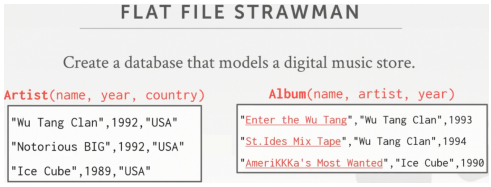
01–02

Course Introduction & Relational Model (CMU Databases Systems _ Fall 2019)

01–02

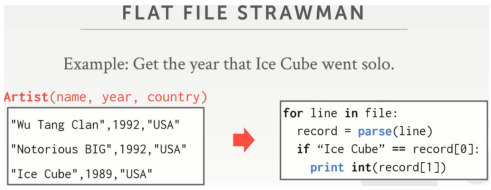
20.05–20.13 虚生花
so let's see how we actually could build now an application that could store this information
So 我们来看下该如何去构建一个可以用来存储该信息的应用
20.13– 20.24 虚生花
so let's say that you know we don't know any my database management systems ,right we don't know MySQL ,we don't know Oracle ,we don't know PostgreSQL ,we don't buy any of that ,so we in our own application we're gonna write this code ourselves
假设我们现在不知道任何数据库管理系统。我们不知道MySQL，不知道Oracle，不知道PostgreSQL。我们也没有买任何数据库系统。因此，我们会在我们的应用中自己手写这些代码
20.24–20.36 ******
so the simplest database we could implement and manage in our application would be just one where we store our data and a bunch of comma separated value files or CSV files
So我们可以在应用程序中实现和管理的最简单的数据库，即可以用它来存储数据和一堆用逗号分隔的值文件或CSV文件
20.36–20.48
And then, in our application code we're gonna write the procedures the methods to read this data and extract information we need to answer for questions or queries for them
接着，在我们的程序代码里，我们会去编写程序和方法，以此来读取数据并提取我们需要的信息来回答问题，或者对它们进行查询
20.48–21.01
Right,So the way to think what this is like for every entity we have in our application or in our database like the Artists and the Albums will store them in a separate file ,like artists CSV Albums CSV
我们程序中或者数据库中的每个实体例如Artist和Album会分为存为单独的csv文件
21.01–21.05 虚生花
And then we'll have some code that knows how to open that file up
然后，我们要有某种可以用来打开这种文件的代码
21.05–21.12
you know parts each along to extract the different attributes about, you know that these things these files are storing

然后提取用于存储的各个部分的不同属性，你也知道这些文件正是用来保存这些东西的



21.12–21.20
right ,so let’s say again we have two entities in our database
再说一遍，在我们的数据库中现在有两个实体
21.20–21.23
right of the Artists and Artists has a name and year in the country
在Artist中，Artist有三个属性，即名称，年份以及国家
21.23–21.28
and then we have their Albums we have the name of the Album the Artists put out the element in the year that put it out
然后我们还有Album这个实体，它里面保存了专辑名称，艺术家信息以及发布年份
21.28–21.38
right so again, if we’re just storing this as a CSV files, we would have quotation marks for each attribute then commas that would separate them
如果我们用CSV文件存储它们，我们会在每个属性上用引号标明，然后用逗号将它们分开

21.38–21.46
so let’s say now we want to write a query that could look at the Artists
假设现在我们想要写一条查询语句来查询Artist
21.46–21.52
you know the Artists file and try to figure out the year that Ice Cube went solo
现在，我们有保存了Artist的文件，并且试图找出Ice Cube在哪一年单飞
21.52–21.55
So Ice Cube was a founding member of NWA here in LA
Ice Cube是当时洛杉矶NWA的创始成员
21.55–22.01
and then he left them because the money disputes anything he went solo
然后，他离开了他们，因为金钱上的纠纷，所以他选择了单飞



22.01–22.11

so if we can we had the CSV file has this artist information ,you know we could write some simple Python code that would just iterate over every single line of the file right
如果我们有保存了艺术家相关信息的CSV文件，这样我们就可以通过写一些简单的Python代码来遍历该文件中的每一行
22.11–22.21
we would have a function that would parse it just basically split the line split each line up by its commas ,and get back an array of attributes
我们有一个函数来对它进行解析，将一行行分开，每行通过逗号分割得到一个数组，然后返回这个关于属性的数组

22.21–22.25
and then we just check to see whether the first attribute equals Ice Cube
然后我们只需检查数组中第一个属性是不是Ice Cube
22.25–22.30
and so then we'll convert the second attribute to an integer because that's the year we just print that
如果是的话，那我们就将第二个属性转换为整数，因为这是我们想打印的年份
22.30–22.37
Right, really simple coded answer this particular query
通过简单的代码就找到了这个查询所要找的内容

FLAT FILES: DATA INTEGRITY

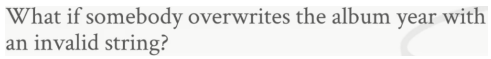
22.37–22.49
So some problems with this approach ,so and this will go through these problems of why you don't want to manage data like this in your application
这种解决方法会存在一些问题，也正是因为这些问题，我们才不想去使用这种方式在应用程序中来管理数据
22.49–22.58 ******
and then there's a motivate for why we want to build it sort of a general purpose or a database management system that can handle all these things
这也是为什么我们要构建一个通用的或者可以处理所有这些事情的数据库管理系统的动机

How do we ensure that the artist is the same for each album entry?

