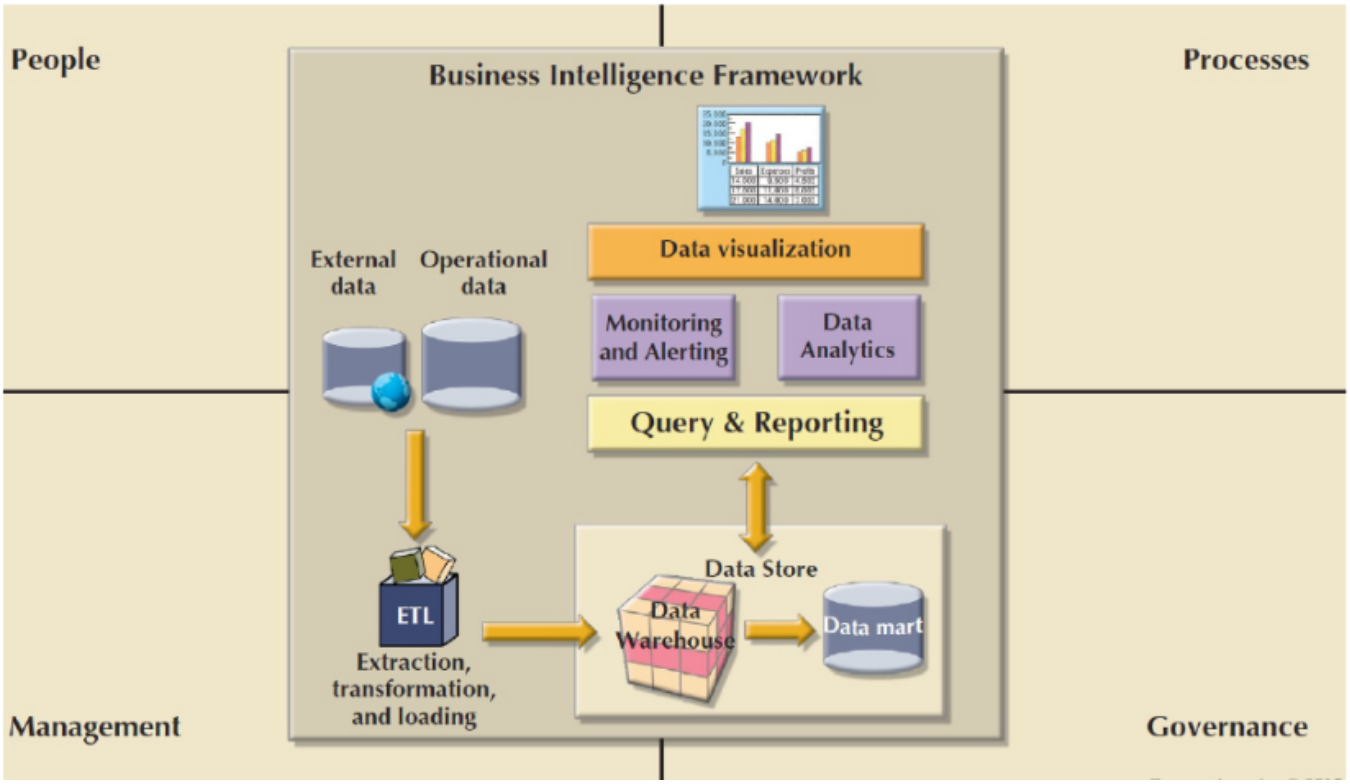
Business Intelligence(BI)

Definition of BI

Framework



What is BI

- Comprehensive, cohesive, integrated set of tools and processes
- Used for capture, collect, integrates, stores, and analyze data

Purpose

- generate and present information to support business decision making
- Allows a business to transform:
  - Data to information
  - Information into knowledge
  - Knowledge into wisdom

Benefits

- Improved decision making
- Integrating architecture
- Common user interface for data reporting and analysis
- Common data repository fosters single version of company data
- Improved organizational performance

Evolution of BI Information Dissemination Formats

Centralized Reporting(1970s) -> Spreadsheets(1980s) -> Enterprise Reporting, OLAP(1990s) ->Dashboards, mobile BI(2000s +)

OLAP(Online analytical processing): a technology that organizes large business databases and supports complex analysis

Operational Data & Decision Support Data

Effectiveness of BI depends on quality of data gathered at operational level

Transactional, operational data: individual units.

'Analytical', decision-support data: aggregated.

Operational Data

- Seldom well-suited for decision support tasks
- Stored in relational database with highly normalized structures
- Optimized to support transactions representing daily operations

Decision Support Data

Differ from operational data in:

- Time span
- Granularity
  - Drill down: decomposing a data to a lower level
  - Roll up: aggregating a data into a higher level
- Dimensionality

Contrasting Operational and Decision Support Data Characteristics

CHARACTERISTIC	OPERATIONAL DATA	DECISION SUPPORT DATA
Data currency	Current operations Real-time data	Historic data Snapshot of company data Time component (week/month/year)
Granularity	Atomic-detailed data	Summarized data
Summarization level	Low; some aggregate yields	High; many aggregation levels
Data model	Highly normalized Mostly relational DBMSs	Non-normalized Complex structures Some relational, but mostly multidimensional DBMSs
Transaction type	Mostly updates	Mostly query
Transaction volumes	High-update volumes	Periodic loads and summary calculations
Transaction speed	Updates are critical	Retrievals are critical
Query activity	Low to medium	High
Query scope	Narrow range	Broad range
Query complexity	Simple to medium	Very complex
Data volumes	Hundreds of gigabytes	Terabytes to petabytes

Decision Support Database Requirements

Characteristics of Data Warehouse Data and Operational Database Data

CHARACTERISTIC	OPERATIONAL DATABASE DATA	DATA WAREHOUSE DATA
Integrated	Similar data can have different representations or meanings. For example, Social Security numbers may be stored as ###-##-#### or as #########, and a given condition may be labeled as T/F or 0/1 or Y/N. A sales value may be shown in thousands or in millions.	Provide a unified view of all data elements with a common definition and representation for all business units.
Subject-oriented	Data are stored with a functional, or process, orientation. For example, data may be stored for invoices, payments, and credit amounts.	Data are stored with a subject orientation that facilitates multiple views of the data and decision making. For example, sales may be recorded by product, division, manager, or region.
Time-variant	Data are recorded as current transactions. For example, the sales data may be the sale of a product on a given date, such as $342.78 on 12-MAY-2014.	Data are recorded with a historical perspective in mind. Therefore, a time dimension is added to facilitate data analysis and various time comparisons.
Nonvolatile	Data updates are frequent and common. For example, an inventory amount changes with each sale. Therefore, the data environment is fluid.	Data cannot be changed. Data are added only periodically from historical systems. Once the data are properly stored, no changes are allowed. Therefore, the data environment is relatively static.

Database schema

- must support complex, non-normalized data representations (需要支持复杂的，非标准化的数据展示)
- data must be aggregated and summarized （数据应该是整合起来的）
- queries must be able to extract multidimensional time slides （数据可以基于时间切出多个维度）

Data extraction and loading

- allow batch and scheduled data extraction （允许批量以及计划的数据提取）
- support different data sources and check for inconsistent data or data validation rules （支持多数据源，检查数据完整性，或者有数据校验规则）
- support advanced integration, aggregation, and classification （支持高级的数据整合与分类）

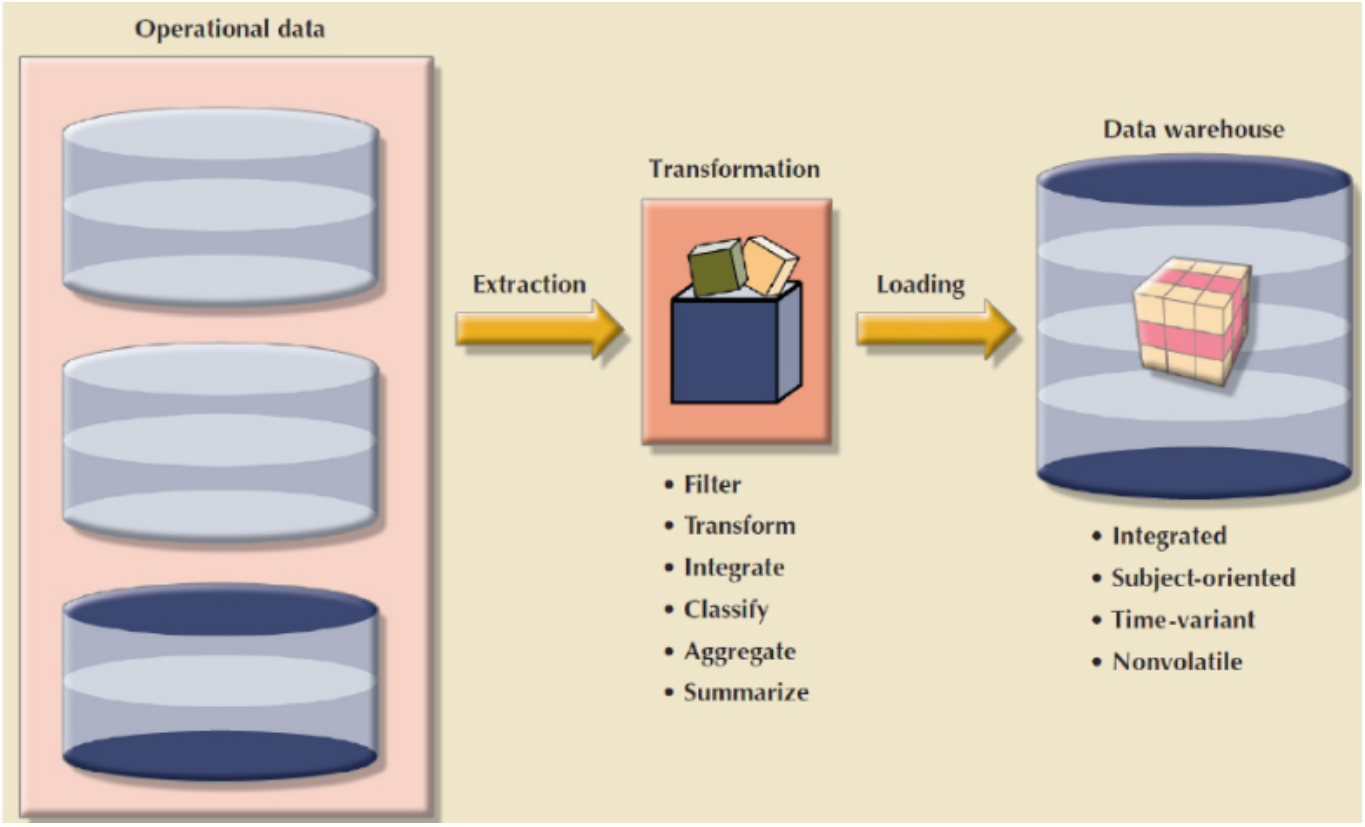
Database size should support

- very large database (VLDBs) (超大型数据库)
- advanced storage technologies (高级存储技术)
- Multiple-processor technologies （多处理器技术）

ETL(Extract Transformation Loading)

ETL is a classic "schema on write" process, where we define a schema (table structure) first, THEN load the data into that structure.

Process of ETL



Data Marts

Definition

- Small, single-subject data warehouse subset (data warehouse的子集，单个的data warehouse)
- provide decision support to a small group of people

Benefits over data warehouses

- Lower cost and shorter implementation time
- Technologically advanced
- Inevitable people issues

Star Schema

Definition

- Data-modeling technique
- Maps multidimensional decision support data into a relational database
- Creates the near equivalent of multidimensional database schema from existing relational database（从现有的关系型数据库中创造近似于多维度的数据库构架）
- Yield an easily implemented model for multidimensional data analysis

Components of Star Schemas

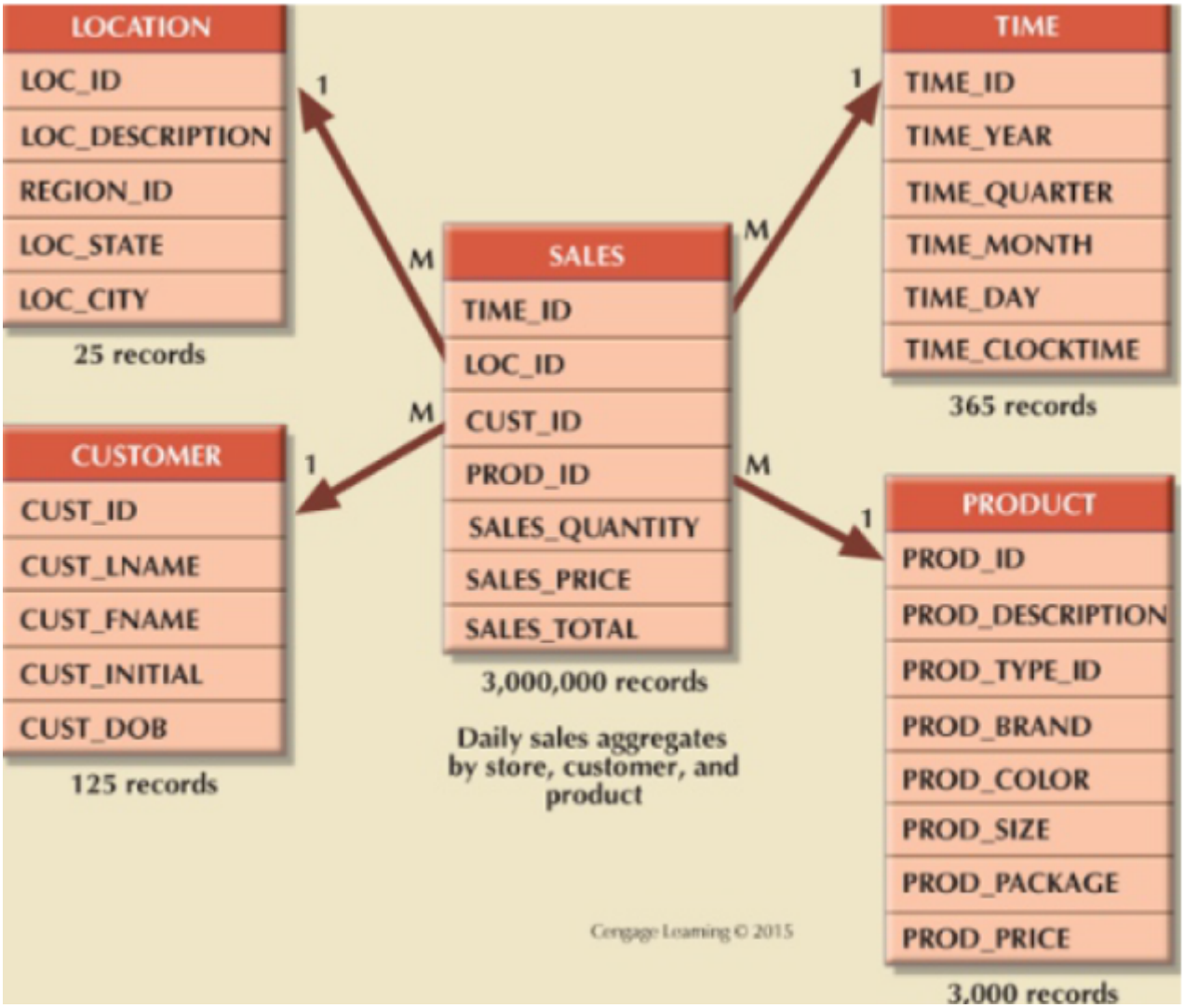
- Facts
  - Numeric values that represent a specific business aspect
- Dimensions
  - Qualifying characteristics that provide additional perspectives to a given fact

- Attributes
  - Used to search, filter, and classify facts
  - Slice and dice
    - ability to focus on slices of the data cube for more detailed analysis
- Attributes Hierarchy
  - provides a top-down data organization

Star Schema Representation

- Fact and dimensions represented by physical tables in data warehouse database
- Many-to-one(M:1) relationship between fact table and each dimension table
  - related by foreign keys
  - subject to primary and foreign key constraints
- Primary key of a fact table
  - Is a composite PK，because the fact table is related to many dimension tables
  - always formed by combining the foreign keys pointing to the related dimension tables （通过合并来自相关的dimension tables的foreign keys来生成Fact table）

A sample star schema



We have the fact table (of transactions) at the center, and denormalized (all-in-one) dimension tables all around.

Here is another representation.

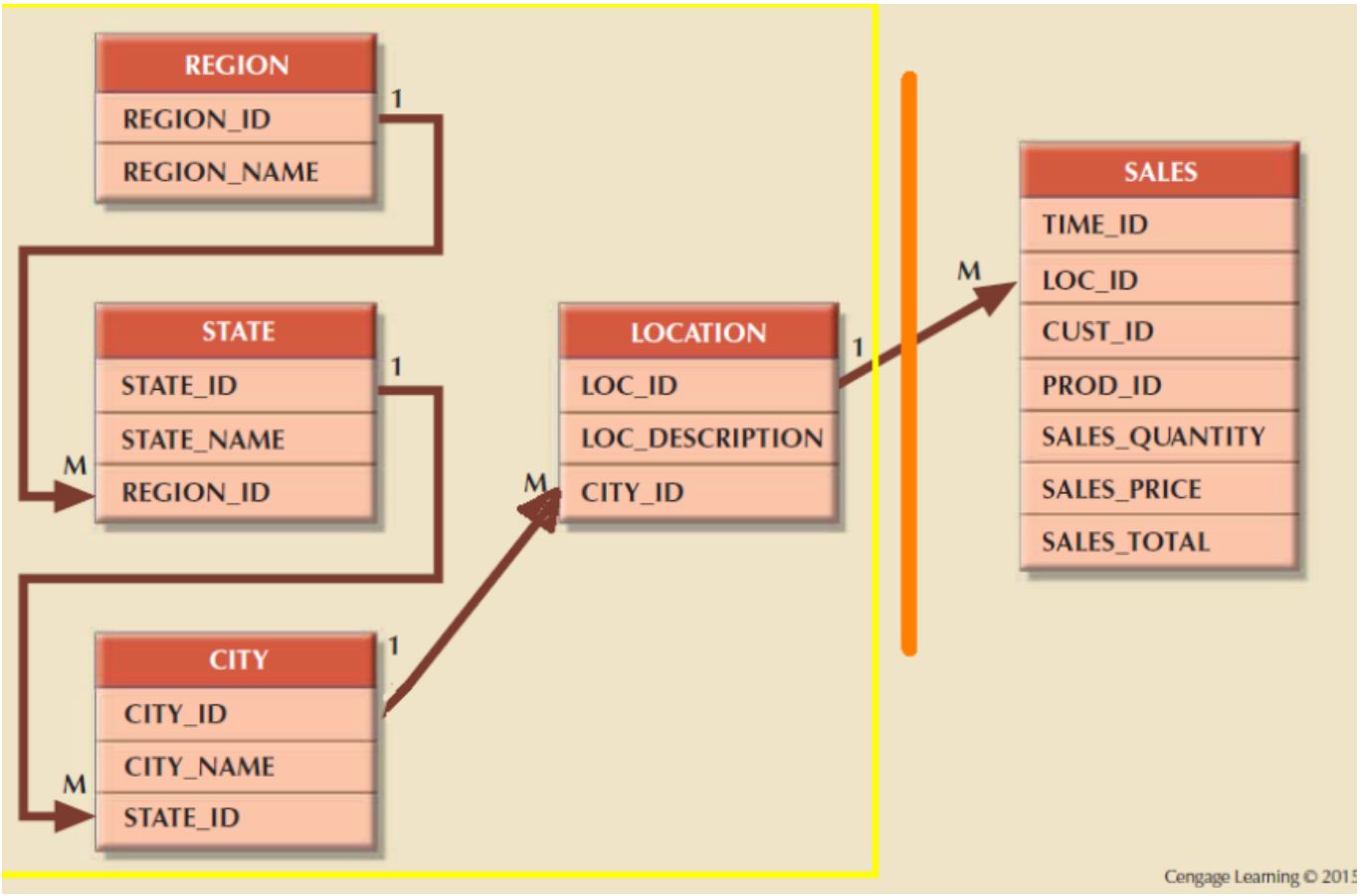
Note: "dimensions are qualifying characteristics that provide additional perspectives to a given fact; dimensions provide descriptive characteristics about the facts through their attributes."

Each fact (transaction) can now pictured to be located in a multi-dimensional cube where the axes are dimensions. Eg. a 3D representation

Slicing and dicing the cube provides specific insights..

Additionally, an attribute hierarchy would provide drill-down/roll-up capability as well, eg.

Snowflake schema



Dimensional tables can be normalized so that they have their own dimensional tables - this is done to simplify the design, but requiring navigation across the normalized chains.

Here is another representation.

