

W4111 – Introduction to Databases

Lecture 12: Data Engineering and Insight



Contents

Contents

- Finalizing the semester
 - Final exam
 - Homework 4 and project
 - Recitation/OHs schedule
- A brief overview of scalability and availability
- "Big Data:"
 - Big Data
 - Data Engineering
 - A little bit of Data Analytics
 - Some examples from HWs 3b and 4b
- Conclusions

Finalizing the Semester

Contents

- Final exam
 - 13-DEC from 9:00 AM to 12:00 AM in class lecture room.
 - If you require accommodation or modification
 - You must complete and submit the form specified in the Ed post.
 - We will contact you with instructions.
 - We have **much less** flexibility than the midterm because we have grade submission deadlines.
 - The structure, approach, format and rules are the same as for the midterm
 - We will provide a “cheat sheet” that you must print in advance.
 - You may optionally also have your own one-page, double-sided cheat sheet.
- Homework 4
 - Is written questions and we will release this weekend.
 - Due 15-DEC at 11:59 PM; **You may not use late days.**
 - The material is a subset of the material for the final and will help you study.
- Project
 - For various reasons, we ran out of time and will not have a project this semester.
 - I will publish a project over the winter break. If you want me to write a letter of recommendation in the future, you must complete and submit the project.

OHs/Recitation

- TA OHs
 - The TAs are updating their OH schedule and will publish.
 - I am encouraging them to hold OHs, but they are also students.
- Professor OHs and recitations
 - Unlike most semester, I am on a business trip and in meetings all week.
 - I will try to hold several OH/recitation sections but am not sure of the number and when I will hold them.
 - My schedule is very dynamic and there may be last minute announcements or changes. I will give as much notice as I can.
- I will be as active and responsive on Ed as I can.

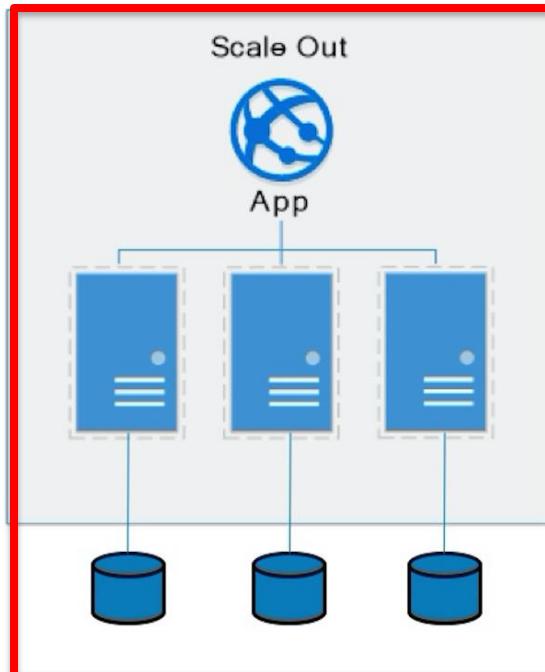
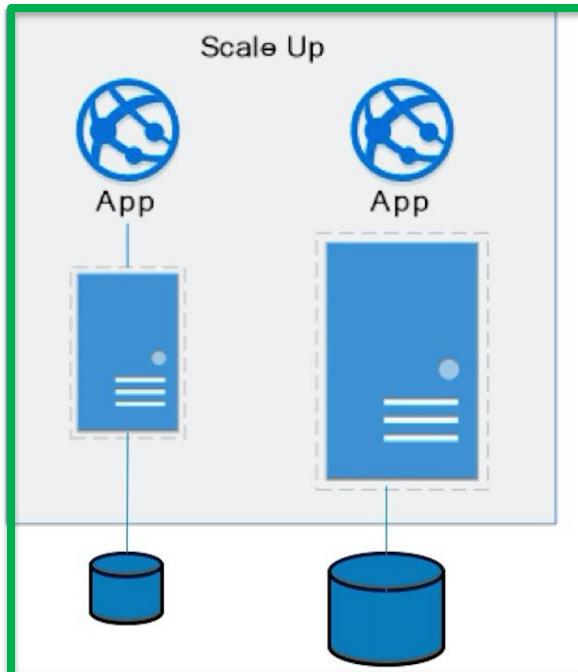
Scalability and Availability

Approaches to Scalability

Scalability is the property of a system to handle a growing amount of work by adding resources to the system.

Replace system with a bigger machine,
e.g. more memory, CPU,

Add another system.



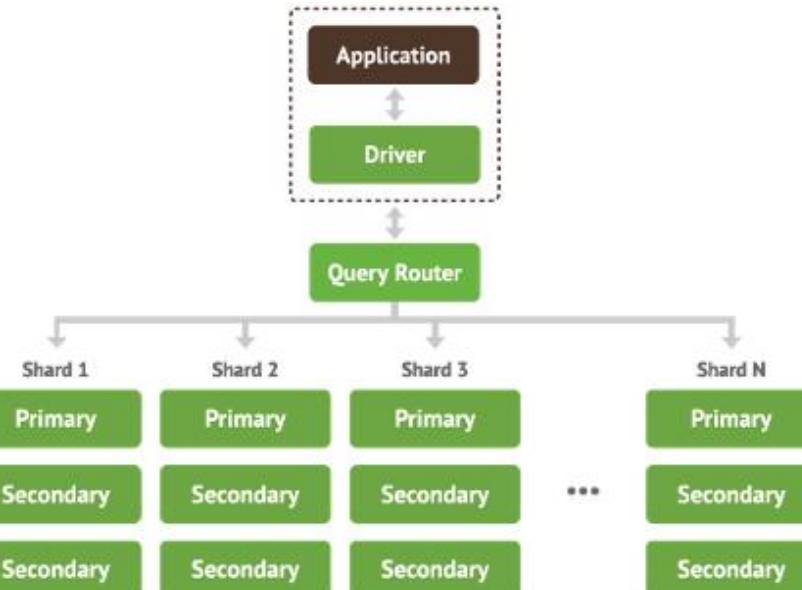
- **Scale-up:**
 - Less incremental.
 - More disruptive.
 - More expensive for extremely large systems.
 - Does not improve availability.
- **Scale-out:**
 - Incremental cost.
 - Data replication enables availability.
 - Does not work well for functions like JOIN, referential integrity,

Disk Architecture for Scale-Out

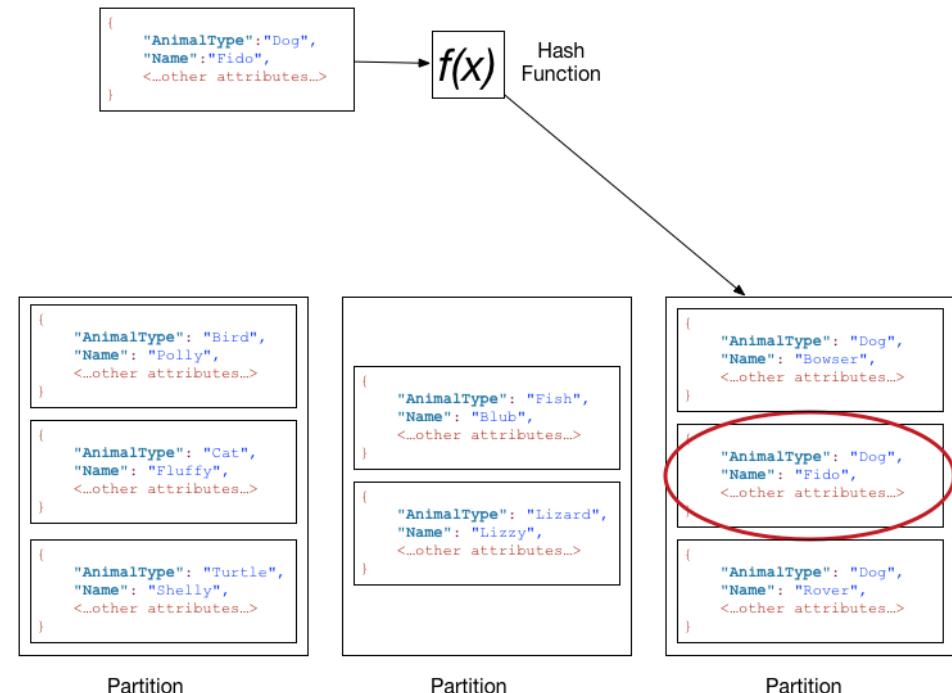
- Share disks:
 - Is basically scale-up for data/disks.
You can use NAS, SAN and RAID.
 - Isolation/Integrity requires distributed locking to control access from multiple database servers.
 - Share nothing:
 - Is basically scale-out for disks.
 - Data is partitioned into *shards* based on a function $f()$ applied to a key.
 - Can improve availability, at the code consistency, with data replication.
 - There is a router that sends requests to the proper shard based on the function.
-
- The diagram illustrates three disk architectures:
- Share Everything:** A single database server (DB) is connected to a single disk via an IP network. This is labeled "eg. Unix FS".
 - Share Disks:** Multiple database servers (DB) are connected to a central SAN Disk via an IP network and Fibre Channel (FC). This is labeled "eg. Oracle RAC".
 - Share Nothing:** Multiple database servers (DB) are connected to their own local storage disks via an IP network. This is labeled "eg. HDFS".

Shared Nothing, Scale-Out

MongoDB Sharding



DynamoDB Partitioning



Big Data

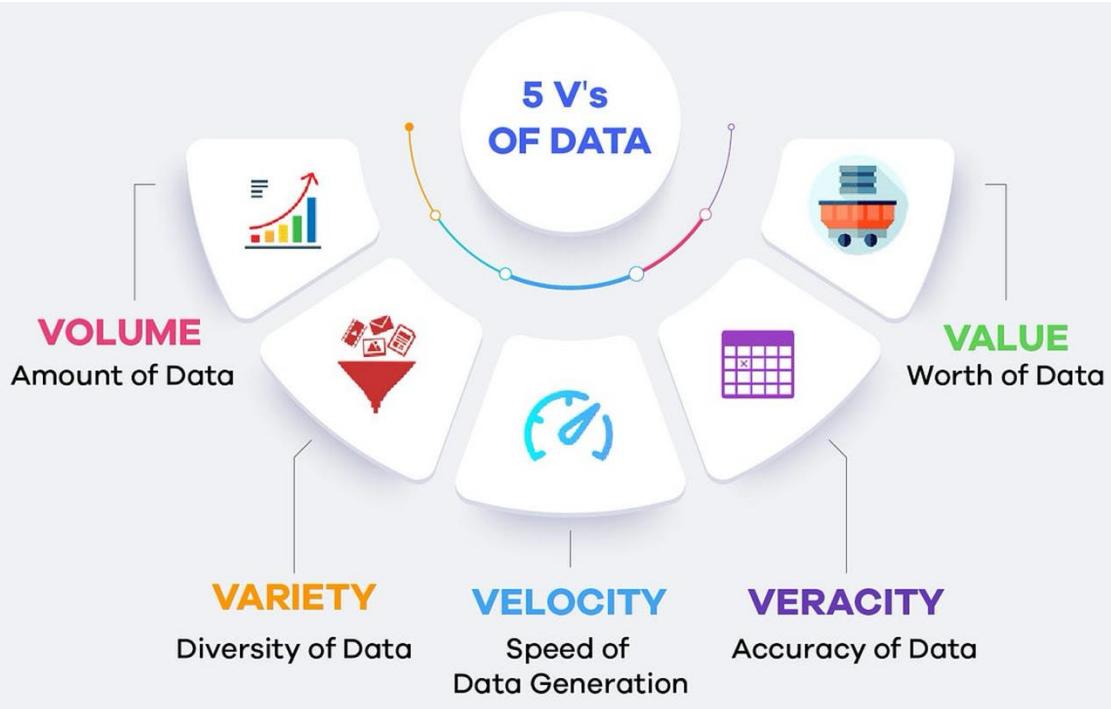
Data Engineering

Data Analysis

Big Data

5 V's of Big Data

<https://www.linkedin.com/pulse/big-data-management-revolution-parichehr-esmailian/>



https://medium.com/@get_excelsior/big-data-explained-the-5v-s-of-data-ae80cbe8ded1

Types Of Big Data

Following are the types of Big Data:

1. Structured
2. Unstructured
3. Semi-structured

Structured

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes.

