**2025-3-4 COMPSCI751**

**英文原文 & 中文翻译**

Paragraph 1 (English):

Um, so this semester, I’m going to team up with another lecturer and two teachers to deliver the database course to you guys, and first of all, I’m going to introduce the topics that we’re going to cover in this semester.

段落 1（中文）：

嗯，这个学期，我会和另一位讲师以及两位助教合作，为大家教授数据库课程。首先，我会介绍一下我们在这个学期里要覆盖的主题。

Paragraph 2 (English):

So this week, we are going to talk about the relational model and the relational algebra. Relational model is the foundation of the relational database, where you describe the data in relational model, while relational algebra is technically not complicated algebra—it’s just the theoretical foundation of the current language SQL (Structured Query Language). By introducing the relational algebra, we will be more familiar with how to query. We will be allowed more possibilities with the SQL query.

段落 2（中文）：

因此，这周我们要讨论关系模型和关系代数。关系模型是关系数据库的基础，你可以在关系模型中描述数据，而关系代数在技术层面并不算复杂的代数——它只是当前语言 SQL（结构化查询语言）的理论基础。通过介绍关系代数，我们会更加熟悉如何进行查询，并在 SQL 查询中拥有更多可能性。

Paragraph 3 (English):

And in the second and third weeks, we are going to talk about SQL querying depth, from the easiest version in the beginning to advanced use of SQL. This part will be very important to us because it will be extensively used in various applications.

段落 3（中文）：

在第二周和第三周，我们将深入探讨 SQL 查询，从最简单的版本开始，到更高级的 SQL 用法。这个部分对我们来说非常重要，因为它将在各种应用中被广泛使用。

Paragraph 4 (English):

So, after that, we want to develop a little bit deeper into the storage of the data. Storage and indexing is not that easy for interfaces, as we will see. Today we’re facing a lot of realistic challenges, but we’d like to create an illusionary, persistent, and large-scale storage for the user, and it’s not easy.

段落 4（中文）：

之后，我们想更深入地探讨数据的存储。正如我们将看到的，存储和索引并不那么容易实现。如今我们面临很多实际的挑战，但我们想为用户打造一个表面上持久且大规模的存储，这并不是一件容易的事。

Paragraph 5 (English):

After that, we’re going to introduce the E-R (Entity-Relationship) model and the design and database normalization. So this part will convert an application description, such as the specifications, and then we will model it with the entity-relationship model, in which we use an entity to describe objects, and relationships to describe their internal relations. It’s quite a logical model so that we can convert this model later onto the relational model that we are teaching today.

段落 5（中文）：

接下来，我们会介绍 E-R（实体-关系）模型以及数据库设计与规范化。这个部分将把应用描述（比如规格说明）转换过来，然后我们会用实体-关系模型对其建模，在这里我们用实体来描述对象，用关系来描述它们之间的内部关联。这是一个相当逻辑化的模型，以便我们之后能将它转换为今天所讲的关系模型。

Paragraph 6 (English):

And then, with this design of the database, it’s not necessarily optimal. So we would like to optimize it with some kind of rules. That’s why we are going to introduce normalization. And then, after that, we’re going to come to a little bit of implementation details about query processing and query optimization. So, for every single SQL query, the users don’t have to know how the query is executed at the backend, but as you probably need to optimize the whole performance of the application, then you need to know how the query is executed.

段落 6（中文）：

然后，对于这种数据库设计来说，并不一定已经是最优的。因此，我们会想用一些规则来对它进行优化，这就是我们要介绍数据库规范化的原因。接着，我们还会谈到一些关于查询处理和查询优化的实现细节。对于每一个 SQL 查询，用户不需要了解它在后台是如何执行的，但如果你需要优化整个应用程序的性能，那么你就需要知道查询是如何被执行的。

Paragraph 7 (English):

So eventually, we’re going to reach a topic called transaction. That’s a very important topic in the way that you think about: if you’re going to transfer money from one bank account to another, and you’re going to reduce, let’s say, one hundred dollars from one account and permit one hundred dollars to another account, these two operations need to be done either entirely or none. You couldn’t do it halfway because otherwise, you know, people will complain it’s a critical error. So this kind of logically atomic operation needs to be done in a concurrent situation because a lot of users are using the same system at the same time. So that comes to the transaction concept, which is the last topic of the semester.

段落 7（中文）：

最后，我们会谈到一个名为“事务”（transaction）的主题。这是一个非常重要的话题，比如，当你要把钱从一个银行账户转到另一个账户时，你会减少一个账户的一百美元，并给另一个账户增加一百美元，这两个操作必须要么全部执行，要么全部不执行。你不能只完成一半，因为否则就会出现严重错误，人们会抱怨。这种在逻辑上不可分割的操作需要在并发的环境中进行，因为很多用户同时在使用同一个系统。这也就是事务这个概念，它是本学期的最后一个主题。

Paragraph 8 (English):

So, in terms of the topics, it’s like this.

段落 8（中文）：

关于这些主题，大致就是这样。

Paragraph 9 (English):

Before we introduce the relational model and the relational algebra, I will want to talk about the team that we are having for this course. So I’m going to teach the course’s first six weeks, and my office is in 303, Science Building 524, and my office hour is Wednesday, 2:30 PM to 3:30 PM. You are more than welcome to join in person or join in via Zoom if you have a question to ask me. And after six weeks, there will be Jerry Weber’s part. So he’s going to teach you for six weeks after me.

段落 9（中文）：

在介绍关系模型和关系代数之前，我想先谈一下本课程的团队安排。我会教授课程的前六周，我的办公室在科学楼 524 的 303 室，我的办公时间是每周三下午 2:30 到 3:30。如果你有问题想问我，可以来现场或通过 Zoom 参加。六周之后，就会轮到 Jerry Weber 来教你们接下来的六周。

Paragraph 10 (English):

We will have three tutors:

• Yi Zhou Dai,

• Hou Yu Li,

• and Sizhuo Zhang.

They are going to deliver tutorials that mainly serve the purpose of deepening the concepts that we introduce in the course or introducing some kind of exercises or explanations so that you will have a deeper hands-on experience on the concepts we’re using in the course.

段落 10（中文）：

我们将有三位助教：

• Yi Zhou Dai，

• Hou Yu Li，

• 以及 Sizhuo Zhang。

他们将负责带领辅导课，主要是为了加深我们在课程中引入的概念，或者提供一些练习和讲解，让你们能对课程中使用的概念有更深入的实践体验。

Paragraph 11 (English):

Any question on this?

段落 11（中文）：

在这方面有什么问题吗？

Paragraph 12 (English):

In terms of assessment, the course will have three assignments of a total of thirty-five percent of the final score, and two projects, fifteen percent in total. And the term test will be twenty percent, which will be arranged on Friday of the seventh week. And the final exam is thirty percent. But the term test will exclusively cover the first six weeks. So, the thing is, the first six weeks will be exclusively examined in the term test. In other words, you will not need to prepare for the final exam for the first six weeks. So the term test is the “final exam” for the first six weeks’ content.

