## Welcome!

## Data Governance

- Data Governance is vital for maintaining data quality, complying with regulations, managing risks, securing data, fostering transparency, and ultimately, driving efficiency and trust in an organization's data-related activities

## Data Governance

Definition:

- Set of processes, policies, standards, and responsibilities that organizations put in place to ensure high data quality, data management, and data protection
- Encompasses the planning, monitoring, and enforcement of policies, procedures, and controls to manage data as a valuable and strategic asset, facilitating effective decision-making, regulatory compliance, and risk management
- Involves defining roles and responsibilities for data management, establishing data-related policies, and maintaining data integrity, security, and privacy throughout an organization.

## Who Cares?

- Data Quality and Accuracy: Data Governance ensures data is accurate and reliable, enhancing decision-making and operational efficiency.
- Regulatory Compliance: It helps organizations comply with data protection laws, avoiding legal and financial consequences.
- Risk Management: Data Governance identifies and mitigates data-related risks, including breaches and security threats.
- Data Security and Transparency: It protects data from unauthorized access and fosters transparency in data usage, building trust with stakeholders.
- Efficiency and Alignment: Well-governed data improves data management efficiency and aligns data practices with organizational goals, driving performance and value.

## Key Concepts

- Data Ownership

## Data Ownership

- Who within the organization has ultimate accountability and responsibility for specific datasets. Data owners are responsible for ensuring that data is used and managed in line with established policies and standards. They make decisions regarding data access, usage, and protection

## Data Ownership Example

- Customer Data in a Retail Company
- A large retail company collects extensive customer data, including personal information, purchasing history, and preferences.

## Data Ownership Example

Customer Information Manager:

- Role: Owns customer personal information (name, address, contact details).
- Responsibilities: Ensuring data accuracy, compliance with privacy laws, granting and monitoring access rights.

## Data Ownership Example

Sales Department:

- Role: Owns sales-related data (purchase history, product preferences).
- Responsibilities: Analyzing purchasing trends, managing loyalty programs, securing sensitive purchase information.

## Data Ownership Example

Marketing Team:

- Role: Owns marketing data (customer engagement statistics, campaign results).
- Responsibilities: Tailoring marketing strategies based on customer behavior, ensuring data is used ethically for targeting.

## Data Stewardship

- Teams responsible for the day-to-day management and care of data. They ensure data quality, integrity, and security. Data stewards work closely with data owners to implement and enforce data governance policies

## Data Stewardship Example

Scenario: A healthcare organization collects and manages sensitive patient data, including medical records, treatment histories, and personal identifiers.

## Data Stewardship Example

- Data stewards are assigned from different departments like IT, clinical operations, and administration.
- They ensure the accuracy, accessibility, and protection of patient data.

## Data Stewardship Example

- Data Quality Management: Ensuring the accuracy and consistency of patient data across various systems.
- Compliance: Adhering to healthcare regulations like HIPAA, ensuring patient data privacy and security.
- Data Access and Sharing: Managing who has access to what data, facilitating data sharing in compliance with laws and policies.
- Data Usage Guidelines: Establishing policies for the ethical and effective use of patient data for treatment and research.

## Data Quality

- Accuracy, consistency, completeness, and reliability of data. Ensuring data quality is a fundamental aspect of data governance. It involves processes, standards, and controls to maintain and improve data accuracy and usefulness, ultimately supporting better decision-making.

## Data Quality Example

- A multinational bank manages a vast amount of customer data, including account details, transaction histories, and personal identification information.
- A dedicated team is responsible for maintaining the quality of customer data across various departments. They collaborate with IT, customer service, and compliance departments.

## Data Quality Example

Key Responsibilities:

- Data Accuracy: Ensuring that customer information is accurate and up-to-date.
- Data Consistency: Making sure data is consistent across different systems and databases.
- Data Completeness: Verifying that all necessary data fields are filled and current.
- Data Timeliness: Ensuring data is updated in a timely manner, reflecting the most current information.

