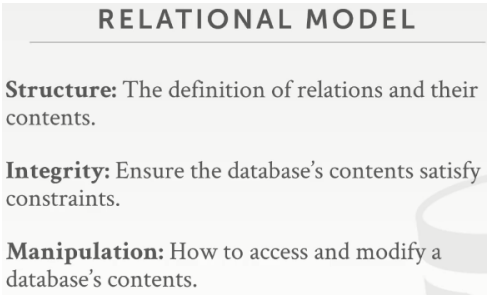
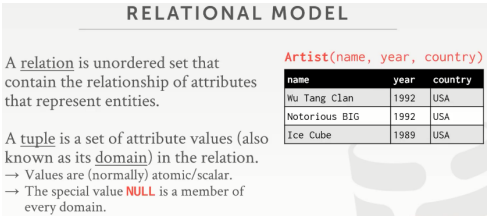
01–03

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

01–03



40.10–40.15 虚生花
So the relational data model is comprised of three parts.
关系型数据模型包含了三部分
40.15–40.19 虚生花
So again the first is the structure of the relations
首先包含的是关系结构
40.19–40.25 虚生花
this is like this schema, how we're going to define, what are in our relation is, what their activities are, whether types are and so forth.
这就像schema那样，即我们在我们的关系中该定义些什么，它们的内容是什么，类型又是什么，等等
40.25–40.28 虚生花
Then we'll have the integrity constraints
接着，我们有数据完整性约束
40.28–40.37 虚生花
that will define to specify what is a valid instance of a database given given the structure then given given the schema you provide.
然后，根据你所给出的结构以及schema，来定义为指定数据库的一个有效实例
~~这就会去定义什么是一个有效的数据库实例~~
40.37–40.44 虚生花
And then we'll have a way to manipulate and access the data in our database.
然后，我们会以某种方式来操纵和访问数据库中的数据
40.44–40.50 虚生花
Right,how do we actually run queries that can either extract information or or modify the contents.
好了，那我们该如何运行查询才能提取信息或者修改内容呢?

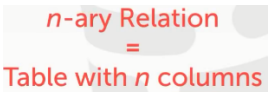


40.50–40.55 虚生花
So let's look let's go back to our music store example
让我们回过头来看下我们的音乐商城例子
40.55–41.00 虚生花
and look at some you know some concrete examples of what a relational model database looks like.
通过看下这些具体例子来了解一个关系模型数据库应该是什么样的
41.00–41.13 虚生花
So again ,an relation is just an unordered set of elements or records that have attributes that represent entities instances of entities in a relation.
~~关系是一组无序元素或者记录的集合，它们代表了某个关系中的实体实例~~
关系只是一组无序的元素或记录，这些元素或记录的属性用来表示关系中实体的实例。
41.13–41.17 虚生花
So again ,an artist has a name has a year in a country
在Artist表中有三个字段，即name，year以及country

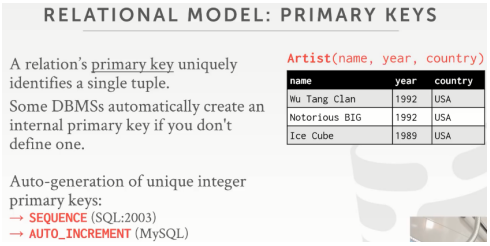
41.17–41.27 虚生花
and we can see in our relation example here wait wait we have, you know for each record is sort of one row in this in this diagram and we have all the those attributes.
我们可以看到在图中所示的例子中的每一行记录上都有这些属性
41.27–41.33
So we would refer to a record in the relational data model as a tuple right.
So，我们使用一个tuple来表达关系数据模型中的一条记录

41.33–41.41 虚生花
It's going to be the set of attributes that for that instance of an entity within our relation.
它是我们关系模型中一个实体实例对应的属性集
41.41–41.53 虚生花
So in the original data model, sorry the original relational model produced written like Ted Cod in the 1970s, all these values had to be atomic or scalar values
在一开始由Ted Codd在1970年代所写的原始关系模型中，所有的值都必须保证原子性或者单个属性值
41.53–41.57 虚生花
I mean it couldn't be arrays, they couldn't be nested objects and so forth
我的意思是它不能是数组也不能是嵌套对象等等
41.57–42.01 虚生花
always had to be sort of you know with one string one integer one one float.
它往往是我们所熟知的一个字符串，一个整数，一个浮点数之类
42.01–42.06 虚生花
So again, that's what the original data model relational data model talked about
这就是原始关系数据模型所讲的内容
42.06–42.15 虚生花

, but you know in recent times ,you know you now can store arrays you now can store JSON objects in relational databases
但在最近，你可以在关系型数据库里存入数组，JSON对象
42.15–42.18 虚生花
that particular restraint has been relaxed
之前那种限制已经没有了
42.18–42.24 虚生花
Now there's also gonna be a special value in the domain of every attribute store
如今，你也可以在每个属性中存入一个特殊值
42.24–42.28 虚生花
they'll call the null value that means that, you know the value is unknown.
我们将它称之为NULL，这意味着这个值是未知的
42.28–42.31 虚生花
So that'll cause some problem when we start running SQL queries
当我们运行SQL查询时，这会引起一些问题
42.31–42.36 虚生花
but pretty much every database management system supports reports relational database system supports null values.
但现在很多数据库管理系统都支持了值为NULL的这种情况
42.36–42.40 虚生花
But how do you actually store them is left up in the implementation
但实际上你该如何储存它们，这就取决于实现了
42.40–42.42
and there's a bunch of different ways to do that which i think is pretty interesting.
~~我们可以通过许多不同的方式可以做到这点，对此我觉得很有趣~~
这个可以有很多不同的方式来做到，，对此我觉得很有趣



42.42–42.53
All right ,~~so then just so we you have to understand~~ the parlance we'll say that n–ary relation is one is a table that has n columns .
n–ary 关系其实就是一张表上有n列
42.53–43.01
So I'll use the term relation and table and unchangeably they mean the same thing for our discussions
在我们的讨论中，我会使用relation和table这两个术语，实际上它们是一回事
43.01–43.05 ~~（语句有问题，明天再着重听下）~~
I'll say tuples sometimes as a record
有时我会把tuple说成一条记录
43.05–43.09
I trying not to say row, because that's something very specific when we actually start to talk about storage models
而不是把它说成行，因为行是某种很具体的东西，当我们实际开始讨论存储模型时会说到
43.09–43.12
so but all of those works been used interchangeably as well.
~~但这些东西也能互相交换使用~~
但所有这些东西也可以用来换着讲，差不多一个意思



43.12–43.19
All right, so one thing we also have in a relational model is primary keys.
我们在关系模型中还有一个东西就是主键（Primary Key）
43.19–43.28 *******
So primary key is going to be a unique attribute or set of attributes that can uniquely identify a single tuple .
若其中某一个唯一属性或一个属性组能唯一标识一条记录，该属性或属性组就可以称为主键
43.28–43.36 虚生花
So in this example here, there's a we actually don't have a primary key on friend there are attributes
实际上在这个例子中，我们并没有主键
43.36–43.40 虚生花
because there could be other artists called the Wu Tang Clan
因为可能还有其他也叫武当派的艺术家
43.40–43.49
there aren't, but there could be right there's nothing in our in our in our in the world that prevents somebody from calling them calling them that themselves that.
因为在这个世界上，我们也没办法阻止别人重名

Artist(id, name, year, country)