段落 12（中文）：

关于考核方面，本课程将有三次作业，总共占期末成绩的 35%，还有两个项目，总共占 15%。期中测验（term test）占 20%，会安排在第七周的星期五。期末考试占 30%。但是期中测验只考前六周的内容，也就是说，前六周的内容只会在期中测验中出现，你不需要在期末考试中准备这部分内容。因此，期中测验就相当于前六周的“期末考试”。

Paragraph 13 (English):

And so, the late policy for the assignments and projects is the same. Every assignment or project will be due on Friday night, 11:59 PM, and then you will have almost two days of late submission with Canvas and penalty each day. That’s automatically done by Canvas with your files. And after two days, the submission window will be closed, and we will release the model answer after that.

段落 13（中文）：

另外，作业和项目都有相同的迟交政策。每次作业或项目的截止时间都是周五晚上 11:59，在此之后你还有接近两天的迟交时间，Canvas 会自动计算你每天的迟交罚分。两天之后，提交窗口就会关闭，然后我们会发布参考答案。

Paragraph 14 (English):

So note that you must obtain a pass in both the theory and practice. The practical part includes the projects and assignments, and the theory part includes the test and final exam, to pass the whole course. So pass is like fifty percent, right? I think every year the pass bar is a little bit different, but the whole setup is to let you know that both the theory and practical parts will be counted, and they will be counted separately.

段落 14（中文）：

需要注意的是，你必须在理论和实践两部分都达到及格线才能通过整门课程。实践部分包括项目和作业，理论部分包括测验和期末考试。及格分通常在 50% 左右吧？我想每年会有些许不同，但总的来说就是让你们知道，理论和实践这两部分都会被计算，并且是分开计算的。

Paragraph 15 (English):

So, the expectation of this course is that this is a standard 15-point course, and you are expected to spend ten hours on this course per week. So, apart from the three hours of contact in the class and one hour of tutorial, you are expected to spend five to six hours on assignments or reading books or practicing by yourself, to get yourself ready for the assignments or projects. So, we’re not imposing a compulsory attendance requirement, but you are highly encouraged to attend most of the lectures and tutorials. And if you unfortunately get sick, feel free to stay at home and watch the recordings online. And if you have any questions, you can just drop me a message.

段落 15（中文）：

本课程的预期是，这是一门标准的 15 学分课程，你每周应投入大约 10 个小时在这门课上。除了课堂上 3 小时和辅导课 1 小时的接触时间之外，你还需要花 5 到 6 小时做作业、看书或者自行练习，以便为作业和项目做好准备。我们并没有强制考勤的要求，但仍强烈建议你参加大部分的讲座和辅导课。如果你不幸生病，可以安心在家并在网上观看录播。如果有任何问题，你可以随时联系我。

Paragraph 16 (English):

In terms of the resources, all the lecture slides and recordings will be released on Canvas. And here, I will mainly talk about the reference textbooks. The first textbook is called “Database System Concepts.” So this is the most prevalent textbook on databases, and currently, it is in its seventh edition. So please feel free to search for resources online; there are plenty of them. And if you would like to buy a hard copy, you can get a twenty percent discount. I have nothing to do with this—just when I was contacting the publisher for teaching materials, they offered this option.

段落 16（中文）：

关于学习资源，所有的课件和录播都会在 Canvas 上发布。这里，我主要想谈一下参考教材。第一本教材叫做《Database System Concepts》。这是数据库领域使用最广泛的教材，目前已经是第七版。如果你想找线上资源的话，可以放心去搜，网上有很多。要是你想买纸质书，还能享受 20% 的折扣。这和我没有任何个人关系，只是我在和出版社联系教学材料时，对方给出的一个选项。

Paragraph 17 (English):

Another reference book—so this is recommended; you are not required to own a copy—is “Designing Data-Intensive Applications.” So this is quite nicely written for insights on how to design data-intensive applications with all possible database options. I mean, apart from the relational database we are going to introduce in this course, we are going to have a lot of NoSQL options like column store, key-value store, document database, so forth. So we have a wide range of options based on the requirement of your application. It is critical for you to get used to these solutions properly, to organize them in a nice way to support your own application. So this book is about how you want to get it done—so how do you mediate this.

段落 17（中文）：

另外还有一本参考书（推荐但并不强制你拥有实体书）叫做《Designing Data-Intensive Applications》。这本书写得相当不错，能让你理解如何使用各种可能的数据库选项来设计数据密集型应用。我的意思是，除了我们在本课程会介绍的关系数据库之外，我们还会接触很多 NoSQL 选项，比如列式存储、键值存储、文档数据库等等。根据你的应用需求，你有很多选择。正确地熟悉并运用这些解决方案，并以恰当的方式来支持你的应用，这是至关重要的。这本书就是教你怎样做好这些事情——也就是如何在其中进行权衡。

Paragraph 18 (English):

So, as a routine, we are required to select class representatives. The role of the class representative will be to convey your opinions to me or to Jerry, the teaching team. So, the class representative will be attending two staff-student meetings, and if you are interested in taking this role, please come to me after the class.

段落 18（中文）：

按照惯例，我们需要选出班级代表。班级代表的职责是将你们的意见反馈给我或 Jerry 以及整个教学团队。班级代表需要参加两次师生会议，如果你对这个角色感兴趣，请在下课后来找我。

Paragraph 19 (English):

Okay, now let’s move to the database application examples. This part is just to show you how you are so frequently interacting with databases, but you may not be aware of it. So imagine in the morning, you wake up and you check the email system. The email system is a database. After that, you may take a bus to the university. You hop on the bus, tap on, tap off—that is a database system as well. And then, when you, let’s say, go into the classroom, log into Canvas. Canvas is a database—it creates different views for different users. For example, my view will be different from yours, and Canvas loads your personal information, like your user ID and classes you’ve taken or enrolled in, and organizes a specific view exclusively for you. That is using databases.

段落 19（中文）：

好了，现在让我们来看一些数据库应用的例子。这个部分是为了让你们知道自己其实经常在和数据库打交道，但可能没意识到。举个例子，早上醒来你去查看邮件系统，邮件系统本身就是一个数据库。然后，你可能乘公交来学校，刷卡上车和下车——那也是一个数据库系统。接着，比如你进教室，登录 Canvas。Canvas 也是一个数据库——它会为不同的用户创建不同的视图。比如我的视图和你们的是不一样的，Canvas 会加载你的个人信息，比如用户名、你选过或正在上的课程，然后为你生成专属的视图。这就是在使用数据库。

Paragraph 20 (English):

And during lunchtime, you go to a restaurant and pay for your lunch, and this kind of credit card transaction is using databases. So, if you think about it, you are just using databases all the time. If you are checking, let’s say, all kinds of apps, most of the apps’ backends are supported by database systems. So, this is just to give you an idea of how prevalent databases are in our daily life.