Unstructured

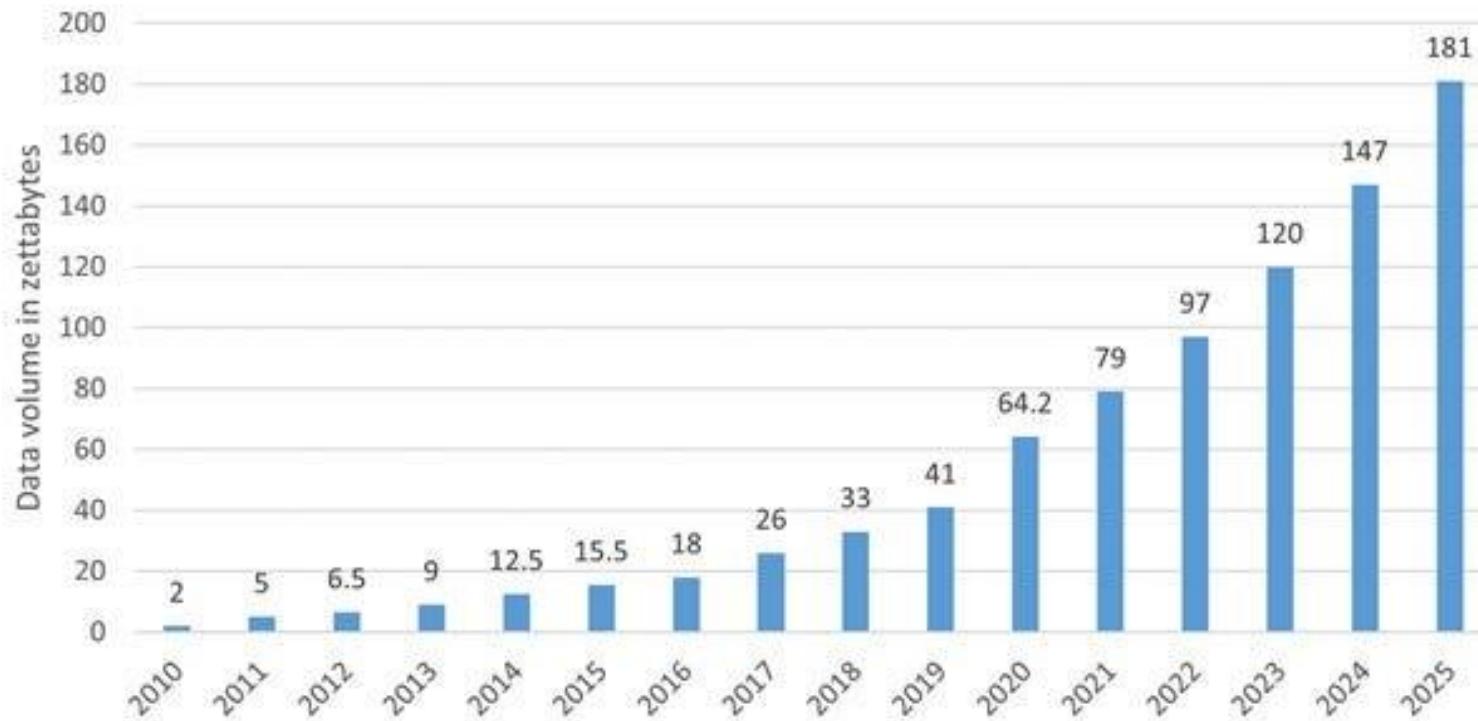
Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have wealth of data available with them but unfortunately, they don't know how to derive value out of it since this data is in its raw form or unstructured format.

Semi-structured

Semi-structured data can contain both the forms of data. We can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational **DBMS**. Example of semi-structured data is a data represented in an XML file.

Growth in Big Data

Volume of data created and replicated worldwide (source: IDC)



<https://medium.com/@mwaliph/exponential-growth-of-data-2f53df89124>



Overview

- **Data analytics:** the processing of data to infer patterns, correlations, or models for prediction
- Primarily used to make business decisions
 - Per individual customer
 - E.g., what product to suggest for purchase
 - Across all customers
 - E.g., what products to manufacture/stock, in what quantity
- Critical for businesses today



Overview (Cont.)

- Common steps in data analytics
 - Gather data from multiple sources into one location
 - Data warehouses also integrated data into common schema
 - Data often needs to be **extracted** from source formats, **transformed** to common schema, and **loaded** into the data warehouse
 - Can be done as **ETL (extract-transform-load)**, or **ELT (extract-load-transform)**
 - Generate aggregates and reports summarizing data
 - Dashboards showing graphical charts/reports
 - **Online analytical processing (OLAP) systems** allow interactive querying
 - Statistical analysis using tools such as R/SAS/SPSS
 - Including extensions for parallel processing of big data
 - Build **predictive models** and use the models for decision making



Overview (Cont.)

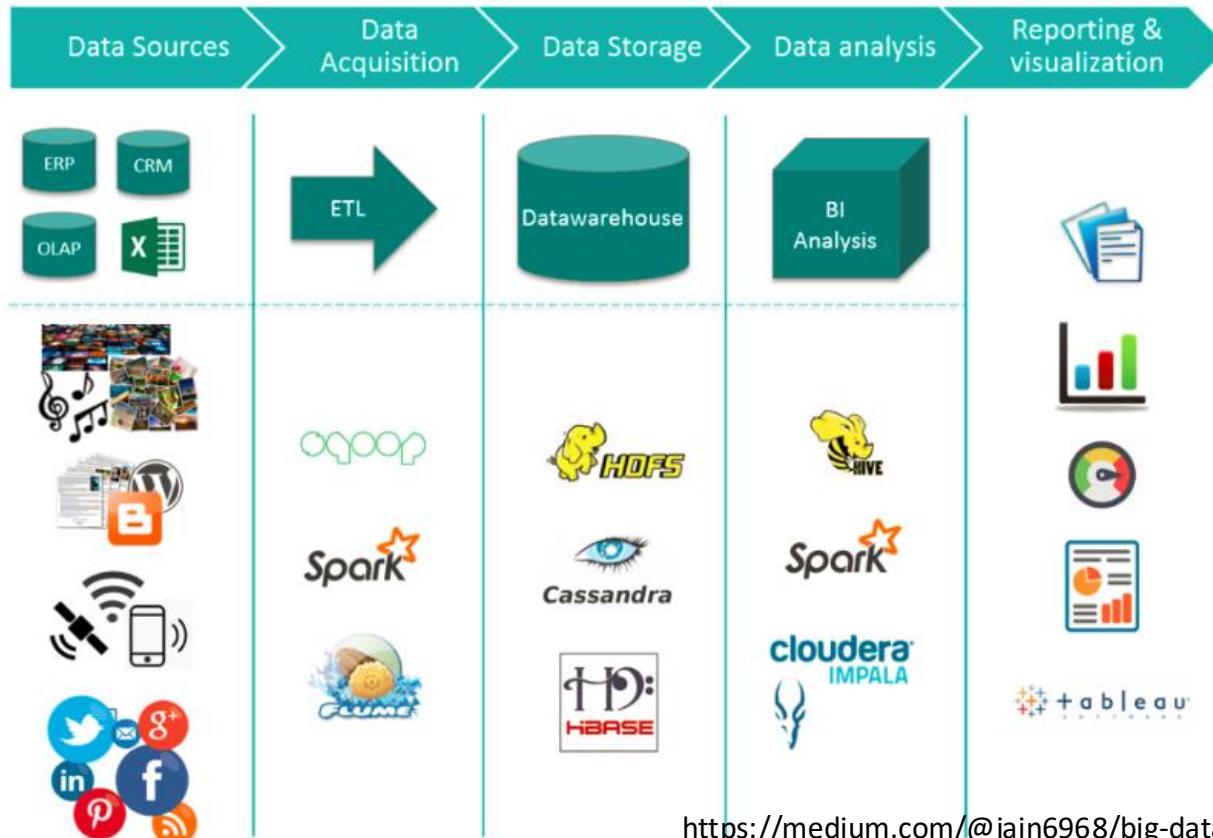
- Predictive models are widely used today
 - E.g., use customer profile features (e.g. income, age, gender, education, employment) and past history of a customer to predict likelihood of default on loan
 - and use prediction to make loan decision
 - E.g., use past history of sales (by season) to predict future sales
 - And use it to decide what/how much to produce/stock
 - And to target customers
- Other examples of business decisions:
 - What items to stock?
 - What insurance premium to change?
 - To whom to send advertisements?



Overview (Cont.)

- **Machine learning** techniques are key to finding patterns in data and making predictions
- **Data mining** extends techniques developed by machine-learning communities to run them on very large datasets
- The term **business intelligence (BI)** is synonym for data analytics
- The term **decision support** focuses on reporting and aggregation

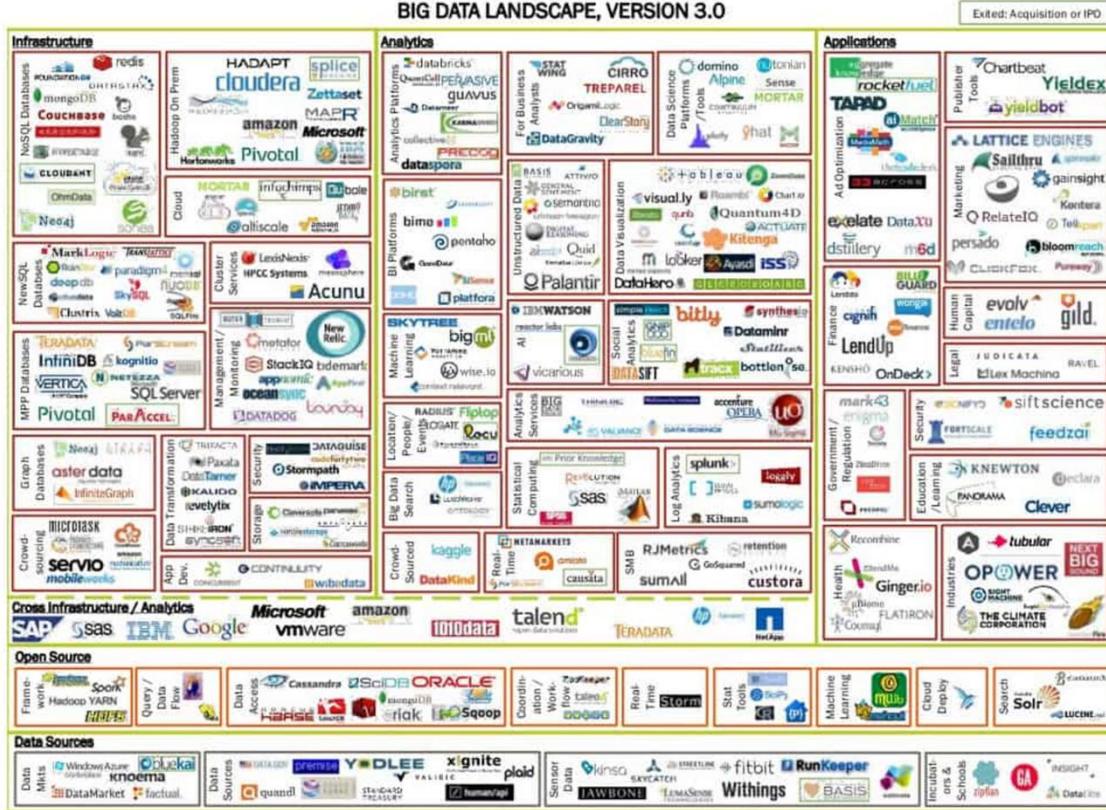
Big Data Ecosystem/Platform



Big Data Ecosystem/Platform

BIG DATA LANDSCAPE, VERSION 3.0

Exited: Acquisition or IPO



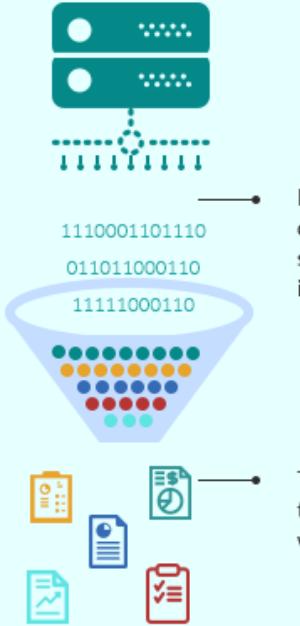
<https://dataconomy.com/2014/06/03/understanding-big-data-ecosystem/>

Data Engineering

Data Warehouse and Data Lake

DATA WAREHOUSE

VS DATA LAKE



- Data is processed and organized into a single schema before being put into the warehouse

- Raw and unstructured data goes into a data lake

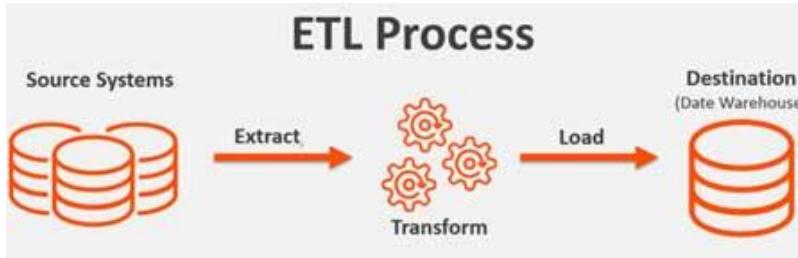


- The analysis is done on the cleansed data in the warehouse

- Data is selected and organized as and when needed

ETL Concepts

<https://databricks.com/glossary/extract-transform-load>



Extract

The first step of this process is extracting data from the target sources that could include an ERP, CRM, Streaming sources, and other enterprise systems as well as data from third-party sources. There are different ways to perform the extraction: **Three Data Extraction methods:**

1. Partial Extraction – The easiest way to obtain the data is if the source system notifies you when a record has been changed
2. Partial Extraction- with update notification – Not all systems can provide a notification in case an update has taken place; however, they can point those records that have been changed and provide an extract of such records.
3. Full extract – There are certain systems that cannot identify which data has been changed at all. In this case, a full extract is the only possibility to extract the data out of the system. This method requires having a copy of the last extract in the same format so you can identify the changes that have been made.