Techniques Used to Optimize Data Warehouse Design

- Normalizing dimensional tables
  - Snowflake schema: Dimension tables can have their own dimension tables
- Maintaining multiple fact tables to represent different aggregation levels 使用多个fact tables来表达不同层级的聚合
- denormalizing fact tables 将fact table去规范化

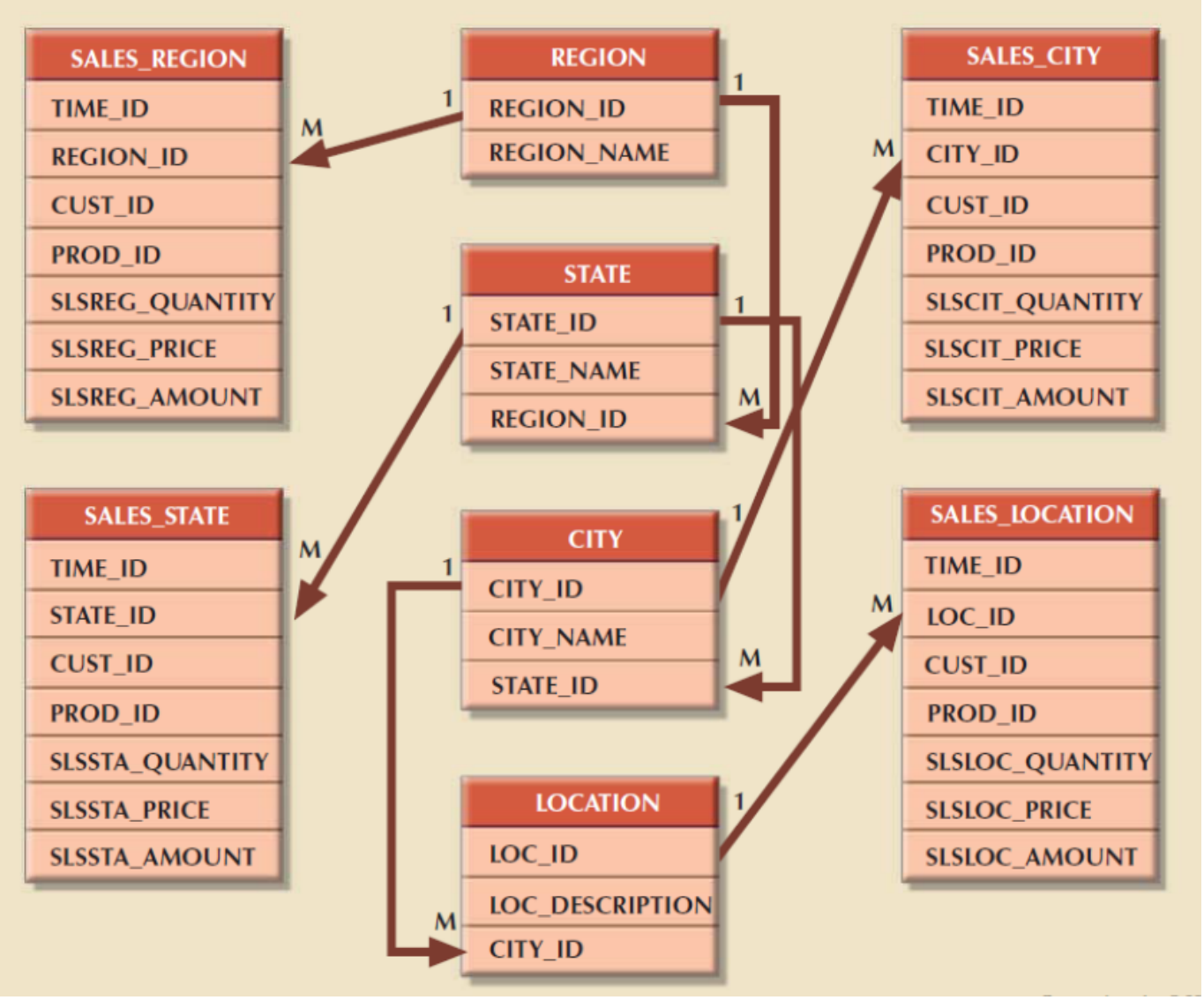
Four different ways in which we can organize (structure) a data warehouse

1. star schema: fact table FKs point to a single level of dimension tables (points of a star)
2. snowflake schema: each dimension table can be normalized to create a 1:M chain
3. the fact table can be supplanted with all the columns in the star/snowflake dimensions
4. a separate fact table can be created for each attribute in a dimension hierarchy

To denormalize a fact table (#3 above), we simply add extra 'dimension' columns to it, and fill them with redundant data - this permits fast queries (no joins needed) at the expense of disk space (and cleanliness of design).

Redundant fact tables

Instead of a denormalized fact table (#3), or a fact table pointing to denormalized star dimensions (#1), or a fact table with lowest attrs pointing to a chain of rolled-up attrs, ie. snowflake schema (#2), we can create multiple fact tables, one for each level in an attr hierarchy (#4) - it is a different form of denormalization, where the redundant data is stored in physically separate tables.



summarization

Fact tables that we see in the middle of star/snowflake schema, are ALWAYS denormalized, with multiple repeating values in the columns that link to dimensions - eg. multiple date values, product values, location values, POS terminal # values etc (because each row in a fact table contains those columns as raw 'facts').

我们在星形/雪花模式中看到的事实表总是去规范化的，在与维度相关的列中有多个重复值，如多个日期值、产品值、位置值、POS 终端 # 值等（因为事实表中的每一行都包含这些列作为原始 "事实"）。

Dimension tables, in a star schema are ALSO denormalized - eg. location dimension, with city,state,region columns, will have repeating values for states (because many cities are in each state), and repeating region values (because many states are in each region).

在星型模式中，维度表也会被去规范化--例如，包含城市、州、地区列的位置维度表会有重复的州值（因为每个州都有很多城市）和重复的地区值（因为每个地区都有很多州）。

Dimension tables in a snowflake schema are normalized, because we create a chain (hierarchy) of them using the star's dimension columns.

雪花模式中的维度表是规范化的，因为我们使用星形维度列创建了一个维度表链（层次结构）。

The fact table ALWAYS stays denormalized. Such a fact table is said to employ star schema, if we use star-like denormalized columns for BI - eg. to find out how much of a product we sold in a city, we'd query the fact rows for city name, and if we need it, can also do state-level analyses (because states are listed in the location dimension table).

事实表始终保持去规范化。如果我们在商业智能中使用类似星型的去规范化列，那么这样的事实表就是采用了星型模式--例如，要想知道我们在某个城市销售了多少产品，我们可以查询事实行中的城市名称，如果需要，还可以进行州级分析（因为州列在位置维度表中）。

Using a snowflake schema, doing location analysis for a product at a city level is similar to the above paragraph - we simply look for the city name, and if necessary, get extra info about the city (eg tax rate) by looking at the dimension table. BUT to do state level analysis, we need to follow the city->state link, and use the state-level dimension table ie traverse a branch of the snowflake.

使用雪花模式，在城市级别对产品进行位置分析与上段类似--我们只需查找城市名称，如有必要，还可通过查看维度表获取有关该城市的额外信息（如税率）。但是，要进行州级分析，我们需要按照城市->州的链接，并使用州级维度表，即遍历雪花的一个分支。

To avoid traversing those branches in a snowflake, we trade off ('waste') space by creating extra 'copies' of the fact table, where a column such as city (lowest value in the hierarchy of 'location') is REPLACED instead with 'state' values, and in another copy, with 'region' values. This lets us do star-like analyses again, because a fact row directly points the state table, and in another copy, directly points to the region table - no traversing the chain necessary (at the expense of extra storage).

为了避免在雪花中遍历这些分支，我们通过创建事实表的额外 "副本 "来换取（"浪费"）空间，在这些副本中，城市（"位置 "层次结构中的最低值）等列被替换为 "状态 "值，而在另一个副本中，被替换为 "地区 "值。这样，我们就可以再次进行星形分析，因为事实行直接指向状态表，而在另一个副本中，事实行直接指向区域表--无需遍历链（以额外存储为代价）。

Which schema (star or snowflake) is used to model the warehouse, determines whether we maintain denormalized (or normalized) dimension tables [fact tables always stay denormalized]. 'For BI purposes, the idea is to take the 'single unified view' of data which is in the fact table (which contains numerous columns (think of a single Amazon purchase order item) - they can be categorized into dimensions, and in each dimension, even be hierarchically grouped - an example would be 'location'), and DERIVE additional tables, with data pre-aggregated along those (hierarchies of) dimensions. This lets us slice-and-dice (along

dimensions), and zoom in/out (along just one dimension), all without expensive querying at runtime (on billions of rows), because the 'group by' calculations have been done already (that resulted in those aggregated data tables).'

使用哪种模式（星形或雪花形）对仓库进行建模，决定了我们是否维护去规范化（或规范化）维度表（事实表始终保持去规范化）。就 BI 而言，我们的想法是采用事实表中的 "单一统一视图 "数据（其中包含许多列（想想亚马逊的单个采购订单项目）--它们可以归类为维度，在每个维度中，甚至可以分层分组--例如 "位置"），并生成额外的表，其中的数据已按照这些（分层的）维度进行了预聚合。这样，我们就可以（按照维度）进行切分，并（只按照一个维度）进行放大/缩小，而无需在运行时（对数十亿行）进行昂贵的查询，因为'分组'计算已经完成（产生了这些聚合数据表）。

Data Analytics

Definition

- Encompasses a wide rage of mathematical, statistical, and modeling techniques to extract knowledge from data
- Subset of BI functionality

Classification of tools

- **Explanatory Analytics**: Focuses on discovering and explaining data characteristics and relationships based on existing data.
- **Predictive Analytics**: Focuses on predicting future outcomes with a high degree of accuracy

Online Analytical Processing(OLAP)

Definition

Advanced data analysis environment that supports decision making, business modeling, and operations research

Characteristics

- Multidimensional data analysis techniques
- Advanced database support
- Easy-to-use end-user interfaces

Relation vs. Multidimensional OLAP

Relational OLAP ('ROLAP')

- Provides OLAP functionality using relational databases and familiar relational tools to store and analyze multidimensional data.
- Extensions added to traditional RDBMS technology
    - Multidimensional data schema support within the RDBMS
    - Data access language and query performance optimized for multidimensional data
    - Support for very large databases(VLDBs)

Multidimensional OLAP ('MOLAP')

- Extends OLAP functionality to multidimensional database management systems(MDBMSs)
    - MDBMS: Uses proprietary techniques store data in matrix-like n-dimensional arrays
    - End users visualize stored data as a 3D data cube
        - Grow to n dimensions, becoming hypercubes
        - Held in memory in a cube cache to speed access
- Sparsity: Measures the density of the data held in the data cube

CHARACTERISTIC	ROLAP	MOLAP
Schema	Uses star schema Additional dimensions can be added dynamically	Uses data cubes Multidimensional arrays, row stores, column stores Additional dimensions require re-creation of the data cube
Database size	Medium to large	Large
Architecture	Client/server Standards-based	Client/server Open or proprietary, depending on vendor
Access	Supports ad hoc requests Unlimited dimensions	Limited to predefined dimensions Proprietary access languages
Speed	Good with small data sets; average for medium-sized to large data sets	Faster for large data sets with predefined dimensions

BI-oriented SQL extensions

ROLLUP and CUBE are GROUP BY modifiers - they help generate subtotals for a list of specified columns (see examples that follow). Depending on granularity of the columns (eg. US_REGION vs STORE_NUMBER), these subtotals help provide a rolled-up (aggregated) or drilled-down (detailed) analysis of data.

ROLLUP

- Used with GROUP BY clause to generate aggregates by different dimensions
- Enables subtotal for each column listed except for the last one, which gets a grand total
- Order of column list important

CUBE

- Used with GROUP BY clause to generate aggregates by the listed columns
- Includes the last column

Data Lakes

A 'traditional' data warehouse is an ETL-based, historical record of transactions - very RDB-like (schema-on-write).

A 'modern' alternative is a 'data lake', which offers a more continuous form of analytics, driven by the rise of unstructured (semi-structured, really) data, streaming, cloud storage, etc. In a data lake, data is NOT ETLd, rather, it is stored in its 'raw' ("natural") form [even incomplete, untransformed...] - it is 'schema on read', where we create a schema AFTER storing (raw) data in a DB.

Also, look up 'lakehouse' (p.19-p.34 in particular), and 'reverse ETL'.

Spatial DBs

What is Spatial DB?

"A spatial database is a database that is optimized to store and query data related to objects in space, including points, lines and polygons."

In other words, it includes objects that have a SPATIAL location (and extent). A chief category of spatial data is geospatial data - derived from the geography of our earth.

geographic data

- has location
- has size
- is [auto-correlated](https://www.sciencedirect.com/topics/computer-science/spatial-autocorrelation#:~:text=Spatial autocorrelation is the term,together to have similar values])
- scale dependent
- might be temporally dependent too

Spatial data analysis

- entity view: space as an area filled with a set of discrete objects
- field view: space as an area covered with essentially continuous surfaces

For our purposes, we will adopt the 'entity' view, where space is populated by discrete objects (roads, buildings, rivers..).

Components

So a spatial DB is a collection of the following, specifically built to handle spatial data:

- types
- operators
- indices

What can be plotted on to a map?

- crime data
- spread of disease, risk of disease [look at this too]
- drug overdoses - over time
- census data
- income distribution, home prices
- locations of Starbucks (!)
- (real-time) traffic
- agricultural land use, deforestation

Where does spatial data come from?

- CAD: user creation

- CAD: reverse engineering
- maps: cartography (surveying, plotting)
- maps: satellite imagery
- maps: 'copter, drone imagery
- maps: driving around
- maps: walking around

what to store for Spatial data?

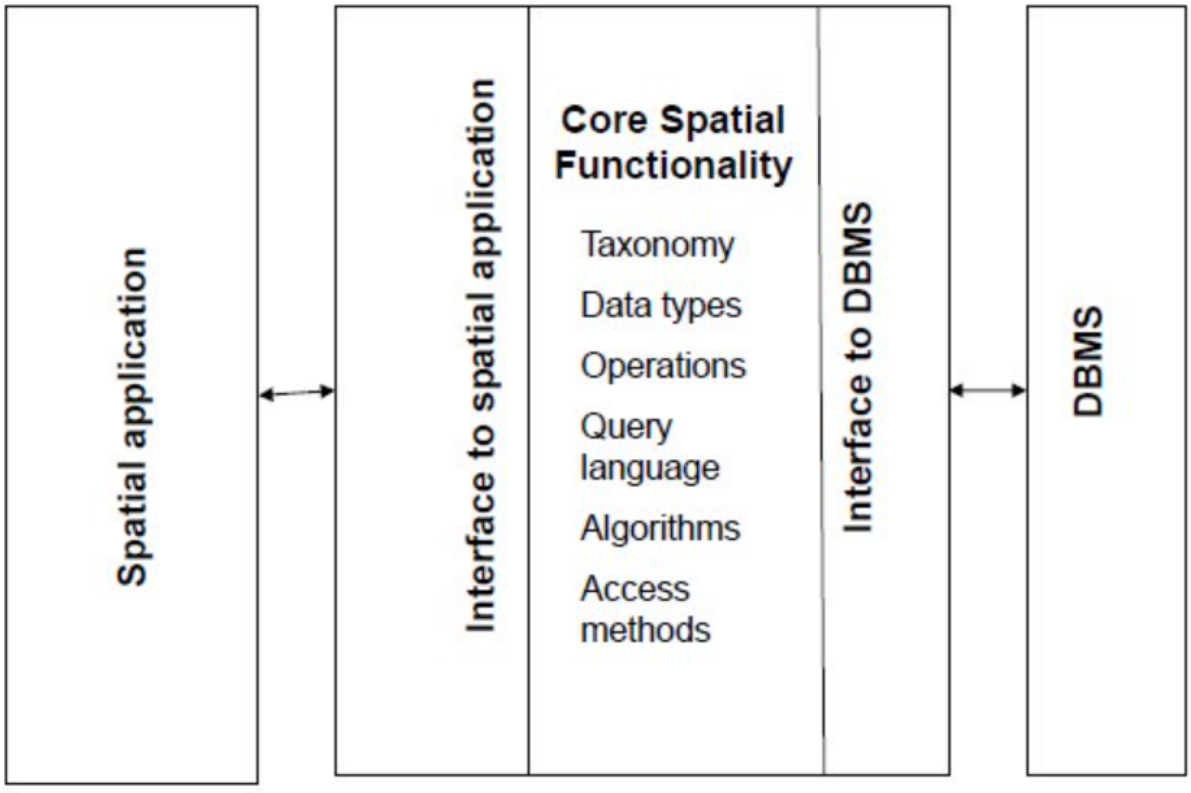
- points/vertices/nodes
- polylines/arcs/linestrings
- polygons/regions
- pixels/raster

Once we have spatial data (points, lines, polygons), we can:

- 'model' features such as lakes, soil type, highways, buildings etc, using the geometric primitives as underlying types
- add 'extra', non-spatial attributes/features to the underlying spatial data

SDBMS architecture

Spatial application <-> interfase to spatial applications/Core Spatial Functionality(data types, Operations, Algos, Query language, ...)/interface to DBMS <-> DBMS



GIS vs SDBMS

GIS is a specific application architecture built on top of a [more general purpose] SDBMS.

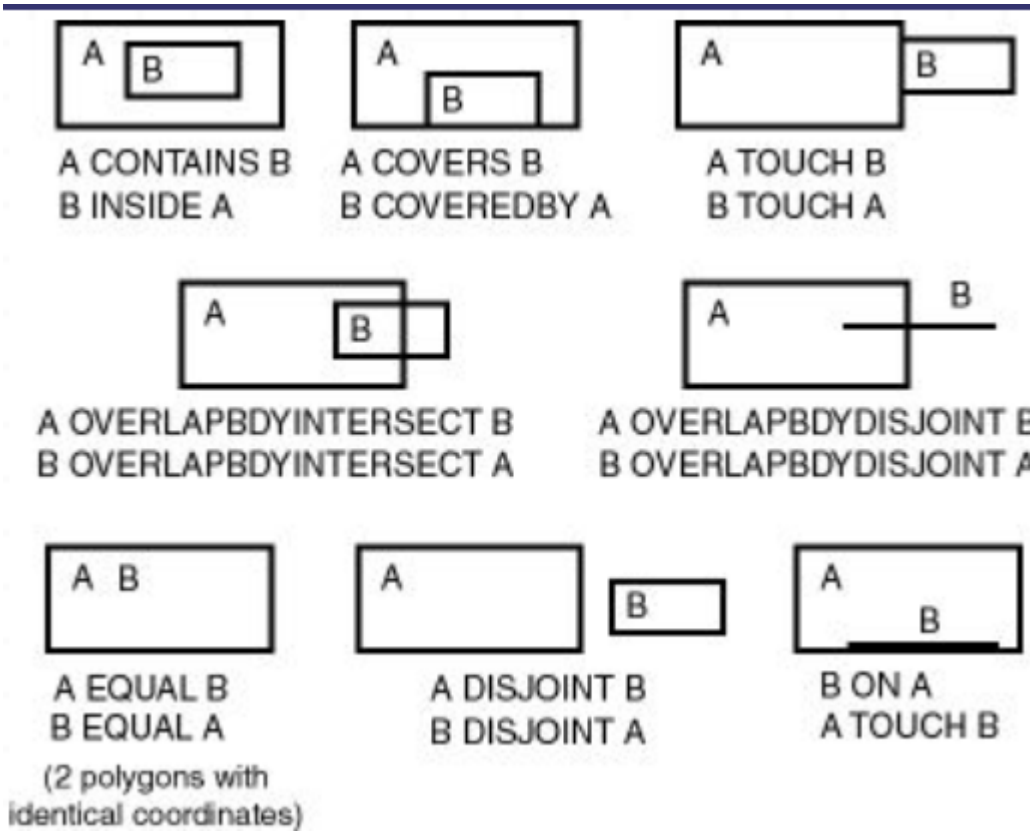
GIS typically tend to be used for:

Search	Thematic search, search by region, (re-)classification
Location analysis	Buffer, corridor, overlay
Terrain analysis	Slope/aspect, catchment, drainage network
Flow analysis	Connectivity, shortest path
Distribution	Change detection, proximity, nearest neighbor
Spatial analysis/Statistics	Pattern, centrality, autocorrelation, indices of similarity, topology: hole description
Measurements	Distance, perimeter, shape, adjacency, direction

Spatial relationships

In 1D (and higher), spatial relationships can be expressed using 'intersects', 'crosses', 'within', 'touches' (these are T/F predicates).

Here is a sampling of spatial relationships in 2D:



Categories

Spatial relationships can be:

- topology-based [using defns of boundary, interior, exterior]
  - proximity 接近性,不同features间距离的关系
  - overlap 重叠性，不同features之间重叠
  - containment 包含性，一个feature在一个空间内

- metric-based [distance/Euclidian, angle measures]
- direction-based
- network-based [eg. shortest path]

How to use these relations

We can perform the following, on spatial data:

- spatial measurements: find the distance between points, find polygon area..
- spatial functions: find nearest neighbors..
- spatial predicates: test for proximity, containment..

Spatial operators, functions

Spatial Functions

Returns a geometry

- Union
- Difference
- Intersect
- XOR
- Buffer
- CenterPoint
- ConvexHull

Returns a number

- LENGTH
- AREA
- Distance

Spatial Operators

Full range of spatial operators

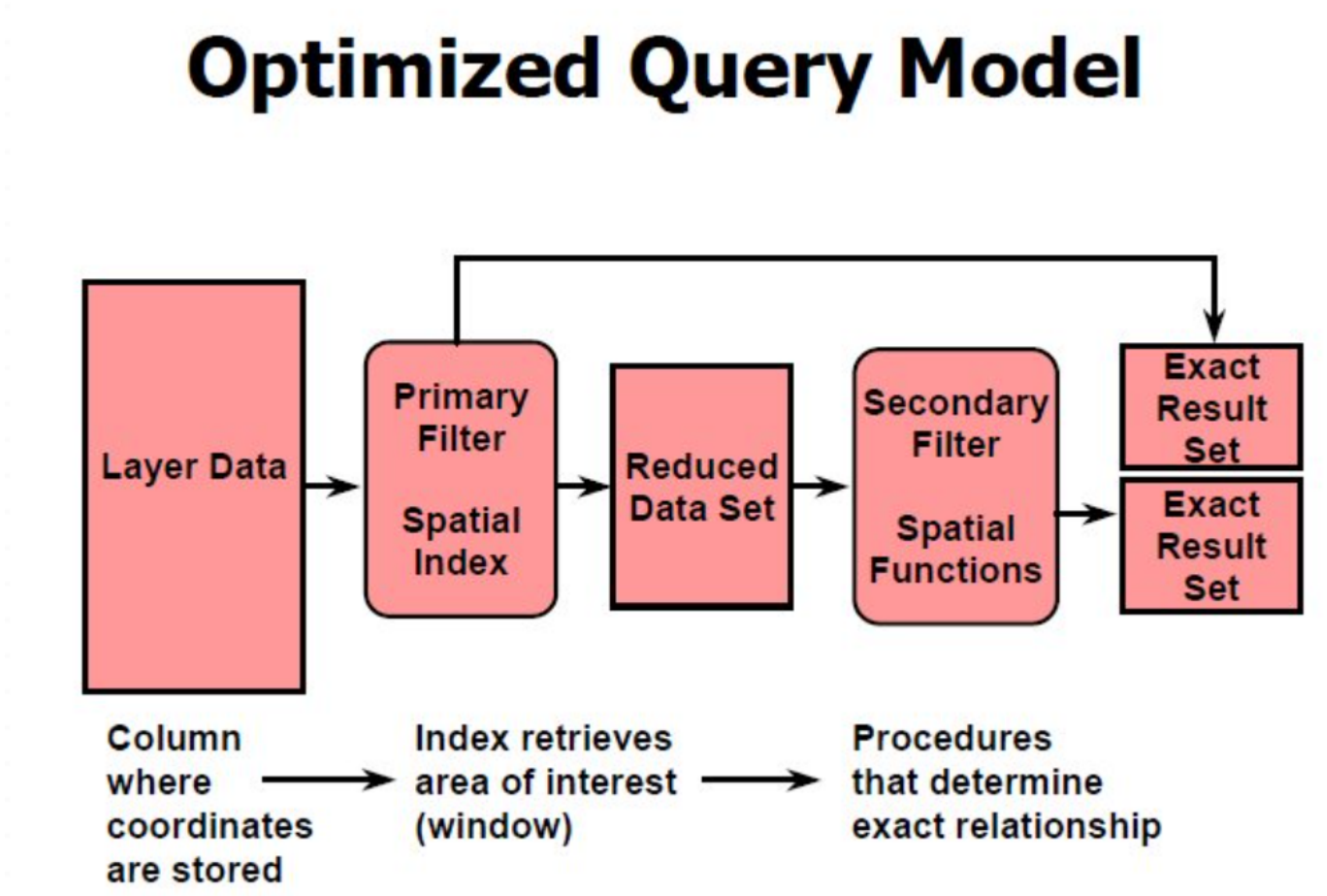
- Implemented as functional extensions in SQL
- Topological Operators
    - Inside Contains
    - Touch Disjoint
    - Covers Covered By
    - Equal Overlap Boundary
- Distance Operators
    - Within Distance
    - Nearest Neighbor