22.58–23.13
so the first question is say ,you know how can we ensure that in our application that for every single album that an Artist puts out the artist field in that Album file is guaranteed to be the same
所以，第一个问题是，我们该如何确保在我们的应用程序中，对于每个album来讲，要确保artist所在的album中的位置都要相同（知秋注：第一个位置是人名，第二个是年份，第三个是国家）
23.13–23.15
but how do we know that we don't have a spelling mistake for Ice Cube
但我们又如何知道我们在Ice Cube中没有拼写错误?

23.15–23.22

Right ,and then if we end up doing that how to be, you know if say Ice Cube changes his name how do we make sure we fix all those things
如果我们出现拼写错误这种事，你知道的，比如Ice Cube改了他的名字，那我们如何保证修复所有问题呢?



23.22–23.30
Right,the next issue is how do we ensure that the data we're storing is a valid for the different type
下一个问题是，我们该如何保证对于不同类型数据的存储是有效的
23.30–23.34
Right,So the Album year should be a four–digit number
例如，专辑的发布年份应该是个四位数
23.34–23.37
but what happens if someone puts in a random string in that place
但如果有人放了一个随机字符串在那，这会发生什么呢?
23.37–23.42
Right,maybe anybody can open up a file and modify it cuz it's just a regular file on disk
可能有人会打开这个文件并修改它。因为它只是硬盘上的一个普通文件
23.42–23.46
but now our application does that parsing
但现在如果我们的应用程序对它进行了解析
23.46–23.50
and it sees a random string one expects to see an integer and it's going to throw an error
然后程序在想要读取数字的地方却读到了一个随机字符串，那它就会报错
23.50–23.53
,because that's like someone modified this data in a way that I know is not expected
因为这就像有人以某种我不希望的方式修改了这个数据

23.53–24..00
and then the next issue is what if we have now an Album that has multiple artists
接着，下一个问题是如果一张专辑是由多个艺术家制作的，那我们该如何保存数据
24.00–24.02
well that's problematic
这其实是个大问题
24.02–24.08
because the way I set up my file ,there's only one Artist expecting field
因为在我的文件中只有一个Artist字段
24.08–24.12
right ,so I could try to store that within the quotation marks a bunch
我可以试着将它存在引号之内
24.12–24.16
you know a bunch of commas separated that is inside of that thing
你知道，我们可以用一些逗号将它里面的东西分开

24.16–24.26
but now I need to go look every single time I'm looking at the attribute and say this is you know is that is the Artist name an array itself or it is just you know just a string
但现在我每次都要去看属性里面放的Artist name本身是一个数组或仅仅只是一个字符串
24.26–24.33
so again ,you have the way all the specialized logic to deal with these particular problems in your application
因此，你需要有某种方式来处理你应用程序中所出现的这些特定问题

FLAT FILES: IMPLEMENTATION

24.33–24.37
so implementing this is not easy
实现这点并不容易
24.37–24.40
Right,so how do we actually find a record
So实际上我们该如何找到一条记录呢?
24.40–24.47
so I showed my simple example was a for loop to iterate and parse every single line to find the record that I was looking for
在我所展示的简单例子中，我用了一个for循环去遍历并解析每一行，以此来找到我要找的记录
24.47–24.52
and so you know my sample file had three lines, so that's not big of a deal
你知道的，我的示例文件中只有三行，所以做起来很简单

How do you find a particular record?

24.52–24.55
all right so that can be done pretty fast
所以这样处理起来很快
24.55–25.04
well you know what if I had a billion items， do I really want be opening the file every single time scanning and parsing every single one to answer every single query
Well，如果我有十亿条数据，难道我要每次打开文件进行扫描，然后解析每行记录来找到每次查询的结果么?
25.04–25.06
no right because that would be really really slow
很明显No，因为这真的很慢很慢

What if we now want to create a new application that uses the same database?

