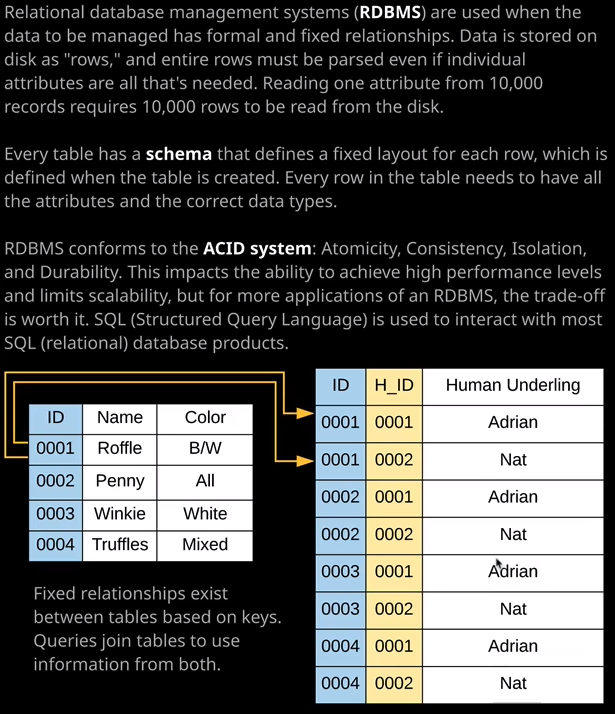
**database fundamentals**

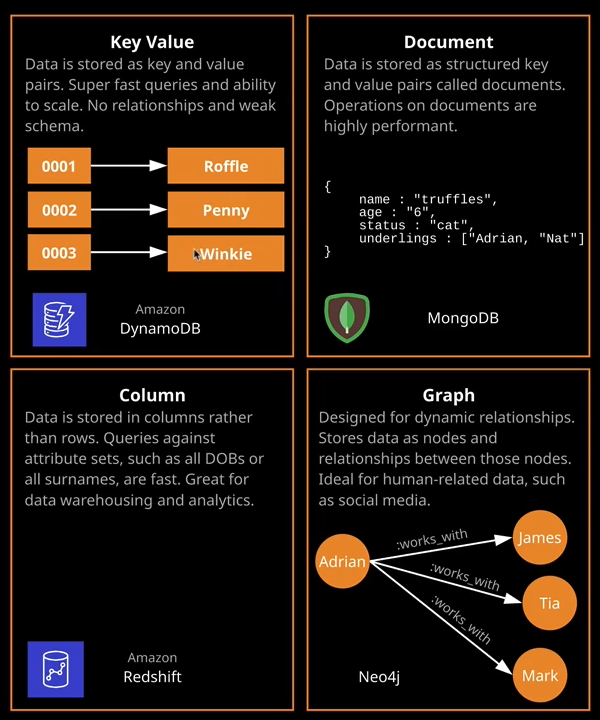


the differences between SQL and noSQL database engines. Now, before I start with that, I want to introduce a term that you might not be aware of and that's DBMS or database management system. Now a database management system, simply put, is a platform designed to manage data. A database service such as mySQL or Microsoft SQL is essentially a database management system. It contains storage, memory, CPU, and software designed to manage the storage, perform optimization, and help retrieve the data through a process known as querying. Now there are many vendors which provide DBMS products. You might have heard of a few, such as Microsoft SQL server, Oracle, mySQL, DynamoDB, MongoDB, Postgres, MariaDB, DB2, and even things such as SAP HANA. They're all examples of DBMS database management systems. Now they have some common aspects, but they all approach how to handle data and how to store data very differently and that's one of the things that I want to talk about in this lesson.