## Zooming in: Ensuring Data Consistency

## Transaction Concept

## Transaction Management

- Data integrity is a major key
- Transaction management of a database system allows conflict-free simultaneous work by multiple users
- Changes are only applied and visible if all integrity constraints are fulfilled
- Transactions describe database operations bounded by rules, which update database states while maintaining consistency

## ACID

- One concept: a transaction has to be atomic, consistent, isolated, and durable
- ACID

## ACID

- Atomicity (A)
- Transactions are applied fully or not at all
- Intermediate states are not visible to other concurrent transactions
- A transaction can be seen as a unit for the resetability of incomplete transactions

## Consistency

- At the end of the transaction, all constraints must be met
- A transaction always moves the database from one consistent state into another

## Isolation

- Parallel transactions generate the same results as transactions in single-user environments
- This makes transactions a unit for serializability

## Durability

- Database states must remain valid and be maintained until they are changed by a transaction
- Durability retains the effects of complete transactions
- Transactions can be considered a unit of recovery

## ACID

- Under ACID, all users can only make changes that lead from one consistent database state to another
- Inconsistent states are invisible and rolled back if they cause errors

## Serializability

- Concurrent access to the same data objects must be serialized in order for database users to be able to work independety of each other
- Concept of serializability: a system of simultaneous transactions is synchronized correctly if there is a serial execution creating the same database state

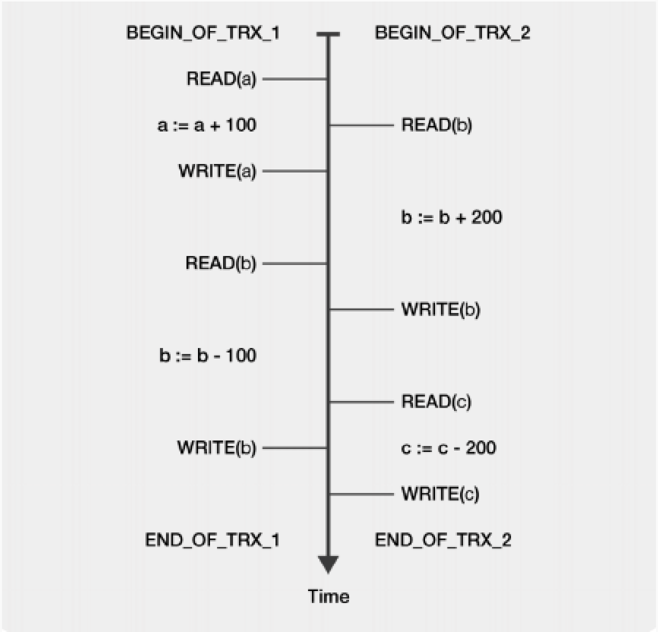
## Example of Serializability: Banking

- Basic integrity constraint is that debit and credit should be balanced

## Imagine the Following Two Transactions

- Three accounts, A, B and C
- Two transactions
    - Transaction 1 (TRX1): a takes 100 units from b
    - Transaction 2 (TRX2): b takes 200 units from c
- Accounts are balanced

## Example

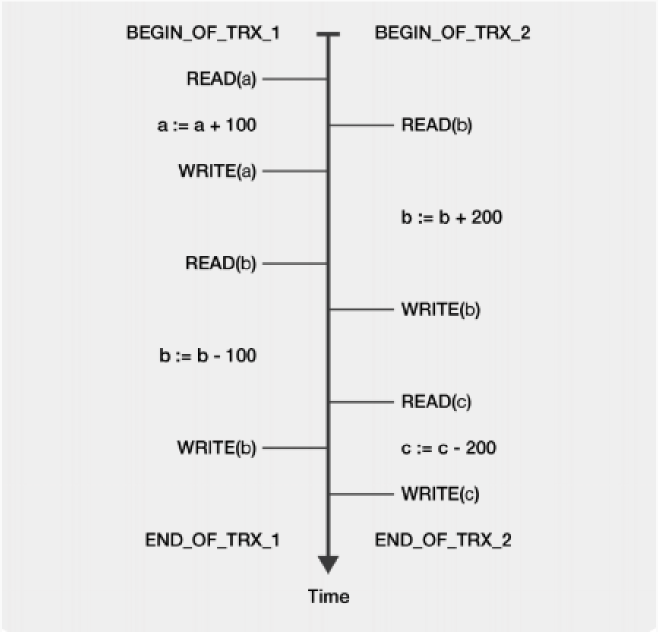


BEGIN_OF_TRX_1
BEGIN_OF_TRX_2
READ(a)
a := a + 100
READ(b)
WRITE(a)
b := b + 200
READ(b)
WRITE(b)
b := b - 100
READ(c)
WRITE(b)
c := c - 200
WRITE(c)
END_OF_TRX_1
END_OF_TRX_2
Time

