**Key points about Data Modeling**

* **Data Organization:** The organization of the data for your applications is extremely important and makes everyone's life easier.
* **Use cases:** Having a well thought out and organized data model is critical to how that data can later be used. Queries that could have been straightforward and simple might become complicated queries if data modeling isn't well thought out.
* **Starting early:** Thinking and planning ahead will help you be successful. This is not something you want to leave until the last minute.
* **Iterative Process:** Data modeling is not a fixed process. It is iterative as new requirements and data are introduced. Having flexibility will help as new information becomes available.

**Example of Why Data Modeling is Important:**

Let's take an example from Udacity. Here, a Udacity data engineer would help structure the data so it can be used by different people within Udacity for further analysis and also shared with the learner on the website. For instance, when we want to track the students' progress within a Nanodegree program, we want to aggregate data across students and projects within a Nanodegree. In a relational database, this requires the data to be structured in ways that each student's data is tracked across all Nanodegree programs that s/he has ever enrolled in. The data also needs to track the student's progress within each of those Nanodegree programs.

The data model is critical for accurately representing each data object. For instance, a data table would track a student's progress on project submissions, i.e., whether they passed or failed a specific rubric requirement. Furthermore, the data model should ensure that a student's progress is updated and aggregated to provide an indicator of whether the student passed all the rubric requirements and successfully finished the project. Data modeling is critical to track all of these pieces of data so the tables are speaking to each other, updating the tables correctly (e.g., updating a student's progress on a project submission), and meeting defined rules (e.g., project completed when all rubric requirements are passed).

**Advantages of Using a Relational Database**

* **Flexibility for writing in SQL queries:** With SQL being the most common database query language.
* **Modeling the data not modeling queries**
* **Ability to do JOINS**
* **Ability to do aggregations and analytics**
* **Secondary Indexes available** : You have the advantage of being able to add another index to help with quick searching.
* **Smaller data volumes:** If you have a smaller data volume (and not big data) you can use a relational database for its simplicity.
* **ACID Transactions**: Allows you to meet a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, and thus maintain data integrity.
* **Easier to change to business requirements**

**ACID Transactions**

Properties of database transactions intended to guarantee validity even in the event of errors or power failures.

* **Atomicity:** The whole transaction is processed or nothing is processed. A commonly cited example of an atomic transaction is money transactions between two bank accounts. The transaction of transferring money from one account to the other is made up of two operations. First, you have to withdraw money in one account, and second you have to save the withdrawn money to the second account. An atomic transaction, i.e., when either all operations occur or nothing occurs, keeps the database in a consistent state. This ensures that if either of those two operations (withdrawing money from the 1st account or saving the money to the 2nd account) fail, the money is neither lost nor created. Source [Wikipedia](https://en.wikipedia.org/wiki/Atomicity_%28database_systems%29) for a detailed description of this example.
* **Consistency:** Only transactions that abide by constraints and rules are written into the database, otherwise the database keeps the previous state. The data should be correct across all rows and tables. Check out additional information about consistency on [Wikipedia](https://en.wikipedia.org/wiki/Consistency_%28database_systems%29).
* **Isolation:** Transactions are processed independently and securely, order does not matter. A low level of isolation enables many users to access the data simultaneously, however this also increases the possibilities of concurrency effects (e.g., dirty reads or lost updates). On the other hand, a high level of isolation reduces these chances of concurrency effects, but also uses more system resources and transactions blocking each other. Source: [Wikipedia](https://en.wikipedia.org/wiki/Isolation_%28database_systems%29)
* **Durability:** Completed transactions are saved to database even in cases of system failure. A commonly cited example includes tracking flight seat bookings. So once the flight booking records a confirmed seat booking, the seat remains booked even if a system failure occurs. Source: [Wikipedia](https://en.wikipedia.org/wiki/ACID).