Transform

Next, the transform function converts the raw data that has been extracted from the source server. As it cannot be used in its original form in this stage it gets cleansed, mapped and transformed, often to a specific data schema, so it will meet operational needs. This process entails several transformation types that ensure the quality and integrity of data; below are the most common as well as advanced transformation types that prepare data for analysis:

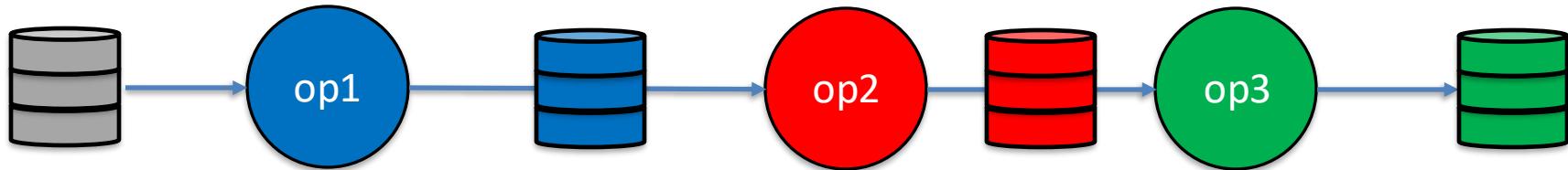
- Basic transformations:
- Cleaning
- Format revision
- Data threshold validation checks
- Restructuring
- Deduplication
- Advanced transformations:
 - Filtering
 - Merging
 - Splitting
 - Derivation
 - Summarization
 - Integration
 - Aggregation
 - Complex data validation

Load

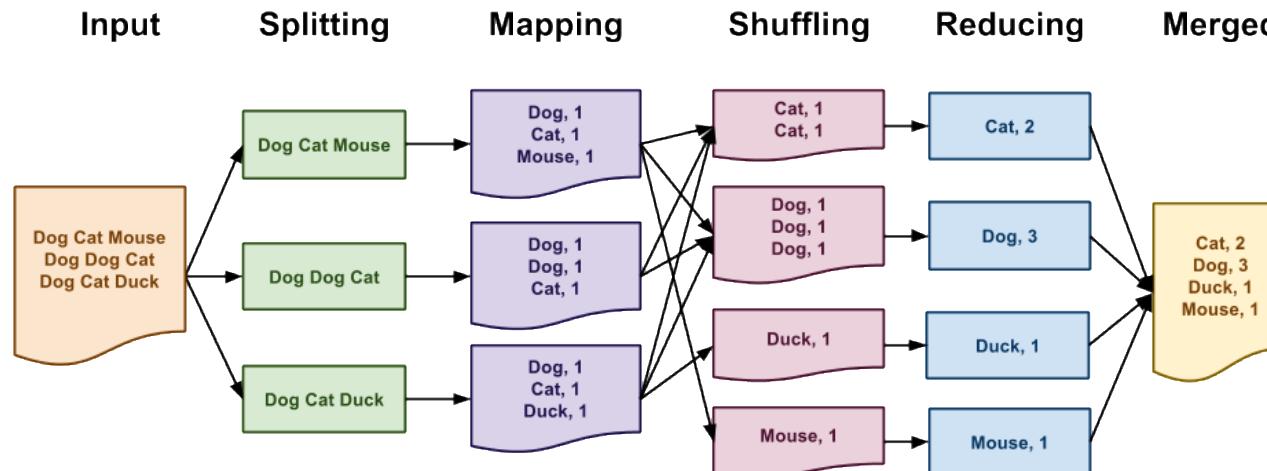
Finally, the load function is the process of writing converted data from a staging area to a target database, which may or may not have previously existed. Depending on the requirements of the application, this process may be either quite simple or intricate.

MapReduce

MapReduce is a data flow program with relatively simple operators on the data set.



With each operator implemented in parallel on multiple nodes for performance.



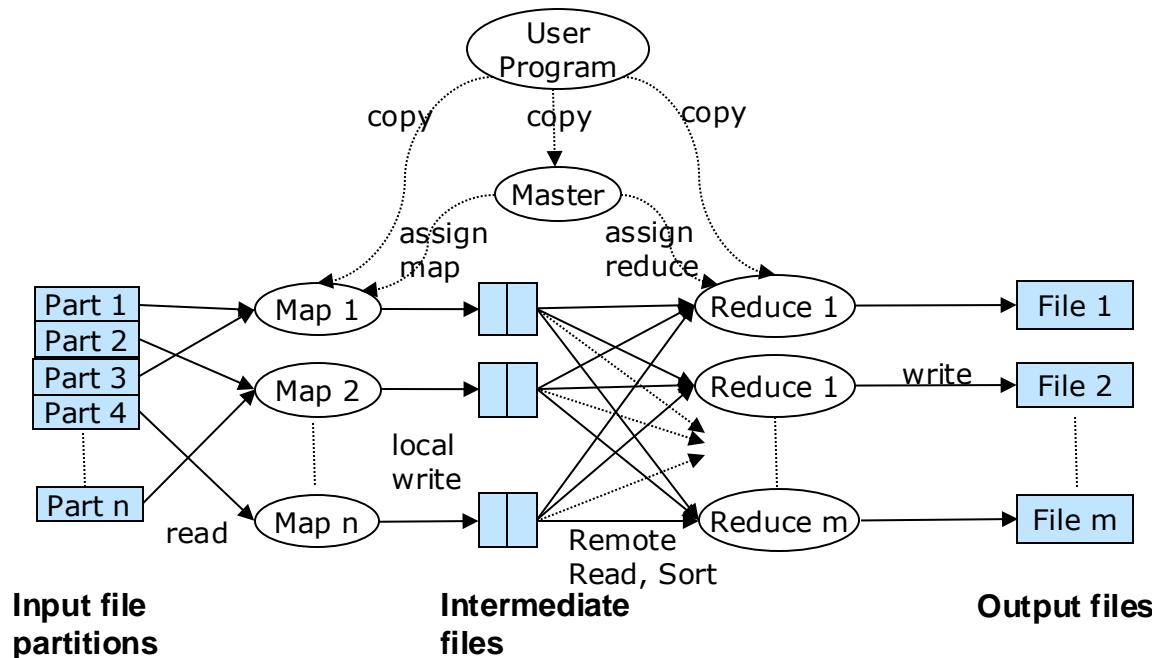
What we want more complex “operators?”

The Value of Blocks

- The “normal” file system abstraction is a stream.
- Consider a very large CSV file:
 - We can read using the normal file system functions and APIs,
 - But we have no idea where records start/end, and
 - Which records are on which blocks.
 - →
 - Parallel processing of data in a file is very hard using the stream model.
- Databases, big data tools, etc. surface concepts of record size, block I/Os, mapping of and organization of records to blocks, block location, →
 - Can exploit multiple processors to parallel process large datasets.
 - Organize the data to optimize processing, file I/Os, etc.



Parallel Processing of MapReduce Job





Map Reduce vs. Databases

- Map Reduce widely used for parallel processing
 - Google, Yahoo, and 100's of other companies
 - Example uses: compute PageRank, build keyword indices, do data analysis of web click logs,
 - Allows procedural code in map and reduce functions
 - Allows data of any type
- Many real-world uses of MapReduce cannot be expressed in SQL
- But many computations are much easier to express in SQL
 - Map Reduce is cumbersome for writing simple queries



Map Reduce vs. Databases (Cont.)

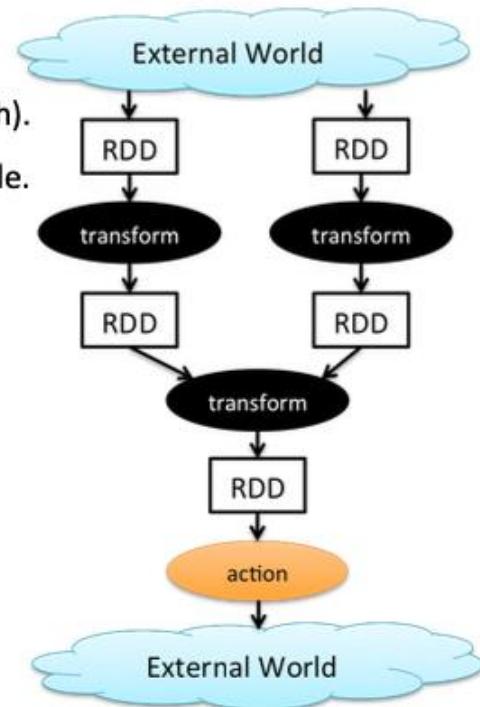
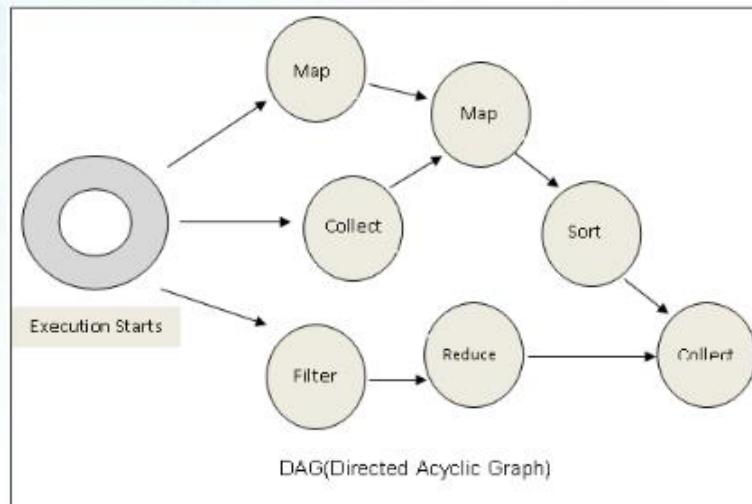
- Relational operations (select, project, join, aggregation, etc.) can be expressed using Map Reduce
- SQL queries can be translated into Map Reduce infrastructure for execution
 - Apache Hive SQL, Apache Pig Latin, Microsoft SCOPE
- Current generation execution engines support not only Map Reduce, but also other algebraic operations such as joins, aggregation, etc. natively.

Algebraic Operations

- Current generation execution engines
 - Natively support algebraic operations such as joins, aggregation, etc. natively.
 - Allow users to create their **own algebraic operators**
 - Support trees of algebraic operators that can be executed on **multiple nodes in parallel**
- E.g. Apache Tez, Spark
 - Tez provides low level API; Hive on Tez compiles SQL to Tez
 - Spark provides more user-friendly API
- In the relational model,
 - The are relations.
 - A fixed set of operators that produce relations from relations (closed).
 - Are declarative languages and compute the execution graph/plan.
- The Spark/... engines:
 - Enable programmers to develop and add new operators.
 - Operators convert reliable distributed datasets/data frames to new RDDs/data frames.
 - Data engineer explicitly defines the execution graph (data flow).

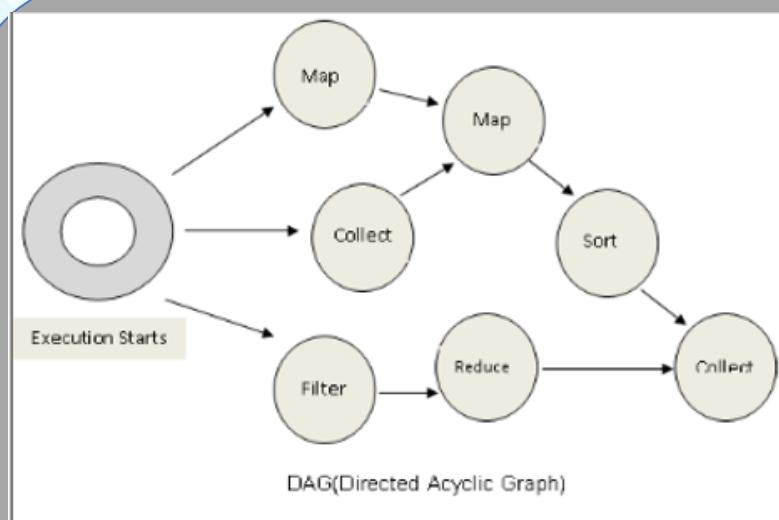
Spark

- All jobs in spark comprise a series of operators and run on a set of data.
- All the operators in a job are used to construct a DAG (Directed Acyclic Graph).
- The DAG is optimized by rearranging and combining operators where possible.



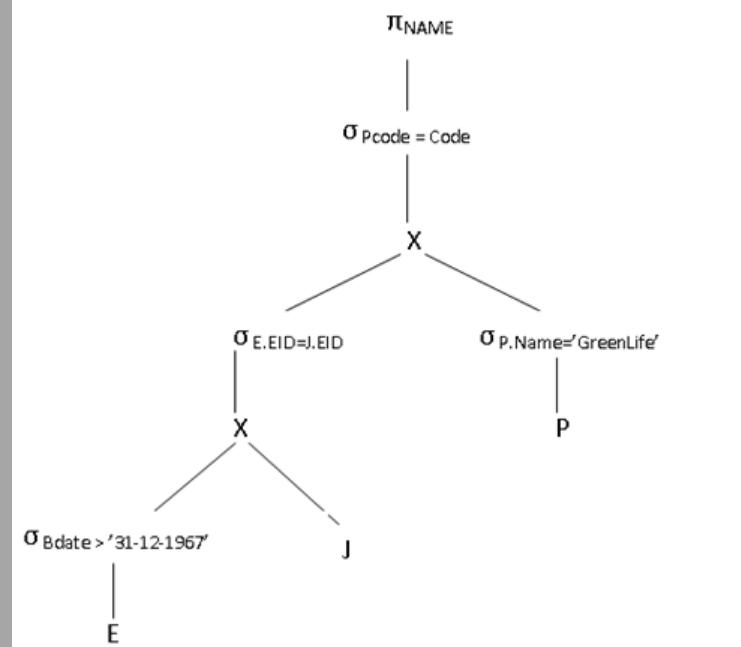
www.edureka.co/apache-spark-scala-training

Spark



Conceptually similar to query evaluation graph, but ...

- You explicitly define the graph.
- You can develop your own operators.



Algebraic Operations in Spark

- **Resilient Distributed Dataset (RDD) abstraction**
 - Collection of records that can be stored across multiple machines
- RDDs can be created by applying algebraic operations on other RDDs
- RDDs can be lazily computed when needed
- Spark programs can be written in Java/Scala/R
 - Our examples are in Java
- Spark makes use of Java 8 Lambda expressions; the code

```
s -> Arrays.asList(s.split(" ")).iterator()
```

defines unnamed function that takes argument s and executes the expression `Arrays.asList(s.split(" ")).iterator()` on the argument
- Lambda functions are particularly convenient as arguments to map, reduce and other functions

Spark/PySpark

DataFrame

edureka!

Inspired by DataFrames in R and Python (Pandas).