## Example

- Both transactions occuring at the same time with read and write updates causes errors
- A conflict wwill arise
- TRX1 misses the credit of b+200 done by TRX2, because the change is not immediately written back
- After both are finished, a holds a+100, but b and c are each missing
- Let's look again to see why this is the case

## Example


BEGIN_OF_TRX_1
BEGIN_OF_TRX_2
READ(a)
a := a + 100
READ(b)
WRITE(a)
b := b + 200
READ(b)
WRITE(b)
b := b - 100
READ(c)
WRITE(b)
c := c - 200
WRITE(c)
END_OF_TRX_1
END_OF_TRX_2
Time

## Example

- This kind of stuff is really bad from a consistency perspective
- Think of all the harms we outlined
- What do we do?

## Example

- Logs are helpful here
- LOG(x) contains, in order, all READ and WRITE operations accessing the object x
- Don't confuse with natural log

## Example

Thinking back to set theory, let's think about this elementally. For account b

- b is read by TRX2
- b is read by TRX1
- b is written by TRX2
- b is written by TRX1

## Example

What does this example tell us?

- b was read by TRX2, and then read again by TRX1 before TRX2 wrote a new value.
- So after 2 reads, there were 2 writes, but these needed to have occured in order

## Example

It's hard to think about this analytically

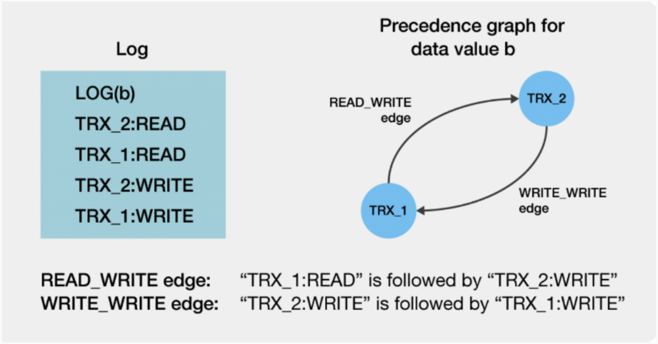
Let's look at a picture

## Example

Precedence graph

- Represents transactions as notes and possible read and write conflicts as arched arrows
- Two reads are fine as long as there are no writes
- So we only draw edges that are read_write or write_write

## Example



READ_WRITE edge: “TRX_1:READ” is followed by “TRX_2:WRITE”
WRITE_WRITE edge: “TRX_2:WRITE” is followed by “TRX_1:WRITE”

## Example

- Starting with TRX1 node, a read on B is followed by a write on it by TRX2
- Then, a write operation by TRX2 follows another write operation, leading to the other edge being drawn
- We only want one arrow!
- We have a cycle, so it isn't serial

## Serializability Condition

- A set of transactions is serializable if the correspondent precedence graph contains no cycles

## How Do We Ensure Serializability?

- Pessimistic methods prevent any concurrent transactions runs that would lead to conflicts
- Optimistic methods accept conflicts and fix them after the fact

## Pessimistic Methods

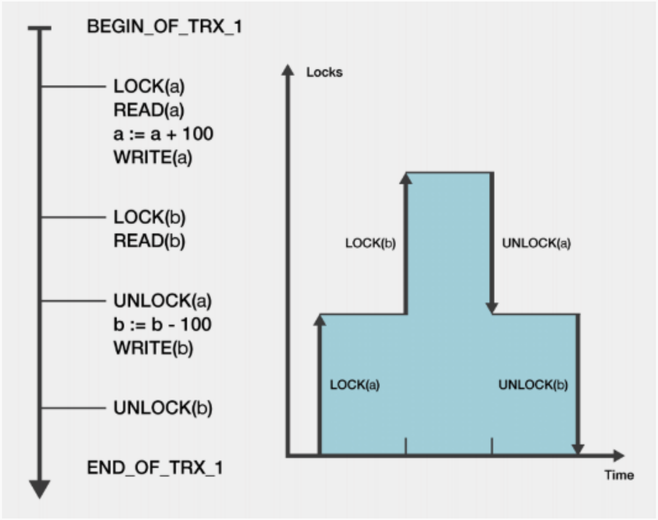
- Exclusive locks: let only one transaction impact an object
- Locking protocol: defines how locks are set and released
- Every object needs to be locked before the transaction starts
- When locked, nothing else can happen to x
- Only after we declare unlock(x) can anything happen