段落 20（中文）：

到了午餐时间，你去餐厅买午餐，刷信用卡付款，这种信用卡交易也是在使用数据库。如果你想想看，你其实一直都在使用数据库。比如你查看各种手机应用，大部分应用的后端都由数据库系统提供支持。这只是想让你了解数据库在我们日常生活中是多么普遍。

Paragraph 21 (English):

With the examples here, for the enterprise, we will have sales, customers, products, and purchases using a database system; an accounting system recording payments, receipts, assets; a human resource system recording information about employees, salaries, payroll, taxes, and so on. For manufacturing, you will have management of production, inventory, orders, supply chain. I mean, if you just want to enumerate, you have so many things backended or supported by the database. For example, the navigation system like Google Maps—it’s also a database system. It not only stores the map but also stores real-time usage or user data to provide online analysis for you so that you will have a real-time estimation of travel time before you reach your destination. This is also a database. Any questions regarding this?

段落 21（中文）：

再举些例子，比如在企业环境中，我们会用数据库系统来管理销售、客户、产品和采购信息；会用会计系统来记录付款、收据、资产；会用人力资源系统来记录员工、薪资、工资单、税务等等。对于制造业，则会用数据库来管理生产、库存、订单、供应链。如果你要一一列举，其实有太多东西都是由数据库在后台提供或支持的。比如像谷歌地图这样的导航系统——它同样是一个数据库系统。它不仅储存地图信息，还储存实时的使用数据或用户数据，为你进行在线分析，让你在到达目的地之前能实时预估行程时间。这也是一个数据库。关于这方面，你们有什么问题吗？

Paragraph 22 (English):

Okay, so given that we are so frequently generating data and we are so frequently using database systems, what is the purpose of the database system? So here, we list “massive system,” but you may not be sure what I mean by these words. By “massive,” we mean if your amount of data is very small, it’s not that necessary for you to use a database; you can just store it in a file. A database is only useful when you have a massive amount of data to manage and you would like to retrieve the data efficiently. You would like to use it in a scenario where you would like to support multi-user usage. “Multi-user” means a lot of users are using it concurrently, so this multi-user requirement will correspond to concurrency control in the database.

段落 22（中文）：

好的，那么既然我们如此频繁地生成数据并使用数据库系统，数据库系统的目的是什么呢？这里提到“massive system”（大规模系统），但你们可能不太清楚这是什么意思。所谓“大规模”，指的是如果你的数据量非常小，那其实没必要用数据库；你可以直接把它存到一个文件里。只有当你需要管理非常海量的数据并想高效检索这些数据时，数据库才有用。而且你还会想在一个多用户的场景下使用它。“多用户”意味着大量用户同时在使用这个系统，这就需要数据库的并发控制来满足需求。

Paragraph 23 (English):

And “persistent” means that your data outlives your database system. Sometimes, the most valuable stuff is the data, not the database system. I mean, data is key—just to manage the data. Sometimes, the company may decide to migrate the database system from one system to another, so you have to shift the data from an older or a legacy system to a new system. So the data itself is outliving the system. And at the same time, you would like the data to be stored in such a way that no matter what happens—including flood, typhoon, or a power outage, or any unexpected situation—you would like the data to be safe. You don’t want to lose your photos; you don’t want to lose your money in the bank, right? So all the data needs to be safe.

段落 23（中文）：

“持久性”（persistent）意味着数据的寿命比数据库系统更长。有时最有价值的东西是数据，而不是数据库系统本身。我的意思是，数据才是关键——我们只是想管理数据。有时公司可能会决定把数据库系统从一个系统迁移到另一个系统，于是你就得把数据从旧系统或遗留系统转移到新系统上。因此，数据本身会比系统的生命周期还长。同时，你还希望这些数据能被以一种安全的方式储存，这样无论发生什么——比如洪水、台风或停电，或者其他意外情况——数据都能保持安全。你肯定不想丢失照片，也不想丢失银行里的钱，对吧？所以所有的数据都需要是安全的。

Paragraph 24 (English):

For “convenience,” we mean that, for the user of the database, they don’t want to know how you store and retrieve it. So, in other words, there’s a separation of the physical storage of data from the user. All I want to know is that I want to get this data out of the database, and then I would just tell the database “I want this data,” but I don’t care how you get this data for me. It’s like you have an API between the user and the database, so the user doesn’t have to know the details of the implementation. It could be possible that one day it’s implemented in one way, another day in a completely different way, but the user doesn’t have to know about it.

段落 24（中文）：

至于“方便性”（convenience），我们的意思是，对于数据库的用户来说，他们并不想知道你是如何存储和检索数据的。换句话说，就是把数据的物理存储与用户隔离。用户只要告诉数据库“我要这个数据”，然后数据库把数据给他就行了，用户并不关心数据库如何取到这些数据。就好比在用户和数据库之间提供了一个 API，所以用户不需要了解底层实现的细节。也许某一天你以一种方式来实现，另一天你换成完全不同的方式，但对于用户来说并没有影响。

Paragraph 25 (English):

And for the efficiency—this is the most critical part—it’s like we buy houses all about location, location, location; databases are all about efficiency. It’s because users are quite intolerant of long delays in query processing. Which means, for example, when you open your bank account management app and you would like to see your bank balance within 30 seconds—if it keeps you waiting for, let’s say, two minutes or three minutes, then you will lose your patience, and that’s pretty bad. You will have a bad experience. That’s why companies like Amazon, when you do checkout—if you have a large cart, a lot of stuff to check out—once you click checkout, there must be a period of time you wait. If this time is too long, Amazon guarantees that if you are waiting for too long, you can ask for compensation. This is kind of a quality of service guarantee. So this is backended or supported by efficient database implementations to ensure every such transaction can be done within a reasonable amount of time.

段落 25（中文）：

至于“效率”，这是最关键的部分，就好比买房看地段、地段、地段，数据库则在乎效率、效率、效率。原因是用户对查询处理中的长时间延迟非常不耐烦。举个例子，当你打开银行账户管理的应用时，你希望能在 30 秒内看到自己的余额；如果让你等两三分钟，你就会失去耐心，这体验可就太糟糕了。正因为如此，像亚马逊这样的公司，在你结账时，如果你的购物车里东西很多，一旦点击“结账”，会有一段等待时间。如果这个时间太长，亚马逊保证如果你等太久，你可以要求补偿。这是一种服务质量保证。能做到这一点，背后都是由高效的数据库实现来支持的，以确保每笔这样的交易都能在合理的时间内完成。

Paragraph 26 (English):

I mean, “reliable” is the last one but not the least one, because people are so intolerant of unreliable databases. Basically, what we require for reliability is 99.999% or 99.9999%. Which means we could barely accept, for example, losing money in the bank, or if somebody, due to another person’s transaction, somehow deducted your bank account balance—that’s completely unacceptable. You would be super angry and unhappy about that. So this is about reliability. Any question about this?

