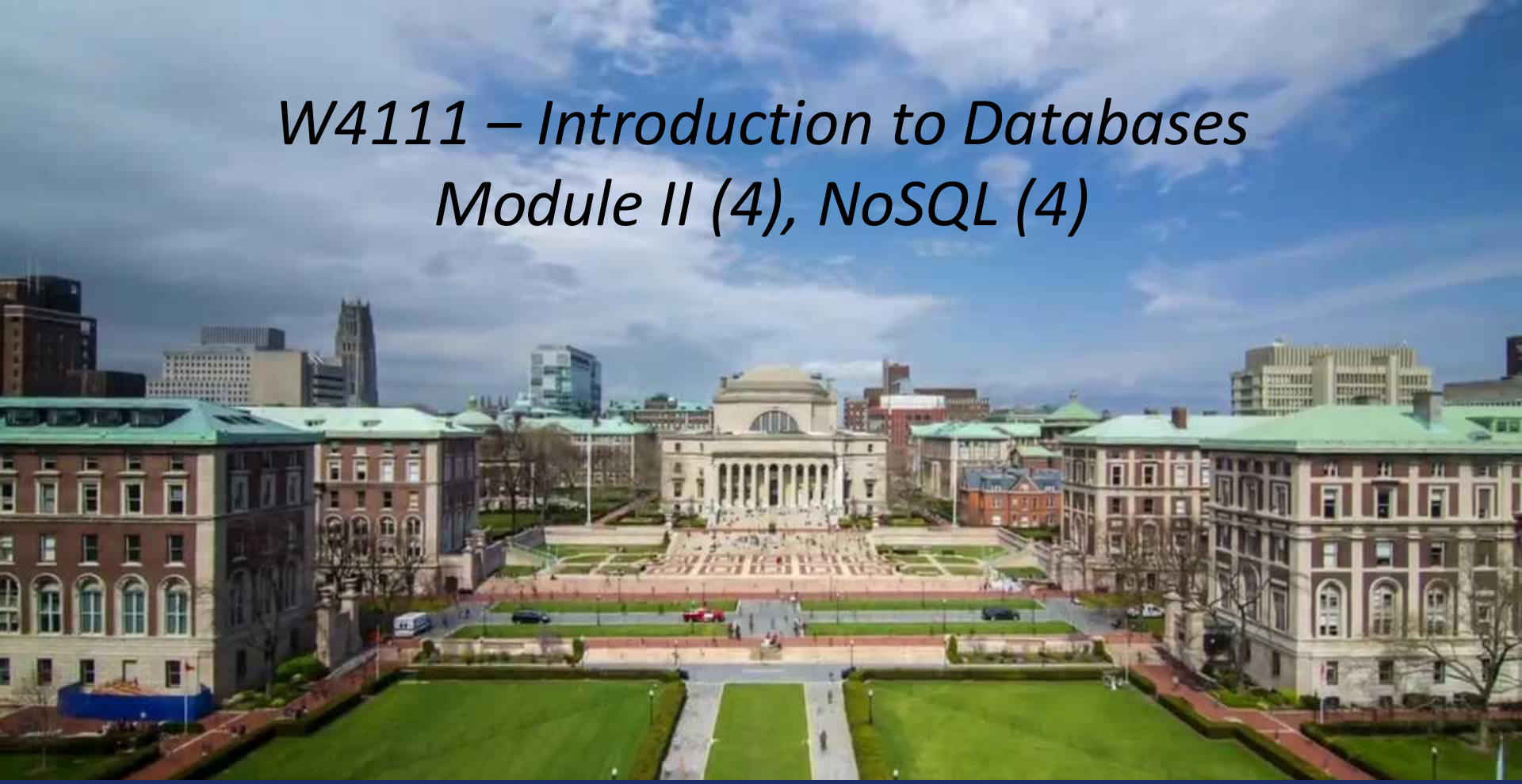
W4111 – Introduction to Databases
Module II (4), NoSQL (4)



COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Contents

Contents

- Course update
- Module IV – EII, ETL/ELT, Data Warehouse, Data Lake reminder
- Star schema – Reminder
- OLAP continued
  - Cubes
  - Operations
- Example of star schema, OLAP, … … in relational
  - Schema definition
  - ETL
  - Some queries
- Big Data and Tools
  - Concepts
  - MapReduce
  - Spark, pySpark and some examples

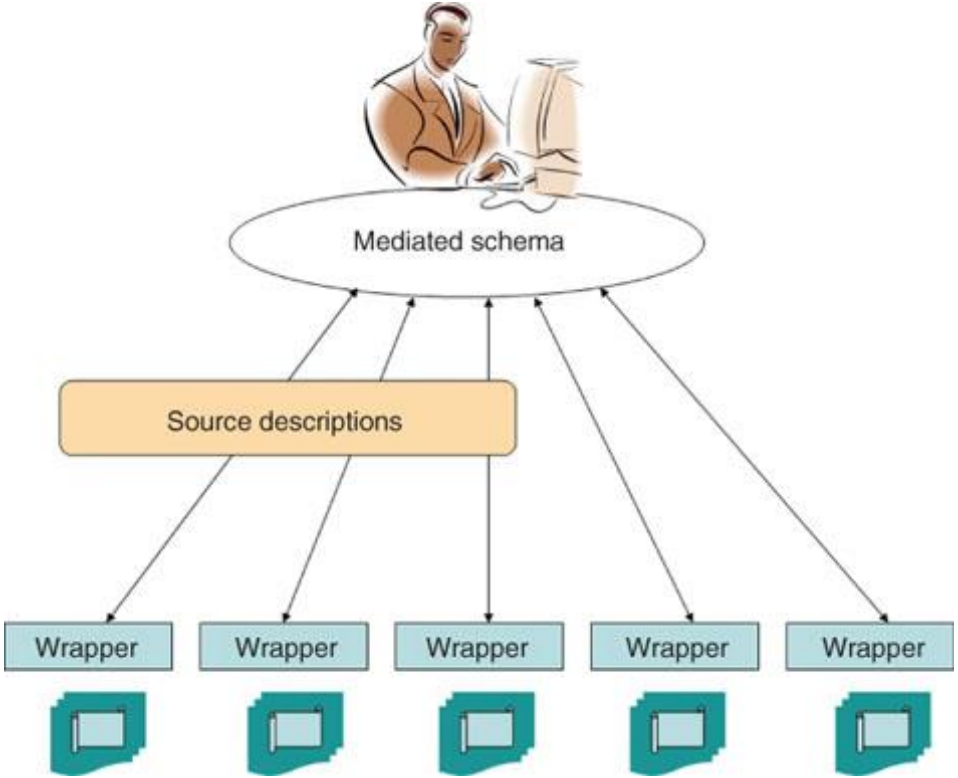
Course Overview

Course Overview

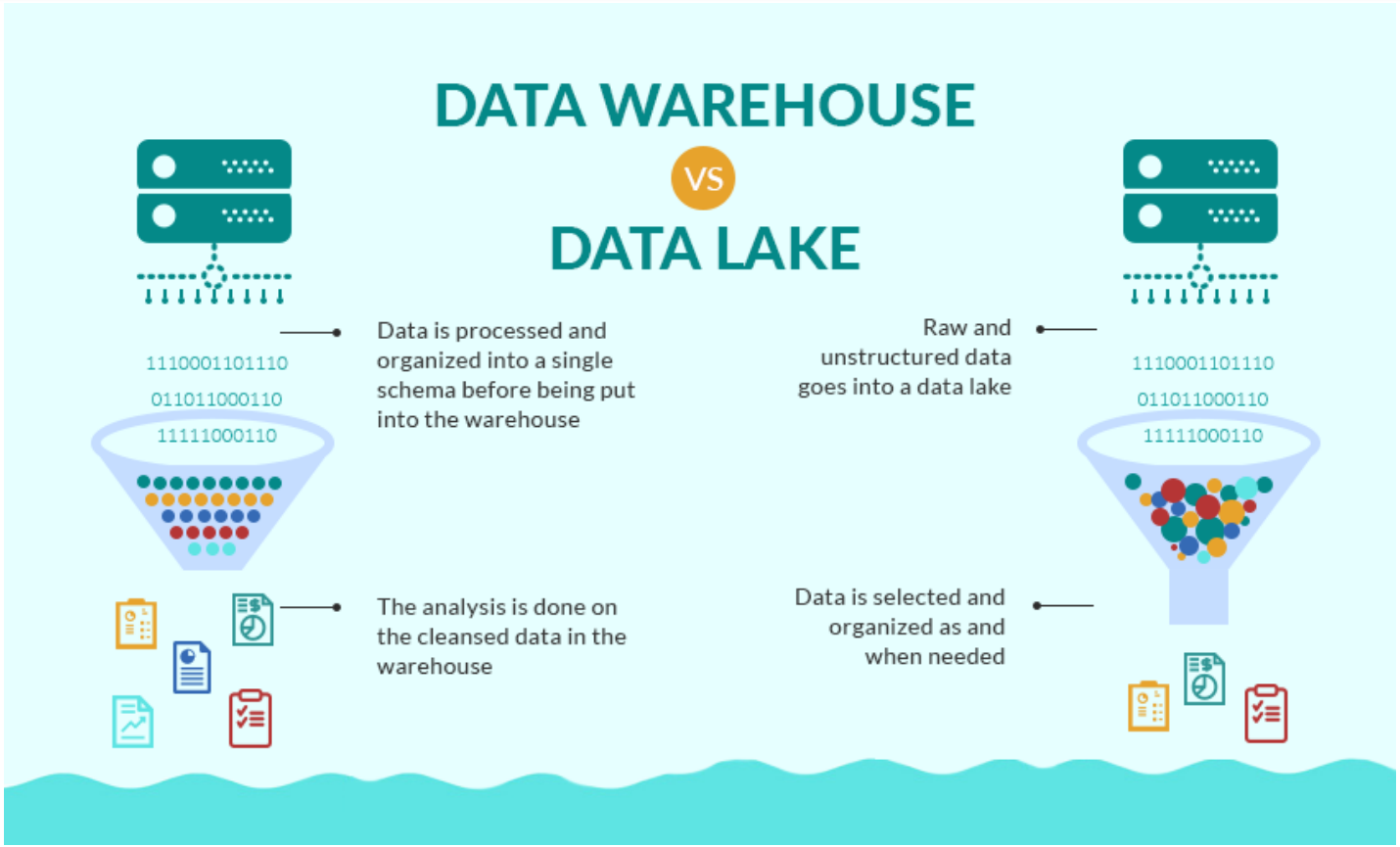
- Final lecture
  - I am in a meeting during lecture time.
  - I will record the lecture, and segments will begin appearing this weekend.
  - It will cover miscellaneous topics and walking through HW 5 project.
- Homework 5
  - HW 5A will be a set of written questions. Due next weekend.
  - HW5B
    - Will be completing the project template.
    - I am in the process of simplifying the requirements and providing additional code.
    - Will be due at the end of the final exam period.
- Final exam
  - Will be released approximately next weekend.
  - There will be two versions:
    - A non-cumulative version
    - An optional set of additional questions that form a cumulative exam
  - You have to pick if you are submitting only the non-cumulative or both parts.