Now at a high level, **there are two main categories of database platforms in use today: relational and nonrelational and you might hear them referred to as SQL or noSQL.** It's important both for the exam and the real world usage to understand the difference together with when and where you would use each of them. So I'm going to start off by talking about relational database systems. **Relational databases are designed to store data which has strong built in relationships. An example of this might be an organizational database, which stores data on its staff members together with who works in what department. In this example, you're storing information about team members and their departments and there is a strong known relationship between both of those entities. Relational databases organize data into tables and link them together based on relationships,** and this is an example of that which we'll talk about in a second. **Now these relationships help you retrieve and combine data from one or more tables with a single query. Now relational database systems use a process called normalization. Normalization is a set of data modeling rules and this is well beyond the scope of this course, but essentially, normalization has two main functions. It eliminates any redundant data, so anything that repeats, and ensures that any data stored together is dependent on each other otherwise it separates it. Now normalization was actually developed a long time ago, when storage was the most expensive part of any data center platform. So if you consider that most data center platforms consists of servers and those servers have storage, memory, and CPU. Well, when normalization was developed, storage was the most expensive component of any of those systems, and so normalization was developed in an attempt to minimize the duplication of data on storage.** Now this has changed because now, relatively speaking, CPU is the most expensive part of any system and so there's a movement **moving away from relational database systems which do have some limitations and some issues with scaling, and that movement is moving more towards a noSQL or nonrelational set of products**, and we'll talk about that later in this lesson.

Now **in a relational database system, you need to define what's called a schema and this is a rigid definition of how data is structured, and you have to do this in advance before you put data into the database system. Now, relational database groups its data into tables, and each of these that are on screen now are examples of simple tables.** The table on the left stores details of my four cats. So we got Ruffle, Penny, Winky, and Truffles. Each table has a number of rows. In this case, it has four rows and each row is identified by a unique key value. So the key for this table, the primary key, is ID and each row in this table is identified by a unique value for that key. So we've got 0001 for Ruffle and 002 for Penny and so on. Now, tables could either have a single primary key or what's known as a composite key and a composite key, simply put, is a key that consists of multiple keys and this table on the right is an example of a table with a composite key. So it's made up by the column in blue and the column in yellow. Now the schema also stores the relationships between different tables. So in this case, the table on the right stores the details of the human underlings that for each of my individual cats. So, for example, Ruffle, which is cat ID 0001 has two underlings Adrian and Nat. The way that this is identified as part of this composite primary key is the key from another table, and this is known as a foreign key. The other half so HID or human ID ensures that for each row in this table, it's unique. So for cat 0001 we've got one human so, Adrian, with a human ID of 0001. The other row also corresponding to Ruffle so with cat ID 0001 we've got human ID 0002 which is Nat and so that's the way that we can represent repeating data. So for each cat, we've got multiple human underlings. **So with relational database platforms, you have to specify the schema in advance and the schema stores the primary key or primary keys for the table. It stores details of the tables and then for each individual table as well as the keys, it stores each attribute that belongs to that table. So every row has to have the same attributes in a relational database platform. So in this case, we've got an attribute name and an attribute color and every row has to store values both of those. In this table as well as the key values, we've got one other attribute called human underling and again, every single row in this table has to have something for the attribute as well as the same attributes. We don't have any rows which have different attributes. They're all consistent and that's a requirement in relational database systems. We have to have this predefined schema.** Now where relational databases excel is the ability to query this data. Relational databases offer a querying language called the structured query language or SQL, or as some people call it S-Q-L. This allows you to query the data inside a database management system. You could, for example, show all rows of the cat table where the IDs are equal to 0001 and that would show the row for Ruffle. You could also show the rows where the ID is between 0001 and 0003 and that would pull the information for Ruffle, Penny, and Winky. Using SQL, you could also query both tables with a join showing all the rows in the cat table with an ID of 0001 and 0002 and for both of those show any of the human underlings that are pulled from this table. So using the structured query language, you can join tables together to provide more rich data.

So in summary**, if your data has relationships, if they're fixed and they don't often change then relational databases are fast. They're mature, they're efficient, and they offer excellent performance. But it's not all good, relational databases do have some issues because the relationships are fixed, they struggle with any situations or data, which has fluid relationships.** So, for example, **they struggle with social media sites. It's hard to scale using a relational database because all of the data is related and a single platform needs to access it all ideally in-memory and so it's really difficult to separate that data.**



**Nonrelational, also known as noSQL, databases are a group of products which aim to address that. So when you hear the term noSQL, it's generally an umbrella term, which describes a number of different database engine types. We've got key value, document databases, column-based databases, and graph databases.** Now I'll quickly talk about these because you'll get some exposure to it throughout the course. We start with key value databases and these are probably the most simple type of databases available. Essentially, it's a list of keys and corresponding values, so in this case, the keys are 0001, 0002, and 0003 and they correspond to the values of Ruffle, Penny, and Winky.

