

Introducing Course – Managing Big Data



Data?

All stored/generated binary (digital) content is data – could be useful or not.



Big Data?

All (un)structured data that requires some novelty in storing and querying for usability (usually distributed across machines)

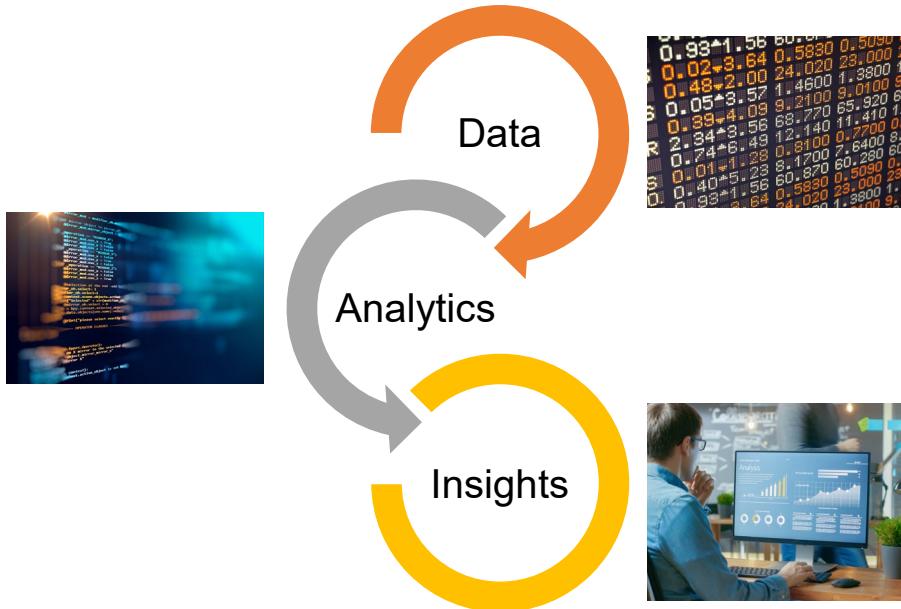


Managing Big Data?

Managing acquisition, storage, querying, and insight development of all usable big data.



Data is the New Gold!



Data is the new gold. This is how it can benefit everyone – while harming no one



A new World Economic Forum paper highlights innovations to advance and refine a new paradigm for the business of data.

Image: Getty Images

29 Jul 2020

Derek O'Halloran

Head of Shaping the Future of Digital Economy and New Value Creation, World Economic Forum

- Digitalization is yielding vast quantities of data, which offer opportunities for business, human well-being and the environment, if used effectively.

Why Data?

- Know what is available and what could be available
- Know potential transformations from data to insights
- Enables going outside the canned off-the-shelf solutions for decision making



Course Content (technologies)



Course Content



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)



Relational Databases

- ER Model
- SQL
- Warehousing

Database Processing

- Pig
- Hive

Unstructured Databases

- NoSQL (Hbase, DocumentDB)
- HDFS
- MapReduce
- Transfer data (RDBMS->HDFS): Sqoop
- Workflow Scheduler: Oozie

Spark

- Faster in-Memory Processing
- Spark SQL
- Spark Streaming



Course Goals:

- Be able to use common big data technologies to:
 - Connect to a database, run queries, and present summaries
 - Read a dataset, process, and analyze
 - Connect and query data on cloud

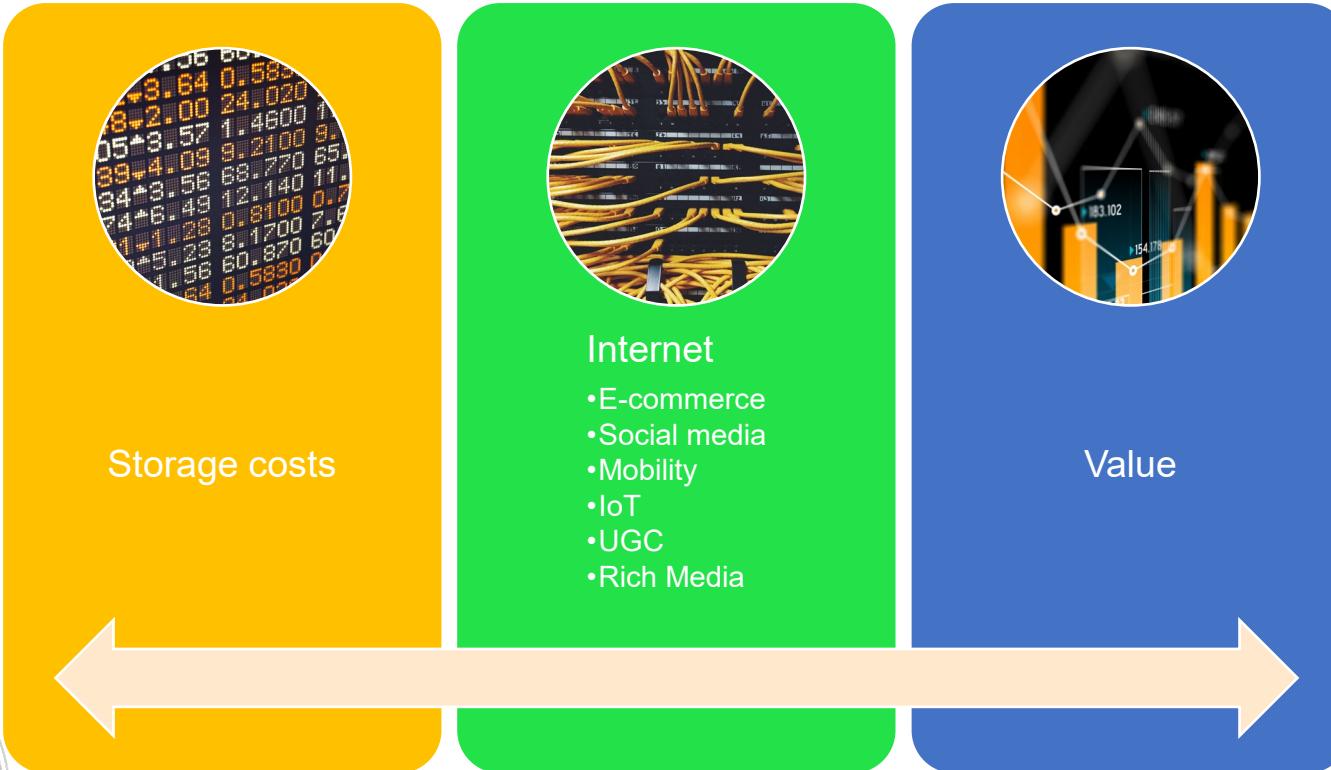




DATA

August 23, 2023

Data Explosion

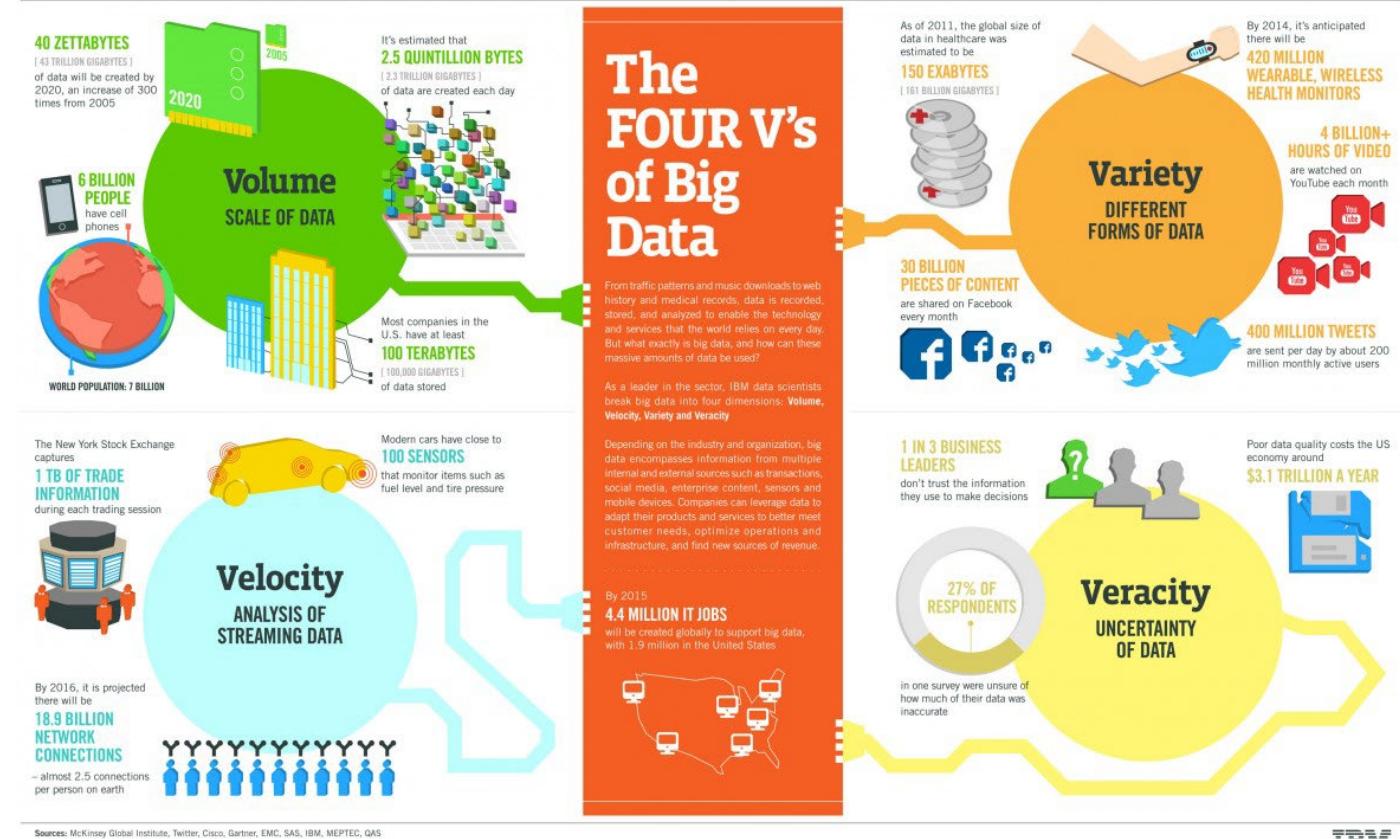


Source: <https://images.techhive.com/assets/2017/04/10/cw-50th-anniversary-storage-trends.pdf>



Data Characteristics

- The 4 Vs of Big Data:
 - Volume
 - Velocity
 - Variety
 - Veracity

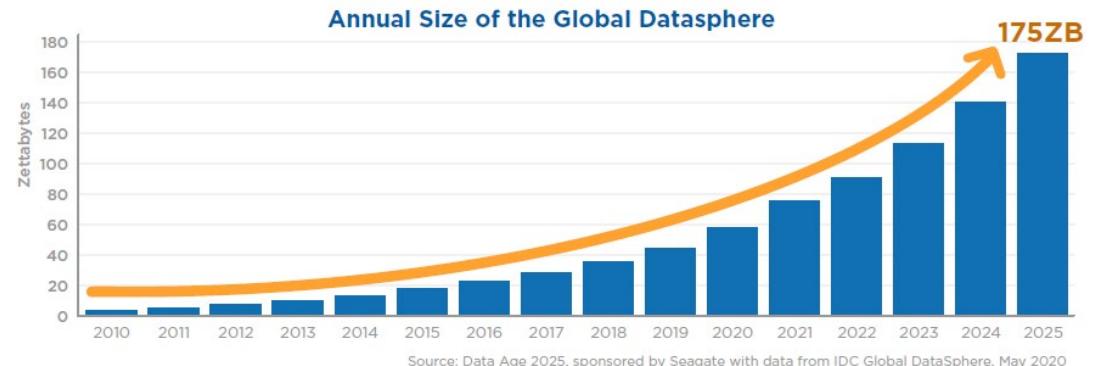


Source: <https://www.ibmbigdatahub.com/infographic/four-vs-big-data>

Data – Volume

- 1 Zettabyte (10^{21}) = 1,000 Exabytes (10^{18})
- 1 Exabytes (10^{18}) = 1,000 Petabytes (10^{15})
- 1 Petabytes (10^{15}) = 1,000 Terabytes (10^{12})
- 175 Zettabytes/yr = 175 Billion 1 TB HDDs deployed every year!

Figure 1 - Annual Size of the Global Datasphere



Source: <https://www.seagate.com/files/www-content/our-story/trends/files/dataage-idc-report-07-2020.pdf>

Data – Variety

- Transactions
- Logs
- Text
- IoT
- Audio
- Visual



Sources:

<https://www.ibm.com/downloads/cas/E4BWZ1PY>

<https://www.seagate.com/files/www-content/our-story/trends/files/dataage-idc-report-07-2020.pdf>

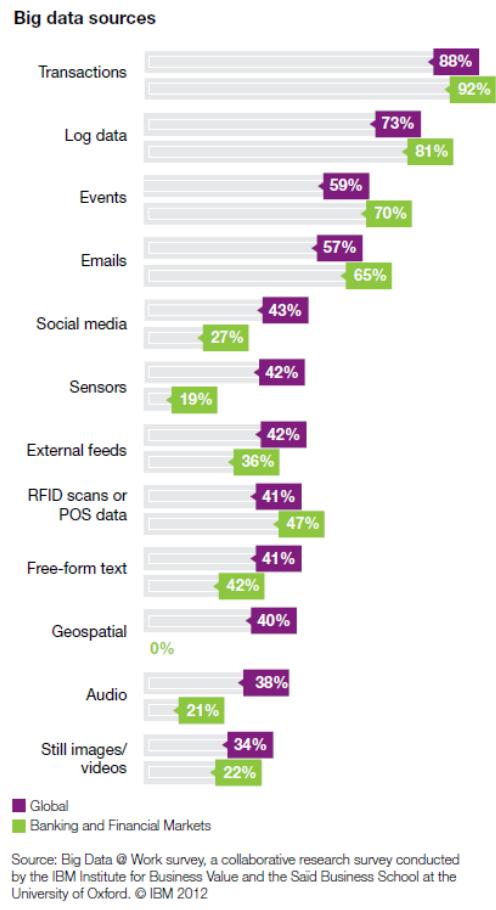
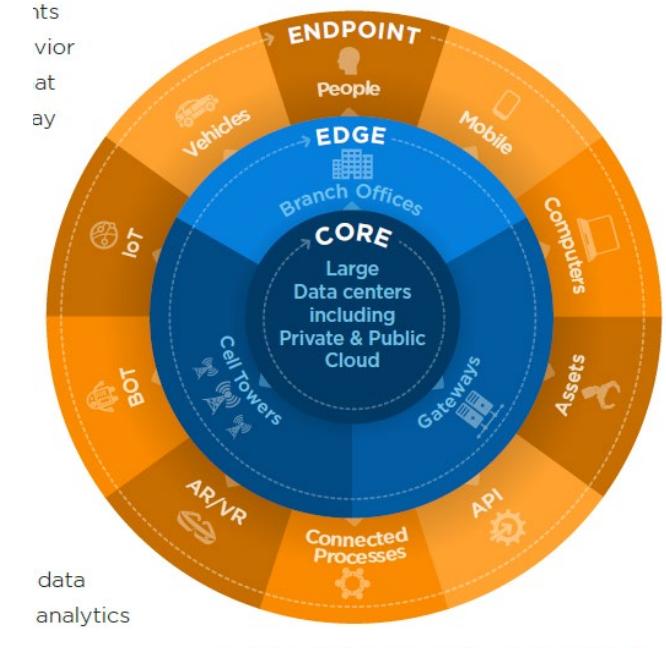


Figure 2
Data propagation from endpoints to core and back



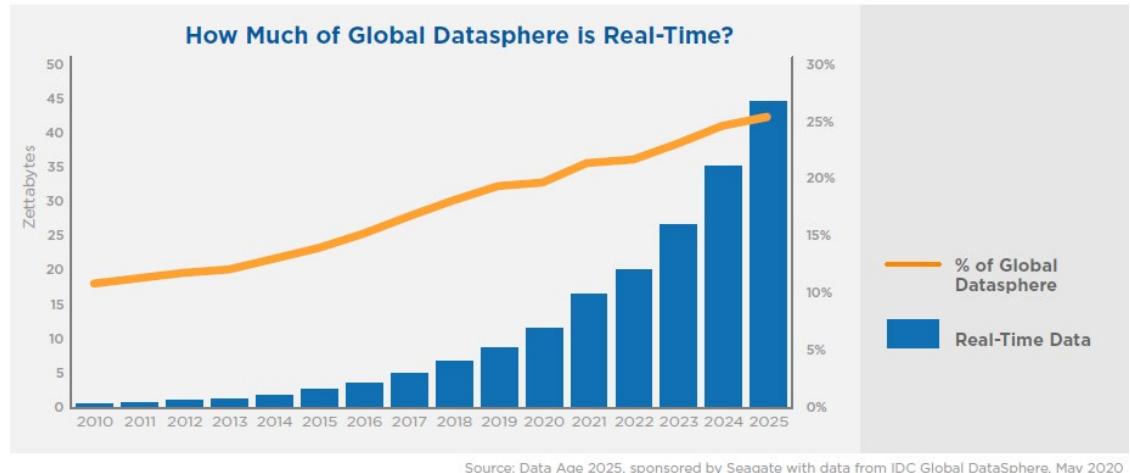
Source: IDC's Data Age 2025 study, sponsored by Seagate



Data – Velocity

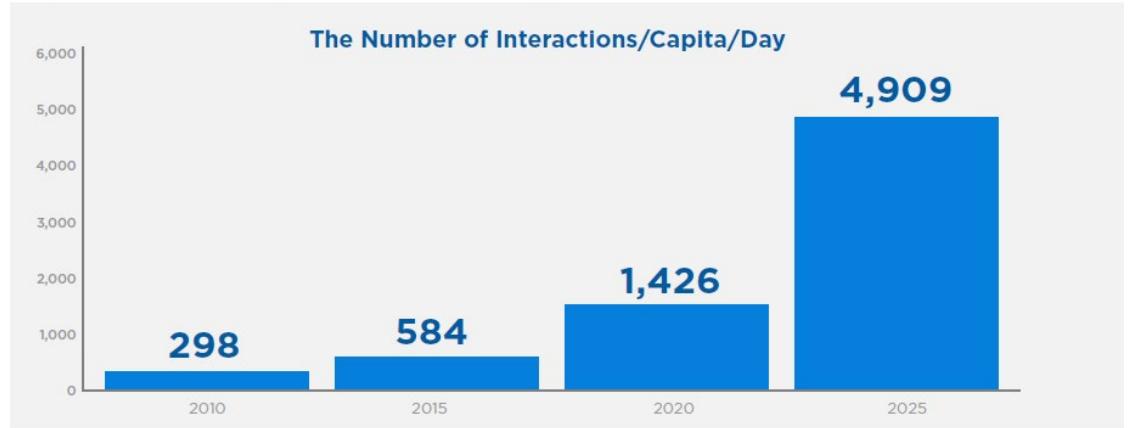
- 25% of data being generated is in real-time
- Every connected human is generating almost 5,000 data points per day

Figure 9 – Real-Time Data



Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, May 2020

Figure 10 – Data Interactions per Connected Person Per Day



Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018

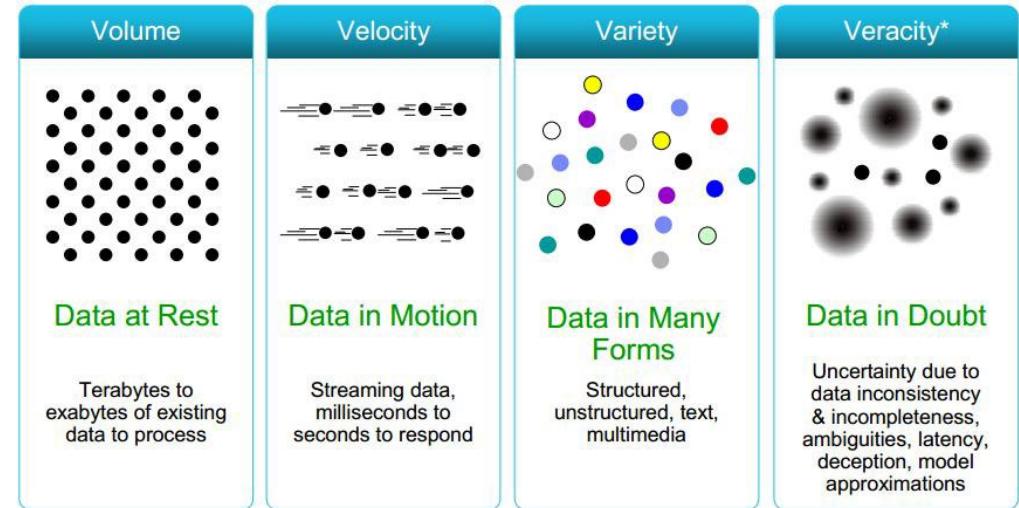


Source: <https://www.seagate.com/files/www-content/our-story/trends/files/dataage-idc-report-07-2020.pdf>



Data – Veracity

- \$3.1T lost in use of poor data
- 33% business leaders don't trust data



Applications and Demand

- Applications
 - Customer-centric outcomes
 - Operations
 - Financial Management
 - New Business Enabling
 - Human collaboration
- Demand:
 - Top Five Skills In-Demand For Data Analysis Jobs
 - Data Analysis
 - SQL
 - Data Management
 - Business Intelligence
 - Data Warehousing

Source: <https://www.ibm.com/downloads/cas/E4BWZ1PY>

Source: <https://www.forbes.com/sites/jeffkauflin/2017/07/20/the-five-most-in-demand-skills-for-data-analysis-jobs/>



Why Big Data?

- Technical skills with biggest increases in demand:
 1. **Big Data (Information Technology): 3,977%**
 2. Node.js (Design): 2,493%
 3. Tableau (Research and Analysis): 1,581%
 4. **NoSQL (Information Technology): 1,002%**
 5. **Apache Hadoop (Information Technology): 704%**
 6. HTML5 (Information Technology): 612%
 7. Python (Research and Analysis): 456%
 8. Oracle (Sales): 382%
 9. JSON (Information Technology): 318%
 10. Salesforce CRM (Sales): 292%



Source: <https://www.forbes.com/sites/jeffkaufin/2017/01/08/the-10-technical-skills-with-explosive-growth-in-job-demand/>

The 10 Technical Skills With Explosive Growth In Job Demand



Jeff Kauflin Forbes Staff

Fintech

I cover fintech, cryptocurrencies, blockchain and investing.



Some aspects of technology seem to change every month. If you work as a data scientist or engineer, the tools you use don't change quite so quickly, but can shift every few years. Bentley University commissioned a study to find which business skills are growing in demand. By looking at millions of job listings posted on more than 40,000 online job sites, jobs analytics firm Burning Glass determined which skills saw the biggest increases in demand when comparing 2011 to 2015.



Why Big Data?

- Most in-demand highest-paying skills for the future:
 1. *Cloud computing*
 2. *Cybersecurity*
 3. *Artificial intelligence and machine Learning*
 4. *Big Data analytics*
 5. *Virtual and augmented reality*
 6. *Blockchain*
 7. *Video production*
 8. *User experience (UX)*



<https://hrforecast.com/a-guide-to-future-oriented-skills-skills-in-demand-to-watch-in-the-next-five-years/>

The 10 Technical Skills With Explosive Growth In Job Demand



Jeff Kauflin Forbes Staff

Fintech

I cover fintech, cryptocurrencies, blockchain and investing.

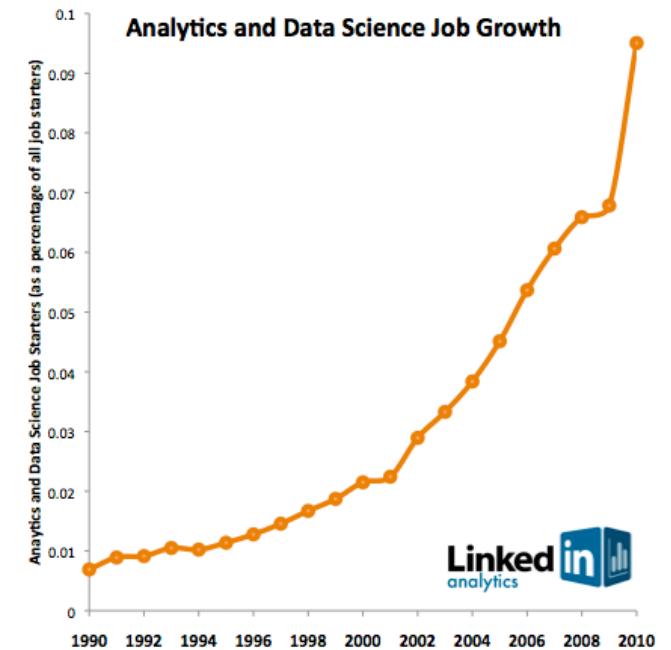


Some aspects of technology seem to change every month. If you work as a data scientist or engineer, the tools you use don't change quite so quickly, but can shift every few years. Bentley University commissioned a study to find which business skills are growing in demand. By looking at millions of job listings posted on more than 40,000 online job sites, jobs analytics firm Burning Glass determined which skills saw the biggest increases in demand when comparing 2011 to 2015.



What Makes a “good” Data Scientist?

- According to DJ Patil:
 - **“Technical expertise:** the best data scientists typically have deep expertise in some scientific discipline.
 - **Curiosity:** a desire to go beneath the surface and discover and distill a problem down into a very clear set of hypotheses that can be tested.
 - **Storytelling:** the ability to use data to tell a story and to be able to communicate it effectively.
 - **Cleverness:** the ability to look at a problem in different, creative ways.”