DataFrames API is designed to make big data processing on tabular data easier.

DataFrame is a distributed collection of data organized into named columns.

Provides operations to filter, group, or compute aggregates, and can be used with Spark SQL.

Can be constructed from structured data files, existing RDDs, tables in Hive, or external databases.

1. DataFrame in PySpark: Overview

In Apache Spark, a DataFrame is a distributed collection of rows under named columns. In simple terms, it is same as a table in relational database or an Excel sheet with Column headers. It also shares some common characteristics with RDD:

- Immutable in nature**: We can create DataFrame / RDD once but can't change it. And we can transform a DataFrame / RDD after applying transformations.
- Lazy Evaluations**: Which means that a task is not executed until an action is performed.
- Distributed**: RDD and DataFrame both are distributed in nature.

My first exposure to DataFrames was when I learnt about Pandas. Today, it is difficult for me to run my data science workflow with out Pandas DataFrames. So, when I saw similar functionality in Apache Spark, I was excited about the possibilities it opens up!

<https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/>

DataFrame features

Ability to scale from KBs to PBs

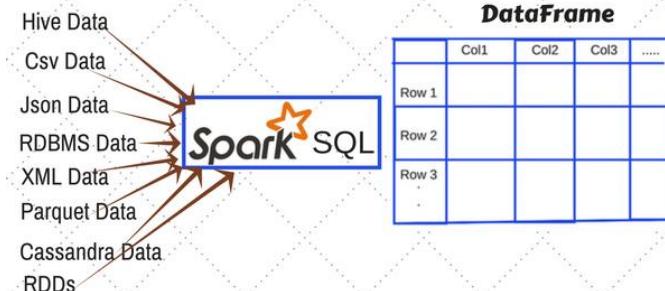
Support for a wide array of data formats and storage systems

State-of-the-art optimization and code generation through the spark SQL catalyst optimizer

Seamless integration with all big data tooling and infrastructure via spark

APIs for Python, Java, Scala, and R

Ways to Create DataFrame in Spark



AWS Glue Supports Spark

AWS Glue is a serverless data preparation service that makes it easy for data engineers, extract, transform, and load (ETL) developers, data analysts, and data scientists to extract, clean, enrich, normalize, and load data.



Automatic schema discovery

AWS Glue crawlers connect to your source or target data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata in your AWS Glue Data Catalog. The metadata is stored in tables in your data catalog and used in the authoring process of your ETL jobs. You can run crawlers on a schedule, on-demand, or trigger them based on an event to ensure that your metadata is up-to-date.

Built-In Classifiers in AWS Glue

AWS Glue provides built-in classifiers for various formats, including JSON, CSV, web logs, and many database systems.

If AWS Glue doesn't find a custom classifier that fits the input data format with 100 percent certainty, it invokes the built-in classifiers in the order shown in the following table. The built-in classifiers return a result to indicate whether the format matches (`certainty=1.0`) or does not match (`certainty=0.0`). The first classifier that has `certainty=1.0` provides the classification string and schema for a metadata table in your Data Catalog.

| Classifier type | Classification string | Notes |
|---------------------|-----------------------|--|
| | | in the document. For information about creating a custom XML classifier to specify rows in the document, see Writing XML Custom Classifiers . |
| Amazon log | ion | Reads the beginning of the file to determine format. |
| Combined Apache log | combined_apache | Determines log formats through a grok pattern. |
| Apache log | apache | Determines log formats through a grok pattern. |
| Linux kernel log | linux_kernel | Determines log formats through a grok pattern. |
| Microsoft log | microsoft_log | Determines log formats through a grok pattern. |
| Ruby log | ruby_logger | Reads the beginning of the file to determine format. |
| Squid 3.x log | squid | Reads the beginning of the file to determine format. |
| Redis monitor log | redismonlog | Reads the beginning of the file to determine format. |
| Redis log | redislog | Reads the beginning of the file to determine format. |

When Do I Use a Classifier?

You use classifiers when you crawl a data store to define metadata tables in the AWS Glue Data Catalog. You can set up your crawler with an ordered set of classifiers. When the crawler invokes a classifier, the classifier determines whether the data is recognized. If the classifier can't recognize the data or is not 100 percent certain, the crawler invokes the next classifier in the list to determine whether it can recognize the data.

Built-In Transforms

ApplyMapping

Maps source columns and data types from a `DynamicFrame` to target columns and data types in a returned `DynamicFrame`. You specify the mapping argument, which is a list of tuples that contain source column, source type, target column, and target type.

DropFields

Removes a field from a `DynamicFrame`. The output `DynamicFrame` contains fewer fields than the input. You specify which fields to remove using the `paths` argument. The `paths` argument points to a field in the schema tree structure using dot notation. For example, to remove field B, which is a child of field A in the tree, type `A.B` for the path.

DropNullFields

Removes null fields from a `DynamicFrame`. The output `DynamicFrame` does not contain fields of the null type in the schema.

Filter

Selects records from a `DynamicFrame` and returns a filtered `DynamicFrame`. You specify a function, such as a Lambda function, which determines whether a record is output (function returns true) or not (function returns false).

Join

Equijoin of two `DynamicFrames`. You specify the key fields in the schema of each frame to compare for equality. The output `DynamicFrame` contains rows where keys match.

Map

Applies a function to the records of a `DynamicFrame` and returns a transformed `DynamicFrame`. The supplied function is applied to each input record and transforms it to an output record. The map transform can add fields, delete fields, and perform lookups using an external API operation. If there is an exception, processing continues, and the record is marked as an error.

MapToCollection

Applies a transform to each `DynamicFrame` in a `DynamicFrameCollection`.

Built-In Transforms

Relationalize

Converts a `DynamicFrame` to a relational (rows and columns) form. Based on the data's schema, this transform flattens nested structures and creates `DynamicFrames` from arrays structures. The output is a collection of `DynamicFrames` that can result in data written to multiple tables.

RenameField

Renames a field in a `DynamicFrame`. The output is a `DynamicFrame` with the specified field renamed. You provide the new name and the path in the schema to the field to be renamed.

ResolveChoice

Use `ResolveChoice` to specify how a column should be handled when it contains values of multiple types. You can choose to either cast the column to a single data type, discard one or more of the types, or retain all types in either separate columns or a structure. You can select a different resolution policy for each column or specify a global policy that is applied to all columns.

SelectFields

Selects fields from a `DynamicFrame` to keep. The output is a `DynamicFrame` with only the selected fields. You provide the paths in the schema to the fields to keep.

SelectFromCollection

Selects one `DynamicFrame` from a collection of `DynamicFrames`. The output is the selected `DynamicFrame`. You provide an index to the `DynamicFrame` to select.

Spigot

Writes sample data from a `DynamicFrame`. Output is a JSON file in Amazon S3. You specify the Amazon S3 location and how to sample the `DynamicFrame`. Sampling can be a specified number of records from the beginning of the file or a probability factor used to pick records to write.

SplitFields

Splits fields into two `DynamicFrames`. Output is a collection of `DynamicFrames`: one with selected fields, and one with the remaining fields. You provide the paths in the schema to the selected fields.

SplitRows

Splits rows in a `DynamicFrame` based on a predicate. The output is a collection of two `DynamicFrames`: one with selected rows, and one with the remaining rows. You provide the comparison based on fields in the schema. For example, `A > 4`.

Unbox

Unboxes a string field from a `DynamicFrame`. The output is a `DynamicFrame` with the selected string field reformatted. The string field can be parsed and replaced with several fields. You provide a path in the schema for the string field to reformat and its current format type. For example, you might have a CSV file that has one field that is in JSON format `{"a": 3, "b": "foo", "c": 1.2}`. This transform can reformat the JSON into three fields: an `int`, a `string`, and a `double`.

Visual Tools (There are similar tools for SQL, ETL,)

TeamRosters

Last Saved at 11/29/2020, 6:27:12 AM Save Run

Visual Script Job details Run details

Data source - S3 bucket Read S3 Teams

Data source - S3 bucket Read S3 Appearances

Transform - ApplyMapping Subset Teams Columns

Data source - S3 bucket Read S3 People

Transform - ApplyMapping Subset Appearances...

Transform - ApplyMapping Renamed keys for Join...

Transform - ApplyMapping Subset People Columns

Transform - Join Join Appearances and...

Transform - Join Join People

Data target - S3 bucket S3 bucket

Node properties Transform Output schema

Join type
Select what kind of join to perform.

Inner join
Select all rows from both datasets that meet the join condition.

Join conditions
Select a key from each data input to set the condition of the join.

Subset Appearances Columns Renamed keys for Join Appearances and Teams

yearid = (right) yearid

Subset Appearances Columns Renamed keys for Join Appearances and Teams

teamid = (right) yearid

Add condition

Conceptual Example

- Consider the IMDB dataset. We should convert (other conversion needed)

| nconst | name | dateOfBirth | dateOfDeath | primaryProfession | knownFor |
|-----------|-----------------|-------------|-------------|-------------------------------------|---|
| nm0000003 | Brigitte Bardot | 1934 | | actress,soundtrack,music_department | tt0057345,tt0059956,tt0054452,tt0049189 |
| nm0000004 | John Belushi | 1949 | 1982 | actor,soundtrack,writer | tt0077975,tt0072562,tt0080455,tt0078723 |
| nm0000005 | Ingmar Bergman | 1918 | 2007 | writer,director,actor | tt0050986,tt0083922,tt0060827,tt0050976 |
| nm0000006 | Ingrid Bergman | 1915 | 1982 | actress,soundtrack,producer | tt0038787,tt0036855,tt0034583,tt0038109 |
| nm0000007 | Humphrey Bogart | 1899 | 1957 | actor,soundtrack,producer | tt0043265,tt0040897,tt0034583,tt0037382 |
| nm0000008 | Marlon Brando | 1924 | 2004 | actor,soundtrack,director | tt0078788,tt0070849,tt0047296,tt0068646 |
| nm0000009 | Richard Burton | 1925 | 1984 | actor,soundtrack,producer | tt0087803,tt0061184,tt0059749,tt0057877 |
| nm0000010 | James Cagney | 1899 | 1986 | actor,soundtrack,director | tt0042041,tt0035575,tt0029870,tt0031867 |

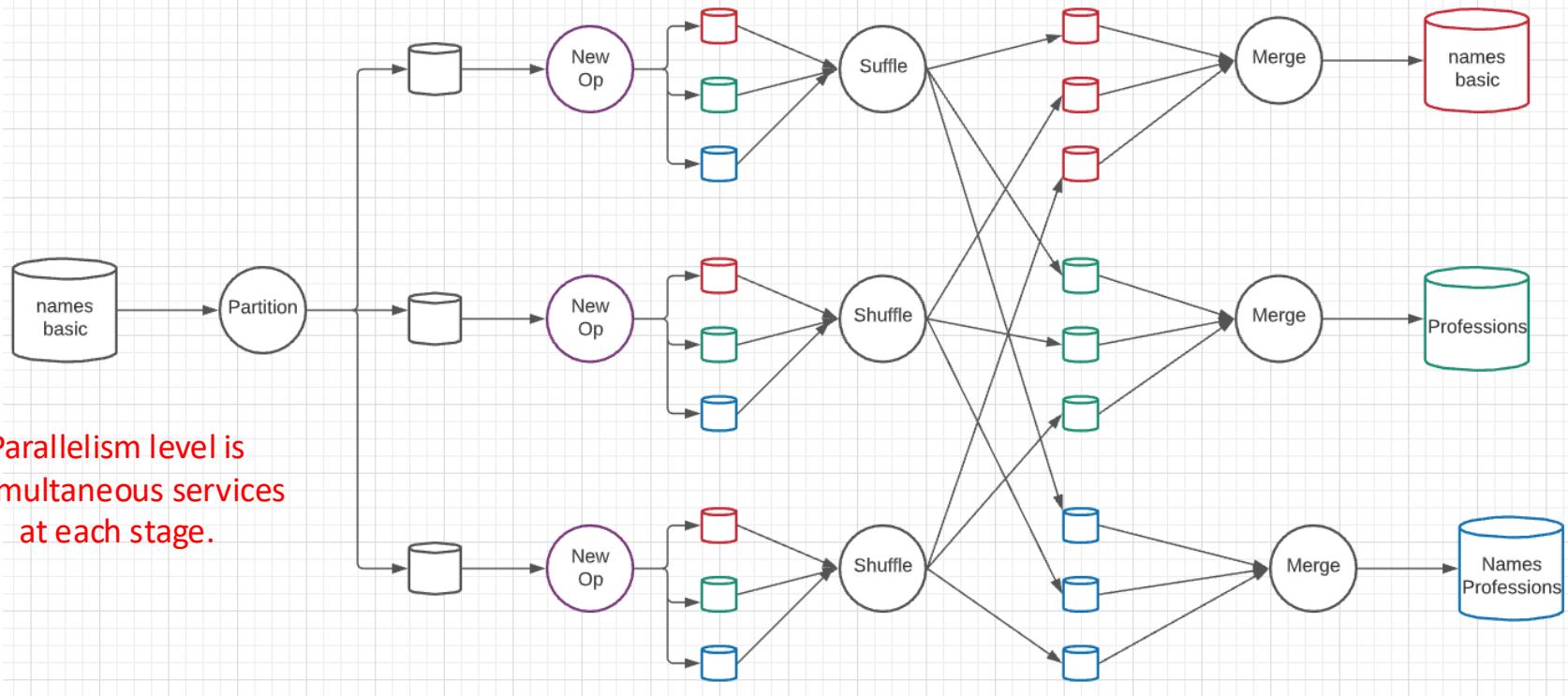
You did some operators for this example
in homework, and I did in class.

| | | | |
|-----------|-----------------|------|------|
| nm0000001 | Fred Astaire | 1899 | 1987 |
| nm0000002 | Lauren Bacall | 1924 | 2014 |
| nm0000003 | Brigitte Bardot | 1934 | |
| nm0000004 | John Belushi | 1949 | 1982 |
| nm0000005 | Ingmar Bergman | 1918 | 2007 |
| nm0000006 | Ingrid Bergman | 1915 | 1982 |
| nm0000007 | Humphrey Bogart | 1899 | 1957 |
| nm0000008 | Marlon Brando | 1924 | 2004 |
| nm0000009 | Richard Burton | 1925 | 1984 |
| nm0000010 | James Cagney | 1899 | 1986 |

| nconst | profession_id |
|-----------|---------------|
| nm0000001 | 1 |
| nm0000001 | 2 |
| nm0000001 | 3 |
| nm0000002 | 3 |
| nm0000002 | 4 |
| nm0000003 | 3 |
| nm0000003 | 4 |
| nm0000003 | 5 |
| nm0000004 | 1 |
| nm0000004 | 3 |

| Profession | Profession_id |
|--------------------|---------------|
| actor | 1 |
| miscellaneous | 2 |
| soundtrack | 3 |
| actress | 4 |
| music_department | 5 |
| writer | 6 |
| director | 7 |
| producer | 8 |
| make_up_department | 9 |
| composer | 10 |
| assistant_director | 11 |

A New Algebraic Operator



- Input is a dataset and column ID
- Output is three datasets: dataset with column removed, table of distinct column values, associative entity.