25.06–25.21
now next issues that would have say ,you know our application in that show here was written in smooth it looked like Python code，but whatever now I want to use write another application that's written in another language ,and then I want to use that same database
下一个问题是，我们的程序代码是用Python写的，但现在我想用另一种语言来写另一个程序，但我仍然想使用同一个数据库
25.21–25.28

all right ,so let's say that the example code I showed you was running on a web server open up a file parsing it producing the answer
假设，我所向你展示的示例代码运行在web服务器上，然后用它打开一个文件并解析生成答案
25.28–25.32
but now I have like a mobile phone application that wants most access to the same database
但现在我有一个手机应用程序，我希望它能访问这同一个数据库
25.32–25.37
well ,my mobile phone application might not be written in Python might be written another language
Well，我的手机应用程序可能不是用Python写的，它可能是用另一种语言写的
25.37–25.45
therefore neither duplicate all my logic to parse that file in my web app or whatever other application
因此，在我的Web app或其他应用程序中都不应该重复这个用来解析文件逻辑（知秋注：无须重复造轮子）
25.45–25.50
and of course now how do I start sharing things right that that becomes probably problematic as well
当然，我该如何去合理地共享数据，这也可能会引起一些问题

25.50–26.03
all right, so again same thing what if what if I have two threads or two programs a few processes running at the same time that want to write to the file at the same time ,what's gonna happen right
同样的，如果我有两条线程或者是两个同时运行一些进程的程序想要同时对一个文件进行写入，这会发生什么呢?

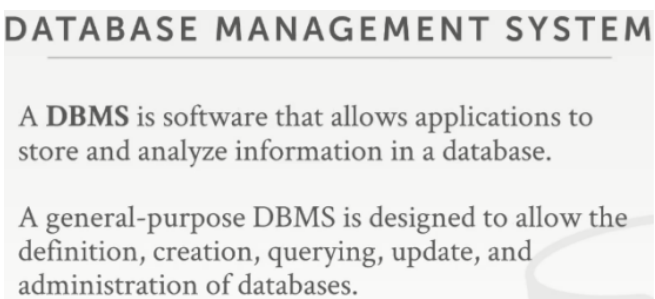
26.03–26.11
if I don't do anything special, then the first guy will write something, then the second guy might just overwrite it and I'll lose changes the first guy
如果我不做什么特殊处理，那么当第一个人对文件进行写入后，第二个人可能会将第一个人写入的数据进行覆盖，那我就丢失了第一个人对文件所做的修改
26.11–26.16
so in that now I start losing data or my data ends up being becoming invalid
因此，我就会丢失数据，数据可能会变成无效数据
26.16–26.19
because it's getting garbled so that's that's problematic
因为它会变成乱码，这就出大事了

FLAT FILES: DURABILITY

26.19–26.30
all right, the the last issue is how do I ensure that my data is safe
好了，最后一个问题是我该如何保证我的数据是安全的

26.30–26.42
so let's say that I'm updating a record I open a file, I start writing to it but then before I finish writing my update,the machine crashes my program crashes
假设我打开文件然后更新了一条记录，在我结束更新记录前，机器出故障或者程序出故障了
26.42–26.43
what happens right
这会发生什么呢?
26.43–26.46
should that update be there should it only be half updated
记录是更新完了还是只更新了一半?
26.46–26.49
I think how do I reason about what the correct state should be
我该如何去推断它的正确状态呢?

26.49–26.58
what if I again I want to say well I don't trust the machine that I'm running on so therefore I want to replicate my database my files to two different machines
即我想说的是，我对当前正在运行的这台机器不能完全信任（知秋注：因为可能会发生宕机之类的生产事故），因此我想将我的数据库文件复制到两台不同的机器上
~~假设如果我不相信我所运行的机器，那么我想将我的数据库和我的文件复制到其他几台机器上~~
26.58–27.04
so that if one machine crashes the other one can just pick up and I can run keep running without anybody noticing
这样如果一台机器崩了，那么在没有人注意到的情况下，我还能使用备用机器
27.04–27.07
I say All American use a distributed file system
所有的美国人都使用分布式文件系统
27.07–27.13
but you know those things aren't general–purpose usually and that can be difficult to do
但你知道这些东西通常并不通用，而且很难去实施



27.13–27.20
so for a variety of these problems as well as others

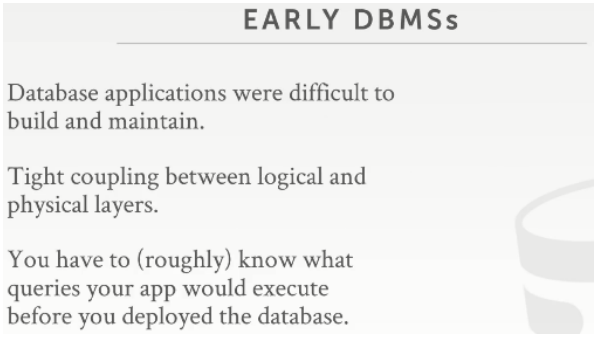
~~所以对于这些问题以及其他问题来说~~
对于以上这些问题来说
27.20–27.27
This is why you don't want to write the kind of stuff that we talked about a parsing a file and reading it in your application
这就是为什么我们不想通过之前讨论的那种解析文件的代码来在我们的应用程序中使用的原因