## Pessimistic Methods

Locking protocol: two-phase locking

- 2PL
- Expanding phase: all locks are requested and placed
- Shrinking phase: locks are released one by one
- Unlocking can only happen during shrinking
- In effect, this will prohibit any intermix of creating and releasing locks

## Pessimistic Methods



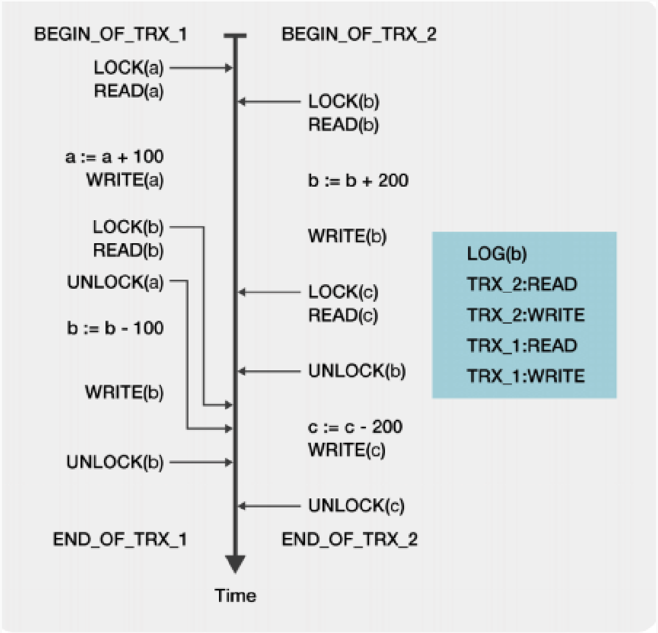
## Pessimistic Methods

- What did we do?
- Locked a, read it, and did the update
- Locked b, read it
- Unlocked a, made our changes to b
- Unlocked b
- Why not lock both at the same time?

## Pessimistic Methods

- Why does this lead to no conflicts?
- Pessimistic concurrency control: any set of concurring transactions is serializable
- Strict seperation of expanding and shrinking prevents any cyclical dependencies

## Pessimistic Methods



## Optimistic Methods

- Assumption: conflicts are rare
- No initial locks to increase sychronization and reduce wait times
- Before a transaction ends, validate retroactively

## Optimistic Methods

- Read phase
- Validate phase
- Write phase

## Optimistic Methods

- Let $TRX_t$ be a transaction we want to validate
- Let $TRX_1$,...,$TRX_k$ be all concurrent transactions that have already been validated during the read phase.
- All objects read by $TRX_t$ must be validated since they could've been modified in $1, ...k$
- The set of objects read by $TRX_t$ is labeled $READ\_SET(TRX_t)$
- The set written by the critical transactions is $WRITE\_SET(TRX_1, ...TRX_k)$

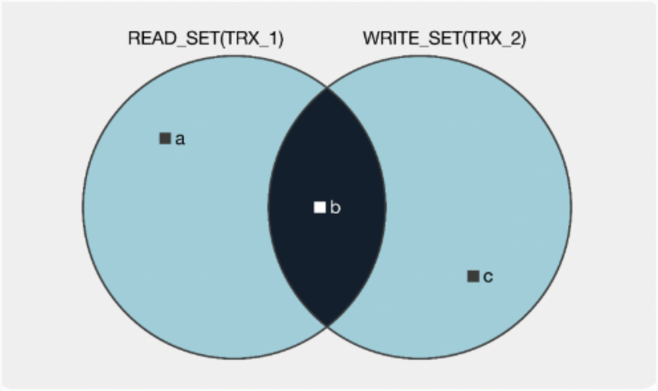
## Optimistic Methods

Optimistic Concurrency Control: The transaction $TRX_t$ is serializable if the sets $READ\_SET(TRX_t)$ and $WRITE\_SET(TRX_1, ...TRX_k)$ are disjoint, such that
$WRITE\_SET(TRX_1, ...TRX_k) \cap READ\_SET(TRX_t) = \emptyset$

## Example

- Let's look back at our example to make this make sense
- We would look at objects read by TRX1 and written by TRX2
- By our definitions, b is an element in both the read set of TRX1 and write set of TRX 2
- We would rollback TRX1 and restart