Introduction to
Enterprise Information Integration
ETL
Data Warehouse, Data Lake
Reminder

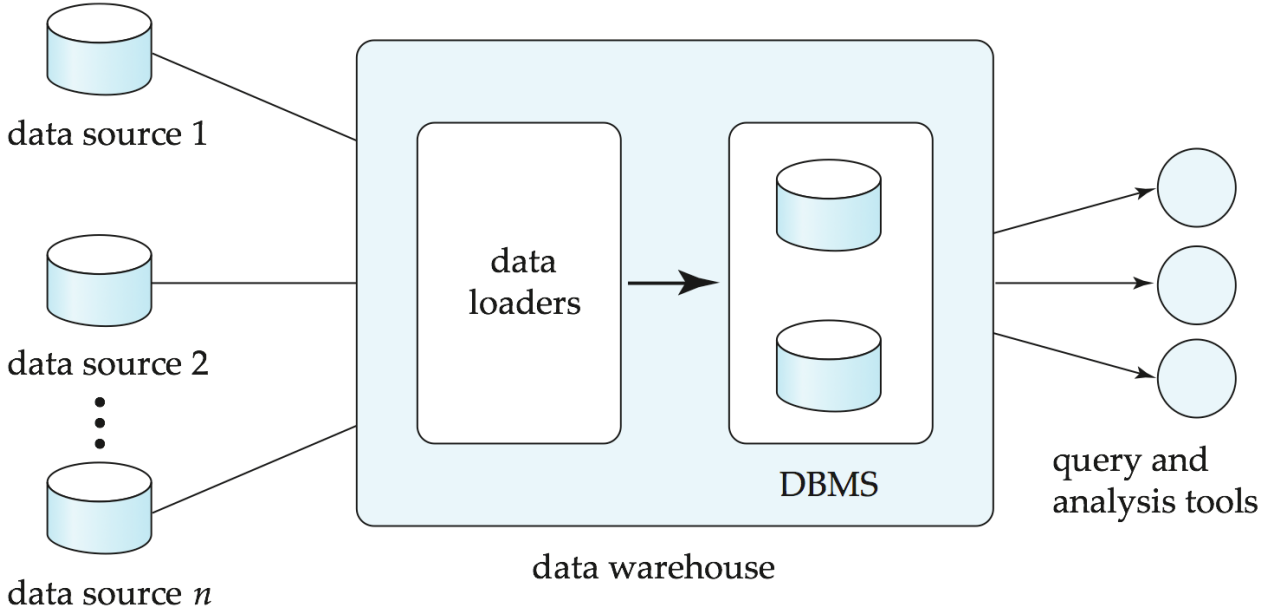
Enterprise Information Integration



Data Warehouse and Data Lake



Data Warehousing

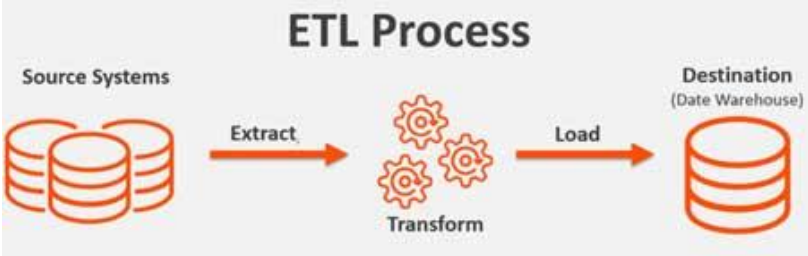


Overview (Cont.)

- 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

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 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

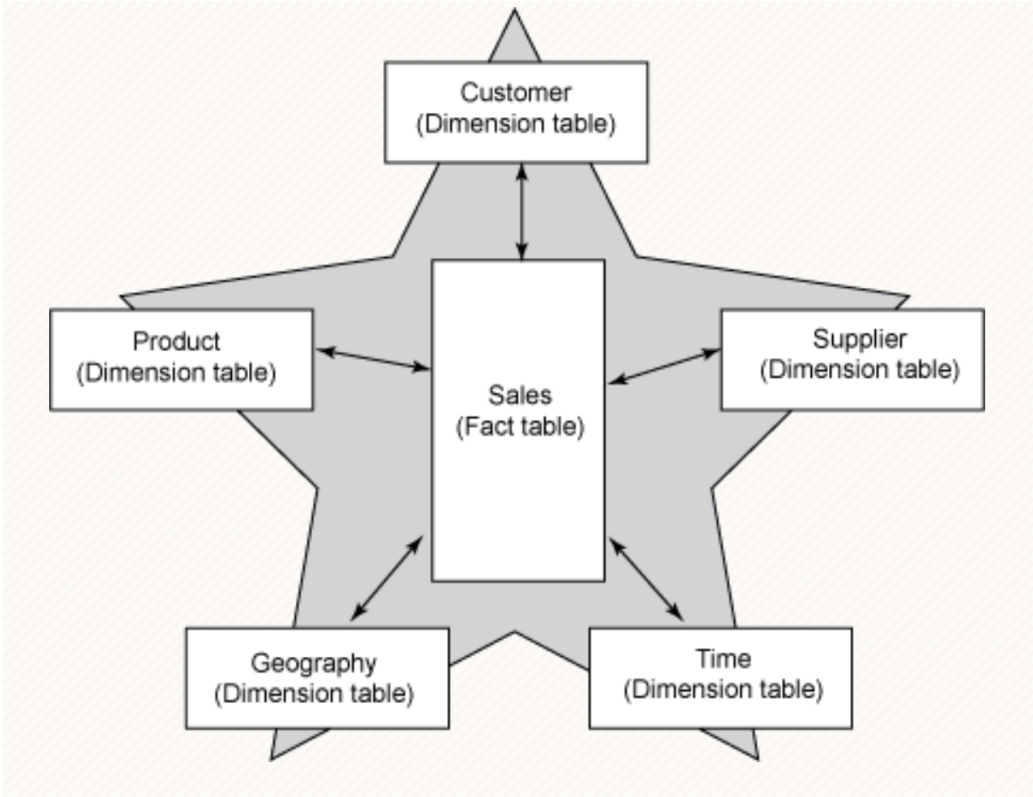
Load

Finally, the load function s 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.

Star Schema

Facts and Dimensions

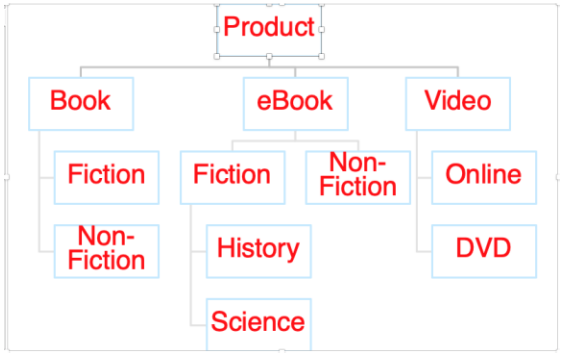
- A common model for (some) data in a date warehouse is a *star schema.*
- There are one or more *fact tables* that have records for small facts, e.g.
  - Sold product A.
  - To customer B.
  - At price C.
- Facts are positioned in dimensions, e.g.
  - Date, time
  - Location
  - Product category
  - … …
- Queries ask for summations and aggregations in dimensions, e.g.
  - Total sales of products in category X, to customers in country Y during year Z.
  - Queries can roll-up, drill down, pivot, … …
- This is a generalization of the spreadsheet concept of *pivot tables.*

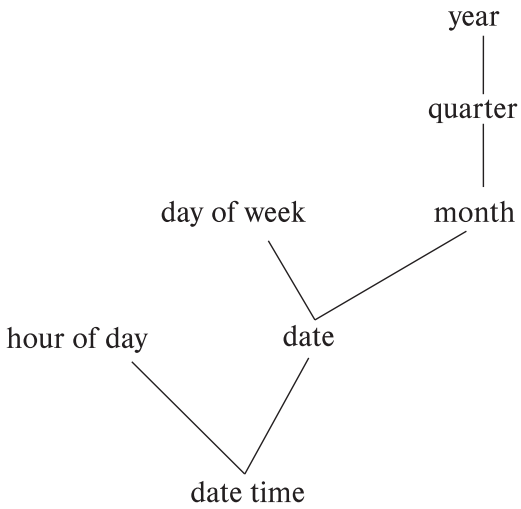


Hierarchies on Dimensions

- **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.