Source: <http://radar.oreilly.com/2011/09/building-data-science-teams.html>



Extracting Value

Data:

- Ingestion
- Integration

Processing:

- Unification
- Reduced Complexity
- Improved Accessibility

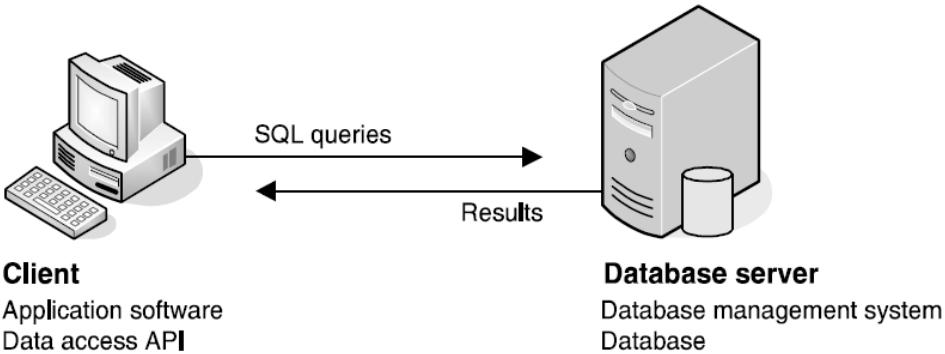
Value:

- Insights
- Visualizations
- Collaboration/Sharing

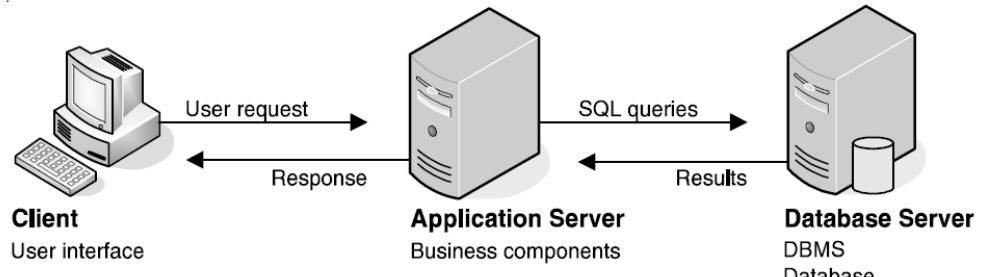


Accessing Data

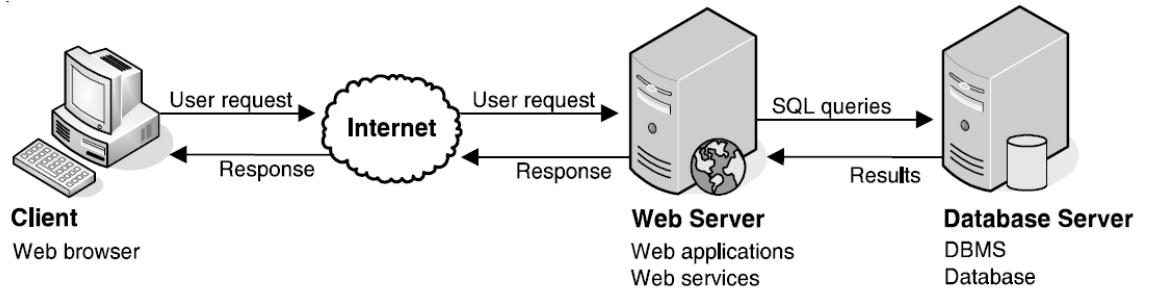
Client software, server software, and the SQL interface



A networked system that uses an application server

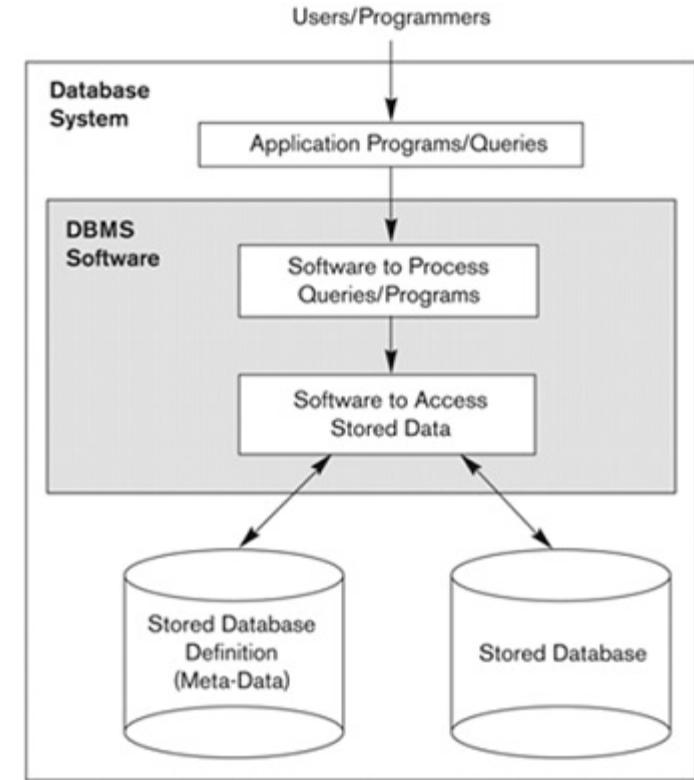


A simple web-based system



DBMS & Data

- Database System
 - Applications
 - DBMS Software
 - Process Queries
 - Access Stored Data
 - Stored Data
 - Meta-Data
 - Data



DBMS (database management system) is a general-purpose software system that facilitates the processes of defining, creating, using, and maintaining databases.



Source: textbook [en] ch1, fig 1.1

Popular DBMS

- Server-based:
 - Commercial: Oracle, MS SQL Server, DB2, Teradata
 - Open source: MySQL, PostgreSQL, Hive (based on Hadoop)
 - Desktop/Personal: MS Access
- Non-relational: MongoDB, Cassandra, Hbase, Redis, Hive, etc.

Rank Aug 2023	Rank Jul 2023	Rank Aug 2022	DBMS	Database Model	Score		
					Aug 2023	Jul 2023	Aug 2022
1.	1.	1.	Oracle	Relational, Multi-model	1242.10	-13.91	-18.70
2.	2.	2.	MySQL	Relational, Multi-model	1130.45	-19.89	-72.40
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	920.81	-0.78	-24.14
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.38	+2.55	+2.38
5.	5.	5.	MongoDB	Document, Multi-model	434.49	-1.00	-43.17
6.	6.	6.	Redis	Key-value, Multi-model	162.97	-0.80	-13.43
7.	↑ 8.	↑ 8.	Elasticsearch	Search engine, Multi-model	139.92	+0.33	-15.16
8.	↓ 7.	↓ 7.	IBM Db2	Relational, Multi-model	139.24	-0.58	-17.99
9.	9.	9.	Microsoft Access	Relational	130.34	-0.38	-16.16
10.	10.	10.	SQLite	Relational	129.92	-0.27	-8.95
11.	11.	↑ 13.	Snowflake	Relational	120.62	+2.94	+17.50
12.	12.	↓ 11.	Cassandra	Wide column, Multi-model	107.38	+0.86	-10.76
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	98.65	+2.55	-15.24
14.	14.	14.	Splunk	Search engine	88.98	+1.87	-8.46
15.	↑ 16.	15.	Amazon DynamoDB	Multi-model	83.55	+4.75	-3.71
16.	↓ 15.	16.	Microsoft Azure SQL Database	Relational, Multi-model	79.51	+0.55	-6.67
17.	17.	17.	Hive	Relational	73.35	+0.48	-5.31
18.	18.	↑ 22.	Databricks	Multi-model	71.34	+2.87	+16.72
19.	19.	↓ 18.	Teradata	Relational, Multi-model	61.31	+1.06	-7.76
20.	20.	↑ 24.	Google BigQuery	Relational	53.90	-1.52	+3.87

Source: <https://db-engines.com/en/ranking>



Why NOT Flat File Systems?

- Why don't we store everything in flat files?
 - Separation and isolation of data
 - Duplication of data
 - Program-data dependence
 - Limited query capability



Why DBMS?

- *Reduce (but not eliminate) redundancy*: all data items are integrated with a minimum amount of duplication
- *Data consistency*: DBMS reduces the risk of inconsistencies occurring
- *Improved data accessibility*: due to data integration and query languages like SQL
- *Program-data independence*: definition of data is separated from the application programs



Data independence

- **Concept:** Applications do not need to worry about how the data is structured and stored
- **Logical data independence:** protection from changes in the logical structure of the data
 - i.e., should not need to ask: can we add a new entity or attribute without rewriting the application?
- **Physical data independence:** protection from physical layout changes
 - i.e., should not need to ask: which disks are the data stored on? Is the data indexed?

One of the most important reasons to use a DBMS!



DB Design

Entity-Relationship Conceptual Model

Data Models

- Big data system – key-value system
 - Document-based data model (JSON like)
 - Graph-based data model (nodes-edges)
 - Column-based data model
 - Key-value data model (every record is a key-value pair)
- eXtended Markup Language (XML) model
 - tree-structured data model
 - Standard for exchanging data online

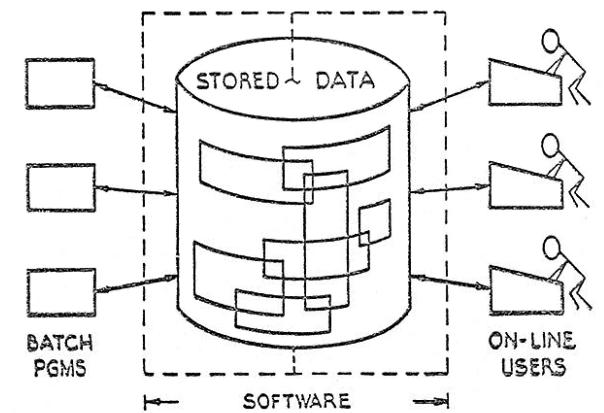
Relation Database (RDBMS)



Relational Database

- In 1970, Dr. E. F. Codd developed a model for a new type of database called a **relational database**.
 - Eliminated problems associated with standard files (e.g. size of data)
 - Reduce redundancy
 - Saves disk storage and leads to efficient data retrieval
 - View and manipulate data in ways that are more intuitive and efficient
- These benefits made relational databases the standard for database applications.

A DATABASE SYSTEM

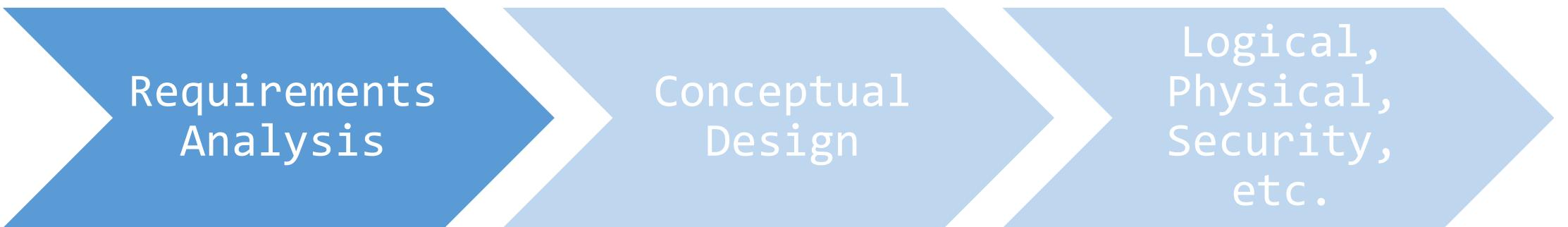


Sources:

<https://www.ibm.com/ibm/history/ibm100/us/en/icons/reldb/>
<https://dl.acm.org/doi/10.1145/362384.362685>



Database Design Process



- Requirements analysis:
 - What is going to be stored?
 - How is it going to be used?
 - What are we going to do with the data?
 - Who should have access to the data?



Database Design Process

- Conceptual Design \ Modeling:
 - A **high-level** description of the database
 - Sufficiently **precise** that technical people can understand it
 - But, not so **precise** that non-technical people can't participate



Database Design Process

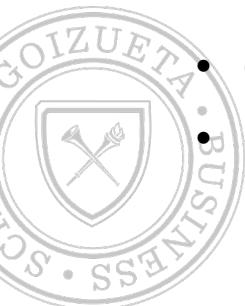


- Physical design and beyond:
 - Logical Database Design
 - Physical Database Design
 - Security Design



Relational Database Conceptual Design

- Conceptual modeling is an important phase in designing a successful database application.
- Relational Database Design Elements:
 - A table in a relational database typically represents an object or entity
 - Each column (field) of a table is used to store an attribute associated with the entity
 - Each row (record) represents one instance of the entity.
- Table -> Entity
- Column -> Attribute
- Row -> Instance



How Relational Databases Work

Computerized databases help people store and track huge amounts of information. The smallest unit of information in a database is called a **field**. Fields are grouped together to form **records**. Records are then grouped together to form **tables**.

Flat-file databases take all the information from all the records and store everything in one table. This works fine when you have a small number of records related to a single topic, such as a person's name and phone number, but if you have hundreds or thousands of records, each with a number of fields, the database quickly becomes difficult to use.

SID	SFName	SLName	SteleNumber	CID	Cname	TID	Trainer	TrnTeleNumber
1	Mary	Hinkle	555.123.4567	101	Data Basics	T01	Charles Hill	555.987.6543
2	Paul	Litz	555.258.8963	101	Data Basics	T01	Charles Hill	555.987.6542
1	Mary	Hinkle	555.123.4567	102	Web Design	T02	Glen Barber	555.879.4652
3	Dee	Coleman	555.357.9514	203	Relational Design	T03	Rick Dobson	555.324.2986
4	Don	Charney	555.369.8741	204	VBA Programming	T03	Rick Dobson	555.324.2986



Sources:

<https://www.ibm.com/ibm/history/ibm100/us/en/icons/reldb/>
<https://dl.acm.org/doi/10.1145/362384.362685>

Entity-Relationship Diagram

- An **Entity-Relationship (ER) Diagram** is a conceptual model that identifies the entities that exist in a system and the relationships between those entities.
 - The ER model describes data as entities, relationships, and attributes.
- Example: Employee entity (table) and attributes (columns)



Entity-Relationship Diagram(example)

- Conceptual Design:

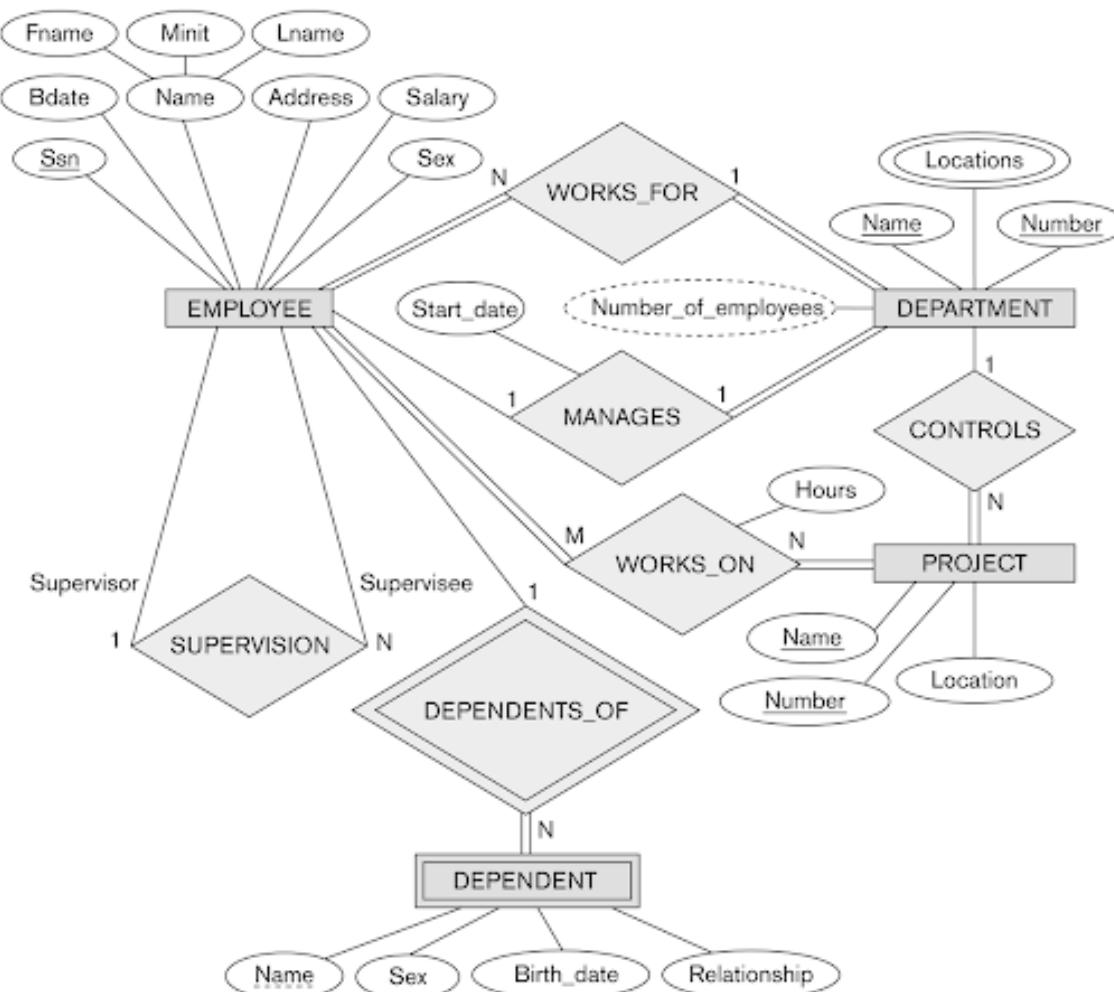


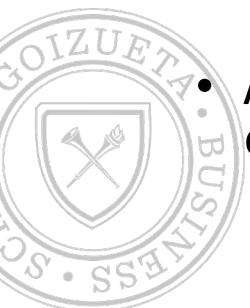
Figure 3.2

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Source: Course textbook ("Database Systems" by Elmasri & Navathe)

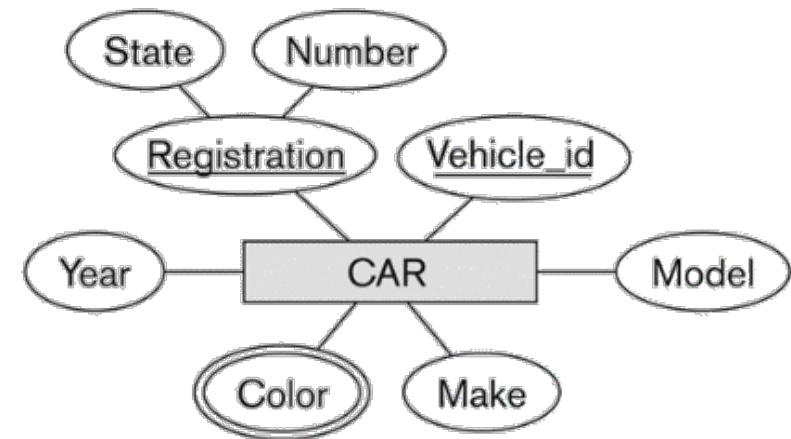
ER Model Constructs (Entities)

- Entities are things or objects in the real world with an independent (physical or conceptual) existence.
 - Entity instance (or simply entity) - person, place, object, event, concept
 - Often corresponds to a row in a table
 - e.g., a vendor, an invoice, a line in an invoice.
 - Entity type (or set) - collection (or set) of entities of a particular entity type
 - Often corresponds to a table
 - e.g., vendors, invoices, invoice line items.
- Each entity has attributes - specific properties that describe it.
- An entity type is represented in ER diagrams as a rectangular box enclosing the entity type name.
 - Entity names are usually singular names, rather than plural ones.



ER Model Constructs (Attributes)

- Attributes:
 - property or characteristic of an entity or relationship type.
 - Often corresponds to a field in a table.
- For example:
 - Entity “Car” could have an attribute “Year”
- Attribute names are enclosed in ovals and are attached to their entity type by straight lines.
 - Composite attributes are attached to their component attributes by straight lines (e.g., Registration -> State + Number)
 - Multivalued attributes are displayed in double ovals (e.g., color)
 - Each key attribute has its name underlined; different from primary keys (e.g., registration, vehicle_id)



Key Attributes

- Criteria for choosing key attributes:
 - Uniquely identifies a row
 - Will not change in value
 - Will not be null
 - no intelligent identifiers
 - e.g., containing location that might change
- We often use meaningless integers as keys for entities
 - e.g., Social security number, Student IDs, vendor IDs
- But keys can also be composite
 - e.g., we use (invoice_id, invoice_sequence) as key for invoice_line_items

Entity types without keys are called weak entity types.



ER Model Constructs (Relationships)

- Relationships:
 - Relationship type – a category of relationships between entity types.
 - e.g., Vendors send invoices; Invoices have terms; vendors are assigned to general_ledger_accounts.
 - The degree of relationship type is the number of participating entity types.
 - Relationship instance – link between two entity instances.
 - e.g., Vendor 112 (Office depot) sent invoice 6.
 - Vendor 112 has a default account number 570 (for office supplies)
- Relationships are shown in diamond-shaped boxes attached to the participating entity types with straight lines.
 - Identifying relationship (double diamond-shaped box) implies that the child table cannot be uniquely identified without the parent table



Entity-Relationship Diagram(example)

- Identify:
 - Entities
 - Attributes
 - Composite
 - Multivalued
 - Key
 - Relationships
 - Identifying relationship

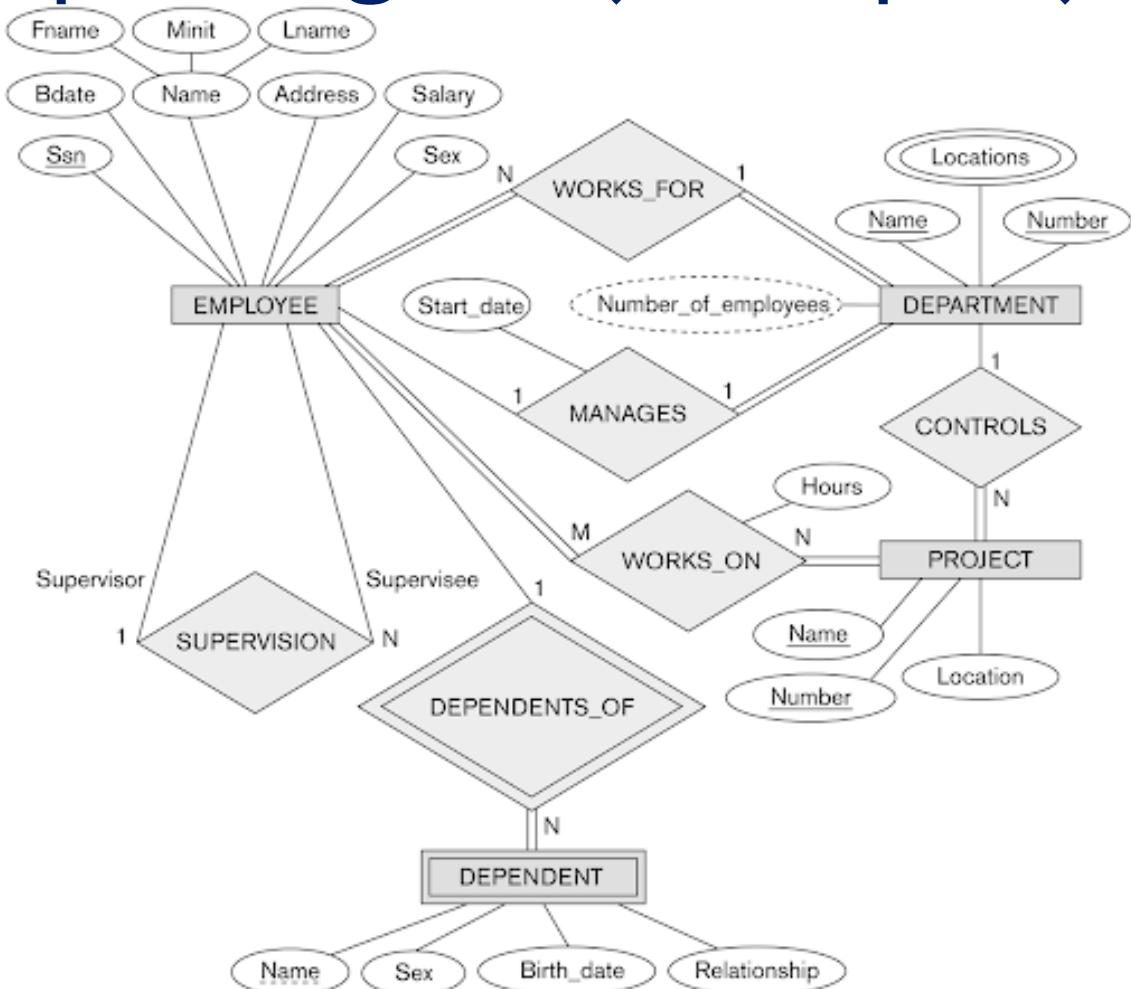


Figure 3.2

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Source: Course textbook ("Database Systems" by Elmasri & Navathe)



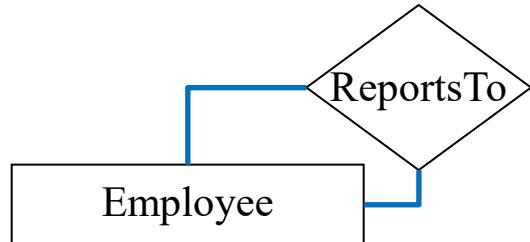
ER Model Constructs (Relationships)

- Degree of a relationship
 - Degree of a relationship is the number of entity types that participate in it:
 - Unary Relationship
 - Binary Relationship
 - Ternary Relationship
- Cardinality of a relationship
 - Describes the number of instances of one entity associated with another entity.
 - One-to-One (1:1)
 - One-to-Many (1:N)
 - Many-to-Many (M:N)
- Optionality of a relationship
 - Optionality expresses whether the relationship is optional or mandatory.
 - A circle (o) indicates that the relationship is optional.
 - A stroke (|) indicates that the relationship is mandatory.

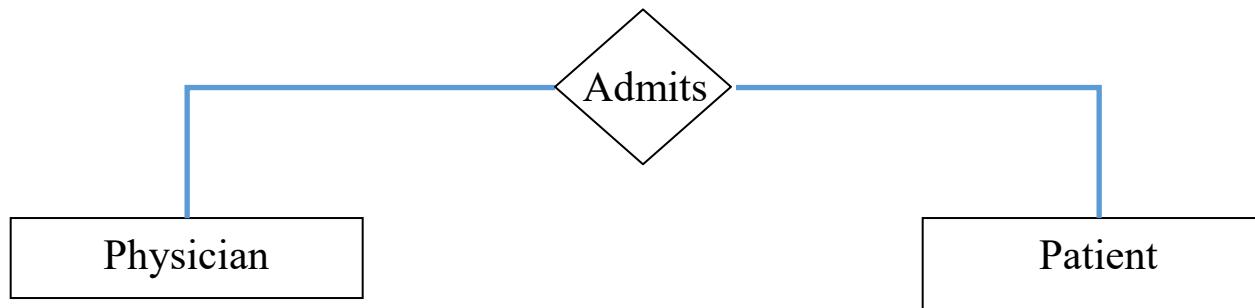


Degree of Relationships

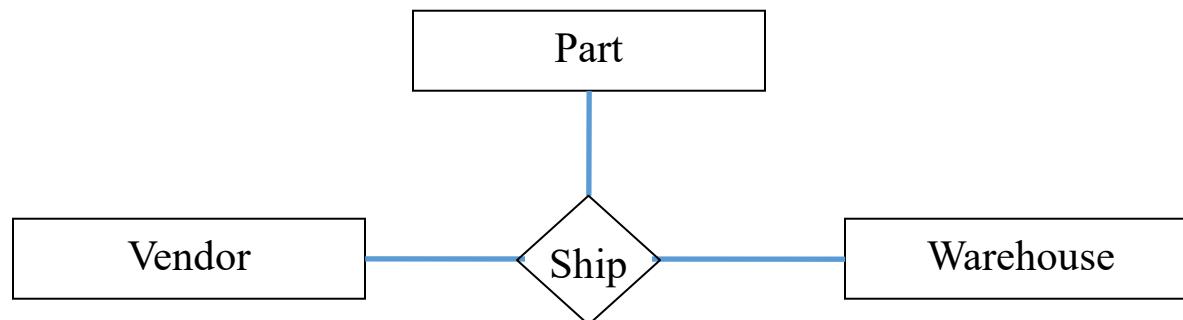
- Unary:



- Binary:



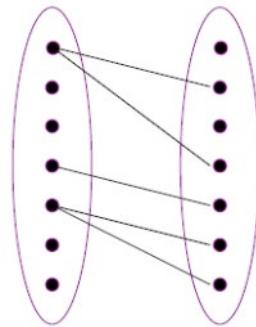
- Ternary:



Cardinality of Relationships

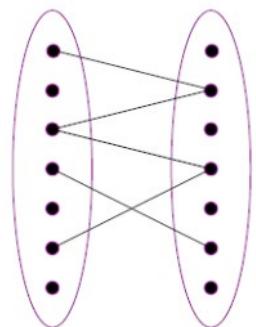
- One-to-Many (1:N)

- An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity
- Examples:
 - Senders & emails
 - Courses and course sessions