段落 26（中文）：

我还想说，“可靠性”是最后提到但绝不意味着不重要，因为人们对不可靠的数据库是完全无法容忍的。一般来说，我们对可靠性的要求会达到 99.999% 或者 99.9999% 这样的级别。这意味着我们几乎不能接受任何类似“银行里的钱莫名其妙丢了”或者“因为别人的交易导致你的账户余额被扣除”这种事。这种情况完全不可接受，你肯定会非常生气和不满。所以这就是可靠性的问题。关于这点有什么疑问吗？

Paragraph 27 (English):

Okay.

段落 27（中文）：

好的。

Paragraph 28 (English):

So, during the course, we will use a running example as a toy example. Let’s use this university database. It consists of objects or entities like students, instructors, classes, and some relations between these entities, like departments. Application programs could be “add new students to the system” or “add new instructors” or “add new courses.” You could register students for courses and generate class rosters or assign grades to students, compute GPAs, or generate transcripts. So these are possible application scenarios of this toy scenario.

段落 28（中文）：

在接下来的课程中，我们会使用一个贯穿始终的示例当作玩具例子。让我们来看一个大学数据库的例子。它由一些对象或实体组成，比如学生、教师、课程，以及它们之间的一些关系，比如院系。可能的应用程序包括“向系统中添加新学生”“添加新教师”或者“添加新课程”。你也可以为学生注册课程并生成课程名单，或者给学生评分，计算 GPA，或生成成绩单。这些都是这个玩具示例中可能出现的应用场景。

Paragraph 29 (English):

Okay, so now we are going to talk about data models. So, it’s like databases are about data, and we would like to model the data in a decent way. By observing the data, different data models may be derived. They describe what the data is, what the relations among the data are, what the semantics of the data are, and what the constraints of the data are. For example, in the university database case, the entities could be the department, the students, the instructors; the relations could be “the instructors deliver courses” or “give classes,” “the student takes classes,” and so on. Data constraints could be, for example, “every department should have at least five classes to offer,” or “every department should have at least one instructor.” These kinds of constraints may be specified by the application to describe the data.

段落 29（中文）：

好了，现在我们要讨论数据模型。数据库就是围绕数据展开的，我们希望能以合适的方式来对数据进行建模。通过观察数据，可以衍生出不同的数据模型。它们描述了数据是什么、数据之间的关系是什么、数据的语义是什么、以及数据有哪些约束。比如在大学数据库的例子中，实体可以是院系、学生、教师；关系可以是“教师开设课程”或者“学生修读课程”之类的。数据约束可能包括“每个院系至少要提供五门课程”或“每个院系至少要有一位教师”等。这些约束可以由应用来指定，以描述数据。

Paragraph 30 (English):

In this sense, we have a number of possible data models. Apart from the relational model, which we will talk about in the next slide—it’s all about the relation—we will have the E-R (entity-relationship) model that will be introduced in the sixth week. So, this model is not an ultimate artifact. This E-R model is generally used as a pathway from the application to the relational model, and by the relational model, you will have a database design immediately. So this entity-relationship model just helps you walk from the application to the database schema for the relational model.

段落 30（中文）：

在这种情况下，我们就有多种可能的数据模型。除了接下来我们要讲的关系模型（它就是围绕关系展开的），我们还会在第六周介绍 E-R（实体-关系）模型。不过，E-R 模型并不是最终的产物。它通常被用来作为从应用到关系模型之间的过渡，通过关系模型你就能直接获得数据库设计。所以，实体-关系模型只是帮助你从应用过渡到关系模型数据库架构的一个过程。

Paragraph 31 (English):

And the graph model applies to data like Facebook, many-to-many relationships. For example, in Facebook, if you think about it, the data could describe people—every person will be a self-contained page—and then between each pair of people, there might be friendships or relationships. This could be better described with a graph because every node is a person, and the edge is a relation. So you can attach the information of a person to the node and attach the interactions on the edges. This is better for you to store and do graph data queries.

段落 31（中文）：

图模型（graph model）适用于类似 Facebook 这样有大量多对多关系的数据。比如在 Facebook 中，如果你想一想，数据可以描述用户——每个人都有一个独立的页面——然后在每两个人之间可能存在好友关系或者其他关系。用图来描述这个就更合适，因为每个节点代表一个人，每条边代表一种关系。这样你就可以把用户的信息附着在节点上，把交互信息附着在边上。这种方式更适合你存储和执行图数据查询。

Paragraph 32 (English):

The document data model is essentially a tree-shaped data model, so think of a file like XML or JSON. It has only one root, and the root will have multiple children, and each child will have multiple children as well. So that’s the XML or JSON model. This document model is more suitable for one-to-many relationships.

段落 32（中文）：

文档数据模型（document data model）本质上是一种树形数据模型，你可以联想到 XML 或 JSON 这样的文件。它只有一个根节点，根节点可以有多个子节点，每个子节点也可以有多个子节点。这就是 XML 或 JSON 的模型。文档模型更适用于一对多的关系。

Paragraph 33 (English):

And there are other simplified data models like key-value store. In a key-value store, the data is modeled with pairs, where each pair has only one key (which is a unique key) and then a value, and the value can be huge. For example, the email system can be stored in a key-value store, where the key is the user ID, and all the emails can be compiled into a huge file and stored on a disk, or so. This key-value store has been widely used by Google—its whole Google storage system has been fundamentally constructed by layers and layers of key-value store abstractions.

段落 33（中文）：

另外还有一些更简化的数据模型，比如键值存储（key-value store）。在键值存储中，数据以键-值对的形式建模，每个键-值对只有一个键（这是一个唯一的键），然后是对应的值，值可以很大。举例来说，邮件系统就可以存储在键值数据库里，键就是用户 ID，所有邮件都可以汇总成一个巨大的文件并存储在磁盘上之类的。键值存储也被谷歌广泛使用——它整个谷歌存储系统在底层就是由一层又一层的键值存储抽象构建起来的。

Paragraph 34 (English):

So, talking about this is to note that apart from the relational model, we have other options. But in terms of commercial databases, most of the current commercial databases are still using the relational model. It is simply because the relational model provides a lot of functionalities that the other models aren’t sufficiently providing nowadays, and another reason is once you have chosen one model, it is very hard for you to change to another model. So, basically, you don’t want any error or problem to occur when users are using the application, which makes you be very cautious in deciding to shift your fundamental data model from one to another.

段落 34（中文）：

之所以要谈到这些，是为了让大家知道，除了关系模型之外，我们还有其他选择。不过，就商业数据库而言，目前大多数主流的商业数据库依然采用关系模型。这主要是因为关系模型提供了很多其他模型目前还不够完善的功能，另一个原因是一旦你选定了一种模型，想要再换成别的模型就会非常困难。毕竟，你不想在用户使用应用时出现任何错误或问题，这就使得在决定转换基础数据模型时必须非常谨慎。

Paragraph 35 (English):

Okay, so now we’re going to talk about the relational model. The relational model describes the data using a construction of relations. In this model, the data is just a collection of relations, where each relation is a table. So, in this example, a university database will have two relations, and each relation will have a name. So the first relation is called “instructor,” and the second relation is called “department.” And each relation is described by a table, and for each relation, it will have a schema. By schema, we mean a set of attributes.