27.27–27.39
you want to offload this or, you want all that sort of complex logic or how to manage the data in the database ,you want a database management system to manage that for you
你想要对此进行减负，你想实现各种复杂逻辑或者如何在数据库中管理数据，你想用一个数据库管理系统来帮你对此进行管理
27.39–27.43
so a database management system is specialized software
因此，数据库管理软件是一种专业软件
27.43–27.51
that allows applications to store and analyze information in the database without having to worry about the underlying details of how to do that
它允许程序在无须关心底层实现的情况下，对数据库中的信息进行存储和分析
27.51–27.59
right ,and it's software that can be reused from one application to the next, so that you're not reinventing the wheel all over again
它是一种能够被多种应用所复用的软件，这样你就不必总是重复造轮子了

27.59–28.14
so a general–purpose the administers from the kind of things that we'll talk about in this semester are designed to allow applications to define, create, write, queries against update and manage of databases
我们这学期要设计一个通用的DBMS，即设计用于可以允许应用程序来对数据库进行定义，创建，查询，更新以及管理
28.14–28.19
Right, and for our purposes we'll assume our databases are stored in disks
为了我们的目的，我们假设我们的数据库是存在硬盘里的
28.19–28.24
they don't necessarily have to be like this in memory databases or the GPU databases or other things
它们不必是内存数据库或者GPU数据库以及其他类型数据库
28.24–28.28
for these we dont discuss those things yet,
对于这些，目前我们还不会去讨论
28.28–28.38
but just know that just be there's a variety of different databases out there that. can do a bunch of different things. to have search be specialized in different ways for a variety of applications
但要知道世界上有很多种不同的数据库，它们可以用来做许多不同的事情，针对不同的应用使用不同的方式来进行搜索

28.38–28.47
so again, I love databases, I think like database all the time, I love writing my databases or reading my databases
再说一遍，我对数据库非常痴迷，我喜欢写我的数据库或者读我的数据库
28.47–28.52
Um and you may think this is crazy why would anybody love databases of data systems so much
你可能会觉得为什么会有人如此痴迷于数据库或者数据系统
28.52–28.58
so you have to understand and think about this at CMU it's a we're a large school we have courses and everything
因此，你必须理解，这里是CMU，它是一所很大的学校，我们有各种各样的课以及事情
28.58– 29.08(and fernette bringing **没听出来**)
right ,you know in this courses for operating systems course and for networks things like that ,but database management systems are sort of a special class of software
通过这些课程，你可以学到操作系统以及网络相关的东西，但数据库管理系统是一类特殊的软件
29.08–29.10
that are so important
这非常重要
29.10–29.13
that like there's full-time people like me that teach of course just on this
有些人和我一样专门只教数据库相关的内容
29.13–29.16
right ,like a web browser is important
当然，web浏览器是很重要
29.16–29.19
but there's no class I'm like how to build a web browser at least a little as far as I know
但目前为止，我都没见过任何一门课是教如何构建一个web浏览器的
29.19–29.26
Right, whereas like databases are so prevalent and so widely used everywhere
像数据库这样的存在，它十分普遍而且无处不在
29.26–29.32
that like and ,they're really hard to you know work me and implement them that correctly
因为真的需要付出大量的精力对它们进行正确实现
29.32–29.35
that you know we there's an entire course in this course and talk about how to how to build it
基于此，你知道这门课就是专门讨论如何构建出数据库的
29.35–29.45
so again ,I think they're a unique piece of software that our class of software that is definitely a hot area right now
我觉得数据库是一种相当独特的软件，而且目前它绝对是热门领域
29.45–29.49
you know the end of the day machine learning these data you store that in the database ,
你知道，到最后，机器会学习你存储在数据库中的那些数据
29.49–29.57
and you know so at every single application as I said well at the end of the day there's a database underneath it running almost everything
你知道的，对于每个应用而言，它们最终底层都运行着一个数据库

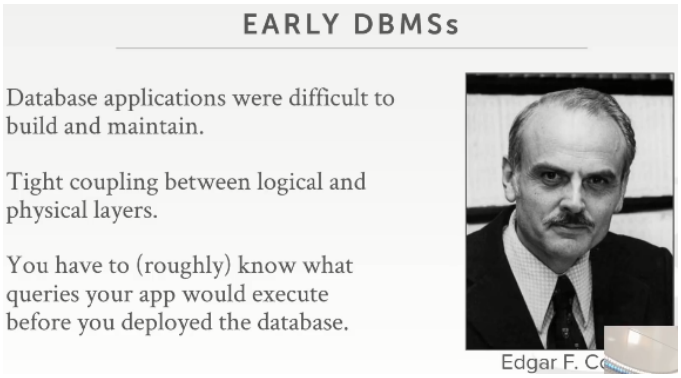
29.57–30.01
so this is why of course like this and my opinion is super important
这就是为什么我的观点是如此重要的原因了
==========分割线
==========



