Module 1: Video Lecture Notes

[**Video 1.1 1**](#_nhimnsn6cyst)

[**Intro to DBMS 1**](#_hjsg0hohp6vi)

[**When to use DBMS 1**](#_w7jsjvkdjfc3)

[**Stages of Database Design 3**](#_7p5pn7o43ab0)

[**Types of Databases 3**](#_fth6ec68v6wd)

[**Characteristics of the DBMS Approach 4**](#_9qijutaf36jy)

[**Advantages of the DBMS Approach 6**](#_uio3hcv38p1k)

[**Video 1.2 6**](#_6rfm8zn1d9bj)

[**Advantages of the DBMS Approach 6**](#_svk8db5jq3g)

[**Advantages of Databases 7**](#_gi4jhvvlcfe8)

[**Redundancy Control 7**](#_ykn12qmgj31d)

[**Efficient Search 9**](#_cpu2yvvxy9g9)

[**Integrity Constraints 12**](#_jl71cl6c96t8)

[**Data models, schemas, and instances 16**](#_mixrh8r6oki1)

# Video 1.1

<https://merrimack.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=27b81e94-d63d-4658-b918-b049016c64e2&start=134.33045>

## Intro to DBMS

| **Notes** |
| --- |
| Hello. Welcome to Week 1. Welcome to Lecture 1, Part 1. This week we’re going to be talking about introduction to database management systems. That should be Chapter 1 and Chapter 2 of your textbook. So please read that. Use these lecture notes kind of as a guide to get the most important parts out of the text, and make sure you master and understand each concept we talk about in these notes. |

## When to use DBMS

| **Notes** |
| --- |
| So the first chapter is going to look at when to use a database management system and what is a database? Right? So it’s going to talk about what exactly a database is, why you should use it and when you should use it. Well, when you use it, typically you’re going to have a set of users or an audience who needs to use that data in some form or fashion. Right? So they may be using, and the classic example is they may be using Excel sheets or Excel workbooks, and that’s how they access data. That’s how their data is stored. And multiple users may have different versions of that same data. And that creates all sorts of problems. Right? We’re not going to get into those problems. But you can imagine if your database today is a bunch of Excel workbooks, there’s a lot of inconsistency between those workbooks, especially if multiple people are using them. So that’s the first use case. You have an audience who needs to acquire and use the data for known or unknown purposes. And if you know the purpose, typically you’re going to design your database to be most efficient for those purposes. There may be some unknown purposes, but it’s very hard to plan for those. The second is you have some type of software or web application that needs a medium to store the inputs and outputs of that data. Most of your web applications, if not all of your web applications, are going to be built on some type of database, whether it’s a relational data model or a NoSQL data model or even a graph database, its back end is most likely going to be some type of database. Transactional system—so banking is the most common example used. A bank teller has to debit and run debits and credits on accounts. That’s typically done through transactional systems. So these three examples here highlight two different types of database systems. OLAP, which is an online analytical processing database—that’s typically used for analytic types of queries. Those are analytic reads. That’s when users are querying the database to extract data to provide some insight to the business. The other is OLTP, or online transactional processing systems. Those are systems that are highly transactional, which means there’s a lot of rights to the system, there’s a lot of updates to the system. The perfect example is a banking software where they’re storing your account information in the database. So when you are deciding to use a database management system, there are some considerations you have to take in mind. That’s the data type, structures, constraints. Those need to be understood and defined before building your database, and it’s very important to do that work before you start creating your database. A lot of bad things can happen if you just go out and create a database, especially if you don’t understand who your target audience is, who your users are, and how they’re going to use it. How long do we need to store the data? You know, you don’t want to store your data forever. Right? If we’re storing data all the way back from the 1900s into today and we’re getting, you know, updates every 15 minutes, that could potentially be a lot of data, especially when you’re talking about network traffic or network type of data. It’s just not reasonable to store that amount of data, simply because that requires a lot of disk space to do so. So how long do you need to store it after it’s kind of met that threshold? You need to do something with it, typically take it out of the database and put it in some type of cold storage. What types of queries? This comes back to understanding who your users are, how they’re going to be using it. Are you going to be doing a lot of reads, a lot of updates? Is it better to use an OLAP type of system or OLTP type of system? And again, that comes back to understanding who your users are and how they’re going to be interacting with the data. What types of security constraints are there? Do you need to set the database to read only for specific users? Do you need to lock out specific users? How will they be using the data? Are they allowed to do updates? Are they allowed to do inserts? Are they allowed to create tables? All of these things need to be understood. Scale—how big is your data in that? And that “big data” terminology is very ambiguous. So how much data do you have? What type of system do you need to support that data? Do you need to scale up on a single-node database? Meaning you have one server where your database lives and you’re giving it more RAM, disk space, and CPU power to do the transactions. That’s what we mean when we say “scale up.” Or do you need to scale out and build a database on a distributed type of system like Hadoop or a file system? So that scale is important because that’s going to define what type of database ultimately you’re going to be using. |

