

Introduction to Databases And Concepts

David Liou

Table of Contents

1. [What Is a Database?](#) 10m
2. [Data Modeling: Data As Entities](#) 25m
3. [Normalization](#) 35m
4. [Ternary Logic](#) 25m
5. [Database Predicates](#) 35m
6. [CAP Theorem](#) 35m
7. [NoSQL](#) 15m

What Is a Database?

A database management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Most database systems seek to mitigate the problems of:

- **Data consistency** - interrelated sets of data may become inconsistent when updates result in the separate components no longer agree
- **Difficulties with accessing data** - modern database systems offer querying languages that allow users to succinctly specify data to be retrieved
- **Data isolation** - information not stored in database systems tend to become scattered in different locations and formats
- **Concurrent access** - modern database systems allow concurrent access to stored information.
- **Security** - databases allow administrators to specify rules regarding who and what may be viewed and modified

Perhaps the most important feature of modern databases is the flexibility it offers for data storage.

Administrators must be able to structure the data in ways that make sense to their business. This process is called **data modeling**.

Data Modeling: Data As Entities

Introduction

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world **entities**. The following is a brief introduction to relational data modeling.

Table

A table is a collection of related data held in a structured format within a database. Each table may be interpreted as a collection of homogeneous entities.

Table Row

A row also called a record or tuple, represents a single, structured data item in a table. Each row may be viewed as a single entity.

Table Column

A table data structure is specified as a list of column definitions, each of which specifies a unique column name and the type of the values that are permitted for that column. A column is a set of data values, one for each row of the table. As such, the columns may be construed as attributes for each entity held by the table.

Let's consider a data table with study participant information:




ParticipantId	ParticipantSex	ParticipantAge
101	M	29
102	F	31
103	M	23

Each row is a representation of an entity. In our example, the entity is a study participant. The columns are attributes for each participant, in this case, sex and age.

Also, notice that each column conforms to a specific data type; the ParticipantId is an integer, ParticipantSex is a character, and ParticipantAge is an integer. The columns also serve to constrain the allowable data type for each entity attribute.

Database Diagrams

Below is a representation of a table as an entity and its column attributes. This part of a **entity-relationship diagram**:




Participants
 ParticipantId: INTEGER
 ParticipantSex: CHARACTER(1)
 ParticipantAge: INTEGER

Primary Key

A primary key is a column, in some cases several columns, which uniquely identifies a row.

In our example dataset, the ParticipantId is designed to be a unique identifier for each participant. The ParticipantId guarantees that we can always specify a single participant entity unambiguously.

In entity-relationship diagrams the primary key is often designated with a key icon:

Participants
 ParticipantId: INTEGER
 ParticipantSex: CHARACTER(1)
 ParticipantAge: INTEGER

Foreign Key

A foreign key is a field (or collection of fields) in one table that uniquely identifies a row of another table or the same table. In simpler words, the foreign key is defined in a second table, but it refers to the primary key in the first table.

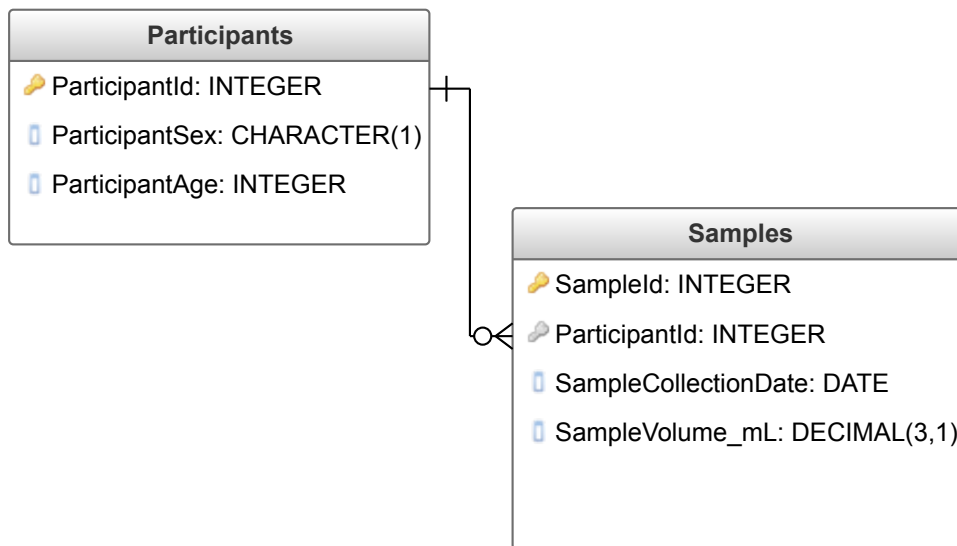
The foreign key establishes a connection to the table with the primary key called a table relationship.

Let's expand our previous example to include samples drawn from the study participants.

ParticipantId	ParticipantSex	ParticipantAge	SampleId	SampleCollectionDate	SampleVolume_mL
101	M	29	1001	4-Apr-2017	10.2
101	M	29	1002	5-Apr-2017	11.5
102	F	31	1003	6-Apr-2017	11.3
102	F	31	1004	7-Apr-2017	10.5
103	M	23	1005	8-Apr-2017	10.8
103	M	23	1006	9-Apr-2017	11.0

We are now dealing with two different entities, *participants* and *samples*. Additionally, there is a conceptual relationship between each participant and the samples taken from that participant.

Expanding our entity-relationship diagram to include the sample entity looks like:

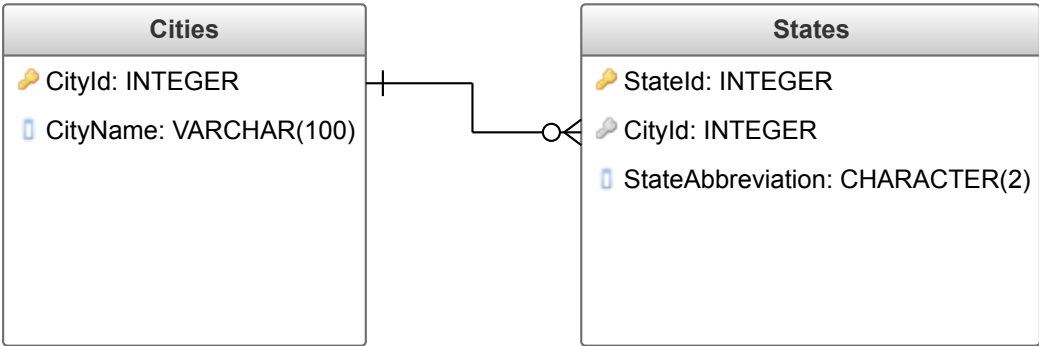
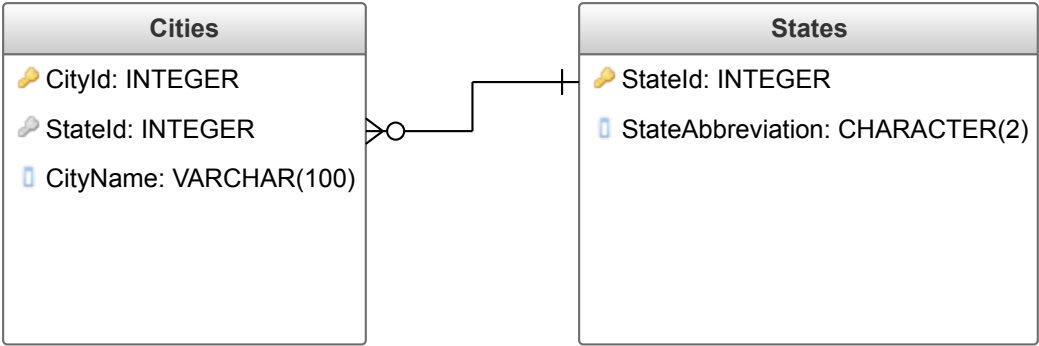


Entity-relationship diagrams represent table relationships with the connector $\text{---}\text{+}\text{---}\text{O}\text{---}\text{<}$, also called a **crow's foot** due to its reminiscently avian claw-like semblance.