Spatial Indexing

- Used to optimize spatial query performance

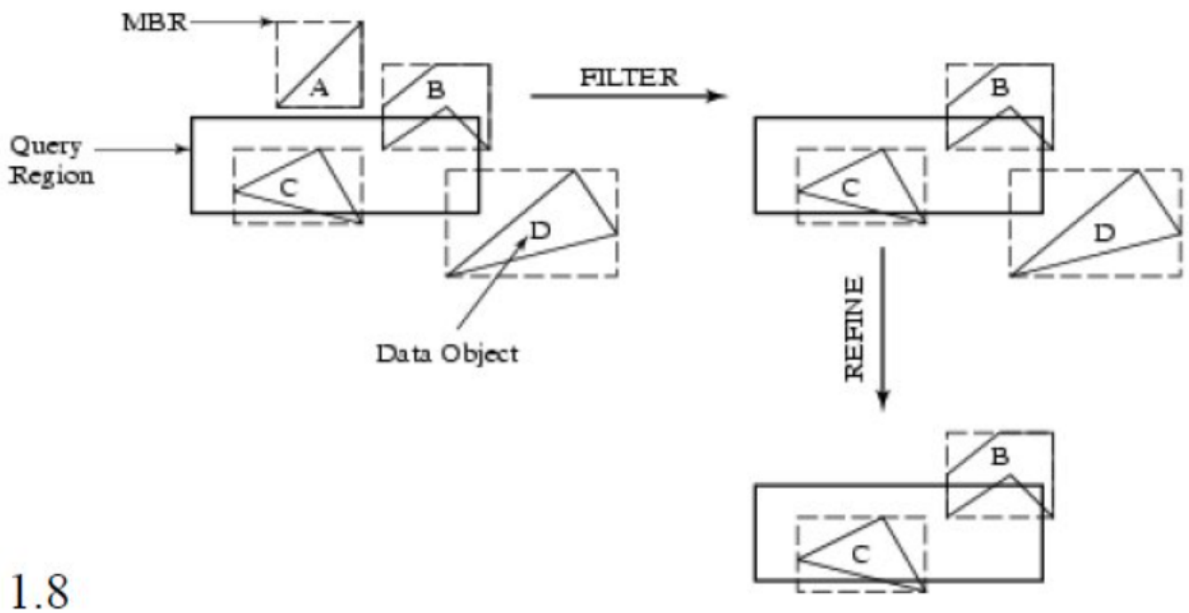
- R-tree Indexing
  - Based on minimum bounding rectangles (MBRs) for 2D data or minimum bounding volumes (MBVs) for 3D data
  - Indexes tow, three or four dimensions
- Provides an exclusive and exhaustive coverage of spatial objects
- Indexes all elements within a geometry including points, lines, and polygons

Optimized Query Model



Query Processing

- Efficient algorithms to answer spatial queries
- Common Strategy - filter and refine
  - Filter Step: Query Region overlaps with MBRs of B, C and D
  - Refind Step: Query Region overlaps with B and C



g 1.8

Visualizing spatial data

A variety of non-spatial attrs can be mapped on to spatial data, providing an intuitive grasp of patterns, trends and abnormalities. Like Dot map, Proportional symbol map, Diagram map, Choropleth maps

Spatial extensions from companies

- Oracle: Locator, Spatial, SDO
- Postgres: PostGIS
- DB2: Spatial Datablade
- Informix: Geodetic Datablade
- SQL Server: Geometric and Geodetic Geography types
- MySQL: spatial library comes 'built in'
- SQLite: SpatiaLite
- Google KML （用来encode google earth的空间数据）
  - 样例链接
- OPEN LAYER
  - 链接
- ESRI
  - is the home of the powerful, flexible family of ArcGIS products - and they are local
  - https://www.esri.com/en-us/home
- 其他
  - QGIS
    - https://www.qgis.org/en/site/
  - MapBox
    - https://www.mapbox.com/
  - Carto
    - https://carto.com/
  - GIS Cloud
    - https://www.giscloud.com/

NoSQL

Big Data

3 Vs of Big Data: volume, variety, velocity

Big Data can lead to, or result from, 'datafication'.

Why we need NoSQL?

- lots of new data, new types of data, are being rapidly generated
- developers are finding it hard to 'shoehorn' all this data into a relational model
- also hard to scale up to fit more data, more users
- and, hard to keep up performance too

So we need a **flexible, efficient, available, scalable** solution/DB design! THAT is what NoSQL provides - **high performance, high availability at a large scale**.

- avoidance of complexity [eg. no need to worry about immediate consistency]
- high throughput
- easy, quick scalability
- ability to run on commodity hardware
- avoidance of need for O-R mapping
- avoidance of complexity associated with cluster setup
- ability to use DB APIs for various programming languages, that mirror the languages' own structures

NoSQL

Means

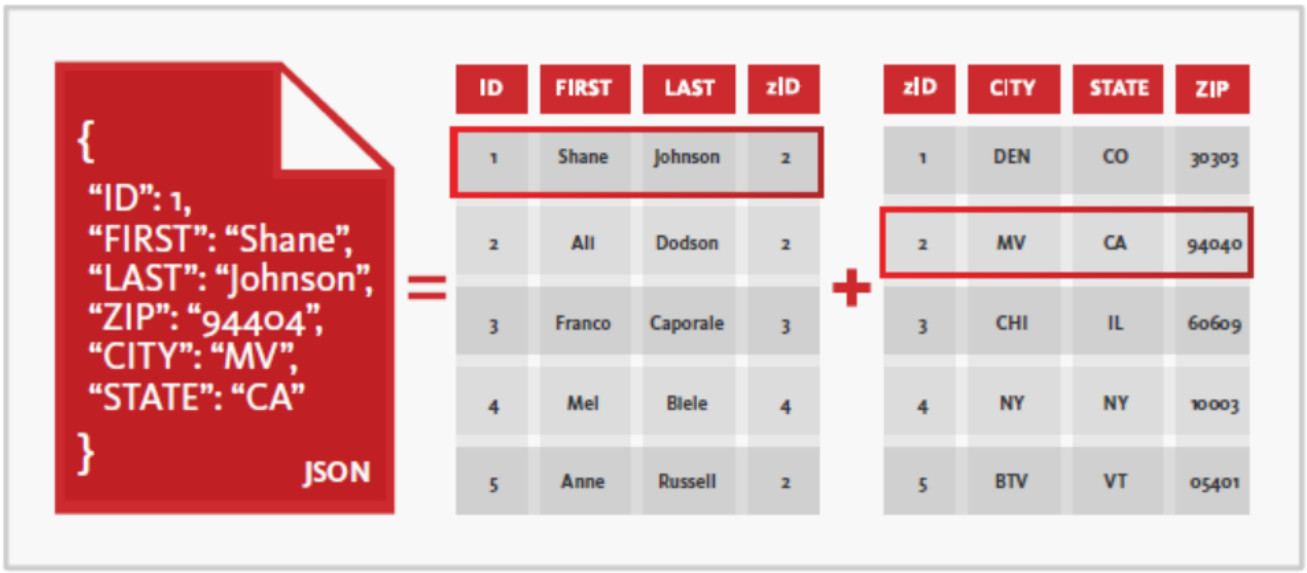
- Non relational, non SQL
- NO SQL
- NotOnly SQL

Characters

- schema-less : no tables, no relations!
  - 没有表格，没有关系
  - NoSQL的框架在应用代码中
- flexible 灵活性: easy to add new types of data - which are semi-structured (aka unstructured) [as opposed to being structured]
  - 容易增加新的数据类型
- (data) scalable 数据可扩展
  - 可以增加更多节点
  - specifically, ability to 'scale out', ie. do 'horizontal scaling' - both terms means that we can simply add more nodes (eg. servers) to an existing cluster, to accommodate more users, or to add more data to existing users.
- fast 快速: easy to process large (massive) volumes of data

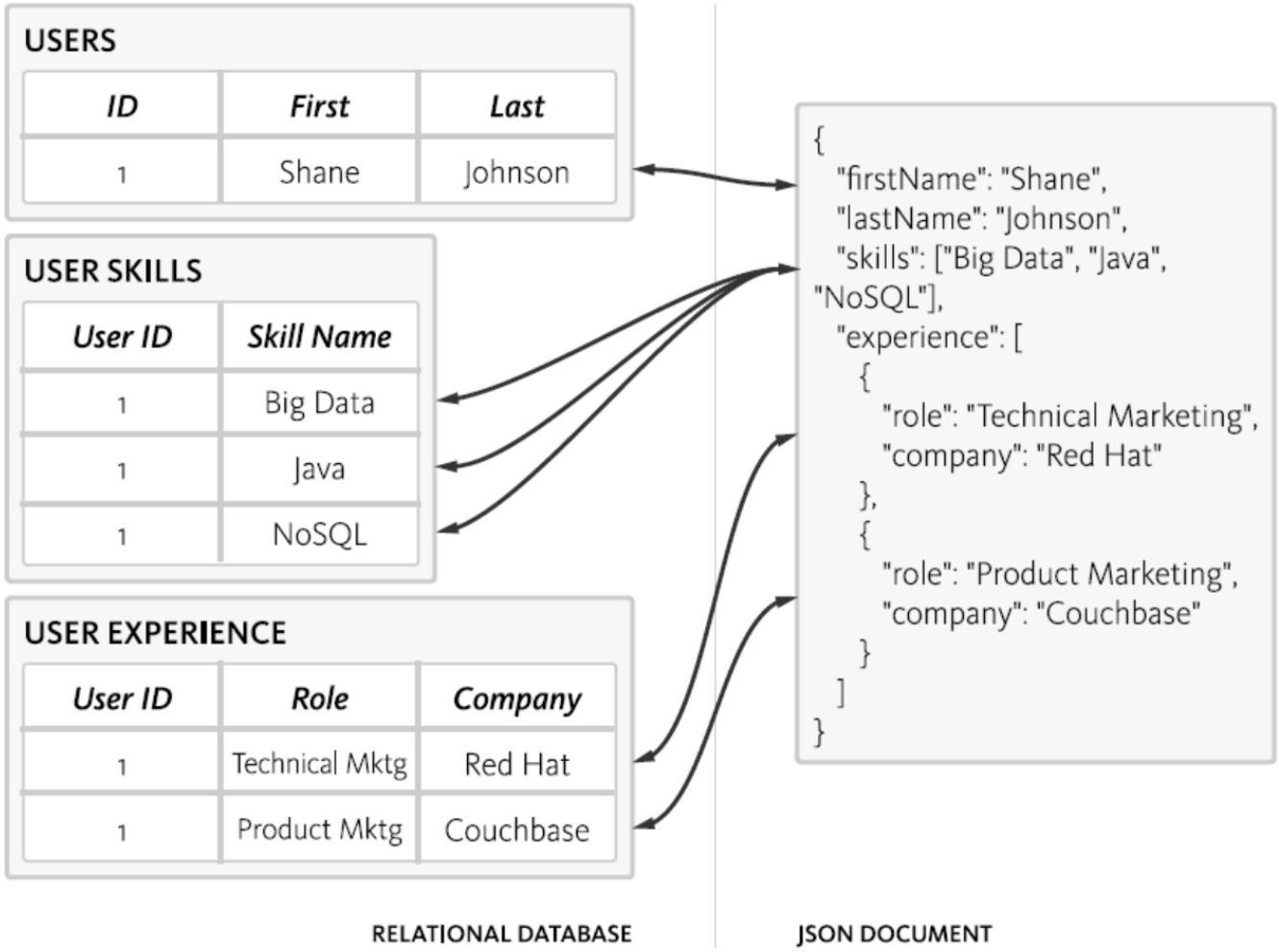
- 可以处理大量数据

JSON to store an Entire DB

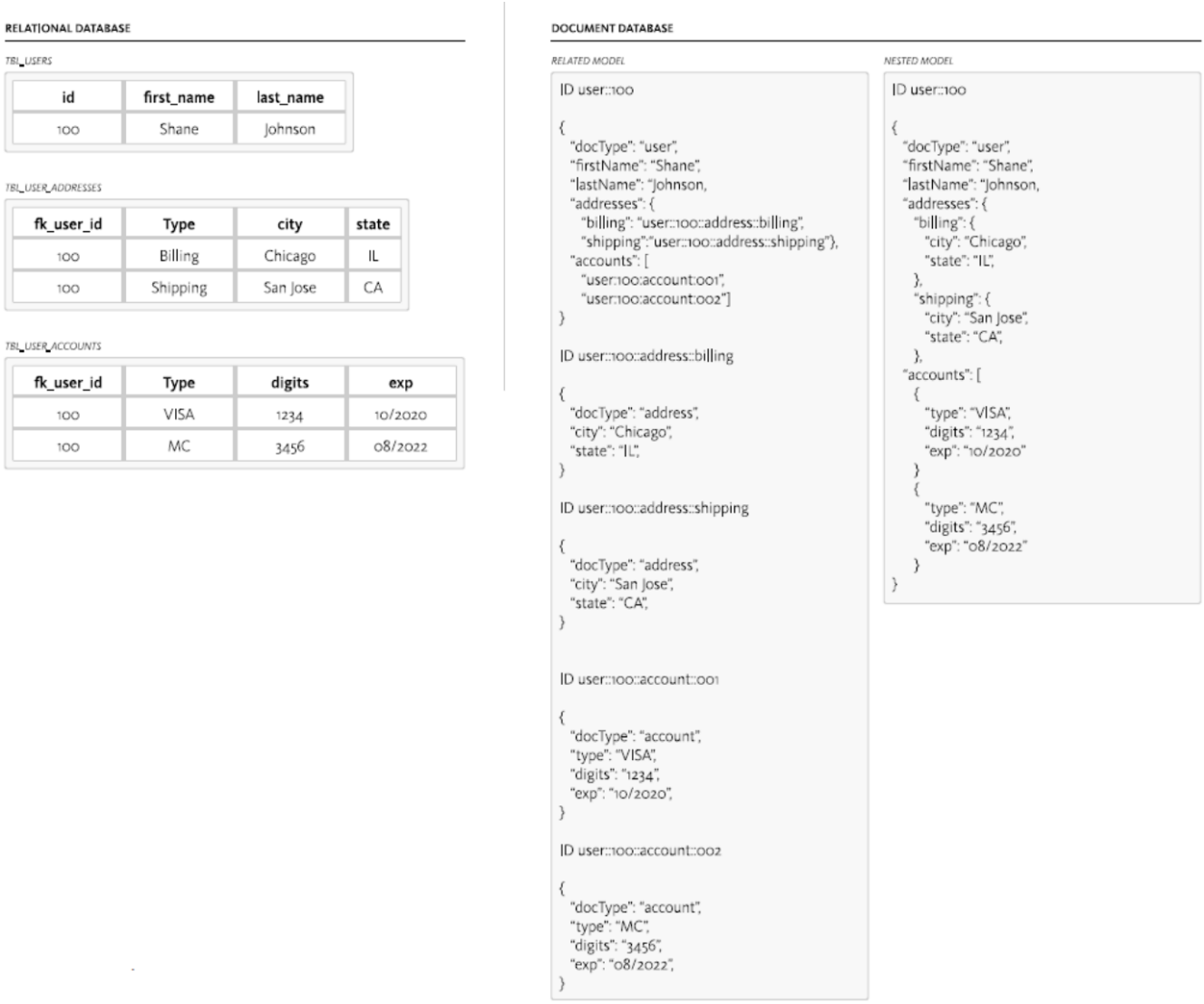


- The simplicity of the JSON scheme also makes it possible/easy to add extra data (eg metadata) to the files.
- JSON can also be returned by webservers, which can even call a client function with the returned data - this is called JSON-P.
- JSON can used to describe structured data in a specific format - this is called JSON-LD.
- JSON value can be a string, a number, a boolean, an array, or another object [a key is ALWAYS a string], **this makes for a powerful representation!**

RDB to JSON



Note - we also have the choice of using a 'related model' (where a JSON value is contained in a related (pointed-to) structure), instead of a 'nested model' (representing joined data):



RELATIONAL DATABASE

TBL_USERS

id	first_name	last_name
100	Shane	Johnson

TBL_USER_ADDRESSES

fk_user_id	Type	city	state
100	Billing	Chicago	IL
100	Shipping	San Jose	CA

TBL_USER_ACCOUNTS

fk_user_id	Type	digits	exp
100	VISA	1234	10/2020
100	MC	3456	08/2022

DOCUMENT DATABASE

RELATED MODEL

```
ID user::100

{
  "docType": "user",
  "firstName": "Shane",
  "lastName": "Johnson,
  "addresses": {
    "billing": "user::100::address::billing",
    "shipping":"user::100::address::shipping"},
  "accounts": [
    "user:100:account:001",
    "user:100:account:002"]
}

ID user::100::address::billing

{
  "docType": "address",
  "city": "Chicago",
  "state": "IL",
}

ID user::100::address::shipping

{
  "docType": "address",
  "city": "San Jose",
  "state": "CA",
}


ID user::100::account::001

{
  "docType": "account",
  "type": "VISA",
  "digits": "1234",
  "exp": "10/2020",
}

ID user::100::account::002

{
  "docType": "account",
  "type": "MC",
  "digits": "3456",
  "exp": "08/2022",
}
```

NESTED MODEL

```
ID user::100

{
  "docType": "user",
  "firstName": "Shane",
  "lastName": "Johnson,
  "addresses": {
    "billing": {
      "city": "Chicago",
      "state": "IL",
    },
    "shipping": {
      "city": "San Jose",
      "state": "CA",
    },
  "accounts": [
    {
      "type": "VISA",
      "digits": "1234",
      "exp": "10/2020"
    }
    {
      "type": "MC",
      "digits": "3456",
      "exp": "08/2022"
    }
}
```

BASE, not ACID

- Basically Available 基本可用
  - db is up most of the time
  - NoSQL允许分布式系统中某些部分出现故障，那么系统的其余部分依然可用。它不会像ACID那样，在系统出现故障时，进行强制拒绝，允许继续部分访问。
- Soft State 软状态
  - of consistency - consistency is not guaranteed while data is written to a node, or between replicas
  - NoSQL在数据处理过程中，允许这个过程，存在数据状态暂时不一致的情况。但经过纠错处理，最终会一致的。
- Eventually Consistent 最终一致性
  - at a later point in time, by push or pull, data will become consistent across nodes and replicas
  - NoSQL的软状态允许数据处理过程的暂时不一致，但是最终处理结果将是一致的，说明NoSQL对数据处理过程可以有短暂的时间间隔，也允许分更细的步骤一个一个地处理，最后数据达到一致即可。这在互联网上进行分布式应用具有其明显的优势。

A NoSQL database does not have an explicit schema that describes the relationships between its data items - it is said to be 'schema-less'

To put it differently, the DB itself "doesn't care" about what it is storing, it is the application code that imparts meaning to the data. As a result, changing the data model (eg. adding or deleting an attribute) is trivial - just write and run (application) code to make the change in the DB!

In a schema-less environment, developers use intuitive data structures (well supported by underlying host languages) to do data manipulation (including querying and updating).

ACID	BASE
Strong consistency	Weak consistency – stale data OK
Isolation	Availability first
Focus on “commit”	Best effort
Nested transactions	Approximate answers OK
Availability?	Aggressive (optimistic)
Conservative (pessimistic)	Simpler!
Difficult evolution (e.g. schema)	Faster
	Easier evolution

Types of NoSQL DB(Based on under lying data model)

- **key-value store**: DynamoDB (Amazon), Project Voldemort, Redis, Tokyo Cabinet/Tyrant..
- **column-family store**: Cassandra, HBase..
- **document store**: MongoDB, CouchDB, MarkLogic..
- **graph store**: Neo4j, HyperGraphDB, Seasame..

Polyglot persistence 多语言持久性

- use of different storage technologies (ie. NoSQL DBs) to store different parts (data) of a single application.使用不同的存储技术(NoSQL DBs) 来存储单个应用的不同部分
- These individual data stores are then tied together by the application, using DB APIs or web services APIs.通过使用应用API来统一输出
- a single application can "speak" several different storage technologies, each one leveraging its strength一个应用可以使用不同的存储技术来根据数据选择合适的存储技术优势

Key-value DB

Characteristic

- is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash.
- whole db is a dictionary, which has records ("rows") which have fields (columns).
- there is no schema, the records all don't have to contain identical fields!记录不必包含相同字段
- key-PK, value-non PK columns
- Querying occurs only on keys. Return entire value. Entire value have to be update when update
- Rapid querying of keys is possible.
- **k/v DBs are lightweight (simple), schema-less, transaction-less.**

Memcached

is a high-performance, in-memory, data caching system, with a VERY simple API:

- store (SET) a value, given a key (eg. memcache->set(key, val))
- retrieve (GET) the value, given the key (eg. val = memcache->get(key))

The value that is stored does not have to be atomic, it can even be k-v pairs, so store values are usually **small**.

Memcached is commonly used with a 'backend' SQL store (relational DB) - a COPY of the frequently accessed data is held in memcached, for fast retrieval and update.用于需要常访问的数据

Redis

和Memcached差不多

For values, rather than just atomic datatypes (number, string, boolean), Redis offers richer types such as list, set, dictionary.

Amazon's Dynamo

- the infrastructure is made up by tens of thousands of servers and network components located in many datacenters around the world.
- commodity hardware is used.
- component failure is the 'standard mode of operation'.
- 'Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services'.

"Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance."

values are stored as **BLOBS**. Operations are limited to a single k/v pair at a time.

Only two API ops are supported:

- get(key) - returns a list of objects and a context
- put(key, context, object) - no return value

'context' is used to store metadata about the BLOB values, eg. version numbers (this is used during 'eventual consistency' DB updating).

Keys are hashed using the MD5 hashing algorithm, to result in appropriate storage **locations** (nodes).

To ensure availability and durability, each k/v pair is replicated N times (N=3, commonly) and stored in N adjacent nodes starting from the key's hash's node.

