Zor islerdir.

% = hersey demektir.

**Select customer\_name from customer where customer\_street like %Main%** (Main kelimesi icinde gecen seyleri ariyor. Mainden once de herhangi bisey olabilir sonra da ‘%’ o anlama geliyor)

\ = escape character

|| = bir dizgiyle otekini birlestirmek icin kullanilir.

Siralama

**Select distinct customer\_name from borrower, loan where borrower loan\_number = loan.loan\_number and branch\_name = ‘Perryridge’ order by customer\_name**

- Example: order by customer\_name desc (ters sirada yazar) (asc – duz sirada yazar)

Siralama yaparken programda kodlarken de yapmayi dusunebilirsiniz ama emin olun SQL veriye bir adim daha yakindir ve daha hizlidir. Yani siralamalarinizi SQL de yapin ☺

Kume Operatorleri

(select customer\_name from depositor) **union** (select customer\_name from borrower)

- Iki sorgunun sonuclarini alir ve birlestirir. Tek bir tablo haline getirir.

(select customer\_name from depositor) **intersect** (select customer\_name from borrower)

- Birinci sorgudan gelenle ikinci sorgudan gelen verilerin sadece ortaklarini alir. (kesisenleri alir)

- Hem borc alan hem borc veren kim var mesela diye tablo isimleriyle iliskilendirebiliriz bu olayi.

(select customer\_name from depositor) **except** (select customer\_name from borrower)

- Yani borc verenlerden borc alanlari cikar demek gibi bisey mesela bu tablo isimlerine gore. (Borc vermis olsunlar ama hic borc almamis olsunlar mesela bu sorgunun sonucu)

Aggregate (birlesme) fonksiyonlari

**Avg** – average value (**select avg(balance) from account where branch\_name = ‘Perryridge’**)

**Min** – minimum value

**Max** – maximum value

**Sum** – sum of values

**Count** – number of values (**select count(\*) from customer**)

(**select count (distinct customer\_name) from depositor**)

Group By

Mesela musteri tablomuz var ve bu tabloda her musterinin hangi subede oldugu bilgisi var. Dolayisiyla biz subelere gore GroupBy yaptigimizda o musteriler birlestirilerek birlikte gosterilir. Herbir sube icin bir tane satirimiz olacak cunku) Ama o subede 100’lerce musteri var mesela. Iste o zaman aggragate yaparak mesela average alabiliriz. (Sube ve karsisinda o subenin musterilerinin hesaplarinin bir ortalamasi olacak bu sayede 1.sube, 2. Sube vs hep boyle listelenecek)

**Select branch\_name, count(distinct customer\_name) from depositor, account where depositor.account\_number = account\_number group by branch\_name**

Having

Where gibi kullanilir ama eger group by kullanmissak kosulu where ile degil having ile belirtiriz.

**Select branch\_name, avg(balance) from account group by branch\_name having avg(balance) >1200**

Null (bos) Degerler

**Select loan\_number from loan where amount is null** (atiyorum tum bilgiler girildi ama fax numarasini girmedi adam ne yapacagiz fax numarasini girmedi diye kayit etmeyecek miyiz adami yani elbet null degerler olacak)

Unknown, null gibi biseyler vardi burada (dakika 35 civarlari)

Null Values and Aggreagates

**Select sum (amount) from loan** (sum null olanlari toplamiyor veya 0 kabul edip topluyor ayni sey zaten)

Mesela count(\*) kullandigimizda null olanlari saymiyor.

Ic ice sorgular (nested query)

Subquery = Alt sorgu gibi bisey. Genelde select-from-where tarzi seylerle yazilir. Mesela tablolardan gecici tutacak sekilde veriler cekecegiz ve iste her bir subquery’den gelen verileri en son birlestirip birsey yapacagiz mesela bu tarz seyler. Bir tane de olabilir direkt alir onun uzerinde islem yapariz.

**- Select distinct customer\_name from borrower where customer\_name in (select customer\_name from depositor)**

- **Select distinct customer\_name from borrower where customer\_name not in (select customer\_name rom depositor)**

??? Find all customers who have both an account and a loan at the Perryridge branch

- **select distinct customer\_name from borrower, loan where borrower.loan\_number = loan.loan\_number and branch\_name = ‘Perryridge’ and (branch\_name, customer\_name) in (select branch\_name, customer\_name from depositor, account where depositor.account\_number = account.account\_number)**

\*\*\* Bu tip query leri okurken icerden okumak daha faydali olacaktir!