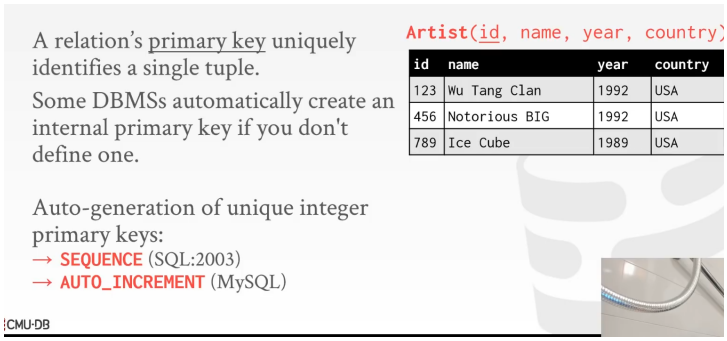
id	name	year	country
123	Wu Tang Clan	1992	USA
456	Notorious BIG	1992	USA
789	Ice Cube	1989	USA

43.49–43.54 虚生花
So we can introduce a new ID field as a unique primary key
因此，我们可以引入一个新的字段id，使用它来作为唯一主键
43.54–44.01 虚生花
so that you know if you're looking at the tuple with the ID 1 2 3 ,you're explicitly looking at you know the original Wu Tang Clan.
这样你就知道如果你查看id为123的元组(tuple)，你就能明确找到这个原始的武当派记录
44.01–44.11
So in this example here ,this is like a synthetic key that I introduced into my relation my schema here
so 在这个例子中，这就像将一个起代理作用的key引入到了我这里的关系schema中

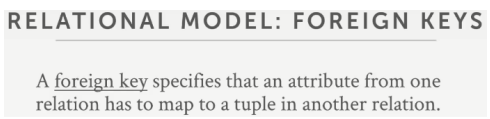
44.11–44.19
right there in a real artist they don't have an ID 123 ,this is something we're using in our database management system for this particular real relation to uniquely identify the tuple.
现实生活中的艺术家并没有123这样的id，这只是我们在数据库管理系统中唯一能用来识别这个Tuple的东西
44.19–44.23

So there's different ways to generate these things.
有多种方法可以生成这些东西
44.23–44.30
So there's some systems at least in the single standard you can automatically generate these auto incrementing keys.
在某些数据库管理系统中至少会有一个标准可以用来自动生成自增主键

44.30–44.35
So at every single time certain in tuple there's some counter that increments by one.
每次生成一条tuple时，它的主键就会自动加一
44.35–44.37
So that new tuple will get that unique identifier.
这样新的tuple就会得到一个唯一标识符
44.37–44.44
Other systems if you don't specify a primary key internally about create one ,they don't really expose it to you
在其他系统中，如果你在内部不指定一个主键，它们就不会将它暴露给你
44.44–44.49
but like they'll use that to keep track of and this is the particular physical record or physical to boat that you're looking at.
但它们会使用主键用来跟踪你要查看的这条物理记录



44.49–44.54
All right, for this example here it's a synthetic one that's exposed through the logical layer.
对于我们这里的这个例子，它是通过逻辑层来暴露的一个代理键
44.54–44.58
So they could be generate through it auto increment key .
So它们可以通过自增键来生成

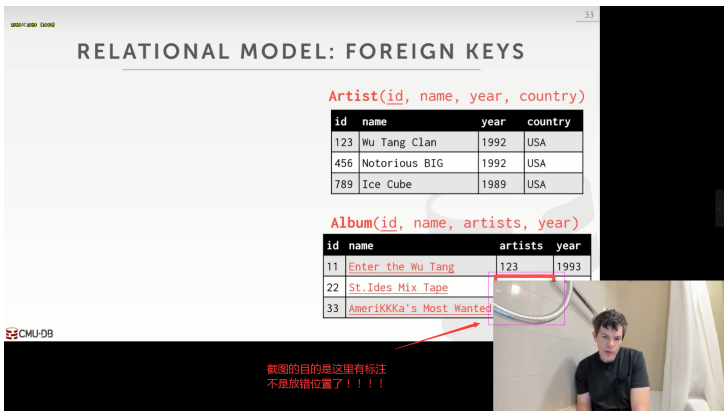


44.58–45.02 虚生花
There's also foreign keys in the relational data model
在关系型数据模型中也存在了外键（foreign keys）
45.02–45.11
so foreign key is a way to specify that an attribute from one relation has to exist in at least one tuple in in another relation.
~~外键用于与另一张表相关联。是能确认另一张表记录的字段~~

外键用于指定一张表中的属性必须存在于另一张表中（知秋注：上文有说，relation 和table就是一个语义）
45.11–45.15 虚生花
So this is how you're gonna maintain integrity across different relations
这就是你如何用来维护不同关系间的数据一致性
45.15–45.22
so that you don't insert things that map to unknown or non existing entities in another table.
这样你就不会在插入数据时，将这些数据映射到另一张表上未知或者不存在的实体之上

45.22–45.25 winston
So going back and to to our example here.
来回到我们的例子中

45.25–45.42 winston
So we have the Artist ,we have the Album, and so for the Album relation,we may want to store multiple artists that could all collaborate on a on a on an album together.
我们有Artist，我们有Album，因此对于Album表，我们可能想要存储一张专辑中所合作的多个艺术家名字
~~需要在一张专辑存储一起合作的多位艺术家。~~



45.42–45.50 winston
So we have this artists field now here, but in this particular example,right for this mixtape there's multiple artists.
因此，我们这里有这样一个artists字段，但在这个特殊的例子中，这个混音专辑有多位艺术家。

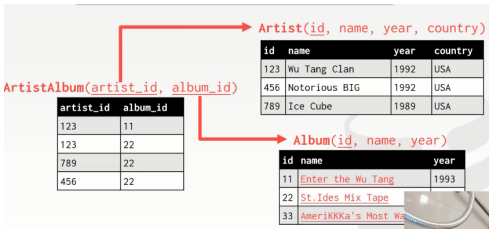
45.50–45.54 winston
So how do I can't encode multiple values in the artists field,because I said they had to be scalar.
所以，我不能在artists字段中编码多个值，因为我说过他们必须具备原子性（即单个值）。

45.54–45.59 虚生花

So instead I'll make this separate cross-reference table here .

~~所以 我会让这个独立的交叉引用表在这里代替~~

So，这里我会单独使用一张中间表来做到这点



45.59–46.07 虚生花

I'll have a foreign key reference from the artist_ID that is part of this album and then album_ID that they're involved in.

我会将Artist中的id字段作为外键使用，然后和这些艺术家所参与的专辑id产生关联

46.07–46.17 winston

So now,I can store multiple artists on a single album ,and the Databases management system will keep track of the whether these albums and artists_IDs actually exist in the real relation.

所以现在我可以将多位艺术家存储在一个专辑中，数据库管理系统将追踪这些albums和artists的id之间是否存在真正的关联关系。

46.17–46.24 winston

So that prevents me from inserting an album that has artists that don't actually exist.

所以这可以防止我在album表中插入一位不存在的艺术家。

46.24–46.29 虚生花

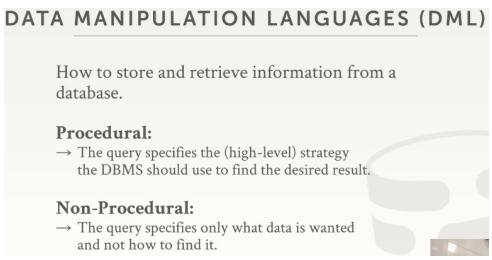
Right, the Databases system can protect me from inserting bad data

这样，数据库系统就可以防止我插入错误数据了

46.29–46.36

again I haven't implement all that extra code and myself in my application which again we're leaving from running really hard code.