The + end identifies the parent, whereas the $\text{O}\text{---}\text{<}$ is attached to the child. In our example, a study participant is connected to samples. The connector depicts a relationship between a sample and the participant from which it was taken.

Discussion

- Which of the following diagrams is a correct representation of the relationship between US cities and states? How should each representation below be interpreted?



- Consider how each entity-relationship diagram would handle modifications. For example, in 1891 the North Carolina town of Hamburg changed its name to Glenville. What would happen in each relationship above?

City	State
Hamburg	MI
Hamburg	MN
Hamburg	NC
Hamburg	NJ
Hamburg	NY
Hamburg	PA

Conclusion

Tables may be viewed as a collection of homogeneous data entities. Each row is a specific entity instance with their attributes represented by the columns. The ability to establish relationships between tables creates a dimension in our data that replaces the flat spreadsheet model. The process of translating flat data to tables with relationships is called normalization.

Normalization

Introduction

Database normalization is the process used to organize a database into tables and columns. The idea is that a table should be about a specific topic and that only those columns which support that topic are included. For example, a spreadsheet containing information about scientific projects serves several purposes:

- List studies conducted by research organizations
- Identify pertinent keywords and search terms for individual projects
- Name the lead researcher(s) for each effort

By limiting a table to one purpose you reduce the number of duplicate data that is contained within your database, which helps eliminate some issues stemming from database modifications. To assist in achieving these objectives, some rules for table organization have been developed. The stages of organizing are called normal forms; there are three normal forms most databases adhere to using. As tables satisfy each successive normalization form, they become less prone to database modification anomalies and more focused toward a sole purpose or topic.

Reasons for Normalization

There are three main reasons to normalize a database. The first is to minimize duplicate data, the second is to minimize or avoid data modification issues, and the third is to simplify queries. As we go through the various states of normalization we'll discuss how each form addresses these issues, but to start, let's look at some data which hasn't been normalized and discuss some potential pitfalls. Once these are understood, it's easier to appreciate the reason to normalize the data.

Consider the following table downloaded from the [NIH Research Portfolio site¹](#):

ProjectId	ProjectTitle	Keywords	StartDate	Organization	City	State	District
9001	Developing reciprocal chromosomal translocations for wild population replacement ...	breeding; disorder prevention; fitness; insect genetics; mediating; population genetics; positioning attribute; structure; zika	2016-12-19	Univ of California Riverside	Riverside	CA	41
9002	Zika virus evolutionary dynamics in host adaptation	biological; disease; disease outbreaks; event; laboratories; laboratory experiment; locales; methods; mosquito control; neurologic; perinatal; // zika	2017-02-07	2019-01-31	Univ // zika	2017-02-07	Univ // of Wisconsin-Madison
9006	A Rapid and Specific Diagnostic for Immunoglobulin Response to Zika Virus Exposure ...	capsid proteins; color; development; diagnostic reagent; fetus; lateral; link; paper; pathogen; point of care; staging; zika	2016-08-20	University of Washington	Seattle	WA	7
9007	Discovering host factors impacting ZIKV infection via forward genetic screens	andes virus; funding; human genome; insertional mutagenesis; link; pathogen; reporting; response; venezuela; viral; west nile virus; zika	2017-01-01	University of Pennsylvania	Philadelphia	PA	2
9008	Molecular and Antibody Detection of Zika Virus in Saliva at the Point of Care	design; development; heating; infection; laboratory facility; neurologic; pennsylvania; point-of-care diagnostics; process; seasons; tool; zika	2016-09-09	University of Pennsylvania	Philadelphia	PA	2
9009	Zika virus in the human genital tract and implications for transmission	body fluids; care seeking; case study; diagnostic; epidemic; left; nicaraguan; novel; pregnancy; reporting; vero cells; zika	2017-02-15	Univ of North Carolina Chapel Hill	Chapel Hill	NC	4
9133	Mechanisms of sexual Zika virus transmission and early immunopathogenesis	cell type; dendritic cells; immune; injection of therapeutic agent; insight; publishing; risk; sexually transmitted diseases; viral transmission; virus replication; world health organization; zika	2016-12-01	University of Washington	Seattle	WA	7
9085	A New Filter Paper Technology for Flavivirus Collection, Shipping, and Analysis	award; chemicals; collection; development; dna; eligibility determination; formulation; link; mutate; new technology; small business innovation research grant; zika	2017-03-08	GenTegra, LLC	Pleasanton	CA	15

Column Descriptions

- **ProjectId:** A unique number assigned to each project number.
- **ProjectTitle:** The grantee submitted title descriptive to the project.
- **Keywords:** Text used for site searches taken from project titles, abstracts, and scientific terms.
- **StartDate:** The date the project began.
- **Organization:** A generic term used to refer to an educational institution or other entity, including an individual, which applies for or receives an NIH grant, contract, or cooperative agreement.
- **City:** The business address city of the grantee organization.
- **State:** The business address state of the grantee organization.
- **District:** Congressional District: The congressional district to which each grant is assigned is based on the business address of the grantee organization.

Data Duplication and Modification Anomalies

Notice that for each research organization, the primary city and state for that entity is also listed. This information is duplicated for each organization.

Duplicated information presents two problems:

1. It increases storage
2. Duplication decreases performance
3. Enacting changes is more complicated

Additionally, let's suppose the census necessitates redistricting of some cities requiring congressional district numbering changes within the dataset. For large datasets, this would involve an extensive number of individual updates.

This situation is an example of a modification anomaly. There are three modification anomalies that can occur:

1. Insert Anomalies
2. Update Anomalies
3. Deletion Anomalies

These anomalies can be mitigated by apportioning the data into separating tables by function and domain. This process is called normalization. Normalization is assessed by a progressive series of criteria called normal forms.

Definition of Normalization

There are three common forms of normalization: 1st, 2nd, and 3rd normal form. The forms are progressive, meaning that to qualify for 3rd normal form a table must first satisfy the rules for 2nd normal form, and 2nd normal form must adhere to those for 1st normal form.

1. **First Normal Form** – The information is stored in a relational table and each column contains atomic values and there are no repeating groups of columns.
2. **Second Normal Form** – The table is in first normal form and all columns depend on the table's primary key.
3. **Third Normal Form** – The table is in second normal form and all of its columns are not transitively dependent on the primary key.

1NF – First Normal Form

The first steps to making a proper SQL table is to ensure the information is in first normal form. Once a table is in the first normal form it is easier to search, filter, and sort the information. The rules to satisfy first normal form, abbreviated 1NF, are:

- The table stores information in rows and columns where one or more columns, called the primary key, uniquely identify each row.
- Each column contains atomic values and there are not repeating groups of columns.