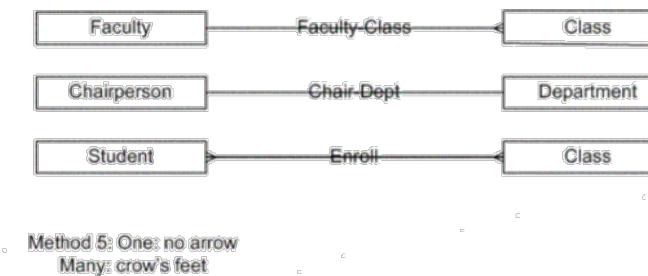
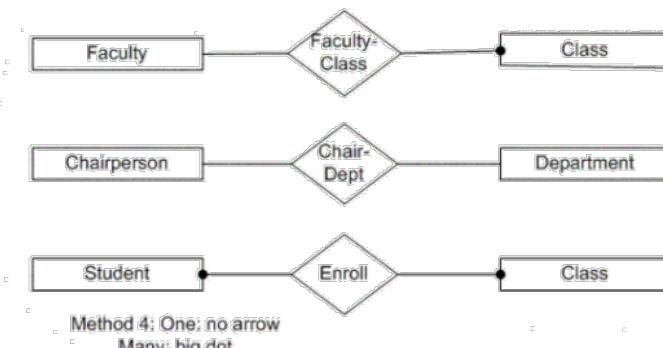
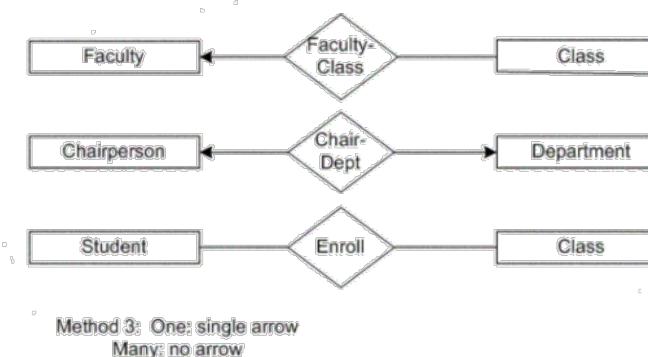
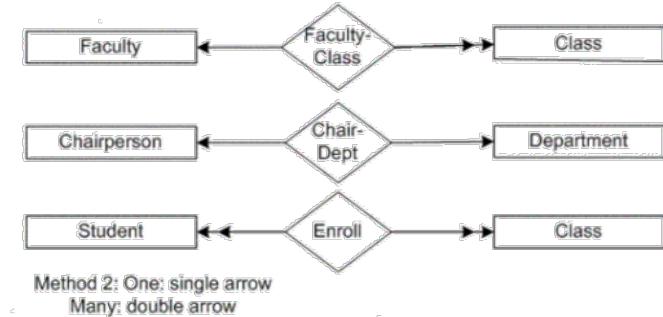
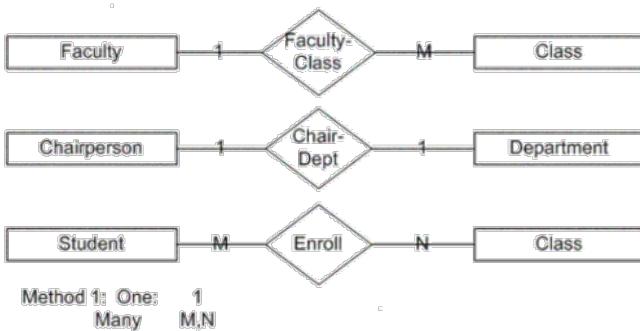


- Many-to-Many (M:N)

- Entities on both sides of the relationship can have many related entities on the other side.
- Examples:
 - Students & courses
 - Facebook users “like” posts

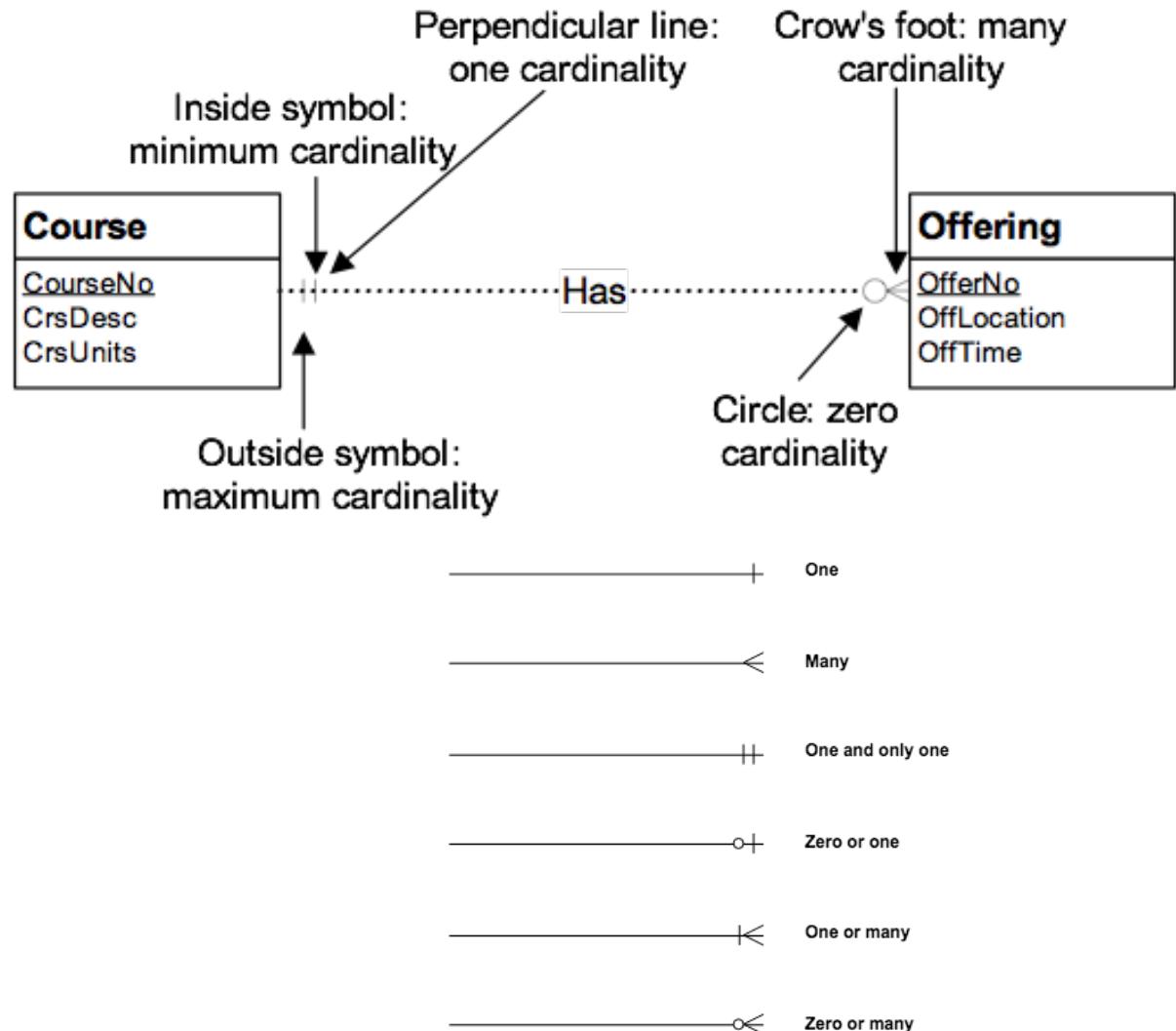


Cardinality of Relationships



Cardinality of Relationships

- Notice that:
 - The “1-1” cardinality next to the Course means that “Each Offering has at least one and at most one Course”
 - The “0-many” cardinality next to the Offering means that “Each Course may have zero offerings and may have many offerings”



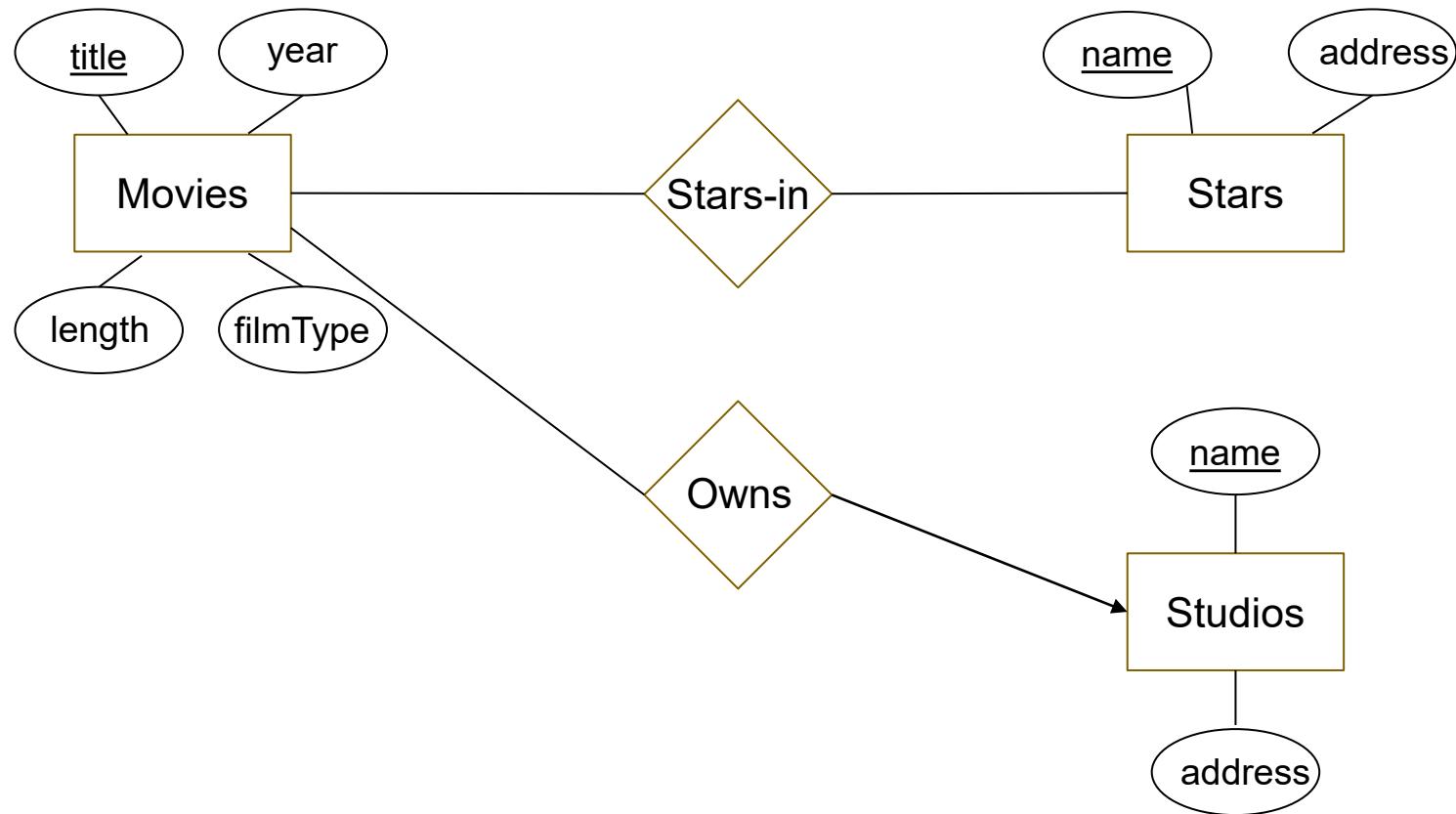
Notation for ER Diagrams

Figure 3.14
Summary of the
notation for ER
diagrams.

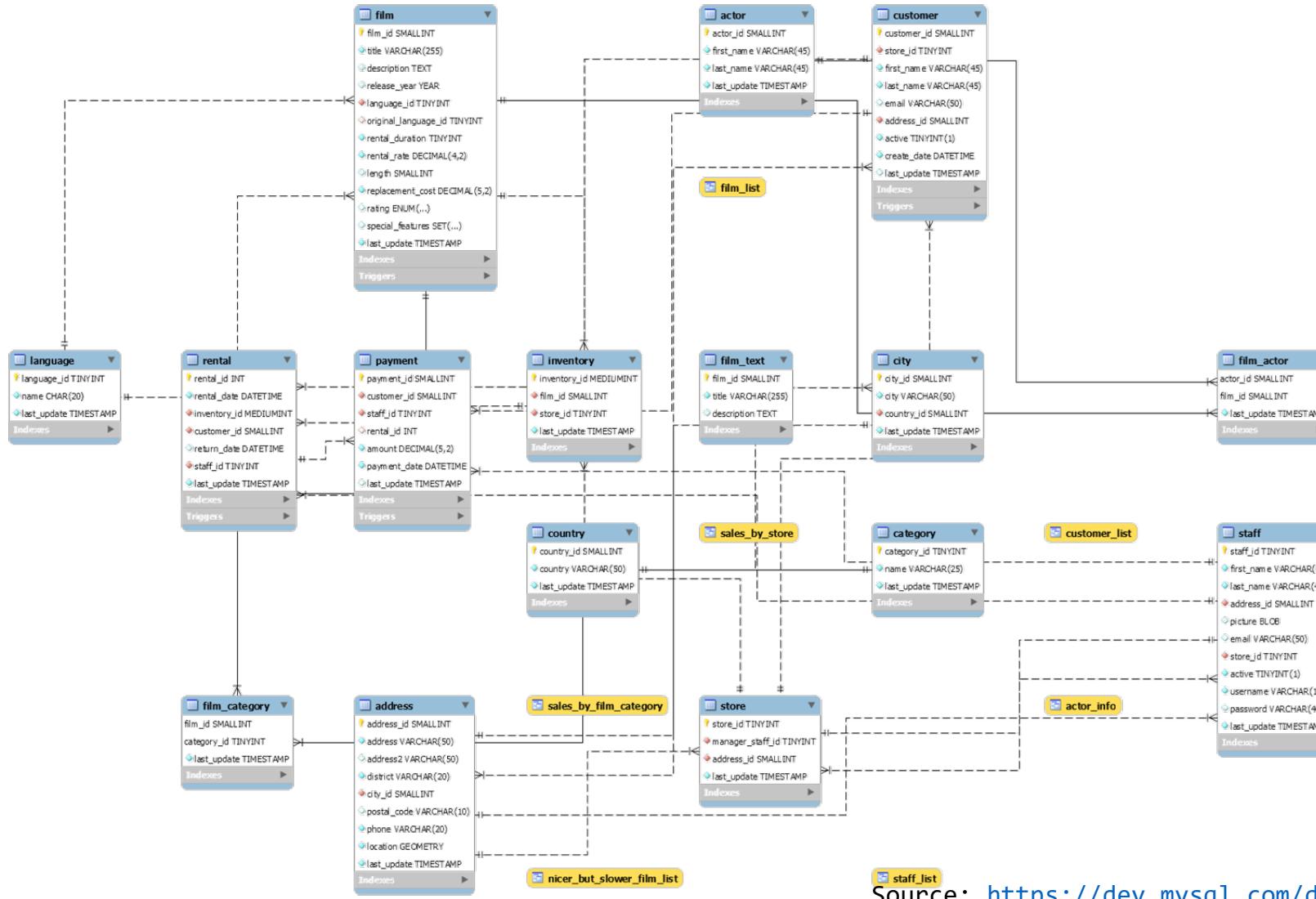
Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in R



Example ER Diagram: Movies (sakila)



Reverse Engineer (MySQL Workbench)



Source: <https://dev.mysql.com/doc/sakila/en/sakila-installation.html>

DB Concept Design: Coffee Shop!

Requirements analysis:

- What is going to be stored?
- How is it going to be used?
- What are we going to do with the data?
- Who should have access to the data?

Conceptual Design \ Modeling:

- A **high-level description** of the database
- Sufficiently **precise** that technical people can understand it
- But, not so precise that **non-technical** people can't participate

How will it be implemented in Excel?

Requirements
Analysis

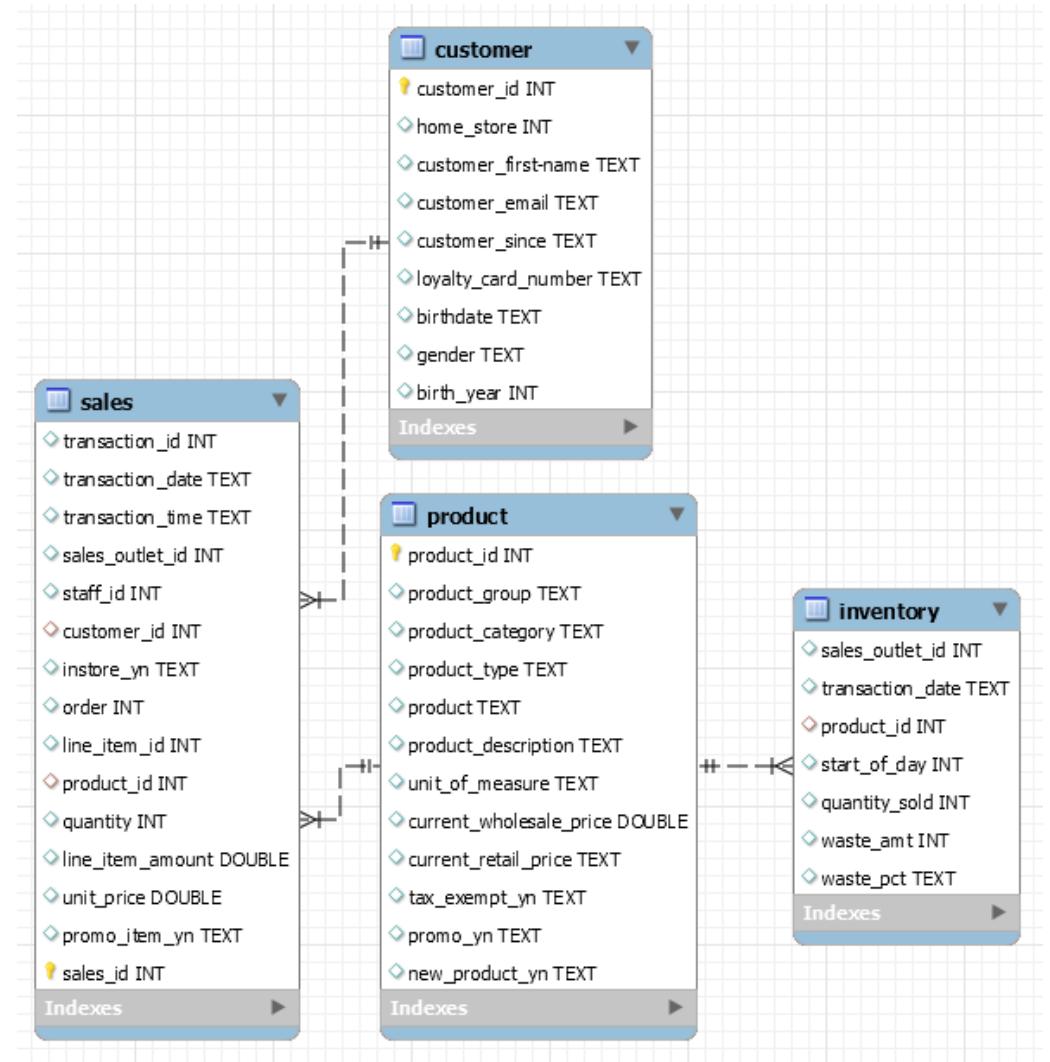
Conceptual
Design

Logical,
Physical,
Security,
etc.



ER Model: Coffee Shop!

- Entities:
 - Sales
 - Customer
 - Inventory
 - Product
- Relationships:
 - Sales - N:1 - product
 - Inventory - N:1 - product
 - Customer - 1:N - sales



DB Concept Design: Streaming Media

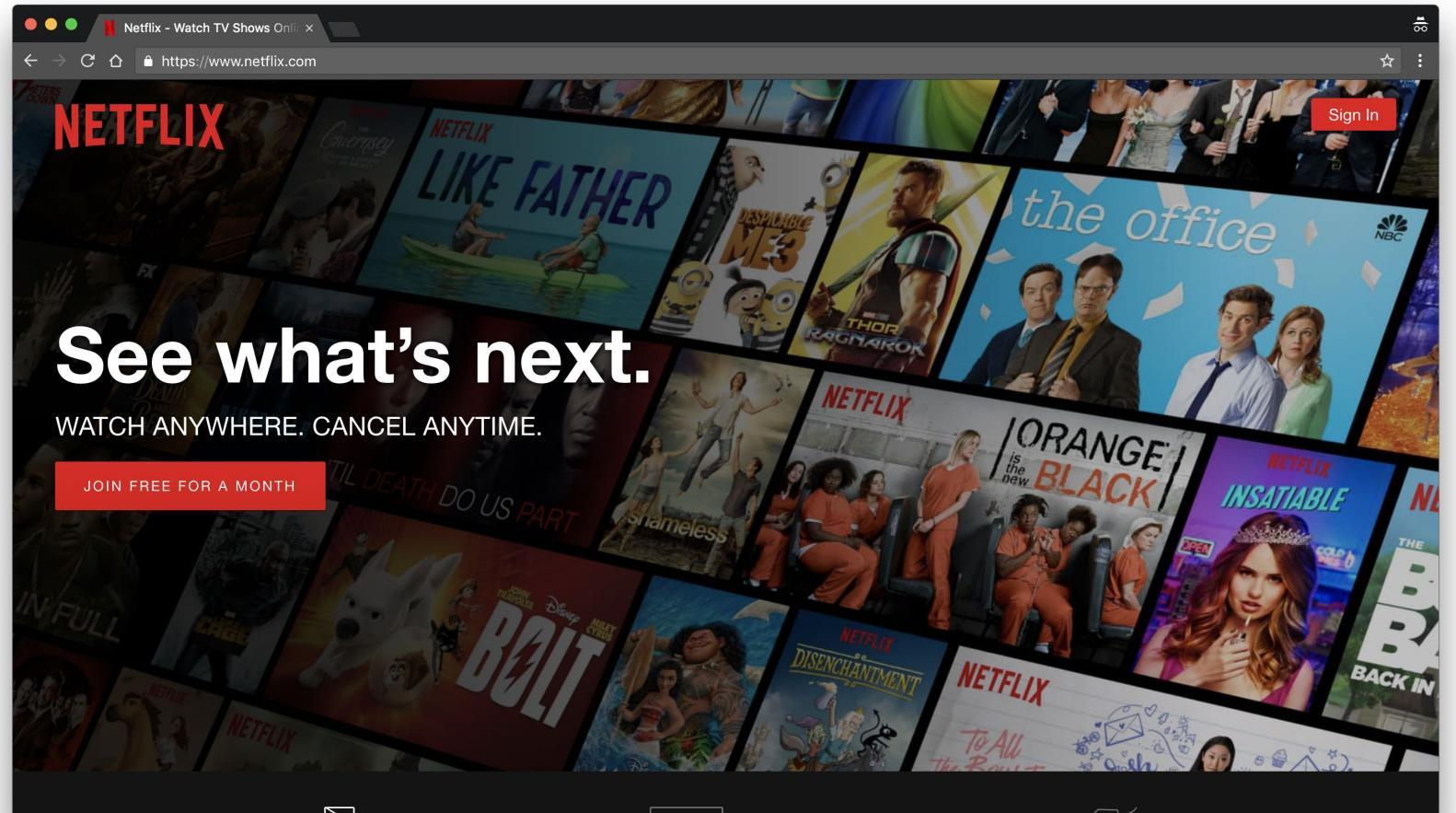
Try



Requirements
Analysis

Conceptual
Design

Logical,
Physical,
Security,
etc.



Rajiv Garg

Review

- An entity is a collection of objects with the same properties (attributes):
 - Students (student name, student id, age, sex, etc.)
 - Courses (course id, section id, course description, location, etc..)
- A primary key is an attribute whose value is unique in each instance:
 - Student id in “student” entity
- A relationship is an association among entities:
 - Students take courses (take is a relationship)
- Cardinalities describe the number of instances that participate in a relationship





DATA FLOW

DFD

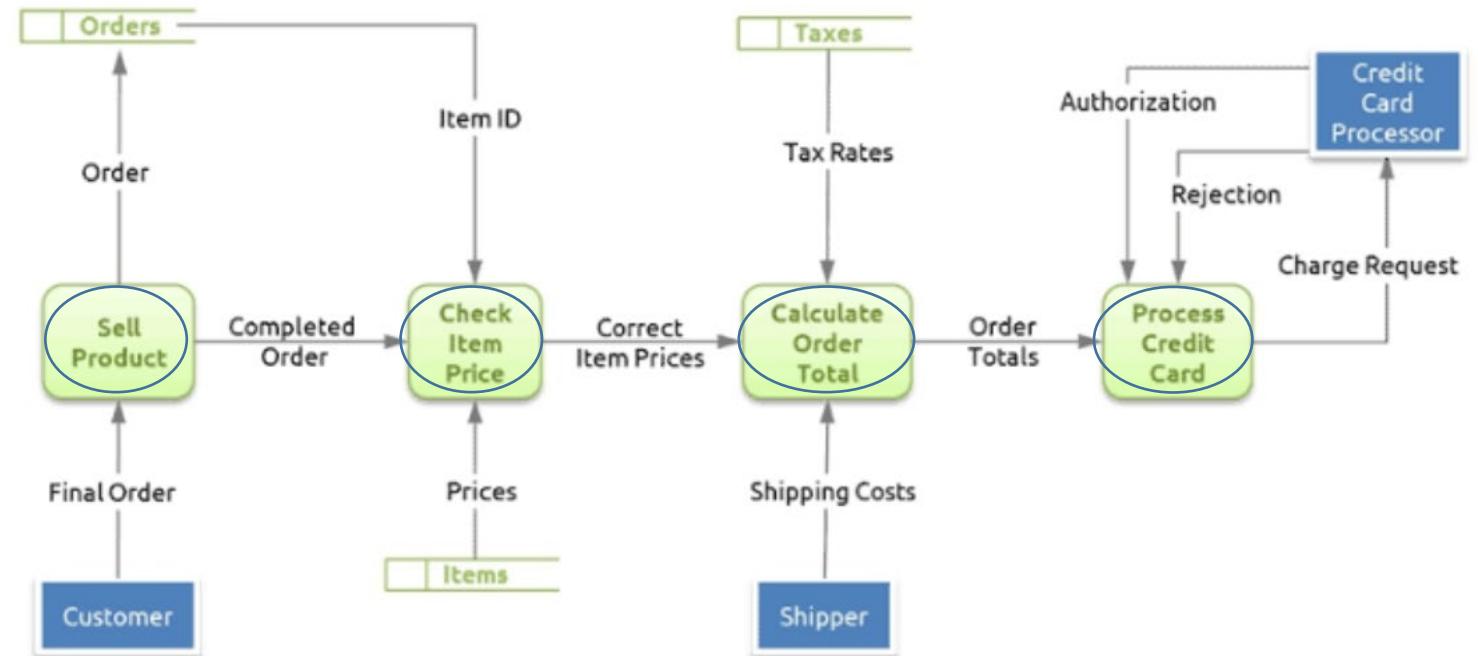
ERD vs DFD

- **Entity-Relationship (ER) Diagram** is a conceptual model that identifies the entities that exist in a system and the relationships between those entities.
- **Data Flow Diagram (DFD)** is a conceptual model of flow of data between different databases and entities. It describes the process of taking data as input, storing data, and providing data as output.