我并没有实现所有这些额外的代码，在我的应用程序中我们也不需要运行很难的代码



46.36–46.42 winston
All right ,so now we want to talk about how we actually want to get data out of our database.
好的，现在我们谈论下怎么在数据库中取出我们想要的数据

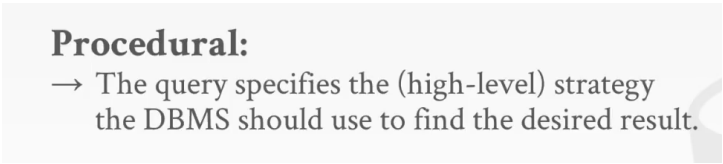
46.42–46.45 虚生花
Right again, I show that simple example,I'm writing the for loop
之前我已经展示过我用for循环来提取数据的简单例子
46.45–46.46 虚生花
but I said that was a bad idea
但我说过这是一个很糟糕的思路
46.46–46.49 虚生花
we don't wanna have to keep doing that for application
我们不想一直在我们的应用程序中干这种事情

46.49–46.50
so what do we want to do.
那我们想要做的是什么呢

46.50–46.59 虚生花
So the DML is a way to manipulate the data and you're accessing it or modifying a way to produce it the result of I'm looking for.
~~DML 是操作数据的一种方法，你访问它或修改的方式可以得到要找的结果~~

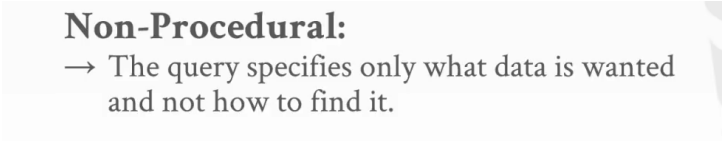
DML是一种操作数据的方式，通过它，你可以用来访问或修改数据库来生成你想要的结果
46.59–47.02 虚生花
So there's two ways we could do this .
我们有两种方法可以做到这点

47.02–47.05 虚生花
So the first is procedural
第一种方法是通过程序来做到

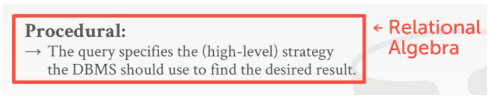


47.05–47.13 虚生花
meaning we'll specify at a high–level strategy of how the data system should find our particular result that we're looking for .
我们通过指定一种高级策略来让数据系统帮我们找到我们想要的结果
47.13–47.17 虚生花
So I don't mean exactly like the procedural code the Python example I showed before.
我想说的并不是像我之前用Python所写的代码那样
47.17–47.22 虚生花
But just sort of a high–level say ,hey this is what we won't need to execute.
但从上层的角度来讲，这并不是我们需要去执行的东西
47.22–47.27 虚生花

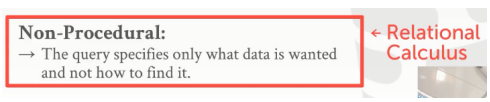
The other approach is called non-procedural or declarative and this is where we're gonna say.
另一种方法被称为非程序性或者声明式，这也正是我们要说的
47.27-47.32
This is what we want this is the answer we want the data system to generate for us
~~这也正是我们想让数据系统为我们生成的答案~~
这就是说，所执行东西我们希望数据系统为我们生成



47.32-47.36 虚生花
but we're not gonna specify how to actually produce it or actually how to go about and getting it
但实际上我们不会去指定想要数据该如何生产并返回给我们
~~明确表示如何生成这些数据并返回给我们~~
47.36-47.42
So we're gonna say ,hey at a high-level do this and we hope the data system can do it for me to figure out an efficient way to do it for us.
因此，我要说的是，从高层次上讲，我们希望数据系统能够为我做这件事，以找出一种为我们做这件事的有效方法。



47.42-47.48 虚生花
So the we talk about now is relational algebra this would be an example of a procedural language .
我们现在所讨论的是关系代数，它是过程语言(procedural language)的一个例子
47.48-47.52 虚生花
And relational algebra is it's not hard to understand
要去理解关系代数并不是很难
47.52-47.57
it'll come up later on when we talk about query execution .
在我们谈论执行查询时我们会再去讨论它
47.57-48.02
But you know and this is what – what Ted.Codd controversially proposed in 1970s.
但你知道这也是Ted Codd在1970年代所提出的论点

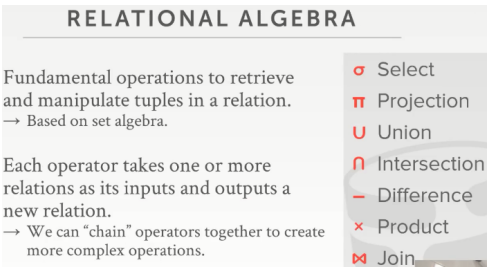


48.02-48.06 虚生花
Non-Procedural example would be something like relational calculus.
非过程化的例子看起来就像是关系演算之类的东西
48.06-48.09 虚生花
And so we don't need to discuss this at all.
我们也不需要讨论这个
48.09-48.17 虚生花

Not because I don't you know I don't think it's actually necessary at least for our purposes in this course .
但至少对于我们课程学习的目标而言，我觉得没必要去提到它
48.17–48.21 虚生花 zzzzzzzzz
But relational calculus will come up if you be certain on in detail about query optimization
如果你想深入了解查询优化，那么你就需要学习关系演算
==========分割线==========

48.21–48.27 虚生花
because you have to use calculus to derive efficient query plans and do pruning and things like that.
因为你必须使用微积分去推导出更高效的查询方案
48.27–48.31 虚生花
So again ,for this course we all need to focus on relation algebra
再说一遍，对于这门课，我们只关注关系代数

48.31–48.37 虚生花
I'm just bringing up relational calculus to say that it exists that it's out there it's an example of a non-procedural language.
~~这里我提到关系演算，也只是为了说存在子这么一个东西，它在我们这里一个非过程化语言的例子中使用子~~
我只是在提出关系演算，以指出它存在于非过程语言的一个例子中。
48.37–48.40 虚生花
But for our purposes here we don't need it.
但对于我们这门课的学习目标而言，我们不需要去学习它



48.40–48.49 虚生花
All right，so Ted Cobb proposed seven fundamental operators in relational algebra .
~~所以Ted.Cobb提出了关系代数的七项基本算子~~

All right，Ted Codd提出了在关系代数中七种基本运算符
48.49–48.55 虚生花
So these again these are the fundamental operations we have to retrieve manipulate tuples in a relation to produce answers that we're looking for.
~~这些是我们用来操作某个表中记录来生成并返回我们要找的结果~~
so，这些又是我们检索处理表中记录以生成我们要找的结果所必须的基本操作。
48.55–49.02
So one important thing to mention is that this algebra is based on on sets.
有一件重要的事需要提一下，那就是这种代数是基于集合的
49.02–49.10

So the set is an unordered list or unordered collection of data where you cannot have unique values.
So 这种集合是一种数据的无序列表或无序集合，这里面你的元素可以重复
49.10–49.19 虚生花
Now when we talk about SQL，SQL not going to be based on said algebra even though it uses relational algebra as the underlying operators for doing query execution.
当我们谈论SQL时，即使SQL使用关系代数作为执行查询的基础运算符，SQL也不会基于上述代数。
~~SQL并不依赖于关系代数，即使它使用关系代数作为查询执行时底层的操作符来使用~~

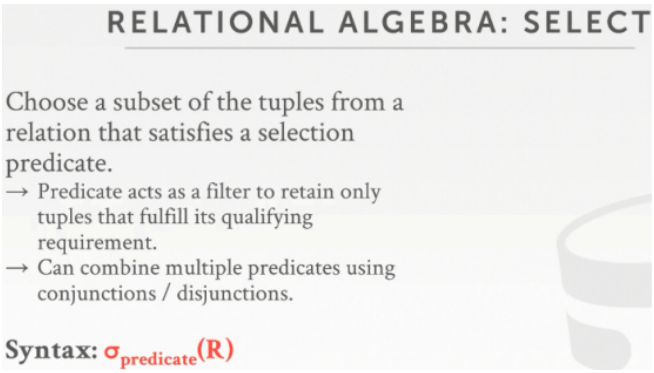
49.19–49.22 虚生花
And I'll explain again when we talk about SQL next time what that means.
我会在下次我们讨论SQL时解释这是什么意思