Column family DBs

terminology

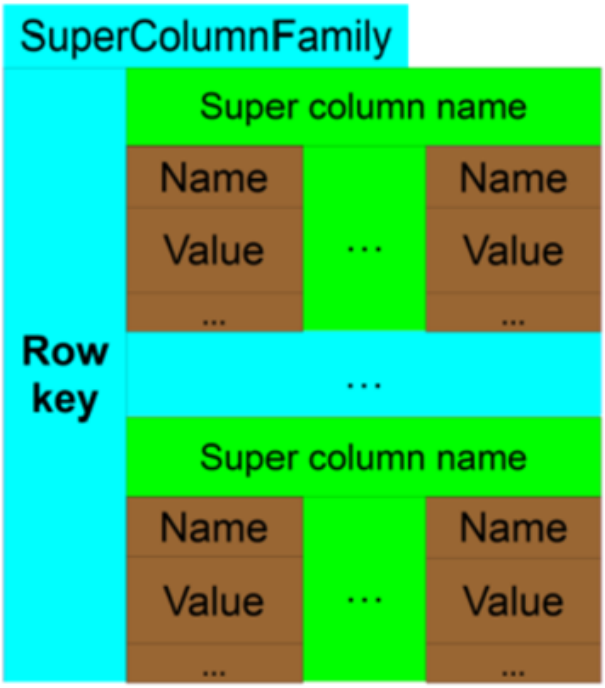
data is stored as groups of columns (column family),

- 不是处理数据行，而是处理它们的列
- 因此，此类数据库

  适用于聚合查询

  （例如，公司员工的平均年龄）

  - 当查询语句只涉及部分列时，只需要扫描相关的列
- 以及仅涉**及所有列的子集的查询**（例如，检索学生的学术信息 [但不是个人信息]）
- 主要用来做**inverted indexing**
- 适合压缩
  - 每一列的数据都是相同类型的，彼此间相关性更大，对列数据压缩的效率较高




```
Musician:                           ColumnFamily 1
    bootsy:                         RowKey
        email: bootsy@pfunk.com,    ColumnName:Value
        instrument: bass            ColumnName:Value
    george:                         RowKey
        email: george@pfunk.com     ColumnName:Value

Band:                               ColumnFamily 2
    george:                         RowKey
        pfunk: 1968-2010            ColumnName:Value
```

- **Column**: name(key):value pair

```
{
       firstname:'A','B','C',
}
```

- **Supercolumn: **is a named grouping of columns. Also a key-value pair, key is the supercolumn's name and value is column family. This is used to index values consisting of raw column data. 类似于一条数据

```
username:{firstname:'Cath', lastname:'Yoon'}
```

- **Column family**: contains columns of related data, for all rows. A column family would have many rows of data, where for each row, there would be multiple columns and values.可以理解成一个表

```
Cath:{
         firstname:'Cath',
         lastname:'Yoon'
     }
,
Terry:{
          firstname:'Terry',
          lastname:'Cho'
      }
```

- **Supercolumn family**: a collection of supercolumns. Note that a supercolumn family is ALSO equivalent to a table, but with extra info (which would be the names of its supercolumns)像是增加了表名

```
UserList={
   Cath:{
     username:{firstname:'Cath',lastname:'Yoon'}
     address:{city:'Seoul',postcode:'1234'}
   },
   Terry:{
     username:{firstname:'Terry',lastname:'Cho'}
     account:{bank:'hana',accountID:'5678'}
   }
 }
```

Google's BigTable

- a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers用于管理结构化数据的分布式存储系

统，旨在扩展到非常大的规模：数千个商品服务器中的 PB 级数据

- **wide applicability, scalability, high performance,** and **high availability**.
- The storage data structure is "a sparse, distributed, persistent multidimensional sorted map"
- Values are addressed by (row-key, column-key, timestamp).

Hypertable

- Written in C++, based on HDFS and distributed lock managing.
- Can have column-families with an arbitrary number of distinct columns.
- Tables are partitioned by ranges of row keys (like in BigTable) and the resulting partitions get replicated between servers.
- Features HQL, a C++ native API and the 'Thrift' API.

HBase

- BigTable clone written in Java, as part of the Hadoop implementation.
- An HBase db can be the source or target for a MapReduce task running on Hadoop.
- Has a native Java API, Thrift API and REST web services.

Cassandra

Data model

- Rows - which are identified by a string-key of arbitrary length. Operations on rows are atomic per replica no matter how many columns are being read or written.
- Column Families - which can occur in arbitrary number per row.
- Columns have a name and store a number of values per row which are identified by a timestamp (like in Bigtable). Each row in a table can have a different number of columns, so a table cannot be thought of as a rectangle. Client applications may specify the ordering of columns within a column family and supercolumn which can either be by name or by timestamp.
- Supercolumns have a name and an arbitrary number of columns associated with them. Again, the number of columns per super-column may differ per row.

Supercolumns have a name and an arbitrary number of columns associated with them. Again, the number of columns per super-column may differ per row.

API

- get()
- insert()
- delete()

Query language

Cassandra's query language is "CQL" - somewhat like SQL, but no WHERE, JOIN, GROUP BY, ORDER BY; also, results are returned as JSON.

UPDATE: "these days", Cassandra DOES implement full SQL-like functionality.

Documents DB

DBs: MongoDB, CouchDB, MarkLogic

terminology

Definition

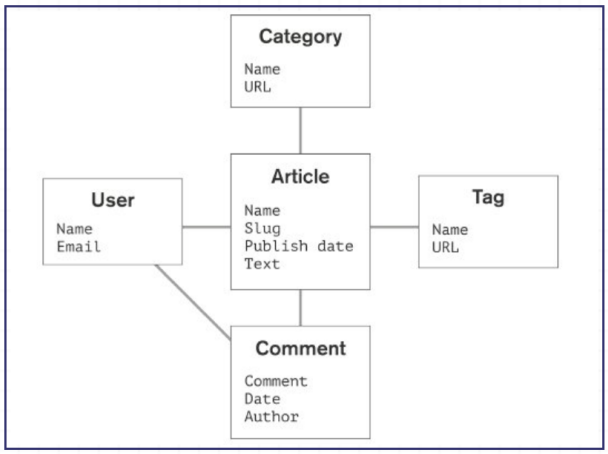
- a collection of documents
- The basic unit of storage is a document - this can be JSON, XML, etc. There can be an arbitrary number of fields (columns and values, ie. k/v pairs) in each document.
- A document DB can be considered to be a more sophisticated version of a k-v store.kv加强版

Benefits

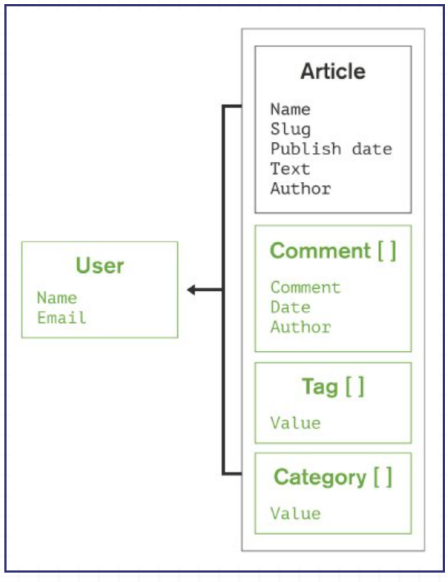
- 不需要Join来查看所有信息
- 同时queries可以通过MapReduce来并行化

Example

Consider the following diagram, which shows how blog posts could be organized:



The corresponding document store would look like this:



querying(non-SQL)

- simple k/v queries - can return a value for any field in the stored documents; usually, we do a simple PK search (given the doc's key, retrieve the entire doc)
- range queries: return values based on >, <=, BETWEEN..
- geo-spatial queries
- text queries - full text search, using AND, OR, NOT
- aggregation framework: column-oriented operations such as count, min, max, avg
- MapReduce queries - get executed via MapReduce

Graph DBs

A graph database uses (contains) graph entities such as nodes (vertices), relations (edges), and properties (k-v pairs) on vertices and edges, to store data.

index-free adjacency

A graph DB is said to be 'index free', since each node directly stores pointers to its adjacent nodes.

In a graph db, the focus is on relationships between 'linked data'.

multiple types of graphs

A rich variety of graphs can be stored and manipulated - directed graphs, trees, weighted graphs, hypergraphs, etc.

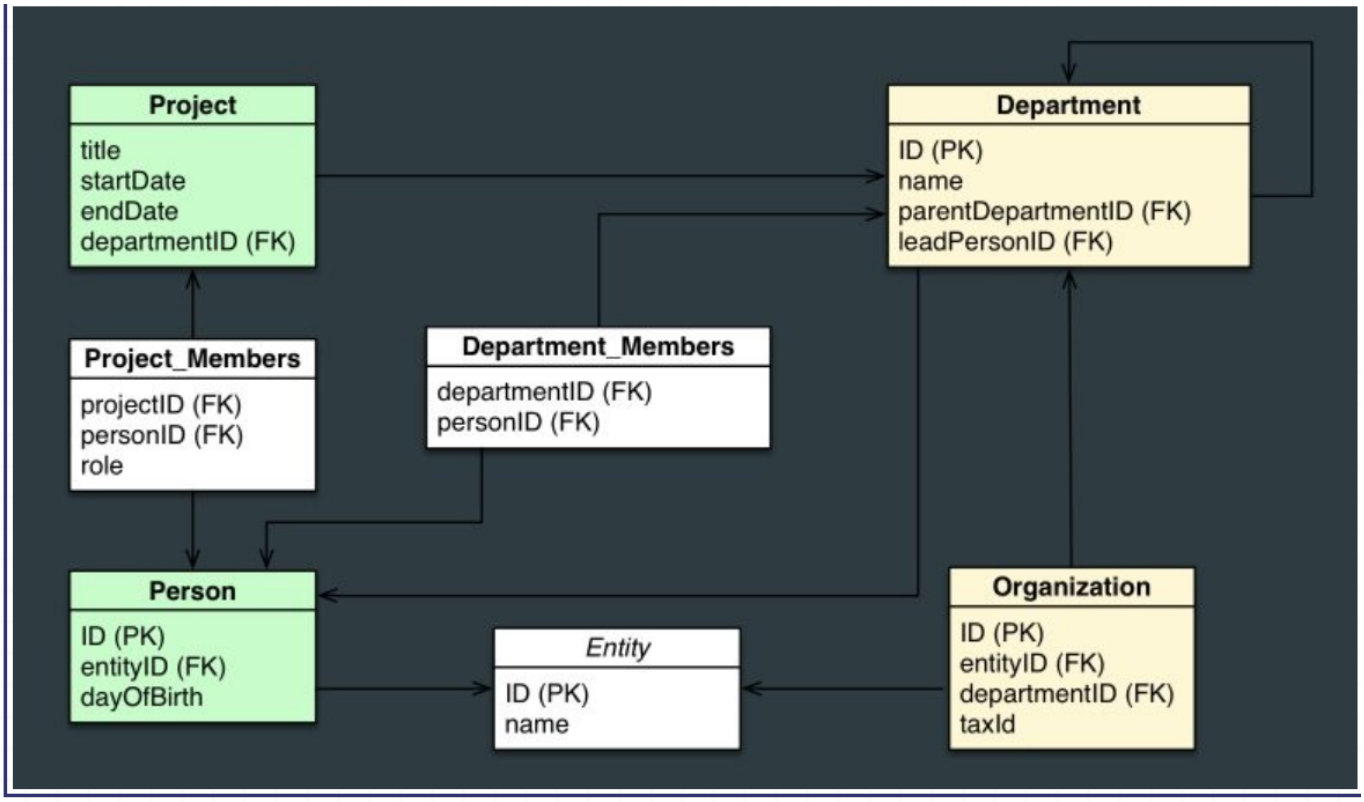
implementations

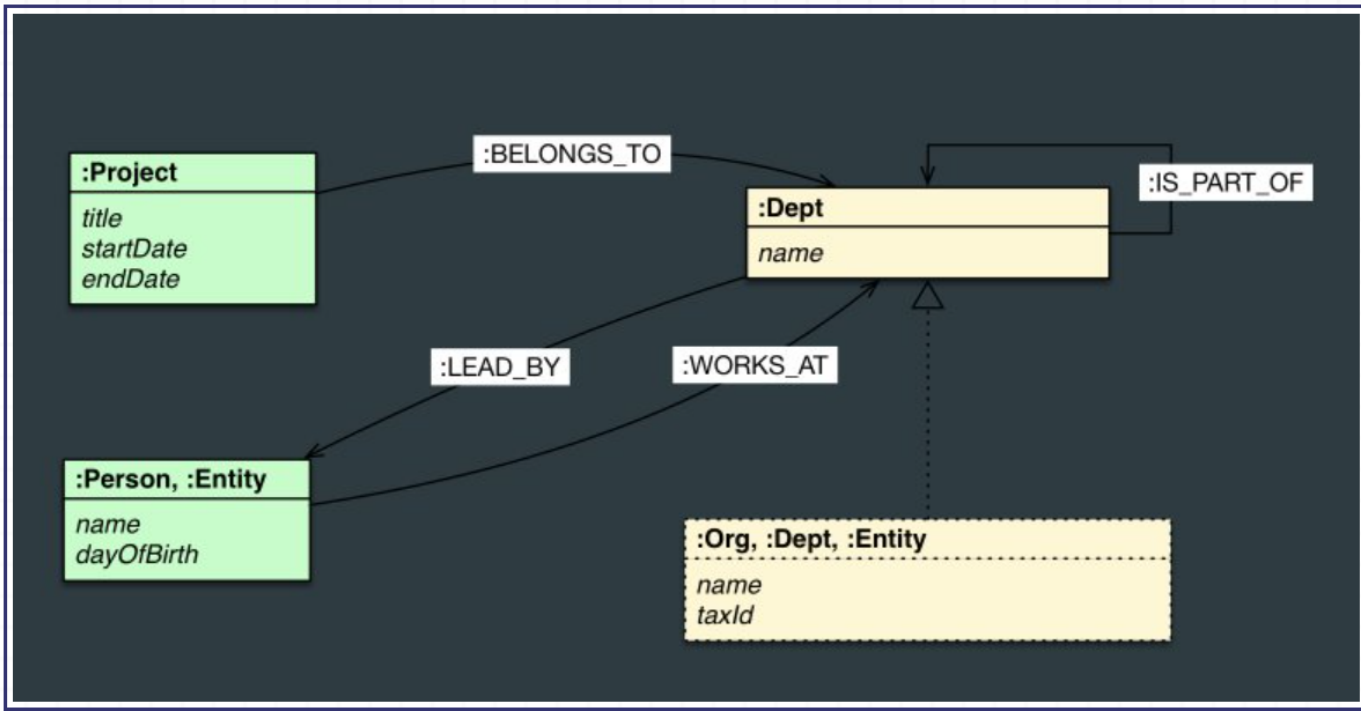
- FlockDB (from Twitter)
- Neo4J
- HyperGraphDB
- InfiniteGraph
- InfoGrid (makers of MeshBase and NetMeshBase)
- OrientDB [wasn't this also listed as a document DB?!]
- Giraph (from Apache)
- GraphLab

Uses: social networks, recommendation engines..

comparisons with relational modeling

Compared to a relational scheme, a graph offers a compact, normalized, intuitive way of expressing connections/relationships.每一个row都变成一个节点，columns变成节点的属性



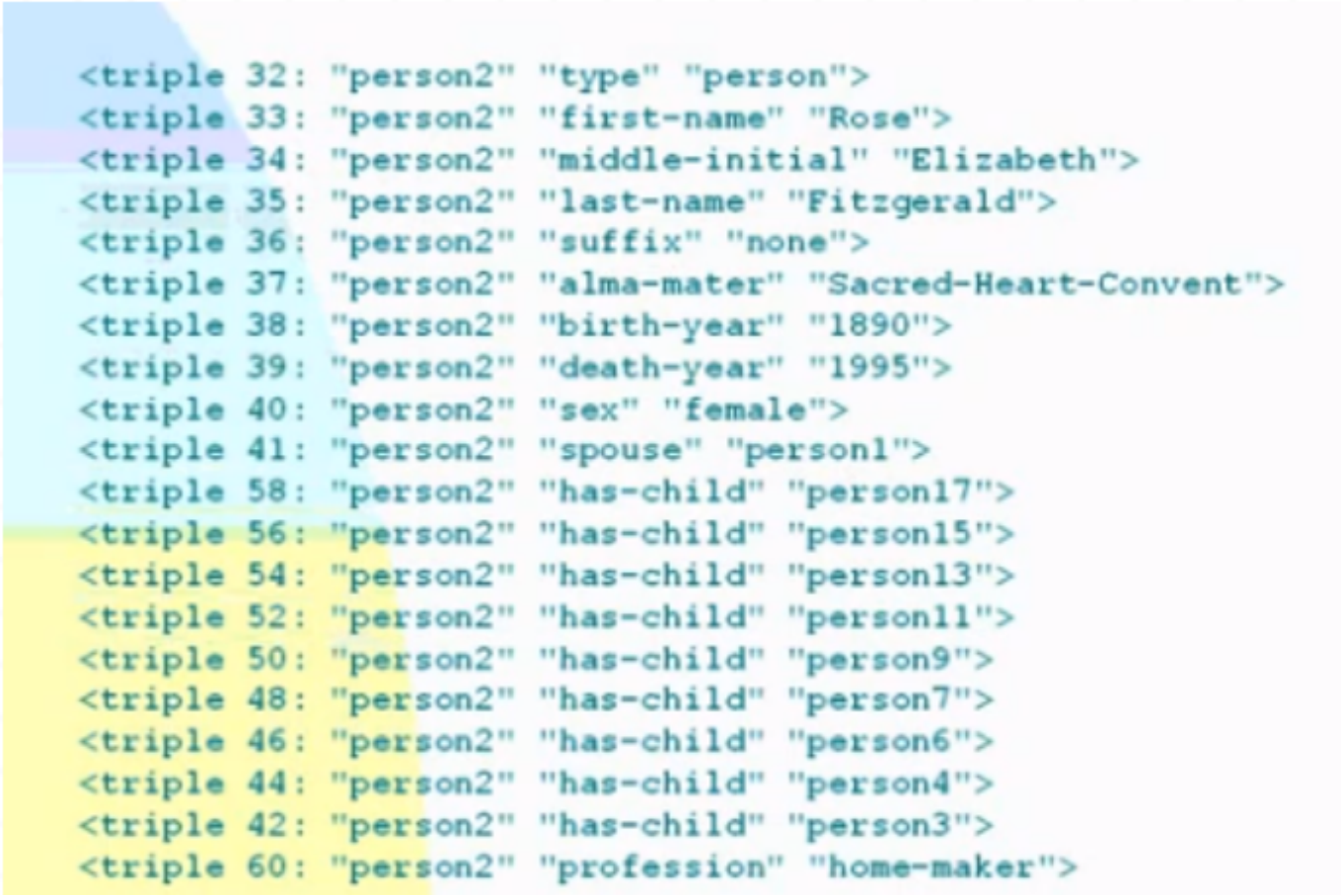


triple store (RDF) Database

- stores triples of (subject,predicate,object) [or equivalently, (subject,attribute(property),value)]
  - **Subject**: what we are describing. 描述的东西
  - **Predicate**: a property of the subject. 描述东西的属性
  - **Object**: the predicate's (property's) value. 属性的对应数值
- A triple defines a **directed binary relation**
- We issue 'semantic queries' to search a triple store DB.
- Note that a triple store DB is a restricted form of a graph DB.

Example

Here is an example of a triple store - it is a flat (non-hierarchical) list (bag) of triplets, specified as subject (node), predicate (relationship), object (another node). The column on the left shows node IDs, that's not part of the triple.



Benefits

- a triplet list can be grown 'endlessly', eventually connecting EVERYTHING to EVERYTHING ELSE! 可以无限增长，然后所有东西都可以连接在一起
- There is no schema to modify - just keep adding triplets to the DB! 没有架构，只要增加triplets就可以

implementations

- AllegroGraph
- MarkLogic
- SparkleDB
- Stardog (http://stardog.com/)

querying

- Querying a triplet store can be done in one of several RDF query languages, eg. RDQL, SPARQL, RQL, SeRQL, Versa
- The output of a triple store query is called a 'graph'.
- Queries tend to span disparate data, perform complex rule-based logic processing or inference chaining (AI-like).

Triple store DBs: equivalent to a form of RDF databases!

- RDF's serialization formats is N-Triples
- every triple store DB is an RDF DB as well (but not the other way around).

architecture

- in-memory: triples are stored in main memory
- native store: persistence provided by the DB vendors, eg. AllegroGraph
- non-native store: uses third-party service, eg. Jena SDB uses MySQL as the store

MapReduce

Comp. machinery to process large volumes of data

Modern databases store data as key/value pairs, resulting in explosive growth when it comes to number of rows and file sizes.

Traditional, sequential, single machine oriented access does NOT work at all - what is needed is a massively parallel way to process the data.

Note that we are talking about SIMD form of parallelism.

Functional construct in Python

lambda

A lambda operator lets us define anonymous, JIT functions. We can use lambda to define a function inside another function's parameter list 定义一个匿名函数

```
f = lambda x, y : x + y # f is a var that receives an anonymous fn
assignment
```

map

map() applies the function func, to a sequence (eg. a list) seq. 将函数应用在list中每一个元素上

```
r = map(func, seq)
```

filter

filter() filters (outputs) all the elements of a list l, for which the passed-in Boolean-valued function func returns True. 将list中函数返回为True的元素留下

```
fib = [0,1,1,2,3,5,8,13,21,34,55]
result = filter(lambda x: x % 2, fib)
```

reduce

reduce() applies func repeatedly to the elements of l, to generate a single value and output it.

```
reduce(lambda x,y: x+y, [47,11,42,13])
res = 113
```

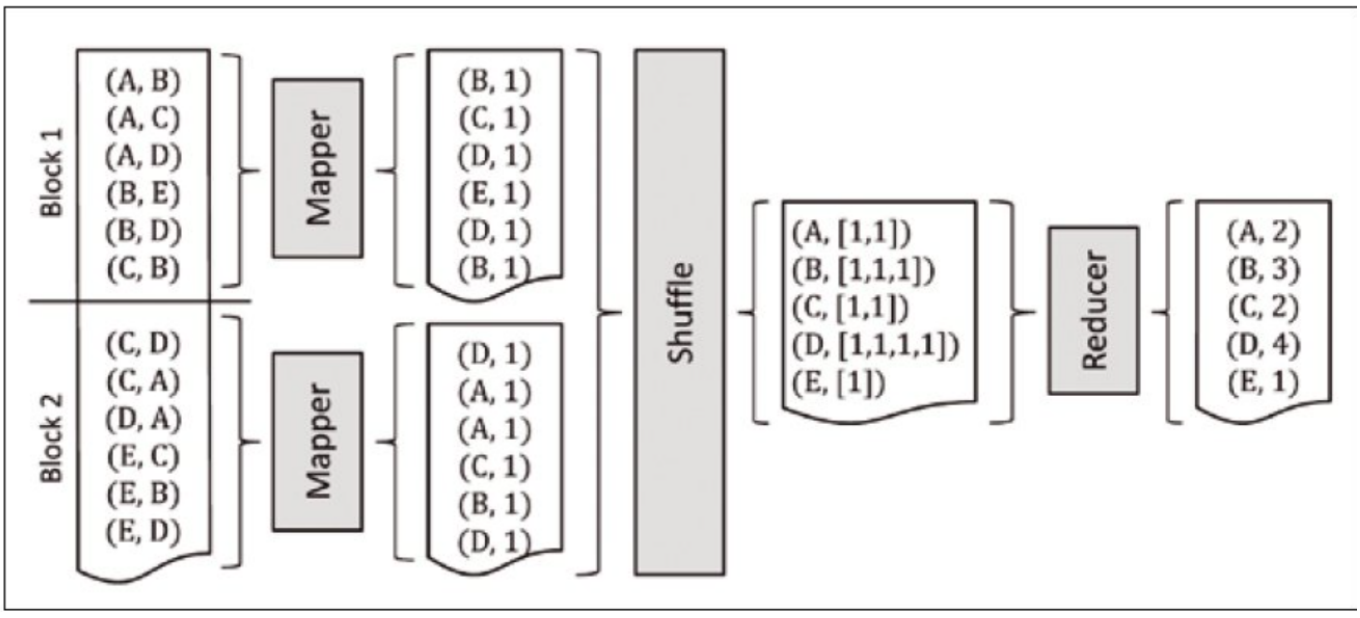
In the above, reduce() repeatedly calls the lambda function with the 'current' result ('x'), along with the next element in the list ('y'); the starter value of the current result is initialized to the first element of the incoming list (47, in our example), so the function is called with (47,11), (58,42), (100,13).