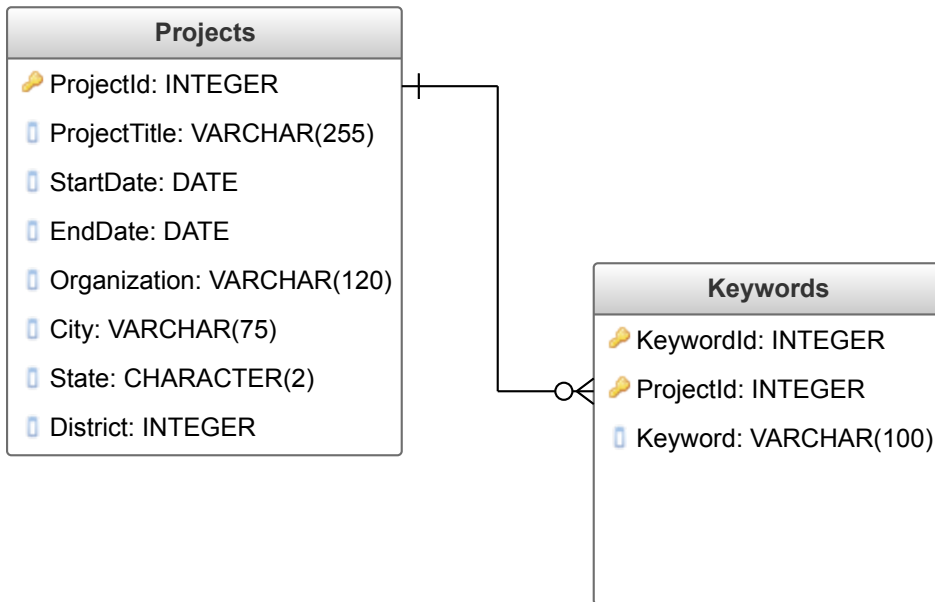
Atomic values are distinct terms that cannot be further subdivided. For example, values should not be concatenated into a single string separated by a delimiter such as a semi-colon (;). The text “*zika*” is atomic; whereas a concatenation of terms such as “*zika; vector; transmission*” is not. The keywords column of the NIH Research Portfolio example table contains non-atomic values.

Related to this requirement is the concept that a table should not contain repeating groups of columns which are functionally equivalent to concatenating several values with a delimiter. Extending on the keywords example, simply splitting “*zika; vector; transmission*” into separate columns called *Keyword1*, *Keyword2*, and *Keyword3* still violates 1NF.

Projects	
ProjectId: INTEGER	
ProjectTitle: VARCHAR(255)	
Keywords: VARCHAR(500)	
StartDate: DATE	
Organization: VARCHAR(125)	
City: VARCHAR(75)	
State: CHARACTER(2)	
District: INTEGER	

The Keywords column holds a list of values separated with a semicolon, e.g. “*transmission;vector;zika*”. This is non-atomic and therefore violates **1NF**.

Our example table is transformed to **1NF** by placing the individual Keywords values into their own table.



The concatenated values are split and become separate rows in the Keywords table linked to Projects. The data is now 1NF.

This design is improved in several ways:

- The original table limited the maximum length of the concatenated keywords text to 500 characters. In the new design, the possible number of keywords associated with each project is unlimited.
- It is straightforward to sort and filter by keyword values. Instead of having to read and parse the concatenated keywords text, it's now possible to directly match values in the Keywords table.
- Potential modification anomalies for Projects have been reduced. Previously, modifying any single keyword would entail updating the entire string of keywords. Now updates to specific keywords do not disturb other non-targeted keywords.

2NF – Second Normal Form

A table is in 2nd Normal Form or **2NF** if:

1. The table is 1NF
2. All the value (non-key) columns are directly dependent on all keys belonging to the table

The primary key serves to uniquely identify each row in a table. When all the columns directly relate to the primary key, they naturally share a common purpose. A table is in **2NF** if it solely serves a single entity. For the Keywords table, the purpose is to track user-submitted keywords.

An equivalent way of stating the second requirement is: a table infracts **2NF** if there is a value column not dependent on at least one of the table keys. An important consequence of this criteria is that **2NF** only applies to tables with multiple keys. **If a table is 1NF and only has a single key, it automatically satisfies the requirements for 2NF.**

Issues With Our Example

In our example, the only possible violation of **2NF** can occur with our Keywords table. The only other table, Projects, has a single primary key.

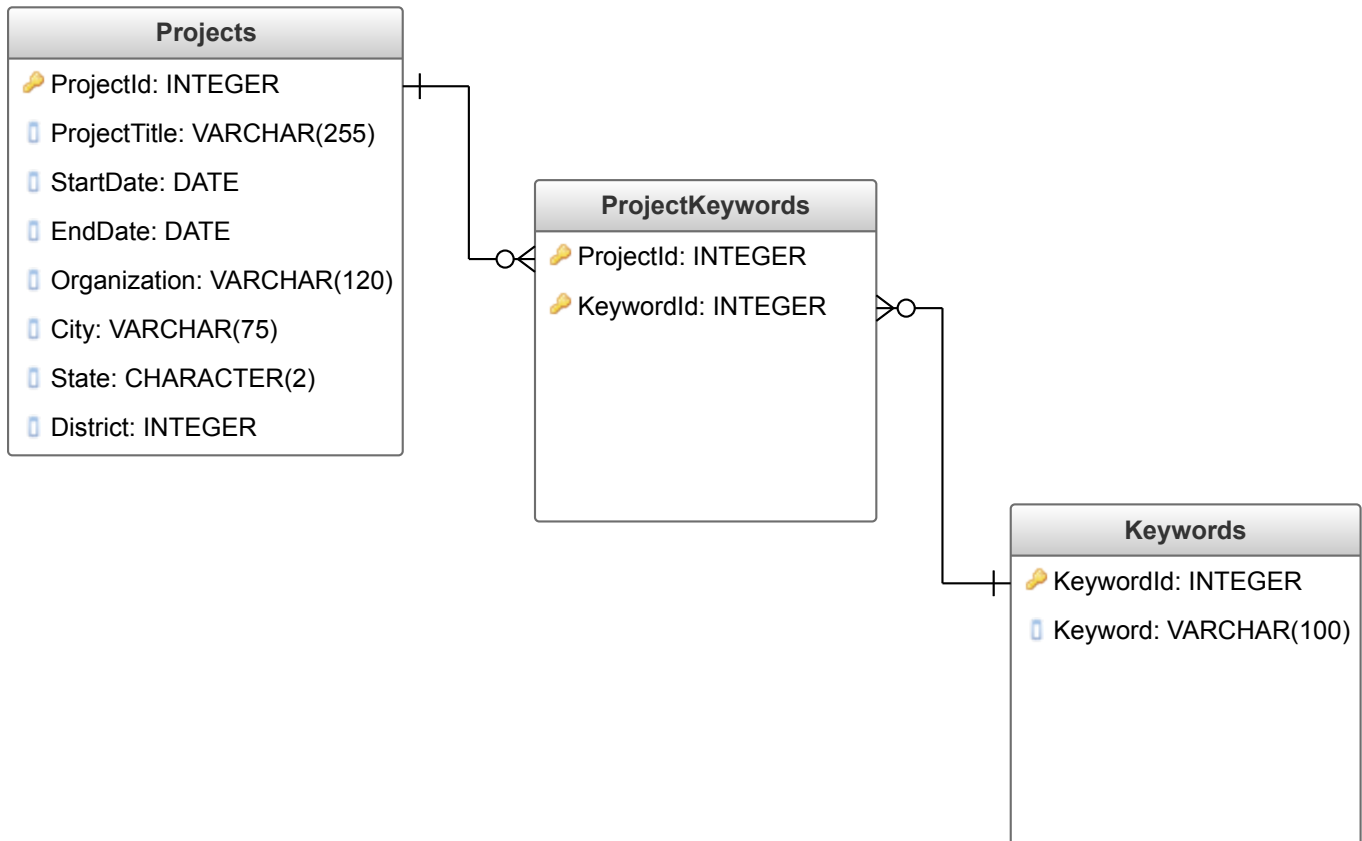
The Keywords table, the way it is, has two separate functions. the first role is to hold the actual value. The second is to track of the project to which it belongs. Additionally, the Keyword value doesn't conceptually seem to be dependent on the ProjectId. The Keywords table violates 2NF.