49.22–49.32 虚生花 !!!!!!!!!
So the way this is going to work is that each of our relational operators is going to take in one or more relations as this input, and then it's always going to output a new relation.
我们每个关系操作符会使用一个或多个关系作为输入，接着它始终会输出一个新的关系

49.32–49.42 虚生花
And so the idea is that we're gonna compose complex queries by chained together these relational operators together to produce the answer that we're looking for.
这里的思路是我们将复杂的查询通过这些关系操作符链接在一起，以此来生成我们想要的答案

49.42–49.48 虚生花
So these are like the primitives you would use to do some operation on the data on tuples
通过使用这些原语（知秋注：可以理解为复杂函数中的基本函数，即复杂查询操作中包含的基本查询操作），你可以对这些数据或者tuple进行某些操作
49.48–49.52 虚生花

and then to produce a comp more complex query and we put them together.
然后生成一个更复杂的查询，接着我们将它们放在一起
49.52–49.58 虚生花·
So just select projection Union intersection difference product and join.
So我们就只用select、projection、Union、intersection、difference、product以及join这些操作符就能完成
49.58–50.00 虚生花
So again these are the fundamental ones and we'll go through each of these .
它们是这些的基础，我们会对它们逐个进行讨论



50.00–50.09
So the first one is select ,select is basically taking a subset of the tuples that satisfy some selection Predicate.
~~首先是select操作，它可以从根据某些条件来选出一些tuple的集合~~
首先是select操作，它可以根据一些查询条件来得到符合条件的tuples的一个子集
50.09–50.11 虚生花
So you can sort of think this is like a filter
因此，你可以把它当做是一个过滤器
50.11–50.18 虚生花
you say, here's for a given input relation, here's the predicate that I want to evaluate for every single tuple
假设这里有一组给定的输入关系，然后我要对每个tuple进行条件判断
50.18–50.29 虚生花
and if it evaluates to true meaning the predicate is satisfied particular tuple will add it to our output relation for this for the Select operator .
如果结果为true，这也就意味着满足条件的tuple就会被添加进我们输出的关系
50.29–50.37 虚生花
So the easy way to remember this the word select starts in S and so the relational operator is a sigma which also starts with S.
有个记住这个select的简单方法，那就是select是S开头，并且它的关系操作符σ也是S开头
50.37–50.44
So you would say here's I want to for this given input relation here it's a predicate to run and may produce our output .
so，你可以说，我想在这里为它放置一个关系，该关系是一个条件，用于产生我们的输出

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a2	103
a3	104

50.44–50.50 虚生花
So during a real simple example here say we have a simple relation has two attributes a_ID and B_ID.
来看下这个简单的例子，在这个例子的关系中，我们有两个属性，即a_id和b_id

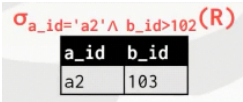
$\sigma_{a\_id='a2'}(R)$

a_id	b_id
a2	102
a2	103

50.50–50.57 winston
So we could do a select operator like this that says, find me all the tuples where a_ID equals a2 tuple
我们可以做一个像上面写的查询操作，找到我所有a_ID是a2的tuple 。

50.57–51:06 winston
and then our output would be a new relation with these two tables, and the same schema or the same attributes as the as the input relation.
~~我们查出的（数据）将与这2张表做关联，同结构或同属性作为输入关系。~~

然后我们的输出即是这两张表在某个新关系下的产物，并且该关系具有与输入关系相同的结构或相同的属性。

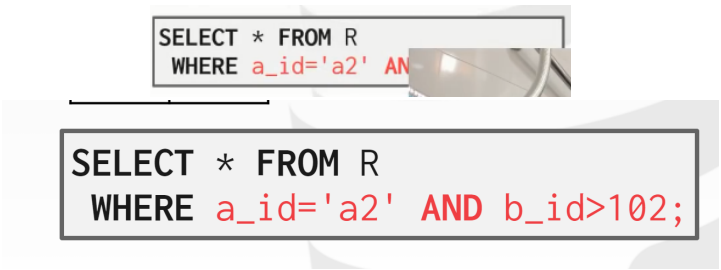


51:06–51:15 虚生花
I also can combine together using boolean logic together more complex predicates.
我也可以通过使用布尔逻辑将条件判断部分变得更加复杂

51:15–51:21 虚生花
So now you say where A_ID equals 2 and B_ID is greater than 102.
~~现在你说找a_id = ‘a2’ ^ b_id > 102.~~
比如我们可以这样说，我们要找的是a_id为a2并且b_id要大于102的结果

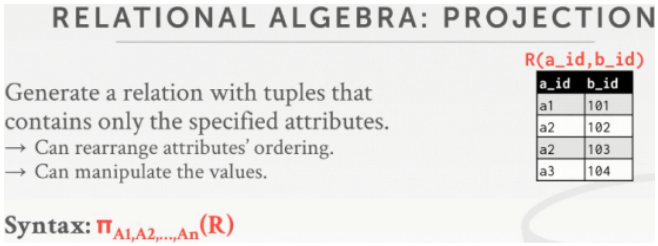
51:21–51:28 虚生花
And again ,we’re not specifying in the order in which you apply those predicates,right because it could be more expensive to do one versus the other
这里我们并没有指定这些条件判断的顺序,因为交换条件判断顺序的话，代价可能会变得更高
~~51.32–51.28 虚生花~~

51.28–51.31 虚生花
that’s all left up the database system just to decide on its own what to do.
这一切都留给数据库系统自己去决定该怎么做

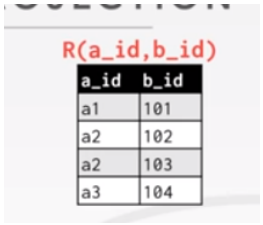


81
51:32–51:41 虚生花
So now if you know SQL the way this would rewrite this in a SQL that for that second example, if you think of the select operator I’d like the where clause .
~~so,现在如果你知道第二个例子SQL重写方式，如果你想查询操作，我想在where子句。~~
如果你知道如何用SQL来重写第二个例子的话，你可以在这个Select语句中的WHERE处进行修改
~~将select操作符当做是SQL中的WHERE关键字~~
82
51:41–51:53 winston
As you say, where a A_ID equals 2 and B_ID is greater than 102,right that’s a translation of that of this select operator into into SQL here.
~~正如你所说，where a_id = ‘a2’ ^ b_id > 102，这就是查询操作在SQL里的翻译。~~

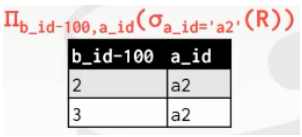
你可以这样来写：where a_id = ‘a2’ ^ b_id > 102，这是将此select操作翻译为这里的SQL
~~这是用SQL的方式来表达select操作符~~



51.52–51.55
All right, so the next operator is projection
好了，下一个操作符就是projection
51.55–52.06
and ,and here what we’re doing is we’re gonna generate a new output relation that contains only a subset of the specified attributes from our input relation.
我们要做的就是生成一个新的输出关系，它里面只包含了一个来自我们给定输入关系中的指定属性（知秋注：即对应表中定义的字段）的子集
52.06–52.15
Right, so you would say here’s my in population, and then my projection is here’s the attributes that I only want you to admit to police in the output.
Right, so 你可以说这里我有一些人， 接着，我的projection 操作在这里的属性（知秋注：即所操作表中的属性为population）包含的数据为：我只想允许police （警察）在我的输出中
52.15–52.22 虚生花
And again projection starts at the P it uses a lowercase PI symbol so it’s very easy to remember that there’s which one it is.
projection的开头字母为p，并且它使用小写的π作为符号，因此很容易记住它是哪个。



