# Viewpoint: Information

CSCE 742 - Lecture 11 - 10/09/2018

# Data Manipulation

- Most software manipulates data.
  - Most organizations possess massive amounts of data on customers and products.
- The architecture requires a summary view of static information structure and dynamic information flow.
  - Answers questions around ownership, latency, relationships, identifiers, and more.
- **The Information View** details how the system will manipulate, manage, and distribute information.

# Information View

- Describes the way the system stores, manipulates, manages, and distributes information.
  - Modeled through static information structure models, information lifecycle models, information ownership models, information quality analysis, metadata models, and volumetric models.
  - Addresses concerns around information structure, ownership, data usage, volatility, storage models, flow, consistency, quality, timeliness, latency, age.

# Information Concerns

# Information Structure and Content

- Find the elements of information structure that have system-wide impact, and leave the rest to data designers.
  - Focus on a small number of items core to primary system responsibilities.
  - Focus on information-rich items that:
    - Are fundamental to stakeholder concerns.
    - Are significant to the users.
    - Have complex internal structure.
    - Can impact quality properties.
    - Are heavily used or are expected to change frequently.

# Information Structure and Content

- Early in the project, focus on abstract - not physical - information.
  - Data implementation will change, important to start controlling guiding ideas.
- Focus on system functionality. Model data so that it supports functionality.
- Later, worry about physical considerations.
  - Location, ownership.

# Information Purpose and Usage

- The same information can be used to support operational processes, present operational status, analyze historical trends.
  - How it is used is important. Different usage patterns often have different ownership rules and may require architectural variation.
- Many systems have a transaction store.
  - Manages information required to support operations.
  - Highly volatile.
  - System must process a large number of concurrent read/write operations.

# Information Purpose and Usage

- Significant reporting requirements strain the transaction store.
  - Long-running queries disrupt access by users.
    - Increases response time and lowers throughput.
  - Use a separate reporting database to service complex queries, fed from transaction store.
    - Optimize for compex ad hoc queries, not updates.
- Transaction store biased to current activity.
  - Historic information usually managed in a analytical processing (OLAP) database.
  - Specialized data stores may manage information from a specific domain or time period.

# Information Purpose and Usage

- Most systems rely on reference data.
  - Information on people, places, things that classify system transactions.
  - Varies by organization, but changes infrequently and is lower in volume than transactional or operational information.
- Important to be able to split data across multiple databases and data warehouses.
  - Architecture must control for impact of partitioning, speed of data read/write, data duplication.

# Information Ownership

- Information is often distributed across multiple locations.
  - Which copy of a data item is the most up-to-date?
  - How do you keep information synchronized in multiple places?
  - How do you deal with information derived from information managed and owned elsewhere?
  - What validation should be applied to data modification, and what assumptions can be made about data that has been validated elsewhere?
  - If the same data item can be modified in several places, how are conflicts reconciled?

# Information Ownership

- Insurance company sends workers to customer homes.
  - Customer information is updated on a laptop, sent to central database when they return to the office.
  - Customers can also update information and make purchases online.
  - What if an online update is made before the laptop update is made?
  - What if a laptop update fails strict validation on the central database?
- Requires rules on how to deal with update conflicts and failures.

# Information Ownership

- Develop a model of information ownership.
  - An owner of an item is the system or data store that contains the definitive version of that item.
  - Always has the “correct” value for that information.
- Defining data owners helps ensure that consumers have the right data and that producers write to correct location.
  - Also clarifies interfaces - interfaces are required between data owners and consumers.

# Identifiers and Mappings

- All data items need unique identifiers distinguishing them from other items.
  - Customer number, serial number, ISBN.
  - The *primary key*, *object ID*, *identifier*.
- If different systems use different means to identify items, these mechanisms must be reconciled when data exchange occurs.
  - Key assignment can be volatile.
  - Reconciliation must be kept up to date with new information as it arrives.

# Identifiers and Mappings

- Identifiers should be invariant.
  - Not always possible. Mechanisms for creating and changing identifiers must be carefully designed.
    - Ex: Financial derivatives are assigned temporary ID while going through approval. Once approved, they are given a permanent ID. Link must be established between the two.
- Difficult to decide if two items are the same.
  - Ex: New edition of a book. Could only have minor corrections, or could be entirely new.
    - Should it have the same ISBN identifier?
- Should identifiers be user-visible?
  - Ex: Credit card numbers are unique, but may not be shown to protect privacy.