Adjusted to 2NF Standards

Our problems are

1. The Keywords table is maintaining occurrences in projects
2. The keyword value isn't dependent on the ProjectId key

Since the Keywords column isn't directly dependent on the ProjectId key, it is a reasonable strategy fix this by moving the ProjectId field somewhere else. We also have to address the issue with the Keywords having an extraneous function. Creating a new table appeals to both of these needs.



3NF – Third Normal Form

A table is in third normal form, **3NF**, if:

1. It satisfies **2NF**
2. It contains only columns that are non-transitively dependent on the primary key

What It Means to Be Transitive

When something is transitive, then a meaning or relationship is the same in the middle as it is on the whole. If it helps think of the prefix trans as meaning “across.” When something is transitive, then if something applies from the beginning to the end, it also applies from the middle to the end.

Since ten is greater than five, and five is greater than three, you can infer that ten is greater than three.

In this case, the greater than comparison is transitive. In general, if A is greater than B, and B is greater than C, then it follows that A is greater than C.

If you’re having a hard time wrapping your head around “transitive” I think for our purpose it is safe to think “through” as we’ll be reviewing to see how one column in a table may be related to others, through a second column.

Dependence

An object has a dependency on another object when it relies upon it. In the case of databases, when we say that a column has a dependence on another column, we mean that the value can be derived from the other. For example, my age is dependent on my birthday. Dependence also plays an important role in the definition of the second normal form.

Transitive Dependence

In mathematics and economics, the Axiom of Transitivity is formally described as:

If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

If X defines Y , and Y defines Z , then X (also) defines Z

Similarly, with databases, given three columns A, B, and C:

If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

If A is defined by B, and B is defined by C, then A and C are transitively dependent.

Transitive Dependence Applied to Our Example

Let's examine the first entry of our example data model, specifically the ProjectId, Organization, City, State, and (Congressional) District:

ProjectId	...	Organization	City	State	District
9001	...	Univ of California Riverside	Riverside	CA	41

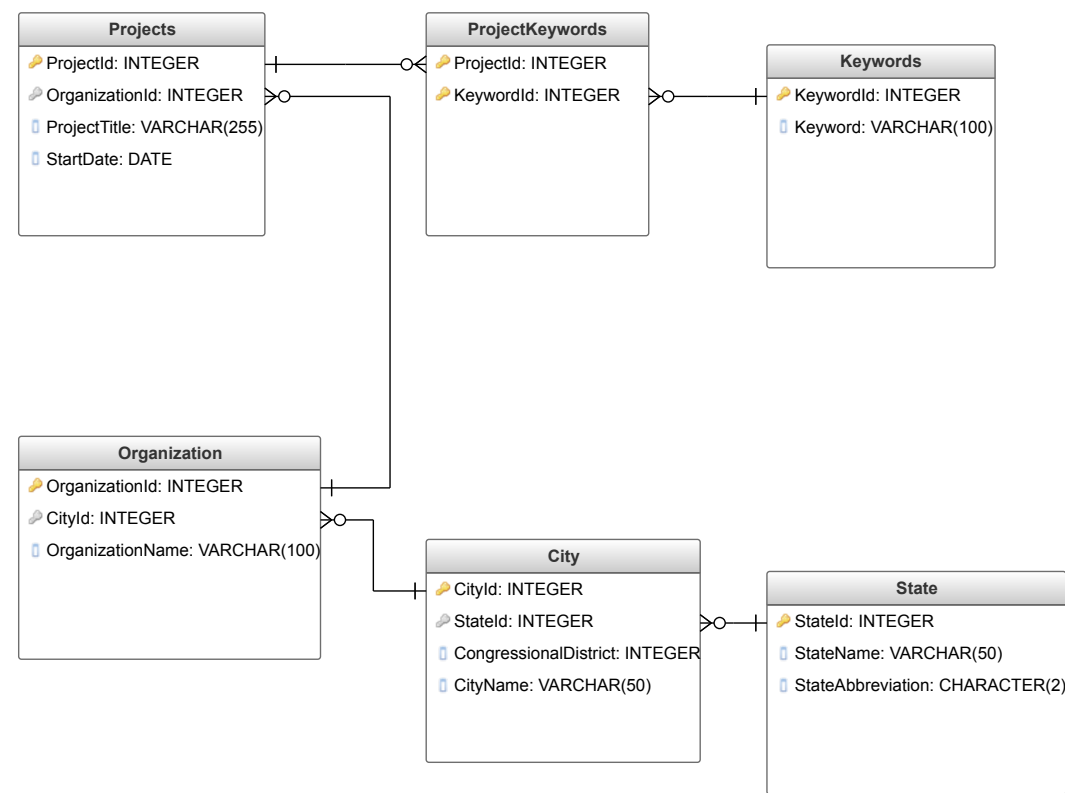
It is reasonable to state,

"The project is dependent on the University of California Riverside which, in turn, is defined by the city Riverside. Therefore the ProjectId is dependent on the city of Riverside."

The columns ProjectId, Organization, and City are parts of a transitive dependency.

Are there other transitive dependencies that exist between these columns?

Adjusted to 3NF Standards



Excercise

ProjectId	ProjectTitle	Keywords	StartDate	Organization	City	State	District	PI	OtherPI
9001	Developing reciprocal chromosomal translocations for wild population replacement ...	breeding; disorder prevention; fitness; insect genetics; mediating; population genetics; positioning attribute; structure; zika	2016-12-19	Univ of California Riverside	Riverside	CA	41	Akbari, Omar	
9002	Zika virus evolutionary dynamics in host adaptation	biological; disease; disease outbreaks; event; laboratories; laboratory experiment; locales; methods; mosquito control; neurologic; perinatal; zika	2017-02-07	Univ of Wisconsin-Madison	Madison	WI	2	Aliota, Matthew	
9006	A Rapid and Specific Diagnostic for Immunoglobulin Response to Zika Virus Exposure ...	capsid proteins; color; development; diagnostic reagent; fetus; lateral; link; paper; pathogen; point of care; staging; zika	2016-08-20	University of Washington	Seattle	WA	7	Baker, David	Yager, Paul
9007	Discovering host factors impacting ZIKV infection via forward genetic screens	andes virus; human genome; insertional mutagenesis; pathogen; response; venezuela; viral; west nile; zika	2017-01-01	University of Pennsylvania	Philadelphia	PA	2	Bates, Paul	
9008	Molecular and Antibody Detection of Zika Virus in Saliva at the Point of Care	design; development; heating; infection; laboratory facility; neurologic; pennsylvania; point-of-care diagnostics; tool; zika	2016-09-09	University of Pennsylvania	Philadelphia	PA	2	Bau, Haim	
9009	Zika virus in the human genital tract and implications for transmission	care seeking; case study; epidemic; nicaraguan; novel; pregnancy; vero cells; zika	2017-02-15	Univ of North Carolina Chapel Hill	Chapel Hill	NC	4	Becker-Dreps, Sylvia	Bucardo, Filemon
9133	Mechanisms of sexual Zika virus transmission and early immunopathogenesis	cell type; dendritic cells; injection of therapeutic agent; transmitted diseases; viral transmission; virus replication; zika	2016-12-01	University of Washington	Seattle	WA	7	Vojtech, Lucia	
9085	A New Filter Paper Technology for Flavivirus Collection, Shipping, and Analysis	chemicals; collection; dna; eligibility determination; formulation; mutate; new technology; small business innovation research grant; zika	2017-03-08	GenTegra, LLC	Pleasanton	CA	15	Nasarabadi, Shanavaz	Hogan, Michael; Langenbach, Kurt