How MapReduce work

- [big] data is split into file segments, held in a compute cluster made up of nodes (aka partitions) 数据被切分成N个文件段落分散在多个节点上
- a mapper task is run in parallel on all the segments (ie. in each node/partition, in each of its segments); each mapper produces output in the form of multiple (key,value) pairs Mapper的操作会跑在所有的节点上进行操作
- key/value output pairs from all mappers are forwarded to a shuffler, which consolidates each key's values into a list (and associates it with that key) Mapper的输出会汇聚到shuffler上；将每个K,V汇聚到一个key上
- the shuffler forwards keys and their value lists, to multiple reducer tasks; each reducer processes incoming key-value lists, and emits a single value for each key Shuffler把key和value发到reducer上面，reducer输出最终结果

Optionally, before forwarding to shufflers, a 'combiner' operation in each node can be set up to perform a local per-key reduction - if specified, this would be 'step 1.5', in the above workflow.

The cluster user (programmer) only needs to supply a mapper task and a reducer task, the rest is automatically handled!



Summary: "MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key."

GFS

Since MapReduce involves accessing (reading, writing) distributed (in clusters) data in parallel, there needs to be a high-performance, distributed file system that goes along with it - Google created GFS to fill this need.

GFS abstracts details of network file access so that remote reads/writes and local reads/writes are handled (in code) identically.

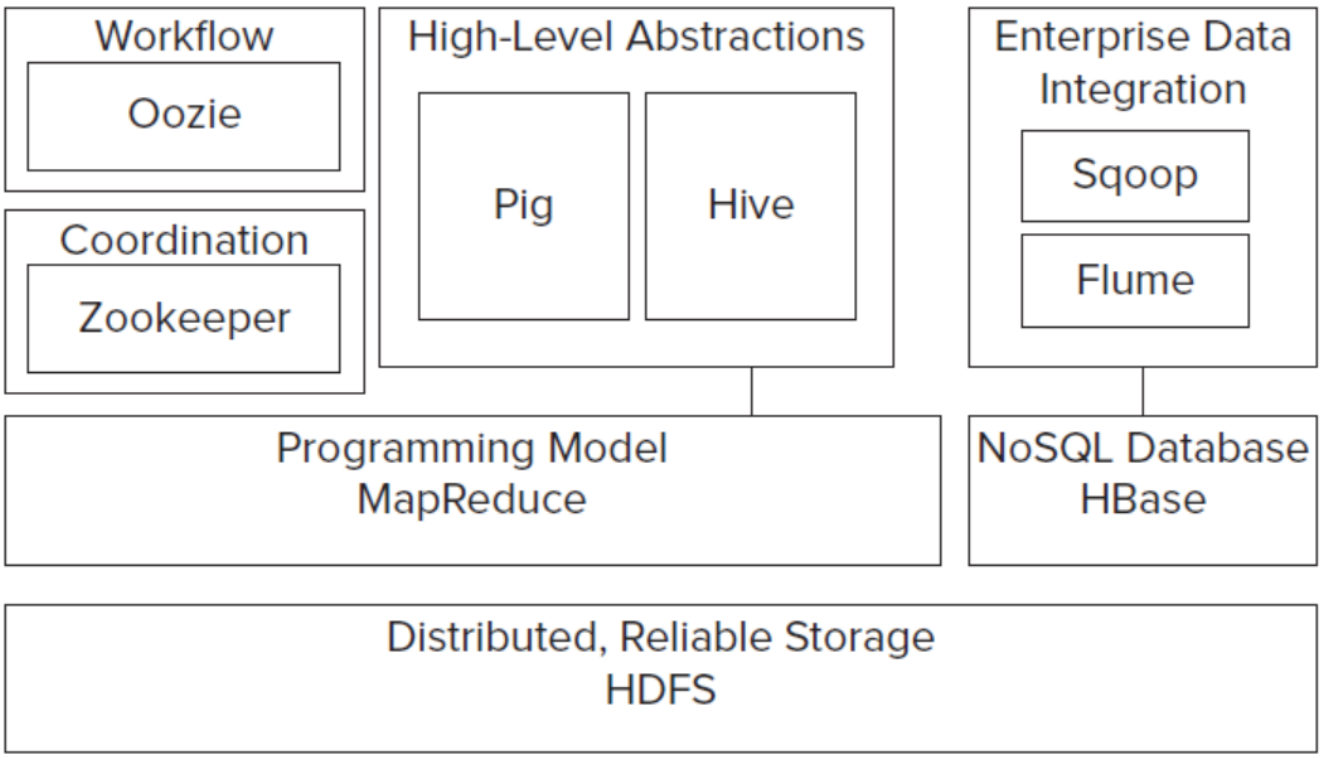
GFS differs from other distributed file systems such as (Sun's) NFS, in that the file system is implemented as a process in each machine's OS; striping is used to split each file and store the resulting chunks on several 'chunkservers', details of which are handled by a single master.

Hadoop

Hadoop is modeled after the MapReduce paradigm, and is utilized identically (by having users run mappers and reducers on (big) data).

HDFS is modeled after Google's GFS, but with some important differences

Hadoop ecosystem



The following databases are most commonly used inside a Hadoop cluster:

- MongoDB

- Cassandra
- HBase
- Hive
- Spark
- Blur
- Accumulo
- Memcached
- Solr
- Giraph

Hive

Hive provides a SQL-like scripting language called HQL. no need to create relational table schemas and populate with data.

Hive translates most queries to MapReduce jobs, thereby exploiting the scalability of Hadoop, while presenting a familiar SQL abstraction. 把大部分查询转化为MapReduce的工作，来最大化利用hadoop的扩展性

Pig

Pig provides an engine for executing data flows in parallel on Hadoop. It includes a language, Pig Latin, for expressing these data flows. Pig Latin includes operators for many of the traditional data operations (join, sort, filter, etc.), as well as the ability for users to develop their own functions for reading, processing, and writing data.

Pig Latin scripts are compiled into MR jobs that are then run on the cluster.

- 在Hadoop上并行处理数据流
- 包含语言Pig Latin，用于展示数据流。脚本会自动将内容转换为MapReduce的工作
    - 支持传统的数据操作 join，sort，filter
    - 同时用户可以自己写函数

Note that with Pig Latin, there is no explicit spec of mapping and reducing phases - the Pig=>MR compiler figures this out by analyzing the specified dataflow.

Pig Latin is a dataflow language. This means it allows users to describe how data from one or more inputs should be read, processed, and then stored to one or more outputs in parallel. These data flows can be simple linear flows like the word count example given previously. They can also be complex workflows that include points where multiple inputs are joined, and where data is split into multiple streams to be processed by different operators. To be mathematically precise, a Pig Latin script describes a directed acyclic graph (DAG), where the edges are data flows and the nodes are operators that process the data. Pig Latin 是一种数据流语言。这意味着它允许用户描述如何读取、处理来自一个或多个输入的数据，然后将其并行存储到一个或多个输出。这些数据流可以是简单的线性流，如单词 count 前面给出的例子。它们也可以是复杂的工作流，包括多个输入连接的点，以及数据被分成多个流以由不同的运算符处理的点。为了在数学上精确，Pig Latin 脚本描述了一个有向无环图 (DAG），其中边是数据流，节点是处理数据的算子。

Musketeer

Currently, front end workflows (eg. written using Hive) are *coupled* with back-end engines (such as Hadoop), making them less usable than if these could be decoupled.

Musketeer is an experimental approach to do the decoupling. Three benefits:

- Users write their workflow once, in a way they choose, but can easily execute it on alternative systems;
- Multiple sub-components of a workflow can be executed on different back-end systems; and
- Existing workflows can easily be ported to new systems.

MRv2: YARN

YARN (Yet Another Resource Negotiator) on the other hand makes non-MR applications (eg. graph processing, iterative modeling) possible (but is fully backwards compatible with v.1.0, ie. can run MapReduce jobs), and offers better scalability and cluster utilization (compared to MRv1). It also makes it possible to create (near) real-time applications.

- 支持non-MR 应用 （比如**graph processing, iterative modeling**）
- 同时支持v1.0的Mapreduce工作
- 有更好的扩展性和集群利用，因此可以实现准实时应用

Cloud infrastructure

Amazon's EC2/S3, Microsoft's Azure, Google's GCP, etc. offer a 'cloud platform' on which to build apps and services. Such cloud services offer 'elastic scaling' (of resources, ie. computing power, storage), guaranteed uptime, speedy access, etc. One hassle-free (somewhat!) way to set up a Hadoop compute cluster is to do so inside EC2 or Azure or GCP.

VM infrastructure

A virtual machine (VM) is a piece of software that runs on a host machine, to enable creating self-contained 'virtual' machines inside the host - these virtual machines can then serve as platforms on which to run programs and services.

So, another way (not cloud based) to experiment with Hadoop is to download implementations meant for virtual machines, and load them into the VMs.

Beyond MR

Spark（内存型数据处理，可以处理批式数据+流式数据）

Spark makes Big Data real-time and interactive - it is an **in-memory data processing engine** (so it is FAST), specifically meant for iterative processing of data. It is considered an alternative to MapReduce, and runs on top of HDFS.

- Better efficiency: general execution graphs, in-memory data storage. 更高的效率：通用执行图、内存数据存储。
- Query lang is SparkSQL (used to be called Shark; Shark itself was an alternative to Hive). 查询语言是 SparkSQL（以前称为 Shark；Shark 本身是 Hive 的替代品）。

- MR could not deal with complex (multi-pass) processing, interactive (ad-hoc) queries or real-time (stream) processing. Spark addresses all these. MR 无法处理复杂（multi-pass）处理、交互式（ad-hoc）查询或实时（流）处理。 Spark 解决了所有这些问题。
- A Spark application consists of a **'driver'** that converts high level queries into tasks, which **'executors'** run in parallel. Spark 应用程序由一个“驱动程序”组成，该驱动程序将高级查询转换为任务，“执行程序”并行运行。
- resilient distributed datasets (RDDs) 弹性分布式数据集
    - distributed collections of objects that can be cached in memory across cluster 可以跨集群缓存在内存中的分布式对象集合
    - manipulated through parallel operators 通过并行运算符操作
    - automatically recomputed on failure 失败时自动重新计算

Spark's modular architecture has been instrumental in enabling the following add-on functionalities:

- Spark Streaming
- Spark SQL
- Spark MLib
- Spark GraphX

Flink (并行数据处理平台，可以处理批式数据+流式数据)

Similar to MR, Flink is a parallel data processing platform.

- Apache Flink's programming model is based on concepts of the MapReduce programming model but generalizes it in several ways.
    - Flink offers Map and Reduce functions
    - additional transformations like Join, CoGroup, Filter, and Iterations. These transformations can be assembled in arbitrary data flows including multiple sources, sinks, and branching and merging flows.
    - Flink's data model is more generic than MapReduce's key-value pair model and allows to use any Java (or Scala) data types. Keys can be defined on these data types in a flexible manner.
- Consequently, Flink's programming model is a **super set** of the MapReduce programming model.
- It allows to define many programs in a much more convenient and concise way.
- it is possible to embed unmodified Hadoop functions (Input/OutputFormats, Mapper, Reducers) in Flink programs and execute them jointly with native Flink functions.

Storm （只能处理流式数据）

Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process **unbounded streams of data**, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language

Storm use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more.

Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Complementing Storm, Kafka is a distributed pub-sub real-time messaging system that provides strong durability and fault tolerance guarantees. Kafka nodes are called brokers, and are used by producers/publishers (who write data, into 'topics'), and consumers/subscribers (who read data off topics).

Storm does stream processing. A stream is a sequence of tuples. Streams are originated at spouts, that read data (from live sources, files...), and are passed on to bolts for processing (streams in, streams out).

Samza （将批数据和流式数据用相同的方式进行处理，可以处理批式数据+流式数据）

Both streams and batch data are stored in partitions, which are then read by 'tasks' which comprise the application.

Samza SQL can be used to create pipelined jobs that can utilize stream or batch data.

Spark vs Storm vs Flink vs Samza

Spark, Flink and Samza, can handle batch as well as **streaming data**.

Storm does **stream processing**.

So to summarize, we have Hadoop+Yarn for batch processing, Spark, Flink and Samza for batch+stream processing, Storm for stream processing, and Kafka for handling messages. Here is one way how it could all fit together.

BSP: MR alternative

The Bulk Synchronous Parallel (BSP) model is an alternative to MR.

A BSP computation is executed on a set of processors which are connected in a communication network but work independently by themselves. The BSP computation consists of a sequence of iterations, called supersteps. In each superstep, three actions occur:

- (i) concurrent computation performed LOCALLY by a set of processors. Each processor has its own local memory and uses local variables to independently complete its computations. This is the asynchronous part. 本地计算：由一组处理器本地执行的并发计算。 每个处理器都有自己的本地内存，并使用本地变量独立完成其计算。 这是异步部分。
- (ii) communication, during which processors send and receive messages (exchange/access data). 通信：在此期间处理器发送和接收消息（交换/访问数据）。
- (iii) synchronization which is achieved by setting a barrier - when a processor completes its part of computation and communication, it reaches this barrier and waits for the other processors to finish. 屏障同步：通过设置屏障实现的同步——当一个处理器完成其计算和通信部分时，它会到达该屏障并等待其他处理器完成。

Giraph

Giraph is an open source version of Pregel, so is Hama, Golden Orb, Stanford GPS.

Specifically designed for iterative graph computations (and nothing else!). 只能用来做iterative graph computations 迭代图运算

HAMA

Apache HAMA is a general-purpose Bulk Synchronous Parallel (BSP) computing engine on top of Hadoop. It provides a parallel processing framework for massive iterative algorithms

HAMA performs a series of supersteps based on BSP - it is suitable for iterative computation, since it is possible that input data which can be saved in memory, is able to get transfered between supersteps (unlike MR).

HAMA's vertex-centric graph computing model is suggestive of MapReduce in that users focus on a local action, processing each item independently, and the system (HAMA runtime) composes these actions to run over a large dataset.

But HAMA is not merely a graph computing engine - instead it is a general purpose BSP platform, so on top of it, any arbitrary computation (graph processing, machine learning, MRQL, matrix algorithms, network algorithms..) can be implemented; in contrast, Giraph is ONLY for graph computing.

- 通用型BSP运算引擎，基于Hadoop
- 可以用于大规模迭代算法运算
- HAMA可以在Supersteps间传递信息（不像MR）
- HAMA 的以顶点为中心的图计算模型类似于MapReduce，因为用户专注于本地动作，独立处理每个项目，系统（HAMA 运行时）组合这些动作以在大型数据集上运行。
- 但是 HAMA 不仅仅是一个图计算引擎，而是一个通用的 BSP 平台，因此在它之上，可以实现任意计算（图处理、机器学习、MRQL、矩阵算法、网络算法..）； 相**比之下，Giraph 仅用于图形计算。**

Data Mining

What is Data Mining

- the science of extracting useful information from large datasets
- heart of data mining is the process of discovering **RELATIONSHIPS** between parts of a dataset.
- Data mining is the analysis of observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner. The relationships and summaries derived through a data mining exercise are often referred to as **models** or **patterns**
- The term **'trend'** is used to describe patterns that occur or change over time

ML different from DM

Machine learning is the process of TRAINING an algorithm on an EXISTING dataset in order to have it discover **relationships** (so as to create a model/pattern/trend), and USING the result to analyze NEW data

data mining cycle

- Starting with **data**
- mining leads to **discovery**
- which leads to action (**"deployment"**)
- in turn leads to new data

Data mining typical uses

- predicting which customers will purchase what products and when
- deciding should insurance rates be set to ensure profitability
- predicting equipment failures, reducing unnecessary maintenance and increasing uptime to optimize asset performance
- anticipating resource demands
- predicting which customers are likely to leave and what can be done to retain them
- detecting fraud
- minimizing financial risk
- increasing response rates for marketing campaigns

Data mining algorithms

Categories

Practically all data mining ('DM') algorithms neatly fit into one of these 4 categories:

- **Classification 分类**: involves LABELING data
- **Clustering 聚类** : involves GROUPING data, based on similarity
- **Association 关联**: involves RELATING data
- **Regression 回归**: involves COUPLING data [incl finding 'outliers']
- Also can be divided into '**supervised**' learning method (where we need to provide categories for, ie. train, using known outcomes), or an '**unsupervised**' method (where we provide just the data to the algorithm, leaving it to learn on its own), or '**semi-supervised**', or even '**self-supervised**'.

Decision trees (Classification and regression trees)

are machine-learning methods for constructing prediction models from data. The models are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition.

Work like this

- user provides a set of input (training) data, which consists of features (independent parameters) for each piece of data, AND an outcome (a 'label', ie. a class name) 用户提供一组输入（训练）数据，其中包含每条数据的特征（独立参数）和结果（“标签”，即类名）
- the algorithm uses the data to build a 'decision tree' [with feature-based conditionals (eqvt to 'if' or 'case' statements) at each non-leaf node], leading to the outcomes (known labels) at the terminals 该算法使用数据构建“决策树”[在每个非叶节点处使用基于特征的条件（eqvt 到 'if' 或 'case' 语句）]，从而在终端产生结果（已知标签）
- the user makes use of the tree by providing it new data (just the feature values) - the algorithm uses the tree to 'classify' the new item into one of the known outcomes (classes) 用户通过提供新数据（只是特征值）来使用树 - 算法使用树将新项目“分类”为已知结果之一（类）

If the outcome (dependent, or 'target' or 'response' variable) consists of classes (ie. it is 'categorical'), the tree we build is called a **classification tree**. On the other hand if the target variable is continuous (a numerical quantity), we build a **regression tree**.

nominal data results in classification trees, and ordinal data results in regression trees.

Support Vector Machine (SVM)

An SVM always partitions data (classifies) them into TWO sets - uses a 'slicing' hyperplane (multi-dimensional equivalent of a line), instead of a decision tree. The hyperplane maximizes the gap on either side (between itself and features on either side). This is to minimize chances of mis-classifying new data.

On either side, the equidistant data points closest to the hyperplane are the 'support vectors'. Special case - if there is a single support (closest point) on either side, in 2D, the separator is the perpendicular bisector of the line segment joining the supports; if not, the separating line/plane/hyperplane needs to be calculated by finding two parallel hyperplanes with no data in between them, and maximizing their gap. The goal is to achieve "margin maximization".

How do we know that a support data point is indeed a support vector? If we move the data point and therefore the boundary moves, that is a support vector 😀

kNN (k Nearest Neighbors)

the new point's type will be the type of the majority of its 'k' neighbors

Also, to eliminate excessive influence of large numerical values, we usually normalize our data so that all values are 0..1.

kNN is a 'lazy learner' - just stores input data, uses it only when classifying an unlabeled (new) input.

Naive Bayes

probability-based, supervised, classifier algorithm

Assumption: each of the features are statistically independent of each other

For each training feature set, probabilities are assigned for each possible outcome (class). Given a new feature, the algorithm outputs a classification corresponding to the max of the most probable value of each class (which the algorithm calculates, using the 'maximum a posteriori', or 'MAP' decision rule).

k-means clustering

creates 'k' number of "clusters" from the input data, using some measure of closeness (items in a cluster are closer to each other than any other item in any other cluster).

This is an example of an unsupervised algorithm

Approach: start with 'n' random locations ('centroids', ie means) in the dataset; assign each input point (our data) to the [current] closest centroid; compute new centroids (from our data, for each centroid's "membership"); iterate (till convergence is reached) - this is the 'mean shift' algorithm.

Hierarchical clustering

n some situations (where it is meaningful to do so), it is helpful to separate items in a dataset into **hierarchical** clusters (clusters of clusters of..). There are two ways to look at this - as the merging of smaller clusters into bigger superclusters, or dividing of larger clusters into finer scale ones.

How do we decide what to merge? A popular strategy - pick clusters that lead to the smallest increase in sum of squared distances (in the above diagram, that is what the vertical bar lengths signify).

EM (Expectation Maximization)

EM is frequently used to 'auto cluster' data.

We do not know the values for the model parameters, or latent/missing variables! We DO have observed/collected/measured data, which the model should be able to act on, ie we have 'outcomes'. **The question is, what model parameters would explain (result in, produce...) the outcomes?** In other words, we want to explain why/how those outcomes result.

The algorithm works as follows:

- pre-step: start with random (!) values for the model parameters
- step1: use current param values to compute probabilities for all possible values for each hidden (latent) var, then do a weighted average (weighted by probability) to compute the best value for each latent var (this is the 'E' step)
- step2: use the hidden vars' values found in the above step, to improve/update the model parameters ('M' step) - do this by maximizing likehood [for the outcomes, given the params]
- iterate the above two steps till param values converge

Here is another explanation, from a StackOverflow post:

There's a chicken-and-egg problem in that to solve for your model parameters you need to know the distribution of your unobserved data; but the distribution of your unobserved data is a function of your model parameters.