段落 35（中文）：

好了，现在我们正式进入关系模型的讨论。关系模型使用“关系”的结构来描述数据。在这种模型里，数据就是一组关系，每个关系对应一张表。比如，在这个大学数据库的例子里，我们会有两个关系，每个关系都有一个名字。第一个关系叫 “instructor”，第二个叫 “department”。每个关系都通过一张表来描述，而每个关系都有一个“模式”（schema）。所谓的模式，就是一组属性。

Paragraph 36 (English):

So here, the schema of “instructor” includes four attributes: ID, name, department\_name, and salary. And for each attribute, you will have a domain. So the domain can be “string,” can be “integer,” or can be more specific, like people’s age can never be negative and never go over one hundred and fifty. So you can specify the age to be an integer in between zero and one hundred fifty—that is your domain. So, in this case, ID, name, and department\_name are strings, and salary is an integer.

段落 36（中文）：

在这里，“instructor”这个关系的模式包含四个属性：ID、name、department\_name 和 salary。每个属性都有一个域（domain）。这个域可以是“字符串”，也可以是“整数”，或者更加具体，比如人的年龄不可能是负数，也不可能超过 150，所以你可以将年龄限定在 0 到 150 之间的整数作为它的域。在这个例子里，ID、name 和 department\_name 都是字符串，salary 是整数。

Paragraph 37 (English):

Every attribute is corresponding to a column of this table, so the values of this attribute will be in the column, whereas each row of this table describes the instances. For example, we record a row or a tuple—I mean, “tuple” is more of a database term—so the tuple of “Einstein” includes the assignment of all these attributes, like his ID, his name, department\_name, and salary. So, rows represent tuples; columns represent the attributes; and that’s a table and a relation.

段落 37（中文）：

每个属性对应这张表中的一列，这个属性的值就储存在该列当中，而这张表的每一行则描述一个实例。举个例子，我们把 “Einstein” 这行数据称为一个元组（tuple 是更偏数据库领域的术语），这个元组包括所有属性的取值，比如爱因斯坦的 ID、姓名、院系名和薪水。因此，行就代表了元组，列代表属性，这就是一张表，也是一个关系。

Paragraph 38 (English):

So, it seems like at some point in time, we may take a snapshot of these relations and record it as a database instance. The snapshot (like the content of these relations at some point in time) is called the instance. The thing is, this instance, especially these tuples, can change dramatically over time, because new instructors may get in, old instructors may retire, but the schema rarely changes. Once it’s set, it’s rarely changed, because changing the schema could be very costly and can lose information, or it’s quite a big change to hold the analysis. Any questions?

段落 38（中文）：

也就是说，在某个时刻，我们可能会对这些关系进行一次快照并将其记录为一个数据库实例。这个快照（比如在某个时刻这些关系所包含的内容）就称为该数据库实例。问题在于，这个实例尤其是这些元组，可能会随着时间而剧烈变化，因为新教师可能加入，老教师可能退休，但模式却很少改变。一旦确定了模式，就很少会变动，因为更改模式的代价很高，可能会导致信息丢失，或者是一次非常大的变动，会影响到后续的分析。有任何问题吗？

Paragraph 39 (English):

So, talking about the attributes, apart from the domain that we talked about, the attribute could have atomic types, something similar to a programming language variable—maybe integer, maybe string. It could have a structured type, meaning a composite type of different fields or different subtypes getting together. And there is a special value I need to talk about now—this “null” means “I don’t know the value of this field.” It appears in every single domain. It’s a shared value of every single domain.

段落 39（中文）：

回到属性本身，除了我们所说的域，这些属性可以具有原子类型（atomic types），类似于编程语言里的变量类型——比如整型或字符串。它们也可以是结构化类型，这意味着它可以由不同的字段或不同的子类型组合而成。这里有一个需要特别提到的特殊值——“null”，表示“我不知道这个字段的值”。它可以出现在任何一个域中，是所有域的一个共享值。

Paragraph 40 (English):

So, in this example, if we don’t know the salary of Einstein, we could put null here. And this null can incur a lot of complications for the database. For example, if we would like to know who among the instructors has a salary over ninety thousand, let me just pose a query: “Give me all the instructors with a salary greater than ninety thousand.” We can also ask a query, “Give me all the instructors whose salary is smaller or equal to ninety thousand.” These conditions are complementary, which means if we impose these two questions, we should have the entire table. But it’s not the case: Einstein will not be returned in either query because we don’t know his salary. So this kind of tricky scenario will cause a lot of problems in data processing for query evaluation.

段落 40（中文）：

在这个例子里，如果我们不知道爱因斯坦的薪资，就可以在该处填上 null。而 null 会给数据库带来很多复杂性。比如，如果我们想知道哪些教师的薪资超过 9 万美元，可以提出一个查询：“给我所有薪资大于 9 万美元的教师”。我们也可以问：“给我所有薪资小于或等于 9 万美元的教师”。从逻辑上说，这两个条件是互补的，意味着如果我们把这两个条件都用来查询，应该能得到整个表里的所有人。但是实际上并不是这样：因为我们不知道爱因斯坦的薪资，他不会出现在任何一个查询的结果中。这种棘手的情况会给查询评估带来不少问题。

Paragraph 41 (English):

There is another important concept: the key in the relational model. So the key is some attribute (either one attribute or a combination of attributes) which we think is unique. For example, your student ID is unique—no two students will have the same student ID. That’s a key. Sometimes, we will have multiple attributes combined together to become a key. In this example, for the instructor, the ID is a key, obviously. But if you combine the ID together with the name, it is also a key, because it’s also unique. We call it a superkey. But the superkey sometimes can be smaller, which means that we can remove some attributes and still have another key, right? But if a superkey cannot be reduced, which means it’s minimal, then we call it a candidate key. It cannot be made smaller—if you remove any attribute, it won’t be a key. Among all candidate keys, we then pick one as the primary key. So, for the primary key, each relation will have at most one primary key. And this key is pretty important because we are going to use this key later on to identify tuples, to arrange our storage, to establish the whole key–foreign-key relationship structure.

段落 41（中文）：

另一个重要的概念是关系模型中的键（key）。键指的是某个属性（或多个属性的组合），它在逻辑上是唯一的。比如，你的学号是唯一的，不会有两个学生拥有同样的学号，这就是一个键。有时我们会用多个属性组合成一个键。在本例中，对于教师这个关系，ID 显然就是一个键。但是如果你把 ID 和 name 一起作为一个组合，它也是一个键，因为它依然是唯一的，我们称它为超级键（superkey）。不过有些超级键可以进一步简化，也就是去掉一些属性后依然是键。而如果一个超级键无法再被简化（也就是它是最小的），我们就称它为候选键（candidate key）。如果你去掉其中任何一个属性，它就不是键了。在所有候选键中，我们会选一个作为主键（primary key）。一个关系最多只能有一个主键。主键在这里非常重要，因为之后我们会用它来标识元组、安排存储，以及建立整个主键-外键关系结构。

Paragraph 42 (English):

So, in this “instructor,” ID should be the primary key. For the “department” table, department\_name is the primary key.