Column Descriptions

- **ProjectId**: A unique number assigned to each project number.
- **ProjectTitle**: The grantee submitted title descriptive to the project. **Keywords**: Text used for site searches taken from project titles, abstracts, and scientific terms.
- **StartDate**: The date the project began
- **PI & OtherPI**: Individual(s) designated by the grantee to direct the project or activity being supported by the grant. He or she is responsible and accountable to the grantee and NIH for the proper conduct of the project or activity. Also known as Program Director or Project Director.
- **Organization**: A generic term used to refer to an educational institution or other entity, including an individual, which applies for or receives an NIH grant, contract, or cooperative agreement.
- **City**: The business address city of the grantee organization.
- **State**: The business address state of the grantee organization.
- **District**: Congressional District: The congressional district to which each grant is assigned is based on the business address of the grantee organization.

The table above is our example table with two additional columns, PI and OtherPI.

1. Modify the diagram created for the original table to include PI and OtherPI while observing the three normal forms.
2. Let's suppose, a policy to discourage usage of the term 'swine flu' mandates that all public sites within your organization's purview replace instances of this phrase with 'Influenza A (H1N1)'. Given that your database is properly normalized, how would you remove and replace all keyword entries of "swine flu" with 'Influenza A (H1N1)'? Let's assume that there originally is no 'Influenza A (H1N1)' entry; you don't need to worry about merging existing occurrences.

Conclusion

"Colorless green ideas sleep furiously"

-- Noam Chomsky

"Colorless green ideas sleep furiously" is an example by the linguist Noam Chomsky of a sentence that is grammatically correct, but semantically nonsensical.² Similarly, a formulaic application of normal forms is likely to result in the same outcome; a database design that is technically and structurally sound but is senseless in the context of the subject matter to which it is applied and information it houses.

The cardinal rule of database design, and arguably of any engineering effort, is form should naturally reflect the subject for which it is intended. An intuitive design, elegance, and simplicity are always prized above technical configuration.

I believe the forms presented in this section should be used as the criterion rather than procedure. Planning is an iterative effort. Each cycle begins with an intuitive design and ends with a technical scrutinization. When conflicts are discovered the cycle should begin again with a revisit to the relevant segment.

Ternary Logic

Introduction

Consider the question "Is it raining on Easter Island?"

Chances are most people cannot immediately answer this question with a definitive yes or no (especially without the use of Google). And, as a practical truth, the collection of information sets are never 100% complete.

Modern databases take this into account through the use of **ternary logic**.

Three-valued logic

In logic, **ternary** logic, also called **three-valued** logic is any of several many-valued logic systems in which there are three truth values indicating *true*, *false* and a third value representing an *unknown* result.

This is contrasted with the more commonly known boolean logic which provides only for *true* and *false* values and results.

Truth Tables

Truth tables are used to summarize the results for expressions in logic. It codifies the outcomes for all legitimate input values. Most people are familiar with the boolean logic tables. Ternary logic tables differ only in that they include the third *unknown* value.

The most common logical operations³ are *not*, *and*, & *or*.

A	¬A	NOT A
T	F	If A is <i>true</i> then NOT A is <i>false</i>
F	T	If A is <i>false</i> then NOT A is <i>true</i>
U	U	If A is <i>unknown</i> (not known to be true or false), then NOT A is (still) <i>unknown</i>

A ∩ B		B			A AND B		
		T	F	U			
A	T	T	F	U	If A is <i>true</i> and B is <i>true</i> , then A AND B is <i>true</i>	If A is <i>true</i> and B is <i>false</i> , then A AND B is <i>false</i>	If A is <i>true</i> and B is <i>unknown</i> (not known to be true or false), then A AND B is <i>unknown</i>
	F	F	F	F	If A is <i>false</i> and B is <i>true</i> , then A AND B is <i>false</i>	If A is <i>false</i> and B is <i>false</i> , then A AND B is <i>false</i>	If A is <i>false</i> and B is <i>unknown</i> (not known to be true or false), then A AND B is <i>false</i>
	U	U	F	U	If A is <i>unknown</i> (not known to be true or false) and B is <i>true</i> , then A AND B is <i>unknown</i>	If A is <i>unknown</i> (not known to be true or false) and B is <i>false</i> , then A AND B is <i>false</i>	If both A and B are unknown <i>unknown</i> (not known to be true or false), then A AND B is <i>unknown</i>

A ∪ B		B			A OR B		
		T	F	U			
A	T	T	T	T	If A is <i>true</i> and B is <i>true</i> , then A OR B is <i>true</i>	If A is <i>true</i> and B is <i>false</i> , then A OR B is <i>true</i>	If A is <i>true</i> and B is <i>unknown</i> (not known to be true or false), then A OR B is <i>true</i>
	F	T	F	U	If A is <i>false</i> and B is <i>true</i> , then A OR B is <i>true</i>	If A is <i>false</i> and B is <i>false</i> , then A OR B is <i>false</i>	If A is <i>false</i> and B is <i>unknown</i> (not known to be true or false), then A OR B is <i>unknown</i>
	U	T	U	U	If A is <i>unknown</i> (not known to be true or false) and B is <i>true</i> , then A OR B is <i>true</i>	If A is <i>unknown</i> (not known to be true or false) and B is <i>false</i> , then A OR B is <i>unknown</i>	If both A and B are unknown <i>unknown</i> (not known to be true or false), then A OR B is <i>unknown</i>

Couple of Thought Experiments

false AND unknown = false

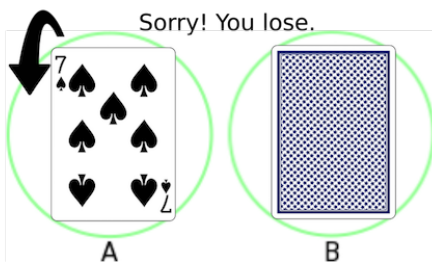
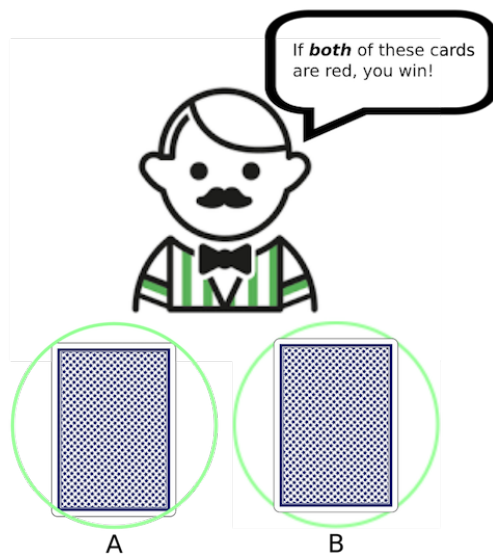
true OR unknown = true

Most of the ternary truth table outcomes can be accepted superficially. However, there are a couple of assertions which may not be clear at first glance. Specifically, why are we able to derive a known result from **false AND unknown** and **true OR unknown**. Instinctively, it would seem that any equation with an unknown value would preclude a definite result.