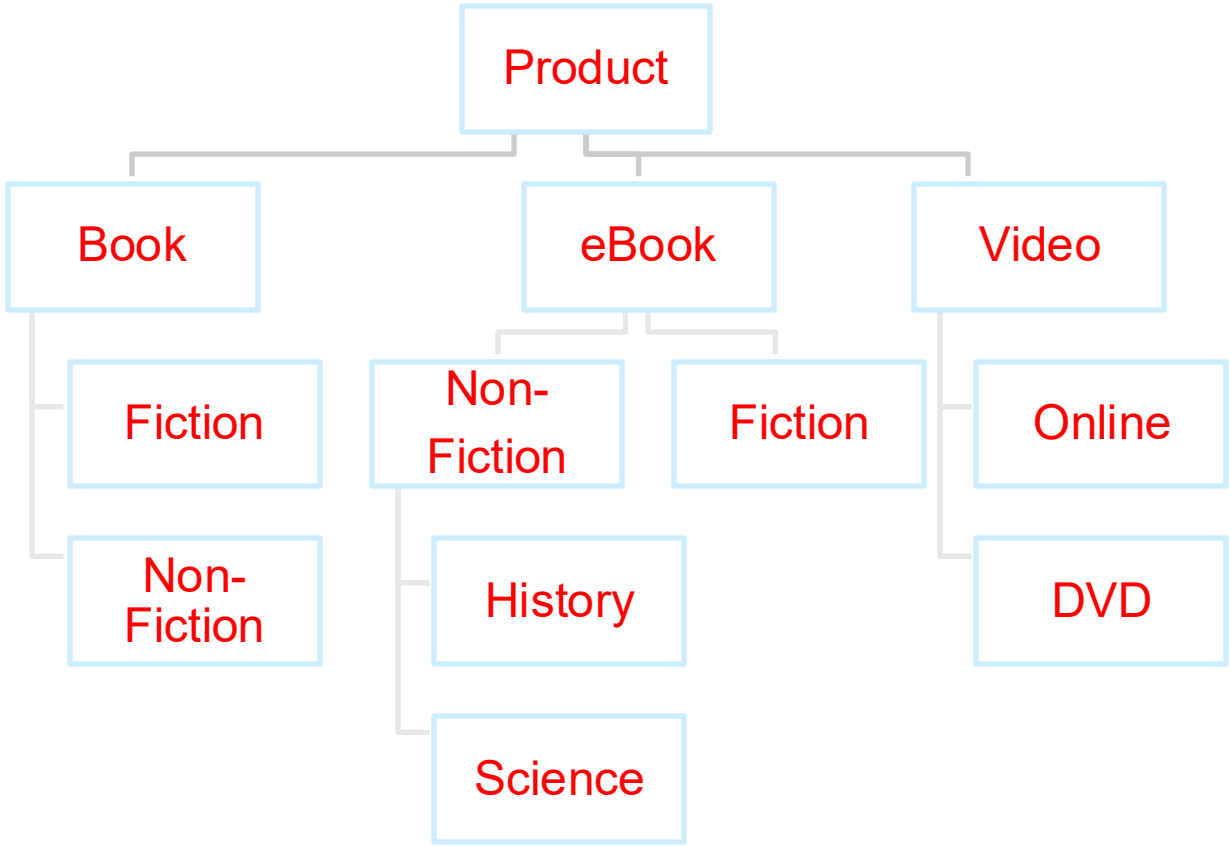




(a) time hierarchy

(b) location hierarchy

Another Dimension Example – Product Categories



OLAP

Data Analysis and OLAP

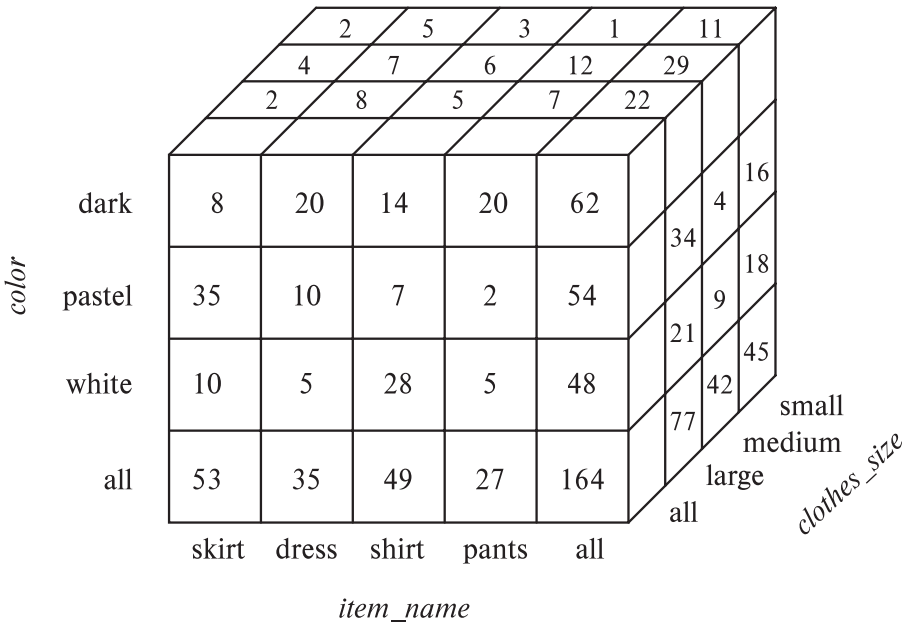
- **Online Analytical Processing (OLAP)**
  - Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)
- We use the following relation to illustrate OLAP concepts
  - *sales* (*item_name*, *color*, *clothes_size*, *quantity*)

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

- For Classic Models, this would be:
  - sales(productCode, quantityOrdered, priceEach)
  - But, I need to link with dimensions.
    - Date
    - Location
    - ProductType
  - ➔
  - sales(productCode, quantityOrdered, priceEach,
    date_dimension_id, location_dimension_id, product_type_dimension_id)
  - We are only going to worry about date and location for now.

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



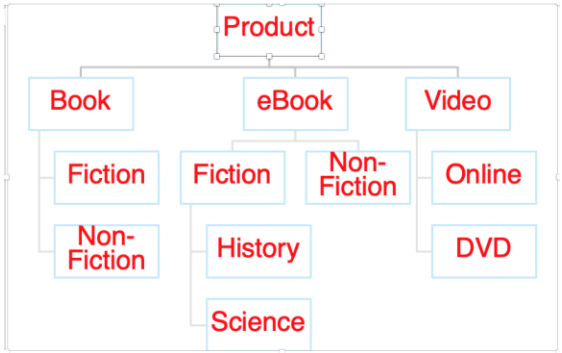
Online Analytical Processing Operations

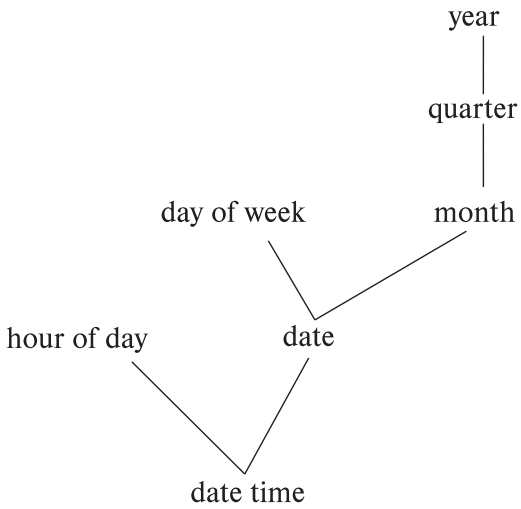
- **Pivoting:** changing the dimensions used in a cross-tab
  - E.g., moving colors to column names
- **Slicing:** creating a cross-tab for fixed values only
  - E.g., fixing color to white and size to small
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **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.

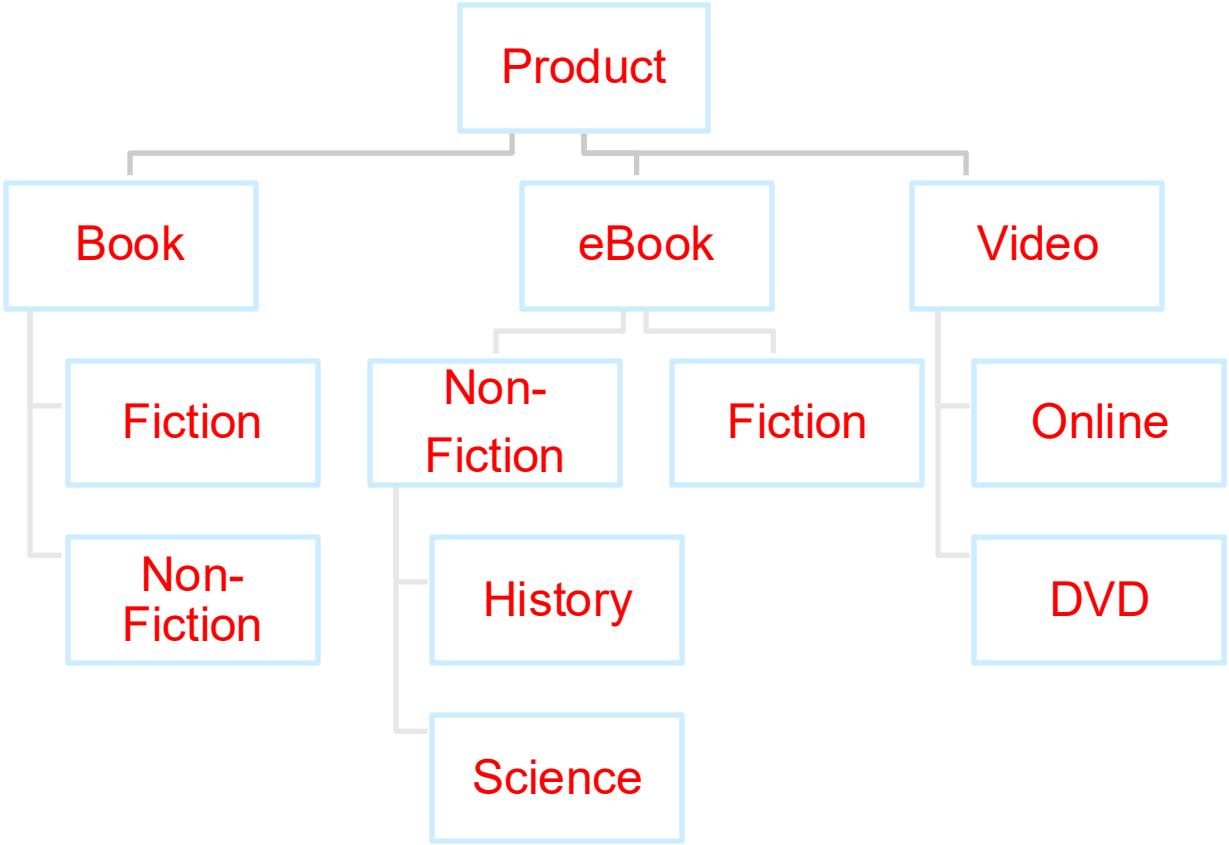




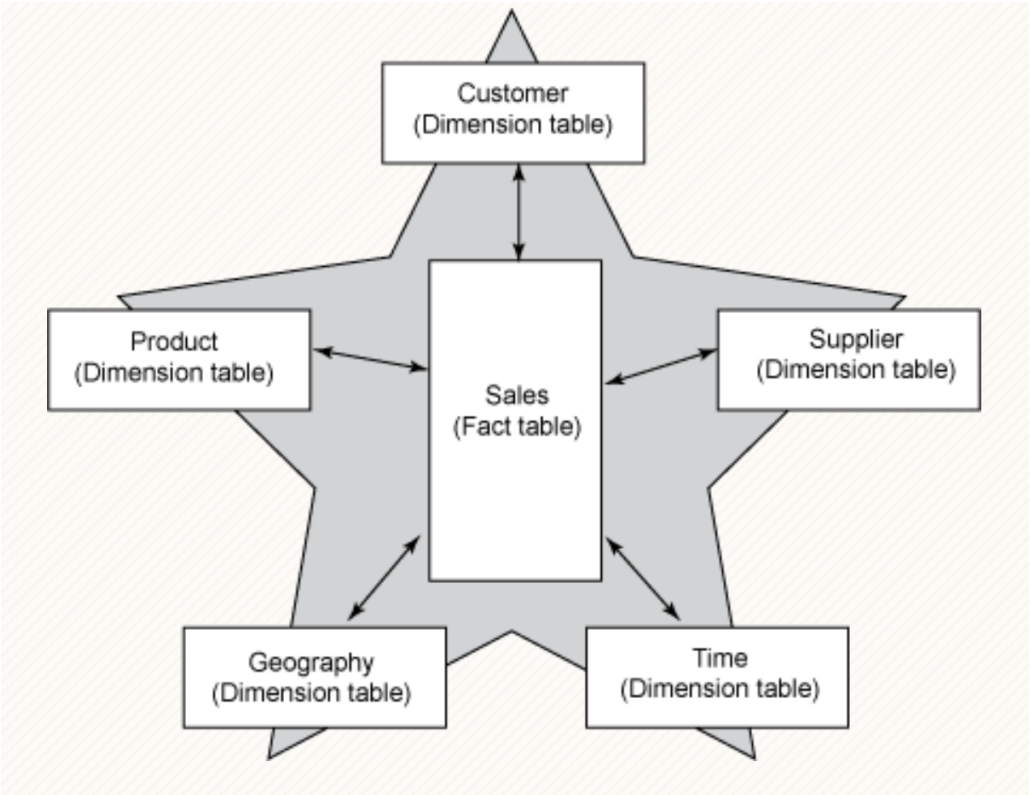
(a) time hierarchy

(b) location hierarchy

Another Dimension Example – Product Categories



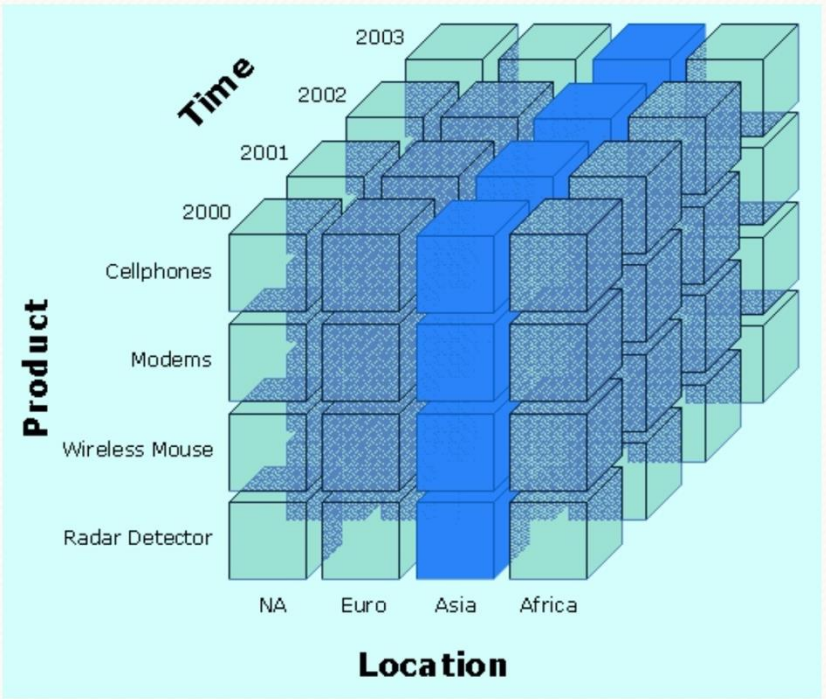
Facts and Dimensions



Slice

Slice:

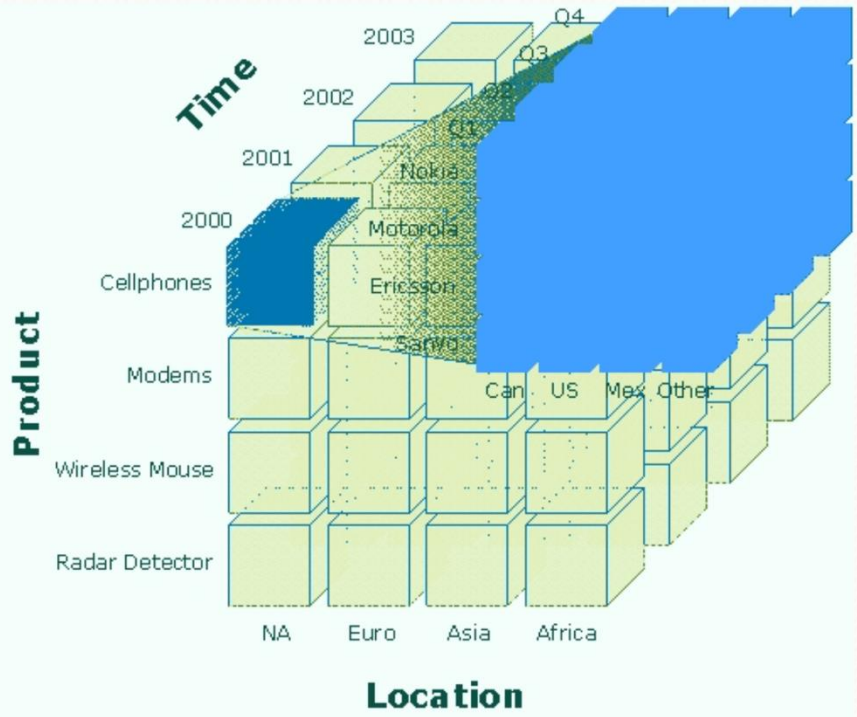
A slice is a subset of a multi-dimensional array corresponding to a single value for one or more members of the dimensions not in the subset.



Dice

Dice:

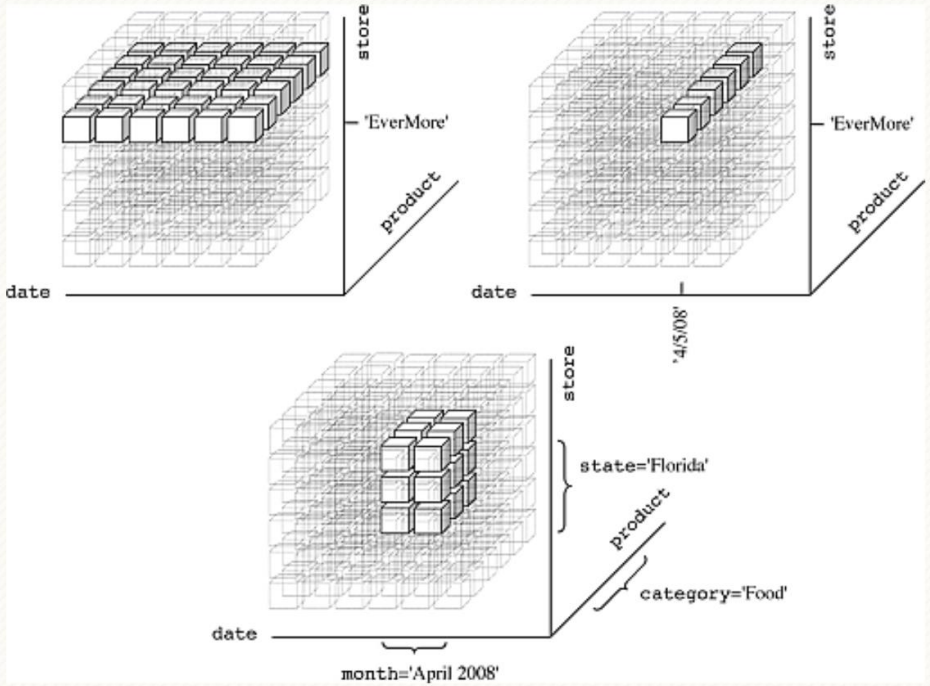
The dice operation is a slice on more than two dimensions of a data cube (or more than two consecutive slices).



Drilling

Drill Down/Up:

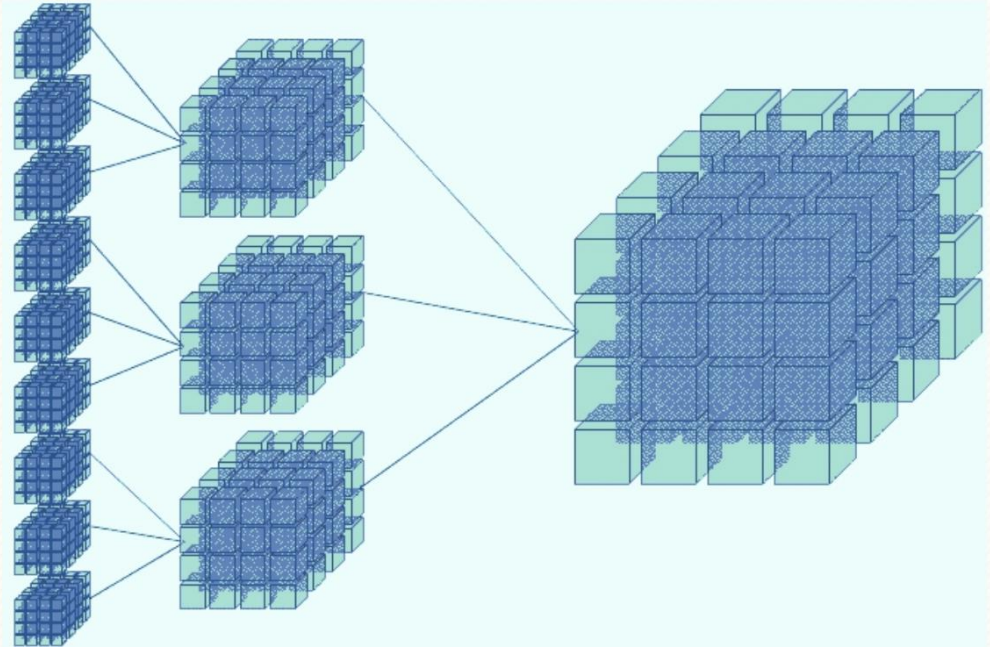
Drilling down or up is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down).



Roll-up

Roll-up:

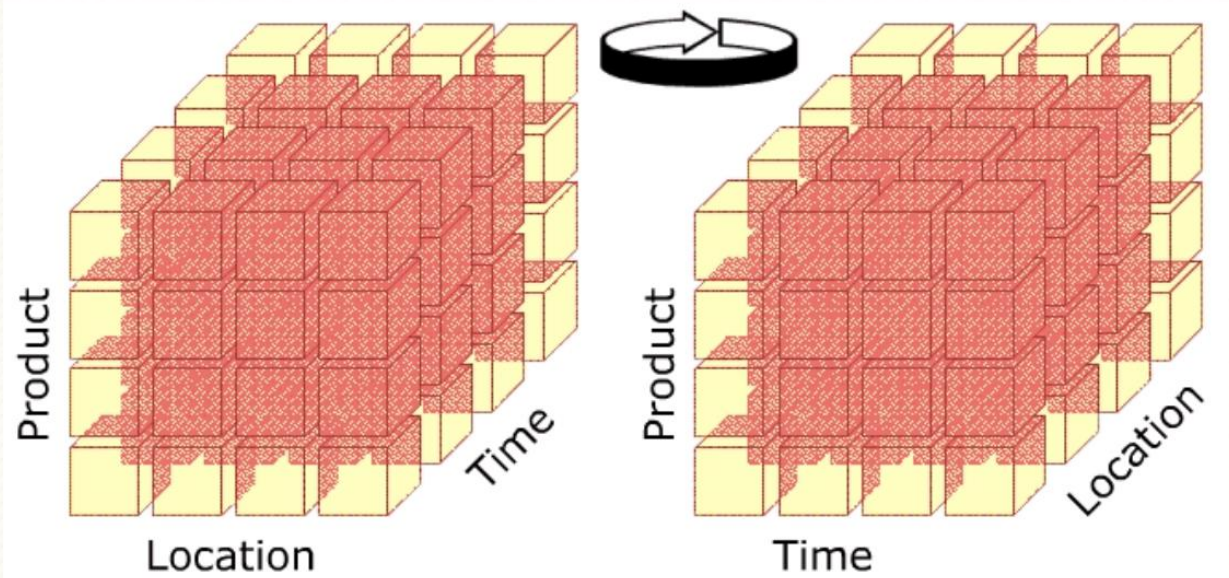
(Aggregate, Consolidate) A roll-up involves computing all of the data relationships for one or more dimensions. To do this, a computational relationship or formula might be defined.



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**

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

Switch to Notebook

- ./W4111-2024S-13-Examples.ipynb

Summary

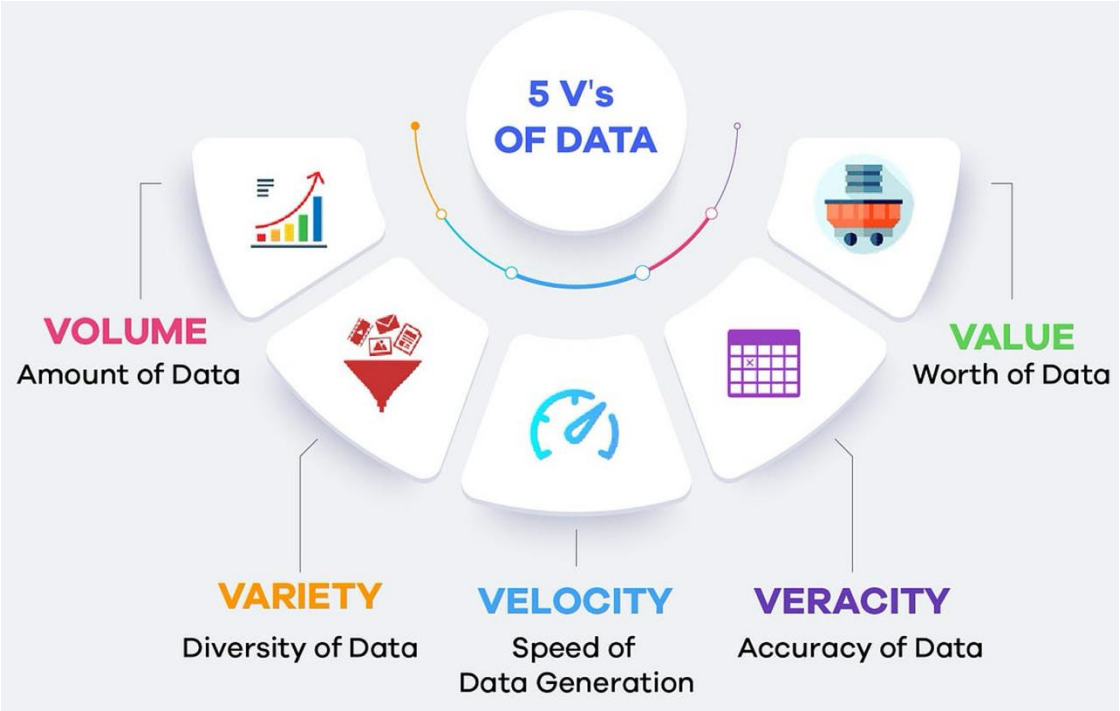
- OLAP is a very useful model for analyzing business data, typically data that is inherently tabular to begin with.
- There are pure OLAP databases, but a more common approach is:
  - Use a relational database.
  - Define a star schema.
  - ETL from the operational data into the star schema.
  - "Translate" the logical OLAP operations into SQL queries and predefined views.
- What is it that you really care about?
  - In the past, I had students build a star schema from Classic Models and do OLAP-like queries. You will not have OLAP on a HW.
  - For the final, you will be responsible for understanding the concepts. For example, it is very common for me to provide some schema/data and ask you to define a star schema and some hypothetical queries.