段落 42（中文）：

因此，在 “instructor” 这个关系里，ID 应该就是主键；在 “department” 这张表里，department\_name 就是主键。

Paragraph 43 (English):

With the key, the database doesn’t have the concept of a pointer, but it has something similar, which is a foreign key. So, by “foreign key,” in this example, it is the department\_name in the “instructor.” Every instructor will have a department\_name—it must exist. So this department\_name, the value here, must appear in the “department” table’s department\_name. So this is a foreign key. And this kind of foreign key imposes a constraint. In other words, you couldn’t insert an instructor whose department\_name is just “engineering,” if you have “electronic engineering” but not “engineering,” and you couldn’t find this “engineering” department, which means that this instructor must have some misinformation in this line and violates this foreign key constraint.

段落 43（中文）：

有了主键之后，数据库并没有“指针”的概念，但它有一个类似的概念，叫做外键（foreign key）。比如在这个例子里，在 “instructor” 这个关系里，department\_name 就是一个外键。每位教师都有一个 department\_name，而且必须是真实存在的。因此，这里的 department\_name 的值必须出现在 “department” 表的 department\_name 列中，这才称得上是一个外键。这种外键会带来一个约束。换句话说，如果你想插入一位教师的信息，而他的 department\_name 只是 “engineering”，但在 “department” 表里只有 “electronic engineering” 而没有 “engineering”，那你就找不到这个名为 “engineering” 的院系，这说明这条记录存在错误，违反了外键约束。

Paragraph 44 (English):

So, these relations—we need to realize they are unordered. We don’t have a specific ordering among these tuples, and it could be stored in absolutely any order. But if you would like to have the result to be returned in a certain order, you could tell the database system to give you the result based on, for example, the ascending order of the name, the descending order of the ID; you could specify that. On the other hand, because the relation doesn’t have an order, you are free to set an order when you store this relation so that your query processing is more efficient.

段落 44（中文）：

需要注意的是，这些关系是无序的。它们的元组没有特定的顺序，存储时可以采用任何顺序。但如果你想让查询结果按照某种顺序返回，你可以告诉数据库系统按照某个字段的升序或另一个字段的降序来排序。与此同时，由于关系本身没有顺序，你也可以在存储这个关系时设置一种顺序，从而让你的查询处理更高效。

Paragraph 45 (English):

Okay, so this is the schema diagram for the university database. In this diagram, each relation is a rectangle, and the light blue part is the name of the relation. Each relation will have a set of attributes. The attributes that are underlined are the primary keys—so, for example, this “takes” has only one primary key, which is the combination of all these attributes together. So every single attribute here individually wouldn’t be unique. Some of the attributes may be foreign keys—that’s why we are having these pointers. So this ID of “takes” is the foreign key of the ID of the “student.” So the department\_name is a foreign key of the department’s department\_name, etc.

段落 45（中文）：

好了，这是大学数据库的模式图。在这张图里，每个矩形表示一个关系，浅蓝色的部分是关系的名称。每个关系都有一组属性。带下划线的属性是主键——比如说，这个 “takes” 只有一个主键，它是由所有这些属性组合在一起形成的。因此，这里每一个属性单独拿出来并不唯一。有些属性可能是外键，所以我们在图中画了指针。比如 “takes” 里的 ID 是 “student” 里的 ID 的外键，department\_name 则是 “department” 表中 department\_name 的外键，等等。

Paragraph 46 (English):

A little bit of wrap-up in the middle of the way. So, so far, we have introduced the database application examples to show how close you are to these database applications, and we’ve talked about the purpose of database systems: including “assist with massive data,” “save multi-user,” “convenience,” “efficiency,” and “reliability.” We’ve talked about the different types of data models briefly, and then we delved deeper to show what is a relational model: a lot of relations, each relation is a table, attributes as we name, as we type. And we talked about specifically the null value, the keys, and the schema.

段落 46（中文）：

这里我们稍微做个中途总结。到目前为止，我们已经举了一些数据库应用的例子，说明了数据库应用和我们的生活有多么密切。我们还讨论了数据库系统的目标，包括“大规模数据处理”“多用户并发”“方便性”“效率”和“可靠性”。我们简单介绍了不同类型的数据模型，然后深入讲解了什么是关系模型：包含多个关系，每个关系都是一张表，拥有我们定义好的属性。我们还特别讲到了 null 值、键以及模式。

Paragraph 47 (English):

A little bit more on the second part. Relational algebra is not technically so hard. Why we have this name of “relational algebra” is because algebra is like “a + b = c.” Well, you are taking two numbers, and you are outputting one number; multiplication is the same—two real values input, one real value as output. So, in this relational algebra, all the operations we are going to introduce will take relations as input and output another relation as output. So that’s the analogy. But we’re not going to have very complicated matrix or macro operations or something like that.

段落 47（中文）：

接下来再多说一点。关系代数从技术上讲并不是很难。之所以叫“关系代数”，是因为代数就好比“a + b = c”，你输入两个数，得到一个数；乘法也一样——两个实数输入，一个实数输出。那么在关系代数里，我们将介绍的所有操作都是输入一个或多个关系，然后输出一个新的关系。这就是它与代数的类比。但我们不会涉及到非常复杂的矩阵运算或宏观运算之类的东西。

Paragraph 48 (English):

So from this university database schema, we’re going to look at how we get data out of the database by queries using that schema, right? Remember that we have relations, where each relation has a schema and a lot of tuples. And now, if you look at the relation “instructor,” you have so many tuples, and how do you want to get information out of this relation? The simplest way is to just release the relation name—by giving the relation name, you can get everything out of that relation. But sometimes, you don’t want everything out of the relation, because you might only want to select rows, or you might only want to select columns, or you might want to select both rows and columns—some part of the relation like a rectangle sub-portion, or specific rows. So we’re going to use operators to filter, slice, or combine information from the relations.

段落 48（中文）：

接下来，我们就用这个大学数据库的模式，来看看如何通过查询从数据库中获取数据。记住，我们有若干个关系，每个关系都有一个模式，以及大量元组。现在，如果你查看名为 “instructor” 的关系，它里面有很多元组，那么我们该如何获得这些关系里的信息呢？最简单的方式就是直接写出这个关系的名字——你就能得到这个关系里的所有内容。但有时你并不想要全部内容，只想获取其中的某些行或者某些列，或者同时只要某些行和某些列（类似一个矩形子区域），或者更具体的几行。为此，我们就会使用一些运算符，对关系进行过滤、切片或组合。

Paragraph 49 (English):

So the first operator we’re going to talk about is the select operator. The select operator picks certain rows out of the relation. For example, if someone would like to select the instructors where the instructor is in the “Physics” department, he or she could use this select operator. It looks like this: sigma (subscript: department\_name = “Physics”) of instructor. So with this expression, we are applying the selection operator over the “instructor,” with the selection condition that “department\_name = Physics.” And by applying it to this relation, you will get two tuples, because both “Einstein” and “Newton,” for instance, are from Physics.