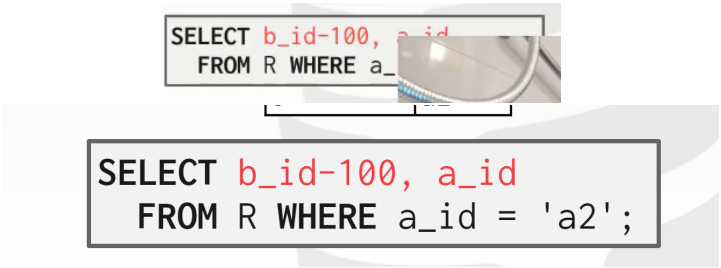
52:22–52:28
So again here’s our simple example with relation R with a_ID,B_ID.
如此反复，这是我们简单的例子关系R跟A_ID，B_ID。
so，再次，这是我们的那个简单的查询操作（查询关系为R，R操作的表属性为a_id,b_id）



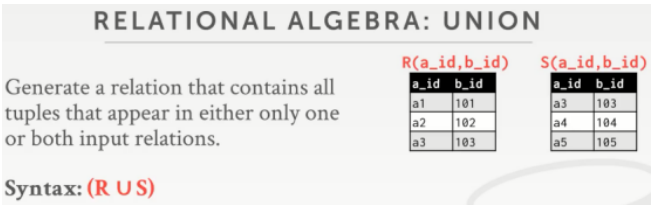
87
52:28–52:47
So in this case here I can have a I first do a select to just produce all the tuples of R where a_ID equals 2, and then now in my projection clause I can say, I want the B_ID value subtracted by a hundred, and then well as the a_ID value.
so，在这里，我可以先做一个select 操作,得到R关系（即a_id=’2’）下的查询结果，然后就是我的projection 操作，我可以说，我想要b_id的值减去100，即根据a_id的值查询得到的结果再去做这个操作

52:47–52:49 虚生花
So I can reorder my attributes any way that I want
因此，我可以以我想要的方式来对我的属性进行重排序

52.49–52.55
and then I can manipulate that an energy way to want with the any kind of arithmetic or kind of string function operator.
然后我可以使用一种算术或某种字符串函数运算符来得到一种想要的方式来进行操作。

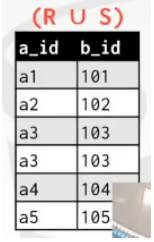


52.55–52.58 虚生花
So again, in SQL it would look like this
如果用SQL来写，那它可能就长这样
52.58–53.03 虚生花
so you have in the output Clause that your Select statement you say B_ID listen hundred in a_ID.
为了对应你的output输出形式，你的select语句应该这样写SELECT b_id–100, a_id
90
53:03–53:06 虚生花
And then the where clause is just which get it in the Select operator.
然后WHERE子句中就放入select操作符的条件判断部分的内容，即a_id= 'a2'



53.06–53.20 虚生花
All right, Union operator is where we're going to take two relations, and we're going to produce a new output relation that contains all the tuples in either the first relation or the second relation or both of them.
All right, Union操作符的作用是将两个关系组合生成一个新的关系，这其中包含了这两个关系中的全部tuple

53:20–53:24 虚生花
Right, you just you just combine these two relations together
Right, 你只需将这两个关系合并即可



(R ∪ S)

a_id	b_id
a1	101
a2	102
a3	103
a3	103
a4	104
a5	105

53:24–53:36 虚生花
Right, so if you do the Union on R and S， you get a giant, you know you get a combination here basically concatenated S on R to own are as the output .
如果你对R和S进行Union操作，你就会得到一个大的集合，其本质就是将S对R进行级联然后得到一个输出

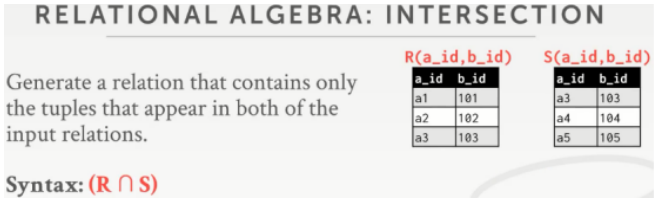
53:36–53:42 虚生花
And so for this particular operator you have to have the same attributes with the same type in the two relations you're trying to Union together.
对于这个操作符而言，当你想对两个关系进行Union操作时，这两个关系必须具有相同属性以及相同类型才行

53:42–53:49 虚生花
All right, in this case here if s had you know had a third attribute, you wouldn't be able to be Union because it wouldn't match .
在这个例子中，如果S有第三个属性，那么你就不能进行Union操作，因为它们没办法匹配

```
(SELECT * FROM R)
  UNION ALL
(SELECT * FROM S);
```

53:49–53:53 虚生花
So in SQL there is a Union all operator.
在SQL中有一个UNION ALL操作符
53.53–54.02 虚生花
So the Union one is not exactly the same as the Union operator in in relational algebra
但这个UNION ALL和关系代数中的Union操作符并不是完全一样的
54.02–54.06 虚生花
you have to add this all qualifier to get it to do what exactly what you want to do.
你必须添加这些限定符才能按照你想的那样进行

54:05–54:10 虚生花
So again what we'll discuss what the difference of these two is in the next lecture.
我们会在下节课去讨论它们俩之间的区别



54:10– 54.20 虚生花

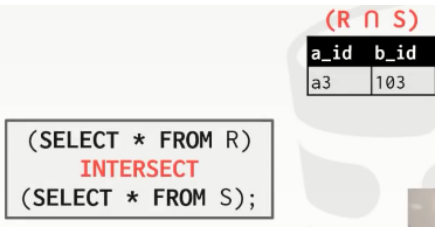
There's also intersection same thing, so now what you're doing is taking all the you're taking for both the good for 2 input relations

对于Intersection操作符也是一样,你将两个关系作为输入

54.20–54.25 虚生花

you're just going to produce the output relation that has tuples that appear in both the two input relations

然后你就会生成一个包含了在两个关系中都出现过的tuple的输出关系



54.25–54.36

So again taking the Union R and s here, we'll just produce a single tuple that has no sorry, a single relation that has one tuple a a3 and 103 .

只有一个tuple在R和S中都出现了，那就是a3和103

54:35–54:44 虚生花

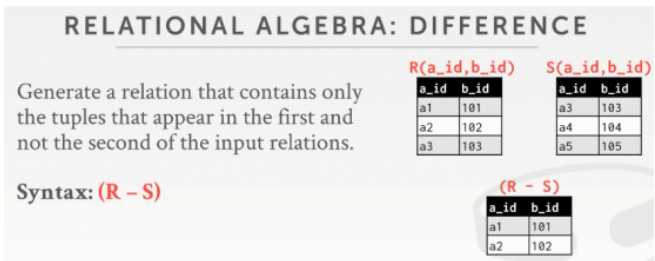
So again you have to have the same number attributes the same type in the same name in relational algebra in order for this union to work .

只有当R与S都具有相同属性，相同类型以及相同名字时，关系代数中的Union才能正常工作

54:44–54:51 虚生花

And then there's the intersect keyword and in SQL that would do the same thing.

然后，对于SQL中这个INTERSECT关键字，同样如此



54.50–54.56 虚生花

Difference is where you take all the tuples that appear in the first relation but not the second relation .