# Information Semantics Volatility

- Information changes frequently.
  - New fields, constraints, relationships, or entities may be added to existing data.
- Small changes can have implications for systems that use changing data.
  - New mandatory field added to a database requires every process that creates or updates data to provide a value for that field.
- Managed through formal process of data model change control.
  - Data change only implemented once all parties have implemented required code changes.

# Information Semantics Volatility

- Decouple information semantics from the physical structure used to store it.
    - Store information structure in a structured text format (JSON, XML, YAML).
    - XML data management standards allow definition of a schema for XML documents.
    - Changes to schema can be implemented quickly.
    - Trade-off: XML-based systems can be less scalable due to XML management overhead and lack of database optimization support.

# Information Flow

- How does information move around the system? How is it accessed and modified by system elements?
  - Where is data created and destroyed?
  - Where is data accessed, modified, and enriched?
  - How do individual data items change as they move around the system?
- Architecture must identify the most important information flows.
  - May be part of functional view.

# Information Consistency

- Information held in different parts of the system must be compatible, congruent, and not in conflict.
- Transactions are updates that occur as an atomic unit (all updates accepted or none).
    - Transaction management ensures right outcome by committing updates only if all updates can be applied.
    - Ex: Customer transfers $500 from CHECKING to SAVINGS. Implemented as two updates. Important that either both updates work or neither do.

# Information Consistency

- Transaction management across systems or processes is complicated to build or operate.
- Compensating Transactions:
  - Each data update is committed individually.
  - If a later update fails, each committed update is reversed with a new transaction with an equal and opposite effect.
    - If withdrawal succeeds and deposit fails, a compensating deposit of $500 to the checking account will restore the original state.
  - Do not require locks over separate data stores at the same time.
  - Problem: what if the compensating transaction fails?

# Information Consistency

- Eventual Consistency:
  - Favor high availability over consistency.
  - Guarantees that all instances of the same data will eventually be updated to a value, without guarantee of *when* this will occur.
    - Used by DNS system, NoSQL databases
  - BASE principles:
    - Basic Availability: Data should be available in the presence of multiple failures. Instead of a central data store, spread data across many systems with high replication.
    - Soft State: Data consistency is left to developer, not the database.
    - Eventual Consistency: Data will converge to consistency.

# Information Quality

- Are data values in your system accurate?
- Affects the architecture of systems that use information from a variety of sources.
  - How will data quality be assessed and monitored?
  - What minimum quality criteria must be met?
  - How will these criteria be enforced?
  - How will poor-quality information be improved?
  - Can good-quality information be corrupted by information of lesser quality?
  - If so, should this be prevented or checked?
  - Is it possible for information quality to degrade as it flows around the system?

# Information Quality

- May be necessary to develop tools for monitoring or assessing information quality.
- Data may need to be held in a “holding” state for human repair.
  - Often managed through workflow.
  - List of tasks (i.e., correct customer names) is managed in a central database.
  - Tasks are assigned to workers and the system tracks status.
  - Tasks can be standard or ad hoc.
  - Company sets target service level.

# Information Timeliness

- In distributed environments, information can be out of date.
  - Commodity brokerage system accepts information feeds and filters through a central gateway.
  - After returning from downtime, the gateway floods subscribers with messages that contain old price information.
  - Gateway should be modified so that, after a failure, it discards cached messages older than a threshold. Will enable faster recovery.

# Information Timeliness

- Information transfer from producers to consumers takes time.
    - If lag cannot be reduced to near-zero, architecture must deal with impact of inconsistent information.
    - Time lag measured in terms of length of time between data update at source and the updated value being available throughout system.
- Take into account age of data items (since last update by data source).
    - Discard information that is older than a threshold to prevent misuse.

# Information Timeliness

- Identify when time-based inconsistency can occur and handle them:
    - Tag important data items with a “last updated” date and time.
    - Define “currency windows” for significant data items.
    - Warn users when information may be outdated.
    - Hide or discard information that may be too old.
    - Reduce latency by means of faster interfaces or direct access to data sources.

# Information Archival and Retention

- Information is often retained for legal and historical analysis.
- Eventually, older and less-useful information should be transferred to offline storage.
  - Scope of archived information must be carefully defined.
  - Cannot be information needed to support production activities or used in regular analysis.
  - Selected on basis of age and usefulness.