段落 49（中文）：

我们要介绍的第一个运算符是选择（select）运算符。选择运算符会从关系中选取特定的行。举例来说，如果有人想要选取所有院系是 “Physics” 的教师，便可以使用这个选择运算符。写法大概是：σdepartment\_name = “Physics”(instructor)。也就是说，我们对 “instructor” 这个关系应用一个选择操作，条件是 “department\_name = Physics”。通过在这个关系上应用这个选择条件，我们就能得到两个元组，比如 “Einstein” 和 “Newton” 都来自物理系。

Paragraph 50 (English):

Okay, now let’s think about a little bit harder query, like “select instructors in Physics with salary greater than ninety thousand.” So to cope with that, we are going to use this as a base. The only difference between these two queries is an additional condition, which is “salary > 90000.” So we’re going to extend this condition by using AND logic connection: “department\_name = Physics AND salary > 90000.” That’s it. So by looking at these two examples, we can see that the selection operator just uses sigma and the condition in the subscript, followed by the relation in parentheses. Then the condition could be comparisons, like something equals, something not equals, greater than, greater than or equals, less than, less than or equals, and it can combine different predicates with logic connectors like AND, OR, NOT. So this is selection.

段落 50（中文）：

好的，现在我们再想一个稍微复杂点的查询，比如“选取院系为 Physics 且薪资大于 90000 美元的教师”。为应对这个需求，我们可以在上一个例子的基础上稍作修改。两者的唯一区别就是额外增加了 “salary > 90000” 这个条件。所以我们把这个条件和之前的条件通过 AND 逻辑运算符连接起来，也就是 “department\_name = Physics AND salary > 90000”。就这样了。通过这两个例子可以看到，选择运算符的写法就是在 σ 符号下标中写出选择条件，然后在括号里写要操作的关系。条件可以是各种比较运算，比如相等、不相等、大于、大于等于、小于、小于等于，也可以用 AND、OR、NOT 之类的逻辑运算符组合不同的谓词。以上就是选择运算的用法。

Paragraph 51 (English):

After selecting rows, let’s see how we select columns. So to select columns, we are going to use another operator called projection. In order to pick ID, name, or salary of “instructor,” we use pi. Pi is the project operator. And then we will have (ID, name, salary) in the subscript, because we want these three columns from the relation “instructor.” So by executing this query, we are going to have these three columns—essentially, we just do away with the department\_name column and then keep everything else. So by observing this example, we can see that the project operator is just pi, and these are the attribute names you would like to select. Then this “R” is the relation that you would like to perform projection on.

段落 51（中文）：

选择完行之后，让我们再来看如何选择列。为此我们要用到另一个运算符，叫做投影（projection）。如果我们想选取 “instructor” 关系的 ID、name 和 salary 这几列，就可以使用 π 运算符（pi）。写法是：π(ID, name, salary)(instructor)。执行这个查询后，我们就能得到这三列——本质上我们把 department\_name 那一列去掉了，保留了其他列。通过这个例子我们可以看到，投影运算符就是 π，这些括号里的属性名称就是你想要的列，然后后面再跟一个关系 “R”，表示要在哪个关系上进行投影。

Paragraph 52 (English):

So now, it comes to a question: can we select both rows and columns at the same time? In order to do so, we could compose operators together. For example, we would like to find the names of all the instructors in the Physics department. So here, “find the names” means we need to do a projection—originally, we have four columns, now we only need to know the name. And “in the Physics department,” we need to do the selection operation. So we first select, and then project. And this kind of concatenation of the two operators shows how to compose two operators together.

段落 52（中文）：

现在出现一个问题：能不能同时选取某些行和某些列？要实现这一点，我们可以把运算符组合起来。举例来说，如果我们想要获取所有物理系教师的姓名，这里，“获取姓名”意味着我们只要投影出姓名这一列，原本有四列，现在只要姓名这一列；而“物理系教师”意味着我们要对“院系 = Physics”执行选择操作。所以，我们先选择，再投影。这种将两个运算符串起来的方式，就示范了如何把多个运算符组合使用。

Paragraph 53 (English):

One question: can we switch the order of these two operators? Yes or no? You cannot switch. You can’t switch—why? Because if you switch it first, then you have just the name column, and then you cannot pick more than one name. So we couldn’t do that, because once we project over the name, the relation will only have one column (the name). Then your selection operator won’t be able to check the department\_name column anymore, because that column doesn’t exist in the projected relation. And that’s why the selection will not do anything. So the switch will make an impact.

段落 53（中文）：

这里有个问题：我们能不能交换这两个运算符的顺序？答案是不能。为什么？因为如果你先投影出姓名这一列，那么得到的关系就只剩下 name 这一列，之后再进行选择操作时，已经找不到 department\_name 这一列了，因为它被丢掉了。所以在这种情况下，选择操作就失效了。这就是为什么我们不能交换它们的顺序。

Paragraph 54 (English):

Okay, next we’re going to talk about: in the previous three slides, we’re talking about how to select stuff from only one relation, right? You can select rows, select columns, and select both rows and columns. What if we want to select things from multiple relations? In this case, we’re going to introduce the basic construct “cross product” or “Cartesian product.” It combines two relations together. So see this example: we have “instructor,” we have “teaches.” Then, if we do a cross product of these two relations, what we generate is a relation—remember, in relational algebra, input relation, output relation—so now we’re going to generate a new relation. The schema of the new relation will be the union of the attributes of both “instructor” and “teaches.” So here, you should notice that they both have an “ID,” right? In this sense, we need to distinguish the ID from the instructor and the ID from teaches. So all the other attributes are just copied/pasted: name, department\_name, salary, course\_id, sec\_id, semester, year. How many attributes will we have in this new relation? Four plus five is nine, right?

段落 54（中文）：

好了，接下来要说的是：在之前的示例中，我们都是在单个关系里做选择或投影，对吧？可以选行、选列、或者同时选行和列。但如果我们想从多个关系里选取数据该怎么办？这里我们就要介绍一个基本的结构——笛卡儿积（cross product 或 Cartesian product）。它能把两个关系组合起来。比如，我们有 “instructor” 和 “teaches” 这两个关系。如果我们对这两个关系做一个笛卡儿积，就能生成一个新的关系——记住，在关系代数中，输入是关系，输出也是一个新的关系。那么这个新关系的模式就会是 “instructor” 和 “teaches” 两个关系的属性集合的并集。需要注意的是，这两个关系都有一个 “ID” 属性，对吧？因此我们要区分开 “instructor” 的 ID 和 “teaches” 的 ID。除此之外，其他属性如 name、department\_name、salary、course\_id、sec\_id、semester、year 等等都拷贝过来。一共会有多少属性？4 个加 5 个等于 9 个，对吗？

Paragraph 55 (English):

So how to populate this relation is like: for every single tuple in “instructor,” we’re going to combine it with every single tuple in “teaches.” In other words, we’re going to enumerate all pairs of tuples (one from each relation) to form the new tuple in the new relation. So here comes the question: if we have X tuples in “instructor” and Y tuples in “teaches,” how many tuples will we have in the result? X times Y. Yes, X times Y. So this operation is pretty expensive if you think about it, right? If you have ten relations, each having one hundred tuples, you will have 10^(100 \* 10)… well, effectively an enormous cross product. That’s what Cartesian product means. And if you look at this table, sometimes it doesn’t make much sense, because an instructor may be connected with the courses he or she is teaching, and sometimes it will be matched with some courses not taught by that instructor.