30.01–30.06
all right ,so databases are obviously systems are not new
很明显，数据库并不是什么新系统
30.06–30.13
you know the first one came online I think in like 1965 at General Electric
第一个上线的数据库是1965年由通用电气所制造的
30.13–30.25
and you know people were sort of in the early days of computing ,and then people quickly realized, hey it'd be nice to have specialized software Database system that can manage large data sets for us
在计算机早年发展期间，人们便很快意识到如果有专门的数据库管理系统来帮助我们管理数据，那会非常nice
30.25–30.30
so you have to understand back in the day it's not like how it was now
你要知道当时和现在完全不一样
30.30–30.38
In the early days ,some things that we'll talk about this course that we sort of take for granted now because ,oh of course this is how you want to do certain things
在早期，所讨论的一些内容已经在我们的课上被视为理所当然，因为当我们看到本课程这块内容的时候，你就会说，Oh，这当然是我们想要执行某些操作的方式
30.38–30.41
back like in the 1960s ~ 1970s ,it wasn't obvious that this is the way to do things
而在1960–1970年代，它可就不是这么一回事了
30.41–30.57
so in particular, the story I like to talk about is back in like the mid nineteen late 1960s, There was this guy Ted Codd who worked at IBM research
特别是我想讲的这个故事。它发生在20世纪中期1960年代左右。有一个叫Ted Codd的人，他当时在IBM进行研究工作
30.57–31.00
he was a mathematician when he got hired to work at IBM research in in New York
当他被聘请在纽约的IBM研究部门工作时，他是一名数学家
注：Ted.Codd 关系型数据库理论的奠基人

31.00–31.11
and he noticed that people that were working on databases spent a lot of their time rewriting their database application over and over again
他注意到，当时那些用数据库工作的人们花了很多时间一次又一次地重写他们的数据库应用程序
31.11–31.22
Because there was a tight coupling between the logical layers of what's in the database and the physical layers of how the data system was actually going to store it
因为在数据库的逻辑层和数据系统用来存储数据的物理层存在了一种紧密耦合
31.22–31.33
So what I mean by that is like ,say you have a database you wanna store in this Database system, and you have to tell you Database system ,oh I want you to store this as a hash table I want you to store this as a tree
我想表示的是，假设你有一个数据库，你想在数据库系统中存一些数据，你必须告诉数据库系统，你想让它用哈希表存储数据或者用树来存数据

31.33–31.40
and then when you did that and then expose a different API for you based on what data structure you chose
当你这样做时，然后基于你选择的数据结构，数据库会给你暴露一个不同的API
31.40–31.49
but then now say you change your mind on ,oh yeah I told you I was a hash table but I really want to run range queries on that, so now I want to do a I want to be you know stored as a tree
但现在你说你改变主意了，我已经告诉你我是一个hash table，但是我真的要在它之上进行一系列的查询操作，So现在我想要以一个树的数据结构来进行存储
31.49–32.00
So what you have to do is dump all your data out, change your application code to now make calls for the tree it's the API it's at the hash table API
你为了要转存你的数据，那就要去修改你的程序代码，也就是调用树的API而不是hash table的API进行处理
32.00–32.02
and then you have to reload all your data back
然后，你需要将你的数据重新加载回来
32.02–32.08
~~in write,~~ this again that whatever you told the database system how you want it destroy the data
无论是你要告诉数据库，你想要销毁数据，你都要重来一次（知秋注：发生转存后，要销毁数据的话，两种数据都要销毁）



32.08–32.15
so the Ted Codd realized that this was kind of stupid right this is problematic
Ted Codd意识到这样做非常蠢，而且很有问题
注：EdgarF.Codd(通常称为Ted.Codd)

32.15–32.22
Because it was the people basically refactoring the code all over and over again every single time there was ever a change like this
因为人们得一次又一次的因为一个改变而对代码进行重构
32.22–32.27
so now back then humans were cheaper than the computers
那时的人力成本要比电脑低廉的多
32.27–32.30
so but you can easily see how this was not scalable
你很容易就看到，这种做法是不可扩展的
32.30–32.38
and nowadays computing is cheap with Amazon ,and Microsoft ,and Google ,and the cloud computing and humans are expensive thing
但如今，计算成本非常廉价。我们有Amazon，Microsoft，Google以及云计算。然而人工成本才是最贵的
32.38–32.40
so it’s this problem is eat more problematic
这个问题就变成了大问题
32.40–32.42
you know this issue is even more problematic today
这个问题如今可能甚至变得更加严重
32.42–32.50
and then it was back in the 1970s ,but Ted Codd you know quickly realized that people were wasting their time in fixing up software that they didn’t need to
但回到1970年代，Ted Codd很快意识到人们是在浪费时间修复那些他们不需要的软件

DERIVABILITY, REDUNDANCY AND CONSISTENCY OF RELATIONS STORED IN LARGE DATA BANKS

E. F. Codd
Research Division
San Jose, California

ABSTRACT: The large, integrated data banks of the future will contain many relations of various degrees in stored form. It will not be unusual for this set of stored relations to be redundant. Two types of redundancy are defined and discussed. One type may be employed to improve accessibility of certain kinds of information which happen to be in great demand. When either type of redundancy exists, those responsible for control of the data bank should know about it and have some means of detecting any "logical" inconsistencies in the total set of stored relations. Consistency checking might be helpful in tracking down unauthorized (and possibly fraudulent) changes in the data bank contents.