**When Not to Use a Relational Database**

* **Have large amounts of data:** Relational Databases are not distributed databases and because of this they can only scale vertically by adding more storage in the machine itself. You are limited by how much you can scale and how much data you can store on one machine. You cannot add more machines like you can in NoSQL databases.
* **Need to be able to store different data type formats:** Relational databases are not designed to handle unstructured data.
* **Need high throughput -- fast reads:** While ACID transactions bring benefits, they also slow down the process of reading and writing data. If you need very fast reads and writes, using a relational database may not suit your needs.
* **Need a flexible schema:** Flexible schema can allow for columns to be added that do not have to be used by every row, saving disk space.
* **Need high availability:** The fact that relational databases are not distributed (and even when they are, they have a coordinator/worker architecture), they have a single point of failure. When that database goes down, a fail-over to a backup system occurs and takes time.
* **Need horizontal scalability:** Horizontal scalability is the ability to add more machines or nodes to a system to increase performance and space for data.

#### **Common Questions:**

**What type of companies use Apache Cassandra?**  
All kinds of companies. For example, Uber uses Apache Cassandra for their entire backend. Netflix uses Apache Cassandra to serve all their videos to customers. Good use cases for NoSQL (and more specifically Apache Cassandra) are :

1. Transaction logging (retail, health care)
2. Internet of Things (IoT)
3. Time series data
4. Any workload that is heavy on writes to the database (since Apache Cassandra is optimized for writes).

**Would Apache Cassandra be a hindrance for my analytics work? If yes, why?**  
Yes, if you are trying to do analysis, such as using GROUP BY statements. Since Apache Cassandra requires data modeling based on the query you want, you can't do ad-hoc queries. However you can add clustering columns into your data model and create new tables.

**When to use a NoSQL Database**

* **Need to be able to store different data type formats**: NoSQL was also created to handle different data configurations: structured, semi-structured, and unstructured data. JSON, XML documents can all be handled easily with NoSQL.
* **Large amounts of data**: Relational Databases are not distributed databases and because of this they can only scale vertically by adding more storage in the machine itself. NoSQL databases were created to be able to be horizontally scalable. The more servers/systems you add to the database the more data that can be hosted with high availability and low latency (fast reads and writes).
* **Need horizontal scalability**: Horizontal scalability is the ability to add more machines or nodes to a system to increase performance and space for data
* **Need high throughput**: While ACID transactions bring benefits they also slow down the process of reading and writing data. If you need very fast reads and writes using a relational database may not suit your needs.
* **Need a flexible schema**: Flexible schema can allow for columns to be added that do not have to be used by every row, saving disk space.
* **Need high availability**: Relational databases have a single point of failure. When that database goes down, a failover to a backup system must happen and takes time.

NEXT

### When NOT to use a NoSQL Database?

* **When you have a small dataset**: NoSQL databases were made for big datasets not small datasets and while it works it wasn’t created for that.
* **When you need ACID Transactions**: If you need a consistent database with ACID transactions, then most NoSQL databases will not be able to serve this need. NoSQL database are eventually consistent and do not provide ACID transactions. However, there are exceptions to it. Some non-relational databases like MongoDB can support ACID transactions.
* **When you need the ability to do JOINS across tables**: NoSQL does not allow the ability to do JOINS. This is not allowed as this will result in full table scans.
* **If you want to be able to do aggregations and analytics**
* **If you have changing business requirements**: Ad-hoc queries are possible but difficult as the data model was done to fix particular queries
* **If your queries are not available and you need the flexibility**: You need your queries in advance. If those are not available or you will need to be able to have flexibility on how you query your data you might need to stick with a relational database

### Caveats to NoSQL and ACID Transactions

There are some NoSQL databases that offer some form of ACID transaction. As of v4.0, MongoDB added multi-document ACID transactions within a single replica set. With their later version, v4.2, they have added multi-document ACID transactions in a sharded/partitioned deployment.

* Check out this documentation from MongoDB on [**multi-document ACID transactions**](https://www.mongodb.com/collateral/mongodb-multi-document-acid-transactions)
* Here is another link documenting [**MongoDB's ability to handle ACID transactions**](https://www.mongodb.com/blog/post/mongodb-multi-document-acid-transactions-general-availability)

Another example of a NoSQL database supporting ACID transactions is MarkLogic.

* Check out this link from their [**blog**](https://www.marklogic.com/blog/how-marklogic-supports-acid-transactions/) that offers ACID transactions.