Big Data
Data Engineering
Data Analysis

Big Data

5 V's of Big Data

https://www.linkedin.com/pulse/big-data-management-revolutio-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.

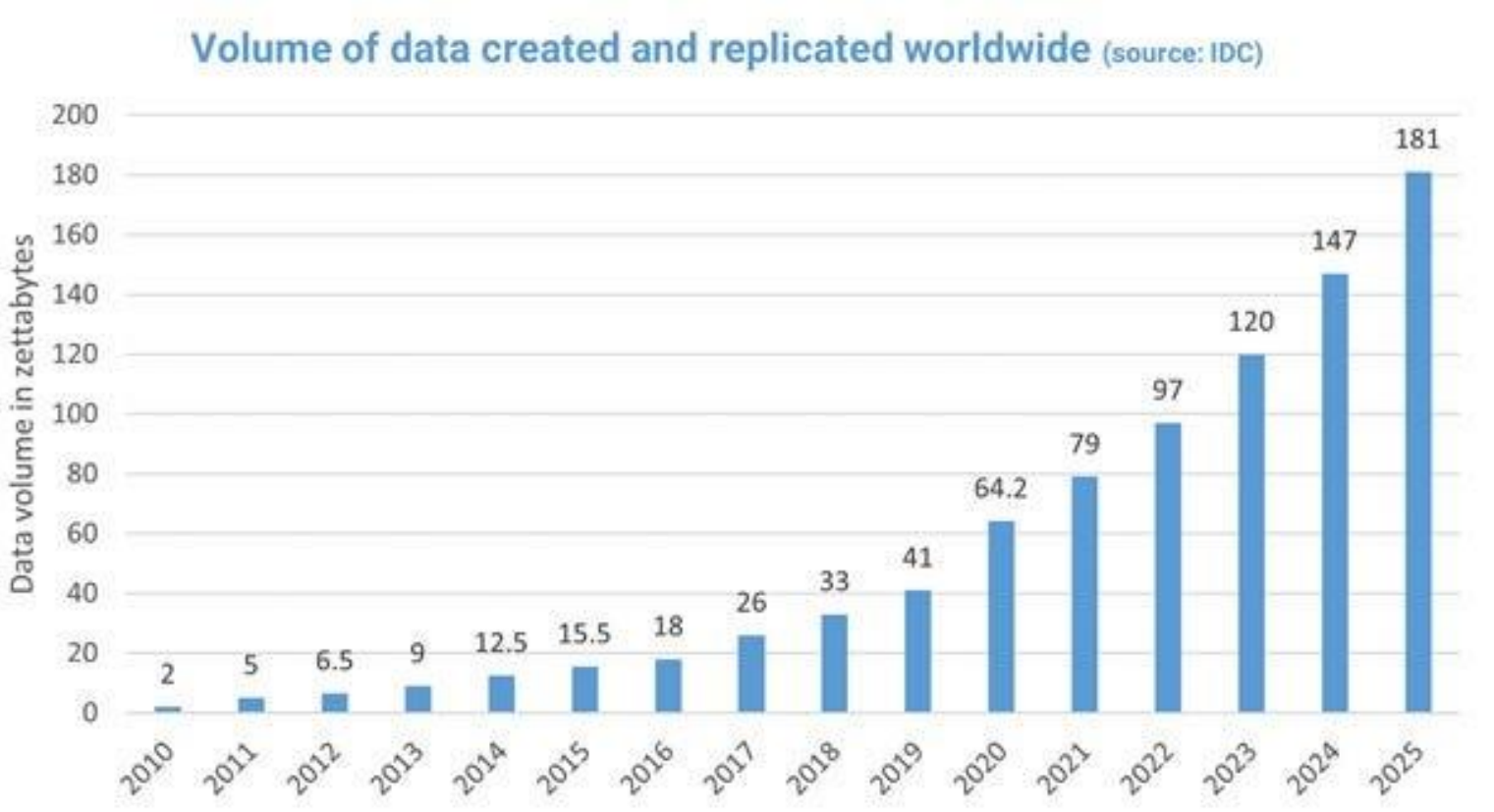
Unstructured

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



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

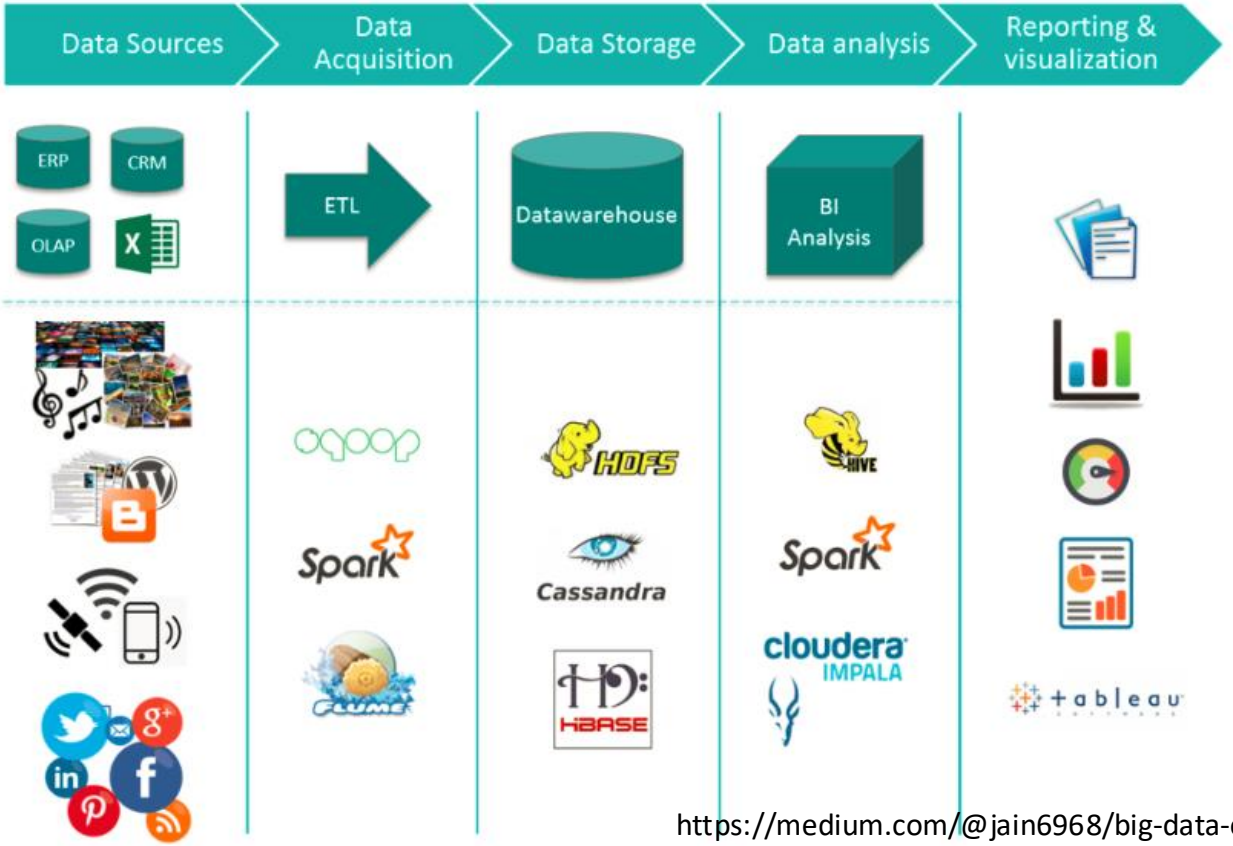
Overview (Cont.)

- 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.)

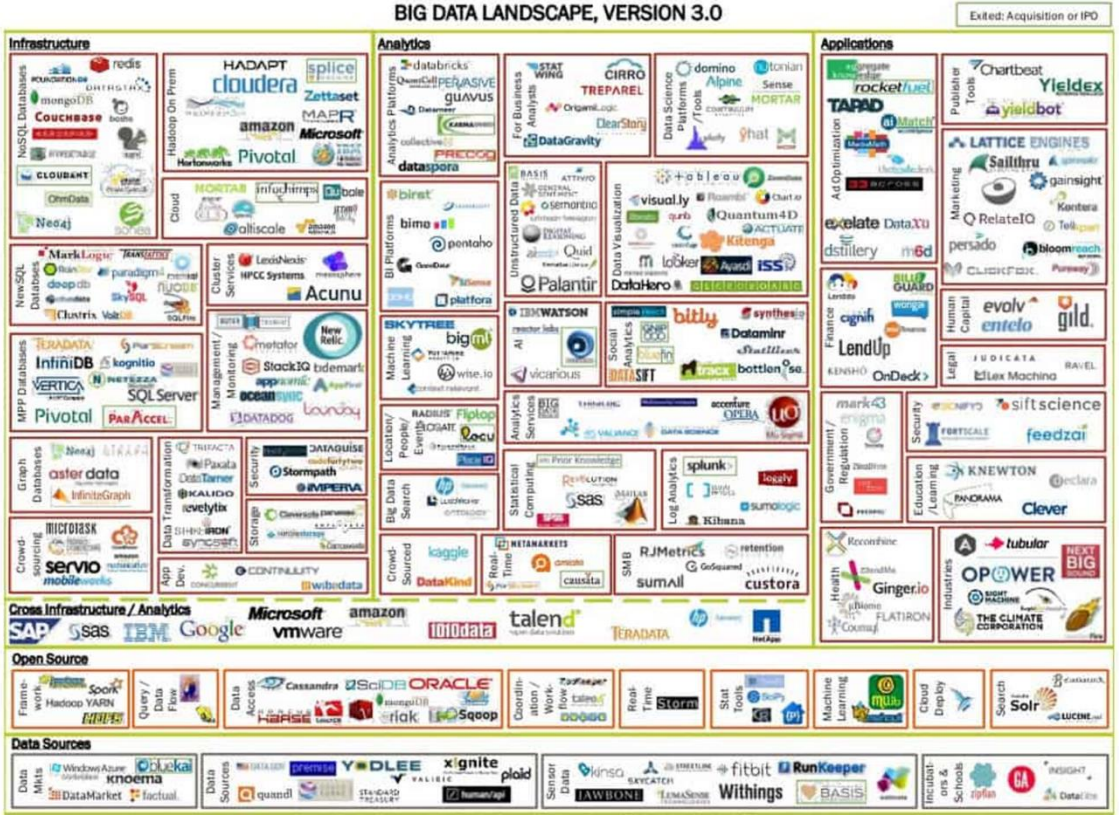
- **Machine learning** techniques are key to finding patterns in data and making predictions
- **Data mining** extends techniques developed by machine-learning communities to run them on very large datasets
- The term **business intelligence (BI)** is synonym for data analytics
- The term **decision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