DFD Example: Online Retail Sales

- Customer adds products to cart
- Platform processes order and payment
- No standardized structure:
 - Circles: process
 - Arrows: data flow
 - Boxes: Entities

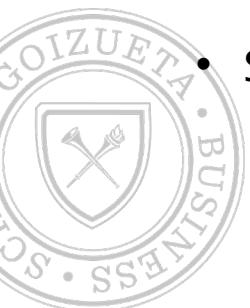


Hathaway and Hathaway, "Data Flow Diagrams" (<https://amzn.to/3pNRQx8>)



DFD

- Usually starts with the requirement analysis:
 - What are we going to do with the data?
- Iterative development and identification process:
 - Identify databases involved in a process
 - Identify data flow between any two databases
 - Identify entities within the databases that hold the information
 - Identify data flow between all entities within a database
 - Identify data flow between all entities within all databases
- Simplifies the development of databases and applications



DFD Design: Streaming Media

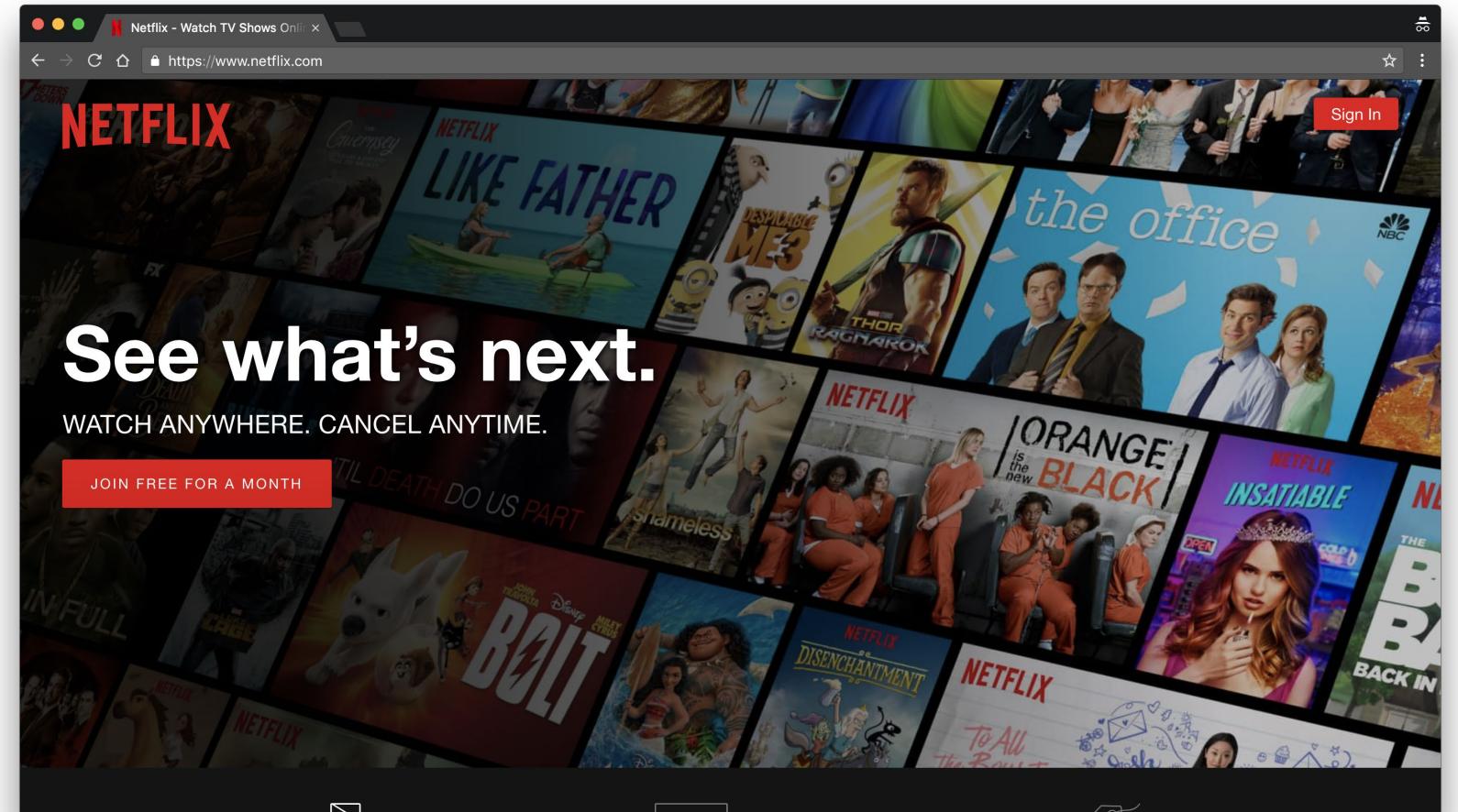
Requirements
Analysis

Conceptual
Design

Logical,
Physical,
Security,
etc.

Try

Before next class (connect
with ER model)



Rajiv Garg

Catchup

ER Model & Conceptual Design

August 31, 2023

Rajiv Garg



Database Design

Relational Databases

August 31, 2023

[This Photo](#) by Unknown Author is licensed under [CC BY](#)

Database Design Process

- Conceptual Design \ Modeling:
 - A **high-level** description of the database
 - Sufficiently **precise** that technical people can understand it
 - But, not so precise that **non-technical** people can't participate



- Requirements analysis:
 - What is going to be stored?
 - How is it going to be used?
 - What are we going to do with the data?
 - Who should have access to the data?
- Physical design and beyond:
 - Logical Database Design
 - Physical Database Design
 - Security Design



Logical Database Design Process

Designing a database structure



Identify

- Identify all attributes/data-elements (e.g., name, address)

Divide

- Subdivide each element into its smallest useful components (e.g., address = street, city, state, zip)

Tables

- Identify tables and assign columns (e.g., customer table, sales table)

Keys

- Identify the primary and foreign keys (e.g., order id, customer id)

Normalize

- Review whether the data structure is normalized

Index

- Identify the indexes



Identify data elements

- Depending on the nature of the system, data elements can be identified by:
 - Analyzing existing systems
 - Evaluating comparable systems
 - Documents in the real-world (e.g., invoice)
 - Interviewing users/stakeholders
- As you identify the data elements of a system, you should begin thinking about the entities that those elements are associated with.



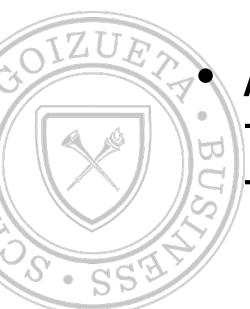
Divide data elements

- If a data element contains two or more components, you should consider subdividing the element into those components.
 - That way, you won't need to parse the element each time you use it.
 - e.g., address
- The extent to which you subdivide a data element depends on how it will be used.
 - Because it's difficult to predict all future uses for the data, most designers subdivide data elements as much as possible.
- When you subdivide a data element, you can easily rebuild it when necessary - by concatenating the individual components. ►►►



Identify **Tables** and assign columns

- After you identify and subdivide data elements for a database, you should group them by the entities with which they're associated.
 - These entities will later become the tables of the database, and the elements will become the columns.
- If a data element relates to more than one entity, you can include it under all entities it relates to.
 - Then, when you normalize the database, you may be able to remove the duplicate elements.
- As you assign the elements to entities, you should omit elements that aren't needed, and you should add any additional elements that are needed.



Identify Keys (primary and foreign)

- Each table should have a primary key that uniquely identifies each row.
 - If a suitable column doesn't exist for a primary key, you can create an ID column that increments by one for each new row as the primary key.
- If two tables have a one-to-many relationship, you may need to add a foreign key column to the table on the “many” side.
- If two tables have a many-to-many relationship, you'll need to define a linking table to relate them.
 - Then, each of the tables in the many-to-many relationship will have a one-to-many relationship with the linking table.
 - The linking table doesn't usually have a primary key.
- If two tables have a one-to-one relationship, they should be related by their primary keys.
 - This type of relationship is typically used to improve performance.
 - Columns with large amounts of data can be stored in a separate table.



Normalize if needed

- Redundancy
 - Information may be repeated unnecessarily in several tuples (rows).
- Insertion Anomalies
 - Adding new values for some attributes requires adding NULL or undefined values for other columns in that tuple.
- Update Anomalies
 - Changing information in one tuple leaves same information unchanged in another.
- Deletion Anomalies
 - If a set of values becomes empty, we may lose other information as a side effect.



Insertion anomaly

- Inability to insert a piece of information about an object without having to insert a (bogus) piece of information about something else
 - Example: Adding a new customer/book before it is ordered. How can you add the book “Harry Potter” in the file below?

Order #	Date	Customer ID	Last Name	First Name	Address	ISBN	Book Name	Author	Price
1	9/1/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0465039138	Code and other laws of cyberspace	Lessig, Lawrence	\$25.00
2	9/2/17	C1004	Sproull	Lee	NYU	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
3	9/3/17	C1002	Student	Pat	Emory Village	#0072952849	MIS in the Information Age	Haag, Stephen	\$98.75
4	9/4/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206679	Linked: The New Science of Networks	Barabasi, Albert-Laszlo	\$34.95
5	9/5/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
6	9/6/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
7	9/7/17	C1002	Student	Pat	Emory Village	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
8	9/8/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95



Update anomaly

- Need to change multiple times the same piece of information about an object
 - Example: Changing Jeff Bezos address in order 1 leaves orders 6 and 8 unchanged...

Order #	Date	Customer ID	Last Name	First Name	Address	ISBN	Book Name	Author	Price
1	9/1/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0465039138	Code and other laws of cyberspace	Lessig, Lawrence	\$25.00
2	9/2/17	C1004	Sproull	Lee	NYU	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
3	9/3/17	C1002	Student	Pat	Emory Village	#0072952849	MIS in the Information Age	Haag, Stephen	\$98.75
4	9/4/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206679	Linked: The New Science of Networks	Barabasi, Albert-Laszlo	\$34.95
5	9/5/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
6	9/6/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
7	9/7/17	C1002	Student	Pat	Emory Village	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
8	9/8/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95



Deletion anomaly

- The loss of a piece of information about one object when a piece of information about a different object is deleted
 - Example: Deleting order 2 → deleting customer Lee Sproull
 - Example: Deleting order 1 → deleting book “Code...”

Order #	Date	Customer ID	Last Name	First Name	Address	ISBN	Book Name	Author	Price
1	9/1/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0465039138	Code and other laws of cyberspace	Lessig, Lawrence	\$25.00
2	9/2/17	C1004	Sproull	Lee	NYU	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
3	9/3/17	C1002	Student	Pat	Emory Village	#0072952849	MIS in the Information Age	Haag, Stephen	\$98.75
4	9/4/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206679	Linked: The New Science of Networks	Barabasi, Albert-Laszlo	\$34.95
5	9/5/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
6	9/6/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
7	9/7/17	C1002	Student	Pat	Emory Village	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
8	9/8/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95



Normalization

- Normalization is a formal process of removing redundant and interdependent data from a database model
 - Data redundancy can cause storage and maintenance problems.
 - Normalization improves data consistency, maintainability, storage efficiency, and scalability.
 - In a normalized data structure, each table contains information about a single entity, and each piece of information is stored in exactly one place.
- The normalization process relies on the analysis of functional dependencies among fields.
- To normalize a data structure, you apply the normal forms in sequence.
 - Although there are seven normal forms, a data structure is typically considered normalized if the first three normal forms are applied.



The seven normal forms

Normal form	Description
First (1NF)	The value stored at the intersection of each row and column must be a scalar value - a table must not contain any repeating columns.
Second (2NF)	Every non-key column is functionally dependent on the primary key.
Third (3NF)	Every non-key column must depend <u>only on</u> the primary key.
Boyce-Codd (BCNF)	A non-key column can't be dependent on another non-key column.
Fourth (4NF)	A table must not have more than one multivalued dependency, where the primary key has a one-to-many relationship to non-key columns.
Fifth (5NF)	The data structure is split into smaller tables until all redundancy has been eliminated.
Domain-key (DKNF) or Sixth (6NF)	Every constraint on the relationship is dependent only on key constraints or domain constraints, where a domain is the set of allowable values for a column.

- Each normal form assumes that the database is already in the previous normal form.



Normalization

- Raw data:

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	201-01	211-02	214-01



Normalization – 1NF

- 1NF:
 - Eliminate repeating columns

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	201-01
4123	Smith	216	211-02
4123	Smith	216	214-01



Normalization – 2NF

- 2NF:
 - Redundant data and lack of functional dependency of non-key on key



Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	201-01
4123	Smith	216	211-02
4123	Smith	216	214-01



Normalization – 2NF

- 2NF:
 - Eliminate redundant data (non-functional dependency)

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01



Normalization – 3NF

- 3NF:

- Eliminate non-dependency on primary key

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01



Normalization – 3NF

BCNF: If advisor rooms (21x and 41x) are all in department 42, then the BCNF can be achieved by creating a new relationship table with room and dept.

- 3NF:
 - Eliminate data not dependent on key

Student#	Advisor
1022	Jones
4123	Smith

Name	Room	Dept
Jones	412	42
Smith	216	42

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01



Example 1

RAW DATA:

StudentID	Name	Advisor	AdvOffice	Class1	Class2	Class3
1011	Amy	Garg	420	ISOM671	ISOM672	ISOM673
1012	Becky	Chellappa	400	ISOM672	ISOM675	
1013	Chris	Todri	410	ISOM671	ISOM673	ISOM674
1014	David	Garg	420	ISOM671	ISOM672	ISOM675

```
## 1-NF
SELECT StudentID, `Name`, Advisor, AdvOffice, Class1 FROM enrollment
UNION ALL
SELECT StudentID, `Name`, Advisor, AdvOffice, Class2 FROM enrollment
UNION ALL
SELECT StudentID, `Name`, Advisor, AdvOffice, Class3 FROM enrollment;
```

```
## 2-NF
SELECT DISTINCT StudentID, `Name`, Advisor,
AdvOffice FROM enrollment;

SELECT StudentID, Class1 FROM enrollment
UNION ALL
SELECT StudentID, Class2 FROM enrollment
UNION ALL
SELECT StudentID, Class3 FROM enrollment;
```

```
## 3-NF
SELECT DISTINCT StudentID, `Name`, Advisor FROM enrollment;
SELECT DISTINCT Advisor, AdvOffice FROM enrollment;
```



Example 2 (try)

Order #	Date	Customer ID	Last Name	First Name	Address	ISBN	Book Name	Author	Price
1	9/1/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0465039138	Code and other laws of cyberspace	Lessig, Lawrence	\$25.00
2	9/2/17	C1004	Sproull	Lee	NYU	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
3	9/3/17	C1002	Student	Pat	Emory Village	#0072952849	MIS in the Information Age	Haag, Stephen	\$98.75
4	9/4/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206679	Linked: The New Science of Networks	Barabasi, Albert-Laszlo	\$34.95
5	9/5/17	C1003	Gates	Bill	Microsoft, Redmond	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
6	9/6/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95
7	9/7/17	C1002	Student	Pat	Emory Village	#1573928895	Digital Copyright: Protecting Intellectual Property	Litman, Jessica	\$55.00
8	9/8/17	C1001	Bezos	Jeff	1 Amazon Plaza	#0738206083	Smart Mobs: The Social Revolution	Rheingold, Howard	\$29.95



Example 2 (try)

- Copy data table into a CSV file
- Load and normalize CSV data in mySQL
 - Import csv in mySQL
 - Analyze the data structure and browse data
 - Identify ER Model with tables, columns, keys
 - Normalize using SQL:
 - Create new tables
 - Insert data in those tables



Benefits of Normalization

- A normalized database has more indexes. This makes data retrieval more efficient.
 - A normalized database has more tables than an un-normalized database, and each table has an index on its primary key.
- Each index has fewer columns (usually one) and fewer rows. This makes data retrieval and insert, update, and delete operations more efficient.
- Each table has fewer indexes, which makes insert, update, and delete operations more efficient.
- Data redundancy is minimized, which simplifies maintenance and reduces storage.



Disadvantages of Normalization

- More tables (typically)
- Query complexity (due to multiple table joins)
- Slower retrievals (occasionally because of joins)



When to de-normalize a database

- When a column from a joined table is used repeatedly in search criteria
 - you could consider moving that column to the primary key table if it eliminates the need for a join.
- If a table is updated infrequently, you should consider de-normalizing it to improve efficiency.
 - Because the data remains relatively constant, you don't have to worry about data redundancy errors once the initial data is entered and verified.
- Include columns with derived values when those values are used frequently in search conditions.



Indexing: identifying columns

- Indexes for primary keys and foreign keys are automatically created.
- An index provides a way for a database management system to locate information more quickly.
 - Indexes speed performance when searching and joining tables.
- You should use an index when:
 - When the column is used frequently in search conditions or joins.
 - When the column contains a large number of distinct values.
 - When the column is updated infrequently.

Because indexes must be updated each time you add, update, or delete a row, you shouldn't create more indexes than you need.



Indexing

- Easily lookup by customer name or order location
 - How many orders from Chicago?
 - How many orders by Jones?

```
CREATE TABLE ft (fname
VARCHAR(10), lname
VARCHAR(20), INDEX
(fname, lname(10)) );
```

Index Table

Secondary Key (Town)	Customer Reference (ID)
Chicago	ID: 5
Chicago	ID: 9
Chicago	ID: 1000
...	...
Portland	ID: 3
Portland	ID: 7
Redmond	ID: 1
Redmond	ID: 4
Redmond	ID: 6
Redmond	ID: 8
Seattle	ID: 2
...	...

Primary Key (Customer ID)	Customer Data
1	LastName: Smith, Town: Redmond, ...
2	LastName: Jones, Town: Seattle, ...
3	LastName: Robinson, Town: Portland, ...
4	LastName: Brown, Town: Redmond, ...
5	LastName: Smith, Town: Chicago, ...
6	LastName: Green, Town: Redmond, ...
7	LastName: Clarke, Town: Portland, ...
8	LastName: Smith, Town: Redmond, ...
9	LastName: Jones, Town: Chicago, ...
...	...
1000	LastName: Clarke, Town: Chicago, ...
...	...

Fact Table

Primary Key (Customer ID) Customer Data

1	LastName: Smith, Town: Redmond, ...
2	LastName: Jones, Town: Seattle, ...
3	LastName: Robinson, Town: Portland, ...
4	LastName: Brown, Town: Redmond, ...
5	LastName: Smith, Town: Chicago, ...
6	LastName: Green, Town: Redmond, ...
7	LastName: Clarke, Town: Portland, ...
8	LastName: Smith, Town: Redmond, ...
9	LastName: Jones, Town: Chicago, ...
...	...
1000	LastName: Clarke, Town: Chicago, ...
...	...

Index Table

Secondary Key (LastName)	Customer Reference (ID)
Brown	ID: 4
Clarke	ID: 7
Clarke	ID: 1000
Green	ID: 6
Jones	ID: 2
Jones	ID: 9
...	...
Robinson	ID: 3
Smith	ID: 1
Smith	ID: 5
Smith	ID: 8
...	...



Terms to know

- Relational database
 - Schema
- Table
- Column
- Row
- Cell
- Primary key
- Composite primary key
 - Combination of two or more columns as a primary key
- Unique key (non-primary key)
 - used to prevent duplicate values in a column
- Index

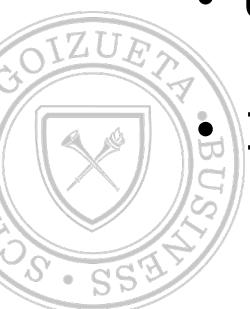
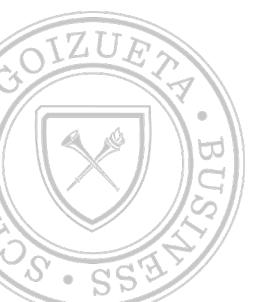
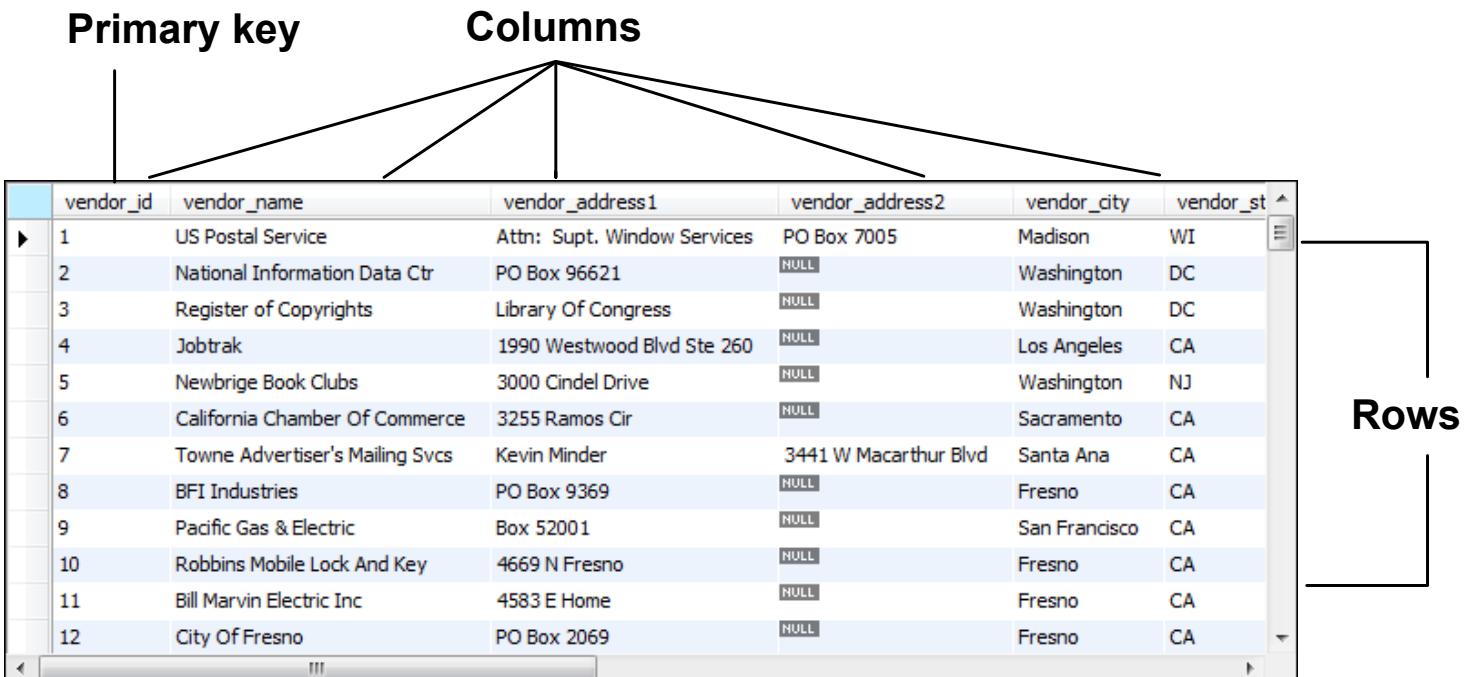


Table in a relational database

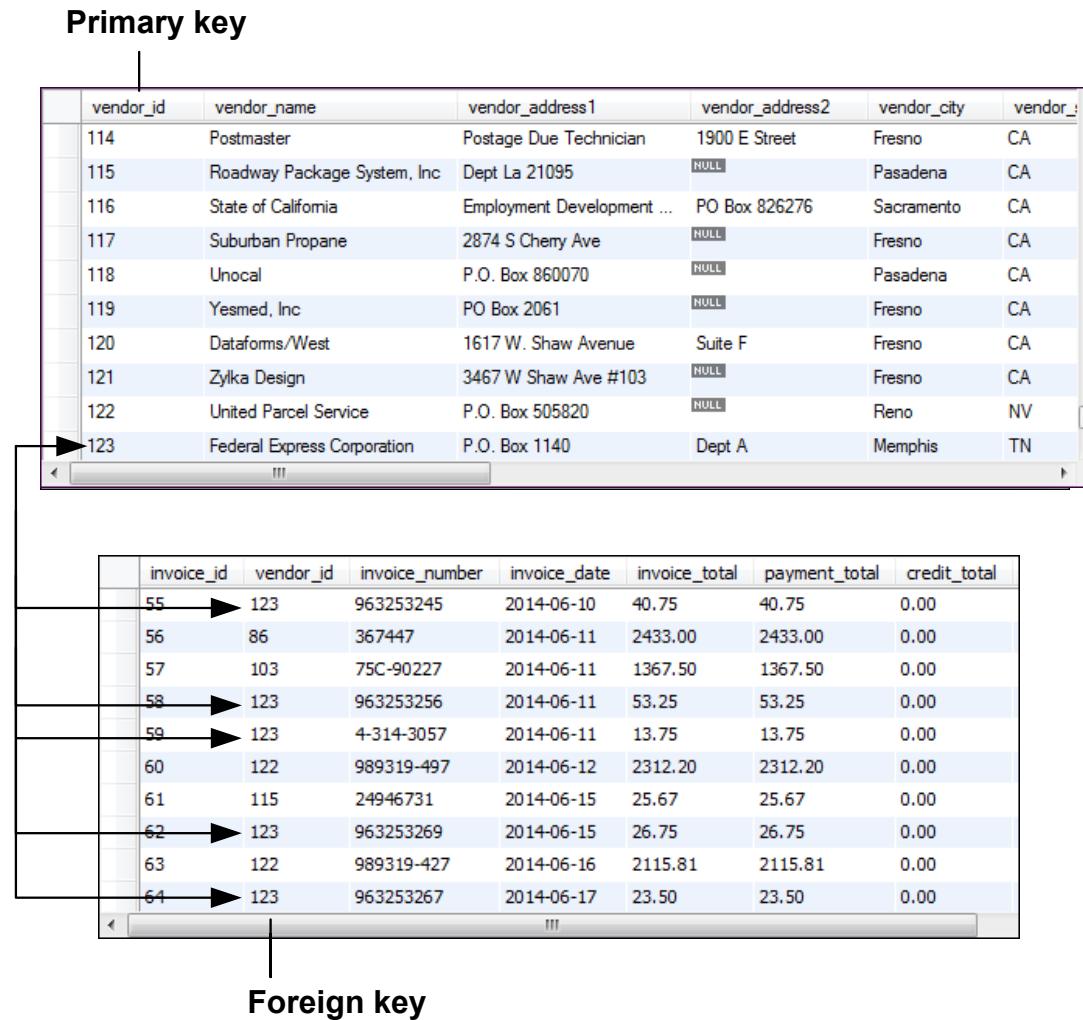


Terms to know

- Foreign key
 - Column in a table whose value corresponds to primary key in another table
- Referential integrity
 - Referential integrity is the logical dependency of a foreign key on a primary key - if you destroy a PK, you destroy the meaning of rows that use that PK as FK
- One-to-many relationship
- One-to-one relationship
- Many-to-many relationship

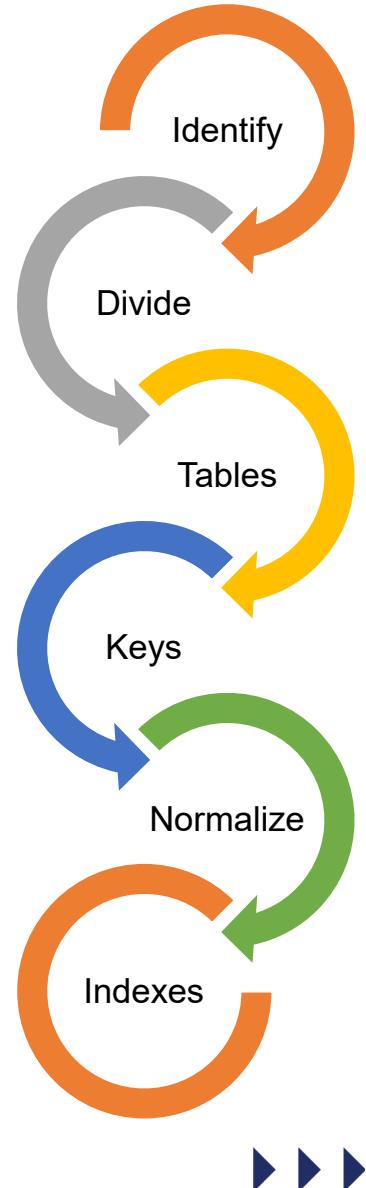


How tables are related



Design – Airbnb database!