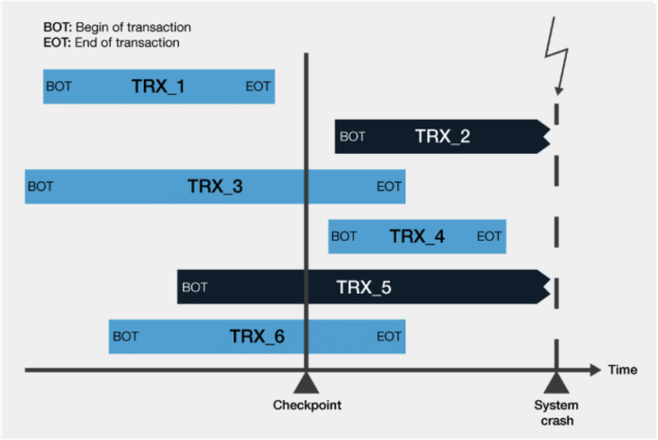
## Optimistic Methods



## What Happens With Errors

- After system failure, the log file is read backwards until the last checkpoint
- Look at transactions that had not concluded with an EOT (end of transaction) marker
- Restore the previous database state for those

## What Happens With Errors



## CAP Theorem

- Consistency seems great! We must always prioritize that
- Well about that...

## CAP Theorem

- Web-based applications call for high availability and the ability to continue working if a computer node or a network fails
- Partition tolerant systems use replicated computer nodes and softer consistency requirements called BASE
- Basically available, soft state, eventually consistent

## CAP Theorem

- Well, sure, its important to have available partition tolerant systems, why do we throw out consistency?
- CAP Theorem: In any massive distributed data management system, only two of three properties (consistency, availability and partition tolerance) can be ensured

## CAP Theorem

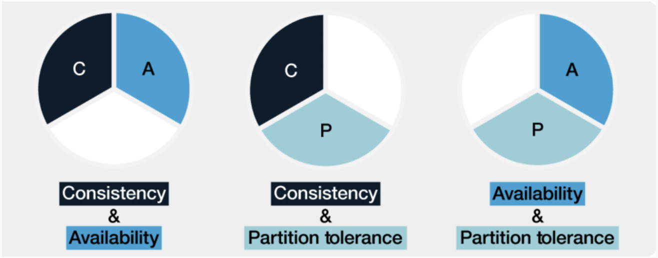
- Consistency: When a transaction changes data in a distributed database with replicated nodes, all reading transactions receive the current state, no matter from which node they access the database
- Availability Running applications operate continuously and have acceptable response times
- Partition tolerance: Nodes can be added or removed at any time without stopping operations, failure doesn't impact the system as a whole

## CAP Theorem

In real-world systems, network failures are inevitable. Therefore, handling partitions (P) is not optional for a robust distributed system.

- If a system prioritizes consistency (C), it must ensure that all nodes have the most recent data. In the event of a partition, some nodes might not be reachable. The system must then choose to either provide outdated information (violating consistency) or provide no information at all (ensuring consistency but sacrificing availability).
- If a system prioritizes availability (A), it must respond to all requests even if it can't guarantee that it's providing the most recent data. This means in the event of a partition, the system will continue to serve requests but might serve outdated data, thus sacrificing consistency.

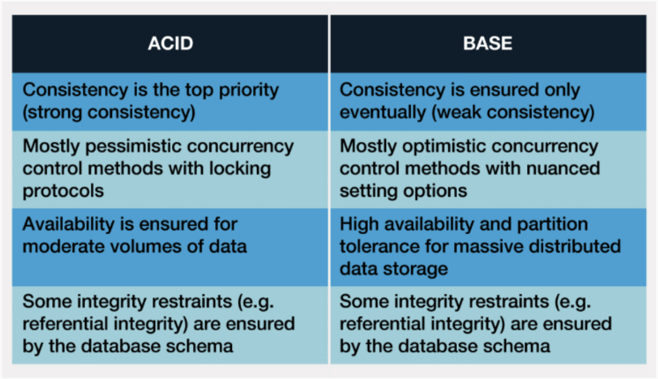
## CAP Theorem


C
A
Consistency
&
Availability
C
P
Consistency
&
Partition tolerance
A
P
Availability
&
Partition tolerance