How can we add unknown to something and get a known result?

Why does False AND Unknown Equal False?

Let's suppose we're in Las Vegas playing at a card table. The rules are simple. The player places a bet. The dealer shuffles a deck of cards, then deals two cards on the table. If both cards are red, either the suit of hearts or diamonds, then the player wins and doubles his or her wager amount. Otherwise, the player loses the bet placed.

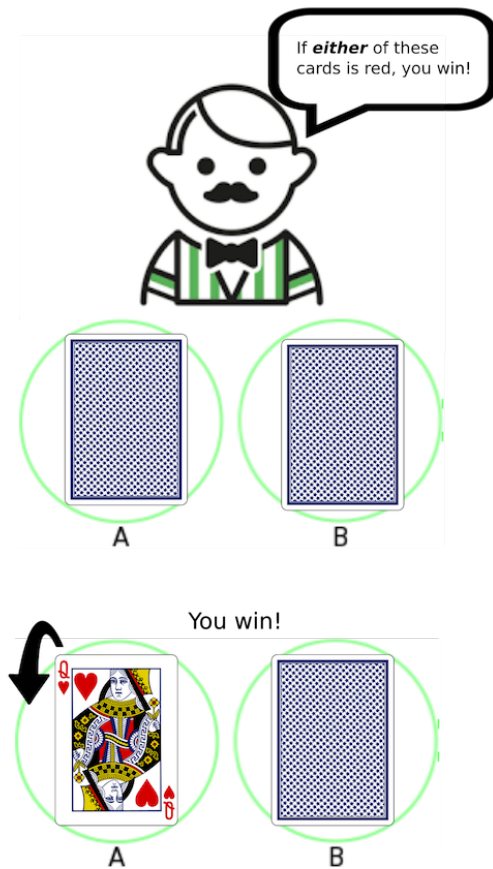


Even though the color of the second card is still unknown, we know we've lost. This is because the facedown card will not change the outcome.

This is why False AND Unknown must evaluate to False.

Why does True OR Unknown Equal True?

After losing at this game, we decide to try our luck at a different game. We find a game that is similar to the first, but the rules are slightly changed. The player is dealt two cards, but in this game, the player wins if *either* card is red.



The color of the second doesn't need to be known to determine the outcome. This is similar to the prior game, but with a happier outcome.

This is the result of True OR Unknown evaluating to True.

Discussion

Discuss how these games might be adjusted to illustrate the statements:

- If A is unknown and B is true, then A OR B is true
- If A is unknown and B is false, then A AND B is false

Conclusion

Ternary logic may at first feel academic. However, familiarity with these principles is essential to both the design and use of databases. It is important to understand that ternary logic seeks to more accurately reflect real-world situations under which data is stored, queried, and interpreted.

Database Predicates

Introduction

Familiarity with the principles of ternary logic is essential to both the design and use of databases. It is important to understand that ternary logic seeks to more accurately reflect real world situations under which data is collected.

Consider the following item asked in standard United States blood donor eligibility questionnaires:

From January 1, 1980, through December 31, 1996, have you spent (visited or lived) a cumulative time of 3 months or more, in the United Kingdom (UK)?

☐ Yes ☐ No

This example illustrates the potential of a true/false question with three possible outcomes. The first two possible values are fairly obvious; the result may be true or false depending on the selected option. But what if the question was left unanswered? How would this be indicated in the database? This is where the ternary value **unknown** becomes applicable.

The three possible ways to record how the example question was answered are:

1. **true** - Yes, I spent 3 months or more in the UK from Jan 1980 to Dec 1996.
2. **false** - No, I did not spend the specified amount of time within the specified date range in the UK
3. **unknown** - if the answer is ambiguous and cannot be determined as *true* or *false*, e.g. neither option was checked

It is important to note that it is common practice for software developers and database administrators to avoid unknown (also known as null) values by using appropriate default values. However, default values are highly context specific therefore applied by explicit declaration by individuals with subject matter expertise.

In this section we will apply ternary logic to the evaluation database selection statements also known as **predicates**.

Excercise: Clinical Report Form

Assessment of Severe Malaria (For children only)

A diagnosis of severe malaria in children is typically based on a polythetic type classification system. Polythetic refers to the fact that a diagnosis is defined by multiple symptoms, and not all listed symptoms are necessary classify an individual with the severe malaria criteria. Rather, it is sufficient that a number of symptoms must be observed to consider the diagnosis present.

	Clinical Presentations	Yes	No
1.	Can the child sit unaided?	[]	[]
2.	Is the child fitting now?	[]	[]
3.	Is the child jaundiced (i.e., is there scleral icterus)?	[]	[]
4.	Signs of dehydration: (sunken eyes or decreased skin turgor)	[]	[]

Blantyre Coma Scale

The Blantyre coma scale⁴ is derived from questions which assess responsiveness in children, It is used to judge the presentation of malarial coma in children, an indicator of cerebral malaria. A total score of 5 is considered a normal and healthy result. Any score under 5 is considered abnormal with increasing severity towards a score of 0.

5	Assessment		Score
5a.	Eye movements	1 = Directed (e.g., follows mother's face) 0 = Not directed	
5b.	Verbal response	2 = Appropriate cry 1 = Moan or inappropriate cry 0 = None	
5c.	Best motor response	2 = Localized painful stimulus 1 = Withdraws limb from pain 0 = None specific or absent response	

Following is a theoretical set of 6 returned clinical case report forms (CRFs). The values have been recorded to a database verbatim. Assume that no custom data validation has been applied to the data entry process.

Which participants will be returned by the queries that follow?*

Participant Id:

A

1	Is the child prostrate?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
2	Is the child fitting now?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
3	Is the child jaundiced?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
4	Signs of dehydration?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
5	5a	Eye movements	1 = Directed 0 = Not directed
	5b	Verbal response	2 = Appropriate cry 1 = Moan or inappropriate cry 0 = None
	5c	Best motor response	2 = Localized painful stimulus 1 = Withdraws limb from pain 0 = None specific / absent response

Participant Id:

B

1	Is the child prostrate?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
2	Is the child fitting now?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
3	Is the child jaundiced?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
4	Signs of dehydration?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
5	5a	Eye movements	1 = Directed 0 = Not directed
	5b	Verbal response	2 = Appropriate cry 1 = Moan or inappropriate cry 0 = None
	5c	Best motor response	2 = Localized painful stimulus 1 = Withdraws limb from pain 0 = None specific / absent response

Participant Id:

C

1	Is the child prostrate?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
2	Is the child fitting now?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
3	Is the child jaundiced?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
4	Signs of dehydration?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
5	5a	Eye movements	1 = Directed 0 = Not directed
	5b	Verbal response	2 = Appropriate cry 1 = Moan or inappropriate cry 0 = None
	5c	Best motor response	2 = Localized painful stimulus 1 = Withdraws limb from pain 0 = None specific / absent response