## Stages of Database Design

| **Notes** |
| --- |
| Stages of database design—so I mentioned that it’s very important to understand the use cases before you go out and just build a database. Right? So there’s something called an SRS—software requirement specification. We’re not going to dig in too deeply into this class what that is. If you were getting a degree in information systems management, you’d actually have to write one. But an SRS is where you’re going to define who your users are, how they’re going to use it, and then gather requirements. Conceptual design, which is kind of more along the lines of what we’re going to be doing in this class, is where we get into our entity relationship diagrams or our ERDs. That is how all of the tables and schemas interact with each other within the database, how you combine data from two different tables, et cetera. So that’s called an entity relationship diagram. We’re going to go pretty heavy on those next week. Logical design is more of your data models and your flow charts. That’s kind of a high level of how everything kind of works together between schemas and tables and kind of how everything looks at a high level. And then you get into your physical design, which is how the database management system is built and maintained. We’re going to focus more on our ERDs, our conceptual and logical design in this class, not necessarily as much as the physical design, although that’s important. You should know how the database is built, how it’s maintained. But we’re going to be focusing more on data modeling, which is, again, entity relationship diagrams, data models, and flowcharts. |

## Types of Databases

| **Notes** |
| --- |
| So there are several different types of databases. Right? So relational database management systems, which is what most of you, if you’ve interacted with a database in the past, have been introduced to. So again, there’s two different types. There’s the transactional type of systems and the analytical types of systems. Classical examples are PostgreSQL, MySQL, MySQL Server, Oracle. Oracle has its own SQL dialect called PL/SQL. So those are relational database management systems. Those assume you’re building out a relational model where you have something called a star schema, or you’ve got control tables, or you have tables with some type of relationship to other tables. Right? And that kind of goes into the process of normalization, which we’ll also talk about. But those are relational database management systems, and that’s kind of what we’re going to focus on in the first four weeks of class. Another one is NoSQL. And this is a term, I believe, that was coined on Twitter in 2014, describing a unstructured database that is more along the lines of, you know, if you’re familiar with JSON data, typically NoSQL databases are JSON based where you’ve got, you know, very unstructured data. You can have nested structures, but they’re more of a tree type of structure as opposed to the kind of that tabular structure that you’re most probably accustomed to. So we’re going to focus on NoSQL, which, you know, I don’t know that I agree with that term. I think it was a hashtag on Twitter in 2014, and it exploded. And so now everybody calls it NoSQL, but typically these are tree types of data structures stored as JSON or sometimes XML, most likely not XML, most likely JSON. Examples of those are going to be MongoDB, which we’re going to use in class. And then Couchbase, which is a newer kind of NoSQL database that we’re not going to talk about. We may talk about it, you know, kind of casually, but we’re not going to focus on it in class. Now, there are several other NoSQL types of databases, Cassandra. I think Redis might be one. We’re going to focus on MongoDB. They do a great job with their education materials and training materials. So that’s a big reason why. And they’re very popular and used widely throughout the industry. So we’re going to focus on MongoDB in the NoSQL portion of class. There’s also graph databases used to store geographic data or data that is highly interconnected, and that’s typically the best use case I’ve seen personally for graph databases. You may have data that has all sorts of relations amongst itself and between other entities within the data, and those complexities are far too large and far too inefficient for a relational database. So you’d want to use a graph database. Now MongoDB actually has graph technology. We may touch on that briefly when we talk about MongoDB. But probably the most popular graph database out there is Neo4j. And then you get into your big data systems, and these are typically file systems. I wouldn’t call them a database. They’re file systems, and they’re distributed. I’ve mentioned scaling up versus scaling out. These big data types of systems focus on commodity hardware, so very cheap servers. You get hundreds if not thousands of these very commodity servers. And you build either, you know, some type of Hadoop file system or, you know, the new thing is databricks that’s in the cloud. But it is a distributed file system built on a file type called Parquet, typically. There’s also ORC. And then they have high-level query engines on top, either Spark or Hive. When we get into the big data class, we’re probably going to be focusing on Spark. Spark is an in-memory query engine that’s able to access and process that distributed file system quite efficiently. But when you get into the really immense amounts of data, you’re typically going to do this on a distributed type of file system using Spark or Hive. So those are the different types of databases. And again, this class is going to be focused mainly on one and two, relational databases and NoSQL. |