## CAP Theorem Applications

- Stock exchange needs consistency and availability
- ATMs need to be consistent but also partition tolerant, wait times can be okay
- Domain Name System (DNS) resolves URLs into IP addresses, we need partition tolerance and availability

## ACID versus BASE



| ACID | BASE |
|---|---|
| Consistency is the top priority (strong consistency) | Consistency is ensured only eventually (weak consistency) |
| Mostly pessimistic concurrency control methods with locking protocols | Mostly optimistic concurrency control methods with nuanced setting options |
| Availability is ensured for moderate volumes of data | High availability and partition tolerance for massive distributed data storage |
| Some integrity restraints (e.g. referential integrity) are ensured by the database schema | Some integrity restraints (e.g. referential integrity) are ensured by the database schema |

## Summary

- Database consistency is important but sometimes we need to sequence its importance a bit later
- As DBA's, ensuring data consistency - or considering how to prioritize is - is a key part you will pay in a broader data governance framework

## Data Governance

- So wait, is governance just about the CAP Theorem?
- No!
- Its not even just about DBAs - impacts every level of the organization

## Data Governance and Privacy

Data governance serves as the foundation for establishing and maintaining compliance with data protection and privacy regulations. It provides the structure, policies, and practices necessary to protect personal data, meet legal obligations, and build trust with individuals.

## Privacy

- Data Protection Laws: These laws, such as the General Data Protection Regulation (GDPR) in Europe, the California Consumer Privacy Act (CCPA), and the Health Insurance Portability and Accountability Act (HIPAA) in the United States, set the legal requirements for how personal data should be handled.
- Norms

## Privacy

Personal Data

- Name and surname
- Address
- Contact (email, phone number)
- Geolocation
- Personal ID

## Privacy

Sensitive data

- Medical history
- Geometric data
- Race
- Political/religious identification
- Criminal history

## Privacy

Principles

- Data minimization
- Purpose limitation
- Consent
- Transparency
- Security

## Privacy

Role of governance

- Create policies and frameworks for these issues
- Communication to ensure trust
- Procedures to respond to compromised data
- Data retention and deletion policies

## Privacy

How can we do it?

- Data Protection Impact Assessments
    - Identify Data processing activities
    - Access necessity and proportionality
    - Identify risks
    - Mitigation measures
    - Document the process and compliance
    - Consultation
    - Review and update

## Privacy

- Privacy by design
    - Data Minimization: Collect only necessary data to reduce privacy risks
    - Consent Management: Obtain and manage user consent effectively
    - Data Transparency: Inform individuals about data usage clearly
    - Default Privacy Settings: Use privacy-friendly defaults, requiring informed choices for data sharing
    - Data Security: Implement strong security measures, including encryption and access controls
    - Accountability: Assign data protection responsibility

## Privacy

- Privacy-Enhancing Technologies (PETs)
    - Federated learning
    - Blockchain for privacy
    - Homomorphic encryption

## Introduction to Federated Learning

- Federated Learning enables machine learning models to be collaboratively trained across multiple devices or servers.
- It ensures that raw data remains on the user's device, enhancing privacy and security

## Designing a Federated Learning System

- Traditional machine learning models require centralizing data, raising privacy concerns
- Federated Learning overcomes this by training models locally on each user's device and only sharing model updates

## Collaborative Model Training

- In Federated Learning, users contribute to a global model by sharing their locally trained models.
- The central server aggregates these models, typically by averaging, to improve the global model

## Secure Aggregation

- Secure aggregation techniques ensure that the server only sees the aggregated model, not individual contributions
- This maintains user privacy and prevents data exposure.

## Training a Federated Model

- Complex models may require multiple rounds of local training and federated averaging
- Challenges include ensuring model accuracy and dealing with data variability across devices

## Blockchain for Privacy

- Decentralized network that maintain a ledger
- Data is stored in blocks and linked together with chains
- Once blocks are added, they can't be changed

## Blockchain for Privacy

- Since its decentralized no central authority controls anything, so there is a moving target
- Encrypted
- Transactions are observable even though data is not, preventing fraud
- Hospital application

## Governance Summary

- Data governance is the overarching framework to ensure data is secure, users informtaion is private, and information is consistent
- Database management is a subset of governance
- The manager has a special role in thinking about consistency of data and addressing privacy and security related challenges
- Let's see that in action