# Information Archival and Retention

- Archival strategy impacts architecture.
  - Archiving large volumes of information may make some systems fully or partly unavailable for significant periods of time.
  - Physical disk sizing needs to take into account the length of time information will be retained.
  - Need to define the processes that move production information to archive media,
  - Need to take special actions to ensure the integrity and consistency of production and archive storage.
  - There may be an impact on the network infrastructure if archive storage is remote.

# Information Modeling

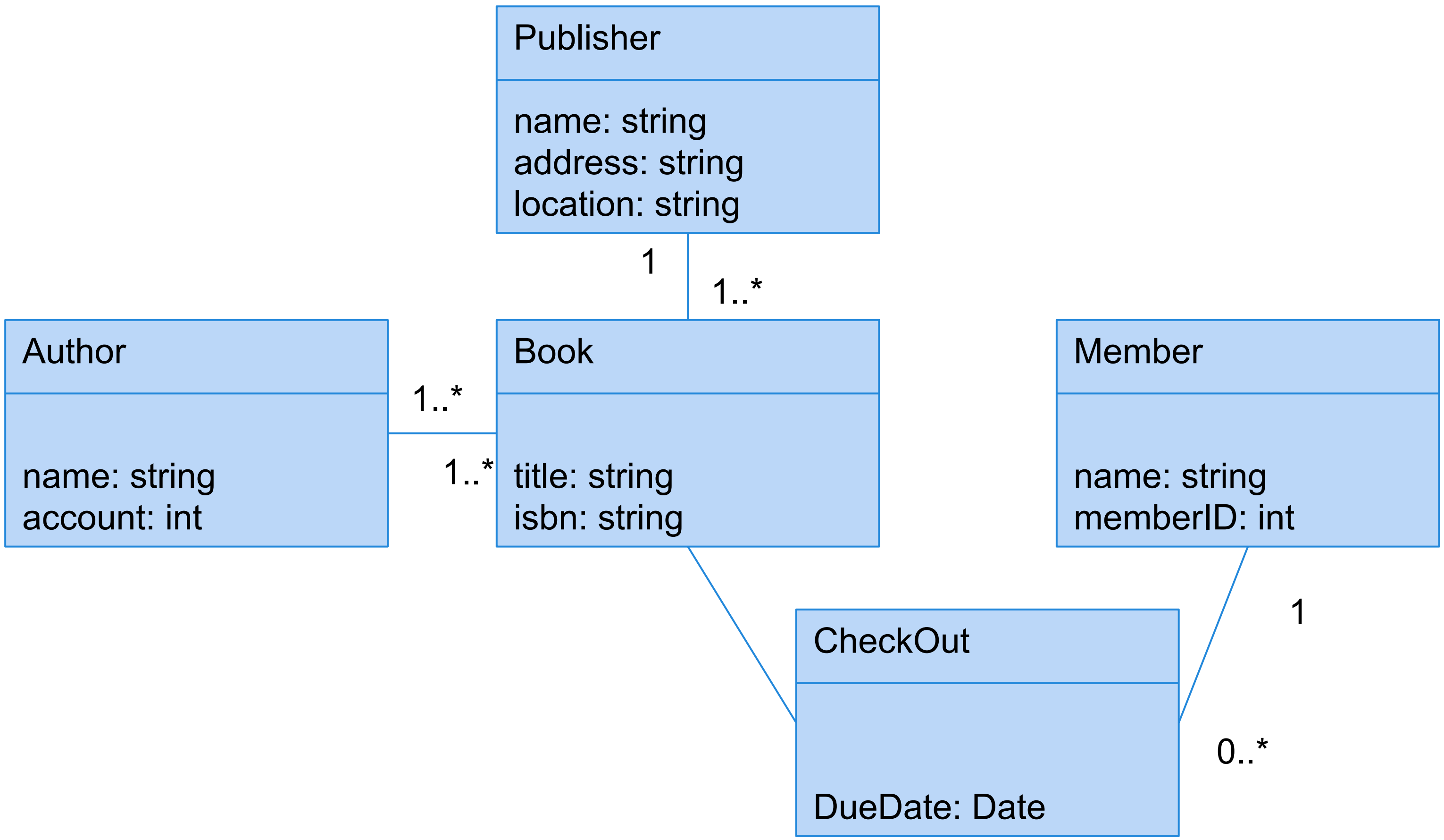
# Information Models

- Static Information Models
  - Analyze the static structure of information.
- Information Flow Models
  - Analyze the dynamic movement of information between elements of the system and the world.
- Information Lifecycle Models
  - Analyze the way information changes over time.

