BIG DATA

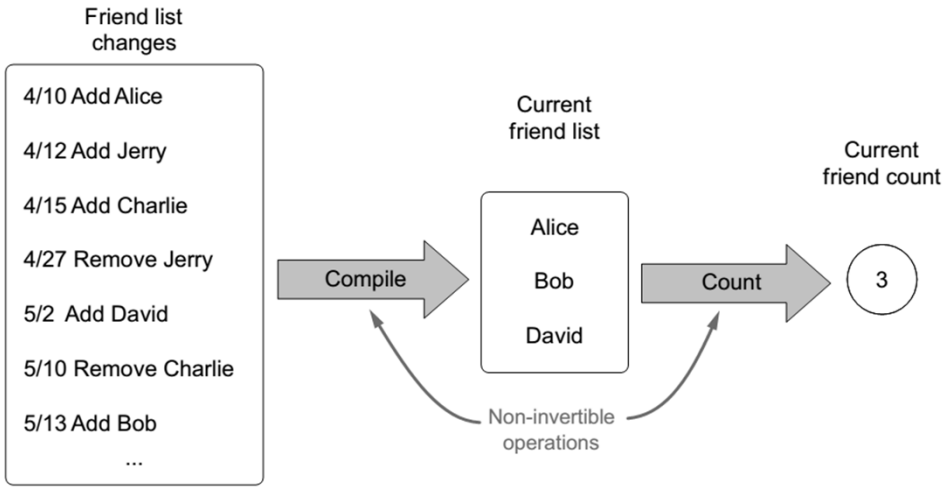
Data model for Big Data

INTRODUCTION

- The master dataset: the only part that must be safeguarded from corruption
  - Must be carefully engineered regarding two components
    - The data model
    - How you physically store the master dataset
  - This chapter is about designing a data model
  - You'll learn about physically storing a master dataset in the next chapter

THE PROPERTIES OF DATA

- Suppose you're designing the next big social network: FaceSpace
  - Information you should store regarding Tom's connections
    - The sequence of Tom's friend and unfriend events
    - Tom's current list of friends
    - Tom's current number of friends
  - Terms we'll use
    - Information
    - Data
    - Queries
    - Views
  - Important note:
    - one person's data can be another's view



THE PROPERTIES OF DATA

- key properties of data
  - Rawness
    - The rawer your data, the more questions you can ask of it
    - Semantic normalization: the process of reshaping freeform information into a structured form
      - Store the result of simple and accurate
      - Store the unstructured form of the data if the algorithm is subject to change
    - More information doesn't necessarily mean rawer data
      - What exactly should you store once Tom provides the URL of his blog?
  - Immutability
  - Eternal trueness

THE PROPERTIES OF DATA

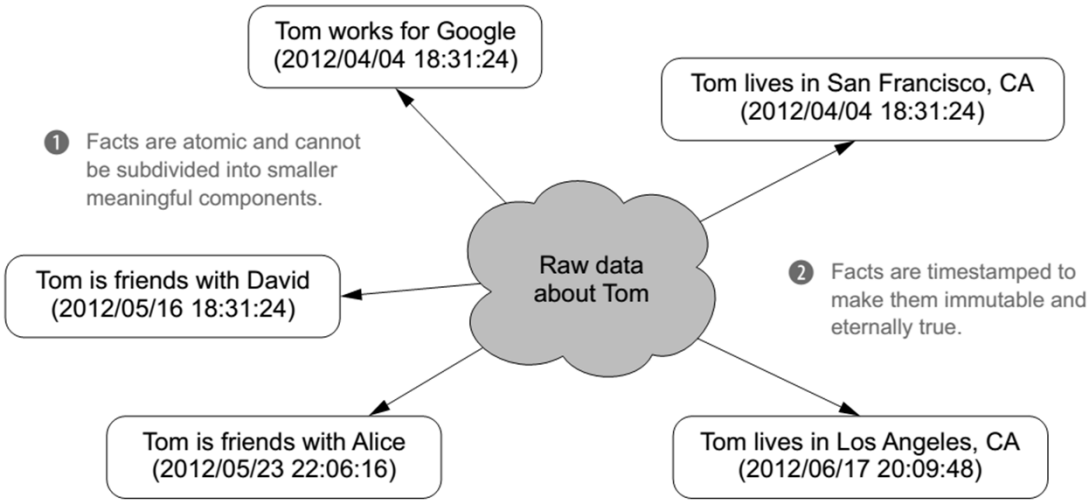
- key properties of data
  - Rawness
  - Immutability
    - you don't update or delete data, you only add more
    - Two vital advantages
      - Human-fault tolerance
      - Simplicity
    - It uses more storage
      - But Big Data isn't called "Big Data" for nothing
  - Eternal trueness

THE PROPERTIES OF DATA

- key properties of data
  - Rawness
  - Immutability
  - Eternal trueness
    - Master dataset grows by adding new immutable and eternally true pieces of data
    - A piece of data, once true, must always be true
    - Special cases in which you do delete data
      - Garbage collection
        - you delete all data units that have low value
      - Regulations
        - Government regulations