## Characteristics of the DBMS Approach

| **Notes** |
| --- |
| What are the characteristics of the database management system approach? Well, it’s a collation of files related to the same thing. I gave an example. I believe your book talks about this as well. You may have several, I guess teachers, all with data related to students, related to their classes, related to their departments, et cetera. And they may be storing it all in Excel today. Well, all of those files that these teachers may have are related to a common theme. And that theme can be described and built properly with a database. So that’s really what a database is. It’s a collation of all of these files related to the exact same thing. The self-describing nature of the database system—so you’ve got a catalog that’s your metadata that if you don’t know what metadata is, it’s the data about the data. It describes what the data is. You also have your DDLs, DMLs, DCLs, and TCLs. These are more of the data modeling languages. It’s more the data modeling language. So DDLs are your create statements. I want to create a table. I want to alter a table, so make changes to that existing table. I need to just drop the table. That table doesn’t relate to our database anymore. I need to do some type of truncation, make a comment, or rename a column. That is what the DDL is, and we’re going to discuss this further. There’s also DMLs. You can select data. This is more of the analytical type of processing, select, inserts, updates, deletes, merges, merging to tables, calls, explains, and locks. We’ll talk about locks when we get into concurrency control. DCLs are your grants and revokes. We’re not going to talk about that a lot. And then we’ll also have TCLs, transaction controls—commits, rollback, savepoints, set transaction, commit. You make a change, you commit it to the database. As long as you’ve got good ACID properties, you made a change. It was the wrong change. You can rollback those changes. You’ve got savepoints in your database if you do need to do those rollbacks. So that’s what we talk about by transaction controls. It’s actually quite important, especially in your lab. There’s a question on concurrency control. So that talks about your TCLs there. Insulation between programs and data and data abstractions. Databases are more commonly used as abstraction layers on top of or beneath the application layers. If you’ve got a web application or some type of software, you don’t want to handle all of that data related to that application in memory or save files and create your own database or own file system with the application. Typically you’re going to use a database like Oracle or MongoDB or some other type of database to handle the changes in the data that are generated through that application and updates to the data through some ETL process—extract, transform, load—instead of doing that all in the app design. So that’s all going to be handled through the database. And the idea here is your software data and dependents, you want your back end or your data architecture and infrastructure to be independent from your user-facing application, even though your user application is going to interact with that database. Right? So that’s what we mean by that insulation between the two. Support for multiple views of the data—so databases do allow you to create something called a view. This could be some type of commonly used query that maybe combines different data or does some type of operation on your data, and users need to interact with this view. Again, that’s an abstraction on top of maybe it’s raw data, and maybe that raw data isn’t massaged or does not have the features that it needs to for your customers. You need to create a view on top of that. So the database does typically support that, and that’s what we mean by support of multiple views of the data. It’s the same data, it’s the same raw data. You’ve just got some type of abstraction on top of it that’s more customer friendly. And kind of the last bullet point here is probably the most important part here. There is a question on concurrency control in your assignments. So that’s sharing of data and multiuser transactional processing. OK? So what is concurrency control? So a database must have some type of concurrency control. What do we mean? So in a multiuser environment, you may have someone who is trying to, maybe they’re altering a table or they’re updating a row. Right? And another user is trying to query that same exact data. So maybe they’re either trying to query that table, or they’ve got a query to query that specific row. What we mean by concurrency control is the first person who does that alter or update statement will put a lock. So we go to the DML here. They will put a lock on either that row, or if they’re altering the whole table, they’ll put a lock on that table to ensure consistency. Right? So that’s what concurrency control is going to do, is ensure consistency. If you’ve got multiple people trying to access the data at the same time, then you want to be able to lock those rows out so that that person can change those rows, and then that other person coming behind them after they’ve made their changes and query that data. Now, there can be some problems with that. Right? Depending on the database, there can be something called a dirty read. I’m not sure if your textbook talks about dirty reads. But what a dirty read is, is to say someone is doing an update to a row, and they put a lock on that row. So a lock means nobody else can change that row while that person has a lock on it. Right? So there’s a lock on that row. Someone comes in and queries the data—they can query that row—and get whatever value from that row that was before the update. Right? So they’ve got that value. They’re looking at that value. Well, this person then goes and finishes their update and updates that. The person who queried that result has what is called a dirty read. They read that data before it was updated, so they have old data. Right? That is expired, et cetera. That is called a dirty read. So there are problems with concurrency control that you do need to be aware of. Right? So dirty read is probably the biggest problem. Specific databases handle it differently. So when you’re choosing your database technology, if concurrency is a big concern, make sure you understand how they’re handling concurrency control. So ACID—so what do we mean when we talk about ACID transactions? OK, so atomicity, consistency, isolation, and durability. What are each of these? OK. So atomicity means that every single transaction in a database is a single unit. So a read on a row is a single unit. An update to a row is a single unit. And everything is considered a single unit. OK? And what this prevents is data loss and corruption from occurring. If you’re, you know, streaming data and something fails midstream, you’ve got one right done on one row. The database understands where it left off and you can kind of pick back up after you’ve written maybe 30 rows incrementally. Right? So say if it fails after 30 rows, you’ve got the 30 rows done, the database knows those 30 rows are done. You can pick back up and say you’ve got 70 more rows to write. You can finish those 70 right after you fix whatever failure happened in the database. So that’s what we mean by atomicity. And I’m struggling to enunciate that word for whatever reason today. Consistency, we kind of talked about this when we talk about concurrency control, but that means, you know, transactional consistency ensures that corruption or errors in your data do not create unintended consequences for the integrity of your table. Again, dirty reads are a perfect example. Isolation—when multiple users are reading and writing from the same table at once, isolation of those transactions ensures that the concurrent transactions don’t interfere with or affect one another. OK? Important. Durability—so ensures that changes of your data made by successfully executed transactions will be saved even in the event of a system failure. So that kind of goes back to our atomicity point, where I gave the example we’re streaming data where we write one row at a time and we get to 30 rows. Durability ensures that if the database fails after those 30 rows have been written, we can pick back up and write the last 70 rows once the database is back online. Right? OK. So I’m going to give you all a break. |