# Static Information Structure Models

- Analyze the static structure of the data.
  - The important data elements and the relationships among them.
- Entity-relationship modeling
  - Important data items are *entities*.
  - Their parts are called *attributes*.
  - We define *relations* between entities based on information semantics.
  - Relations have *cardinality* based on how many of an entity can be related to an instance of the other.
  - Similar to class diagrams, but omitting methods.

# Entity-Relationship Example



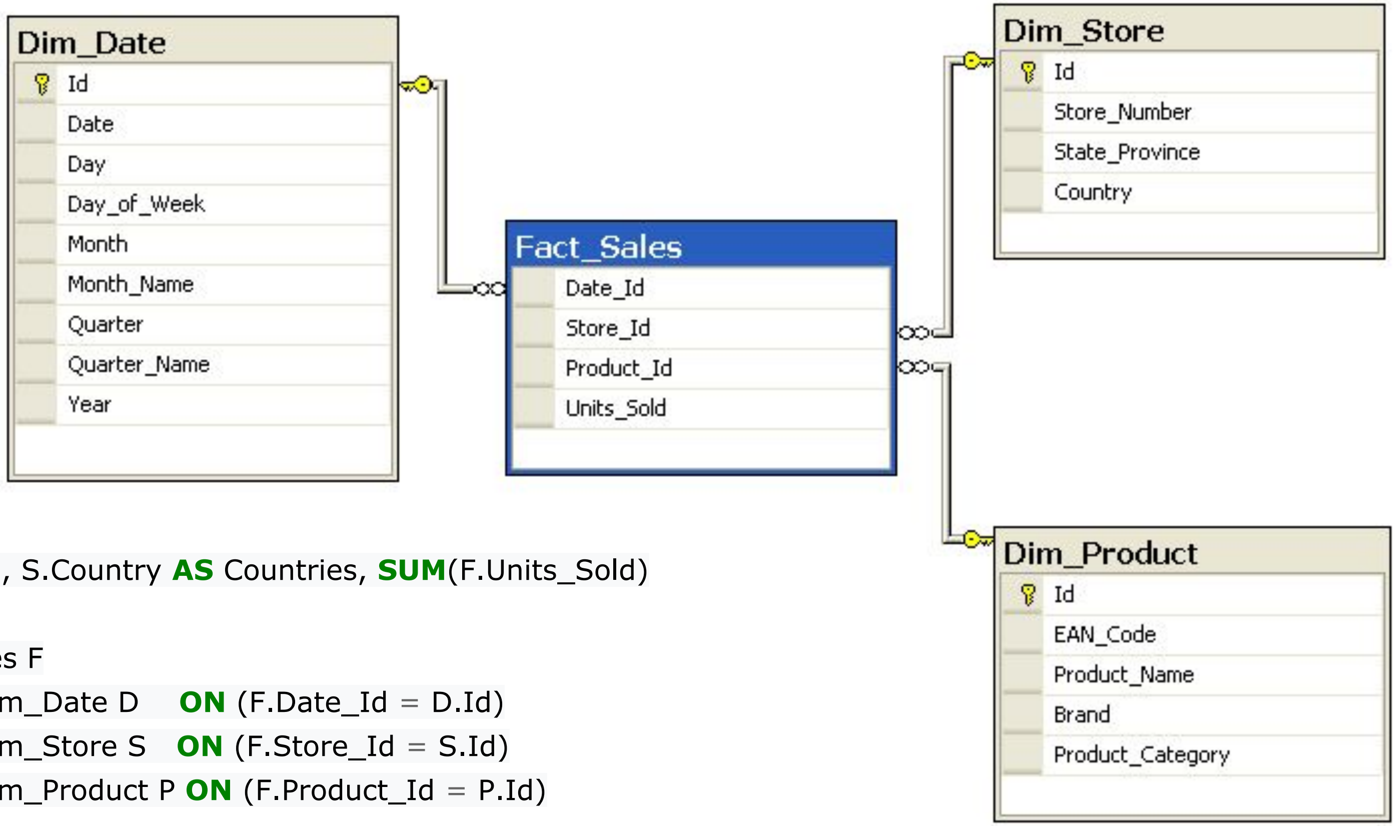
# Star Schema

- Data model used in data warehouses.
- *Facts* hold measurable, quantitative data.
  - Sale price, sale quantity, sale time
- *Dimensions* are descriptive attributes related to fact data.
  - Product models, product colors, product sizes
- Fact tables contain facts at different levels.
  - Generally consist of numeric values and foreign keys to dimensional data where descriptive information is kept.