Big Data Ecosystem/Platform



https://medium.com/@jain6968/big-data-ecosystem-b0e4c923d7aa

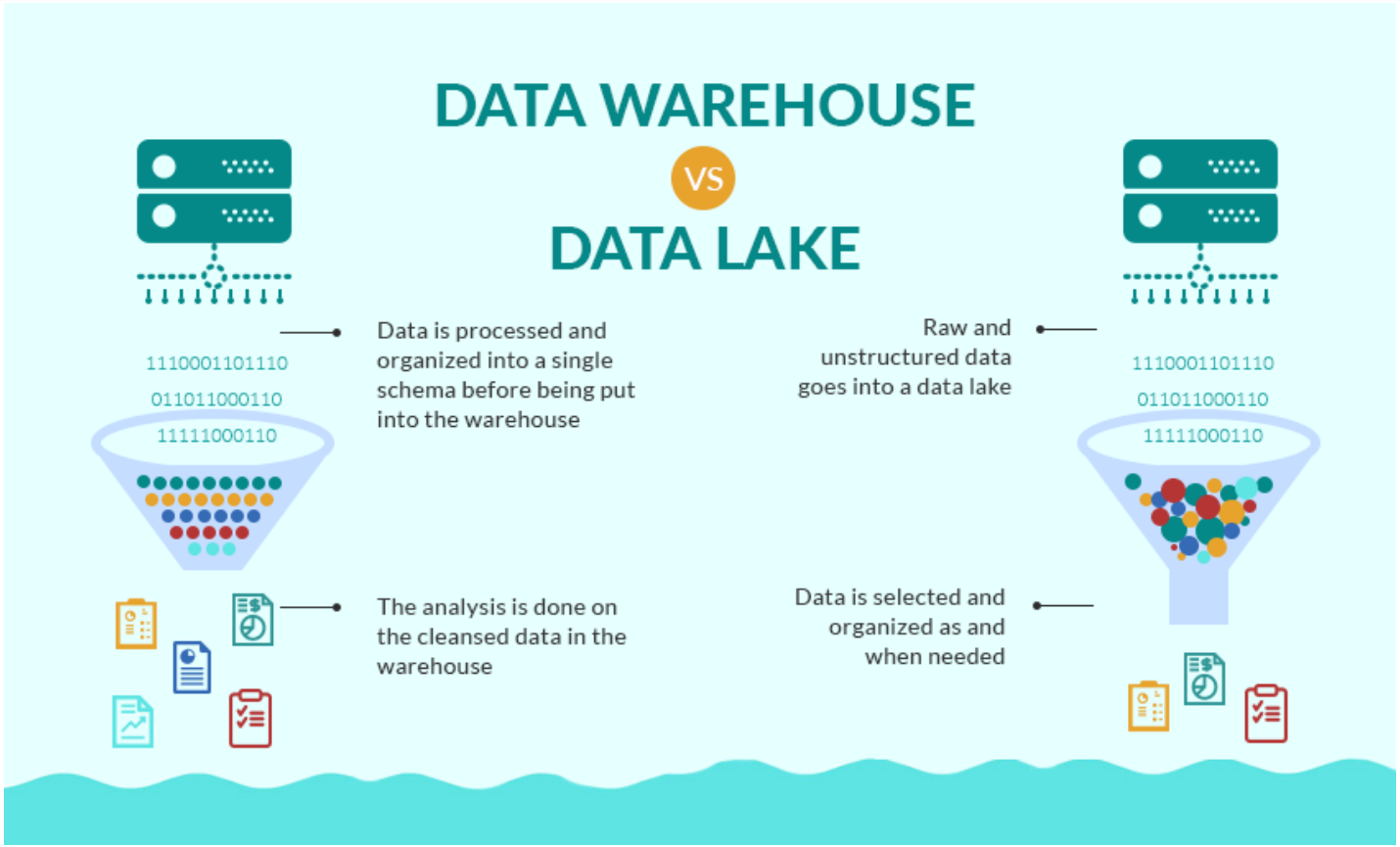
Big Data Ecosystem/Platform



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

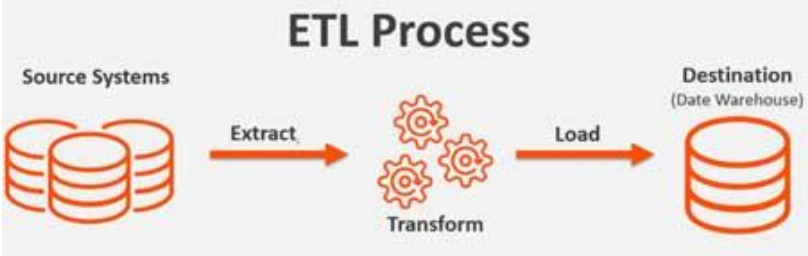
Data Engineering

Data Warehouse and Data Lake



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 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:

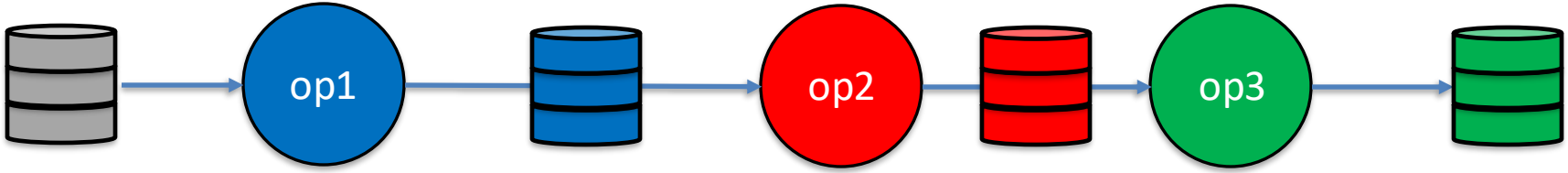
- Basic transformations:
- Cleaning
- Format revision
- Data threshold validation checks
- Restructuring
- Deduplication
- Advanced transformations:
- Filtering
- Merging
- Splitting
- Derivation
- Summarization
- Integration
- Aggregation
- Complex data validation

Load

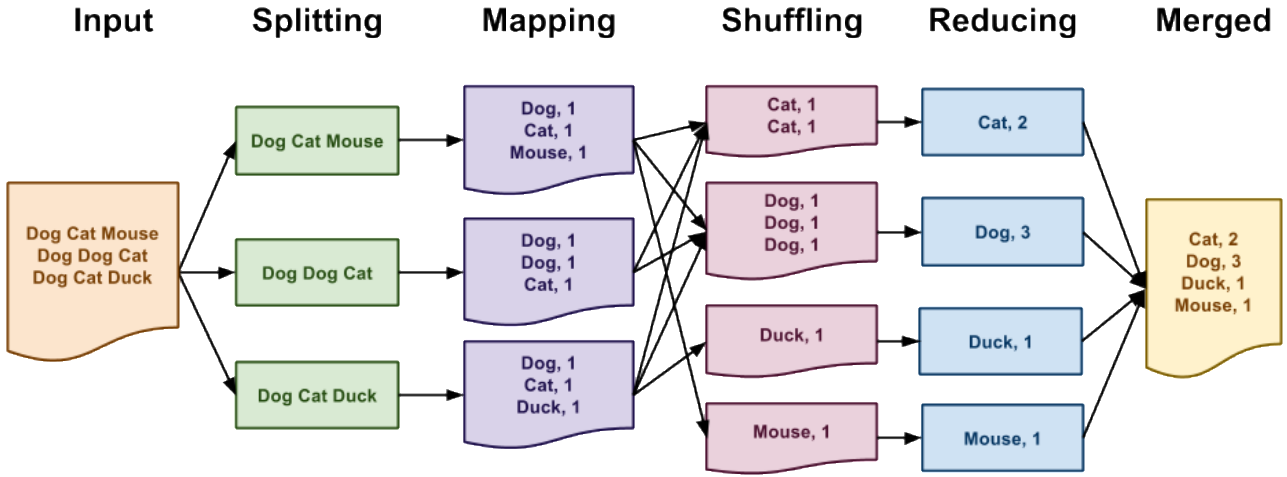
Finally, the load function s is the process of writing converted data from a staging area to a target database, which may or may not have previously existed. Depending on the requirements of the application, this process may be either quite simple or intricate.

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.

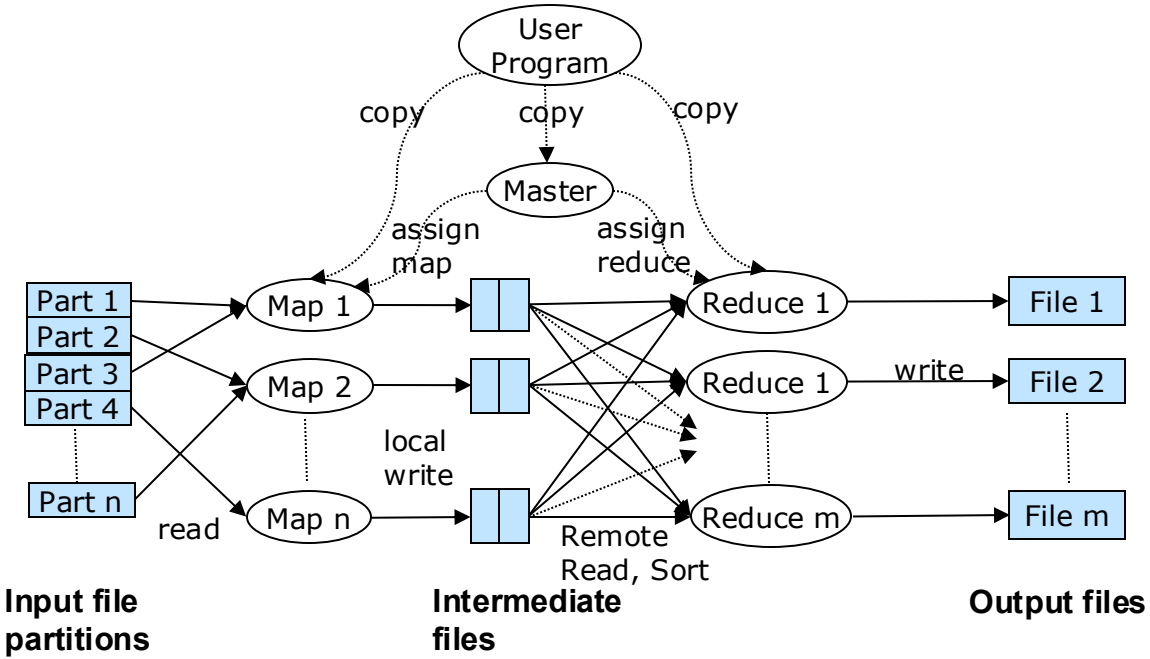


What we want more complex “operators?”

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 on 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 module 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.)