## Advantages of the DBMS Approach

| **Notes** |
| --- |
| We’re going to do these videos and short chunks. The next video, we’re going to talk about the advantages of the database management system approach. The second is you have some type of software or web application that needs a medium to store the inputs and outputs of that data. |

# Video 1.2

## Advantages of the DBMS Approach

| **Notes** |
| --- |
| Hello. Welcome to Part 2. Hopefully you can see my screen. Before we get started, I want to clarify one thing from the last early Part 1. When we talk about NoSQL, what we’re really talking about is a document model, and a document can be thought of as a singular JSON record, and then your database or your, I guess, table—it’s not really a table. It’s called a collection. So your collection would be a collection of JSON documents where one JSON document may have several different characteristics. So that’s what we mean when we talk about NoSQL. They’re considered document models. And again, NoSQL was coined back in 2014. I think there was a meeting some folks had out West. And to get people, I guess, to show up to the meeting, they posted on Twitter and put the hashtag NoSQL, and that’s how it started being called NoSQL. But really what we’re talking about is the document model. OK? Instead of the relational model, it is a document model.  So I just wanted to clarify that so that there was no confusion. NoSQL is just a buzzword, and it’s not really a technical term. OK. So advantages of the database management system approach. |

## Advantages of Databases

First, there's three primary ones we’re going to hit heavy on. There’s more in the book, but these are the ones I really want to drive home.