RJ 599(# 12343) August 19, 1969

LIMITED DISTRIBUTION NOTICE - This report has been submitted for publication elsewhere and has been issued as a Research Report for early dissemination of its contents. As a courtesy to the intended publisher, it should not be widely distributed until after the date of outside publication.

Copies may be requested from IBM Thomas J. Watson Research Center, Post Office Box 218, Yorktown Heights, New York 10598

32.50–32.55
so what Ted Codd proposed was this thing called the relational model
Ted Codd所提出的东西被称为关系模型

Information Retrieval

P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity

CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting [illegible] element to participate [illegible] those existing systems [illegible] elements to be stored in [illegible] closely associated with [illegible] of addresses. For exam[illegible] parts might be stored [illegible] number. Such systems [illegible] grams to assume that t[illegible] from such a file is ident[illegible]

32.55–33.00
so he had the first paper came out on this in 1969
他的第一篇论文是在1969年发布
33.00–33.07
but the one that everyone's cites is this one from the communications of the ACM that came out in 1970
但每个人都引用的是1970年ACM所发布的那个版本
33.07–33.11
so most people read the one that relational model of data for large shared data banks
So大部分人都阅读了关于大型共享数据库的数据关系模型
33.11–33.17
but this is sort of the seminal paper that started the whole relational data model revolution
但这是引发整个关系数据模型革命的原始论文
33.17–33.20
of saying ,this is how you want to store the a databases systems
即你如何在数据库系统中存储数据
33.20–33.23
so this is very influential influential
因此，它的影响十分巨大

33.23–33.27
and this is this is sort of the backbone of what we talking in this course
这是我们在本课程中所讲内容的基础
33.27–33.34
so there's sort of three key tenants of the relational model that Ted Cobb proposed
Ted Cobb提出了关于关系模型的三要素

33.34–33.40
so the first that we were going to store the database in simple data structures as relations
所以首先我们要以将关系转化为简单的数据结构然后存入数据库
33.40–33.50
so a relation doesn't mean like you know I'm ,you know I'm related to my parents or whatever it realtions essentially its anomalous for a table
这种关系并不是意味着像你知道的我与我父母或者与之相关的事物之间的那种关系，从本质上来说，它与table无关
33.50–33.57
it's the the relationship can be actually stored in the tuple for for that given table
对于一张给出的表来说，所谓的关系实际上可以以元组（tuple）的形式进行存储
33.57–34.06
So again we would define at a high-level that would store all our tables at our database as these relations
So，我们从一个高级层面来对此进行定义，将所有的表存储在数据库中以建立关系（知秋注：表与表之间建立关系）

34.06–34.10
and then we would access to them through a high-level language
然后，我们通过使用高级语言去访问它们
34.10–34.18
meaning we would write in the exact code that we want the dataSet execute in order to retrieve the data that we wanted
这意味着，我们可以通过编写确切的我们想要的数据集执行代码来查找想要的数据
34.18–34.23
we would just say hey we want you to compute this answer please do it for me we wouldn't say how to do it
我们是想说，只想让你帮我回答这个问题，并没有说具体该怎么去做
34.23–34.26
so this idea was actually pretty was really revolutionary
实际上，这真的是一种革命性想法
34.26–34.30
because back then everyone was writing explicit procedural code
因为那时的人们还在编写明确的代码
34.30–34.33
you know here's the for loop to iterate through the table and find the data that I wanted
例如，使用for循环遍历整张表，然后找到我要的数据
34.33–34.38
the example that I showed in the very beginning, this is how people write database applications before the relational model
一开始我所展示的例子就是在关系模型出现前，人们编写数据库应用所用的方式
34.38–34.43
and you know this is actually pretty controversial at the time
在当时，这种做法其实非常有争议
34.43–34.50
because everyone was saying, there's no way software can ever produce a query plan that's as efficient as what a human can do
因为当时的人们表示，没有一种软件可以生成像人工那样高效的查询计划