E-M tries to get around this by iteratively guessing a distribution for the unobserved data, then estimating the model parameters by maximizing something that is a lower bound on the actual likelihood function, and repeating until convergence:

- start with guess for values of your model parameters
- E-step: For each datapoint that has missing values, use your model equation to solve for the distribution of the missing data given your current guess of the model parameters and given the observed data (note that you are solving for a distribution for each missing value, not for the expected value). Now that we have a distribution for each missing value, we can calculate the expectation of the likelihood function with respect to the unobserved variables. If our guess for the model parameter was correct, this expected likelihood will be the actual likelihood of our observed data; if the parameters were not correct, it will just be a lower bound.
- M-step: Now that we've got an expected likelihood function with no unobserved variables in it, maximize the function as you would in the fully observed case, to get a new estimate of your model parameters.
- repeat until convergence.

Classif vs clustering algorithm[s]: a clarification

Classification algorithms (when used as learning algorithms) have one ultimate purpose: given a piece of new data, to place it into one of several pre-exising, LABELED "buckets" - these labels could be just names/nonimal (eg. ShortPerson, Yes, OakTree..) or value ranges/ordinal (eg. 2.5-3.9, 100,000-250,000)..

Clustering algorithms on the other hand (again, when used as learning algorithms) take a new piece of data, and place it into a pre-existing group - these groups are UN-LABELED, ie. don't have names or ranges.

Also, in parameter (feature/attribute) space, each cluster would be distinct from all other clusters, by definition; with classification, just 'gaps' don't need to exist.

Note that we use multiple terms to denote data. If we imagine data to be tabular, each row would constitute one sample, with each column being refered to as an attribute/parameter/feature/input/independent variable/descriptor/dimension, and the label column (if present) being refered to as a label/class/target/output/dependent variable/response variable/dimension.

Each sample (row) would constitute a single point in the multidimensional space/axes/coordinate system created by the columns, including the label column (if present), and the collection of rows/samples would lead to a distribution - data mining consists of finding patterns in such a multi-dimensional point distribution.

A priori

Looking for hidden relationships in large datasets is known as association analysis or association rule learning.

The A priori algorithm comes up with association rules (relationships between existing data, as mentioned above).

What is specified to the algorithm as input, is a **[support (ie. frequency),confidence (ie. certainty, ie. accuracy)] pair** - given these, the algorithm outputs all matching associations that satisfy the [support,confidence] criteria. We would then make appropriate use of the association results (eg. co-locate associated items in a store, put up related ads in the search page, print out discount coupons for associated products while the customer is paying their bill nearby, etc.).

Linear regression

This (mining) technique is straight out of statistics - given a set of training pairs for a feature x and outcome y, fit the best line describing the relationship between x,y. The line describes the relationship/pattern.

Non-linear regression

Here we fit a higher order polynomial equation (parabola,) to the observed data:

Two parameter, non-linear regression

Here we need to fit a higher order, non-linear surface (ie. non-planar) to the observed data:

Non-parametric modeling

no assumptions on what our surface (the dependent variable) would look like; we would need much more data to do the surface fitting, but we don't need the surface to be parameter-based!

While non-parametric models might be better suited to certain data distributions, they could lead to a poor estimate as well (if there is over-fit)..

Logistic regression

Logistic regression is a classification (usually binary) algorithm:

- compute regression coeffs (linear) corresponding to a 'decision boundary' (line dividing two classes of training data)
- use the derived regression coeffs to compute outcome for new data
- transform the outcome to a logistic regression value, and use the 0..1 result to predict a binary outcome (class A or class B)

Result (the whole point of doing the three steps above) - we are transforming a continuous, regression-derived value (which can be arbitrarily large or small) into a 0..1 value, which in turn we transform into a binary class. we'd need to transform our regression results to a 0-centered distribution before using the logistic equation - this is because 1/(1+exp(-x)) is 0.5, when x=0.

Ensemble Learning

What if we used a training dataset to train several different algorithms - eg. decision tree, kNN, neural net(s)..? You'll most likely get (slightly!) different results for target prediction.

We could use a voting scheme, and use the result as the overall output. Eg. for a yes/no classification, we'd return a 'yes' ('no') if we got a 'yes' ('no') majority.

This method of combining learners' results is called 'boosting', and resulting combo learner is called an 'ensemble learner'. Why do this? "Wisdom of the crowds" 😀 We do this to minimize/eliminate variances between the learners.

RandomForest (TM)

RandomForest(TM) is an *ensemble* method where we:

- grow a 'forest' (eg. with count=500) decision trees, run our new feature through all of them, then use a voting or averaging scheme to derive an ensemble classification result
- keep each tree small - use sqrt(k) features for it, chosen randomly from the overall 'k' samples

Machine learning

AI types

Type I: Reactive machines - make optimal moves - no memory, no past 'experience'. Ex: game trees.

Type II: Limited memory - human-compiled/provided , one-shot 'past' 'experiences' are stored for lookup. Ex: expert systems, neural networks.

Type III: Theory of Mind - "the understanding that people, creatures and objects in the world can have thoughts and emotions that affect the AI programs' own behavior".

Type IV: Self-awareness - machines that have consciousness, that can form representations about themselves (and others).

Type I AI is simply, application of rules/logic (eg. chess-playing machines).

Type II AI is where we are, today - specifically, this is what we call 'machine learning' - it is **"data-driven AI"**! Within the last decade or so, spectacular progress has been made in this area, ending what was called

the 'AI Winter'.

As of now, types III and IV are in the realm of speculation and science-fiction, but in the general public's mind, they appear to be certainty in the near term 😃

AI types have been pursued

- inference-based: 'symbolic'
- goals/rewards-based: 'reinforcement'
- connection-based: 'neuro'

ML types

ML is the ONE subset of AI that is revolutionizing the world.

Machine learning focuses on the construction and study of systems that can learn from data to optimize a performance function, such as optimizing the expected reward or minimizing loss functions. The goal is to develop deep insights from data assets faster, extract knowledge from data with greater precision, improve the bottom line and reduce risk.

Supervised learning

Supervised learning algorithms are "trained" using examples (**DATA!**] where in addition to features [inputs], the desired output [label, aka target] is known. The goal is to LEARN the patterns inherent in the training dataset, and use the knowledge to PREDICT the labels for new data.

Unsupervised learning

Unsupervised learning is a type of machine learning where the system operates on unlabeled examples. In this case, the system is not told the "right answer." The algorithm tries to find a hidden structure or manifold in unlabeled data. The goal of unsupervised learning is to explore the data to find intrinsic structures within it using methods like clustering or dimension reduction.

For Euclidian space data: k-means clustering, Gaussian mixtures and principal component analysis (PCA)

For non-Euclidian space data: ISOMAP, local linear embedding (LLE), Laplacian eigenmaps, kernel PCA.

Use matrix factorization, topic models/graphs for social media data.

Semisupervised learning

Semisupervised learning is used for the same applications as supervised learning. But this technique uses both labeled and unlabeled data for training - typically, a small amount of labeled data with a large amount of unlabeled data. The primary goal is unsupervised learning (clustering, for example), and labels are viewed as side information (cluster indicators in the case of clustering) to help the algorithm find the right intrinsic data structure.

Reinforcement Learning

With **reinforcement learning** 'RL'), the algorithm discovers for itself which actions yield the greatest rewards through trial and error. Reinforcement learning has three primary components:

- agent - the learner or decision maker
- environment - everything the agent interacts with
- actions - what the agent can do

The objective is for the agent to choose actions that maximize the expected reward over a given period of time. The agent will reach the goal much quicker by following a good policy, so the goal in reinforcement learning is to learn the best policy. Reinforcement learning is often used for robotics and navigation.

Markov decision processes (MDPs) are popular models used in reinforcement learning. MDPs assume the state of the environment is perfectly observed by the agent. When this is not the case, we can use a more general model called partially observable MDPs (or POMDPs).

The core ideas [backprop, nonlinear activation, composition]: DNN

Neuro/connectionist/data-driven AI, works like so:

- data is gathered and cleaned; if necessary, each datum is assigned a **LABEL**; or the data used as-is
- the data+labels are used to TRAIN an algorithm (NN) iteratively - to reduce mislabeling 'loss'; more training is done if necessary
- the trained model is DEPLOYED to production - it can now label NEW data on its own [the 'machine' has 'learned' (the PATTERNS in the DATA)!]

A neural network is a form of 'AI' - uses neuron-like connected units to **learn patterns** in training (existing) data that has known outcomes, and uses the learning to be able to gracefully respond to new (non-training, 'live') data.

Definition: a neural net(work) is **an interconnected set of weighted, nonlinear functions** [this compact definition will become clear:

The overall idea is this:

- existing data is used to TRAIN a neural network - the network 'learns' patterns in the data, by adapting weights in each interconnected unit ('neuron')
- the network can now go 'live', ie. be deployed
- new data can be processed on the DEPLOYED network, which would make predictions about it based on the patterns learned

Neural networks (NNs) can be used to:

- recognize/classify features - traffic, terrorists, expressions, plants, words..
- detect anomalies - unusual CC activity, unusual machine states, gene sequences, brain waves..
- predict exchange rates, 'likes'..
- calculate numerical values (eg. home prices)
- ... [HUNDREDS, if not THOUSANDS, of uses - ANY form of data, that has ANY pattern in it, can be learned!!]

The brain (specifically, learning/training) is modeled after strengthening relevant neuron connections - neurons communicate (through axons and dendrites) dataflow-style (neurons send output signals to other

neurons):

- Each neuron receives inputs from other neurons
- The effect of each input line on the neuron is controlled by a synaptic weight
- The synaptic weights adapt so that the whole network learns to perform useful computations
- You have about 10^11 neurons each with about 10^4 weights

The functions we use to generate the output, are called activation functions - the ones we looked at are identity, binary threshold, rectifier and sigmoid. The gradients of these functions are used during backprop. There are more (look these up later) - symmetrical sigmoid, ie. hyperbolic tangent (tanh), soft rectifier, polynomial kernels...

* we create LAYER upon LAYER of neurons - each layer is a set (eg. column) of neurons, which feed their (stochastic) outputs downstream, to neurons in the next (eg. column to the right) layer, and so on

* each layer is responsible for 'learning' some aspect of our target - usually the layers operate in a hierarchical (eg. raw pixels to curves to regions to shapes to FEATURES) fashion

* a layer 'learns' like so: its input weights are adjusted (modified iteratively) so that the weights make the neurons fire when they are given only 'good' inputs.

Learning (ie. iterative weights modification/adjustment) works via 'backpropagation', with iterative weight adjustments starting from the last hidden layer (closest to the output layer) to the first hidden layer (closest to the input layer). Backpropagation aims to reduce the ERROR between the expected and the actual output [by finding the minimum of the [quadratic] loss function], for a given training input. Two hyper/meta parameters guide convergence: learning rate [scale factor for the error], momentum [scale factor for error from the previous step].

NN-based learning has started to REVOLUTIONIZE AI, thanks to three advances:

- Big Data (BILLIONS of images, tens of thousands of hours of video/audio, terabytes of text, billions of tweets..), to use for training: more training predictably leads to better learning
- better algorithms - fruits of decades' worth of academic research in ML; more recently, 'industry' (Google, Facebook, Microsoft, IBM) seems to be taking the lead
- faster cloud computing platforms and libraries

What we do in (supervised) ML is IDENTICAL to what we do in BI, DM!

It's ALL about **calculating quantities derived from patterns in existing data**.

How can we calculate f()?

- using LOTS of data
- by iteration - via 'hyper params' such as architecture, learning rate, momentum; by optimizing our solving; by computing and using error between computed and expected outputs
- using nonlinearity
- using LOTS of small, simple 'neuron' subfunctions (out of which our f() is COMPOSED) [connected using specific *architectures*]

'Deep Learning' is starting to yield spectacular results, to what were once considered intractable problems..

Q: so what makes it 'deep'? A: the number of intermediate layers of neurons.

GPUs and other forms of hardware are used to accelerate deep learning - advantages: massively parallel processing, and possibility of arbitrary speed increases over time just by upgrading hardware!

GPUs (multi-core, high-performance graphics chips made by NVIDIA etc.) and DNNs seem to be a match made in heaven!

RNN, LSTM, Transformers

An RNN is a history-dependent network where past predictions are used for future ones (by having outputs fed back)

An LSTM is a special kind of RNN, for being able to process longer chains of dependencies.

RNNs/LSTMs are especially good for 'sequence' problems such as speech recognition, language translation, etc.; they are not massively parallelizable the way CNNs can be.

Temporal Convolution Nets (TCNs) are a good, parallelizable alt to RNNs; Numenta's HTM - also a better alternative to RNNs.

CNN [Convolutional Neural Network]

In signal processing, a convolution is a blending (or integrating) operation between two functions (or signals or numerical arrays) - one function is convolved (pointwise-multiplied) with another, and the results summed.

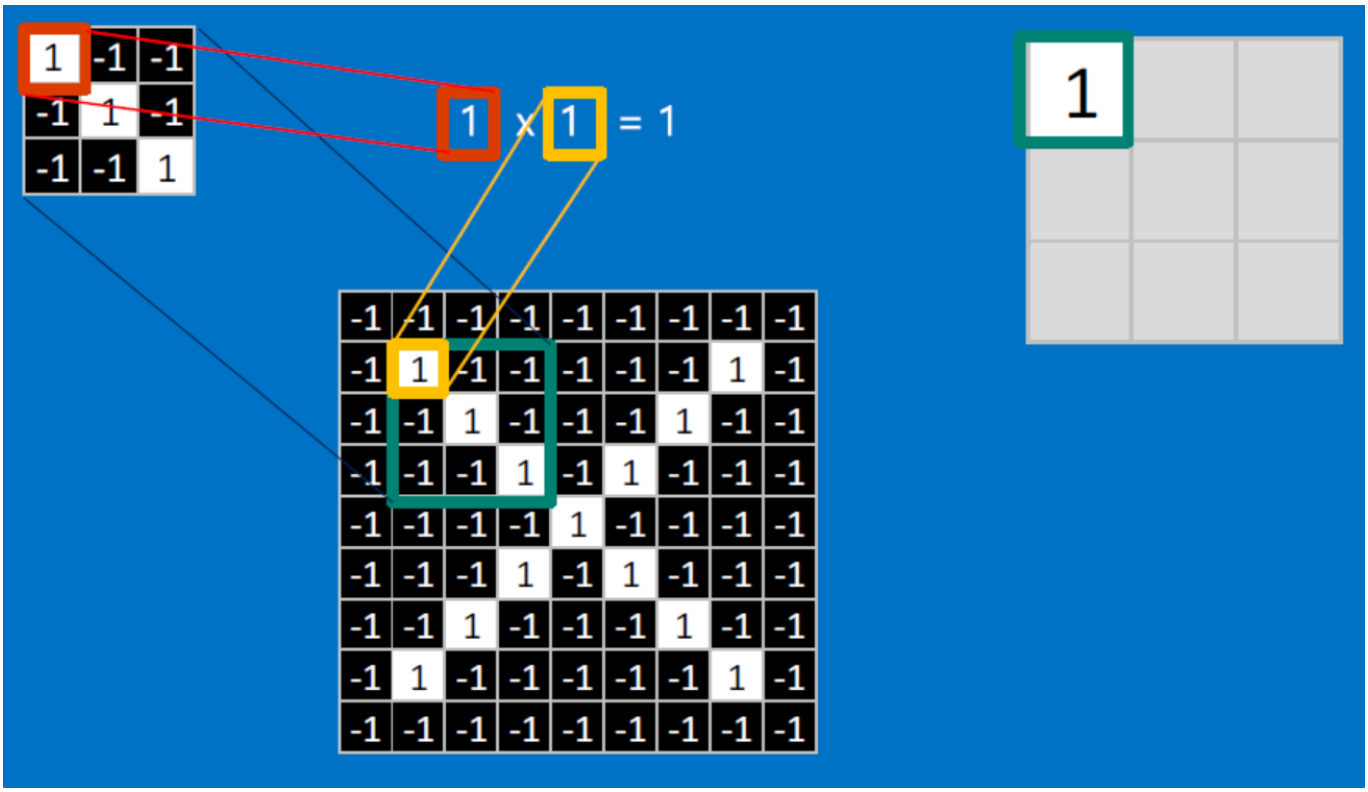
Convolution is used heavily in creating image-processing filters for blurring, sharpening, edge-detection, etc. The to-be-processed image represents the convolved function, and a 'sliding' "mask" (grid of weights), the convolving function (aka convolution kernel):

CNNs are biologically inspired - (convo) filters are used across a whole layer, to enable the entire layer as a whole to detect a feature. Detection regions are overlapped, like with cells in the eye.

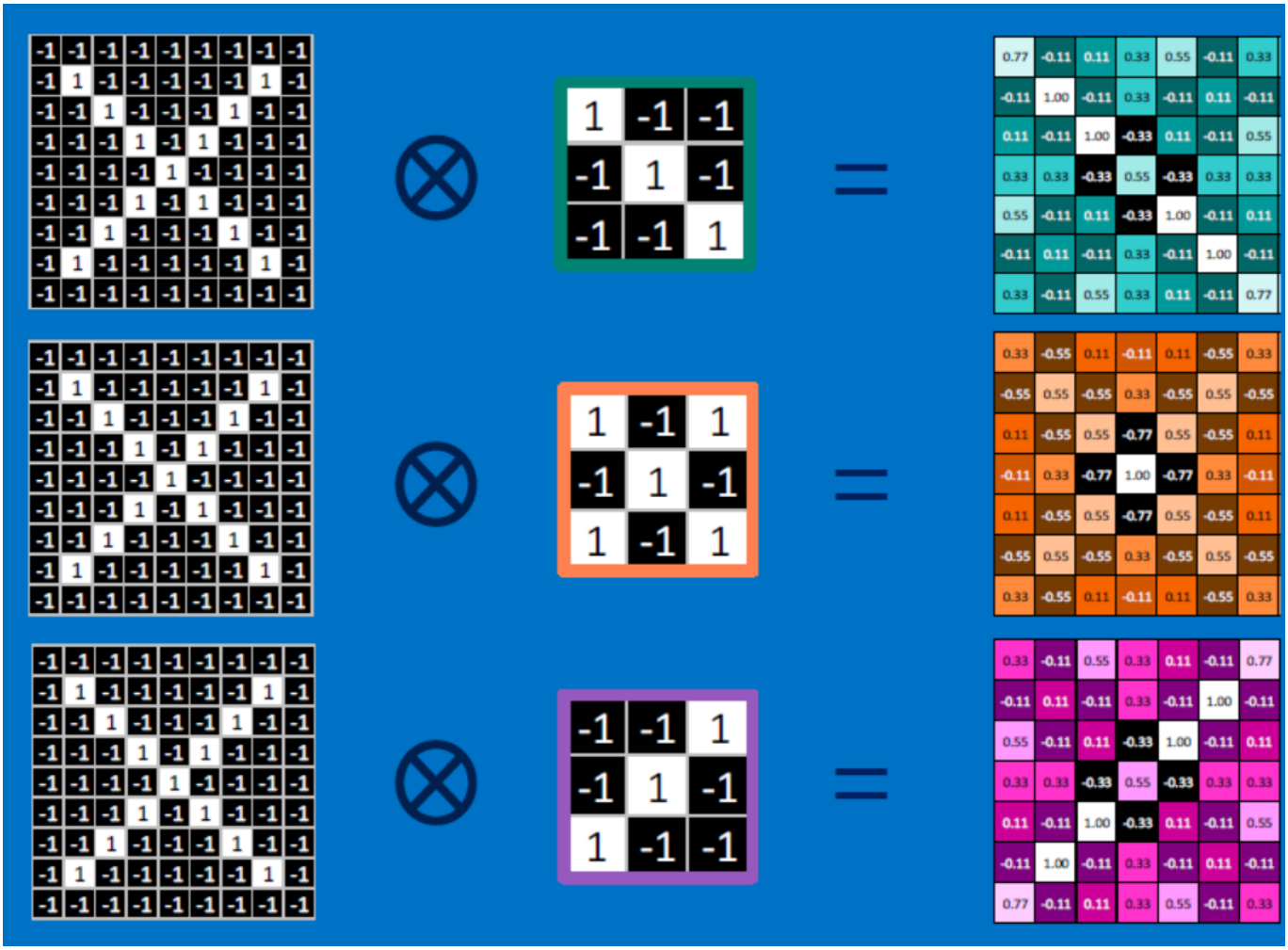
In essence, a CNN is where we represent a neuron's weights as a matrix (kernel), and slide it (IP-style) over an input (an image, a piece of speech, text, etc.) to produce a convolved output.

EACH NEURON IS CONVOLVED OVER THE ENTIRE INPUT (again, IP-style), AND AN OUTPUT IS GENERATED FROM ALL THE CONVOLUTIONS. The output gets 'normalized' (eg. clamped), and 'collapsed' (reduced in size, aka 'pooling'), and the process repeats down several layers of neurons: input -> convolve -> normalize -> reduce/pool -> convolve -> normalize -> reduce/pool -> ... -> output.