Difference操作的作用是只取在第一个关系中出现的元素，而不是第二个关系中的元素（相当于从第一个元素集中将第二个元素集中出现的东西都过滤掉）

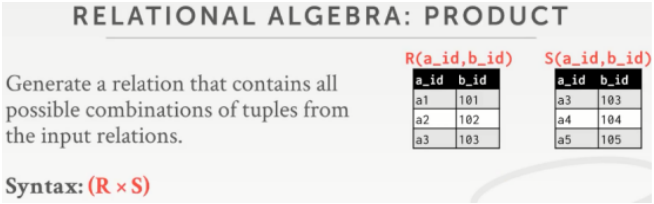
54:55–55:04 虚生花
So again I can take the difference of RS is just all the tuples that don't appear in S that appear in R .
也就是说，对于R–S，我可以拿到所有出现在R但并没有出现在S的tuple

55:04–55:08 虚生花
So in this case a3 103 appears both in R and S.
在这个例子中，a3和103在R和S中都出现了

55:08–55:15 虚生花
And so it's excluded in the output, but a1 a2 don't appear in S. So they're in the output
因此，它并不在输出里面,但a1和a2并没有出现在S中，因此它们在输出结果里面

```
(SELECT * FROM R)
    EXCEPT
(SELECT * FROM S);
```

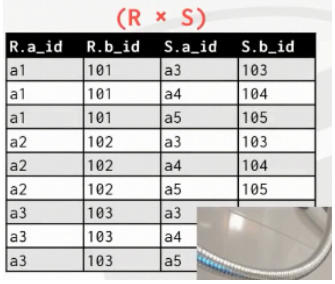
55:15–55:23 虚生花
And in SQL you use the except keyword, except operator to perform the same kind of operation.
在SQL中，你可以使用EXCEPT关键字来做到相同的事情



55:23–55:28 虚生花
All right then the second last one to do is the product operator .
好了，我们要讲的倒数第二个操作符是Product操作符

55:28–55:31 虚生花
So this is also sometimes called a Cartesian product
有时它也被叫做笛卡尔积
55.31–55.34 虚生花
or if you know SQL ， this is I think called a cross join.
如果你了解SQL的话，它在SQL被称为交叉连接（Cross Join）

55:34–55:40 虚生花
So basically you just want to generate all combinations of all tuples from the two input relations
你可以通过它来从两个输入关系中生成所有tuple的可能组合



(R × S)

R.a_id	R.b_id	S.a_id	S.b_id
a1	101	a3	103
a1	101	a4	104
a1	101	a5	105
a2	102	a3	103
a2	102	a4	104
a2	102	a5	105
a3	103	a3	
a3	103	a4	
a3	103	a5	

55:40–55:47
So you're basically just taking the cross product means to and just producing a giant output of every single unique combination of all of them.
当你使用笛卡尔积时，这就会产生一个巨大的输出，这里面包含了所有可能的不重复的组合
~~这里你基本上讲的笛卡尔积指 产生大量的它们之间每一种不重复组合的输出。~~

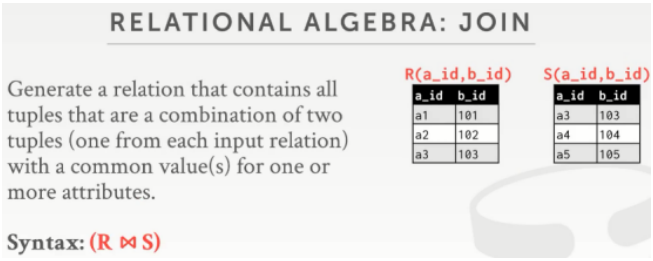
```
SELECT * FROM R CROSS JOIN S;

SELECT * FROM R, S;
```

55:47–55:56 虚生花
So again ~~in~~ in SQL use cross join or if you don't actually specify that what the join is at all, you get this output here like that .
在SQL中你可以使用CROSS JOIN，实际上如果你不明确使用哪种JOIN，你就会得到这样的输出
113
55:56–56.00 虚生花
So you may think this is kind of stupid this is like what you not I don't want to do .
So，你可能觉得这会很蠢，这也是你不想去做的事情

56.00–56:02 虚生花
You never want to do this
你永远不会想要做这种事情
56.02–56.12 虚生花
but this actually appears sometimes in testing some testing applications when you want to say, give me all the unique combinations of different configurations of the, I think I'm gonna actually trying to test.
但实际上这有时会出现在测试应用程序的时候，即当你想要得到不同配置的所有不重复组合时，你可以试着这么去测试

56:12–56:16 虚生花
And this is a really simple way to actually do that.
~~实际上这是做到这点的一种简单方法~~
通过这种很简单的方法就可以做到这一点



116

56:16–56:22 虚生花

All right the last time got to talk about is to do joins .

好了，我们最后要讲的就是Join了

56:22–56:25

So this particular join is called a natural join .

这种Join被称为自然连接（Natural Join）

56:25–56:28

It's not the kind of joins we'll talk about in the future .

这并不是我们之后要讲的那种Join

56:27–56:33

But not necessarily type of join type in the future, but this is very explicitly talking about a natural join .

尽管这种并不是我们未来必须的，但很明显会去涉及到这个自然连接

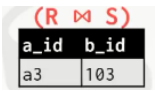
56:33–56:49

So way the natural join works is that you for every single tuple in the output out one relation,

自然连接的工作方式是，对于一个关系中的每个tuple

you see whether it matches all the attributes in the other relation that have the same name the same type ， so the common guys of them.

你会看到它是否与另一个关系中具有相同名称，相同类型的所有属性匹配，so 这就是这两个关系共同拥有的元素。



56:49–56:55

Right, so in this case here if I want to do a natural join on R and S .

So在这个例子中，如果我想对R和S进行自然连接

56:55–57:10

I look at a_ID and see whether I find a match, I look at the a_id are see where that I've got a_ID match and S and then if I do then I check to see whether I've also B_ID match from R and S,and then if I do then that's produces the output here .

~~在a_id中我看看是否匹配，我看R(表)中a_id和S(表)中a_id对上了；如果我检查出b_id在R(表)和S(表)中匹配，然后如果是这样就产生输出。（此处翻译不敢保准）~~

我通过 a_id来查找是否有匹配，我看R(表)中a_id和S(表)中a_id有相同，然后我检查出b_id在R(表)和S(表)中对应的是否也匹配，然后如果是，就产生输出

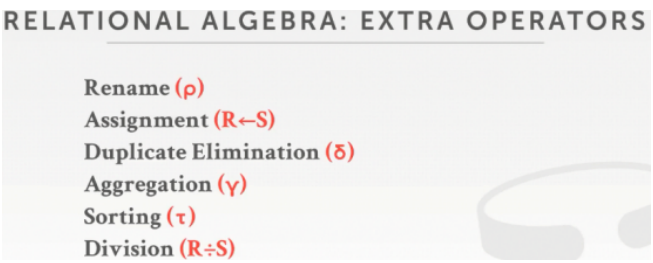
57:10–57:17 虚生花
So again a3–103 exists both an R and S both an S and it can produce the output.
a3 103在R和S中都存在，因此，在这它作为输出结果

```
SELECT * FROM R NATURAL JOIN S;
```

57:17–57:19
So in SQL there's a natural join operator like this
so，在SQL中，这就是NATURAL JOIN操作符

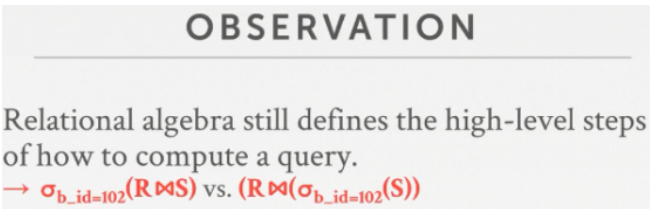
57.19–57.28！！！
again notice I'm not specifying how I want to do the join, it knows that I'm looking to see whether ID matches based on just the name here .
注意，这里我没有明确我想如何进行join，但数据库知道我想看看R中的a_id是否匹配S中的a_id

57:29–57:40
So the reason why this is different than the difference, because there could be additional attributes in R and S that don't they don't have the same name in, you know shared across the two relations .
之所以它和Difference不同的原因是因为在R和S中，它们可能会有一些额外属性，但是这些属性的名字在这两个关系中并不相同
126
57:40–57:42
So they'll they won't actually produced in the output .
so，实际上它们不会出现在输出结果中（知秋注：这里指Difference，估计老师的脑回路跳跃比较快，把Difference省略了）
127
57:42–57:49
So the scheme is allowed to be different as long as there's some common common attributes that you're doing then after joint on
在使用Join时，我们允许这些公共属性的名字可以不同
57.49–57.54
we're in the difference and the difference operator you have it you have to have exactly the same attributes of the same type .
但在DIfference操作符中，我们必须有完全相同的属性以及类型