### Redundancy Control

| **Notes** |
| --- |
| The first is redundancy control. So what do we mean by redundancy control? That is in a relational model also called normalization.  2:01  And normalization means in each table you have unique records.  2:11  You don’t have any data duplication. So if you look at our example here, we’ve got three tables.  2:17  They all have some type of relationship with one another. But our primary  2:23  keys are student ID, course ID, and instructor ID.  2:30  Right? OK? So you notice that all of these IDs are unique,  2:35  the primary keys are.  2:40  And we’re not storing courses in the students, because you can see here, we’ve got kind of this one-to-many relationship going on here.  2:43  And what that means is you would have potentially multiple courses for each student, and you’d have data duplication on your courses.  2:52  Right? So if we look at CS 1, there’s three students that take that.  2:59  So you’d have Java, Java, Java here  3:05  for all three of those students. You would have this data duplication across all of these rows.  3:09  And yes, we’re duplicating the course ID here,  3:16  but the point is you’re not duplicating any of this other data and thus saving space on your hard disk and  3:20  also improves read performance because you’re not reading as much data back to the driver, but you have to join that data.  3:29  And sometimes in web applications or other use cases, you may not want to do those joins because those joins could also be compute intensive.  3:39  And so we have this thing called controlled redundancy. Right?  3:51  And typically in a relational database, you’re going to have controlled redundancy in a view.  3:55  So you wouldn’t do that directly in your tables.  4:02  You would have some kind of view, some type of abstraction on top of your raw tables, and you would have your controlled redundancy in that view.  4:05  OK? So that’s kind of the best use for views in the relational model.  4:14  Now in the NoSQL, or the document model, typically you don’t do joins at all ever in document  4:20  models, and we’ll talk more about that in Week 5. But that is another example of controlled redundancy.  4:32  Your document would have all of these values within the document.  4:37  So if you’re familiar with JSON, what you would have is you would have your JSON record.  4:44  So one record would have maybe a another sub JSON record called Students.  4:50  And inside that you would have all your student IDs and the courses they’re taking. And then maybe you have, you know,  4:57  your course table and then your courses and instructors in another sub document,  5:04  or you just combine all of those into one singular JSON record. Right?  5:08  That’s something you would have to plan for. Excuse me one sec.  5:13  Apologies. The children are running free. So yeah, that’s what we mean by controlled redundancy,  5:18  or D normalization.  5:26  Normalization is kind of separating everything into your new unique records in individual tables, such as we have here. D normalization—  5:27  one second again. OK. I think that might have taken care of it.  5:39  OK. So that is what we mean by redundancy control. |