Participant Id:

D

1	Is the child prostrate?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
2	Is the child fitting now?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
3	Is the child jaundiced?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
4	Signs of dehydration?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
5	5a	Eye movements	1 = Directed 0 = Not directed
	5b	Verbal response	2 = Appropriate cry 1 = Moan or inappropriate cry 0 = None
	5c	Best motor response	2 = Localized painful stimulus 1 = Withdraws limb from pain 0 = None specific / absent response

Participant Id:

E

1	Is the child prostrate?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
2	Is the child fitting now?	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
3	Is the child jaundiced?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
4	Signs of dehydration?	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
5	5a	Eye movements	1 = Directed 0 = Not directed
	5b	Verbal response	2 = Appropriate cry 1 = Moan or inappropriate cry 0 = None
	5c	Best motor response	2 = Localized painful stimulus 1 = Withdraws limb from pain 0 = None specific / absent response

Participant Id:

F

1	Is the child prostrate?	<input type="checkbox"/> Yes	<input type="checkbox"/> No
2	Is the child fitting now?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
3	Is the child jaundiced?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
4	Signs of dehydration?	<input type="checkbox"/> Yes	<input checked="" type="checkbox"/> No
5	5a	Eye movements	1 = Directed 0 = Not directed
	5b	Verbal response	2 = Appropriate cry 1 = Moan or inappropriate cry 0 = None
	5c	Best motor response	2 = Localized painful stimulus 1 = Withdraws limb from pain 0 = None specific / absent response

Query 1:

```
1 | SELECT participant_id
2 | FROM severe_malaria_crfs
3 | WHERE
4 |     (prostrate OR fitting OR jaundiced OR dehydration)
5 |     AND
6 |     (eye_movement + verbal_response + motor_response) <= 3
```

SQL

Query 2:

```
1 | SELECT participant_id
2 | FROM severe_malaria_crfs
3 | WHERE
4 |     (prostrate AND fitting AND jaundiced AND dehydration)
5 |     OR
6 |     (eye_movement + verbal_response + motor_response) <= 3
```

SQL

Query 3:

```
1 | SELECT participant_id
2 | FROM severe_malaria_crfs
3 | WHERE
4 |     prostrate
5 |     AND
6 |     (fitting OR jaundiced OR dehydration OR (eye_movement + verbal_response + motor_respon
```

SQL

**These queries are formatted as SQL, this is covered with more depth in the lab portion. For the purposes of this exercise, evaluate the conditional part of the statement and decide which participant ids will be returned.*

Discussion

Are the following two questions equivalent?

1. Is the child prostrate? ☐ Yes ☐ No

2. Is the child prostrate? (*Check for presentation, otherwise leave blank*): ☐ Yes

Conclusion

GIGO is an old acronym in the computer industry standing for "garbage in, garbage out." It is a venerable maxim used by technology experts (sometimes as an excuse) coupling the usefulness of their systems to the quality of the data fed into the system. Frighteningly, its first widespread use may have been the result of an AP reporter's observations while visiting the IRS.⁵

Needless to say, database administrators proactively attempt to prevent the potential ambiguities and problems that crop up from these situations. Modern relational database implementations have many mechanisms to provide *data consistency*. This may be a constraint which prevent unknown or null values to be committed in the system. However, implementing safeguards and rules to a data sharing system is a double-edged sword. There is the danger of defining a course of rules which results in a rigid and inflexible container. And there is a danger that enforcing consistency may inherently reduce other desirable characteristics. One instance of this phenomenon is asserted by the CAP Theorem.

CAP Theorem

Introduction

Relational database systems are designed with data consistency as its primary dogma. Database consistency, loosely defined, is a guarantee that every transaction returns a reliable outcome. This means that all transactions are applied only after prior requests have fully completed and other requests wait until the current request is complete. However, guaranteed consistency is a complex problem. For database systems, this is accomplished by locking access when its state is in flux. This makes performance problematic, especially for large systems with high volumes of concurrent requests.

Relational databases are centralized systems; they are designed to run on single servers. This makes scaling available resources (memory, disk space, etc.) complicated and costly. The architecture of these systems prevents a distributed layout offered by the cloud for many other computer systems.⁶

In 2000, a computer scientist named Eric Brewer presented a conjecture stating that a shared data system will inherently have trade-off characteristics.⁷ These characteristics are fundamentally relevant to the operation of databases.

Brewer's Conjecture

The CAP theorem, also known as Brewer's theorem, makes the following assertion. At most only two of the following three characteristics can be true of a shared data system:

1. The state of the data is always consistent
2. A view of the data is always available
3. The system is partitioned or remotely distributed

These components of *consistency* (**C**), *availability* (**A**), and *partitions* (**P**) across shared systems was coined by Brewer as the **CAP Theorem**.

Consistency - All nodes see the same data at the same time.

Simply put, performing a read operation will return the value of the most recent write operation causing all nodes to return the same data. A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state. In this model, a system can (and does) shift into an inconsistent state during a transaction, but the entire transaction gets rolled back if there is an error during any stage in the process.

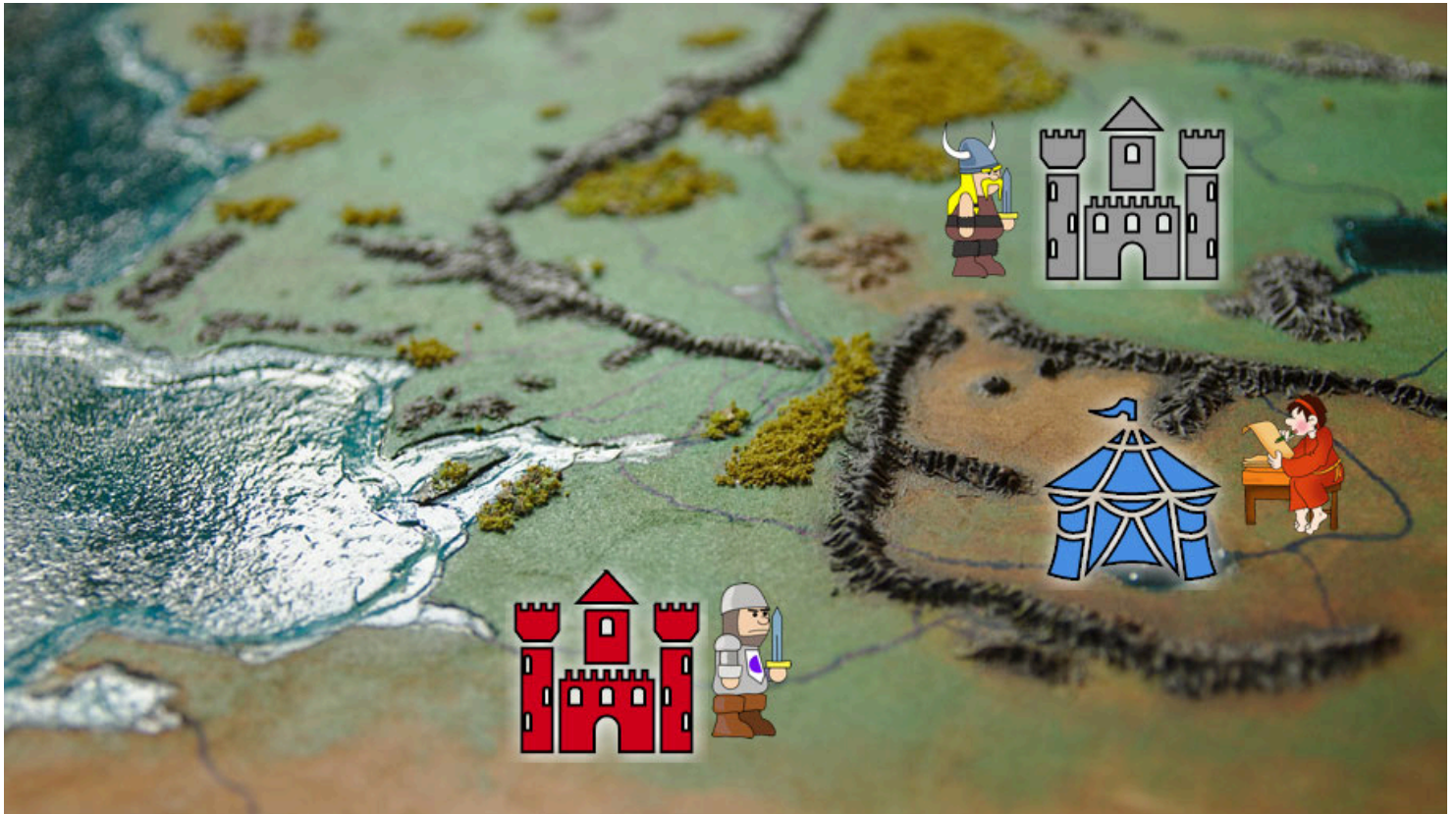
Availability - Every request gets an immediate response.