- Create the conceptual design (ER model)
- Create the logical design (schema/tables)



Catchup

Normalization

September 7, 2023

Terms to know

- Relational database
 - Schema
- Table
- Column
- Row
- Cell
- Primary key
- Composite primary key
 - Combination of two or more columns as a primary key
- Unique key (non-primary key)
 - used to prevent duplicate values in a column
- Index

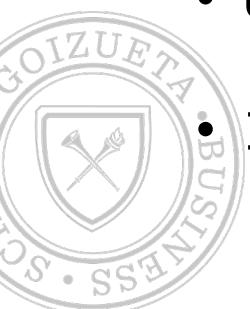
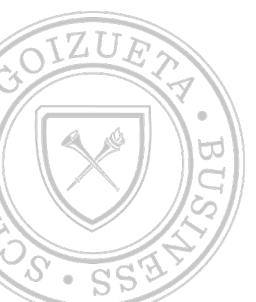
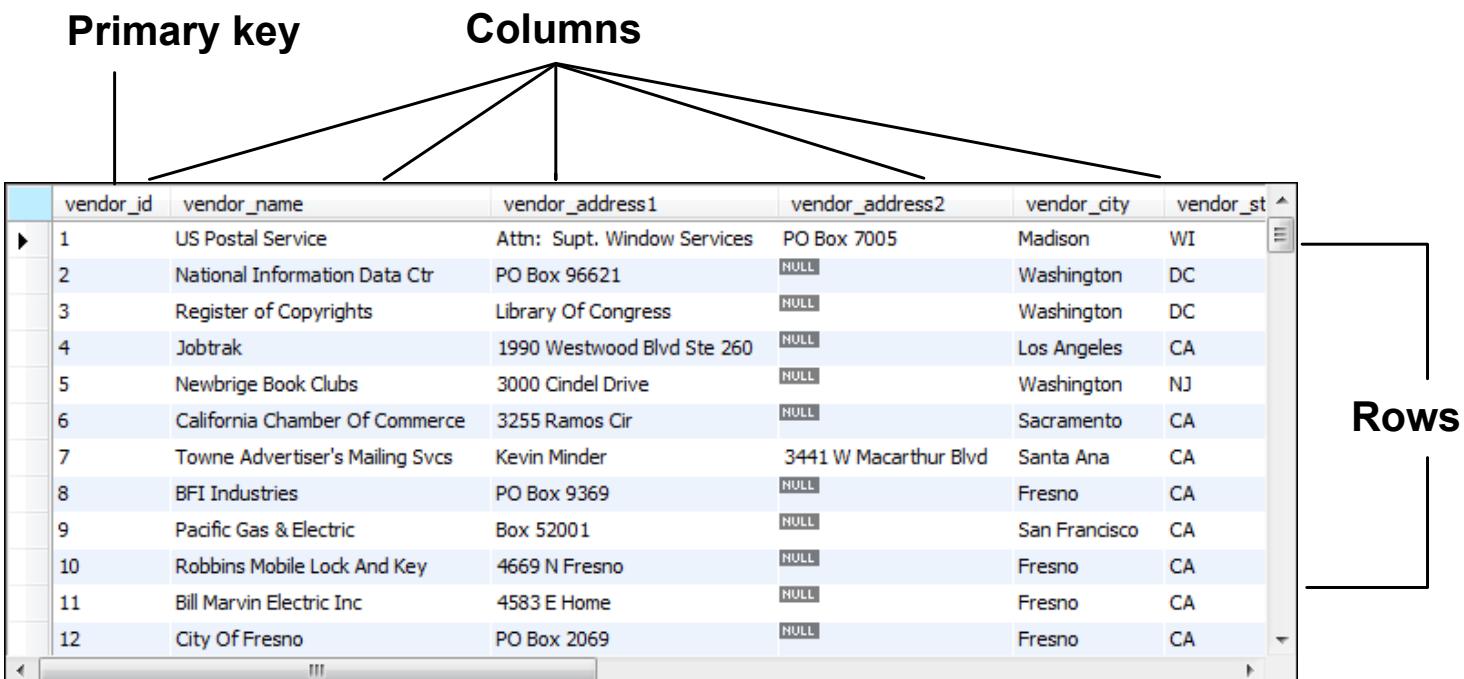


Table in a relational database

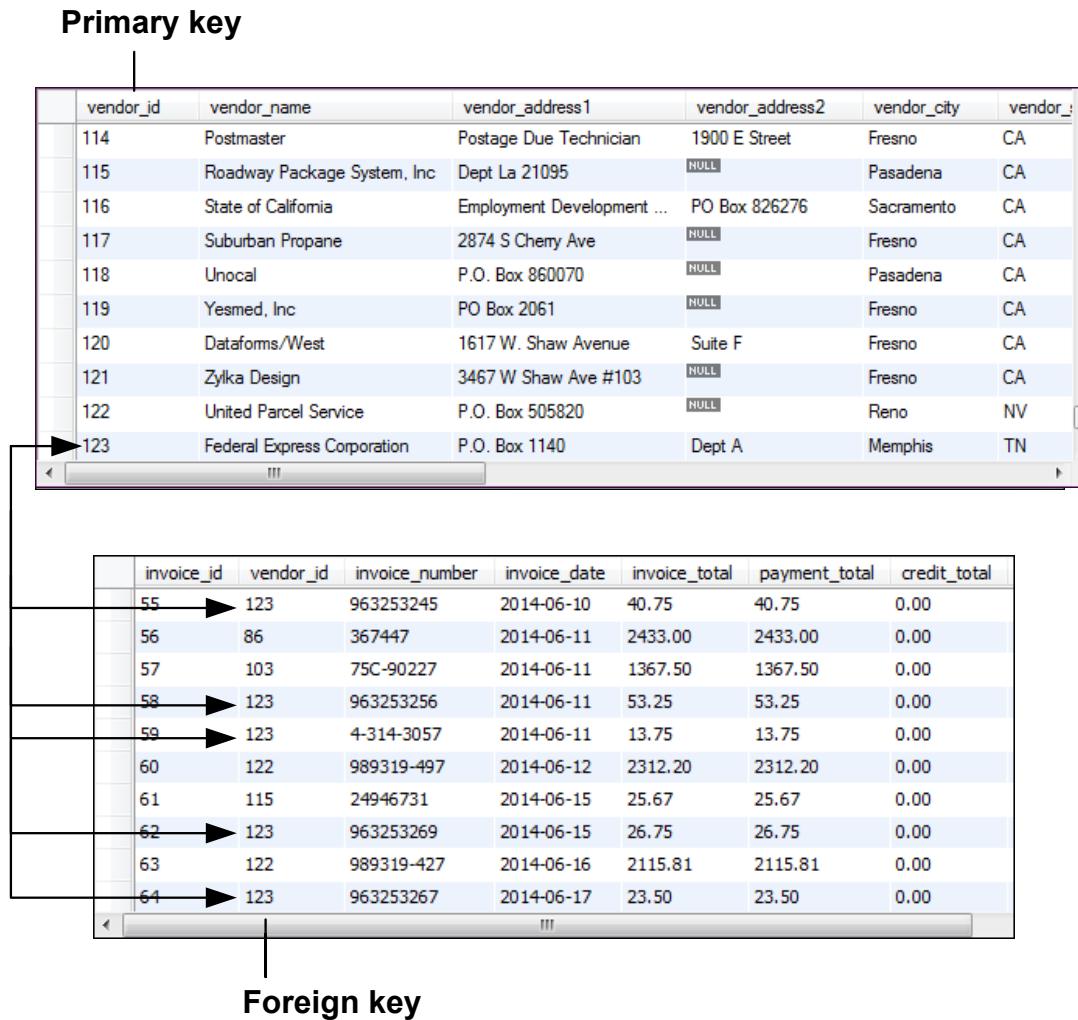


Terms to know

- Foreign key
 - Column in a table whose value corresponds to primary key in another table
- Referential integrity
 - Referential integrity is the logical dependency of a foreign key on a primary key - if you destroy a PK, you destroy the meaning of rows that use that PK as FK
- One-to-many relationship
- One-to-one relationship
- Many-to-many relationship



How tables are related





SQL

Structured Query Language

September 7, 2023

[This Photo](#) by Unknown Author is licensed under [CC BY](#)

SQL

- What is SQL?
 - Higher level language to query, modify, and create databases
 - Resulted from relational database model proposed by Edgar Codd (IBM researcher) in 1969
- Why SQL?
 - Bigger (can handle increasing volume of data - both structured and NoSQL)
 - Faster (binary storage, file splitting, indexing)
 - Cheaper (open source, ODBC drivers, platform agnostic)



SQL

- Structured Query Language (SQL) is a standard language for relational database management systems (RDBMS). It specifies the syntax for data definition and manipulation:
- Data Definition Language (DDL)
 - Defines the schema
 - Table – CREATE, ALTER, DROP
 - User – ADD, ALTER, DROP
 - View – CREATE
 - Index – CREATE
 - CONSTRAINT (primary key, unique key)
- Data Manipulation Language (DML)
 - Modifies and queries the data
 - Data – INSERT INTO, DELETE, UPDATE
 - SELECT
 - JOIN – inner, left, right

For more information: <https://docs.microsoft.com/en-us/office/client-developer/access/desktop-database-reference/microsoft-access-sql-reference>



SQL: Add Data

- Load from external source:
 - Import (e.g., csv, excel, etc.)
- Programmatically (e.g., in mySQL, Python, Excel, etc.):
 - Using DB software/command-line:
 - Create table
 - Insert data
 - Using ODBC and other connecting applications



SQL: Load Data (csv in mySQL)



The screenshot shows the SSMS interface. In the top navigation bar, the database 'class1' is selected. The 'File' menu is open. In the 'Navigator' pane, the 'temp' schema is expanded, showing tables like 'customers', 'housing', 'orders', and 'products'. A context menu is open over the 'housing' table, with the 'Table Data Import Wizard' option highlighted and surrounded by a red box.

```
1 • USE temp;
2 • CREATE TABLE `temp`.`housing` (`NeighborhoodID` int, `StreetAddress` varchar(50), `City` varchar(50), `State` varchar(50), `PostalCode` varchar(50), `Region` varchar(50));
3 #PREPARE stmt FROM 'INSERT INTO `temp`.`housing` VALUES (?, ?, ?, ?, ?, ?)';
4 #DEALLOCATE PREPARE stmt;
5 • LOAD DATA LOCAL INFILE "housing.csv"
6 INTO TABLE housing
7 FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
8 IGNORE 1 ROWS;
SHOW VARIABLES LIKE "secure_file_priv";
```

The screenshot shows the 'Table Data Import' wizard. The title bar says 'Table Data Import'. The main area is titled 'Select Destination' with the sub-instruction 'Select destination table and additional options.' Below it, there is a radio button group for 'Use existing table:' followed by a dropdown menu set to 'temp.housing'. There are also options for 'Create new table:' and 'Truncate table before import'. At the bottom right are buttons for '< Back', 'Next >', and 'Cancel'.

SQL - CREATE TABLE

CREATE TABLE

```
{ database_name.schema_name.table_name | schema_name.table_name | table_name }
({  <column_definition>
    | <computed_column_definition>
    | <column_set_definition>
    | [ <table_constraint> ] [ ,... n ]
    | [ <table_index> ] }
[ ,...n ] )
[ ; ]
```



More details: <https://docs.microsoft.com/en-us/sql/statements/create-table-transact-sql?view=sql-server-ver15>

SQL - CREATE TABLE

```
CREATE TABLE customers (
    customer_id INT,
    email_address CHAR(64),
    password CHAR(64),
    fName CHAR(32),
    lName CHAR(32),
    address_billing INT,
    address_shipping INT,
    CONSTRAINT pk_customer PRIMARY KEY (customer_id));
```

Primary Key



SQL - CREATE TABLE

```
CREATE TABLE products (
    product_id int NOT NULL AUTO_INCREMENT,
    category_id int,
    product_code text,
    product_name char(64),
    list_price float,
    discount_percent float,
    date_added datetime,
    CONSTRAINT pk_products PRIMARY KEY (product_id));
```



SQL - CREATE TABLE

```
CREATE TABLE orders (
    customer_id INT,
    product_id INT,
    card_type ENUM('Visa', 'Master', 'Amex'),
    CONSTRAINT pk_order PRIMARY KEY (customer_id, product_id),
    CONSTRAINT fk_customer FOREIGN KEY (customer_id)
        REFERENCES customers (customer_id),
    CONSTRAINT fk_product FOREIGN KEY (product_id)
        REFERENCES products (product_id)
);
```

Would be **composite key** if these were not foreign keys

Compound Key

CONSTRAINT for FK needs REFERENCES



SQL - DATA TYPES

String data types:

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBS (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Date and Time data types:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(fsp)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(fsp)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(fsp)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Numeric data types:

Data type	Description
BIT(size)	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
TINYINT(size)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(size)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)
MEDIUMINT(size)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)
INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
INTEGER(size)	Equal to INT(size)
BIGINT(size)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)
FLOAT(size, d)	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(p)	A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter
DOUBLE PRECISION(size, d)	
DECIMAL(size, d)	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.



Source: https://www.w3schools.com/sql/sql_datatypes.asp



SQL – LOAD DATA

- `LOAD DATA INFILE 'data.csv' INTO TABLE dbname.my_table;`
 - May give security error (need to edit secure-file-priv setting in `my.ini` file)

OR

- `LOAD DATA INFILE 'data.csv' INTO TABLE dbname.my_table;`
 - May give restriction error (need to add `OPT_LOCAL_INFILE=1` in advanced connection settings)
- You may also be required to specify file structure (separator and eol character):
 - `LOAD DATA INFILE 'data.csv' INTO TABLE dbname.my_table
FIELDS TERMINATED BY "," LINES TERMINATED BY "\n";`



More details: <https://dev.mysql.com/doc/refman/8.0/en/load-data.html>

SQL – DROP

- DROP database:
- DROP TABLE customers;



SQL – ALTER TABLE

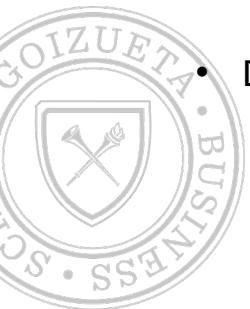
- **ALTER TABLE table {ADD {COLUMN field type[(size)] [NOT NULL] [CONSTRAINT index]} | ALTER COLUMN field type[(size)] | CONSTRAINT multifieldindex} | DROP {COLUMN field I CONSTRAINT indexname} }**
- **ALTER TABLE customer ADD COLUMN Notes TEXT;**
- **INSERT INTO customer (`Notes`) VALUES ('52');**
- **ALTER TABLE customer MODIFY Notes INT;**
- **ALTER TABLE customer DROP COLUMN Notes;**



SQL – ALTER TABLE (ADD Keys)

- Import CSV data files using wizard
- Create primary keys:
 - `ALTER TABLE customer ADD CONSTRAINT pk_customer PRIMARY KEY(customer_id);`
 - `ALTER TABLE product ADD PRIMARY KEY(product_id);`
 - `ALTER TABLE sales ADD PRIMARY KEY(transaction_id);`
- Create foreign keys:
 - `ALTER TABLE inventory ADD CONSTRAINT fk_inv_pid FOREIGN KEY (product_id) REFERENCES product(product_id);`
 - `ALTER TABLE sales ADD CONSTRAINT fk_pid FOREIGN KEY (product_id) REFERENCES product(product_id);`
 - `ALTER TABLE sales ADD CONSTRAINT fk_cid FOREIGN KEY (customer_id) REFERENCES customer(customer_id);`
- Drop keys:
 - `ALTER TABLE product DROP PRIMARY KEY;`
 - `ALTER TABLE orders DROP CONSTRAINT fk_product;`

“CONSTRAINT pk_customer”
is optional (by default, PRIMARY
KEY is a constraint)



SQL – TABLE INFORMATION

- The TABLE_CONSTRAINTS table describes which tables have constraints.
- SELECT *
- FROM information_schema.table_constraints
- WHERE table_schema="isom671";

CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	TABLE_SCHEMA	TABLE_NAME	CONSTRAINT_TYPE	ENFORCED
def	isom671	PRIMARY	isom671	customers	PRIMARY KEY	YES
def	isom671	PRIMARY	isom671	orders	PRIMARY KEY	YES
def	isom671	fk_customer	isom671	orders	FOREIGN KEY	YES
def	isom671	fk_product	isom671	orders	FOREIGN KEY	YES
def	isom671	PRIMARY	isom671	product	PRIMARY KEY	YES
def	isom671	PRIMARY	isom671	products	PRIMARY KEY	YES



- Ref:
 - <https://dev.mysql.com/doc/refman/8.0/en/information-schema-introduction.html>
 - <https://dev.mysql.com/doc/refman/8.0/en/information-schema-table-constraints-table.html>



Schema - Coffee Shop



SQL - DML

INSERT, DELETE, UPDATE



Rajiv Garg

September 7, 2023

28

SQL - INSERT (new rows)

- You can use the INSERT statement to add one or more rows to a table.
- In the INSERT clause, you specify the name of the table that you want to add a row to, along with an optional column list.
 - The INTO keyword is also optional.
- In the VALUES clause, you specify the values to be inserted.
- If you don't include a column list in the INSERT clause, you must specify the column values in the same order as in the table, and you must code a Value for each column.



SQL – INSERT Data

- ```
INSERT INTO customers VALUES
(1,'allan.sherwood@yahoo.com','650215acec746f0e32bdfff387439eefc1358737','Allan'
'Sherwood',1,2);
```
- ```
INSERT INTO customers VALUES  
(2,'barryz@gmail.com','3f563468d42a448cb1e56924529f6e7bbe529cc7','Barry','Zimmer'  
,3,3);
```
- ```
INSERT INTO customers VALUES
(3,'christineb@solarone.com','ed19f5c0833094026a2f1e9e6f08a35d26037066','Christi
ne','Brown',4,4)
```



# SQL - INSERT (new rows) Examples

## The syntax of the INSERT statement

```
INSERT [INTO] table_name [(column_list)]
VALUES (expression_1[, expression_2]...)[,
(expression_1[, expression_2]...)...]
```

## The column definitions for the Invoices table

|                  |              |           |                 |
|------------------|--------------|-----------|-----------------|
| invoice_id       | INT          | NOT NULL  | AUTO_INCREMENT, |
| vendor_id        | INT          | NOT NULL, |                 |
| invoice_number   | VARCHAR(50)  | NOT NULL, |                 |
| invoice_date     | DATE         | NOT NULL, |                 |
| invoice_total    | DECIMAL(9,2) | NOT NULL, |                 |
| payment_total    | DECIMAL(9,2) | NOT NULL  | DEFAULT 0,      |
| credit_total     | DECIMAL(9,2) | NOT NULL  | DEFAULT 0,      |
| terms_id         | INT          | NOT NULL, |                 |
| invoice_due_date | DATE         | NOT NULL, |                 |
| payment_date     | DATE         |           |                 |

## Insert a single row without using a column list

```
INSERT INTO invoices VALUES
(115, 97, '456789', '2011-08-01', 8344.50, 0, 0, 1, '2011-08-31', NULL)
(1 row affected)
```

## Insert a single row using a column list

```
INSERT INTO invoices
(vendor_id, invoice_number, invoice_total, terms_id, invoice_date,
 invoice_due_date)
VALUES
(97, '456789', 8344.50, 1, '2011-08-01', '2011-08-31')
(1 row affected)
```

## Insert multiple rows

```
INSERT INTO invoices VALUES
(116, 97, '456701', '2011-08-02', 270.50, 0, 0, 1, '2011-09-01', NULL),
(117, 97, '456791', '2011-08-03', 4390.00, 0, 0, 1, '2011-09-02', NULL),
(118, 97, '456792', '2011-08-03', 565.60, 0, 0, 1, '2011-09-02', NULL)
```



# SQL - INSERT (NULL and DEFAULT)

- If a column is defined so it allows null values, you can use the **NULL** keyword in the list of values to insert a null value into that column.
- If a column is defined with a default (or auto increment) value, you can use the **DEFAULT** keyword in the list of values to insert the default (or generate the next) value for that column.
- If you include a column list, you can omit columns with default values, auto increment and null values. Then, the corresponding value is assigned automatically.



# SQL - INSERT (NULL and DEFAULT)

The column definitions for the Color\_Sample table

```
color_id INT NOT NULL AUTO_INCREMENT,
color_number INT NOT NULL DEFAULT 0,
color_name VARCHAR(50)
```

Five INSERT statements for the Color\_Sample table

```
INSERT INTO color_sample (color_number)
VALUES (606)
```

```
INSERT INTO color_sample (color_name)
VALUES ('Yellow')
```

```
INSERT INTO color_sample
VALUES (DEFAULT, DEFAULT, 'Orange')
```

```
INSERT INTO color_sample
VALUES (DEFAULT, 808, NULL)
```

```
INSERT INTO color_sample
VALUES (DEFAULT, DEFAULT, NULL)
```

The Color\_Sample table after the rows have been inserted

|   | color_id | color_number | color_name |
|---|----------|--------------|------------|
| ▶ | 1        | 606          | NULL       |
|   | 2        | 0            | Yellow     |
|   | 3        | 0            | Orange     |
|   | 4        | 808          | NULL       |
|   | 5        | 0            | NULL       |



# SQL - INSERT (subquery)

- To insert rows selected from one or more tables into another table, you can code a subquery in place of the VALUES clause.
  - Then, MySQL inserts the rows returned by the subquery into the target table. For this to work, the target table must already exist.
- The rules for working with a column list are the same as they are for any INSERT statement.



# SQL - INSERT (subquery)

**The syntax for using a subquery to insert one or more rows**

```
INSERT [INTO] table_name [(column_list)] select_statement
```

**Insert paid invoices into the Invoice\_Archive table**

```
INSERT INTO invoice_archive
SELECT *
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0
(103 rows affected)
```

**The same statement with a column list**

```
INSERT INTO invoice_archive
 (invoice_id, vendor_id, invoice_number, invoice_total, credit_total,
 payment_total, terms_id, invoice_date, invoice_due_date)
SELECT
 invoice_id, vendor_id, invoice_number, invoice_total, credit_total,
 payment_total, terms_id, invoice_date, invoice_due_date
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0
(103 rows affected)
```



# SQL - UPDATE (existing rows)

- You can use the UPDATE statement to modify one or more rows in a table.
- In the SET clause, you name each column and its new value.
  - You can specify the value for a column as a literal or an expression.
  - You can also use the DEFAULT and NULL keywords to specify default and null values.
- In the WHERE clause, you can specify the conditions that must be met for a row to be updated.
- By default, MySQL Workbench runs in safe update mode.
  - That prevents you from updating rows if the WHERE clause is omitted or doesn't refer to a primary key or foreign key column.



# SQL - UPDATE (existing rows)

## The syntax of the UPDATE statement

```
UPDATE table_name
SET column_name_1 = expression_1[, column_name_2 = expression_2]...
[WHERE search_condition]
```

### Update two columns for a single row

```
UPDATE invoices
SET payment_date = '2011-09-21',
 payment_total = 19351.18
WHERE invoice_number = '97/522'

(1 row affected)
```

### Update one column for multiple rows

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id = 95

(6 rows affected)
```

### Update one column for one row

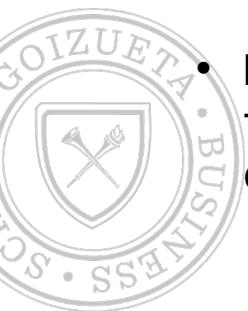
```
UPDATE invoices
SET credit_total = credit_total + 100
WHERE invoice_number = '97/522'

(1 row affected)
```



# SQL – UPDATE Data

- `UPDATE customers SET lName = 'Green' WHERE fName = 'Christine';`
- Note: Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in **Preferences -> SQL Editor** and reconnect.



# SQL - DELETE (existing rows)

- You can use the DELETE statement to delete one or more rows from the table you name in the DELETE clause.
- You specify the conditions that must be met for a row to be deleted in the WHERE clause.
- You can use a subquery within the WHERE clause.
- A foreign-key constraint may prevent you from deleting a row. In that case, you can only delete the row if you delete all child rows for that row first.



# SQL - DELETE (existing rows)

## The syntax of the DELETE statement

```
DELETE FROM table_name
[WHERE search_condition]
```

### Delete one row

```
DELETE FROM general_ledger_accounts
WHERE account_number = 306

(1 row affected)
```

### Delete one row using a compound condition

```
DELETE FROM invoice_line_items
WHERE invoice_id = 78 AND invoice_sequence = 2

(1 row affected)
```

### Delete multiple rows

```
DELETE FROM invoice_line_items
WHERE invoice_id = 12

(4 rows affected)
```

### Use a subquery in a DELETE statement

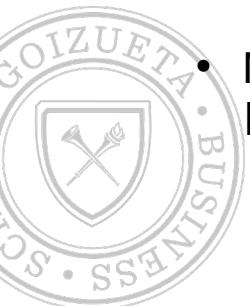
```
DELETE FROM invoice_line_items
WHERE invoice_id IN
(SELECT invoice_id
FROM invoices
WHERE vendor_id = 115)

(4 rows affected)
```



# SQL - DELETE (existing rows)

- `DELETE FROM customers WHERE fName = 'Barry'`
- Note: Error 1175 in MySQL implies that you can't delete without using primary key, or you need to turn off safe mode (`SET SQL_SAFE_UPDATES = 0;`).



# SQL - SELECT

Sakila

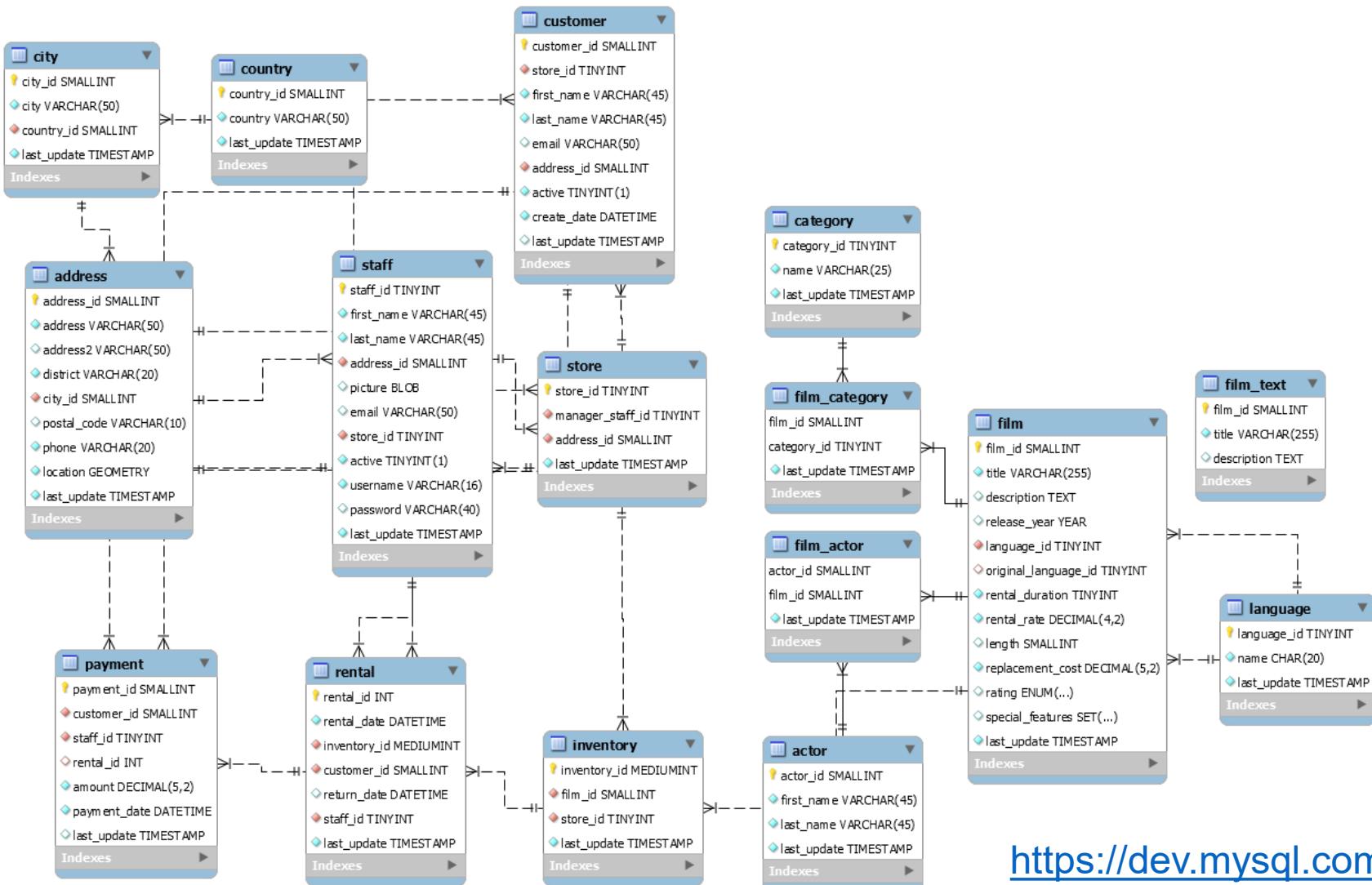


Rajiv Garg

September 7, 2023

44

# ER Model - sakila



<https://dev.mysql.com/doc/sakila/en/>



# Sakila - SQL Create Trigger Error

- Import movie rental (Sakila) database:
  - Source: <https://dev.mysql.com/doc/sakila/en/>
- Possible trigger creation error when loading on AWS RDS:
  - CREATE TRIGGER `ins\_film` AFTER INSERT ON `film` FOR EACH ROW BEGIN
  - INSERT INTO film\_text (film\_id, title, description)
  - VALUES (new.film\_id, new.title, new.description);
  - END;
- The CREATE TRIGGER statement allows you to create a trigger that is executed automatically whenever an event (INSERT, DELETE, or UPDATE) occurs against a table.
  - To create such automatically executing events, you need to enable the setting on the SQL server. By default, the setting is set to “0” on AWS RDS.
  - You can change it on AWS by following the steps described here:
    - <https://aws.amazon.com/premiumsupport/knowledge-center/rds-mysql-functions/>



# SQL - SELECT

- `SELECT [predicate] { * | table.* | [table.]field1 [AS alias1] [, [table.]field2 [AS alias2] [, ...]] } FROM tableexpression [, ...] [IN externaldatabase] [WHERE... ] [GROUP BY... ] [HAVING... ] [ORDER BY... ] [WITH OWNERACCESS OPTION]`
- SELECT Execution Order:
  - FROM - indicates the table from which the data is selected
  - WHERE - indicates conditions under which a row will be included in the result
  - GROUP BY - indicates categorization of results
  - HAVING - indicates the conditions under which a group/category will be included
  - SELECT - columns to be returned
  - ORDER BY - sort results
  - [output]



# SQL – SELECT

- Show movie title, description, release year and rating:

```
SELECT title, description, release_year, rating FROM sakila.film;
```

- List all unique customer locations (district) in address table

```
SELECT DISTINCT district FROM sakila.address;
```

- List all movie names and rental prices in ascending order:

```
SELECT title, release_year, rental_rate FROM sakila.film ORDER BY rental_rate ASC;
```



# SQL – SELECT

- Show release years (in ascending order) of movies in the data:

```
SELECT release_year FROM sakila.film ORDER BY release_year;
```

- Show unique release years (in ascending order) of movies in the data:

```
SELECT DISTINCT release_year FROM sakila.film ORDER BY release_year;
```

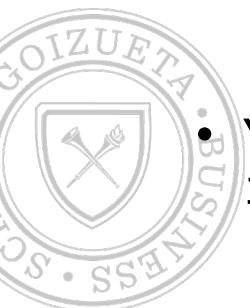
- Show unique rental rate and release year combinations (in ascending order of rental rate) of movies in the data:

```
SELECT DISTINCT release_year, rental_rate FROM sakila.film ORDER BY rental_rate;
```



# ORDER BY clause

- The ORDER BY clause specifies how you want the rows in the result set sorted. You can sort by one or more columns, and you can sort each column in either ascending (ASC) or descending (DESC) sequence.
  - ASC is the default.
- In an ascending sort, special characters appear first in the sort sequence, followed by numbers, then letters.
  - This sort order is determined by the character set used by the server.
- Null values appear first in the sort sequence, even if you're using DESC.
- You can sort by any column in the base table regardless of whether it's included in the SELECT clause.



# SQL – SELECT (example)

A SELECT statement that retrieves all the data from the Invoices table

```
SELECT * FROM invoices
```

|   | invoice_id | vendor_id | invoice_number | invoice_date | invoice_total | payment_total | credit_total | terms_id |
|---|------------|-----------|----------------|--------------|---------------|---------------|--------------|----------|
| ▶ | 1          | 122       | 989319-457     | 2011-04-08   | 3813.33       | 3813.33       | 0.00         | 3        |
| ▶ | 2          | 123       | 263253241      | 2011-04-10   | 40.20         | 40.20         | 0.00         | 3        |
| ◀ | 3          | 123       | 963253234      | 2011-04-13   | 138.75        | 138.75        | 0.00         | 3        |

(114 rows)

A SELECT statement that retrieves three columns from each row,  
sorted in descending sequence by invoice total

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
ORDER BY invoice_total DESC
```

|   | invoice_number | invoice_date | invoice_total |
|---|----------------|--------------|---------------|
| ▶ | 0-2058         | 2011-05-28   | 37966.19      |
| ▶ | P-0259         | 2011-07-19   | 26881.40      |
| ◀ | 0-2060         | 2011-07-24   | 23517.58      |

(114 rows)



# ORDER BY clause

The expanded syntax of the ORDER BY clause

```
ORDER BY expression [ASC|DESC] [, expression [ASC|DESC]] ...
```

An ORDER BY clause that sorts by one column in ascending sequence

```
SELECT vendor_name,
 CONCAT(vendor_city, ' ', vendor_state, ' ', vendor_zip_code) AS address
FROM vendors
ORDER BY vendor_name
```

|   | vendor_name                | address               |
|---|----------------------------|-----------------------|
| ▶ | Abbey Office Furnishings   | Fresno, CA 93722      |
|   | American Booksellers Assoc | Tarrytown, NY 10591   |
|   | American Express           | Los Angeles, CA 90096 |
|   | ASC Signs                  | Fresno, CA 93703      |

An ORDER BY clause that sorts by one column in descending sequence

```
SELECT vendor_name,
 CONCAT(vendor_city, ' ', vendor_state, ' ', vendor_zip_code) AS address
FROM vendors
ORDER BY vendor_name DESC
```

|   | vendor_name             | address              |
|---|-------------------------|----------------------|
| ▶ | Zylka Design            | Fresno, CA 93711     |
|   | Zip Print & Copy Center | Fresno, CA 93777     |
|   | Zee Medical Service Co  | Washington, IA 52353 |
|   | Yesmed, Inc             | Fresno, CA 93718     |



# SQL – SELECT (example)

A SELECT statement that returns all rows

```
SELECT vendor_city, vendor_state
FROM vendors
ORDER BY vendor_city
```

| vendor_city | vendor_state |
|-------------|--------------|
| Anaheim     | CA           |
| Anaheim     | CA           |
| Ann Arbor   | MI           |
| Aubum Hills | MI           |
| Boston      | MA           |
| Boston      | MA           |
| Boston      | MA           |

(122 rows)

A SELECT statement that eliminates duplicate rows

```
SELECT DISTINCT vendor_city, vendor_state
FROM vendors
ORDER BY vendor_city
```

| vendor_city  | vendor_state |
|--------------|--------------|
| Anaheim      | CA           |
| Ann Arbor    | MI           |
| Aubum Hills  | MI           |
| Boston       | MA           |
| Brea         | CA           |
| Carol Stream | IL           |
| Charlotte    | NC           |

(53 rows)



# LIMIT clause

- You can use the LIMIT clause to limit the number of rows returned by the SELECT statement.
  - This clause takes one or two integer arguments.
- If you code a single argument, it specifies the maximum row count, beginning with the first row.
- If you code both arguments, the offset specifies the first row to return, where the offset of the first row is 0.
  - If you want to retrieve all of the rows from a certain offset to the end of the result set, code -1 for the row count.



# LIMIT clause

The expanded syntax of the LIMIT clause

```
LIMIT [offset,] row_count
```

A SELECT statement with a LIMIT clause that starts with the first row

```
SELECT vendor_id, invoice_total
FROM invoices
ORDER BY invoice_total DESC
LIMIT 5
```

|   | vendor_id | invoice_total |
|---|-----------|---------------|
| ▶ | 110       | 37966.19      |
|   | 110       | 26881.40      |
|   | 110       | 23517.58      |
|   | 72        | 21842.00      |
|   | 110       | 20551.18      |

A SELECT statement with a LIMIT clause that starts with the third row

```
SELECT invoice_id, vendor_id, invoice_total
FROM invoices
ORDER BY invoice_id
LIMIT 2, 3
```

|   | invoice_id | vendor_id | invoice_total |
|---|------------|-----------|---------------|
| ▶ | 3          | 123       | 138.75        |
|   | 4          | 123       | 144.70        |
|   | 5          | 123       | 15.50         |



# SQL – WHERE

- You can use a comparison operator to compare any two expressions.
  - If the result of a comparison is a true, the row being tested is included in the result set. If it's a false or null value, the row isn't included.
- MySQL automatically converts the data for comparison.
  - Character comparisons performed on MySQL databases are not case-sensitive.
    - E.g. ‘CA’ and ‘ca’ are considered equivalent.
- If you compare a null value using one of the comparison operators, the result is always a null value. To test for null values, use the IS NULL clause.



# SQL – WHERE

- Find movies with rental rate of over \$4

```
SELECT title, release_year, rental_rate FROM sakila.film WHERE rental_rate>4;
```

- Find customers that live in “Georgia”

```
SELECT * FROM sakila.address WHERE district = "Georgia";
```

- Find customers with first names starting with alphabet between A-D

```
SELECT customer_id, first_name, last_name FROM customer WHERE first_name<'E';
```



# SQL – WHERE (IS NULL)

- A null value represents a value that's unknown, unavailable, or not applicable.
  - It isn't the same as a zero or an empty string ("").

## The syntax of the WHERE clause with the IS NULL clause

```
WHERE expression IS [NOT] NULL
```

## A SELECT statement that retrieves rows with null values

```
SELECT * FROM null_sample
WHERE invoice_total IS NULL
```

|   | invoice_id | invoice_total |
|---|------------|---------------|
| ▶ | 3          | NULL          |

## A SELECT statement that retrieves rows without null values

```
SELECT *
FROM null_sample
WHERE invoice_total IS NOT NULL
```

|   | invoice_id | invoice_total |
|---|------------|---------------|
| ▶ | 1          | 125.00        |
|   | 2          | 0.00          |
|   | 4          | 2199.99       |
|   | 5          | 0.00          |



# SQL – WHERE (NULL)

- Get movie rentals that are not returned:

```
SELECT rental_id, customer_id, return_date
FROM rental
WHERE return_date IS NULL;
```



# SQL – WHERE (example)

A SELECT statement that retrieves two columns and a calculated value for a specific invoice

```
SELECT invoice_id, invoice_total,
 credit_total + payment_total AS total_credits
FROM invoices
WHERE invoice_id = 17
```

|   | invoice_id | invoice_total | total_credits |
|---|------------|---------------|---------------|
| ▶ | 17         | 10.00         | 10.00         |

A SELECT statement that retrieves all invoices between given dates

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_date BETWEEN '2011-06-01' AND '2011-06-30'
ORDER BY invoice_date
```

|  | invoice_number | invoice_date | invoice_total |
|--|----------------|--------------|---------------|
|  | 111-92R-10094  | 2011-06-01   | 19.67         |
|  | 989319-437     | 2011-06-01   | 2765.36       |
|  | 1-202-2978     | 2011-06-03   | 33.00         |

(37 rows)

A SELECT statement that returns an empty result set

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_total > 50000
```

|  | invoice_number | invoice_date | invoice_total |
|--|----------------|--------------|---------------|
|  |                |              |               |



# SQL – WHERE (example)

Examples of WHERE clauses that retrieve...

**Vendors located in Iowa**

```
WHERE vendor_state = 'IA'
```

**Invoices with a balance due (two variations)**

```
WHERE invoice_total - payment_total - credit_total > 0
WHERE invoice_total > payment_total + credit_total
```

**Vendors with names from A to L**

```
WHERE vendor_name < 'M'
```

**Invoices on or before a specified date**

```
WHERE invoice_date <= '2011-07-31'
```

**Invoices on or after a specified date**

```
WHERE invoice_date >= '2011-07-01'
```

**Invoices with credits that don't equal zero (two variations)**

```
WHERE credit_total <> 0
WHERE credit_total != 0
```



# SQL - LIKE and REGEXP

- You use the LIKE and REGEXP operators to retrieve rows that match a string pattern, called a mask. The mask determines which values in the column satisfy the condition.
- The mask for a LIKE phrase can contain special symbols, called wildcards. The mask for a REGEXP phrase can contain special characters and constructs. Masks aren't case-sensitive.
- Most LIKE and REGEXP phrases significantly degrade performance compared to other types of searches, so use them only when necessary.



# SQL - LIKE and REGEXP

- LIKE wildcards

| Symbol | Description                                   |
|--------|-----------------------------------------------|
| %      | Matches any string of zero or more characters |
| -      | Matches any single character                  |

- REGEXP special characters and constructs

| Construct     | Description                                                    |
|---------------|----------------------------------------------------------------|
| ^             | Matches the pattern to the beginning of the value being tested |
| \$            | Matches the pattern to the end of the value being tested       |
| .             | Matches any single character                                   |
| [TEXTlist]    | Matches any single character listed within the brackets        |
| [TEXT1-TEXT2] | Matches any single character within the given range            |
|               | Separates two string patterns and matches either one           |



# SQL - LIKE and REGEXP (Examples)

## The syntax of the WHERE clause with a LIKE phrase

```
WHERE match_expression [NOT] LIKE pattern
```

## The syntax of the WHERE clause with a REGEXP phrase

```
WHERE match_expression [NOT] REGEXP pattern
```

## WHERE clauses that use the LIKE and REGEXP operators

| Example                                           | Results that match the mask        |
|---------------------------------------------------|------------------------------------|
| WHERE vendor_city LIKE 'SAN%'                     | "San Diego", "Santa Ana"           |
| WHERE vendor_name LIKE 'COMPU_ER%'                | "Compuserve", "Computerworld"      |
| WHERE vendor_city REGEXP 'SA'                     | "Pasadena", "Santa Ana"            |
| WHERE vendor_city REGEXP '^SA'                    | "Santa Ana", "Sacramento"          |
| WHERE vendor_city REGEXP 'NA\$'                   | "Gardena", "Pasadena", "Santa Ana" |
| WHERE vendor_city REGEXP 'RS SN'                  | "Traverse City", "Fresno"          |
| WHERE vendor_state REGEXP 'N[CV]'                 | "NC" and "NV" but not "NJ" or "NY" |
| WHERE vendor_state REGEXP 'N[A-J]'                | "NC" and "NJ" but not "NV" or "NY" |
| WHERE vendor_contact_last_name REGEXP 'DAMI[EO]N' | "Damien" and "Damion"              |
| WHERE vendor_city REGEXP '[A-Z][AEIOU]N\$'        | "Boston", "Mclean", "Oberlin"      |



# Logical operators: AND, OR, NOT

- You can use the AND and OR logical operators to create compound conditions that consist of two or more conditions.
  - You use the AND operator to specify that the search must satisfy both of the conditions, and you use the OR operator to specify that the search must satisfy at least one of the conditions.
- You can use the NOT operator to negate a condition.
- When MySQL evaluates a compound condition, it evaluates the operators in this sequence: (1) NOT, (2) AND, and (3) OR.
  - You can use parentheses to override this order of precedence or to clarify the sequence in which the operations are evaluated.



# Logical operators: examples

## The syntax of the WHERE clause with logical operators

```
WHERE [NOT] search_condition_1 {AND|OR} [NOT] search_condition_2 ...
```

## Examples of WHERE clauses that use logical operators

### The AND operator

```
WHERE vendor_state = 'NJ' AND vendor_city = 'Springfield'
```

### The OR operator

```
WHERE vendor_state = 'NJ' OR vendor_city = 'Pittsburg'
```

### The NOT operator

```
WHERE NOT vendor_state = 'CA'
```

### The NOT operator in a complex search condition

```
WHERE NOT (invoice_total >= 5000 OR NOT invoice_date <= '2011-08-01')
```

### The same condition rephrased to eliminate the NOT operator

```
WHERE invoice_total < 5000 AND invoice_date <= '2011-08-01'
```



# IN operator

- You can use the IN phrase to test whether an expression is equal to a value in a list of expressions.
  - Each of the expressions in the list is automatically converted to the same type of data as the test expression.
  - The list of expressions can be coded in any order without affecting the order of the rows in the result set.
- You can use the NOT operator to test for an expression that's not in the list of expressions.
- You can also compare the test expression to the items in a list returned by a subquery.



# IN operator: examples

## The syntax of the WHERE clause with an IN phrase

```
WHERE test_expression [NOT] IN
 ({subquery|expression_1 [, expression_2]...})
```

### Examples of the IN phrase

#### An IN phrase with a list of numeric literals

```
WHERE terms_id IN (1, 3, 4)
```

#### An IN phrase preceded by NOT

```
WHERE vendor_state NOT IN ('CA', 'NV', 'OR')
```

#### An IN phrase with a subquery

```
WHERE vendor_id IN
 (SELECT vendor_id
 FROM invoices
 WHERE invoice_date = '2011-07-18')
```

- Sakila:
- `SELECT * FROM film WHERE length IN (120, 121);`
- `SELECT * FROM film WHERE rental_rate NOT IN (2.99, 0.99);`



# BETWEEN operator

- You can use the BETWEEN phrase to test whether an expression falls within a range of values.
  - The lower limit must be coded as the first expression, and the upper limit must be coded as the second expression.
  - Otherwise, MySQL returns an empty result set.
- The two expressions used in the BETWEEN phrase for the range of values are inclusive.
  - That is, the result set includes values that are equal to the upper or lower limit.
- You can use the NOT operator to test for an expression that's not within the given range.



# BETWEEN operator: examples

The syntax of the WHERE clause with a BETWEEN phrase

```
WHERE test_expression [NOT] BETWEEN begin_expression AND end_expression
```

Examples of the BETWEEN phrase

A BETWEEN phrase with literal values

```
WHERE invoice_date BETWEEN '2011-06-01' AND '2011-06-30'
```

A BETWEEN phrase preceded by NOT

```
WHERE vendor_zip_code NOT BETWEEN 93600 AND 93799
```

A BETWEEN phrase with a test expression coded as a calculated value

```
WHERE invoice_total - payment_total - credit_total BETWEEN 200 AND 500
```

A BETWEEN phrase with the upper and lower limits

coded as calculated values

```
WHERE payment_total BETWEEN credit_total AND credit_total + 500
```

- Sakila:
- `SELECT * FROM film WHERE length BETWEEN 120 AND 125;`
- `SELECT * FROM film WHERE title BETWEEN "AA" AND "AE";`



# SQL – CAST

- **CAST(value AS *datatype*)**

- **Example:**

```
SELECT
 CAST('2021-03-24' AS DATE) today_date,
 CAST('10:00:00' AS TIME) today_time
```

```
SELECT rental_date, cast(rental_date AS
DATE) AS TD FROM rental;
SELECT rental_date, cast(rental_date AS
YEAR) AS TD FROM rental;
SELECT rental_date, cast(rental_date AS
CHAR(7)) AS TD FROM rental;
```

| <b><i>datatype</i></b> | Description                                                                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DATE                   | Converts value to DATE. Format: "YYYY-MM-DD"                                                                                                                         |
| DATETIME               | Converts value to DATETIME. Format: "YYYY-MM-DD HH:MM:SS"                                                                                                            |
| DECIMAL                | Converts value to DECIMAL. Use the optional M and D parameters to specify the maximum number of digits (M) and the number of digits following the decimal point (D). |
| TIME                   | Converts value to TIME. Format: "HH:MM:SS"                                                                                                                           |
| CHAR                   | Converts value to CHAR (a fixed length string)                                                                                                                       |
| NCHAR                  | Converts value to NCHAR (like CHAR, but produces a string with the national character set)                                                                           |
| SIGNED                 | Converts value to SIGNED (a signed 64-bit integer)                                                                                                                   |
| UNSIGNED               | Converts value to UNSIGNED (an unsigned 64-bit integer)                                                                                                              |
| BINARY                 | Converts value to BINARY (a binary string)                                                                                                                           |



# SQL – CASE (logical expression)

- Get customers with their status (active/inactive):

```
SELECT c.first_name, c.last_name,
CASE
 WHEN active = 1
 THEN "ACTIVE"
 ELSE "INACTIVE"
END active_type
FROM customer AS c;
```



# SQL – CASE (logical expression)

- Get number of rentals for the months of May, June, July:

```
SELECT monthname(rental_date) rental_month, count(*) num_rentals
FROM rental
WHERE rental_date BETWEEN "2005-05-01" AND "2005-08-01"
GROUP BY monthname(rental_date);
```

```
SELECT
 SUM(CASE WHEN monthname(rental_date) = "May" THEN 1 ELSE 0 END) May_rentals,
 SUM(CASE WHEN monthname(rental_date) = "June" THEN 1 ELSE 0 END) June_rentals,
 SUM(CASE WHEN monthname(rental_date) = "July" THEN 1 ELSE 0 END) July_rentals
FROM rental
WHERE rental_date BETWEEN "2005-05-01" AND "2005-08-01";
```



# SELECT: built-in functions

- An expression can include any of the functions that are supported by MySQL. A function performs an operation and returns a value.
- To use a function, code the function name followed by a set of parentheses. Within the parentheses, code any parameters (or arguments) required by the function.
  - If a function requires two or more arguments, separate them with commas.
- To code a literal value for a string, enclose one or more characters within single quotes ( ' ) or double quotes ( " ).
- Complete list of functions:  
<https://dev.mysql.com/doc/refman/8.0/en/built-in-function-reference.html>



# SELECT: built-in functions

- Select full name of actors in one column
  - `select CONCAT(first_name, " ", last_name) AS full_name`
  - `from actor`
  - `ORDER BY last_name DESC, first_name DESC`
  - `LIMIT 10;`



# SELECT: built-in functions

## A SELECT statement that uses the LEFT function

```
SELECT vendor_contact_first_name, vendor_contact_last_name,
 CONCAT(LEFT(vendor_contact_first_name, 1),
 LEFT(vendor_contact_last_name, 1)) AS initials
FROM vendors
```

|   | vendor_contact_first_name | vendor_contact_last_name | initials |
|---|---------------------------|--------------------------|----------|
| ▶ | Francesco                 | Alberto                  | FA       |
|   | Ania                      | Irvin                    | AI       |
|   | Lukas                     | Liana                    | LL       |

(122 rows)

## A SELECT statement that uses the DATE\_FORMAT function

```
SELECT invoice_date,
 DATE_FORMAT(invoice_date, '%m/%d/%Y') AS 'MM/DD/YY',
 DATE_FORMAT(invoice_date, '%e-%b-%Y') AS 'DD-Mon-YYYY'
FROM invoices
```

|   | invoice_date | MM/DD/YY | DD-Mon-YYYY |
|---|--------------|----------|-------------|
| ▶ | 2011-04-08   | 04/08/11 | 8-Apr-2011  |
|   | 2011-04-10   | 04/10/11 | 10-Apr-2011 |
|   | 2011-04-13   | 04/13/11 | 13-Apr-2011 |

(114 rows)

## A SELECT statement that uses the ROUND function

```
SELECT invoice_date, invoice_total,
 ROUND(invoice_total) AS nearest_dollar,
 ROUND(invoice_total, 1) AS nearest_dime
FROM invoices
```

|   | invoice_date | invoice_total | nearest_dollar | nearest_dime |
|---|--------------|---------------|----------------|--------------|
| ▶ | 2011-04-08   | 3813.33       | 3813           | 3813.3       |
|   | 2011-04-10   | 40.20         | 40             | 40.2         |
|   | 2011-04-13   | 138.75        | 139            | 138.8        |

(114 rows)





# SQL I

Catch up



# SQL II

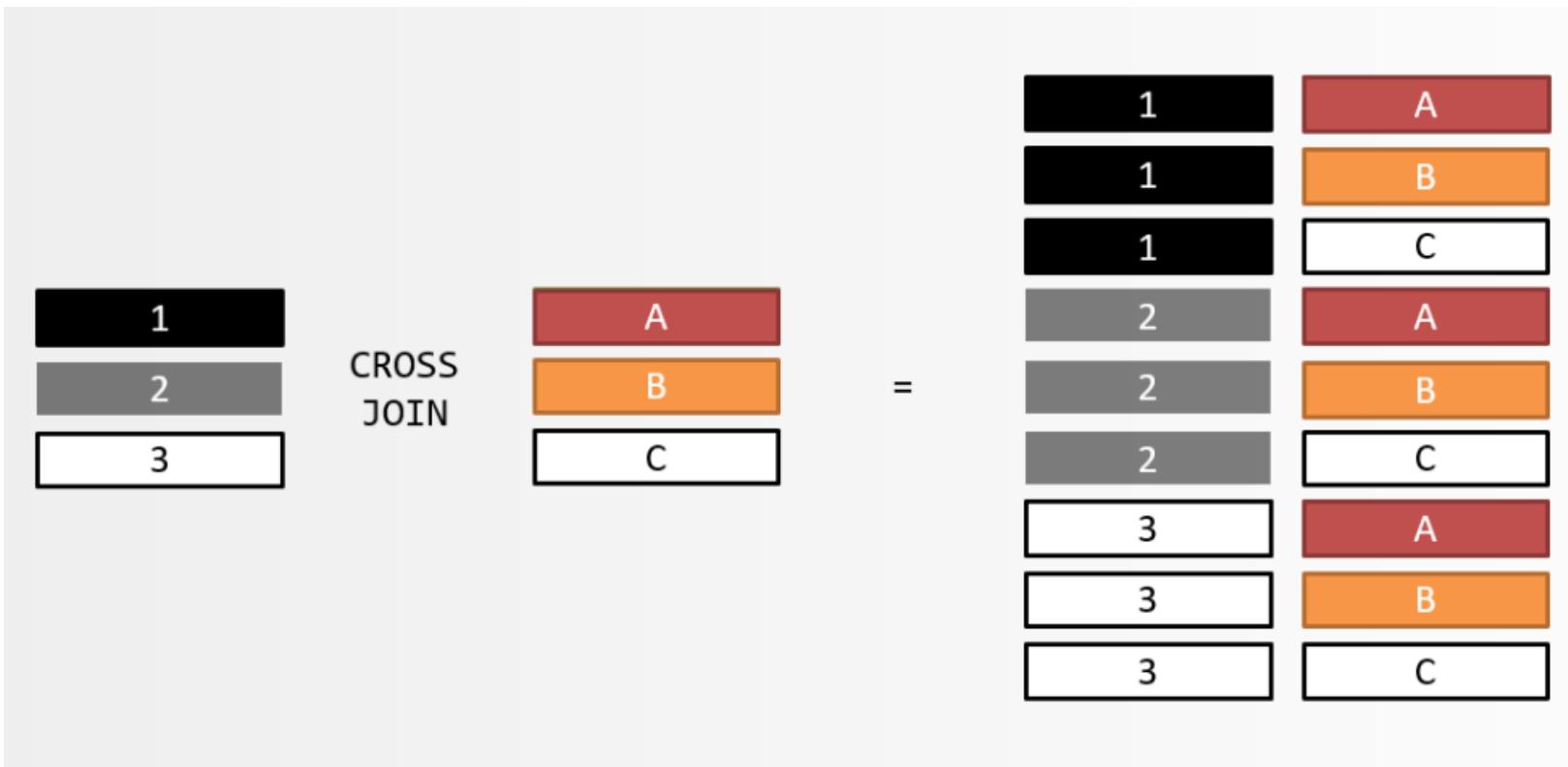
Data Querying

# JOIN data from multiple tables

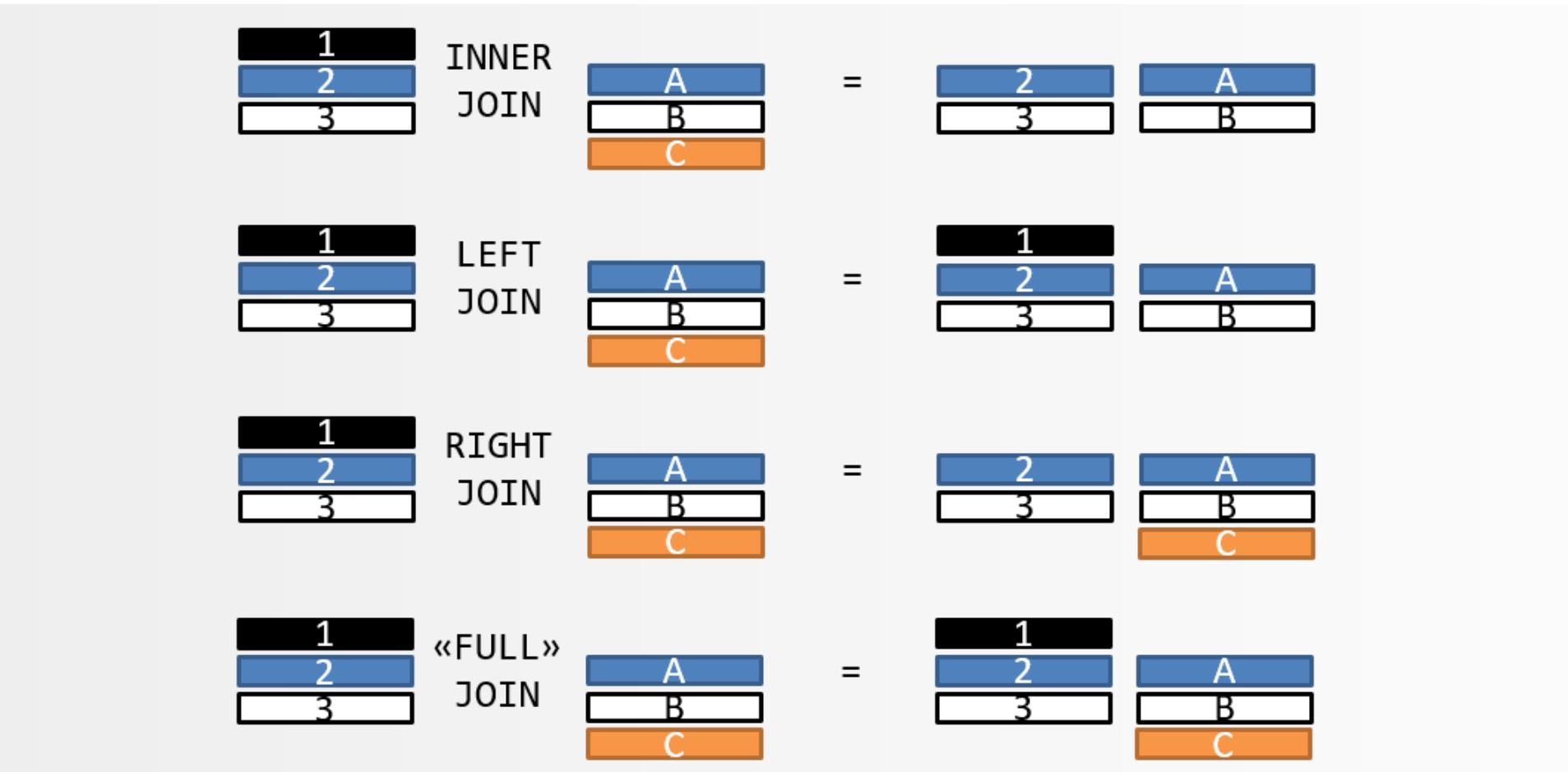
- A **join** lets you combine columns from two or more tables into a single result set.
  - A join between tables returns all unique combinations of their tuples which meet some specified join condition.
- For an **inner join**, only those rows that satisfy the join condition(s) are included in the result set.
- An **outer join** retrieves all rows that satisfy the join condition(s), plus unmatched rows in the left or right table.
- A **cross join** joins each row from the first table with each row from the second table. The result set returned by a cross join is known as a Cartesian product.