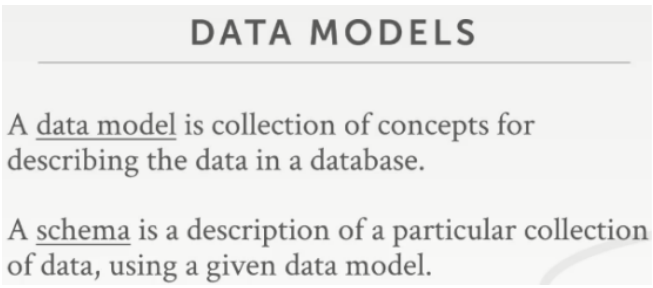
34.50–34.54 ****
this is sort of the same argument about that people made in 1970 about compilers
这与人们在1970年提出的关于编译器的观点相同
34.54–35.01
and they said, oh you know no compiler could ever write you know generate machine code as efficient as handwritten assembly
他们表示，任何编译器都无法生成和手写汇编一样高效的机器码
35.01–35.06
and of course nowadays nobody writes or very few of you people write low–level assembly
当然，如今已经没有人或者非常少的人去写低层次的汇编了
35.06–35.10
everybody writes in ,you know in high–level languages
所有人都使用高级语言去编写代码
35.10–35.15
and the compiler does a pretty good job producing a machine code executed more efficiently than human can do
编译器能很好地生成比手写汇编更高效的机器码
35.15–35.22
same thing was said back then right they a compiler or a database system could produce a query plan as efficient whether human could do
对于当时的编译器或者数据库系统来说，它们同样能够生成和人工一样高效的查询方案
35.22–35.25
and which nowadays we have very complex query plans
如今，我们有了非常复杂的查询计划

35.25–35.27
now optimizers aren’t perfect
现在的优化器并不完美
35.27–35.30
but they can probably do a better job and what most humans could do
但它们所能做的工作要比大部分用人工去做的要来得更好
35.30–35.43
All right, the last key idea was that the the physical storage, or the physical storage strategy for a big database was left up to the implementation of the database management system
最后一个关键思想就是大型数据库的物理存储策略取决于数据库管理系统的实现
35.43–35.49
so again we define our database through these high–level structures as relations
我们通过这些高级结构作为关系来定义我们的数据库
35.49–35.54
but how those relations are actually stored is really up to the implementation of the database system
但实际上如何存储这些关系则取决于数据库系统的实现
35.54–36.04
so the relational model doesn’t say whether it should be in memory, should be on disk how she laid out on disk or, how to organize memory ,all that is is transparent to the application

关系模型并不会去说这些东西它是否应该放在内存中或硬盘上进行布局，也不会说会如何去组织内存，所有这些对应用程序来说都是透明的

99
00:36:05 --> 00:36:13
So now the benefit of this is that if you know for some applications their database, you know may be stored in one way
所以这样做的好处是，如果你知道某些应用程序的数据库可能以某种方式进行存储
100
00:36:13 --> 00:36:21
and then over time, if the application involves, it may be better to store it in another way
然后随着时间的推移，如果这些程序迭代了，那么可能以另一种方式存储数据要来得更好
-36.24
And then we don't want to rewrite our application
我们并不想去重写我们的应用
36.24-36.30
because we're still writing at a high-level language like SQL and we're still accessing these relations
因为我们依然使用例如SQL这种高级语言去访问这些关系

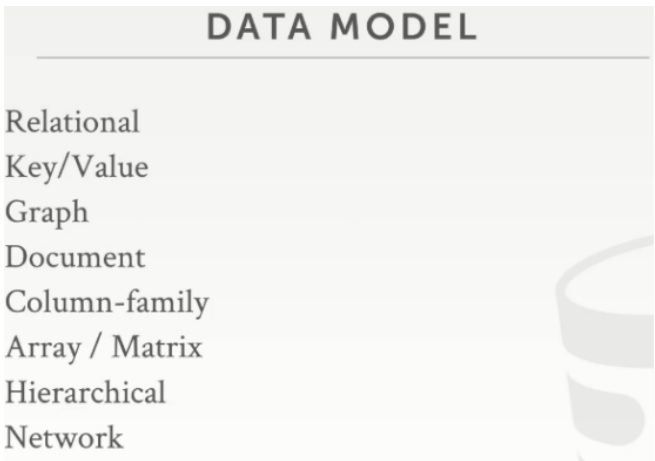
36.30-36.41 ****
but underneath the covers, this new system can start moving things around for us or changing its layout or recompiling certain things changing data structures, and we don't need to change our application code
~~但这背后，新系统可以为我们转移某些东西，或者改变它的布局或者重新编译某些东西，又或者修改数据结构，这样我们就不需要去修改我们的程序代码~~
但是在幕后，这个新系统可以为我们移动或更改其布局或重新编译某些东西以更改数据结构，因此我们不需要更改应用程序代码
36.41-36.47
so we have a clean separation between the logical and physical layers which is absolutely what we want
这样，我们就将逻辑层和物理层完全分开了，这绝对是我们想要的



36.47-36.54
All right ,so the relational data model is is not the only data model
关系型数据模型并不是唯一的数据模型
36.54-37.01
though it's certainly the most widely used data model and in my opinion I think it's the best data model