57:57–58:04
All right so these again, these are just the original main seven relational operators, relational algebra operators that Ted Codd propose
这些运算符是Ted Codd所提出的七个最核心的关系操作符
58.04–58.10
since then, there's been a lot of research and a lot of additional stuff that people have added.
在那之后，许多人为此做了大量研究以及增加了额外的东西

58:10–58:15
Rights, and like there's things like doing sorting order pies where, we know we want to do that SQL,
例如，我们想在SQL中所做的排序
58.13–58.19
the original relational model proposal doesn't talk about these things relational algebra proposal doesn't talk about these things.
原版的关系模型提案中并没有讨论这些东西
130
58:19–58:24
But peanuts since extended that relational algebra included clue these other ones.
但有人将这些东西整合进关系代数，让关系代数变得更加强大
131
58:24–58:29
So doing rename assignment duplicate elimination aggregations of group bys sorting division.
例如，在做重命名作业时，可以通过排序来消除那些名字重复的东西
132
58:29–58:36
So again we may see some of these will see certain see aggregations and sorting later on
我们可能会在之后看到这些操作符中的聚合和排序
58.36–58.48
you know it's just to say that there's placeholder algebra operators that we have may have to consider when we start doing query playing our doing query execution.
当我们开始执行查询时，这里面我们就必须考虑下代数运算符



58:48–58:52
All right, so to finish up I just want to sort of point out one particular thing.
So为了总结下这些内容，我想指出一件事

58:52–59:02
So, the relational algebra is still pretty high-level compared to like our for loops as before
和我们之前的for循环相比，关系代数可能更加高级

59.02–59.08
, because we're not specifying anything of what good to have the data stored in terms of like its data structure and so forth
因为我们并没有明确怎样存储数据是比较好的，或者说它的数据结构该是怎样的等等
59.08–59.12
we're just saying it at a highlevel. Hey you know scan this table and do a filter.
我们只是以高层的角度来告诉数据库系统扫描这张表，然后进行一次过滤

59:12–59:21
Right, but it's still at the end of the day it's still kind of telling you what steps in the order would you should perform these steps .
但它依然告诉你（知秋注：其实就是一些查询优化），你该按照怎样的顺序来执行步骤

59:21––59.26
So again let's say I want to do a join machine R and S
假设我想对R和S进行Join操作
59.26–59.33
and then I want to filter out any tuple where you know filter only produced the tuples where the b_id equals 102
然后，我想找出任何b_id为102的tuple

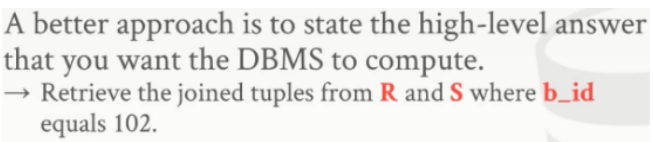
59:34–59:38
So I have two examples of two relational algebra expressions to execute this query.
我有两种关系代数表达式能够来执行这个查询

59:38–59:44
So the first one here I do the natural join of R and S first.
第一种方法是，我先对R和S进行自然连接
140
59:44–59:45
And then I do my filter on B_ID it was 102.
然后我过滤b_id为102的元素
141
59:45–59:52
And then in the second example here I do the filtering on S first where B_ID equals 102 .
然后在第二个例子中，我首先对S进行过滤找出b_id为102的元素

59:52–59:56
And then I think the output of that relation, and then I do the natural join .
然后我再做自然连接
143
59:56–01:00:05
So again you may think at a high–level these are at a high–level these are the same thing they're producing the same result.
你从高层的角度看来看，它们是完全一样的东西，它们生成了相同的结果
144
01:00:05–01:00:11
But the forms characteristics of the how efficient that they will execute these two query plans to be vastly different.

但执行这两种查询方案的效率是完全不同的
145
01:00:11–01:00:16
Right like say I have a billion tuples in R and S .
假设我在R和S中有十亿条数据
146
01:00:15–01:00:23
And there's only one tuple in s where B_ID equals 102.
但在S中，只有一个tuple中的b_id是102
147
01:00:23–01:00:27
So if I do the join first I'm gonna take a join of billion tuples of the billion tuples .
So如果我首先进行Join操作，也就是说对S中的十亿条tuple和R中的十亿条tuple进行Join
148
01:00:27–01:00:32
And then I'm gonna scan through and just find that one tuple where b_id equals 102
然后我对结果进行扫描，只为了找到个b_id为102的tuple
1.00.32–1.00.42
whereas in the second plan I could do the scan on S first find that one tuple where the b_id is 102 .
然而，在第二个方案中，我首先对S进行扫描找到那个b_id为102的tuple

01:00:42–01:00:48
And then now I'm joining that you know with that one tuple with the billion tuples on the R.
然后我用这一条tuple和R中的十亿条tuple进行Join
150
01:00:48–01:01:00
So again, even though we're not going at the low–level like you run this for loop and do that, it's it's still still specifying relation out of the steps we actually want to the day 7 used execute for our query plan.
尽管我们不会使用for循环这种方式来做到这种效果，但我们还是应该明确执行查询方案时我们所做的步骤



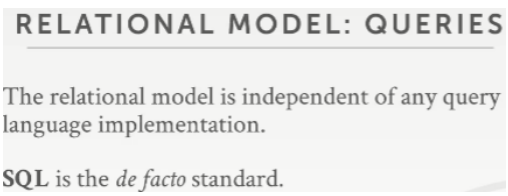
01:01:59–1:01:14.96
So what we really want to do is be able to just say at a high–level tell the database system. Hey this is the answer I want you to compute, and not actually specify how you want it to compute it.
我们真正想做的是以高级的方式来告诉数据库系统我想让你计算出这个答案，而不是指定该以那种方式计算它

01:01:14–01:01:21
So I want to say, hey go just get me a tuple of join from R and S where B_ID goes 102

比如我想从R和S的Join结果中找出b_id为102的那个tuple
1.01.21–1.01.27
, didn't say whether you should scan S first or join join with R first I say, this is the answer that I want.
但我并没有说你该先去扫描S还是先去对R进行Join，我只是想要这个答案

01:01:26–01:01:35
And the reason why we want to do this is again, because now in our application we just specify this high–level answer we want the system to compute.
之所以我们要这么做的原因是，在我们的程序中我们需要以高级的角度来指定我们的系统该如何进行计算
155
01:01:35–01:01:42
And in you know if our table or if our database is really small today.
如果我们现在的表或者数据库非常小
156
01:01:42–01:01:45
And then it grows really big a year from now
但一年之后，我们的数据库就会变得很庞大
1.01.45–1.01.58
I don't have to change out of my code the database could figure out, oh well you had a small team small database before I want to execute it one way, but now you have a billion tuples in S and maybe I want to do the filtering first.
我不需要去修改我的数据库代码就可以搞定（知秋注：只需修改查询方式），在我想以一种方式执行它之前，你有一个小型团队小型数据库，但是现在你的S中有十亿个tuples ，也许我想先进行过滤

01:01:57–01:02:01
But I didn't have to change my application code to make it do that.
但我无须修改我的程序代码（知秋注：指DBMS程序代码）来做到这点
158
01:02:01–01:02:02
Right the day says we could do this for me.
也就是说我可以这么来做就可以（知秋注：修改SQL语句即可）



1.02.04–1.02.11 虚生花
So this is what we're gonna try to do by in running queries and particularly willing to run SQL.
这也就是我们在执行查询，特别是运行SQL时所试图做的事情
1.02.11–1.02.20 虚生花
So the key thing to understand is that although SQL is the standard way people express query run the relational model.
有一个关键点你要去理解，尽管SQL是人们查询关系模型的常规方法
1.02.20–1.02.23 虚生花