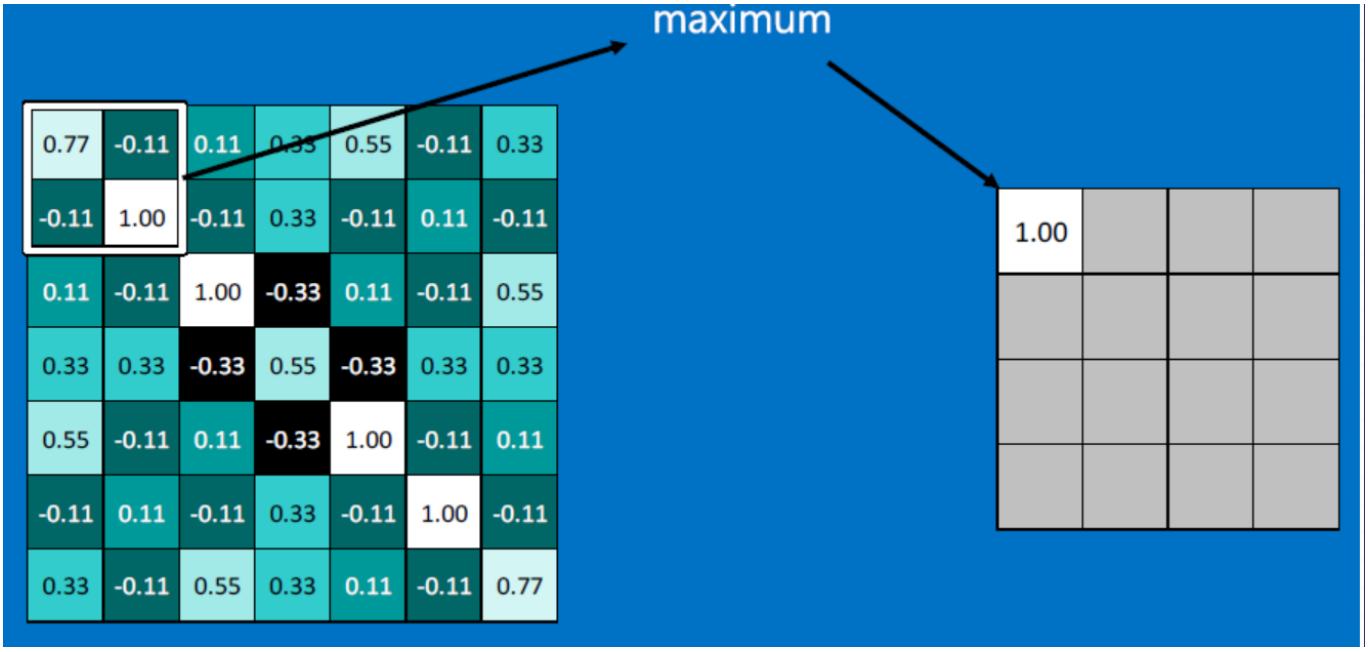
- CONVOLVE, ie. do $x_i \cdot w_i$, then average, output a value:



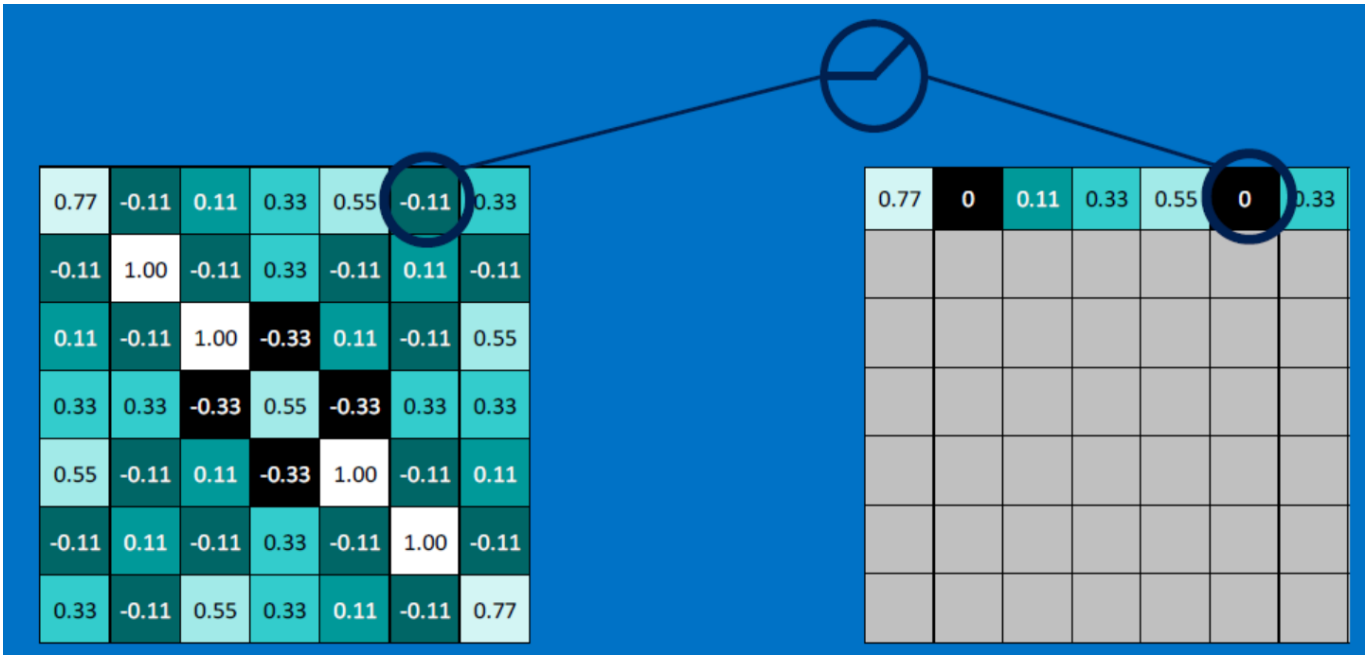
- Each neuron (feature detector) produces an output - so a single input image produces a STACK of output images



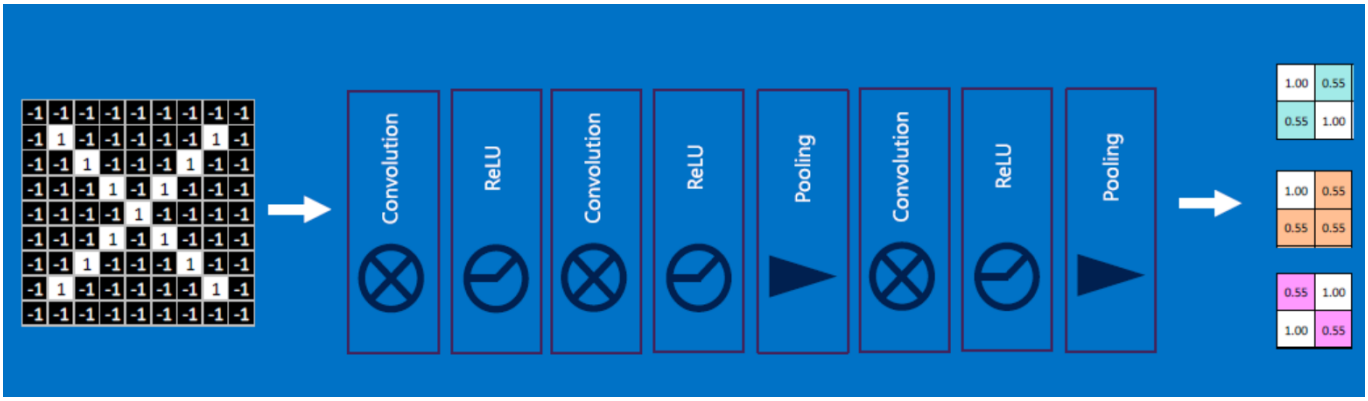
- To collapse the outputs, we do 'max pooling' - replace an mxn (eg. 2x2) neighborhood of pixels with a single value, the max of all the m*n pixels.



- Next, create a ReLU - rectified linear unit - replace negative values with 0s:



- Usually there are multiple stages:(convolution, pooling, ReLU)



When is a CNN **not** a good choice? Answer: when data is not spatially laid out, ie. scrambling rows and columns of the data would still keep the data intact (like in a relational table) but would totally throw off the convolutional neurons!

GANs! And encoder-decoder pairs...

Adversarial learning methods

生成对抗网络（英语：Generative Adversarial Network，简称GAN）是非监督式学习的一种方法，通过让两个神经网路相互博弈的方式进行学习。该方法由伊恩·古德费洛等人于2014年提出。 [1] 生成对抗网络由一个生成网络与一个判别网络组成。生成网络从潜在空间（latent space）中随机取样作为输入，其输出结果需要尽量模仿训练集中的真实样本。判别网络的输入则为真实样本或生成网络的输出，其目的是将生成网络的输出从真实样本中尽可能分辨出来。而生成网络则要尽可能地欺骗判别网络。两个网络相互对抗、不断调整参数，最终目的是使判别网络无法判断生成网络的输出结果是否真实

EBMs: As an alternative to GANs, a similar idea, called an Encoder-Decoder pair, can ALSO generate data (faces, words, music...). The encoder, specifically a 'VAE' learns to create a representation, a 'data generating distribution', of its input data, using latent-space features [of the input data]. Roughly, it learns to map an input datum into a point in multi-dim latent space. REVERSING this, **ANY random point in the latent feature space can be used to GENERATE (via a decoder) a NEW datum!

Architecture pruning

By eliminating 'weak' (small weights) connections (or entire neurons), we can retain overall accuracy, and dramatically improve performance (esp on edge devices).

Problems...

Because it's ALL based on DATA, issues arise:

- bias
- deepfakes
- easy foolability
- lack of explainability
- unchecked power

And there are MUCH, MUCH, M-U-C-H bigger problems 😦

- Rod Brooks
- Melanie Mitchell
- Gary Marcus
- ...

Current work (research directions)

Here is state-of-the-art...

- the Transformer architecture (Google, 2017) is a 'game changer' for language processing - it STACKS encoders and decoders, providing longer 'attention' spans. This has given rise to HUGE pre-trained language models: GPT-3, GPT-4, M'soft+NVIDIA, Wu Dao 2.0...

- optics-based NN
- another approach to AI is to model the brain's structure, in software or in hardware. IBM has its SyNAPSE chip, and TrueNorth NN chip. Numenta is another player in neuromorphic computing. Another approach to neuromorphic chips is to incorporate some wetware into them.
- Vision Transformers! (ViT)
- Neuro-symbolic integration, DeepRL... [combos]
- Geoff Hinton: GLOM
- GANsformers
- synthesizing images from text descriptions, eg. DALL-E and CLIP
- GNN - express data as a graph, learn the graph's structure, then predict properties given a new graph
- GDL - learn real-world shapes (topology)
- hmmm: https://spectrum.ieee.org/special-reports/the-great-ai-reckoning and https://bdtechtalks.com/2021/05/03/artificial-intelligence-fallacies/
- contrastive learning, eg. CLIP (these are behind the magic of 'AI art generators' - a form of "multimodal" learning
- LLMs!!!!! Including HFRL, OPL (?!)...

DL/ML tools

APIs/frameworks

Part 1

These are the most heavily used:

- TensorFlow ('TF')
- Spark MLlib: https://spark.apache.org/mllib/ and https://spark.apache.org/docs/2.2.0/ml-pipeline.html
- Keras: https://keras.io/ [a higher level lib, compared to TF etc]; here are all the types of Keras layers
- Torch, PyTorch: https://pytorch.org, http://torch.ch/
- scikit-learn: https://scikit-learn.org/stable/
- Caffe: https://caffe2.ai/, http://caffe.berkeleyvision.org/ [→ Caffe2 → PyTorch]
- Apache mxnet: https://mxnet.apache.org/ [multi-language APIs, GPU and cloud support...]
- CNTK: https://docs.microsoft.com/en-us/cognitive-toolkit/

TL;DR: simply learn Keras or PyTorch, and if necessary, TF.

Part 2

Upcoming/lesser-used/'internal'/specific:

- here is FBLearner Flow - Facebook's version of TensorFlow 😃
- Apache Mahout - a collection of ML algorithms, in Java/Scala
- .NET ML: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet
- fastai [on top of PyTorch]: https://github.com/fastai/fastai
- OpenVINO: https://software.intel.com/en-us/openvino-toolkit and https://www.youtube.com/watch?v=rUwayTZKnmA&t=1s [a tutorial]
- Turi: an alternative to Apple's CreateML: https://github.com/apple/turicreate

- LibSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvm/
- LightGBM: https://github.com/Microsoft/LightGBM
- XGBoost: https://xgboost.ai/ [and, look at Tianqi's slides and talk]
- CatBoost: https://tech.yandex.com/catboost/
- Google - SEED: https://ai.googleblog.com/2020/03/massively-scaling-reinforcement.html
- Uber's 'Fiber', for distributed ML training: https://venturebeat.com/2020/03/26/uber-details-fiber-a-framework-for-distributed-ai-model-training/
- LOTS of smaller efforts: https://github.com/EthicalML/awesome-production-machine-learning

Cloud

The virtually unlimited computing power and storage that a cloud offers, make it an ideal platform for data-heavy and computation-heavy applications such as ML.

Amazon: https://aws.amazon.com/machine-learning/ Their latest offerings make it possible to 'plug in' data analysis anywhere.

Google: https://cloud.google.com/products/ai/ [in addition, Colab is an awesome resource!]

Microsoft: https://azure.microsoft.com/en-us/services/machine-learning-studio/ [and AutoML] [aside: alternatives to brute-force 'auto ML' include 'Neural Architecture Search' [incl. this], pruning, and better network design (eg using ODEs - see this).

IBM Cloud, Watson: https://www.ibm.com/cloud/ai [eg. look at https://www.ibm.com/cloud/watson-language-translator]

Others:

- h2o: https://www.h2o.ai/products/h2o/ [supports R, Python, Java, Scala, JSON, native Flow GUI [similar to Jupyter], REST...]
- BigML: https://bigml.com/features#platform
- FloydHub: https://www.floydhub.com/
- Paperspace: https://ml-showcase.paperspace.com/
- Algorithmia, eg. https://info.algorithmia.com/ and https://demos.algorithmia.com/

Pretrained ML models

A pre-trained model includes an architecture, and weights obtained by training the architecture on specific data (eg. flowers, typical objects in a room, etc) - ready to be deployed.

TinyMOT: https://venturebeat.com/2020/04/08/researchers-open-source-state-of-the-art-object-tracking-ai

Apple's CreateML is useful for creating a pre-trained model, which can then be deployed (eg. as an iPad app) using the companion CoreML product. NNEF and ONNX are other formats, for NN interchange.

Pre-trained models in language processing, include Transformer-based BERT and GPT-2. Try this demo (of GPT etc). There is GPT-3 currently available, GPT-4 in the works, Wu Dao 2.0, MT-NLG...

There are also, combined (bimodal) models, based on language+image data.

Tools

Several end-to-end applications exist, for DM/ML. Here popular ones.

Weka is a Java-based collection of machine learning algorithms.

RapidMiner uses a dataflow ("blocks wiring") approach for building ML pipelines.

KNIME is another dataflow-based application.

TIBCO's 'Data Science' software is a similar (to WEKA etc) platform. Statistica [similar to Mathematica] is a flexible, powerful analytics software [with an old-fashioned UI].

bonsai is a newer platform.

To do ML at scale, a job scheduler such as from cnvrg.io can help.

SynapseML is a new ML library from Microsoft.

There are a variety of DATAFLOW ('connect the boxes') tools! This category is likely to become HUGE:

- Perceptilabs: https://www.perceptilabs.com/
- Lobe: https://insights.dice.com/2018/05/07/lobe-deep-learning-platform/
- https://www.producthunt.com/posts/datature
- smartpredict: https://smartpredict.ai/
- StackML: https://stackml.com/ [RIP]
- Baseet: https://baseet.ai/ [RIP]

Languages

These languages are popular, for building ML applications (the APIs we saw earlier, are good examples):

- Python
- R
- Julia [Python 'replacement'?!]
- Wolfram
- JavaScript - this is a good list of JS-based libraries [look at ConvnetJS for nice demos]
- Scala - a functional+OO language - here is a roundup of libraries [these are in addition to Spark's MLlib Scala API]
- Java - another robust language for building ML libs [we already saw WEKA] and apps
- Jupyter [an environment, not a language] (eg. here is a collection of ML notebooks - as an exercise, run them all in Colab!) [also, here are notebooks for 'everything'!]

Hardware

Because (supervised) ML is computationally intensive, and detection/inference needs to happen in real-time almost always, it makes sense to accelerate the calculations using hardware. Following are examples.

Google TPU: TF is in hardware! Google uses a specialized chip called a 'TPU', and documents TPUs' improved performance compared to GPUs. Here is a pop-sci writeup, and a Google blog post on it.

Amazon Inferentia: a chip, for accelerating inference (detection): https://aws.amazon.com/machine-learning/inferentia/

NVIDIA DGX-1: an 'ML supercomputer': https://www.nvidia.com/en-us/data-center/dgx-1/ [here is another writeup]

Intel's Movidius (VPU): https://www.movidius.com/ - on-device computer vision

In addition to chips and machines, there are also boards and devices:

- Pixy2: https://pixycam.com/ - camera + ML in a single board
- Coral: https://coral.withgoogle.com/
- Jetson Nano: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/
- Movidius NCS: https://software.intel.com/en-us/movidius-ncs
- ...

Overall, there's an explosion/resurgence in 'chip design', for accelerating AI training, inference. In April '21, NVIDIA announced its new A30 and A10 GPUs, at the annual [GTC] conference.

Data Visualization

What is visualization, and why

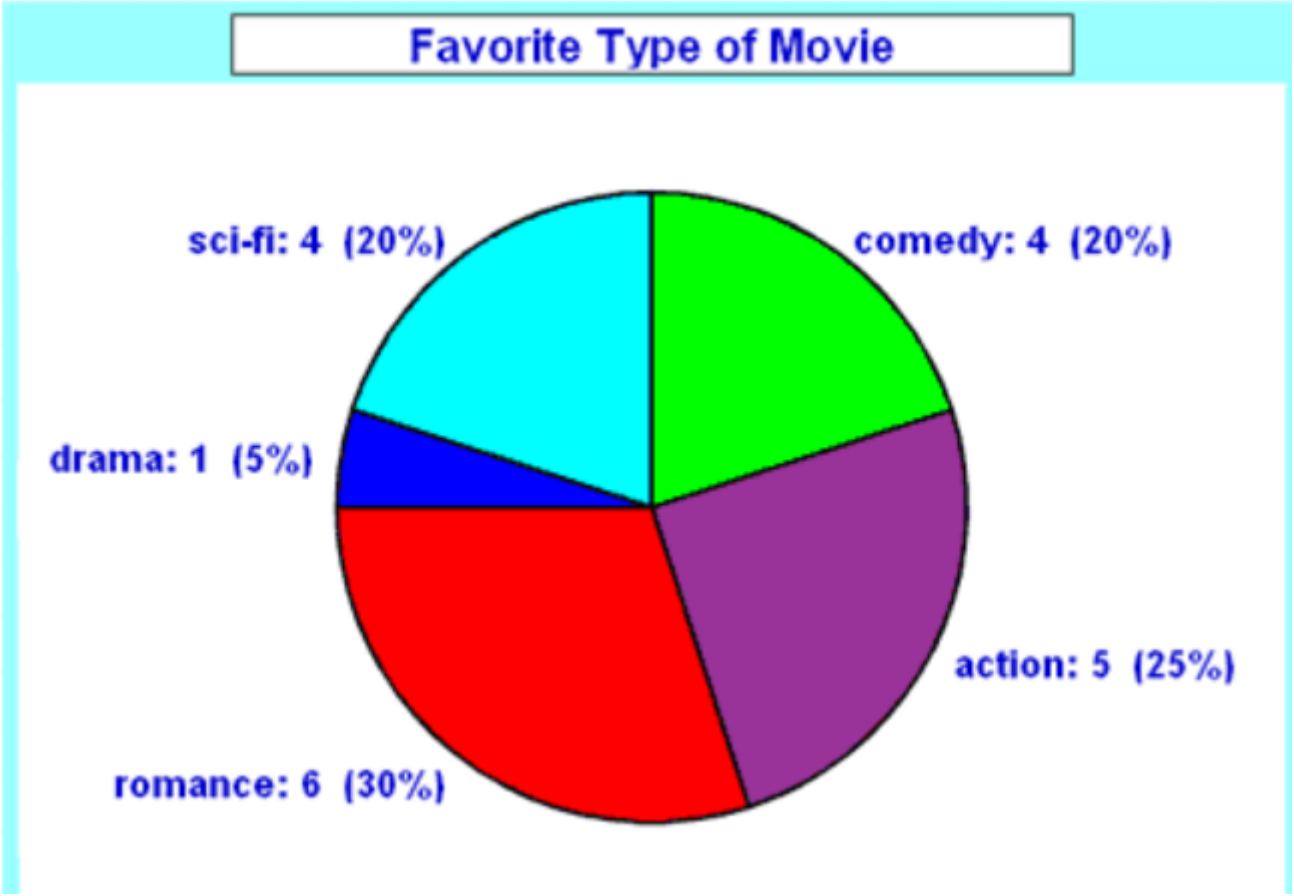
Data visualization ("data viz") involves (the study of) tools and techniques for **turning data into images/graphics** - to obtain BETTER INSIGHT into the data.

In other words, this is about **graphical depictions of data**. **Why do it?** To understand, communicate, act/decide/utilize.

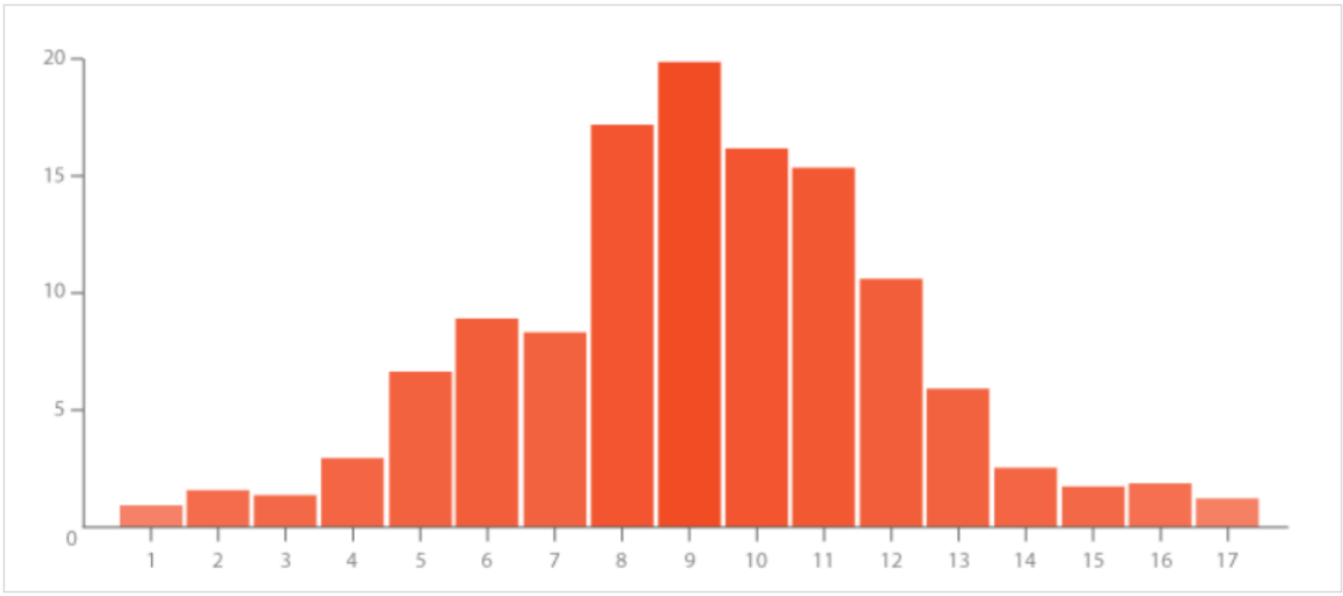
In what follows, we are going to look at what can be visualized, and how. Note: it's not all 'Big Data' viz, it's not all 'mined' results either.