Kume Karsilastirmalari

**Select branch\_name from branch where assets > some (select assets from branch where branch\_city = ‘Brooklyn’)** (some olunca herhangi biryerde saglanmasi yeterli oluyor all dese heryerde saglanmasi gerek)

5 < some (0, 5, 6) = true 5 < some (0, 5) = false

5 = some (0, 5) = true 5 != some (0,5) = true

- All kullanildiginda da sartlari hepsi icin arastiracagiz ;)

Ornek Sorgu (Genel)

**Select distinct T.customer\_name from depositor as T where not unique (select R.customer\_name from account, depositor as R where T.customer\_name = R.customer\_name and R.account\_number = account.account\_number and account.branch\_name = ‘Perryridge’)** (Oyle musteriler bulki Perryridge subesinde en az 2 tane hesabi olsun) (not unique = esi olan demek) (en az 2 oldugunu not unique ile aliyoruz)

With Kelimesi

**With max\_balance (value) as select max (balance) from account select account\_number from account, max\_balance where account.balance = max\_balance.value** (with ile gelen ifadeyi sorgunun devaminda kullanabiliyoruz) ( max\_balance.value = max\_balance (value) )

Views

Tablolar bir relationdur ve Viewler de bir relationdur.

Tablo veritabanin diskte, ram de gercekten yer kaplayan elemanlardir. Viewler ise tablolar uzerinden olusturulmus gecici tablolar gibi dusunulebilir.

Simdiye kadar yaptigimiz cogu sorgu view’ dir ☺☺

Viewler veritabaninda kalici olarak kalmazlar.

Create view <v> as <query expression>

Iste surdaki su kayitlari getir tarzi bir view yaziyorsunuz ve bu viewi artik is arkadaslariniz da kullanabiliyor ve rahatca istenilen verilere erisebiliyor.

**Create view all\_customer as (select branch\_name, customer\_name from depositor, account where depositor.account\_number = account.account.number) union (select branch\_name, customer\_name from borrower, loan where borrower.loan\_number = loan.loan\_number)**

- Depend directly = Bir viewin otekine dogrudan bagli olmasi (siz bir viewi olustururken baska bir bir viewden veri aliyorsaniz bu depend directly oluyor)

- Depend on = Bir viewin otekine dogrudan bagli olmasi (siz bir viewi olustururken baska bir bir viewden veri aliyorsaniz ve olusturulan bu ikinci viewi de kullanip 3. Bir view olusturuyorsunuz)

- Recursive = Bir viewin kendi ustundeki bazi verilerden olusturulmasina denir.

Veri Tabaninda Degisiklikler

Silme

**Delete from account where branch\_name in (select branch\_name from branch where branch\_city = ‘Needham’)** (Delete all accounts at every branch located in the city ‘Needham’)

Ekleme

**- Insert into account values (‘A-9732’, ‘Perryridge’, 1200)** (values sirali olmak zorundadir eger sirali vermeyecekseniz account tablo isminden sonra hangi sirayi kullanacaginizi veya kullanacaklarinizi buraya yazabilirsiniz. Mesela bir attribute e deger atamak istemiyorum ben yani)

**- Insert into account select loan\_number, branch\_name, 200 from loan where branch\_name = ‘Perryridge’**

**- insert into depositor select customer\_name, loan\_number from loan, borrower where branch\_name = ‘Perryridge’ and loan.account\_number = borrower.account\_number**

Update

**Update account set balance = balance \* 1.06 where balance > 10000**

Viewlerin Update edilmesi

View’ler tablodaki her alanlari bulundurmak zorunda degildir yani bulundurabilir ama mesela atiyorum siz sadece iste ad ve soyad bilgileriyle bir view olusturdunuz bu durumda bu view’e update ederken tablodaki ad ve soyad harici degerler view’de olmadigi icin onlar otomatik olarak null degere sahip olacaktir bunu unutmayin.

**Create view loan\_branch as select loan\_number, branch\_name from loan**

eritabani olasi hatalara onlem olarak olusturulan yani kaydi yeni veriyi hemen veritabanina yazmiyor.r RAM'