# Star Schema

- Dimensional tables model different levels at which information can be aggregated.
  - Usually contain fewer records than fact tables, but each record may have a large number of attributes.
  - Describe the fact data.
- An aggregated value can be retrieved in a single database read, rather than querying and summarizing all underlying transactions.
  - Simplified queries - join logic is simpler than in highly normalized schema.
  - Improved query performance

# Star Schema Example




```sql
SELECT P.Brand, S.Country AS Countries, SUM(F.Units_Sold)

FROM Fact_Sales F
INNER JOIN Dim_Date D    ON (F.Date_Id = D.Id)
INNER JOIN Dim_Store S   ON (F.Store_Id = S.Id)
INNER JOIN Dim_Product P ON (F.Product_Id = P.Id)

WHERE D.Year = 1997 AND  P.Product_Category = 'tv'

GROUP BY P.Brand, S.Country
```

From https://en.wikipedia.org/wiki/Star_schema

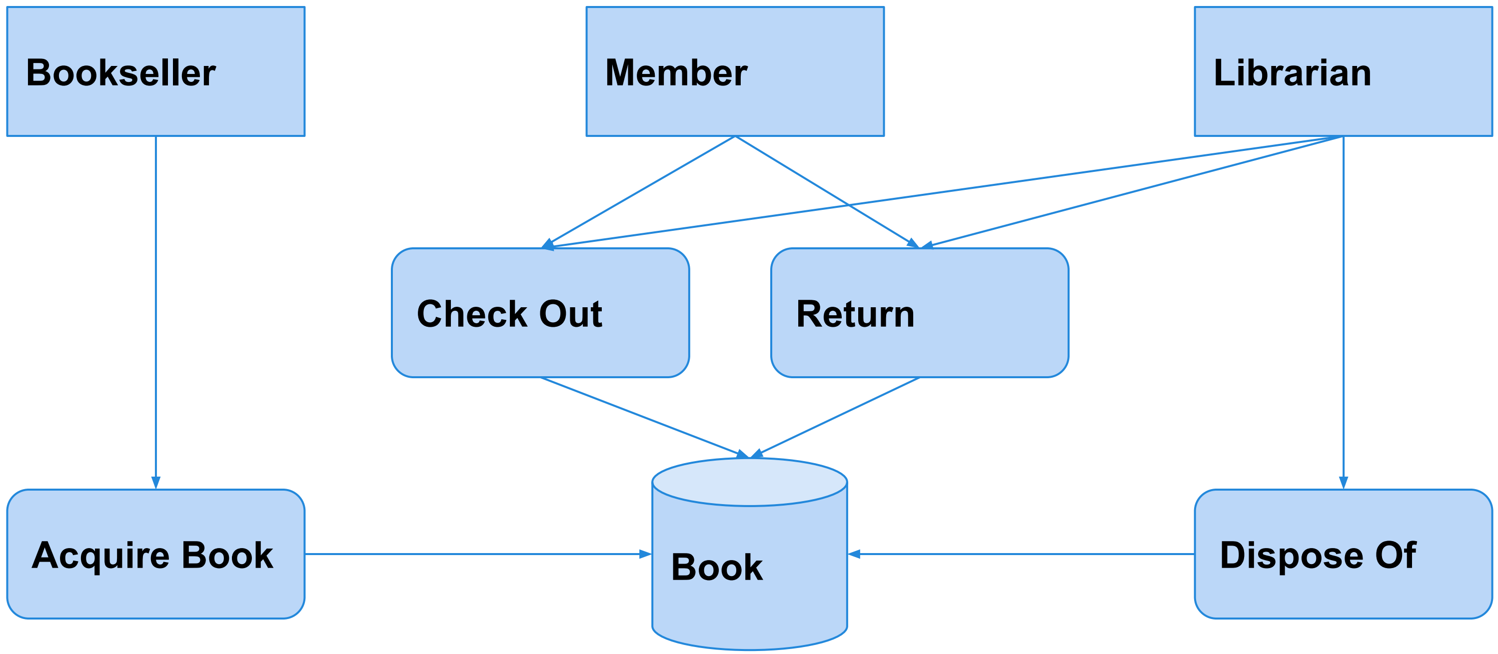
# Structure Modeling Activities

- Identify the important data entities.
  - Inspect the use cases and scenarios for nouns.
- Normalization reduces model to purest form.
  - No repeated, redundant, duplicated information.
  - Sometimes helpful to leave data unnormalized in architecture so that you don't miss anything.
- Perform domain analysis to define legal value ranges for data attributes.
  - Not performed rigorously at architectural level, but a basic pass can inform your model.