Achieving availability in a distributed system requires that the system remains operational 100% of the time. Every client gets a timely response to each request, regardless of the state of the system.

Partitions - Requests arrive from remote clients without a direct connection

The internet is inherently partitioned. Messages are sent over a network and therefore are always at risk to be lost. Outages are a common occurrence and occur in separate regions at different times; connections are partitioned.

Illustration - A Tale of Two Kingdoms



ONCE UPON A TIME...

Two warring kingdoms had grown weary of fighting and decided to call for peace. Despite the break from fighting, the two sides still distrusted each other. The king of each side decided it would be best to enlist the help of a neutral third party rather than meet directly. All the details of the pact would be worked out through a trusted mediator.

The mediator penned an initial proposal which included an item of reparation:

"Kingdom A shall give their fall harvest to Kingdom B"

Copies of the document were sent to each party for inspection.

Kingdom B received the plan and wished to change this statement. He immediately sends the mediator a message to get rid of everything after the 5th word, then append "***fairest princess for marriage to the prince of Kingdom B***".

"Kingdom A shall give their ~~fall harvest to Kingdom B~~"

*"Kingdom A shall give their **fairest princess for marriage to the prince of Kingdom B**."*

Meanwhile, Kingdom A still had the original proposal and also wished to revise this item. They decided that their fall harvest was too steep of a cost. They countered with the following:

"Kingdom A shall give ~~their fall harvest~~ to Kingdom B"

*"Kingdom A shall give **25 dairy cows** to Kingdom B."*

They sent a short message to the mediator instructing him to change the **5th**, **6th**, and **7th** word to "**25**", "**dairy**", then "**cows**".

By this time the mediator had incorporated the revisions from Kingdom B. Nevertheless, he followed King A's explicit instructions to replace the 5th, 6th, and 7th words.

"Kingdom A shall give ~~their youngest princess~~ for marriage to the prince of Kingdom B."

*"Kingdom A shall give **25 dairy cows** for marriage to the prince of Kingdom B."*

Needless to say, when King B read this he believed he has been insulted. The war resumed.

THE END

This story is a depiction of a non-consistent system. It illustrates a silly but possible consequence when shared data is edited simultaneously by two parties.

Discussion

- How could the mediator enforce consistency?
- If consistency is observed, how would this change availability in the context of the story?
- How does partitioning apply to this situation?
- How would the story change if they were not partitioned?

Relevance to Modern Database Systems

Nearly all digital transactions today are done over computer networks and the internet. This means that the partitioned characteristic in the CAP theorem is a given. The consequence is that database systems must choose between consistency or availability.

If this was a game of musical chairs, then there is only one chair left and the last two standing contestants are Consistency and Availability.

Networked database systems must design for **consistency** or **availability**. They cannot be engineered for both.

Conclusion

High availability is becoming more important. In many situations, availability is more desirable than consistency. A new generation of databases has been architected to favor availability. They are called NoSQL databases.

NoSQL

Introduction

NoSQL is a term which encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications. Specifically, NoSQL databases focus on the following needs:

- The ability to store massive volumes of new, rapidly changing data types - structured, semi-structured, unstructured and polymorphic data.
- The ability to rapidly pivot and implement system changes depending on the needs of consumers.
- Applications that once served a finite audience are now delivered as services that must be always-on and available
- Organizations are now turning to scale-out architectures using distributed and cloud computing instead of single large monolithic servers

NoSQL Types

- **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.
- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Examples of key-value stores are Riak and Berkeley DB. Some key-value stores, such as Redis, allow each value to have a type, such as 'integer', which adds functionality.
- **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

NoSQL vs. SQL Comparison

	SQL Databases	NOSQL Databases
Types	One type (SQL database) with minor variations	Many different types including key-value stores, document databases, wide-column stores, and graph databases
Development History	Developed in 1970s to deal with first wave of data storage applications	Developed in late 2000s to deal with limitations of SQL databases, especially scalability, multi-structured data, geo-distribution and agile development sprints
Examples	MySQL, Postgres, Microsoft SQL Server, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Data Storage Model	Individual records (e.g., 'employees') are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., 'manager,' 'date hired,' etc.), much like a spreadsheet. Related data is stored in separate tables, and then joined together when more complex queries are executed. For example, 'offices' might be stored in one table, and 'employees' in another. When a user wants to find the work address of an employee, the database engine joins the 'employee' and 'office' tables together to get all the information necessary.	Varies based on database type. For example, key-value stores function similarly to SQL databases, but have only two columns ('key' and 'value'), with more complex information sometimes stored as BLOBs within the 'value' columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single 'document' in JSON, XML, or another format, which can nest values hierarchically.
Schemas	Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.	Typically dynamic, with some enforcing data validation rules. Applications can add new fields on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.
Scaling	Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required, and core relational features such as JOINS, referential integrity and transactions are typically lost.	Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary.
Development Model	Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open-source
Supports Transactions	Yes, updates can be configured to complete entirely or not at all	In certain circumstances and at certain levels (e.g., document level vs. database level)
Data Manipulation	Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE...	Through object-oriented APIs
Consistency	Can be configured for strong consistency	Depends on product. Most offer eventual consistency (e.g., Cassandra).

Conclusion

Relational databases and NoSQL databases should not be viewed as competing technologies. Rather, each fulfills non-overlapping business needs.

-
1. [NIH Research Portfolio Online Reporting Tools ↵](#)
 2. [Wikipedia: Colorless green ideas sleep furiously ↵](#)
 3. [Three-valued logic ↵](#)
 4. [Blantyre coma scale ↵](#)
 5. [Is This the First Time Anyone Printed, 'Garbage In, Garbage Out'? ↵](#)
 6. [Towards Robust Distributed Systems ↵](#)
 7. [Relational Databases Are Not Designed For Scale ↵](#)