A single variable

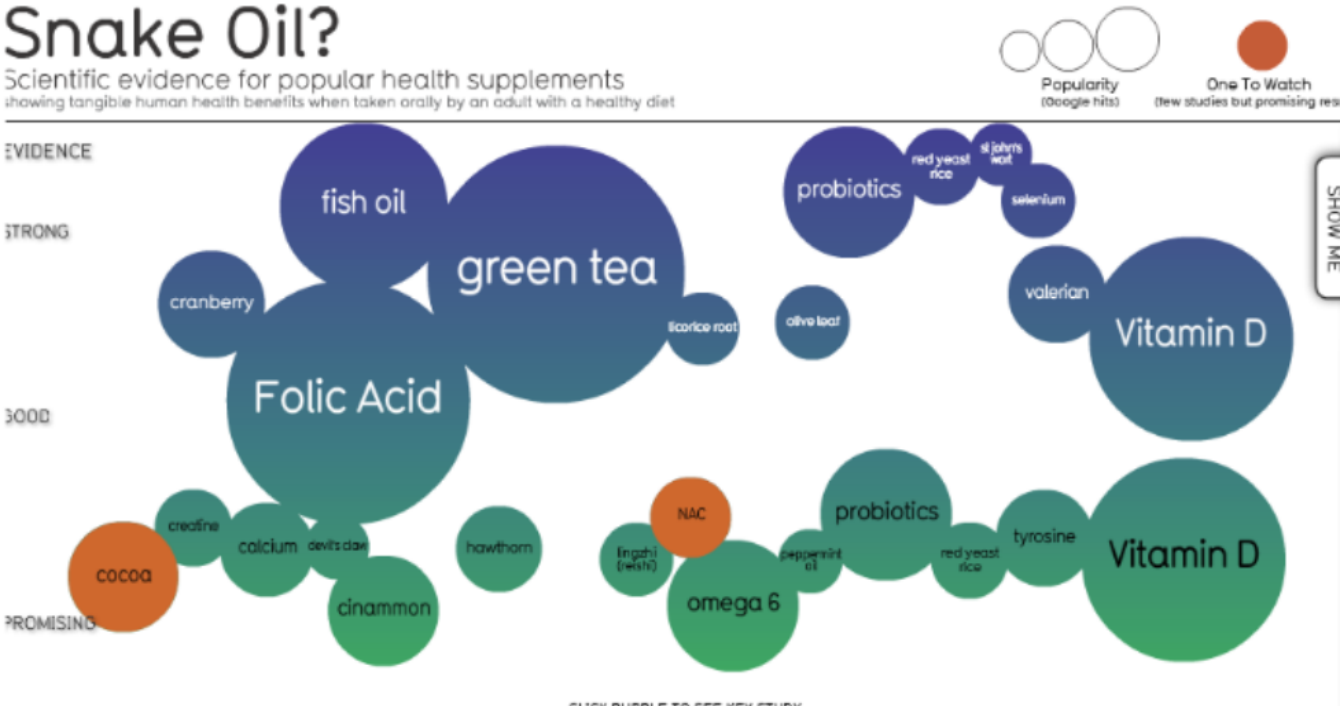
Classically, a pie-chart can be used to express relative fractions of a quantity;



A histogram/bar-chart can be used as well. double histogram, density plot



Bubble plots are also useful:



Wordles can be used to indicate relative strengths of keywords/topics. It is easy to create your own.

Simple graphics (bar charts, pie charts...) can be made more pleasing, using modern typography and layout techniques - here is a case in point.

Multivariate data

'Charles Minard's 1869 chart showing the number of men in Napoleon's 1812 Russian campaign army, their movements, as well as the temperature they encountered on the return path.' Specifically, the graph shows these 6 types of data (in 2D!): the number of Napoleon's troops; the distance traveled; temperature; latitude and longitude; direction of travel; and location relative to specific dates.

Spatial Data

Plotting spatial data (eg. incidence locations) on a map reveals patterns/trends in a 'direct' way - maps are 'intuitive' to humans...

It is quite useful for planning purposes, to visualize data over a map - eg. here are Starbucks locations..

Mined data, eg. associations, can be superposed over a map, eg. in a grocery store. Results can be used to redo the layout. A related topic is product placement.

As we saw earlier, a choropleth map shows spatial, aggregated data (that covers the entire region shown). These come in two varieties - unclassed (continuous scale), classed (discrete ranges).

Spatio-Temporal data 时空

Superposing time-varying data on a map reveals course, trends, etc. Such data could be visualized as animations, too

interactivity

Being able to INTERACT with data provides MORE understanding - we can selectively turn items on/off, drill down or roll up, explore the time dimension..

animation

Even passively watching data being animated, provides us fresh perspectives.

real-time!

Real-time visualization provides a level of immediacy/freshness/relevance/interest that is simply absent in non-real-time data..

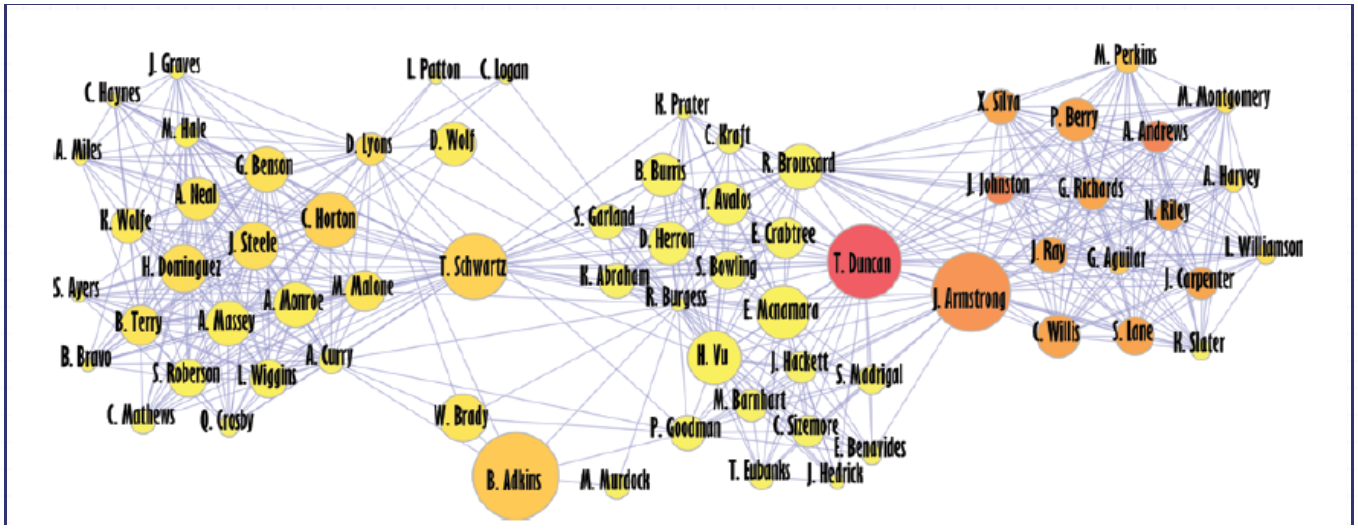
- world population growth [even more real-time stats!]
- local traffic (click on Options->Road Conditions->Fast--Slow) [amazing]
- earthquakes!
- stocks
- cybercrimes (!!); also this
- radio stations!

networks - node attrs

Network visualization is a very popular category - shows RELATIONSHIPS between entities.

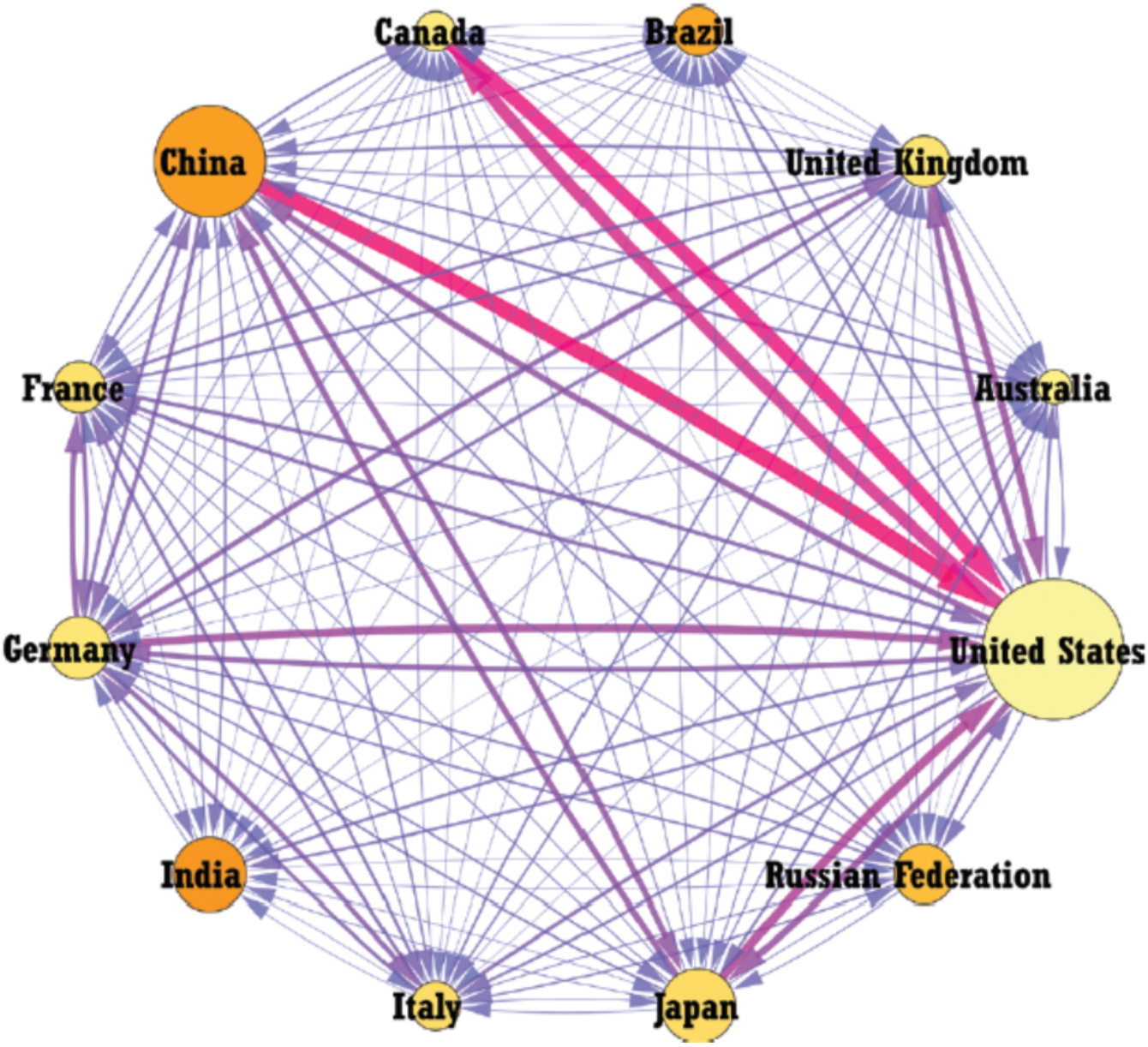
A diagram that maps email exchanges between family members:

Here is an enriched version that uses attrs and labels:



networks - edge attrs

We can use edge attrs (type, eg. arrows, dashes.., color, thickness etc.) to quantify data. The diagram below shows trade quantities between countries (2012, top 12 countries as per GDP):



Tools

During the past lectures, we've looked at a few data viz examples (eg. GIS data). Here is a systematic breakdown of ways to create all manner of data viz.

Data science software

- Weka
- KNIME
- RapidMiner

Using code

- R, Shiny, ggplot2...
- matplotlib [eg. here is a walkthrough]
- d3 (JS) and Protovis
- PGFPlots (LaTeX)

Online tools

- META-CHART
- datavisual
- infogram
- Online Charts
- Slemma

Math, analysis and plotting packages

- Mathematica
- MATLAB
- OriginLab [and alternatives]
- good old Excel

3rd party data-viz software [many are complete platforms, offering 'dashboards']

- Periscope [here is a clip]
- Tableau
- SiSense
- Qlik [intro' video]
- Salesforce dashboards; design tips; how to create one
- domo
- JMP (SAS)
- Mode

As an exercise, learn to use AS MANY of these as you can! Use notebooks for R and Python, and CodePen/jsfiddle for JS.

Data visualization science

Data viz is an art AND a science - there are principles, choices, tradeoffs. As for the principles, these encompass diverse disciplines such as visual perception, color theory, composition (grouping, contrast, harmony, symmetry..), design elements (line, tone, form, texture..), semiotics, etc.

As for what type of graphic to generate for a given type of data analysis, we can follow the guidelines here

Visualization Types (Reference Systems)

1. Charts: No reference system—e.g., Wordle.com, pie charts
2. Tables: Categorical axes that can be selected, reordered; cells can be color coded and might contain proportional symbols. Special kind of graph.
3. Graphs: Quantitative or qualitative (categorical) axes. Timelines, bar graphs, scatter plots.
4. Geospatial maps: Use latitude and longitude reference system. World or city maps.
5. Network layouts: Node position might depends on node attributes or node similarity.Trees: hierarchies, taxonomies,genealogies. Networks: social networks, migration flows.

Data governance & etc

Privacy

The FIP efforts in organizations followed five tenants:

1. Openness.
2. Disclosure.
3. Secondary usage limits.
4. Correctability.
5. Security.

Today, e-commerce companies collect LOTS of info about customers - for immediate sales, and for analytics. Too much data collection makes them vulnerable to theft/leakage etc.

Simply put, our lives ARE NOT 'PRIVATE' ANYMORE!

1. Practically all web sites track us - e-commerce, social media... - and exchange data among themselves and brokers.
2. Your medical data belongs to - your hospital!
3. You are under surveillance, esp. if you are in China or Japan [for now].
4. Your search data is not private. Through data inference, dots can be connected...
5. 'They' know where you've been! Eg. malls, public security cameras, your own phone - all know where you are...

Security

Unfortunately, data breaches are commonplace, and involve theft/exposure of value, identity...

Novel fronts: attacks on IoT [devices, data], including cars.

ETHICS

'Ethical' use of data involves multiple aspects!

First, there's the potential for 'rogue' AI. Eg. autonomous drones, and robot soldiers can act with bias, or equally badly, without bias and without morality.

Then there is the issue of fairness. This plays out for ex, in face recognition. IBM is contributing a dataset to help mitigate this.

A gov't can unfairly target protected groups...

Data, via ML, can be used to generate fake news, eg. fakevideo. This is a specific form of 'disinformation'. Disinformation was defined in Great Soviet Encyclopedia (1952) as "false information with the intention to deceive public opinion". Question: what, then, is 'misinformation'?

Here is one way to reduce bias, at the algorithm level.

Another angle to approach fairness is to improve the interpretability of ML.

Trust

How (much) can we (individuals) TRUST organizations (business, government, non-profit...) to RESPONSIBLY use **our** data?

Trust is proportional to transparency, value delivery, consequence acceptance.

Compliance 遵守

Compliance is a LEGAL/REGULATORY issue - what laws be passed, to help citizens have/gain control over their data? Aside: what is the difference between a law, regulation, and policy?

In the EU, there's GDPR [General Data Protection Regulation].

Interestingly (or not so), the US has a maze of regulations when it comes to digital privacy ("patch quilt protections"). To be fair, so did Europe, pre-GDPR.

Governance

The goal is to tease apart, learn about, and explore the connections between the following (data-related items): governance, curation, stewardship, MDM, provenance, metadata, security, privacy.

As you know, the purpose of capturing and storing data, is to process and benefit from it - this involves the use of statistics, data mining and machine learning.

But, that is not all there is to it! What about policies, procedures, rules, guidelines, practices... regarding the collection, storage and use of data?

Data Curation 管理

Regardless of collection procedures, analysis and usage, the ONE prime characteristic of data is QUALITY ('GIGO').

'Data curation' refers to set of processes and technologies ("methods and tools") that are focused on maintaining high-quality data in an organization, for the purposes of:

- visibility
- accessibility
- interoperability/heterogeneity
- reuse
- repurpose
- transparency
- ...

Any (which means ALL!) data-driven organization/s need(s) a 'data curation infrastructure' that supports curation practices and software.

key elements

Below are important points to keep in mind, while undertaking a data curation effort:

- the type of benefit derived from curated data, depends on the type of organization utilizing the curated datasets [eg. industrial R&D vs government vs new media companies]
- curation can be stimulated via incentives [that help justify the COST of curating]
- economic impact of curation can also help justify its need

- facilitating human-data interaction helps with curating - needs ways for non-technical users to handle data [eg. via natural language interfaces, semantic searching, viz, summarizing, transforming...] - building METADATA is a crucial step towards this
- large-scale curation efforts need to be hybrid between automated and human-involved efforts (curation by demonstration ['CBD'], crowdsourcing platforms, integration with enterprise data...]
- data curators need permission to access data they are curating; they need to be able to assign permissions (digital rights) to end-users of curated data; curators also need 'provenance' (data trail) info in order to determine curation specifics
- standards-based data models and representations (eg. ontology modeling using OWL) is necessary, for curation to include third-party/crowdsourced etc. data

Data Governance

From Wikipedia: 'Data governance is a data management concept concerning the capability that enables an organization to ensure that high data quality exists throughout the complete lifecycle of the data'.

In other words, governance == curation?

NOT REALLY: As per the DAMA International Data Management Book of Knowledge, "Data Governance (DG) is defined as the exercise of authority and control (planning, monitoring, and enforcement) over the management of data assets." In other words, Governance is about POLICIES, which can be seen as complementary to Curation.

So an organization would have Governance **policies** in place, which would aid in Curation's producing **customized business data.**

Data Provenance

Provenance ~= "lineage".

'Data provenance documents the inputs, entities, systems, and processes that influence data of interest, in effect providing a historical record of the data and its origins.'

Provenance has to do with origins, while lineage has to do with tracing data's 'journey' to the current point of usage.

Provenance/lineage is a form of metadata that needs to be added to data, during curation.

Provenance helps establish TRUST (or lack thereof) in data. With scientific data for example, this is crucial [incorrect/invalid data would lead of the acceptance of incorrect hypotheses!].

Here is an interesting list of provenance-related issues related to scientific research (just fyi).

Lineage could have life or death consequences.

MDM [Master Data Management]

MDM is the management of "master data" (similar to a master key): In business, master data management (MDM) is a method used to define and manage the critical data of an organization to provide, with data integration, a single point of reference. [Wikipedia]

The idea is to maintain a single (meaningful, accurate, complete, timely) reference for data that is shared - this is done in order to maintain consistency. The alternative (to replicate such data for each request) would be highly problematic - would lead to inconsistency, errors, wasted disk space, increased network traffic, etc.

Governance, Security, Privacy - a TRINITY!

Security breaches are almost 'normal' - Yahoo, Home Depot, Facebook, Uber, Equifax... what is going on?

Governance is not being followed properly - policies for handling data and accountability, culture of/training in handling data, (pro)actively managing (sensitive) data - these are missing.

Privacy and security breaches are very costly, literally - lost revenue in the form of customer attrition, fines (eg. levied by SEC), lawsuit awards... These losses are monumental, compared to investing in technologies and policies that guard against breaches!

Note that maintaining security and privacy both involve minimizing RISK - something that every business ought to be concerned about.

Security vs Privacy..

Data security/protection has to do with (preventing) UNAUTHORIZED access to data. Data privacy on the other hand, has to do with (limiting) AUTHORIZED access to data - related, but not identical!

Protection/security is a technical issue, related to protecting servers, encrypting data, restricting access (eg via passwords or biometrics), etc; privacy compliance on the other hand is a legal issue.

Also: data needs to be protectable first, before privacy can be ensured!

genAI

'Generative AI' - a revolution

Generative AI, very loosely speaking, 'runs a neural network BACKWARDS'!

Rather than learn to classify new data using existing data, why not GENERATE new data instead?

Researchers tried this, but with unimpressive results.

In 2014, Ian Goodfellow got a much better idea than the 'SOTA' - why not pair up TWO NNs in opposing order - one a generator (eager 'student'), and the other, a discriminator (strict 'teacher')? His invention is called a 'GAN'.

GAN

GAN stands for Generative Adversarial Network

Style transfer

An early use of genAI was/is to "add style" to imagery:

Encoders/decoders, autoencoders, VAEs

Encoder: a function (NN) that maps original input to LATENT/ENCODED space [decoder is the reverse]

Autoencoder: encoder + decoder combination - can GENERATE NEW OUTPUT (by interpolating a random point in latent space)!

Variational AE: the encoder produces a distribution rather than a single point [and the decoder uses a sampled point from the distribution].

Transformers

'Attention is all you need'

Revolution: non-fixed size and parallelizable self-attention mechanism (ie. computing word affinities).

The decoder takes a prompt (new point in latent space), INTERPOLATES over inputs (ALL English!!), generates output.

Transformers to... ChatGPT!

The core pre-trained LLM needs to be augmented with HFRL [a form of fine-tuning], to produce acceptable responses.

GPT extensions

- multimodal (eg text, images)
- plugin API
- GPT Store
- LLM apps

LLM extensions

- larger context [eg. RMT: https://arxiv.org/abs/2304.11062, and https://hazyresearch.stanford.edu/blog/2023-03-07-hyena and https://bdtechtalks.com/2023/11/27/streamingllm/amp/]
- infinite memory! [https://medium.com/@jordan_gibbs/how-to-create-your-own-gpt-voice-assistant-with-infinite-chat-memory-in-python-d8b8e93f6b21]
- architecture alterations (eg Rethinking Attention: https://arxiv.org/abs/2311.10642), alternate position encodings [including NO: https://arxiv.org/pdf/2203.16634.pdf, context-aware, rotary encoding...]
- open source!
- [quantization](https://xailient.com/blog/4-popular-model-compression-techniques-explained/#:~:text=Quantization compresses the original network,bit and even 1-bit.) [of weights to 16/8...bits, to compress model size]
- SLMs! [latest: https://mistral.ai/, https://starling.cs.berkeley.edu/, Orca 2...] (almost all at https://huggingface.co/models)

Prompt extensions/ LLM frameworks

- LangChain

- LlamaIndex
- Haystack
- https://cassio.org/
- ...
- CoT [chain of thoughts], ToT [tree of thoughts]...
- LangChain Expression Language [LCEL]
- agents!

Response extensions

- fine-tuning, eg. https://magazine.sebastianraschka.com/p/practical-tips-for-finetuning-llms and https://www.databricks.com/blog/efficient-fine-tuning-lora-guide-llms
- **RAG**!! Two kinds (eg. https://ai.plainenglish.io/beyond-tables-and-vectors-knowledge-graphs-for-ai-reasoning-46f0f8721894), more kinds... [eg. https://artificialcorner.com/ive-created-a-custom-gpt-that-scrapes-data-from-websites-9086aff58105]
- vector DBs https://medium.com/@zilliz_learn/what-is-a-real-vector-database-b391b0468279 and https://tiledb.com/blog/why-tiledb-as-a-vector-database
- LMSQL! https://towardsdatascience.com/lmql-sql-for-language-models-d7486d88c541

Custom GPTs

As a result of the extensions listed above, there is bound to be numerous, narrow-purpose GPTs,

genAI: other (non-plaintext) content

- CODE!!!
- images
- video
- music
- 3D CG
- ...