# JOIN: Cross



# JOIN: Inner and Outer



# JOIN: Query

- Syntax:
  - ```
SELECT select_list
      FROM [table_1_database_name.] table_1 [table_1_alias]
            [INNER] JOIN [table_2_database_name.]table_2 [table_2_alias]
                      ON join_condition_1 [join_condition_2 ...]
      [[INNER]] JOIN [table_3_database_name.]table_3 [table_3_alias]
                      ON join_condition_3 [join_condition_4 ...] ]..
```
- A join condition names a column in each of the two tables involved in the particular join and indicates how the two columns should be compared.
 - In most cases, you use the equal operator to retrieve rows with matching columns. However, you can also use any of the other comparison operators in a join condition.
- Tables are typically joined on the relationship between the primary key in one table and a foreign key in the other table.



JOIN (INNER)

- **JOIN:**

```
SELECT c.first_name, c.last_name, a.address  
FROM customer AS c  
JOIN address AS a;
```

Cartesian product:
every customer with
every address

- **INNER JOIN:**

```
SELECT c.first_name, c.last_name, a.address  
FROM customer AS c  
INNER JOIN address AS a  
ON c.address_id = a.address_id;
```

Limits join to
customers with
address_id

```
SELECT c.first_name, c.last_name, a.address, ct.city  
FROM customer AS c  
INNER JOIN address AS a  
ON c.address_id = a.address_id  
INNER JOIN city AS ct  
ON a.city_id = ct.city_id;
```

Joining across three
tables



JOIN (OUTER)

- Inner join gets rows that are matched on a criteria in both tables
 - Thus, only rows where (c.address_id = a.address_id) are shown.
- What if you want to see the rows from customer (c) table even if that customer has address_id as NULL?

```
SELECT c.first_name, c.last_name, a.address  
FROM customer AS c  
LEFT OUTER JOIN address AS a  
ON c.address_id = a.address_id;
```

Left outer join: get rows from customer table even if they don't have matching address_id

```
SELECT c.first_name, c.last_name, a.address  
FROM customer AS c  
RIGHT OUTER JOIN address AS a  
ON c.address_id = a.address_id;
```

Right outer join: get rows from address table even if they don't have matching address_id



INNER JOIN: creating query

- A join condition can include two or more conditions connected by AND or OR.
- When you run a SELECT statement against one database, you can join to a table in another database.
 - To do that, you must prefix the table name in the other database with the name of that database. You also need to have appropriate permissions.
- You can think of a multi-table join as a series of two-table joins proceeding from left to right.
- To code a self-join, you must use aliases for the tables, and you must qualify each column name with the alias.
- Instead of coding a join condition in the FROM clause, you can code it in the WHERE clause along with any search conditions as an implicit join.



INNER JOIN: Additional Examples

```
SELECT invoice_number, vendor_name  
FROM vendors v, invoices i  
WHERE v.vendor_id = i.vendor_id  
ORDER BY invoice_number
```

Identify vendor and invoice from two tables

```
SELECT invoice_number, vendor_name  
FROM vendors INNER JOIN invoices  
    ON vendors.vendor_id = invoices.vendor_id  
ORDER BY invoice_number
```

```
SELECT customer_first_name, customer_last_name  
FROM customers c JOIN employees e  
    ON c.customer_first_name = e.first_name  
    AND c.customer_last_name = e.last_name
```

Identify customers that are also employees

```
SELECT vendor_name, customer_last_name, customer_first_name,  
        vendor_state AS state, vendor_city AS city  
FROM vendors v  
    JOIN _____customers c  
        ON v.vendor_zip_code = c.customer_zip  
ORDER BY state, city
```

Identify vendors and customers in the same zip code

```
SELECT DISTINCT v1.vendor_name, v1.vendor_city,  
        v1.vendor_state  
FROM vendors v1 JOIN vendors v2  
    ON v1.vendor_city = v2.vendor_city AND  
        v1.vendor_state = v2.vendor_state AND  
        v1.vendor_name <> v2.vendor_name  
ORDER BY v1.vendor_state, v1.vendor_city
```

Identify distinct vendors in the same city, state



OUTER JOIN: creating query

- When you use a left outer join, the result set includes all the rows from the first (or left) table.
 - Similarly, when you use a right outer join, the result set includes all the rows from the second (or right) table.
 - Syntax:
 - ```
SELECT select_list
 FROM table_1
 {LEFT|RIGHT} [OUTER] JOIN table_2
 ON join_condition_1
 [{LEFT|RIGHT} [OUTER] JOIN table_3
 ON join_condition_2]...
```
- The OUTER keyword is optional.



# OUTER JOIN: Additional Examples

```
SELECT department_name, e.department_number, last_name
FROM departments d
 RIGHT JOIN employees e
 ON d.department_number = e.department_number
ORDER BY department_name
```

Get all employee names and department name (if available)

```
SELECT department_name, last_name, project_number
FROM departments d
 LEFT JOIN employees e
 ON d.department_number = e.department_number
 LEFT JOIN projects p
 ON e.employee_id = p.employee_id
ORDER BY department_name, last_name
 ORDER BY department_name
```

Get all employee names and department name (if available) and employee project number (if available)

```
SELECT department_name, last_name, project_number
FROM departments d
 JOIN employees e
 ON d.department_number = e.department_number
 LEFT JOIN projects p
 ON e.employee_id = p.employee_id
ORDER BY department_name, last_name
```

Get all employee names WITH department name and employee project number (if available)



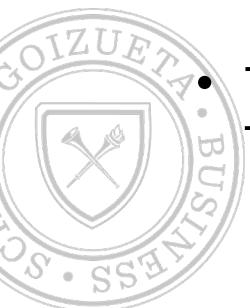
# CROSS JOIN: creating query

- A cross join joins each row from the first table with each row from the second table.
- Syntax:
  - `SELECT select_list  
FROM table_1 CROSS JOIN table_2`
  - or
  - `SELECT select_list  
FROM table_1, table_2`



# UNION

- A union combines the result sets of two or more SELECT statements into one result set.
  - The column names in the final result set are taken from the first SELECT clause.
- Each result set must return the same number of columns, and the corresponding columns in each result set must have compatible data types.
- By default, a union eliminates duplicate rows. If you want to include duplicate rows, code the ALL keyword.
- To sort the rows in the final result set, code an ORDER BY clause after the last SELECT statement.
  - This clause must refer to the column names assigned in the first SELECT clause.



# UNION: creating query

- Syntax of the union operation:

- SELECT\_statement\_1
- UNION [ALL]
- SELECT\_statement\_2
- [UNION [ALL]]
- SELECT\_statement\_3] . . .
- [ORDER BY order\_by\_list]



# UNION

- What if you want to combine data from multiple tables into one table?
    - JOIN allows you to combine two datasets side-by-side, but UNION allows you to stack one dataset on top of the other.
  - Create a union of two different staff tables:
    - `SELECT * FROM staff`
    - **UNION**
    - `SELECT * FROM staff;`
  - Place the union into a new table:
    - `CREATE TABLE staff_new AS`
    - `SELECT * FROM staff`
    - **UNION ALL**
    - `SELECT * FROM staff;`
- UNION combines distinct rows in sorted order
- Output of a UNION can also be inserted into a new table
- UNION ALL combines ALL rows



# UNION

- List movies that have duration 120, 130, or 140 minutes (*you can use OR in SELECT*)

```
SELECT * FROM film WHERE length = 120
UNION
SELECT * FROM film WHERE length = 130
UNION
SELECT * FROM film WHERE length = 140;
```

UNION can be compounded  
and processed in order  
Think: (UNION, UNION ALL)  
vs (UNION ALL, UNION)

- Combine the dataset and order by columns (show all actor and customer names where first name starts with J and last name starts with D).

```
SELECT a.first_name fname, a.last_name lname FROM actor a
WHERE a.first_name LIKE 'J%' AND a.last_name LIKE 'D%'
UNION
SELECT c.first_name, c.last_name FROM customer c
WHERE c.first_name LIKE 'J%' AND c.last_name LIKE 'D%'
ORDER BY fname, lname;
```



# UNION: Additional Examples

```
SELECT 'Active' AS source, invoice_number, invoice_date, invoice_total
FROM active_invoices
WHERE invoice_date >= '2011-06-01'
UNION
SELECT 'Paid' AS source, invoice_number, invoice_date, invoice_total
FROM paid_invoices
WHERE invoice_date >= '2011-06-01'
ORDER BY invoice_total DESC
```

Combine invoices from  
two different tables  
(active and paid)

```
SELECT department_name AS dept_name, d.department_number AS d_dept_no,
 e.department_number AS e_dept_no, last_name
 FROM departments d
 LEFT JOIN employees e
 ON d.department_number = e.department_number
UNION
 SELECT department_name AS dept_name, d.department_number AS d_dept_no,
 e.department_number AS e_dept_no, last_name
 FROM departments d
 RIGHT JOIN employees e
 ON d.department_number = e.department_number
 ORDER BY dept_name
```

Combine the results of  
left join and right join of  
employee and  
department tables



# Summary Query (Aggregate functions)

- A summary query is a SELECT statement that includes one or more aggregate functions.
  - Summary queries provide important analytical functions to DBMSs.
- The expression you specify for the AVG and SUM functions must result in a numeric value. The expression for the MIN, MAX, and COUNT functions can result in a numeric, date, or string value.
- By default, all values are included in the calculation regardless of whether they're duplicated. If you want to omit duplicate values, code the DISTINCT keyword. All of the aggregate functions except for COUNT(\*) ignore null values.



# Aggregate Functions

| Function syntax                    | Result                                               |
|------------------------------------|------------------------------------------------------|
| AVG([ALL   DISTINCT] expression)   | The average of the non-null values in the expression |
| SUM([ALL   DISTINCT] expression)   | The total of the non-null values in the expression   |
| MIN([ALL   DISTINCT] expression)   | The lowest non-null value in the expression          |
| MAX([ALL   DISTINCT] expression)   | The highest non-null value in the expression         |
| COUNT([ALL   DISTINCT] expression) | The number of non-null values in the expression      |
| COUNT(*)                           | The number of rows selected by the query             |

- Example:

```
SELECT COUNT(DISTINCT vendor_id) AS number_of_vendors,
 COUNT(vendor_id) AS number_of_invoices,
 ROUND(AVG(invoice_total), 2) AS avg_invoice_amt,
 SUM(invoice_total) AS total_invoice_amt
 FROM invoices
 WHERE invoice_date > '2011-01-01'
```



# GROUP BY

- The GROUP BY clause groups the rows of a result set based on one or more columns or expressions.
- If you include aggregate functions in the SELECT clause, the aggregate is calculated for each group specified by the GROUP BY clause.
- The HAVING clause specifies a search condition for a group or an aggregate.
  - MySQL applies this condition after it groups the rows that satisfy the search condition in the WHERE clause.
- When a GROUP BY clause is used, the SELECT clause can include the columns used for grouping, aggregate functions, and expressions that result in a constant value.



# GROUP BY and HAVING

- The syntax of the GROUP BY and HAVING clauses:
  - ```
SELECT select_list
      FROM table_source
      [WHERE search_condition]
      [GROUP BY group_by_list]
      [HAVING search_condition]
      [ORDER BY order_by_list]
```
- Example:

```
SELECT vendor_id, ROUND(AVG(invoice_total), 2) AS average_invoice_amount
FROM invoices
GROUP BY vendor_id
HAVING AVG(invoice_total) > 2000
ORDER BY average_invoice_amount DESC
```



GROUP BY: HAVING

- Groups similar data together and enables aggregation functions (max, min, avg, sum):
- Query the number of rentals by each customer_id:

```
SELECT customer_id, count(*)  
FROM rental  
GROUP BY customer_id;
```

Counts in each group

- Get high frequency renters:

```
SELECT customer_id, count(*)  
FROM rental  
GROUP BY customer_id  
HAVING count(*)>40;
```

Note: WHERE will not work
here because it is
executed before GROUP BY



WHERE VS HAVING

- When you include a WHERE clause in a SELECT statement that uses grouping and aggregates, the search condition is applied before grouping the rows and calculating the aggregates.
- When you include a HAVING clause in a SELECT statement that uses grouping and aggregates, the search condition is applied after grouping the rows and calculating the aggregates.
- A WHERE clause can refer to any column in the base tables.
- A HAVING clause can only refer to a column included in the SELECT clause.
- A WHERE clause can't contain aggregate functions.
- A HAVING clause can contain aggregate functions.
- You can use the AND and OR operators to code compound search conditions in both HAVING and WHERE clauses.



GROUP BY: aggregation functions

- Using aggregation functions (max, min, avg, sum):

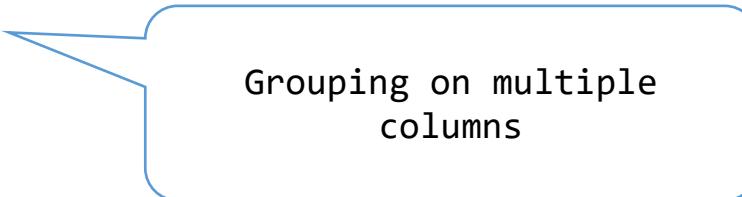
```
SELECT customer_id,  
MAX(amount) max_amt,  
MIN(amount) min_amt,  
AVG(amount) avg_amt,  
SUM(amount) tot_amt,  
count(*) num_payments  
FROM payment  
GROUP BY customer_id;
```



GROUP BY: multiple columns

- Grouping on multiple columns – actor and film rating (number of movies an actor has acted in a specific rated film):

```
SELECT fa.actor_id, f.rating, count(*) num_films  
FROM film_actor AS fa  
INNER JOIN film AS f  
ON fa.film_id = f.film_id  
GROUP BY fa.actor_id, f.rating  
ORDER BY 1, 2;
```



Grouping on multiple columns



5:00

Speed Test! (ungraded)

- Write queries for the following (accuracy + speed matters):
 1. Display the most frequently rented movies in descending order.
 2. For a marketing campaign display all customer names and emails in Canada
 3. Display the total payment processed by each staff member during each month in 2005.
 4. List the top five genres in gross revenue in descending order.



Speed Test! (solutions)

```
#Display the most frequently rented movies in descending order.  
SELECT title, COUNT(title) as 'Rentals'  
FROM film  
JOIN inventory  
ON (film.film_id = inventory.film_id)  
JOIN rental  
ON (inventory.inventory_id = rental.inventory_id)  
GROUP by title  
ORDER BY rentals desc;
```

```
#Display the total payment processed by each staff member during each  
month in 2005.  
SELECT s.first_name, s.last_name, MONTH(p.payment_date),  
SUM(p.amount)  
FROM staff s  
INNER JOIN payment p  
ON (s.staff_id = p.staff_id)  
WHERE YEAR(p.payment_date) = 2005  
GROUP BY s.staff_id, MONTH(p.payment_date)  
ORDER BY MONTH(p.payment_date), s.staff_id;
```

```
#For a marketing campaign display all customer names and emails in  
Canada  
SELECT c.first_name, c.last_name, c.email  
FROM customer c  
JOIN address a ON (c.address_id = a.address_id)  
JOIN city ci ON (a.city_id = ci.city_id)  
JOIN country ctr ON (ci.country_id = ctr.country_id)  
WHERE ctr.country = 'canada';
```

```
#List the top five genres in gross revenue in descending order.  
SELECT c.name AS 'Genre', SUM(amount) AS 'Total Sales'  
FROM payment p  
JOIN rental r  
ON (p.rental_id = r.rental_id)  
JOIN inventory i  
ON (r.inventory_id = i.inventory_id)  
JOIN film_category fc  
ON (i.film_id = fc.film_id)  
JOIN category c  
ON (fc.category_id = c.category_id)  
GROUP BY c.name  
ORDER BY SUM(amount) DESC;
```



SQL & Text



Text Analytics

- While SQL is not the optimal choice for NLP and sophisticated text analytics, it does offer multiple benefits:
 - Data in RDBMS (structured table form)
 - SQL is simple for quantification of insights in text data
 - E.g. number of movies with description containing “scientist”
 - String functions can be nested for deeper analysis
 - Offers a rule-based analysis that is easy to understand
 - Fast when you know what you are looking for



Text Length()

- select title, description, length(description)
- from film
- where description regexp "scientist";

- select length(description), count(*)
- from film
- group by 1
- order by 1 desc;



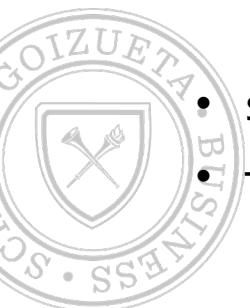
Text Parsing (left, right)

- Left(): returns characters from the left (or beginning) of the string
- Right() : returns characters from the right (or end) of the string
- select description, length(description), left(description, 10), right(description, 10)
- from film
- where description regexp "scientist";
- select left(description, 5), count(*)
- from film
- group by 1
- order by 1 desc;



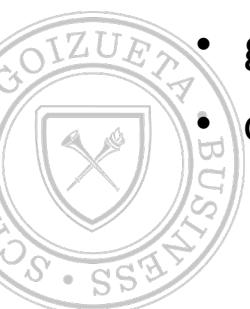
Text Parsing (substring)

- SUBSTRING(string, position, length): returns the “length” of characters from “position” in the “string”
- SUBSTRING_INDEX(string, delimiter, index): returns the entire text to the left of the “delimiter” at “index”
- select description, SUBSTRING(description, 5, 5)
 - from film;
- select description, SUBSTRING_INDEX(description, " ", 2)
 - from film;
- select description, SUBSTRING_INDEX(description, "And", 1)
 - from film;



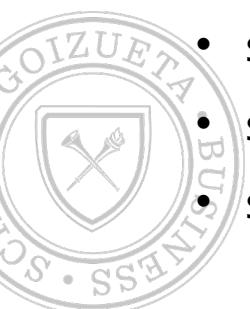
Text Parsing (substring nesting)

- Find text between two words in the description data
- select description,
- **SUBSTRING_INDEX(SUBSTRING_INDEX(description, "And", 1), " a ", -1)**
- from film;
- Select **SUBSTRING_INDEX(SUBSTRING_INDEX(description, "And", 1), " a ", -1)**, count(*)
- from film
- group by 1
- order by 1 desc;



Text Transformation

- `Lower(string)`: converts all characters in a string to lower case
- `Upper(string)`: converts all characters in a string to upper case
- `Trim(string)`: removes blank space from beginning and ending of string
- `Trim(leading/trailing CHAR from string)`: removes CHAR from beginning (leading) or ending (trailing) of string
- `select title, lower(title), upper(title)`
- `from film;`
- `select trim(" I am a short sentence ");`
- `select trim(leading "*" from "*I am a short sentence*");`
- `select trim(trailing "*" from "*I am a short sentence*");`



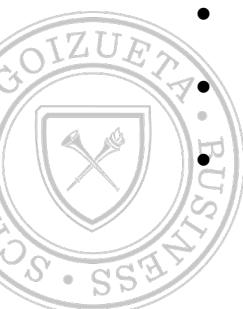
Text Transformation

- Replace(string, substring, replacement): Replace substring with replacement
- select description, replace(description, "And", "OR")
- from film;



Text Labels (case)

- Assign labels to text and count text of those label types
- select
- case
 - when lower(description) regexp "tale" then "tale"
 - when lower(description) regexp "saga" then "saga"
 - when lower(description) regexp "documentary" then "documentary"
 - when lower(description) regexp "drama" then "drama"
 - else "other"
- end as film_type, count(*)
from film
group by 1;



Text Labels (case)

- Assign labels to text and count text of those label types and organize in columns
- select
- count(case when lower(description) regexp "tale" then 1 end) as tale,
- count(case when lower(description) regexp "saga" then 1 end) as tale,
- count(case when lower(description) regexp "documentary" then 1 end) as tale,
- count(case when lower(description) regexp "drama" then 1 end) as tale
- from film;



Text Search - LIKE and REGEXP

- LIKE wildcards

Symbol	Description
%	Matches any string of zero or more characters
-	Matches any single character

- REGEXP special characters and constructs

Construct	Description
^	Matches the pattern to the beginning of the value being tested
\$	Matches the pattern to the end of the value being tested
.	Matches any single character
[TEXTlist]	Matches any single character listed within the brackets
[TEXT1-TEXT2]	Matches any single character within the given range
	Separates two string patterns and matches either one



Text Search - REGEXP

- [:class:]
 - Searches for a class ([:digit:], [:alpha:]) of characters in string
 - SELECT title,
 - title REGEXP '[:alpha:]', title REGEXP '[:digit:]'
 - FROM film;
- {count}
 - Finds the strings with number of instances of characters preceding the count
 - Finds the strings with {min,max} number of instances of characters preceding the count
 - select description, length(description), description regexp '^.{95}\$' from film;
 - select description, length(description), description regexp '^.{95,98}\$' from film;



Text Search - REGEXP

- REGEXP_LIKE(): returns 1 if it finds pattern in a string
- REGEXP_INSTR(): returns index of string where pattern is found
- select description,
- description regexp 'scientist',
- REGEXP_LIKE(description, 'scientist'),
- REGEXP_INSTR(description, 'scientist')
- from film;
- select description,
- REGEXP_LIKE(description, 'a '),
- REGEXP_INSTR(description, 'a '),
- REGEXP_INSTR(description, 'a ', 2)
- from film limit 20;



Text Search - REGEXP

- REGEXP_REPLACE(): replaces the found pattern with another string
- select description,
- REGEXP_LIKE(description, 'a '),
- REGEXP_INSTR(description, 'a '),
- REGEXP_INSTR(description, 'a ', 2),
- REGEXP_REPLACE(description, 'a ', "NEW"),
- REGEXP_REPLACE(description, 'a ', "NEW", 2)
- from film limit 20;



Text Search - REGEXP

- REGEXP_SUBSTR(): return the pattern found in the string
- select description,
- REGEXP_LIKE(description, '[a-z]+'),
- REGEXP_SUBSTR(description, '[a-z]+'),
- REGEXP_SUBSTR(description, '[a-z]+', 3)
- from film limit 20;

Position to start looking for
the pattern in the string.
Here, starting from 3rd character



Text Search - REGEXP

- GROUP_CONCAT(): combines all values in rows into a list in one row
- SELECT GROUP_CONCAT(distinct rating) AS ratings
- FROM film;
- SELECT rating, GROUP_CONCAT(distinct rental_rate ORDER BY rental_rate) AS ratings
- FROM film
- GROUP BY rating;



Subqueries



SQL subqueries

- A subquery is a SELECT statement that's coded within another SQL statement.
- A subquery can return a single value, a list of values (a result set that has a single column), or a table of values (a result set that has multiple columns).
- A subquery can be coded, or introduced, anywhere a single value, a list of values, or a table is allowed.
- The syntax for a subquery is the same as for a standard SELECT statement.
 - However, a subquery can't include an ORDER BY clause.

Subqueries can be nested within other subqueries.



SQL subqueries

- QUERY data from a QUERIED DATA
- Find the name of the customer that was last added into the system (largest customer id)

```
SELECT MAX(customer_id) FROM customer;
```

```
> 599
```

```
SELECT customer_id, first_name, last_name FROM customer WHERE customer_id = 599;
```

- Alternatively:

```
SELECT customer_id, first_name, last_name FROM customer  
WHERE customer_id = (SELECT MAX(customer_id) FROM customer);
```



SQL subqueries: correlated

- A correlated subquery is a subquery that is executed once for each row in the main query. In contrast, an uncorrelated subquery is executed only once.
- A correlated subquery refers to a value that's provided by a column in the main query. For each different value that's returned by the main query for that column, the subquery returns a different result.
- Example:

```
SELECT vendor_id, invoice_number, invoice_total
FROM invoices i
WHERE invoice_total >
    (SELECT AVG(invoice_total)
     FROM invoices
     WHERE vendor_id = i.vendor_id)
ORDER BY vendor_id, invoice_total
```



SQL subqueries: IN operator

- You can introduce a subquery with the IN operator to provide the list of values that are tested against the test expression.
- When you use the IN operator, the subquery must return a single column of values.
- The syntax of a WHERE clause that uses an IN phrase:
 - `WHERE test_expression [NOT] IN (subquery)`



SQL subqueries: IN operator

- Subqueries that don't have any correlation with the main query are called Noncorrelated (scalar) queries:

```
SELECT city_id, city  
FROM city  
WHERE country_id <> (SELECT country_id FROM country WHERE country = "India");
```

```
SELECT city_id, city  
FROM city  
WHERE country_id IN (SELECT country_id FROM country WHERE country IN ("India", "Canada",  
"Mexico"));
```



SQL subqueries: BETWEEN

- Subqueries that have correlation with the main query are called Correlated (dependent) queries:

```
SELECT c.first_name, c.last_name  
FROM customer AS c  
WHERE (SELECT sum(p.amount) FROM payment AS p WHERE p.customer_id = c.customer_id)  
BETWEEN 180 AND 240;
```

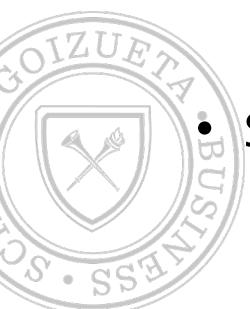


SQL subqueries: comparison

- You can use a comparison operator in a WHERE clause to compare an expression with the results of a subquery.
- If you code a search condition without the ANY, SOME, and ALL keywords, the subquery must return a single value.
- If you include the ANY, SOME, or ALL keyword, the subquery can return a list of values.
 - You can use the ALL keyword to test that a comparison condition is true for all of the values returned by a subquery.
 - You can use the ANY keyword to test that a condition is true for one or more of the values returned by a subquery.
 - The SOME keyword works the same as the ANY keyword.