THE FACT-BASED MODEL FOR REPRESENTING DATA

- Deconstruct the data into fundamental units called facts
  - Two core properties of facts
    - Atomicity
      - no redundancy
    - Timestamped
      - immutable and eternally true facts



THE FACT-BASED MODEL FOR REPRESENTING DATA

- Deconstruct the data into fundamental units called facts
  - Additional recommended property
    - Identifiability
      - Helps in identifying duplicates
      - Removes the need for transactional appends

```
struct PageView:
  DateTime timestamp
  String url
  String ip_address
```

```
struct PageView:
  Datetime timestamp
  String url
  String ip_address
  Long nonce
```

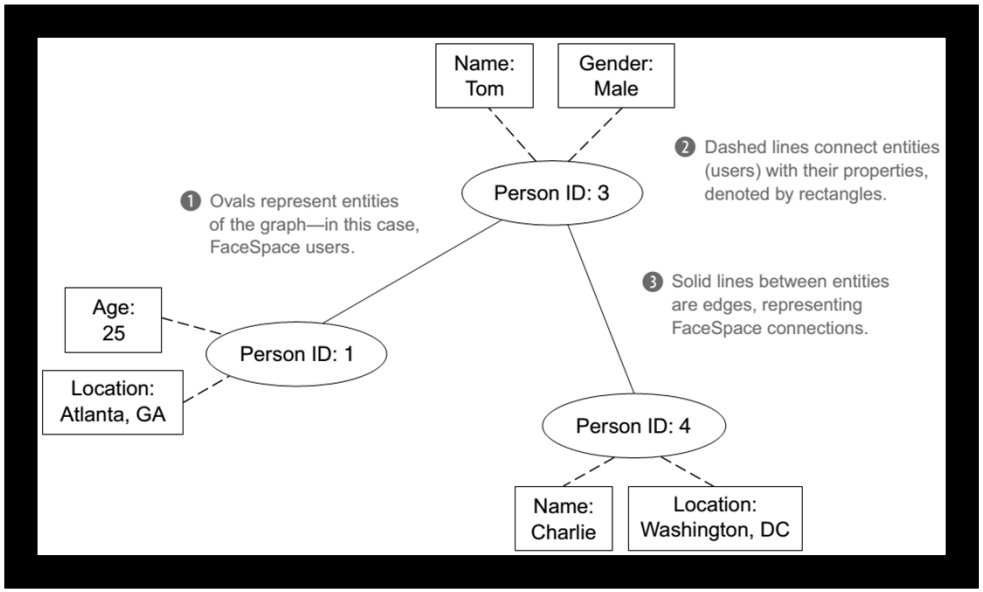
The nonce, combined with the other fields, uniquely identifies a particular pageview.

THE FACT-BASED MODEL FOR REPRESENTING DATA

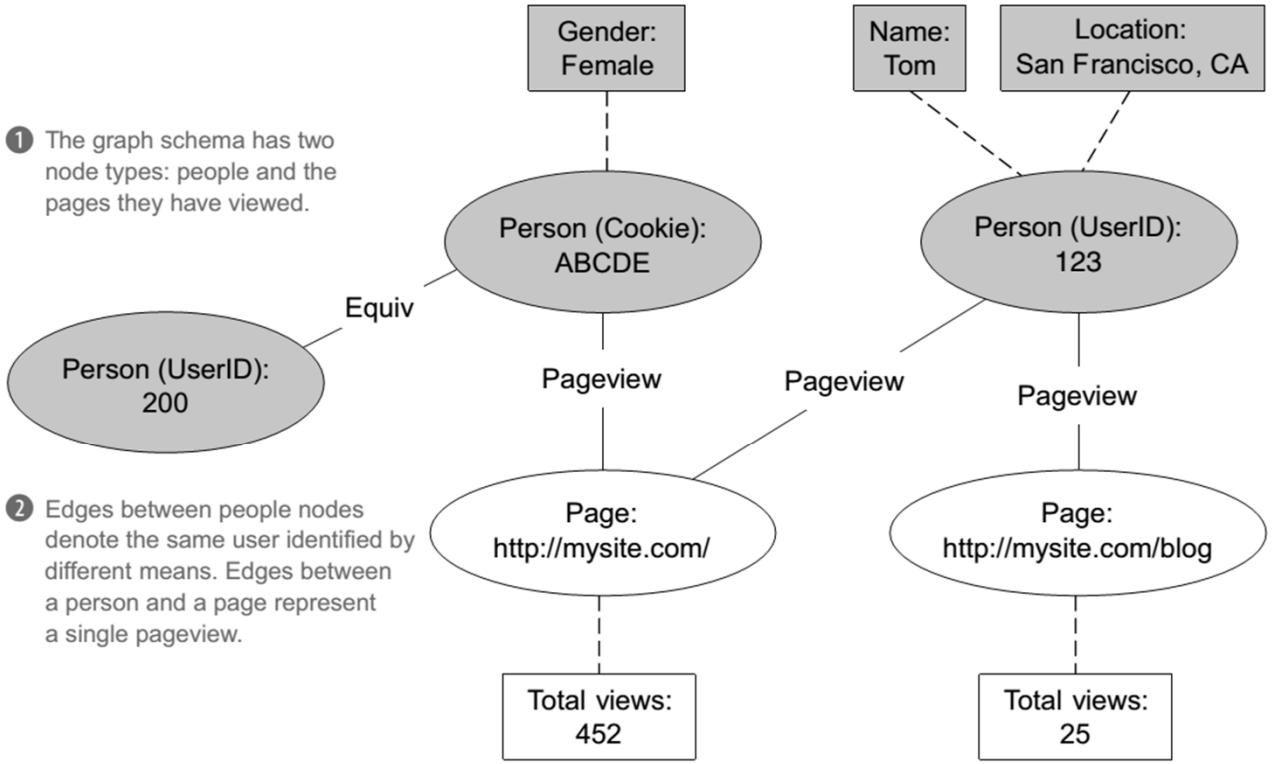
- Benefits of the fact-based model
  - Data is queryable at any time in its history
  - Data tolerates human errors
  - Data handles partial information
    - Dataset only have facts for the known information
    - Additional new information would naturally be introduced via new facts
  - Data has the advantages of both normalized and denormalized forms
    - In relational databases
      - Normalization: data consistency
      - Denormalization: query efficiency
    - In lambda architecture
      - Master dataset is fully normalized
      - Batch views are like denormalized tables