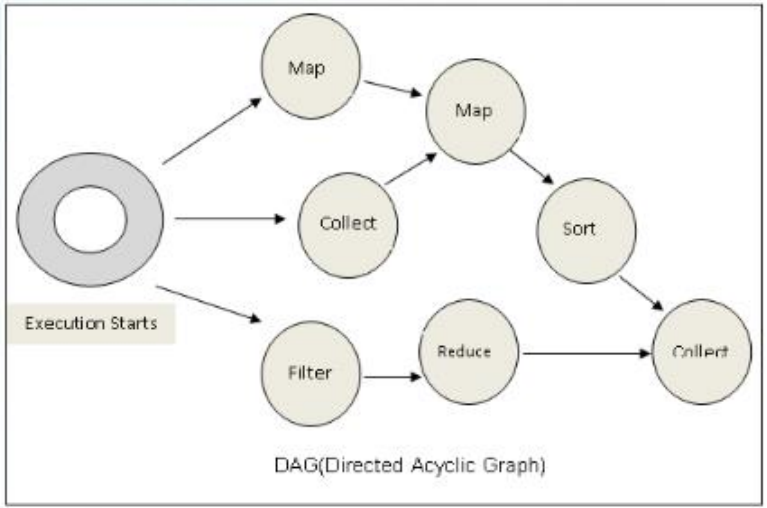
- Relational operations (select, project, join, aggregation, etc.) can be expressed using Map Reduce
- SQL queries can be translated into Map Reduce infrastructure for execution
  - Apache Hive SQL, Apache Pig Latin, Microsoft SCOPE
- Current generation execution engines support not only Map Reduce,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

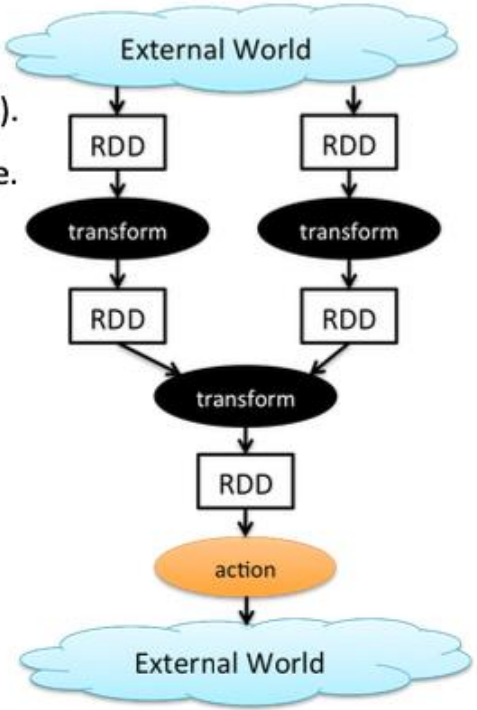
Algebraic Operations

- 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
  - Tex 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).

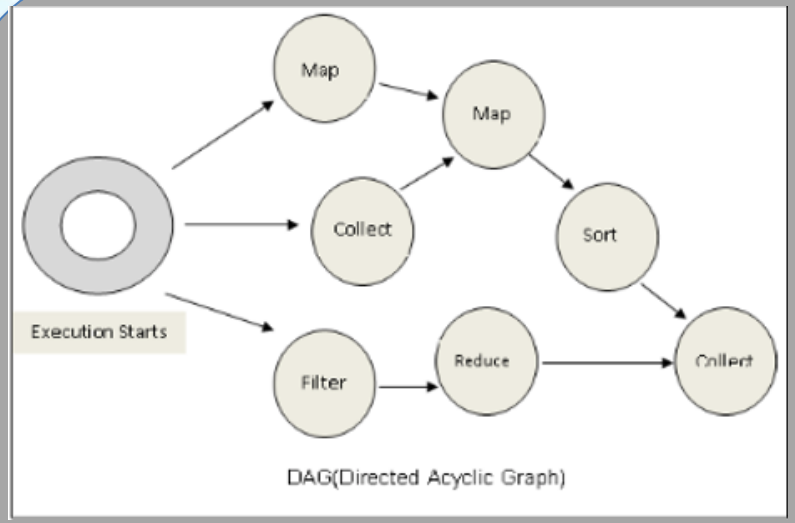
- All jobs in spark comprise a series of operators and run on a set of data.
- All the operators in a job are used to construct a DAG (Directed Acyclic Graph).
- The DAG is optimized by rearranging and combining operators where possible.

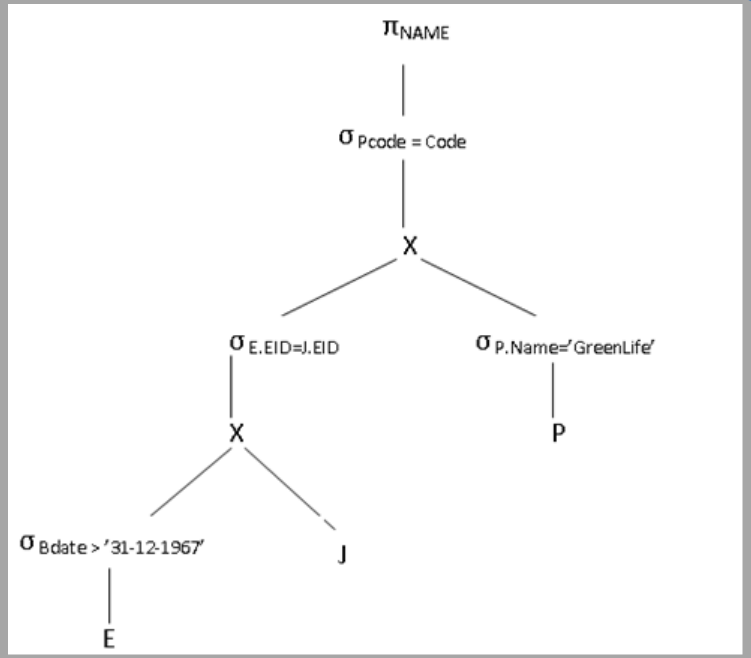




www.edureka.co/apache-spark-scala-training

Spark





Conceptually similar to query evaluation graph, but …

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

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

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.

DataFrame features

edureka!

- 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

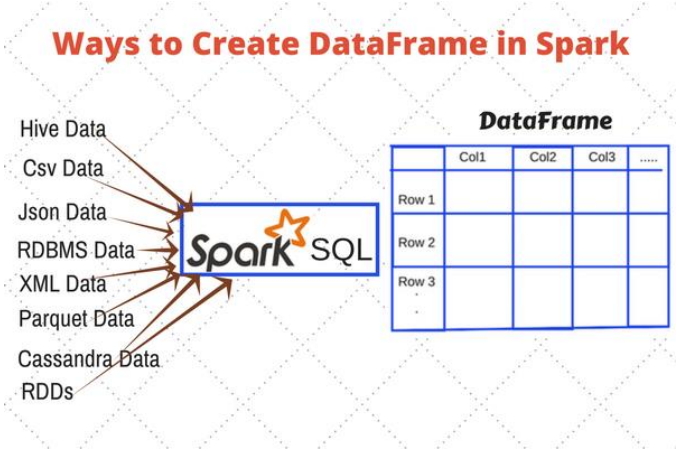
1. DataFrame in PySpark: Overview

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.

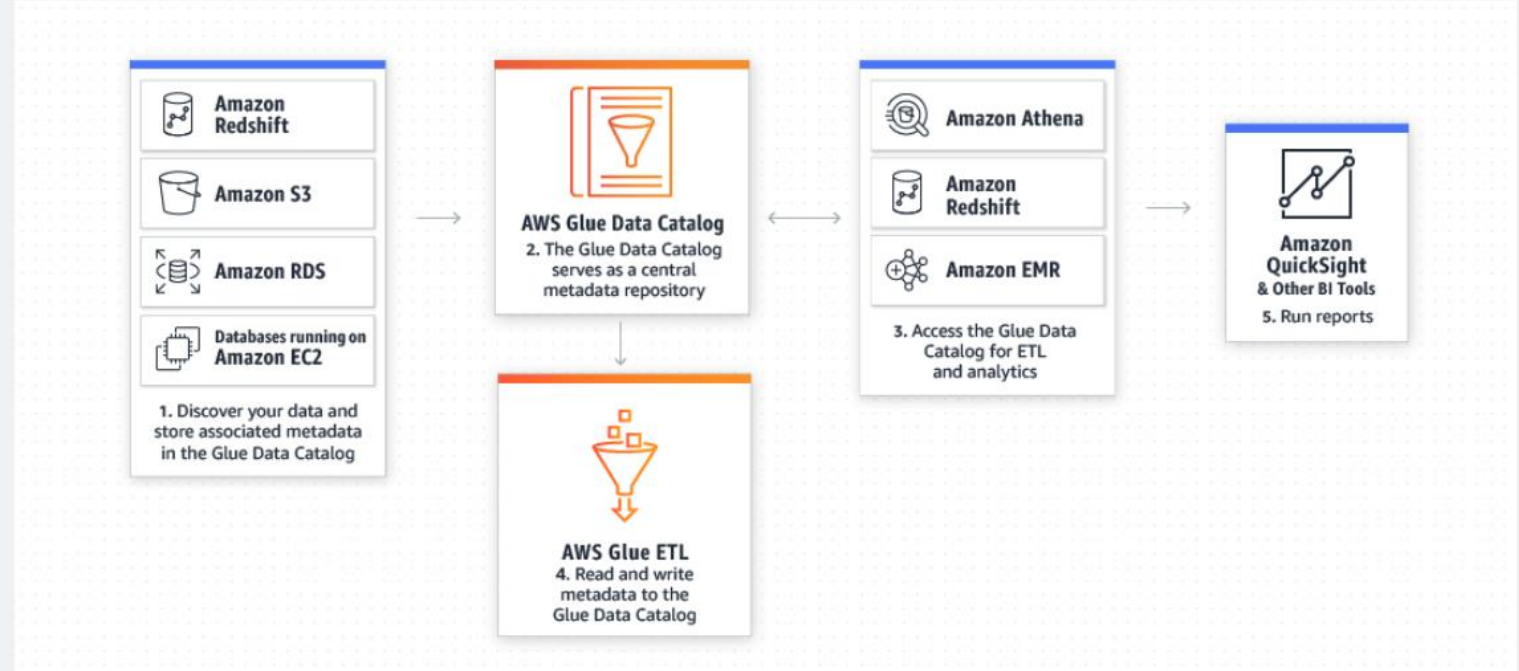
My first exposure to DataFrames was when I learnt about Pandas. Today, it is difficult for me to run my data science workflow with out Pandas DataFrames. So, when I saw similar functionality in Apache Spark, I was excited about the possibilities it opens up!

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



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.



Crawlers/Classifiers

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 Ion	`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.