Map Reduce vs. Databases

- Map Reduce widely used for parallel processing
 - Google, Yahoo, and 100's of other companies
 - Example uses: compute PageRank, build keyword indices, do data analysis of web click logs,
 - Allows procedural code in map and reduce functions
 - Allows data of any type
- Many real-world uses of MapReduce cannot be expressed in SQL
- But many computations are much easier to express in SQL
 - Map Reduce is cumbersome for writing simple queries



Map Reduce vs. Databases (Cont.)

- Relational operations (select, project, join, aggregation, etc.) can be expressed using Map Reduce
- SQL queries can be translated into Map Reduce infrastructure for execution
 - Apache Hive SQL, Apache Pig Latin, Microsoft SCOPE
- Current generation execution engines support not only Map Reduce, but also other algebraic operations such as joins, aggregation, etc. natively.



STREAMING DATA



Streaming Data and Applications

- **Streaming data** refers to data that arrives in a continuous fashion
 - Contrast to **data-at-rest**
- Applications include:
 - Stock market: stream of trades
 - e-commerce site: purchases, searches
 - Sensors: sensor readings
 - Internet of things
 - Network monitoring data
 - Social media: tweets and posts can be viewed as a stream
- Queries on streams can be very useful
 - Monitoring, alerts, automated triggering of actions



Querying Streaming Data

Approaches to querying streams:

- **Windowing:** Break up stream into windows, and queries are run on windows
 - Stream query languages support window operations
 - Windows may be based on time or tuples
 - Must figure out when all tuples in a window have been seen
 - Easy if stream totally ordered by timestamp
 - **Punctuations** specify that all future tuples have timestamp greater than some value
- **Continuous Queries:** Queries written e.g. in SQL, output partial results based on stream seen so far; query results updated continuously
 - Have some applications, but can lead to flood of updates



Querying Streaming Data (Cont.)

Approaches to querying streams (Cont.):

- **Algebraic operators on streams:**
 - Each operator consumes tuples from a stream and outputs tuples
 - Operators can be written e.g., in an imperative language
 - Operator may maintain state
- **Pattern matching:**
 - Queries specify patterns, system detects occurrences of patterns and triggers actions
 - **Complex Event Processing (CEP)** systems
 - E.g., Microsoft StreamInsight, Flink CEP, Oracle Event Processing



Stream Processing Architectures

- Many stream processing systems are purely in-memory, and do not persist data
- **Lambda architecture:** split stream into two, one output goes to stream processing system and the other to a database for storage
 - Easy to implement and widely used
 - But often leads to duplication of querying effort, once on streaming system and once in database



Stream Extensions to SQL

- SQL Window functions described in Section 5.5.2
- Streaming systems often support more window types
 - **Tumbling window**
 - E.g., hourly windows, windows don't overlap
 - **Hopping window**
 - E.g., hourly window computed every 20 minutes
 - **Sliding window**
 - Window of specified size (based on timestamp interval or number of tuples) around each incoming tuple
 - **Session window**
 - Groups tuples based on user sessions



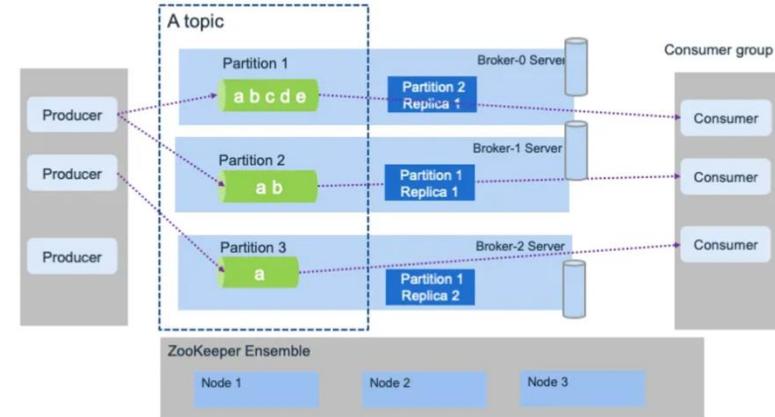
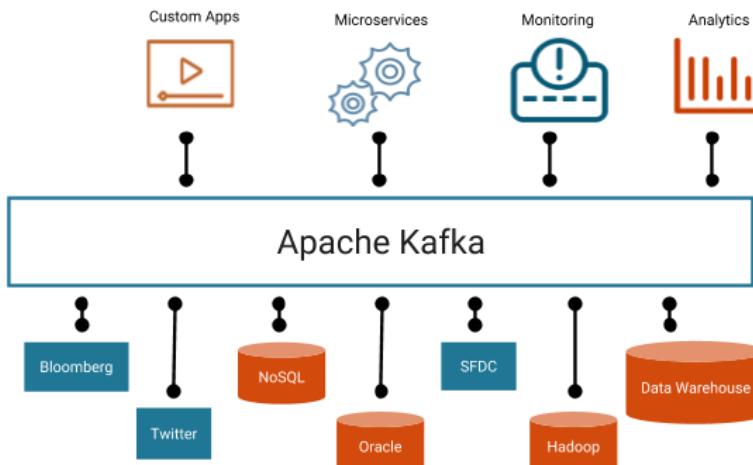
Window Syntax in SQL

- Windowing syntax varies widely by system
- E.g., in Azure Stream Analytics SQL:

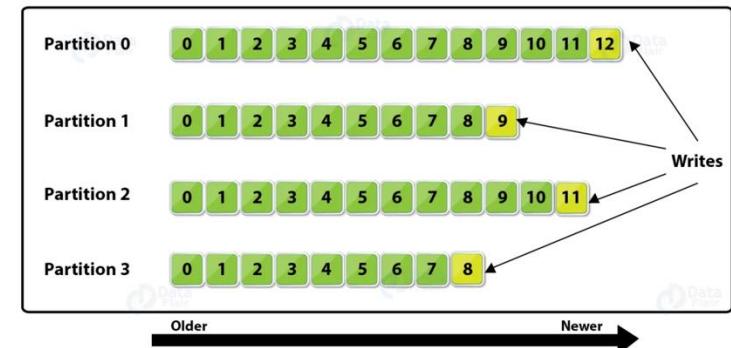
```
select item, System.Timestamp as window end, sum(amount)
from order timestamp by datetime
group by itemid, tumblingwindow(hour, 1)
```

- Aggregates are applied on windows
- Result of windowing operation on a stream is a relation
- Many systems support stream-relation joins
- Stream-stream joins often require join conditions to specify bound on timestamp gap between matching tuples
 - E.g., tuples must be at most 30 minutes apart in timestamp

Stream Processing – Kafka Example



Kafka Topic Partitions Layout

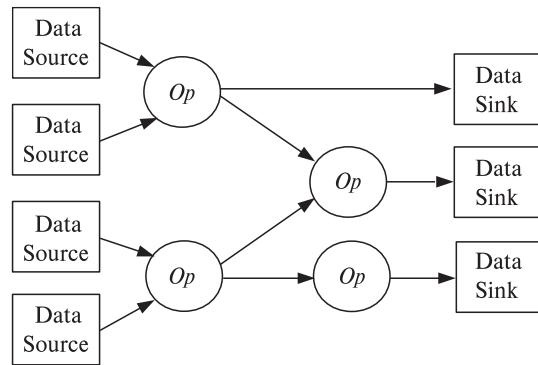


- Kafka architecture has 4 actors:
 - Broker
 - Zookeeper
 - Producer
 - Consumer
- There may be multiple brokers and multiple topics.
- Topics are divided into partitions based on an event/message key to enable parallelism.

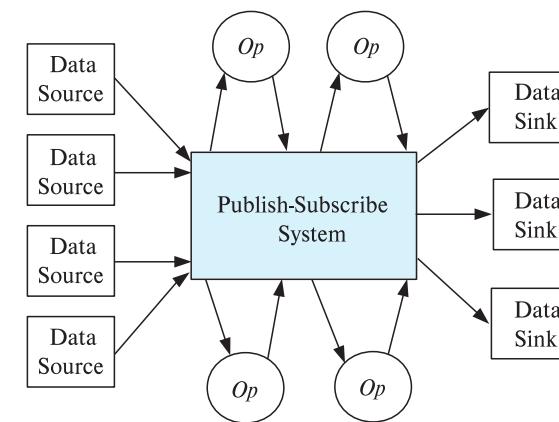


Algebraic Operations on Streams

- Tuples in streams need to be routed to operators
- Routing of streams using DAG and publish-subscribe representations
 - Used in Apache Storm and Apache Kafka respective



(a) DAG representation of streaming data flow



(b) Publish-subscribe representation of streaming data flow

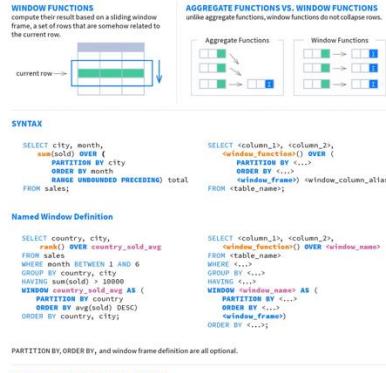


Publish Subscribe Systems

- **Publish-subscribe (pub-sub)** systems provide convenient abstraction for processing streams
 - Tuples in a stream are published to a topic
 - Consumers subscribe to topic
- Parallel pub-sub systems allow tuples in a topic to be partitioned across multiple machines
- **Apache Kafka** is a popular parallel pub-sub system widely used to manage streaming data
- More details in book

Window Functions in SQL

SQL Window Functions Cheat Sheet



LOGICAL ORDER OF OPERATIONS IN SQL

1. FROM, JOIN
2. SELECT
3. GROUP BY
4. aggregate functions
5. ORDER BY
6. window functions

You can use window functions in SELECT and ORDER BY. However, you can't put window functions anywhere in the FROM, WHERE, GROUP BY, or HAVING clauses.

Try out the interactive Window Functions course at LearnSQL.com, and check out our other SQL courses.

- These are useful and powerful.
- They will not be on the final, but you should be aware of them.
- Similar queries and capabilities are extremely useful for stream processing.

LearnSQL.com

SQL Window Functions Cheat Sheet

LIST OF WINDOW FUNCTIONS

- `row_number()` - integer number for each row within partition, with different numbers for tied values
- `rank()` - ranking within partition, with gaps and same ranking for tied values
- `dense_rank()` - ranking within partition, with no gaps and same ranking for tied values

- `count()`
- `avg()`
- `sum()`

Ranking Functions

- `row_number()`
- `rank()`
- `dense_rank()`

Distribution Functions

- `percent_rank()`

Analytic Functions

- `lead()`
- `lag()`
- `ntile()`
- `first_value()`
- `last_value()`
- `nth_value()`

AGGREGATE FUNCTIONS

- `avg(expr)` - average value for rows within the window frame
- `count(expr)` - count of non-null values for rows within the window frame
- `max(expr)` - maximum value within the window frame
- `min(expr)` - minimum value within the window frame
- `sum(expr)` - sum of values within the window frame

ORDER BY and Window Frame

- Aggregate functions do not require an ORDER BY. They accept window frame definition (ROWS, RANGE, GROUPS).

NTILE(n)

- divides rows within a partition as equally as possible into n groups, and sorts each group

ORDER BY and Window Frame

- Aggregate functions do not require an ORDER BY. They accept window frame definition (ROWS, RANGE, GROUPS).

Try out the interactive Window Functions course or explore all LearnSQL.com courses.

DISTRIBUTION FUNCTIONS

- `percent_rank(expr)` - cumulative ranking number of a row—a value in (0, 1] interval: rank + (1 - rank) * (number of rows - 1)

- `cume_dist()` - the cumulative distribution of a value within a group of values, i.e., the number of rows with values less than or equal to the current row's value divided by the total number of rows; a value in (0, 1] interval: rank / (number of rows - 1)

| city | sold | percent_rank |
|--------|------|--------------|
| Paris | 7 | 1 |
| Rome | 7 | 2 |
| London | 8.5 | 3 |
| Berlin | 8.5 | 4 |
| Moscow | 9 | 5 |
| Madrid | 9 | 6 |
| Oulu | 10 | 7 |

* without this row 50% of the values are less than this row's value

* 80% of values are less than or equal to this one's row's value

ORDER BY and Window Frame: distribution functions require ORDER BY. They do not accept window frame definitions (ROWS, RANGE, GROUPS).