GRAPH SCHEMAS

- Captures the structure of a dataset
- Elements
  - Nodes: the entities in the system
  - Edges: relationships between nodes
  - Properties: information about entities



A COMPLETE DATA MODEL FOR SUPERWEBANALYTICS.COM



ILLUSTRATION

- Apache Thrift
  - A tool that can be used to define statically typed, enforceable schemas
  - Provides an interface definition language to describe the schema
    - Can be used to automatically generate the actual implementation in multiple programming languages
  - Initially developed at Facebook
  - Can be used for many purposes
  - We'll limit our discussion to its usage as a serialization framework

ILLUSTRATION

- Apache Thrift
  - Core components: struct and union type definitions composed of
    - Primitive data types (strings, integers, longs, and doubles)
    - Collections of other types (lists, maps, and sets)
    - Other structs and unions
  - Unions are useful for representing nodes
  - Structs are natural representations of edges
  - Properties use a combination of both

ILLUSTRATION

- Apache Thrift
  - Nodes

```
union PersonID {
  1: string cookie;
  2: i64 user_id;
}

union PageID {
  1: string url;
}
```

  - Edges

```
struct EquivEdge {
  1: required PersonID id1;
  2: required PersonID id2;
}

struct PageViewEdge {
  1: required PersonID person;
  2: required PageID page;
  3: required i64 nonce;
}
```

ILLUSTRATION

- Apache Thrift
  - Properties

```
union PagePropertyValue {
  1: i32 page_views;
}

struct PageProperty {
  1: required PageID id;
  2: required PagePropertyValue property;
}
```

```
union PersonPropertyValue {
  1: string full_name;
  2: GenderType gender;
  3: Location location;
}

struct PersonProperty {
  1: required PersonID id;
  2: required PersonPropertyValue property;
}
```

```
struct Location {
  1: optional string city;
  2: optional string state;
  3: optional string country;
}

enum GenderType {
  MALE = 1,
  FEMALE = 2
}
```

ILLUSTRATION

- Apache Thrift
  - Tying everything together into data objects

```
union DataUnit {
  1: PersonProperty person_property;
  2: PageProperty page_property;
  3: EquivEdge equiv;
  4: PageViewEdge page_view;
}

struct Pedigree {
  1: required i32 true_as_of_secs;
}

struct Data {
  1: required Pedigree pedigree;
  2: required DataUnit dataunit;
}
```

ILLUSTRATION

- Apache Thrift
  - Evolving your schema
    - key to evolving: the numeric identifiers associated with each field
    - Evolving rules for backward compatibility
      - Fields may be renamed
      - A field may be removed, but you must never reuse that field ID
      - Only optional fields can be added to existing structs

ILLUSTRATION

- Apache Thrift
  - Evolving your schema
    - Changing schema to store a person's age and the links between web pages

```
union PersonPropertyValue {
  1: string full_name;
  2: GenderType gender;
  3: Location location;
  4: i16 age;
}
struct LinkedEdge {
  1: required PageID source;
  2: required PageID target;
}

union DataUnit {
  1: PersonProperty person_property;
  2: PageProperty page_property;
  3: EquivEdge equiv;
  4: PageViewEdge page_view;
  5: LinkedEdge page_link;
}
```

ILLUSTRATION

- Limitations of serialization frameworks
  - They only check that all required fields are present and are of the expected type
  - They're unable to check richer properties like
    - "Ages should be nonnegative"
    - "true-as-of timestamps should not be in the future."
  - Two approaches to work around these limitations
    - Wrap your generated code in additional code
    - Check the extra properties at the very beginning of your batch-processing workflow