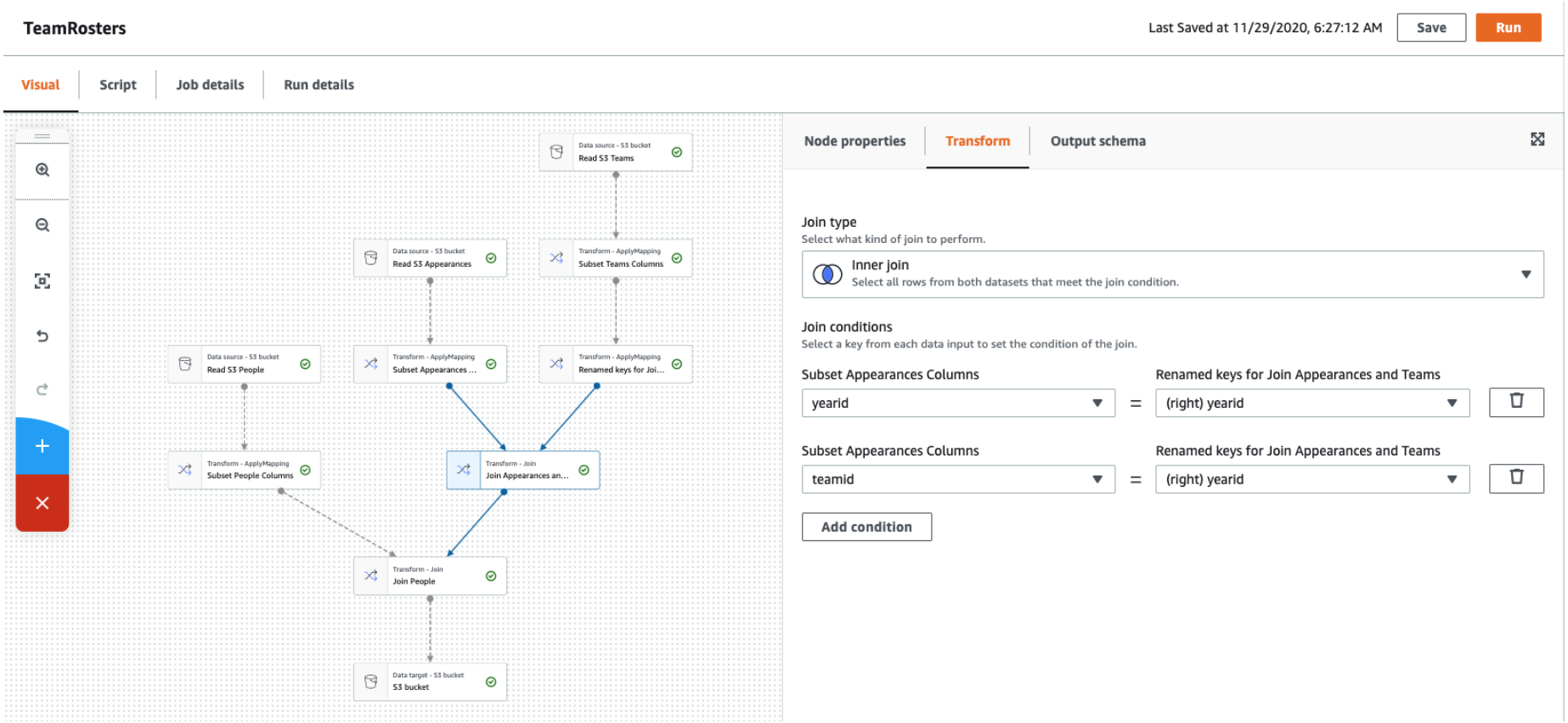
SplitRows

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, ...)



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

- To

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

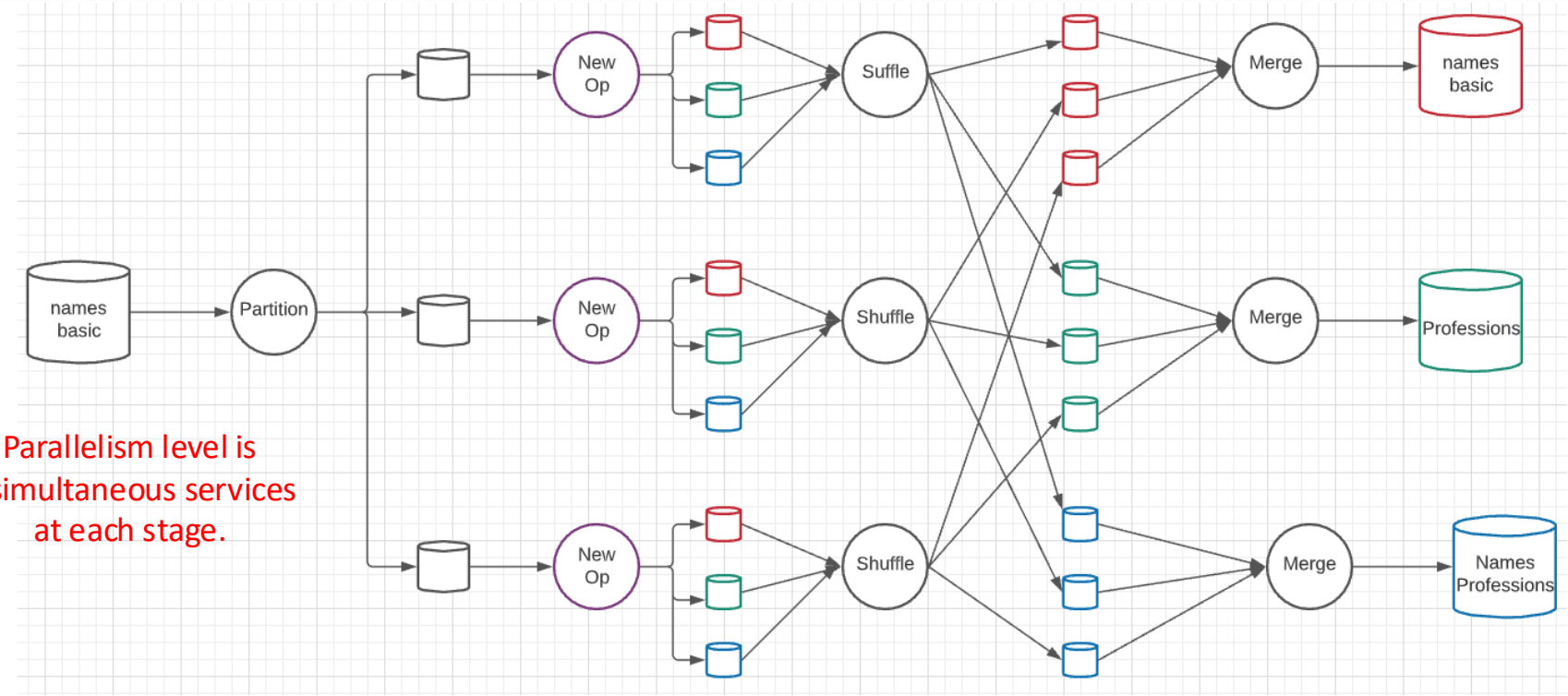
nm0000001	Fred Astaire	1899	1987
nm0000002	Lauren Bacall	1924	2014
nm0000003	Brigitte Bardot	1934	
nm0000004	John Belushi	1949	1982
nm0000005	Ingmar Bergman	1918	2007
nm0000006	Ingrid Bergman	1915	1982
nm0000007	Humphrey Bogart	1899	1957
nm0000008	Marlon Brando	1924	2004
nm0000009	Richard Burton	1925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

-
-



Parallelism level is
3 simultaneous services
at each stage.

External Material

- Show external tutorials
  - https://www.datacamp.com/tutorial/pyspark-tutorial-getting-started-with-pyspark
  - https://www.tutorialspoint.com/pyspark/index.htm
- Page through PDF touching on key components ./sparkknolx-190107072101.pdf
- Go through examples:
  - /Users/donald.ferguson/Dropbox/000/00-Current-Repos/pySpark
    - process_imdb.ipynb
  - Localhost:4040 for Spark console.

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 that 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 overlab
  - **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

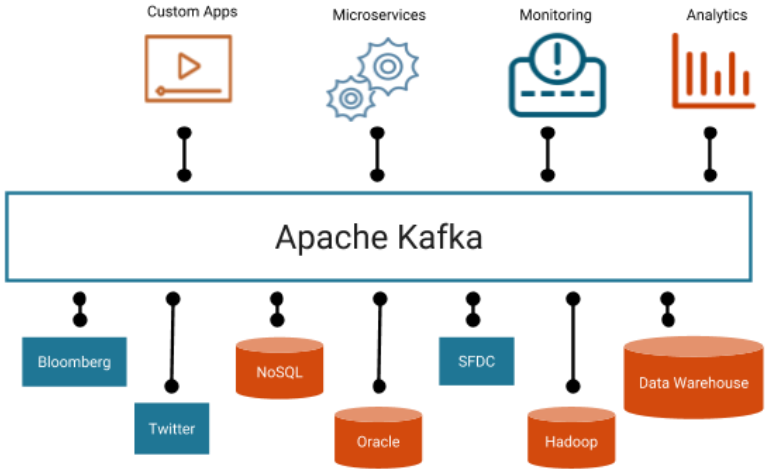
- Windowing syntax varies widely by system
- E.g., in Azure Stream Analytics SQL:

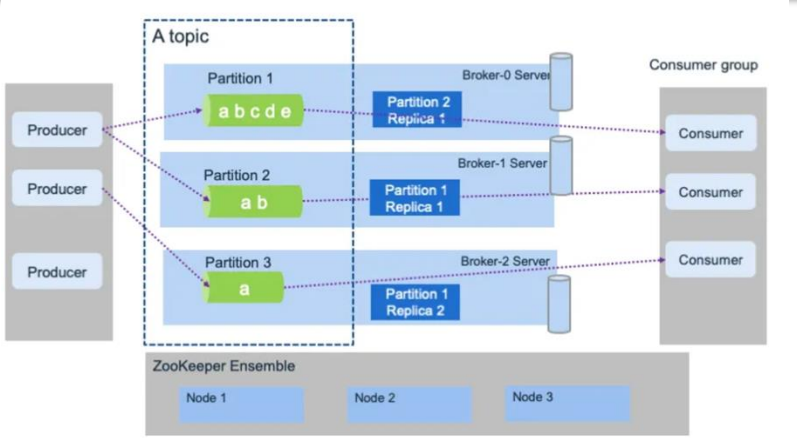
  select *item*, *System.Timestamp* **as** *window end*, **sum**(*amount*)
  from *order* **timestamp by** *datetime*
  group by *itemid*, **tumblingwindow**(*hour*, 1)

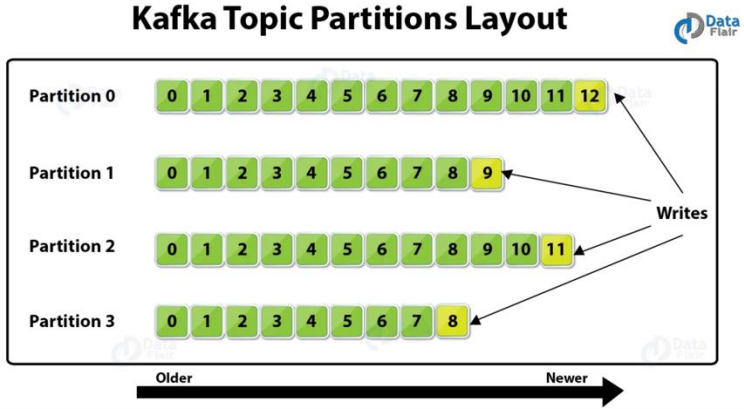
- Aggregates are applied on windows
- Result of windowing operation on a stream is a relation
- Many systems support stream-relation joins
- Stream-stream joins often require join conditions to specify bound on timestamp gap between matching tuples
  - E.g., tuples must be at most 30 minutes apart in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

Stream Processing – Kafka Example

- Kafka 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.

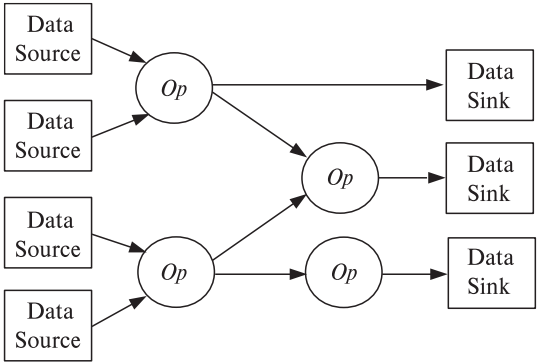




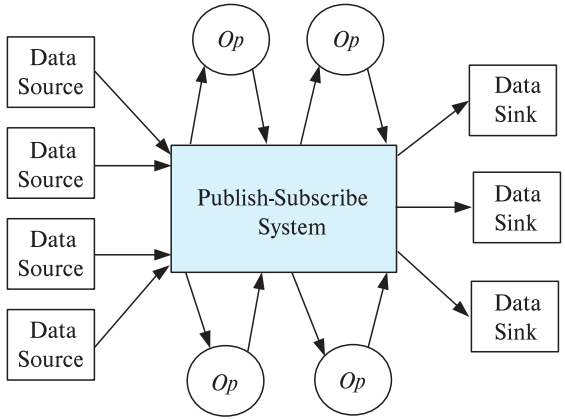


Algebraic Operations on Streams

- 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