# Information Flow Models

- Analyze dynamic movement of information between elements of the system and the outside world.
- *Flows* represent information transferred from one component to another.
  - Associated with a direction, scope of information transferred, volumetric information.
  - In a physical model, also includes how the information is transferred (flat files, JSON, XML messages).

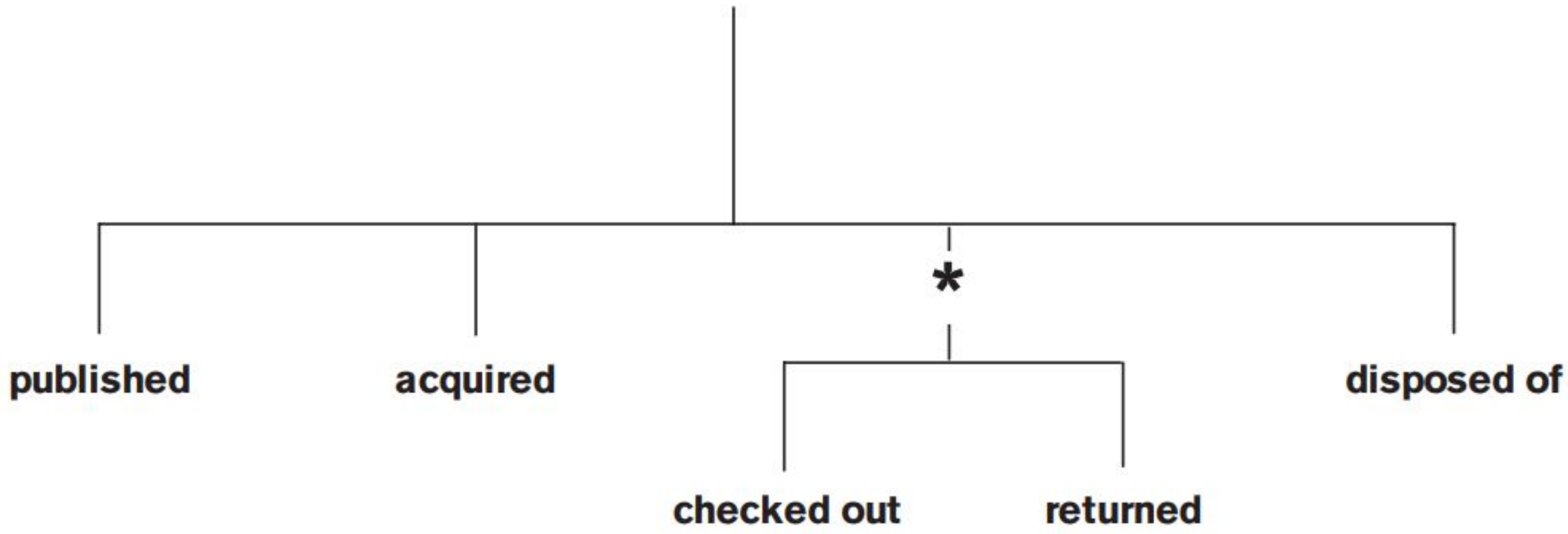
# Information Flow Models



# Information Lifecycle Models

- Lifecycle models analyze the way information values change over time.
- Entity Life Models
  - Model transitions data items undergo in response to external events, from creation to final deletion.
  - Can be a useful cross-check to ensure there is processing to deal with all of the life events for that entity.
  - Helps ensure entities are created in a controlled manner, and that all entities can be deleted.