**Key value databases are excellent ways to store data that doesn't have much structure, and they're often used to store usernames and passwords. They could be used to store session states, and they can also be used for in-memory caching. Now key value databases have a simple or nonexistent schema. Essentially, the only information really that's important is this key. Everything else is just arbitrary data so the value is completely up to the operator or user of the database. They're able to cope with high velocity read or write operations without suffering from performance issues and they also scale really, really well. Now DynamoDB is capable of acting somewhere in between a key value type database and a document type database** and I'll talk more about this and elaborate about DynamoDB later in this section of the course. Essentially, DynamoDB doesn't fit nicely into these categorizations because it delivers a lot of extra functionality. But essentially, an example of a database that can do key value is DynamoDB.

**Now document databases are used for storing semistructured data and they're stored as documents. So rather than having to define the structure in advance rather than requiring a schema as with the case of relational databases, with document databases you essentially store documents and documents are basically JSON structures that contain a varied amount of data with a different structure identified by a document ID and they're really useful for storing individual bits of data that relate to each other. So maybe medical records or information about orders, patient records, information about collections, or different elements of stock in a database. Wherever you've got a fairly varied set of information that you want to store and you can easily group those together into documents, then a document database is an excellent product to use and an example, I have a document database is MongoDB or Amazons Document DB. Now, document databases are really effective for certain types of use cases. So examples of this content management so you can manage bits of content, maybe blog articles or other user generated content images, comments, videos. These could be stored within documents and stored really effectively inside a document database. You can also use it for things like user profiles because user profiles tend to be varied. You want to store varying and perhaps changing information about individual data items, so that's a really effective scenario where we might use a document database.**

Moving on quickly because you don't need to know the details of this for the exam. **A column based database is actually very similar to a row based relational database. It's just that the data is stored on disk in columns, rather than rows. So imagine a given database storing the information about all students for Linux Academy. With a row based database, each student record would be stored in its entirety, and then you have the next one. So my record would be first, and it would store all of my details, and then your record would be next and all of the other Linux Academy students. With a column based database, it would store the data in a different way. All of the first names would be stored and then all of the sir names and then all of the ages and by structuring the data this way, it allows you to perform really efficient analytical type queries across large sets of data. So think about this if you want to look for patterns in the age of Linux Academy students and the location and the interests in other important attributes then by storing the data in columns, you can grab that data off the disk in a single query and interact with it in a really efficient way and so column-based databases are often used for data warehousing or other analytical type applications, and a great example is Amazon Red Shift, which is a petabyte data warehouse product**. We'll talk about that later in the course.

Now, the last type of data base engine that I want to talk about is a graph database and this is not something really that you'll need to know for the exam**. Graph databases are designed for dynamic relationships. So with graph-based database the relationships between the data of first class citizens that are perhaps in some cases more important than the data itself. So graph-based databases are generally used for things like social media sites. So when you need to track the relationships between different data elements and these relationships are not fixed. They can change over time. You don't need to define them upfront, as you do with the relational database system and this makes them ideally suited for products such a social media sites, where these relationships are constantly changing. They're fluid, and you need to interact specifically with the relationships as a primary form of data. It's not just about the data with graph databases. It's about these relationships.**

Now the reason why I've covered these at all in this lesson is that in the exam you might face questions where you need to identify a specific type of data and then look at what database product you would use to store that data. **So if you need to store data that's been housed in a traditional relational database platform, then you'd use one of the AWS relational database products such as RDS or Aurora** that we'll be talking about in later topics in this section. **If you're wanting to store key values or document style databases, then you'd look at using some of the AWS noSQL products. If you wanted to store data for analytical purposes, then you'd pick a column based database product from AWS, such as Red Shift. And then, if you're looking to store an interact with data using a graph database engine, then you'd be looking at something like Neptune,** and I'll be talking about all of these as we go through the course. But for now, the main takeaway from this lesson is an ability to understand the different models of databases. **So relational and nonrelational and then for nonrelational you need to identify these four different types of engines**, and I'll make sure that I put some links in the lesson description with additional information on each of these key types of noSQL databases.