It isn't the only way you could do this
这并不是做到这一点的唯一方法
1.02.23–1.02.26
it just happened to be the one that everyone uses and you know standardize on.
它刚好只是每个人都使用的东西，也就是所谓的标准
1.02.26–1.02.31
And actually when Ted.Codd wrote the the relational model paper first in 1970
实际上，当Ted Codd在1970写关系模型论文时
1.02.31–1.02.33
he actually didn't proposed SQL
他实际上并没有提出SQL
1.02.33–1.02.36
he didn't propose any high–level language he just said hey here's a relational algebra .
也并没有提出任何高级语言，他只是说我们有关系代数
1.02.36–1.02.40
He then later proposed his own language called alpha
接着，他推出了他自己的语言，即alpha
1.02.40–1.02.44
that you know it was at a competitor SQL in 1970s
这对于1970年的SQL来说，它是竞争对手
1.02.44–1.02.47
Of course net, you've never heard of it because no one ever used it .
当然你也没听过它，因为也没人用过它
1.02.47–1.02.52
And actually the 1970, there was two other competing there was another competing language
实际上，在1970年代，还有一种竞争语言
1.02.52–1.02.55 英文有误，但意思是这个
let's see well there's other thing Clark well out of Berkeley for this system called ingres .
伯克利大学推出了一种名为Ingres的数据库系统
165
01:02:55–01:03:06 ~~(quele and alpha 没听出来)~~
And they both look similar there quel ,and alpha, and and SQLs sort of look similar to each other but, because the syntax is very different.
它和Quel，Alpha以及SQL看起来都很相似，但语法与众不同
166
01:03:06–01:03:12
um it just so happen again SQL just sort of one out it's what everyone uses today because IBM invented it .
现在每个人都在使用SQL只是因为是IBM发明的它

```
for line in file:
  record = parse(line)
  if "Ice Cube" == record[0]:
    print int(record[1])
```

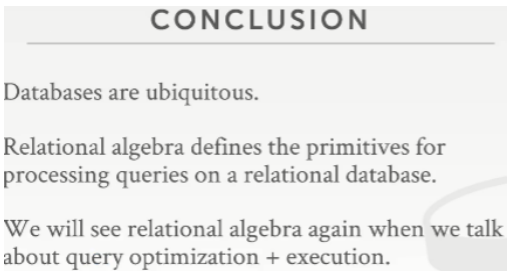
167
01:03:12–01:03:22
So the way you think about this is that again, this is what I showed the very important, this is like the lowest level way you could actually actually this query of

finding all the the, you know finding one Ice Cube went solo .
我所展示的这个例子非常重要，实际上这是你用来查询Ice Cube所有单飞专辑的最底层写法

```
SELECT year FROM artists
 WHERE name = "Ice Cube“;
```

168
01:03:21–01:03:28
But instead I could just write a really simple SQL query like this and say, this is the answer that I want you to produce.
但相反，我可以写这么一条很简单的SQL查询来生成你想要的答案

169
01:03:28–01:03:36
And then the data center could then figure out, oh I I want to write it in this particular for loop you know cuz that's the best way to do this.
然后数据中心就表示我想以这种For循环来做到这点，因为这是做到这点的最佳方式
170
01:03:36–01:03:41 虚生花
And then if I add indexes then the query plan could change right.
然后，如果我添加了索引，那么查询方案也会改变
171
01:03:41–01:03:46 虚生花
So the beauty of the relational model and SQL is that we can write these high-level queries that we want .
关系模型以及SQL的美在于我们能以我们想要的方式来编写这些高级查询
172
01:03:46–01:03:56
And then over time as the database changes as the database system self changes or our workload changes, it can adapt and improve itself about every us having to go back or change our application.
随着时间的推移，数据库也会产生改变，它能不断适应并改进，就像我们必须不断修改我们的程序代码那样
1.03.57–1.04.02
Against for this reason is why I'm super excited about the relational model, and I think it's always the right way to go.
这就是我为什么对关系模型感到兴奋的原因，我觉得使用它是一种正确的选择



1.04.02–1.04.09
All right, so just to finish up here and again I realize it's it's all for me sitting in the bathtub giving me this lecture .
好了，今天就到这里，我想我在浴缸里坐着讲课的时间够长了

1.04.09–1.04.12
So I patient you guys sitting through this.
我为了给你们讲课不得不坐在浴缸里
1.04.12–1.04.17
So databases are ubiquitous, databases are stupid important, databases are everywhere
因此，数据库无处不在，数据库十分重要
1.04.17–1.04.29
so no matter what you do throughout your life even if you don't go, you know you know even, you know going the theater databases, you know in the field of computer science of Cuba programming, I guarantee you throughout the rest your life you're gonna come across databases .
所以无论你一生中做什么，只要你和计算机科学或者编程打交道，那么我保证你的一生中都会和数据库打交道
1.04.29–1.04.34 虚生花
And it's gonna be important for you to understand how they actually work on the inside .
对你们来说，理解数据库内部的工作原理是很重要的
1.04.34–1.04.43 虚生花
Right, because when things don't you know things aren't working the way they should be think they'll be working you need to know, oh it's written this way this is what is actually doing so that you can figure out how to actually improve things.
如果你知道数据库的内部工作原理，你就知道该如何对性能进行调优和提升
1.04.43–1.04.47
So the we discussed the relational model we discussed relational algebra .
我们讨论了关系模型、关系代数

1.04.47–1.04.58
And we showed how we these are the primitives we can use to essentially use them, as the building blocks to generate queries that we can execute on a relational database to update it and drive answers.
接着，我们展示了这些基本操作符的使用，使用它们来生成查询语句，我们可以在一个关系型数据库上执行并得到答案
1.04.58–1.05.05
So we'll see you later on algebra again later in the semester when we talk about query optimization query execution .
在这学期稍微晚点的时间，当我们讨论查询优化和执行查询时我们会用到代数
1.05.05–1.05.17
But just in the back of your mind as you write SQL queries for the first homework you should be thinking about, oh well how would you know, the database is translating these into relational algebra, so everything about how is actually doing it how would actually execute these things .
但当你在写第一个关于SQL查询的作业时，你应该思考下数据库是如何将SQL转换为关系代数以及这些事情是如何执行的
1.05.17–1.05.22
Okay I almost forgot one last thing ,
~~好的，我差点了最后一件事是重要的事情~~
Oh，我差点忘了最后一件事

1.05.22–1.05.27

the most important thing You need to understand about databases throughout this through the rest your life is the following
接下来这件事是你余生中理解数据库时最重要的一件事

1.05.27–1.05.32
when you look back at the 36 chambers understand who were the original nine involved in it.
当你回看36房（知秋注：一个专辑）时，你要知道它里面一开始的9名成员
1.05.32–1.05.46
You have the RZA, the GZA, inspector deck,Ghostface Killah,masta killa， U–God, Method Man, olDirty Bastard Raekwon.
这里面有RZA，GZA，Inspectah Deck，Masta Killa，GhostfaceKillah， Method Man，U–God， Raekwon以及 Ol' Dirty Bastard
1.05.42–1.05.47
But the other important thing too is Cappadonna was in jail at the time
但另一点重要的事是Cappadonna那时候入狱了
1.05.47–1.05.50
so he was actually original member of the clan
他是武当派的初始成员
1.05.50–1.05.53
but because he was in jail he couldn't he couldn't be on the 36 chambers.
但因为他入狱了，所以他没办法参与武当36房

1.05.53–1.05.57
So that's the most important thing you need to understand throughout this entire semester.
因此，这是整个学期中你需要了解的最重要的东西
1.05.57–1.06.00
okay all right guys see you next time.
好了 同学们，下次见。