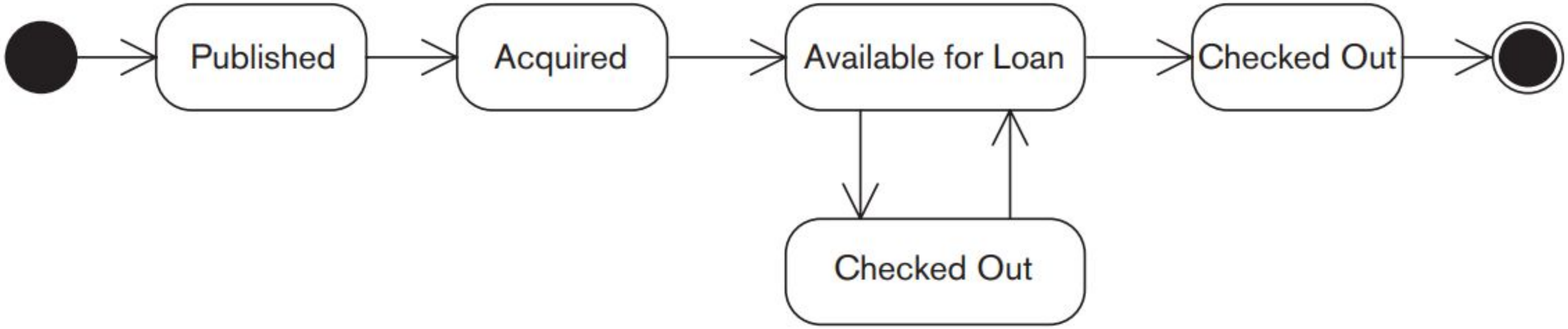
# Entity Life Model




- A book is initially published
- It is then acquired by the library
- Once on the shelves it alternates between available and checked out…
- Until it is disposed of.

# UML State Diagram

- Model overall changes in an element's state in response to stimuli.
- Often used to model systems. Can also model state of a data entity.



# Information Ownership Model

- Define the system that is the "owner" of each data item in the architecture.
  - A data item being a table or field.
- Classes of ownership:
  - *Owners* hold definitive value for an item.
  - *Creators* create new instances of the item.
  - *Updaters* modify existing instances of the item.
  - *Deleters* delete instances of the item.
  - *Readers* can read, but not change, the item.
  - *Copiers* hold a read-only copy of the item.
  - *Validaters* perform validation of the item.
  - Combination of the above.

# Information Ownership Model

- Modeled using a grid, with systems and data stores on one axis and data items on other.
- Shows conflicts in data ownership.

| System | Customer | Product | Order | Fulfillment |
|---|---|---|---|---|
| Catalog | None | Owner | None | None |
| Purchasing | Reader | Updater | Owner | Creator |
| Delivery | Copy | Reader | Reader | Updater |
| Customer | Owner | Reader | Reader | Reader |

# Problems and Pitfalls

# Representation Incompatibility

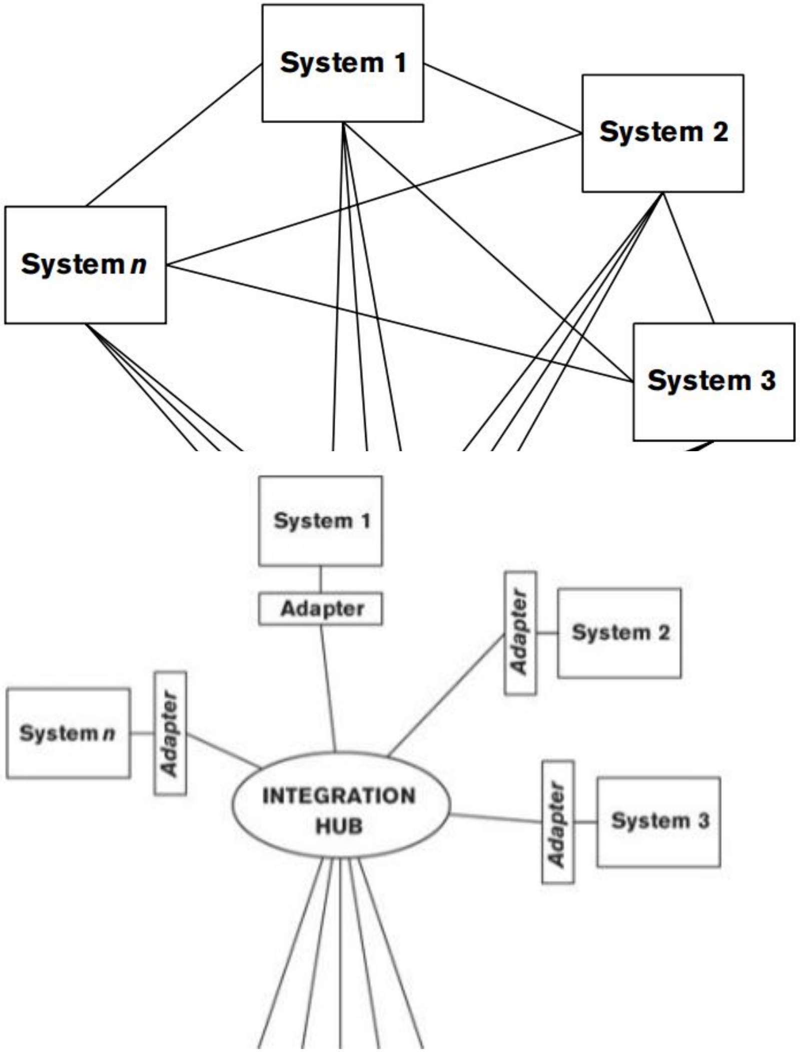
- Simple: Different systems represent field-level information in different ways.
  - Different models (polar vs cartesian coordinates)
  - Different encoding (metric vs imperial)
- Difficult: Different business models
  - Ex: Architecture integrates phone and sales systems
  - Phone customers may have multiple or shared numbers, each number has a “phone account”.
  - Sales system requires “customer accounts”.
  - These two are not compatible.