ANALYTIC FUNCTIONS

- `lead(expr, offset, default)` - the value for the row offset rows after the current offset and default are optional; offset value must be a small integer

- `lag(expr, offset, default)` - the value for the row offset before the current offset; offset and default are optional; default value offset = 1, default = NULL

| city | month | sold | lead(sold, 1, 100) | lag(sold, 1, 100) |
|--------|-------|------|--------------------|-------------------|
| Paris | 1 | 500 | 500 | 500 |
| Rome | 2 | 500 | 500 | 500 |
| London | 3 | 400 | 500 | 500 |
| Berlin | 4 | 400 | 500 | 500 |

* first_value(expr) - the value for the first row within the window frame

* last_value(expr) - the value for the last row within the window frame

| city | month | sold | first_value(sold) | last_value(sold) |
|--------|-------|------|-------------------|------------------|
| Paris | 1 | 300 | 300 | 400 |
| Rome | 2 | 300 | 300 | 400 |
| London | 3 | 400 | 300 | 400 |
| Berlin | 4 | 200 | 300 | 400 |

Note: You usually want to use RANGE BETWEEN UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING with `last_value(expr)`. With the default window frame for ORDER BY, RANGE UNBOUNDED PRECEDING, `last_value(expr)` gives the value for the last row within the window frame.

* nth_value(expr, n) - the value for the n-th row within the window frame; n must be an integer

| city | month | sold | nth_value(sold, 1) | nth_value(sold, 2) |
|--------|-------|------|--------------------|--------------------|
| Paris | 1 | 300 | 300 | 300 |
| Rome | 2 | 300 | 300 | 300 |
| London | 3 | 400 | 300 | 300 |
| Berlin | 4 | 200 | 300 | 300 |

Data Analysis and Insight



Data Analysis and OLAP

- **Online Analytical Processing (OLAP)**
 - Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)
- We use the following relation to illustrate OLAP concepts
 - *sales (item_name, color, clothes_size, quantity)*

This is a simplified version of the *sales* fact table joined with the dimension tables, and many attributes removed (and some renamed)



Example sales relation

| item_name | color | clothes_size | quantity |
|-----------|--------|--------------|----------|
| dress | dark | small | 2 |
| dress | dark | medium | 6 |
| dress | dark | large | 12 |
| dress | pastel | small | 4 |
| dress | pastel | medium | 3 |
| dress | pastel | large | 3 |
| dress | white | small | 2 |
| dress | white | medium | 3 |
| dress | white | large | 0 |
| pants | dark | small | 14 |
| pants | dark | medium | 6 |
| pants | dark | large | 0 |
| pants | pastel | small | 1 |
| pants | pastel | medium | 0 |
| pants | pastel | large | 1 |
| pants | white | small | 3 |
| pants | white | medium | 0 |
| pants | white | large | 2 |
| shirt | dark | small | 2 |
| shirt | dark | medium | 6 |
| shirt | dark | large | 6 |
| shirt | pastel | small | 4 |
| shirt | pastel | medium | 1 |
| shirt | pastel | large | 2 |
| shirt | white | small | 17 |
| shirt | white | medium | 1 |
| shirt | white | large | 10 |
| skirt | dark | small | 2 |
| skirt | dark | medium | 5 |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |



Cross Tabulation of sales by *item_name* and *color*

clothes_size **all**

| <i>item_name</i> | <i>color</i> | | | | total |
|------------------|--------------|--------|-------|--|-------|
| | dark | pastel | white | | |
| skirt | 8 | 35 | 10 | | 53 |
| dress | 20 | 10 | 5 | | 35 |
| shirt | 14 | 7 | 28 | | 49 |
| pants | 20 | 2 | 5 | | 27 |
| total | 62 | 54 | 48 | | 164 |

- The table above is an example of a **cross-tabulation (cross-tab)**, also referred to as a **pivot-table**.
 - Values for one of the dimension attributes form the row headers
 - Values for another dimension attribute form the column headers
 - Other dimension attributes are listed on top
 - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.



Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have n dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube

| | | item_name | | | | | clothes_size | | | |
|--------|----|-----------|-------|--------|-------|-----|--------------|--------|--------|-------|
| | | skirt | dress | shirt | pants | all | all | large | medium | small |
| color | | 2 | 5 | 3 | 1 | 11 | 34 | 4 | 16 | 18 |
| dark | 4 | 7 | 6 | 12 | 29 | 2 | 8 | 5 | 7 | 22 |
| | 8 | 20 | 14 | 20 | 62 | 35 | 10 | 7 | 2 | 54 |
| pastel | 35 | 10 | 7 | 2 | 54 | 10 | 5 | 28 | 5 | 48 |
| | 10 | 5 | 28 | 5 | 48 | 53 | 35 | 49 | 27 | 164 |
| white | 16 | 4 | 9 | 21 | 77 | 77 | 42 | 45 | 42 | 45 |
| | 18 | 18 | 9 | 21 | 77 | all | large | medium | small | all |
| all | | all | large | medium | small | all | large | medium | small | all |



Online Analytical Processing Operations

- **Pivoting:** changing the dimensions used in a cross-tab
 - E.g., moving colors to column names
- **Slicing:** creating a cross-tab for fixed values only
 - E.g., fixing color to white and size to small
 - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **Rollup:** moving from finer-granularity data to a coarser granularity
 - E.g., aggregating away an attribute
 - E.g., moving from aggregates by day to aggregates by month or year
- **Drill down:** The opposite operation - that of moving from coarser-granularity data to finer-granularity data

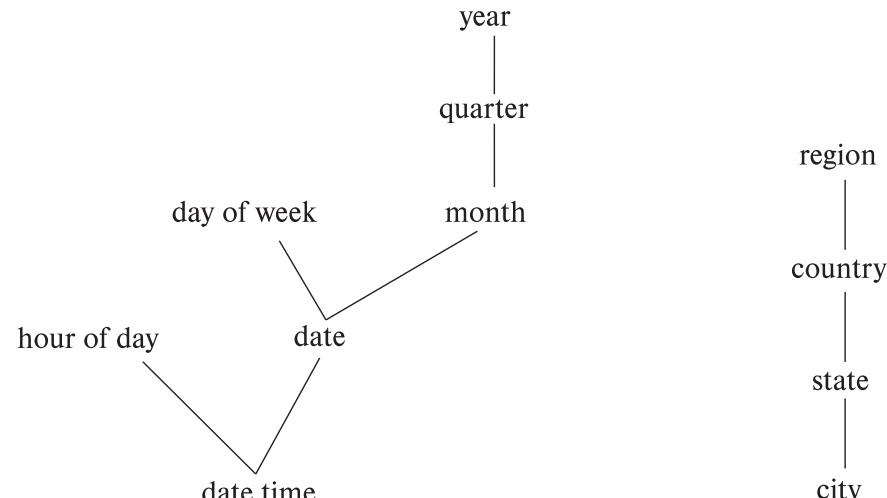
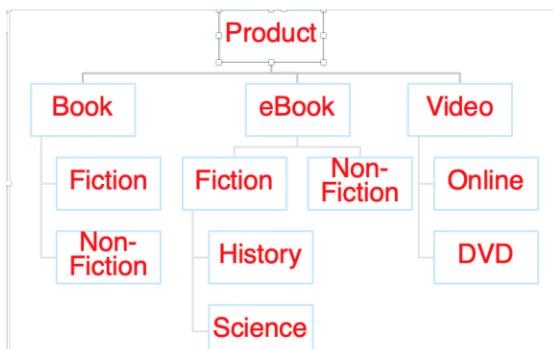


Hierarchies on Dimensions

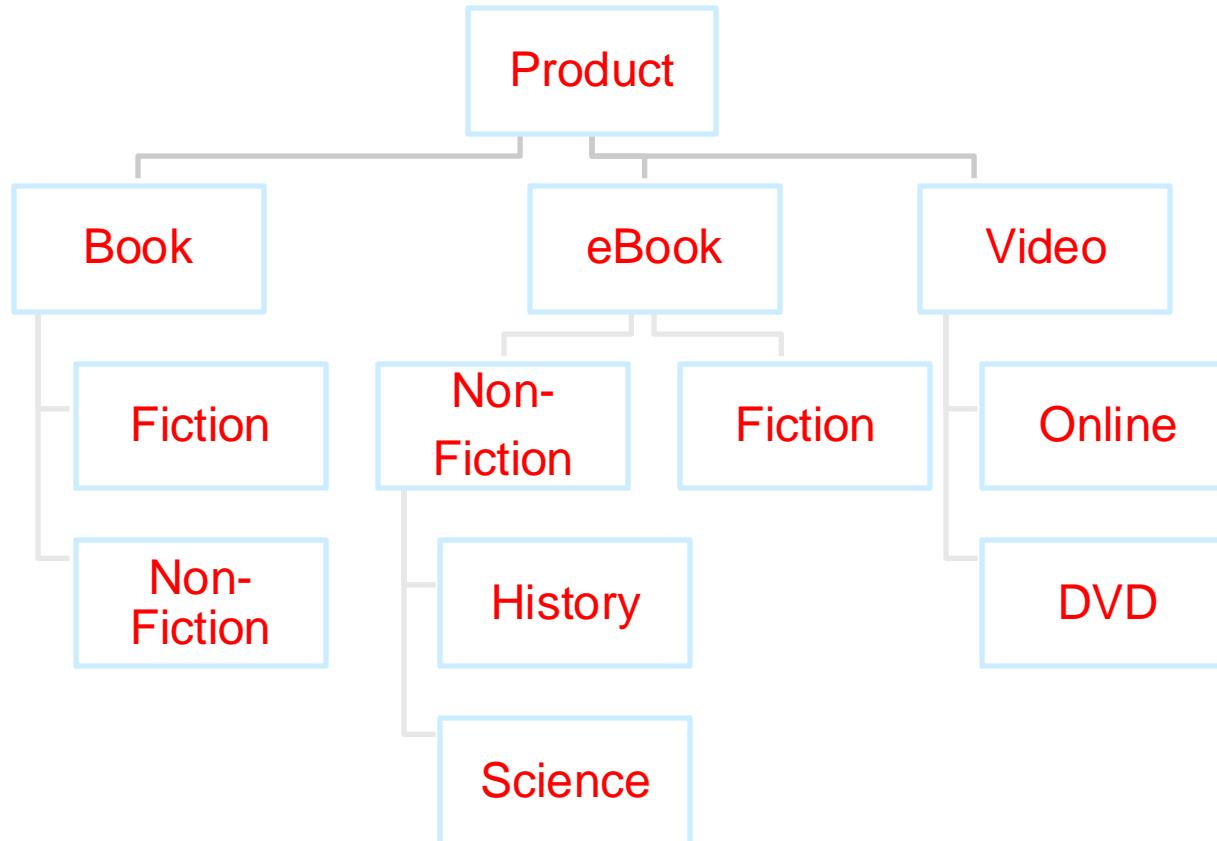
- **Hierarchy** on dimension attributes: lets dimensions be viewed at different levels of detail
- E.g., the dimension *datetime* can be used to aggregate by hour of day, date, day of week, month, quarter or year

Another dimension could be ...

Product category.

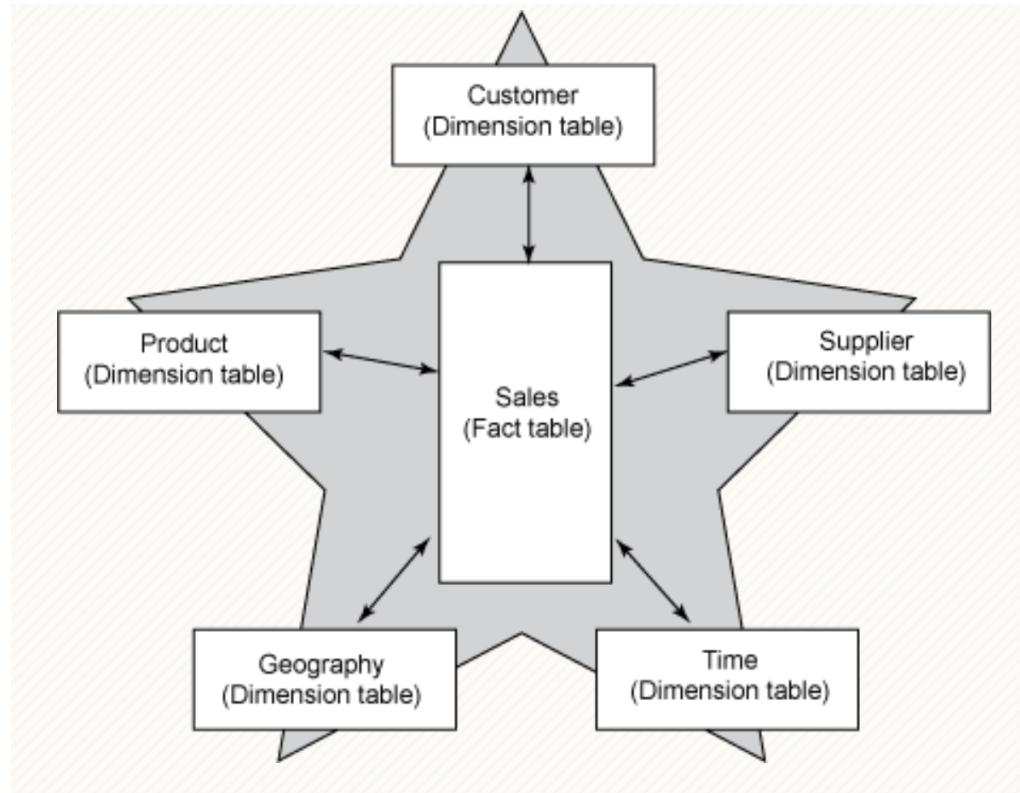


Another Dimension Example – Product Categories





Facts and Dimensions

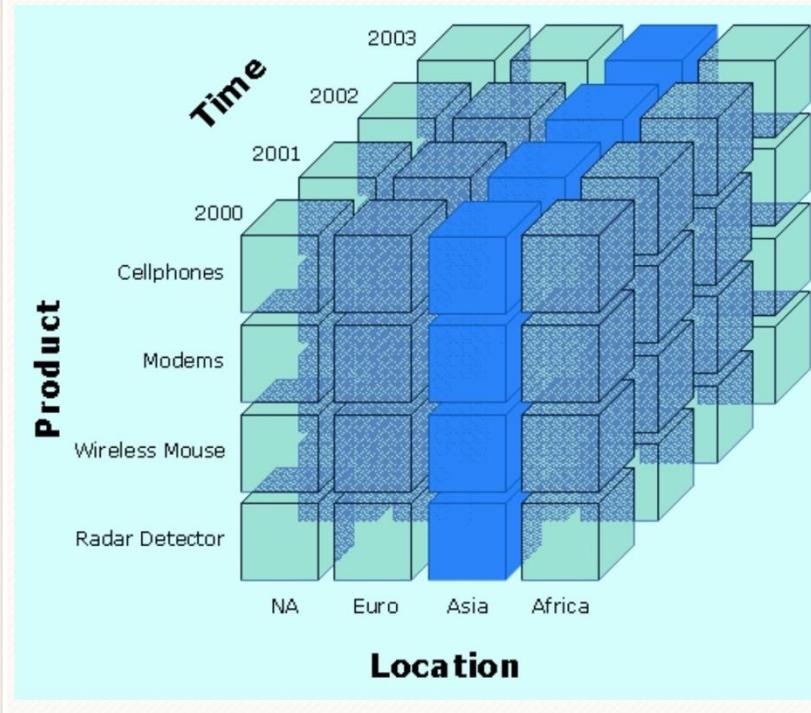




Slice

Slice:

A slice is a subset of a multi-dimensional array corresponding to a single value for one or more members of the dimensions not in the subset.

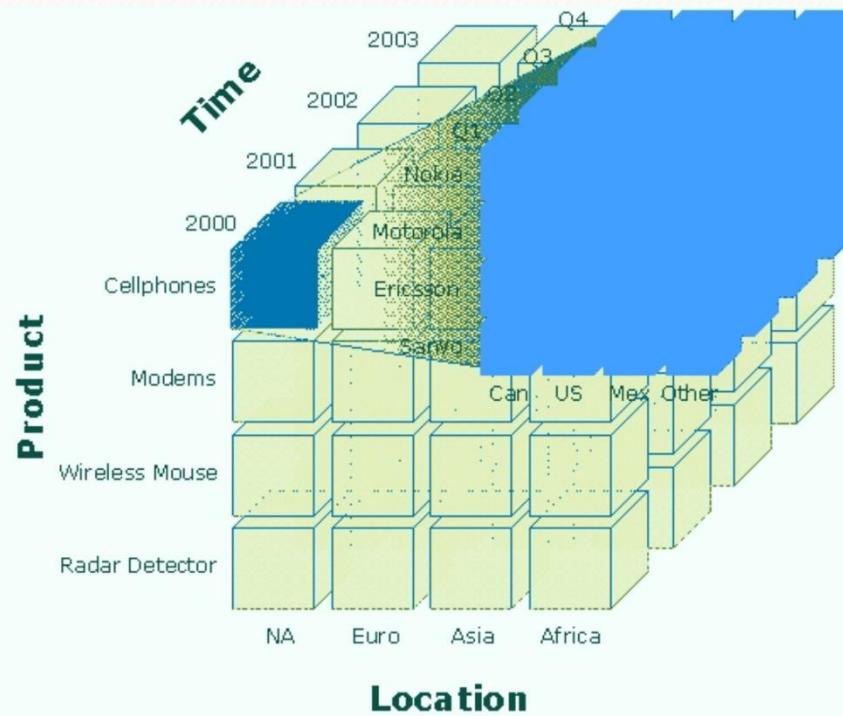




Dice

Dice:

The dice operation is a slice on more than two dimensions of a data cube (or more than two consecutive slices).

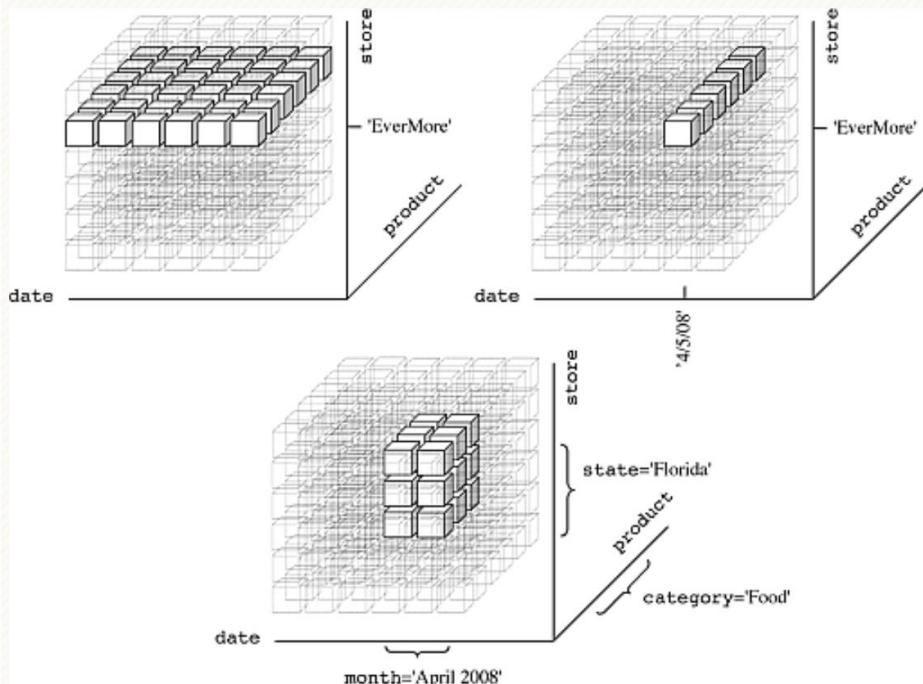




Drilling

Drill Down/Up:

Drilling down or up is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down).

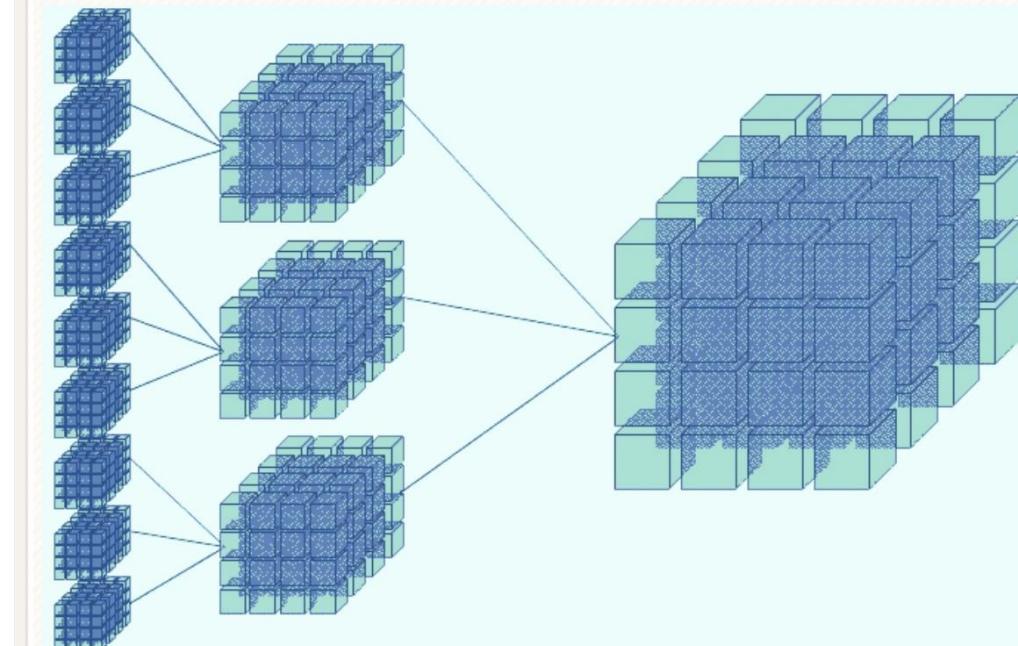




Roll-up

Roll-up:

(Aggregate, Consolidate) A roll-up involves computing all of the data relationships for one or more dimensions. To do this, a computational relationship or formula might be defined.

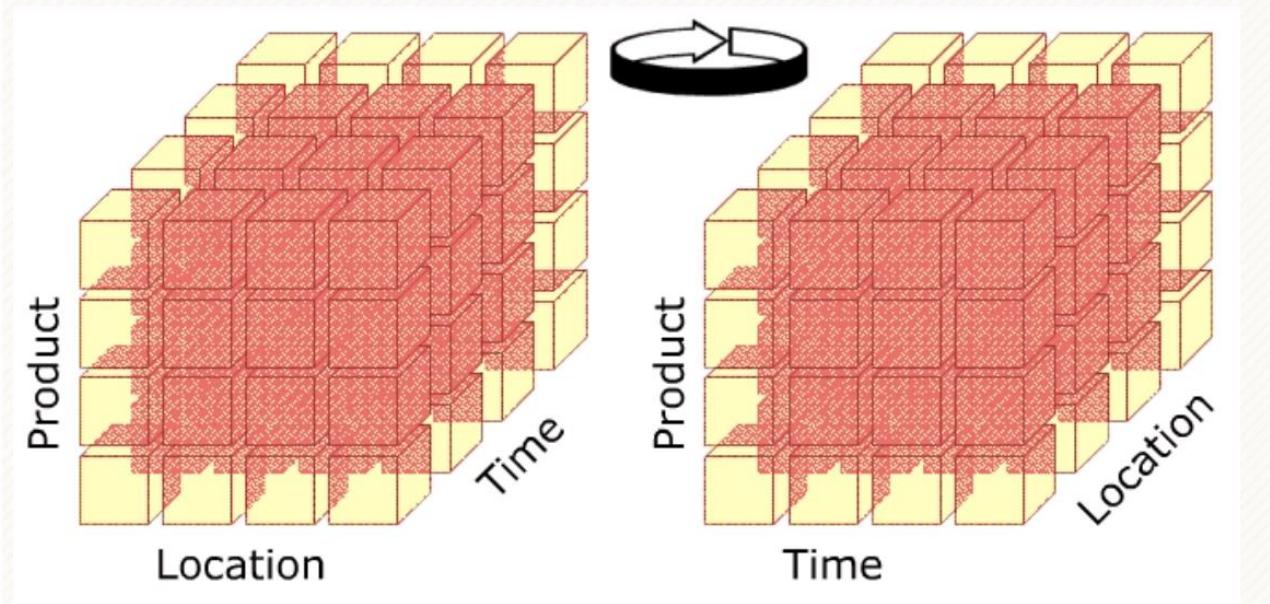




Pivot

Pivot:

This operation is also called rotate operation. It rotates the data in order to provide an alternative presentation of data – the report or page display takes a different dimensional orientation.





Cross Tabulation With Hierarchy

- Cross-tabs can be easily extended to deal with hierarchies
- Can drill down or roll up on a hierarchy
- E.g. hierarchy: *item_name* → *category*

clothes_size: all

| | <i>category</i> | <i>item_name</i> | <i>color</i> | | | |
|------------|-----------------|------------------|--------------|--------|-------|-------|
| | | | dark | pastel | white | total |
| womenswear | skirt | 8 | 8 | 10 | 53 | 88 |
| | dress | 20 | 20 | 5 | 35 | |
| | subtotal | 28 | 28 | 15 | | |
| menswear | pants | 14 | 14 | 28 | 49 | 76 |
| | shirt | 20 | 20 | 5 | 27 | |
| | subtotal | 34 | 34 | 33 | | |
| total | | 62 | 62 | 48 | | 164 |



Relational Representation of Cross-tabs

- Cross-tabs can be represented as relations
- We use the value **all** to represent aggregates.
- The SQL standard actually uses *null* values in place of **all**
 - Works with any data type
 - But can cause confusion with regular null values.

| item_name | color | clothes_size | quantity |
|-----------|--------|--------------|----------|
| skirt | dark | all | 8 |
| skirt | pastel | all | 35 |
| skirt | white | all | 10 |
| skirt | all | all | 53 |
| dress | dark | all | 20 |
| dress | pastel | all | 10 |
| dress | white | all | 5 |
| dress | all | all | 35 |
| shirt | dark | all | 14 |
| shirt | pastel | all | 7 |
| shirt | white | all | 28 |
| shirt | all | all | 49 |
| pants | dark | all | 20 |
| pants | pastel | all | 2 |
| pants | white | all | 5 |
| pants | all | all | 27 |
| all | dark | all | 62 |
| all | pastel | all | 54 |
| all | white | all | 48 |
| all | all | all | 164 |



DATA MINING



Data Mining

- **Data mining** is the process of semi-automatically analyzing large databases to find useful patterns
 - Similar goals to machine learning, but on very large volumes of data
- Part of the larger area of **knowledge discovery in databases (KDD)**
- Some types of knowledge can be represented as rules
- More generally, knowledge is discovered by applying machine learning techniques on past instances of data, to form a **model**
 - Model is then used to make predictions for new instances



Types of Data Mining Tasks

- **Prediction** based on past history
 - Predict if a credit card applicant poses a good credit risk, based on some attributes (income, job type, age, ..) and past history
 - Predict if a pattern of phone calling card usage is likely to be fraudulent
- Some examples of prediction mechanisms:
 - **Classification**
 - Items (with associated attributes) belong to one of several classes
 - **Training instances** have attribute values and classes provided
 - Given a new item whose class is unknown, predict to which class it belongs based on its attribute values
 - **Regression** formulae
 - Given a set of mappings for an unknown function, predict the function result for a new parameter value



Data Mining (Cont.)