### Efficient Search

| **Notes** |
| --- |
| And so the next kind of advantage of the database system approach is efficient search.  5:53  So efficient search means we should be able to run efficient queries.  6:01  We shouldn’t have to scan an entire table to find the records we’re looking for.  6:06  We should be able to scan an index or scan partitions or scan a sharded database.  6:11  Right? And so efficient search is another advantage.  6:19  If you were just doing this in Excel and you had to look for a single record,  6:23  you would have to scan the entire table to find those individual records.  6:27  So databases provide indexes, partitions, or sometimes shard, and sometimes they can cache commonly used queries in memory.  6:31  And we’re actually going to go pretty far in depth into what an index is and what a partition is.  6:41  I have examples in your notes folder. I’ll create another video for that.  6:47  But what we mean by an index, a database index, is essentially if you index a column,  6:52  the database creates a pointer to the disk location of those records.  6:58  OK? So if we look at this kind of top example here, we’ve got, you know, our search keys  7:04  Alice. This is an index column, this name column. That is in this particular disk location.  7:12  And so we would instead of scanning a database of 100 rows, we would scan our index.  7:20  We would find Alice. She’s in disk location 149. We would go pull that.  7:27  This is using B-trees. We’ll get into more depth on that in another video.  7:32  But we basically find that disk location, return that record, or return in this case,  7:37  those records, Alice and Anna are both on this location 149.  7:42  So there would be a predicate push down there or a filter there to get Alice back.  7:47  David is in 2459, so we would search this partition 2459, find those records, return them back. OK?  7:55  So that’s kind of the high level idea of an index. You’re not scanning the entire database  8:04  or table. You’re scanning a much smaller index with disk pointers. Partitions are a little bit different.  8:10  Partitions are used in file systems, so think Hadoop. And remember, Hadoop is a file store.  8:19  It’s not necessarily a database. So how does that work?  8:26  Well, the most common way to partition data, typically this data is going to be time series data.  8:32  It’s going to come in daily, it’s going to come in hourly. At least that’s the use cases that I’ve seen working  8:37  for Samsung and US Cellular and a lot of other telecommunication companies.  8:46  A lot of our data is time series. So we partition on date.  8:50  What does that mean? So basically you would store all of the files for January.  8:55  In this particular case, partitions can be actually much smaller than that.  9:03  We have partitions down to the hour because our data is so big. But in this case you can see our File Group 1 is January data. File  9:07  Group 2 is December data. File  9:14  Group 3—or that’s February.  9:17  This is December, File Group 3. And so when you query big data systems or these big file systems, you want to  9:21  use the partitions in your where clause to filter out the rows for these specific months.  9:31  Right? And so you organize your file system, and I’ve got an example that will cover that again in another video.  9:38  So you organize your file system on those partitions so that that data is all collated kind of in the same folder structure and same files.  9:45  Sharding is very similar. Shards are used more in MongoDB parlance,  9:56  but very, very similar idea.  10:02  It’s not a file store in this case. It’s called the database, but the documents are stored kind of in the same location on disk using shards.  10:06  So very similar to partitions. And then caching just means if you’re running the same query or you just executed a query,  10:18  that query is going to live in memory in the database for n time.  10:26  I think you can configure that.  10:32  And so if somebody else has to run that query or web application is running that query, it can just go to the memory, go to memory,  10:34  pull that data from memory instead of going back to disk, reading from disk, caching back into memory, and then doing whatever you’re doing.  10:41  So that’s what we mean by caching, caching in memory.  10:50  Memory obviously is a lot faster than going to disk, using the CPU to load the data onto memory and then doing that.  10:53  It just lives in memory for n time. That’s what we mean by efficient search.  11:01  If you’re just, you know, using Excel books, they’re not going to have a lot of this capability.  11:07  So that’s Number 2. And I will create another video to kind of more in depth on how indexes and partitions work. |