# Representation Incompatibility

- Reconciling business models requires complex processing.
  - Service that links customers and accounts.
  - Manages and stores links itself.
  - Would sit at center of architecture, require high performance, scalability, availability.
- Risk reduction:
  - Develop a common high-level data structure model.
  - Include external entities in modeling efforts.
  - Develop a data abstraction layer to hide incompatibilities from other parts of architecture.

# Interface Complexity

- Integrating n data-sharing systems requires n (n - 1) / 2 interfaces.
  - Change to a system require changes to all interfaces accessing that system.
- Can be fixed by applying an integration hub.
  - Adapter performs system-specific translation.
  - Hub handles message routing, resilience.
  - If a system changes, only the adapter needs to change.



# Overloaded Central Database

- Central databases are simpler, cleaner. Do not need update reconciliation or complex interfaces.
  - Single point of failure, performance bottleneck, imposes geographical constraints.
  - Can cause data model to be overloaded.
  - Can impose design and runtime constraints.
- Risk Reduction:
  - Consider likely growth rate.
  - Develop strategies for later data partitioning.
  - Plan for scalability.

# Inconsistent Distributed Databases

- Many problems are eliminated by replicating data in different locations.
  - Bring data close to where it is needed. Reduces latency and improves availability.
  - Often lead to information inconsistency due to replication delay.
  - Updates are hard to manage.
- Risk Reduction:
  - Have strategies in place for dealing with inconsistency.
  - Ensure there are tools and processes for detecting data inconsistency.

# Excessive Information Latency

- Overly complex architectures or architectures not designed for information volume can lead to excessive latency.
  - Latency issues from external systems are out of your control.
- Identify expected latency early.
  - Can identify problem areas and address.
  - Close distance between providers and consumers.
  - Obtain agreement on latency requirements from stakeholders and validate that you meet them.

# Food for Thought

- Do you have an appropriate level of detail in your data models?
  - (e.g., no more than 20–30 entities)
- Does the data model support processing requirements now and in the future?
- Are keys clearly identified for all entities?
  - When an entity is distributed across multiple systems or locations with different keys, are the mappings between these keys defined?
  - Do you have processes for maintaining these mappings when data items are created?

# Food for Thought

- Have you taken account of data that is derived from data managed elsewhere?
- Have you defined strategies for resolving data ownership conflicts?
- Are latency requirements clearly identified, and are mechanisms in place to ensure that these are achieved?
- Do you have clear strategies for transactional consistency across distributed data stores?

# Food for Thought

- Have you considered which data storage models to use for the data stores?
- Do you have mechanisms in place for validating migrated data?
- Have you defined sufficient storage and processing capacity for archiving?
    - For restoring archived data?

# Key Points

- The Information View describes the way the system stores, manipulates, manages, and distributes information.
  - Modeled through static information structure models, information lifecycle models, information ownership models, information quality analysis, metadata models, and volumetric models.
  - Addresses concerns around information structure, ownership, data usage, volatility, storage models, flow, consistency, quality, timeliness, latency, age.

# Next Time

- Midterm Review
  - Practice midterm on site.
  - We will go over answers next time.

- Homework:
  - Project Part 2 - Due on the 11th
  - Assignment 2 - Due on the 25th