- **Descriptive Patterns**

- **Associations**

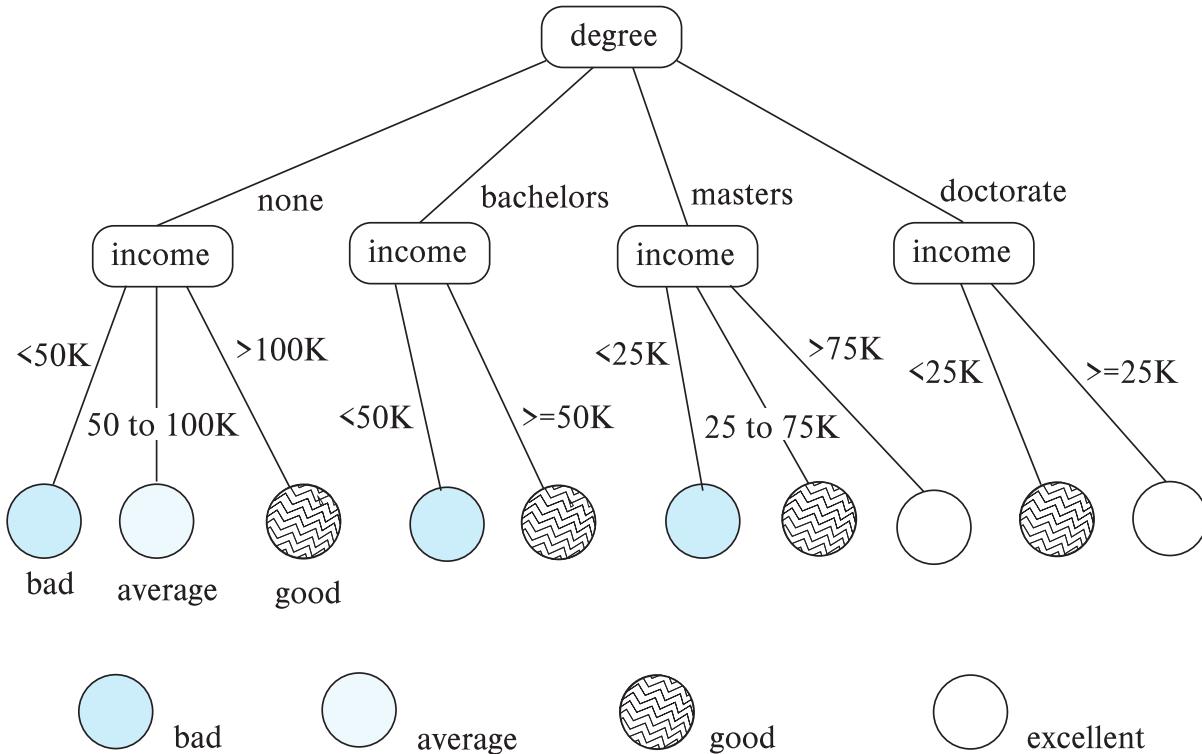
- Find books that are often bought by “similar” customers. If a new such customer buys one such book, suggest the others too.
 - Associations may be used as a first step in detecting **causation**
 - E.g., association between exposure to chemical X and cancer,

- **Clusters**

- E.g., typhoid cases were clustered in an area surrounding a contaminated well
 - Detection of clusters remains important in detecting epidemics



Decision Tree Classifiers





Decision Trees

- Each internal node of the tree partitions the data into groups based on a **partitioning attribute**, and a **partitioning condition** for the node
- Leaf node:
 - all (or most) of the items at the node belong to the same class, or
 - all attributes have been considered, and no further partitioning is possible.
- Traverse tree from top to make a prediction
- Number of techniques for constructing decision tree classifiers
 - We omit details



Bayesian Classifiers

- Bayesian classifiers use **Bayes theorem**, which says

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}$$

where

$p(c_j | d)$ = probability of instance d being in class c_j ,

$p(d | c_j)$ = probability of generating instance d given class c_j ,

$p(c_j)$ = probability of occurrence of class c_j , and

$p(d)$ = probability of instance d occurring



Naïve Bayesian Classifiers

- Bayesian classifiers require
 - computation of $p(d | c_j)$
 - precomputation of $p(c_j)$
 - $p(d)$ can be ignored since it is the same for all classes
- To simplify the task, **naïve Bayesian classifiers** assume attributes have independent distributions, and thereby estimate

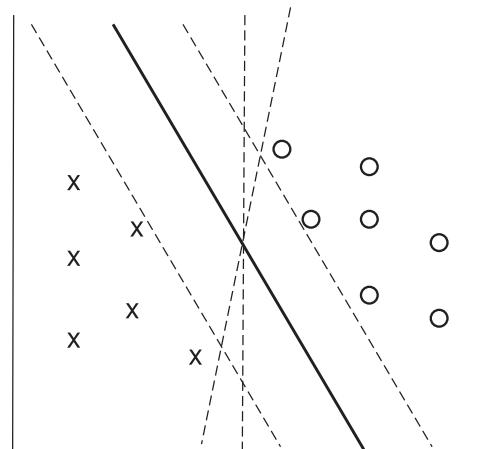
$$p(d | c_j) = p(d_1 | c_j) * p(d_2 | c_j) * \dots * (p(d_n | c_j))$$

- Each of the $p(d_i | c_j)$ can be estimated from a histogram on d_i values for each class c_j
 - the histogram is computed from the training instances
- Histograms on multiple attributes are more expensive to compute and store



Support Vector Machine Classifiers

- Simple 2-dimensional example:
 - Points are in two classes
 - Find a line (**maximum margin line**) s.t. line divides two classes, and distance from nearest point in either class is maximum





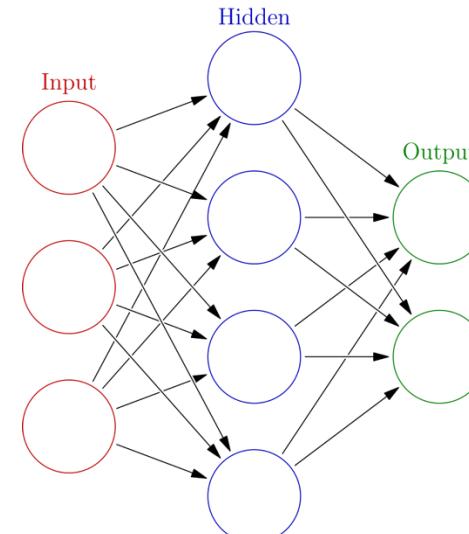
Support Vector Machine

- In n-dimensions points are divided by a plane, instead of a line
- SVMs can be used separators that are curve, not necessarily linear, by transforming points before classification
 - Transformation functions may be non-linear and are called kernel functions
 - Separator is a plane in the transformed space, but maps to curve in original space
- There may not be an exact planar separator for a given set of points
 - Choose plane that best separates points
- N-ary classification can be done by N binary classifications
 - In class i vs. not in class i .



Neural Network Classifiers

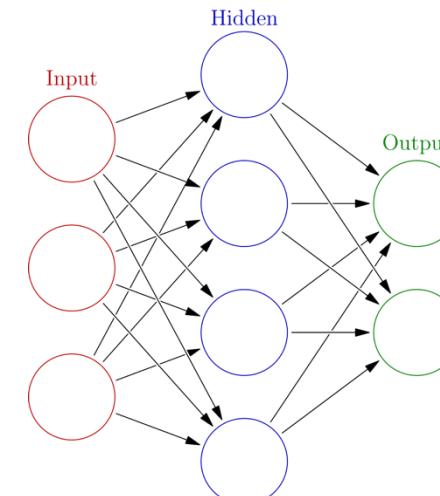
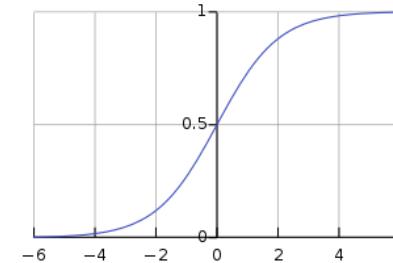
- Neural network has multiple layers
 - Each layer acts as input to next later
- First layer has input nodes, which are assigned values from input attributes
- Each node combines values of its inputs using some weight function to compute its value
 - Weights are associated with edges
- For classification, each output value indicates likelihood of the input instance belonging to that class
 - Pick class with maximum likelihood
- Weights of edges are key to classification
- Edge weights are learnt during training phase





Neural Network Classifiers

- Value of a node may be linear combination of inputs, or may be a non-linear function
 - E.g., sigmoid function
- **Backpropagation algorithm** works as follows
 - Weights are set randomly initially
 - Training instances are processed one at a time
 - Output is computed using current weights
 - If classification is wrong, weights are tweaked to get a higher score for the correct class





Neural Networks (Cont.)

- **Deep neural networks** have a large number of layers with large number of nodes in each layer
- **Deep learning** refers to training of deep neural network on very large numbers of training instances
- Each layer may be connected to previous layers in different ways
 - Convolutional networks used for image processing
 - More complex architectures used for text processing, and machine translation, speech recognition, etc.
- Neural networks are a large area in themselves
 - Further details beyond scope of this chapter



Regression

- Regression deals with the prediction of a value, rather than a class.
 - Given values for a set of variables, X_1, X_2, \dots, X_n , we wish to predict the value of a variable Y .
- One way is to infer coefficients $a_0, a_1, a_2, \dots, a_n$ such that
$$Y = a_0 + a_1 * X_1 + a_2 * X_2 + \dots + a_n * X_n$$
- Finding such a linear polynomial is called **linear regression**.
 - In general, the process of finding a curve that fits the data is also called **curve fitting**.
- The fit may only be approximate
 - because of noise in the data, or
 - because the relationship is not exactly a polynomial
- Regression aims to find coefficients that give the best possible fit.



Association Rules

- Retail shops are often interested in associations between different items that people buy.
 - Someone who buys bread is quite likely also to buy milk
 - A person who bought the book *Database System Concepts* is quite likely also to buy the book *Operating System Concepts*.
- Associations information can be used in several ways.
 - E.g. when a customer buys a particular book, an online shop may suggest associated books.
- **Association rules:**

$bread \Rightarrow milk$ DB-Concepts, OS-Concepts \Rightarrow Networks

- Left hand side: **antecedent**, right hand side: **consequent**
- An association rule must have an associated **population**; the population consists of a set of **instances**
 - E.g. each transaction (sale) at a shop is an instance, and the set of all transactions is the population



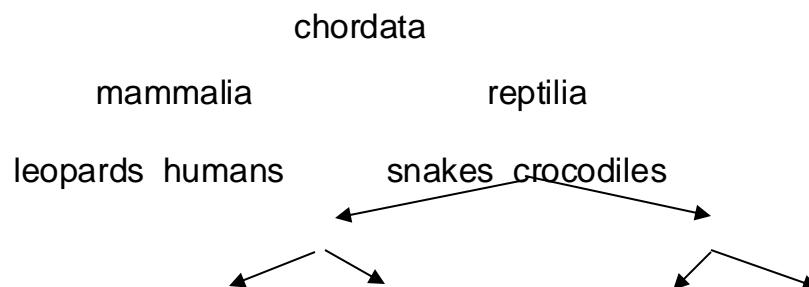
Association Rules (Cont.)

- Rules have an associated support, as well as an associated confidence.
- **Support** is a measure of what fraction of the population satisfies both the antecedent and the consequent of the rule.
 - E.g., suppose only 0.001 percent of all purchases include milk and screwdrivers. The support for the rule $milk \Rightarrow screwdrivers$ is low.
- **Confidence** is a measure of how often the consequent is true when the antecedent is true.
 - E.g., the rule $bread \Rightarrow milk$ has a confidence of 80 percent if 80 percent of the purchases that include bread also include milk.
- We omit further details, such as how to efficiently infer association rules



Clustering

- **Clustering:** Intuitively, finding clusters of points in the given data such that similar points lie in the same cluster
- Can be formalized using distance metrics in several ways
 - Group points into k sets (for a given k) such that the average distance of points from the centroid of their assigned group is minimized
 - Centroid: point defined by taking average of coordinates in each dimension.
 - Another metric: minimize average distance between every pair of points in a cluster
- **Hierarchical clustering:** example from biological classification
 - (the word classification here does not mean a prediction mechanism)





Clustering and Collaborative Filtering

- Goal: predict what movies/books/... a person may be interested in, on the basis of
 - Past preferences of the person
 - Preferences of other people
- One approach based on repeated clustering
 - Cluster people based on their preferences for movies
 - Then cluster movies on the basis of being liked by the same clusters of people
 - Again cluster people based on their preferences for (the newly created clusters of) movies
 - Repeat above till equilibrium
 - Given new user
 - Find most similar cluster of existing users and
 - Predict movies in movie clusters popular with that user cluster
- Above problem is an instance of **collaborative filtering**



Other Types of Mining

- **Text mining:** application of data mining to textual documents
- **Sentiment analysis**
 - E.g., learn to predict if a user review is positive or negative about a product
- **Information extraction**
 - Create structured information from unstructured textual description or semi-structured data such as tabular displays
- **Entity recognition and disambiguation**
 - E.g., given text with name “Michael Jordan” does the name refer to the famous basketball player or the famous ML expert
- **Knowledge graph** (see Section 8.4)
 - Can be constructed by information extraction from different sources, such as Wikipedia

Some Examples

Lecture Notebook

HW 3B

HW 4B Non-Programming