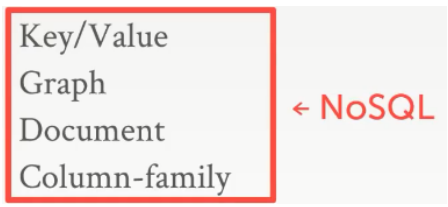
尽管它是使用最为广泛的数据模型，以我的观点来看，它是最好的数据模型
37.01–37.06
but there's you know there's different data models for different types of work listen
但你也知道，对于不同类型的工作就会有不同的数据模型
37.06–37.10
it's the relational data model sort of a catch–all that can come with a lot of things
关系型数据模型是某种能够包罗万象的模型
37.10–37.15
it's not to say that the other ones are bad or wrong
这并不是说其他的数据模型就不好
37.15–37.19
it's just a relational model nine times out of ten that's probably what you need
只是说关系模型十有八九就是你所需要的那个
37.19–37.26
so the again the data model is the high–level concept of how we're going to describe the data in our database
数据模型是我们用来描述数据库中数据的高级概念
37.26–37.35 待修改
and then the schema that provide for a given collection of data in database is the definition of what we're actually storing it
然后在数据库中为给定数据集所提供的schema就是我们存储数据时所使用的定义
37.35–37.38
so I'll show what I mean by this in a few slides
我会在接下来几张幻灯片中解释这点

37.38–37.43 ****
but the data model is essentially the high–level concept of how we organize the data
但数据模型本质上是我们如何组织数据的一种高级概念
37.43–37.49
and then the schema is to say alright what did the data were actually storing with this application for these given data model
然后，该schema就是指给定的数据模型，我们的应用程序通过该给定数据模型来进行存储数据



37.49–37.56
so again as I said ,the relational data model is just one of several other data models I'm sharing a small sample here
正如我所说，关系数据模型只是众多数据模型中的一个，我在这里分享一小部分例子

37.56–38.10
then most DBMS systems that you know about today and the ones we'll cover in this course MySQL ,PostgreSQL ,Oracle, DB2, SQL server, SQLite, all of these are relational data data database systems did use the relational data model
你所知道的当下大部分的数据库管理系统以及我们会在本课程中涉及的MySQL，PostgreSQL ,Oracle, DB2, SQL server, SQLite，这些都是使用了关系数据模型的关系型数据库管理系统



38.10–38.20
if you're familiar with the term you know SQL ,the NoSQL systems are usually these key value graph document or JSON or column family data models
如果你对SQL这个术语熟悉的话，那么NoSQL系统通常是Key/Value，Graph，Document，JSON或者Column-family这些数据模型
38.20–38.26
and again, I'm not saying necessarily that the relational data model is better than these other ones
再说一遍，我并没有说关系型数据模型要比其他数据模型更加优秀
38.26–38.34*****
there's certain application domains where some of these like it could better describe the data than the relational data model
在某些应用程序领域中，其中一些数据模型比关系数据模型能更好地描述数据
38.34–38.39
and the relational data model has certainly adopted some of these some concepts or ideas from these other data models
关系型数据模型已经采纳了其他数据模型的一些概念或者思路
38.39–38.45
in newer versions of it ,it's just you know the relational data model can model all these things
在它的最新版本中，关系数据模型可以为任何东西进行建模
38.45–38.59 （原文磕绊严重）
~~it just good bearing with~~ baring benefit advantages over there yeah sort of the basic data models
通过某些基本数据模型，那就可以获取到很明显的好处

38.59–39.03
All right, you can also store Array/matrix that's considered a data model
你也可以将数组/矩阵认为是一种数据模型
39.03–39.07

this is what people using machine learning
这是人们用于机器学习时所用的数据模型
39.07–39.10
All right ,there's some databases that can store matrix in arrays
某些数据库可以用数组来保存矩阵
39.10–39.20
they're not that common people usually use like data frames ,or you know sort of things that'll be example of a matrix data model
它们并不像人们平时所用的Data frame或者你所知道的东西那样。它是一个矩阵数据模型的例子

39.20–39.27
The Hierarchical and network data model in the multi–value data model there's a bunch of these other ones, these are more esoteric。
在MultiValue数据模型中，有着一系列更加深奥的数据模型，例如：层次数据模型和网络数据模型

39.27–39.36
these are what the original data models were used in the 1960s and 1970s ,you know sort of around the same time the relational data model came out
这些数据模型在1960和1970年代被使用，在同时期关系型数据模型诞生了
39.36–39.41
You almost never want they almost never will see these things
你永远不会想让他们看到这些东西（知秋注：没必要去看到这些底层实现）
39.41–39.45
if you're like a new start up or building an application for the first time
如果你是第一次构建此类应用程序（知秋注：DBMS）
39.45–39.48
you definitely want to use any of these things
你肯定会想去使用这些东西
39.48–39.52
they're mostly and legacy applications
它们大部分是很老的程序
39.52–39.55
and it's just the remnants of old software
这只是旧软件的残留
39.55–39.59
I would say this is not necessary to even consider them
其实我想说，我们甚至都不需要去考虑它们

39.59–40.03
so again ,for our purposes we're focused on the relational data model

为了我们的目标，我们关注关系数据模型
40.03–40.05
that’s what we care about in this course and that’s best we’re gonna focus on
这也是这门课我们所关心的重点

40.05–40.08 虚生花
cuz again the relational data model can be used to model anything
因为关系数据模型可以用来对任何东西进行建模