段落 55（中文）：

那么如何生成这个新关系呢？就是对于 “instructor” 中的每一个元组，都去和 “teaches” 中的每一个元组配对，形成一个新的元组。换句话说，列举所有可能的元组对（每个关系中各选一个元组），放到这个新关系里。由此产生一个问题：如果 “instructor” 里有 X 个元组，“teaches” 里有 Y 个元组，那么结果中会有多少元组？答案是 X × Y。对，这个运算量想想就挺可观的。如果有 10 个关系，每个都有 100 个元组，那么组合起来就可能是一个极其庞大的笛卡儿积。这也正是笛卡儿积的含义。而且，如果你仔细看这个表，就会发现有时它没什么意义，因为一个教师只会教自己所教的课程，却会在结果中与所有课程进行配对，很多对其实是不相干的。

Paragraph 56 (English):

So most of the time, when you have a key and a foreign key—this is a key, and this is a foreign key, right?—you would like them to be equal when you want to combine the two relations. Of course, you can do it by just the Cartesian product and then a selection: you just specify, “We only need the tuples that have these two values the same.” But also, you can use a new notation: ah, so this is “select (instructor.ID = teaches.ID) over the Cartesian product of instructor and teaches.” We can also use a more concise and easy-to-write notation called the natural join. Natural join is represented by a bowtie. It means that when you combine two relations, whatever common attributes they share, they should have the same value, so they should agree with each other on the common attributes before they can be combined together.

段落 56（中文）：

因此，大多数情况下，当你有一个主键和一个外键——比如说这是一个主键，那边是一个外键——如果想要把这两个关系组合起来，你会希望这两个键的值是相等的。当然，你可以先做笛卡儿积，然后再做一次选择来挑出主键和外键相等的行。比如写一个条件“instructor.ID = teaches.ID”，然后在完成笛卡儿积后只保留这些匹配的元组。不过也可以用一个更简洁的记法，就是自然连接（natural join），用一个“⊲⊳”符号来表示。它的含义是，当我们把两个关系合并时，它们所有同名的公共属性都必须取相同的值，才能把那两条元组连在一起。

Paragraph 57 (English):

So that’s another point. With a natural join, think about how you’re going to express “find all the names of instructors whose department building is 303 and who have taught a course in 2024.” So, “find all the names” means we want a projection of instructor. We want “department” where building = 303, so we can just natural join these two relations, because they’re going to be combined on the department\_name. So the department’s building is 303. You can just do it with “instructor,” right? Then we also have to join “teaches,” so you specify year = 2024. So basically, what you want to do is: you would like to do the join of the three relations together: “instructor,” “department,” “teaches.” So in this kind of graph that we’re drawing, it asks that this value be equal to this value, etc. And then, after doing this join, and then joining “department,” you want to have this value equal to this value. Then you want to select conditions: one is building = 303, and year = 2024. So, I mean, the ordering doesn’t matter that much; you can change the ordering. The whole expressions have alternative forms that can be equivalently correct. It’s not a unique answer. You can select building = 303 over “department,” right, so you can impose this selection over “department” first, and then you use the selection year = 2024 on “teaches” before you perform the join. So it will all be the same in terms of final result.

段落 57（中文）：

那么自然连接还有个好处，比如你想表达“找出所有院系建筑是 303 且在 2024 年教过课程的教师姓名”。这里，“找出所有姓名”意味着要对教师做投影；“院系建筑是 303”意味着要对院系进行选择；然后我们可以用自然连接把它们联系起来，因为它们会通过 department\_name 这列进行匹配。接着我们还要把 “teaches” 这个关系连接进来，再加一个选择条件 year = 2024。总结起来，你需要把“instructor”、“department”、“teaches”这三个关系连在一起，在连接时要求公共属性取相同值。连接完后，你再加上 building = 303 和 year = 2024 这样的选择条件，或者你也可以先对 “department” 做选择（building = 303），对 “teaches” 做选择（year = 2024），然后再进行连接。顺序倒不是那么重要，表达式可以有不同的等价写法，并不只有唯一的答案。

Paragraph 58 (English):

Okay, finally, the θ-join (theta-join). So this is the Cartesian product plus a condition. This does not introduce any new expressiveness; it’s just an abbreviation. If you want to do a Cartesian product and then you have some selection condition out of that, you can just put the condition here in the bowtie, and that replaces “select the condition” over the Cartesian product. So it’s just an abbreviation. So this corresponds to—I will introduce later—this corresponds to a very typical simple query: you “SELECT something FROM a set of relations” and you have a “WHERE” clause. This is a typical situation. The WHERE clause will describe the condition; the FROM clause indicates the relations, which basically means they do a Cartesian product behind the scenes. So that’s why it’s the basic operation implemented by the database management system.

段落 58（中文）：

最后说一下 θ-连接（theta-join）。它其实就是笛卡儿积再加上一个条件。这并不会带来任何新的表达能力，只是一种缩写写法。如果你想先做笛卡儿积，然后再用一个选择条件来筛选，你可以直接在连接符号这里加上这个条件，代替先做笛卡儿积再做选择的两步操作。所以它只是一种简写形式。它所对应的也正是我们之后会介绍的典型的简单查询：从一组关系中 SELECT 某些字段，然后在 WHERE 子句里写条件。在后台，FROM 子句指明了要做笛卡儿积，然后 WHERE 子句再对结果进行筛选。这就是数据库管理系统最基本的运作方式之一。

Paragraph 59 (English):

So, to wrap up: we’ve talked about why we use databases, we’ve talked about the relational algebra, we’ve talked about the simplest query, which is just the relation name, and we use operators to filter, slice, and combine. And so far, we’ve introduced the select, project, cross product, natural join, and theta-join. In the next lecture, we will talk about more query operators.

段落 59（中文）：

好了，总结一下：我们聊了为什么要使用数据库，也聊了关系代数；我们介绍了最简单的查询，即直接写出关系的名字，并且可以用各种运算符进行过滤、切分和组合。到目前为止，我们已经介绍了选择（select）、投影（project）、笛卡儿积（cross product）、自然连接（natural join）和 θ-连接（theta-join）。在下一次课中，我们会讲更多的查询运算符。