### Integrity Constraints

| **Notes** |
| --- |
| The third is integrity constraints.  11:21  What that means is in a relational model, much less  11:25  NoSQL, MongoDB, CouchDB, some of these other, you know, document models don’t have as many integrity constraints.  11:32  They’re more for flexible schema designs. But integrity constraints essentially means you have a predefined schema.  11:42  So your table has the same columns. Right?  11:50  And those columns may have the same size.  11:55  If you’ve got a column of names, maybe those names are never going to be more than 75 characters.  11:59  So you can impose a VARCHAR 75 constraint on that column to ensure that at no point will any name be more than 75 characters.  12:06  That’s one. Type—every  12:18  value in this column is an integer.  12:22  That’s another integrity constraint.  12:29  So that’s your basic integrity constraints. There’s also referential integrity.  12:34  That’s more when we get into that kind of that relational model.  12:40  You’ve got foreign keys from your student table that match to a primary key in your course table.  12:42  You can enforce that referential integrity so that every value from the student table has a match in the course table.  12:50  That’s what we mean by referential integrity. There are some more advantages.  12:59  So access controls, that’s pretty basic, user permissions.  13:05  Maybe there’s a group of folks that should only have read privileges.  13:09  There’s a group of database engineers that have read and write. And then there’s a very small select group of individuals,  13:14  the database administrators who have all three of those permissions, read, write, admin.  13:21  So that’s what we mean by access controls. Persistent storage for program objects—  13:28  so as you’re designing software, typically you’re designing classes, and you have functions inside those classes.  13:34  You have objects inside those classes. NoSQL types of databases or document models are great for storing those types of objects.  13:42  You can also do that in, you know, relational databases, storing them, you know, maybe as JSON.  13:51  If you’ve got, you know, Python dictionaries, you can convert sometimes classes to Python dictionaries depending on how you built your class.  13:59  And then you can write that JSON to PostgreS or some other database.  14:07  There’s also. Something called a blob where you can store really just anything.  14:16  So if you take a  14:23  software class and some objects in software, you could essentially write that to a blob  14:27  data store, is kind of this hashed object and then read it back to your software.  14:35  I wouldn’t recommend doing that. NoSQL types of databases,  14:40  the document models are much better for persistent storage for program objects, and that’s why they’re very popular,  14:44  NoSQL, MongoDB, CouchDB, those sorts of things for, you know, web application back ends.  14:52  There’s a lot of software developers that would prefer to go NoSQL than relational  14:59  models just because they’re much more user friendly for software development.  15:02  Backup and recovery—so if you lose an Excel book, you lost the Excel book. Right?  15:09  If you didn’t have your hard disk backed up, sorry Bud, you lost the Excel book.  15:14  So databases do this kind of automatically, or at least they should.  15:22  They should have some type of backup and recovery feature to restore previous dates in case of failures.  15:26  Big data systems like Hadoop are a little different.  15:32  So in Hadoop you have a very large cluster of servers and instead of storing backups for each server or backing up the whole system,  15:37  you have something called redundancy. So your data may live on,  15:48  so one partition of data may live on server one. It’s also going to live on typically two other servers in case that server fails.  15:53  So if that server fails, you still got two copies of that data on other servers.  16:01  So that’s how big data systems, file systems typically handle that backup and recovery. User interfaces—  16:07  if you’re storing data in Excel, you’re limited to Excel. Right?  16:15  There are all sorts of interfaces you can interact with the database with.  16:20  I like DBeaver. It’s probably my favorite.  16:25  You can connect to a variety of different databases.  16:28  I believe I’ve heard you can connect to MongoDB from DBeaver.  16:32  You can connect to Hadoop systems from DBeaver. you can connect to Oracle, PostgreS, MySQL, all the other, you know, SQL-type engines.  16:36  So DBeaver is really great, but there’s also user interfaces that are specific to the database itself.  16:44  So I think MongoDB has their own atlas in the cloud.  16:53  There’s one local, I forget the name of it. I use it every day, which is crazy. But  17:02  each database is most likely going to have their own user base as well. Relationships between tables—  17:09  so we talked about referential integrity and ensuring those, you know, one-to-one matching or one-to-many matchings,  17:17  those can be defined with the database. In Excel, you have to write some nasty Excel  17:25  function. And I don’t use Excel anymore, so I don’t know what that would look like.  17:32  But you have to do something pretty gnarly to create those relationships.  17:37  Rules and triggers—this is something I don’t think you can do with Excel very easily unless you’re writing Visual Basic, which you don’t want to do.  17:41  Rules and triggers—basically, when something in the database happens or you get some new data, you can trigger an action on something else,  17:53  like updating a view or running some type of function against your database.  18:01  Much easier in database doing that in a database than it is, you know, some other application or doing it in some type of self-managed file store.  18:09  So those are the advantages of the database management system approach,  18:19  per your textbook. I am going to create a video on indexes and partitions just to go a little bit deeper and drive that one home  18:24  because that’s probably one of the more important concepts that you’ll need to understand is how those work,  why they work, and why they’re more efficient than just scanning an entire table.  18:42  If you go to take the big data class, one of the worst things you can do is a full table scan. Right?  18:48  You can spend hours or in some cases days doing full table scans in Hadoop.  18:54  So it’s very important to understand how partitions and indexes work because when you go to the cloud, they’ll save you a lot of money.  19:00  You don’t believe me,  19:09  I know some folks who spent $2 million in one month going to the cloud because they weren’t using partitions in indexes.  They started using indexes in partitions. That $2 million they were spending a month went down to about $200,000.  So, you know, a factor of 10. So it is very important that you understand indexes and partitions.  So we’ll talk about that in another video. Let’s take a break.  So we’re about at 20 minutes, which is right around where I want to keep these.  So we will come back and talk about data models, schemas, and instances on the next video. |

## Data models, schemas, and instances

| **Notes** |
| --- |
|  |

# 