Syntax:

- WHERE expression comparison_operator [SOME|ANY|ALL] (subquery)



SQL subqueries: EXISTS

- You can use the EXISTS operator to test that one or more rows are returned by the subquery.
 - You can use the NOT EXISTS operator to test that no rows are returned by the subquery.
- When you use these operators with a subquery, it doesn't matter what columns you specify in the SELECT clause. As a result, you typically just code an asterisk (*).
- Syntax: WHERE [NOT] EXISTS (subquery)
- Example:

```
SELECT vendor_id, vendor_name, vendor_state  
FROM vendors  
WHERE NOT EXISTS  
(SELECT *  
FROM invoices  
WHERE vendor_id = vendors.vendor_id)
```



SQL subqueries: FROM

- A subquery that's coded in the FROM clause returns a result set that can be referred to as an inline view.
- When you code a subquery in the FROM clause, you must assign an alias to it.
 - You can use that alias just as you would any other table name or alias.
- Example:

```
SELECT vendor_state, MAX(sum_of_invoices) AS max_sum_of_invoices
FROM
(
    SELECT vendor_state, vendor_name,
           SUM(invoice_total) AS sum_of_invoices
      FROM vendors v JOIN invoices i
        ON v.vendor_id = i.vendor_id
     GROUP BY vendor_state, vendor_name
) t
GROUP BY vendor_state
```



SQL subqueries

- SQL subqueries allow you to create an extremely large SQL code for almost any complex data querying and processing task.
- To tackle any complex task, think of divide and conquer:
 - **Divide**: data processing into sequential steps
 - **Code**: query for each step
 - **Combine**: queries in a nested structure to combine the output



SQL WITH Clause

SQL Subqueries



WITH Subqueries

- Use a WITH clause that has one or more comma-separated subclauses. Each subclause provides a subquery that produces a result set, and associates a name with the subquery.
- WITH
- cte1 AS (SELECT a, b FROM table1),
- cte2 AS (SELECT c, d FROM table2)
- SELECT b, d FROM cte1 INNER JOIN cte2
- ON cte1.a = cte2.c;
- You can also do recursive coding with “WITH” in SQL. For more information, please see: <https://dev.mysql.com/doc/refman/8.0/en/with.html>



SQL Windows



SQL - Windows

- What is a moving average?
- Where is it used?
- Why is it useful?
- How would you estimate it programmatically?

	fiscal_year	sales_employee	sale	total_sales
▶	2016	Alice	150.00	450.00
	2016	Bob	100.00	450.00
	2016	John	200.00	450.00
▶	2017	Alice	100.00	400.00
	2017	Bob	150.00	400.00
	2017	John	150.00	400.00
▶	2018	Alice	200.00	650.00
	2018	Bob	200.00	650.00
	2018	John	250.00	650.00



Image source: <https://www.mysqltutorial.org/mysql-window-functions/>

SQL - Windows

- Moving window of data:

```
SELECT Ticker, SUM(DivAmt), AVG(DivAmt)
FROM dividends
GROUP BY Ticker
ORDER BY Ticker;
```

```
SELECT Ticker, DivAmt, DivDate, SUM(DivAmt) OVER (PARTITION BY Ticker) total_div
FROM dividends;
```

```
SELECT Ticker, DivAmt, DivDate, SUM(DivAmt) OVER (PARTITION BY Ticker) total_div,
row_number() OVER (PARTITION BY Ticker ORDER BY DivDate) as `row`
FROM dividends;
```



Source: <https://sqlite.org/windowfunctions.html>



SQL – Windows (BETWEEN)

- Define number of **following** rows to consider in moving window computation:

```
SELECT Ticker, DivAmt, DivDate, SUM(DivAmt) OVER  
(PARTITION BY Ticker ROWS BETWEEN CURRENT ROW AND  
UNBOUNDED FOLLOWING) total_div, row_number() OVER  
(PARTITION BY Ticker ORDER BY DivDate) as `row`  
FROM dividends;
```

```
SELECT Ticker, DivAmt, DivDate, SUM(DivAmt) OVER  
(PARTITION BY Ticker ROWS BETWEEN CURRENT ROW AND 2  
FOLLOWING) total_div, row_number() OVER (PARTITION BY  
Ticker ORDER BY DivDate) as `row`  
FROM dividends;
```

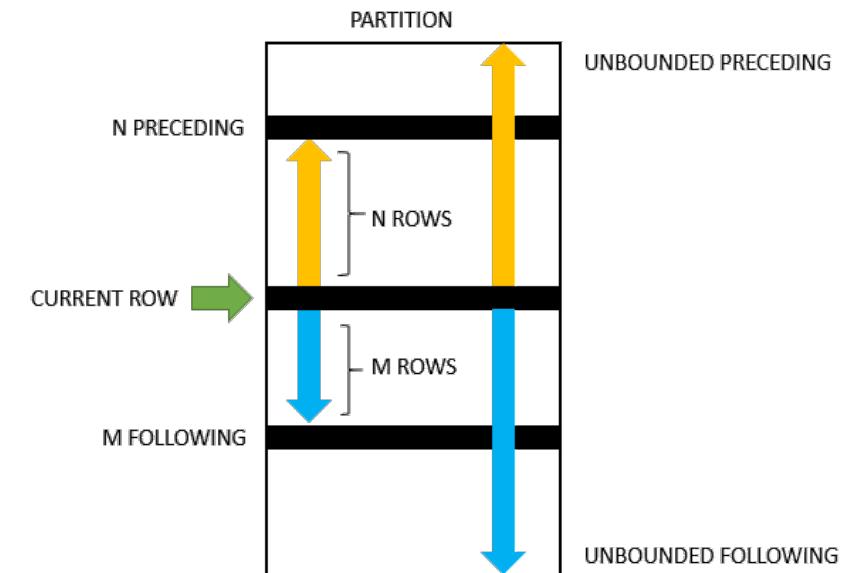


Image source: <https://www.mysqltutorial.org/mysql-window-functions/>



SQL – Windows (BETWEEN)

- Define number of preceding rows to consider in moving window computation:

```
SELECT Ticker, DivAmt, DivDate, SUM(DivAmt) OVER  
(PARTITION BY Ticker ROWS BETWEEN UNBOUNDED PRECEDING AND  
CURRENT ROW) total_div, row_number() OVER (PARTITION BY  
Ticker ORDER BY DivDate) as `row`  
FROM dividends;
```

```
SELECT Ticker, DivAmt, DivDate, SUM(DivAmt) OVER  
(PARTITION BY Ticker ROWS BETWEEN 2 PRECEDING AND CURRENT  
ROW) total_div, row_number() OVER (PARTITION BY Ticker  
ORDER BY DivDate) as `row`  
FROM dividends;
```

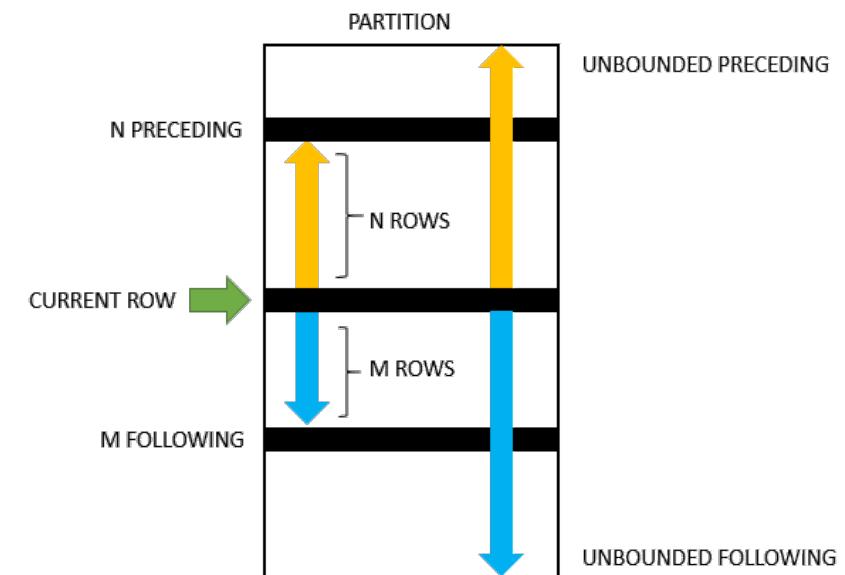


Image source: <https://www.mysqltutorial.org/mysql-window-functions/>

SQL – Windows Comparison (BETWEEN)

- Comparing results from unbounded following and unbounded preceding:

```
SELECT Ticker, DivAmt, DivDate,  
SUM(DivAmt) OVER (PARTITION BY Ticker ROWS BETWEEN  
CURRENT ROW AND UNBOUNDED FOLLOWING)  
total_div_fol,  
SUM(DivAmt) OVER (PARTITION BY Ticker ROWS BETWEEN  
UNBOUNDED PRECEDING AND CURRENT ROW)  
total_div_pre,  
row_number() OVER (PARTITION BY Ticker ORDER BY  
DivDate) as `row`  
FROM dividends;
```

Ticker	DivAmt	DivDate	DivFol	DivPre	r
AACC	2.45	2007-07-17	2.45	2.45	1
AAON	0.13333	2006-06-08	1.10666	0.26666	1
AAON	0.13333	2006-12-11	1.23999	0.13333	2
AAON	0.13333	2007-06-07	0.97333	0.39999	3
AAON	0.16	2007-12-11	0.83999	0.55999	4
AAON	0.16	2008-06-10	0.67999	0.71999	5
AAON	0.16	2008-12-10	0.52	0.87999	6
AAON	0.18	2009-06-09	0.36	1.05999	7
AAON	0.18	2009-12-10	0.18	1.23999	8



SQL – Windows (RANGE)

- ROWS BETWEEN provides window function for rows in data. What if data is missing dates?
 - How do we use windows over dates and not rows?

```
SELECT Ticker, DivAmt, DivDate, SUM(DivAmt) OVER (PARTITION BY Ticker ORDER BY  
DATE(DivDate) RANGE BETWEEN INTERVAL 1 DAY PRECEDING AND INTERVAL 1 DAY FOLLOWING)  
date_sum  
FROM dividends;
```



SQL – LAG/LEAD

- What if you want to compare the values that are a lagging or leading?

```
SELECT Ticker, DivAmt, DivDate,  
lag(DivAmt, 1) OVER (PARTITION BY Ticker) lag_div,  
lead(DivAmt, 1) OVER (PARTITION BY Ticker) lead_div  
FROM dividends;
```

Ticker	DivAmt	DivDate	lag_div	lead_div
AACC	2.45	2007-07-17	NULL	NULL
AAON	0.18	2009-12-10	NULL	0.18
AAON	0.18	2009-06-09	0.18	0.16
AAON	0.16	2008-12-10	0.18	0.16
AAON	0.16	2008-06-10	0.16	0.16
AAON	0.16	2007-12-11	0.16	0.13333
AAON	0.13333	2007-06-07	0.16	0.13333
AAON	0.13333	2006-12-11	0.13333	0.13333
AAON	0.13333	2006-06-08	0.13333	NULL
AAPL	0.03	1995-11-21	NULL	0.03
AAPL	0.03	1995-08-16	0.03	0.03





SQL II

catch up



SQL III

UDF, Python, and more

SQL UDF

User Defined Functions



UDF – Stored Functions

- User defined functions allow simplification of repeated tasks in queries.

```
CREATE FUNCTION function_name(  
    param1,  
    param2,...  
)  
RETURNS datatype  
[NOT] DETERMINISTIC  
BEGIN  
    -- statements  
END
```

Result is “deterministic” (some outcome count received) or “not deterministic” may imply that the function can return different values for same input parameters



For details, please refer to:

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

UDF - Hello World

- CREATE FUNCTION hello (s CHAR(20))
- RETURNS CHAR(50) DETERMINISTIC
- RETURN CONCAT('Hello ',s,'!');
- SELECT hello('world');
- SELECT hello(first_name)
- FROM customer;



For details, please refer to:

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

UDF – Rental Labels (\$, \$\$, \$\$\$)

- DELIMITER //
- CREATE FUNCTION RentalPriceLabel(rentalrate DECIMAL(10,2))
- RETURNS VARCHAR(20) DETERMINISTIC
- BEGIN
 - DECLARE rentallabel VARCHAR(20);
 - IF rentalrate < 1 THEN SET rentallabel = '\$';
 - ELSEIF (rentalrate >= 1 AND rentalrate < 3) THEN SET rentallabel = '\$\$';
 - ELSEIF rentalrate >= 3 THEN SET rentallabel = '\$\$\$';
 - END IF;
 - -- return the rental labels
 - RETURN (rentallabel);
- END//
- DELIMITER ;

- SELECT title, rental_rate, RentalPriceLabel(rental_rate)
FROM film
ORDER BY title;



UDF – Stored Procedures

- Function is invoked to perform an action and return a value to the user/query
- You create a **stored procedure** to call an operation that is not used directly providing a return value.
- Example:
 - Function: `SELECT hello('world');`
 - Function: `SELECT hello('world')`, “<- is function”;
 - Procedure: `CALL hello('world');`



UDF – Country Count

- Write a procedure that returns the number of cities in a country:

```
delimiter //  
CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)  
BEGIN  
    SELECT COUNT(*) INTO cities FROM city  
    WHERE country_id = country;  
END //  
delimiter ;
```

```
SET @count_id = (SELECT country_id from country where country="China");  
CALL citycount(@count_id, @cities);  
SELECT @count_id, @cities;
```

Input parameter

Output of procedure/function

Temporary variable

Note: `delimiter` command to change the statement delimiter from ; to // while the procedure is being defined. This enables the ; delimiter used in the procedure body to be passed through to the server rather than being interpreted by mysql itself.



SQL - Temporary Variables

- `SET @var1 = 5;`
- `SELECT customer_id, COUNT(*) FROM rental GROUP BY customer_id HAVING COUNT(*) > @var1;`
- `SET @var2 = "ANA%";`
- `SET @mycust = (SELECT first_name from customer WHERE first_name LIKE @var2);`
- `SELECT @var1, @var2, @mycust;`

Temporary variable can contain only one value



UDF - Examples

- How would you do word count on a column in a table?
- `SELECT wordcount("THIS IS A TEST");`
- `SELECT title, wordcount(title) as wc`
- `FROM film ORDER BY wc DESC LIMIT 10;`

```
DELIMITER $$  
• CREATE FUNCTION wordcount(str LONGTEXT)  
    RETURNS INT  
    DETERMINISTIC  
    SQL SECURITY INVOKER  
    NO SQL  
  
    BEGIN  
        DECLARE wordCnt, idx, maxIdx INT DEFAULT 0;  
        DECLARE currChar, prevChar BOOL DEFAULT 0;  
        SET maxIdx=char_length(str);  
        SET idx = 1;  
        WHILE idx <= maxIdx DO  
            SET currChar=SUBSTRING(str, idx, 1) RLIKE '[:alnum:]';  
            IF NOT prevChar AND currChar THEN  
                SET wordCnt=wordCnt+1;  
            END IF;  
            SET prevChar=currChar;  
            SET idx=idx+1;  
        END WHILE;  
        RETURN wordCnt;  
    END  
$$  
DELIMITER ;  
  
• SELECT wordcount("THIS IS A TEST");
```



SQL Triggers



SQL - Trigger

- A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.
 - A trigger is defined to activate when a statement **inserts**, **updates**, or **deletes** rows in the associated table.
- Referencing rows in triggers:
 - In an INSERT trigger, only NEW.col_name can be used; there is no old row.
 - In a DELETE trigger, only OLD.col_name can be used; there is no new row.
 - In an UPDATE trigger, you can use OLD.col_name to refer to the columns of a row before it is updated and NEW.col_name to refer to the columns of the row after it is updated.



For details, please refer to:

<https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html>

SQL – Trigger (insert)

- use testDB;
- CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
- CREATE TRIGGER ins_sum BEFORE INSERT ON account
 - FOR EACH ROW SET @sum = @sum + NEW.amount;
- SET @sum = 0;
- INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
- select * from account;
- SELECT @sum AS 'Total amount inserted';
- DROP TRIGGER testDB.ins_sum;

BEFORE inserting NEW rows, add the value from NEW rows into a variable (can be a column in another table)

This will contain sum of NEW amounts added to the table.



SQL – Trigger (update)

- delimiter //
- CREATE TRIGGER upd_check BEFORE UPDATE ON account
- FOR EACH ROW
- BEGIN
- IF NEW.amount < 0 THEN
- SET NEW.amount = 0;
- ELSEIF NEW.amount > 100 THEN
- SET NEW.amount = 100;
- END IF;
- END//
- delimiter ;

- SET SQL_SAFE_UPDATES = 0;
- UPDATE account SET amount = -200 WHERE account.acct_num =97;
- UPDATE account SET amount = 125 WHERE account.acct_num =137;
- select * from account;
- SET SQL_SAFE_UPDATES = 1;

BEFORE updating with NEW data, assign a limit to top and bottom values. Could be used to avoid erroneous updates.

Update value smaller than 0 will be changed to 0 and values greater than 100 will be changed to 100



SQL Execution Plan



SQL – EXPLAIN (index)

- EXPLAIN shows the optimal SQL execution plan
 - *find rentals that happened on May 24, 2005*

```
EXPLAIN SELECT inventory_id, customer_id  
FROM rental WHERE rental_date='2005-05-24';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	rental	NULL	ref	rental_date	rental_date	5	const	1	100.00	Using index

Using index



For details, please refer to:

<https://dev.mysql.com/doc/refman/8.0/en/explain-output.html>



Indexing

- Easily lookup by customer name or order location
 - How many orders from Chicago?
 - How many orders by Jones?

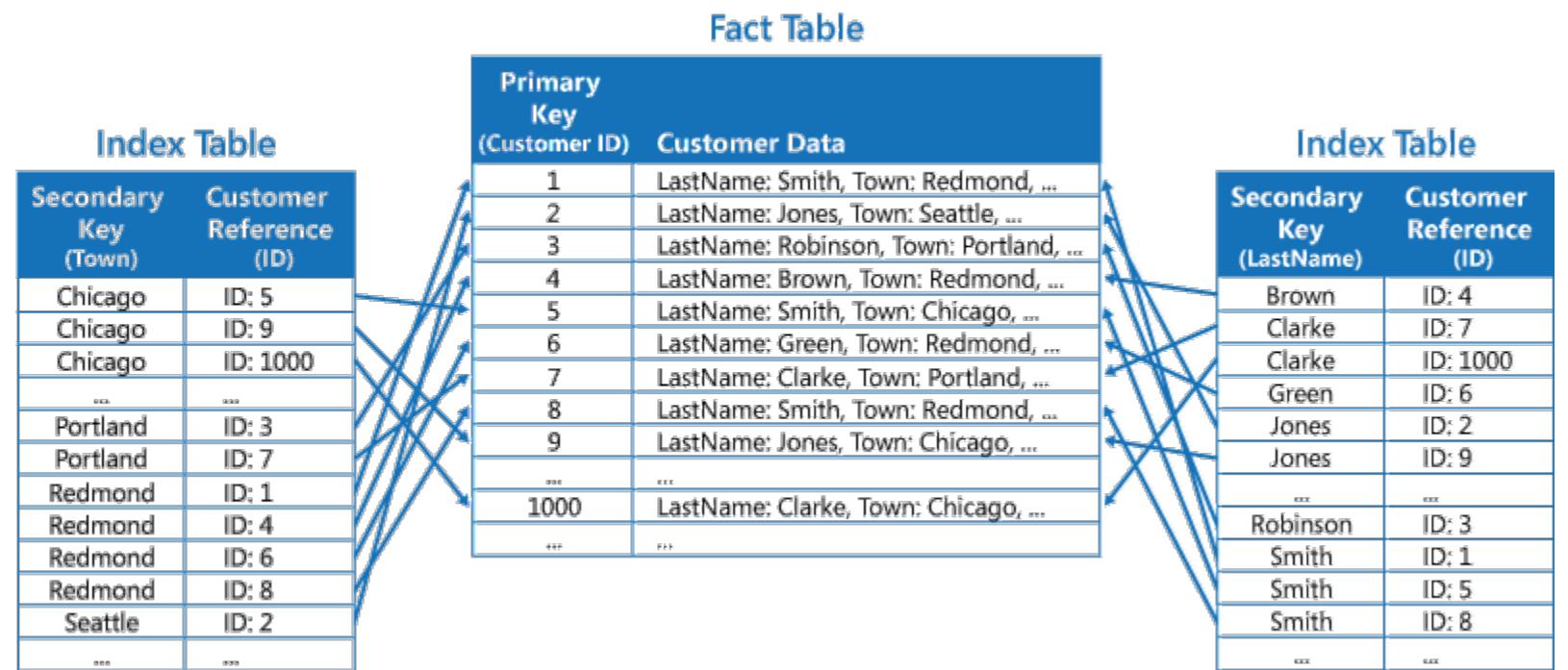


Image source: <https://docs.microsoft.com/en-us/azure/architecture/patterns/index-table>

SQL – EXPLAIN (type and rows)

- A complex query is optimized by the SQL engine
 - *find all films released in the year of 2006 where the actor with a last name of 'MIRINDA' starred*

```
EXPLAIN SELECT title, description FROM film AS f
JOIN film_actor AS fa ON f.film_id = fa.film_id
JOIN actor AS a ON fa.actor_id = a.actor_id
WHERE a.last_name = 'MIRANDA' AND f.release_year = 2006;
```

Execution order is independent of the order in query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a	NULL	ref	PRIMARY, idx_actor_last_name	idx_actor_last_name	137	const	1	100.00	Using index
1	SIMPLE	fa	NULL	ref	PRIMARY, idx_fk_film_id	PRIMARY	2	sakila.a.actor_id	27	100.00	Using index
1	SIMPLE	f	NULL	eq_ref	PRIMARY	PRIMARY	2	sakila.fa.film_id	1	10.00	Using where

rows will be accessed using an index, one row is read from the table for each combination of rows from the previous tables

Number of rows processed

Using 10% rows

SQL – EXPLAIN (select type)

select_type Value	JSON Name	Meaning
SIMPLE	None	Simple <code>SELECT</code> (not using <code>UNION</code> or subqueries)
PRIMARY	None	Outermost <code>SELECT</code>
UNION	None	Second or later <code>SELECT</code> statement in a <code>UNION</code>
DEPENDENT UNION	dependent (true)	Second or later <code>SELECT</code> statement in a <code>UNION</code> , dependent on outer query
UNION RESULT	union_result	Result of a <code>UNION</code> .
SUBQUERY	None	First <code>SELECT</code> in subquery



For details, please refer to:

<https://dev.mysql.com/doc/refman/8.0/en/explain-output.html>

SQL - EXPLAIN

- To optimize queries:
 - **Index check:** Make sure number of rows processed are few and tables are indexed when a column in those tables is likely to be used frequently
 - **Lean data:** When possible, avoid redundant data selection in query compounding (JOIN, WHERE, SUBQUERIES, etc.)
 - EXPLAIN SELECT min(rental_rate) from film;
 - vs
 - EXPLAIN SELECT min(rental_rate) from film WHERE rental_rate>0;

33% of data is filtered



SQL & Python



Rajiv Garg



SQL & Python

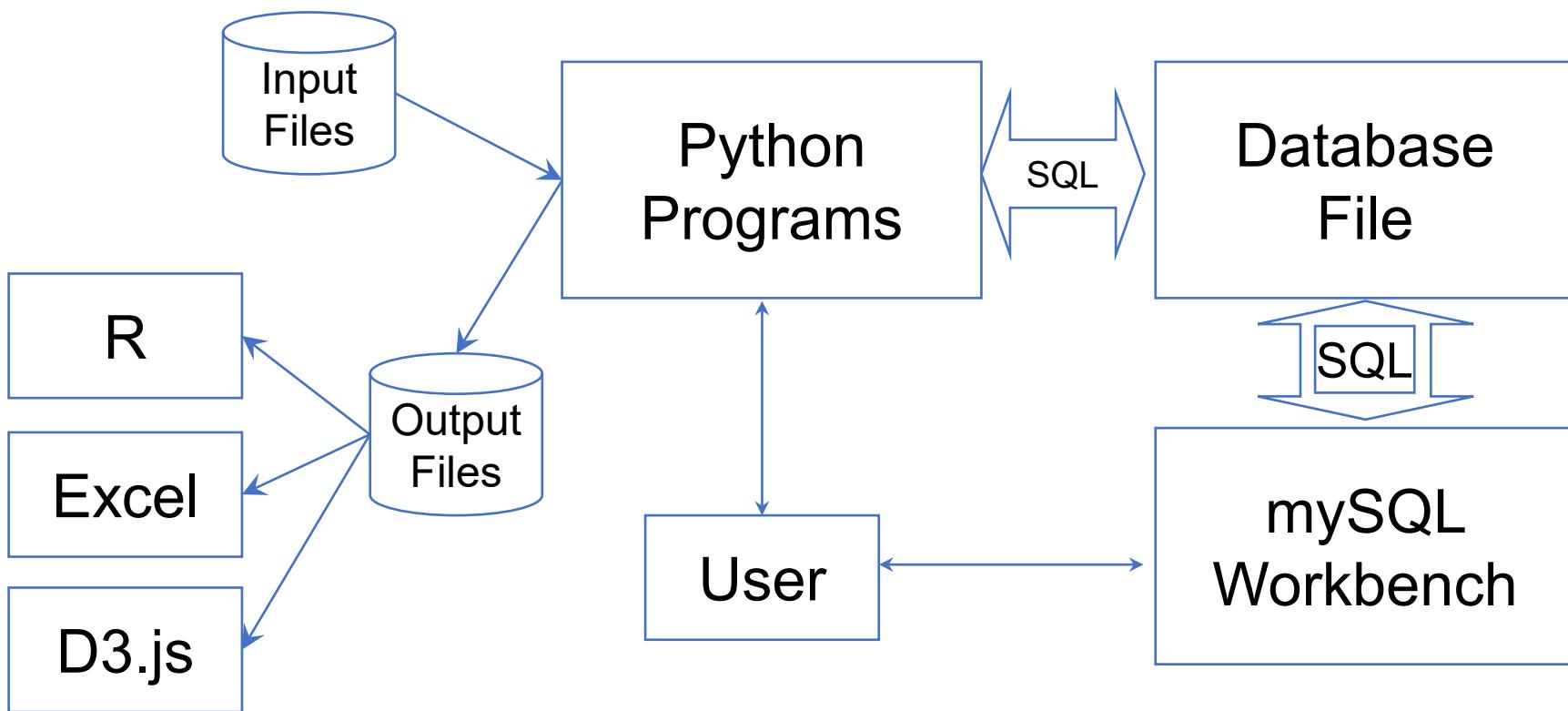


Image source: Charles Severance (py4e.com)



Accessing Databases

- Install: <https://pypi.org/project/pyodbc/>
 - pip install pyodbc
- Reads and writes MS Access database, mySQL, and many more
- ODBC: Open Database Connectivity
 - Microsoft introduced the ODBC standard in 1992.
 - ODBC is a specification for a database API that is independent of any one DBMS or operating system.
 - “The functions in the ODBC API are implemented by developers of DBMS-specific drivers. Applications call the functions in these drivers to access data in a DBMS-independent manner.”
 - <https://docs.microsoft.com/en-us/sql/odbc/reference/what-is-odbc>

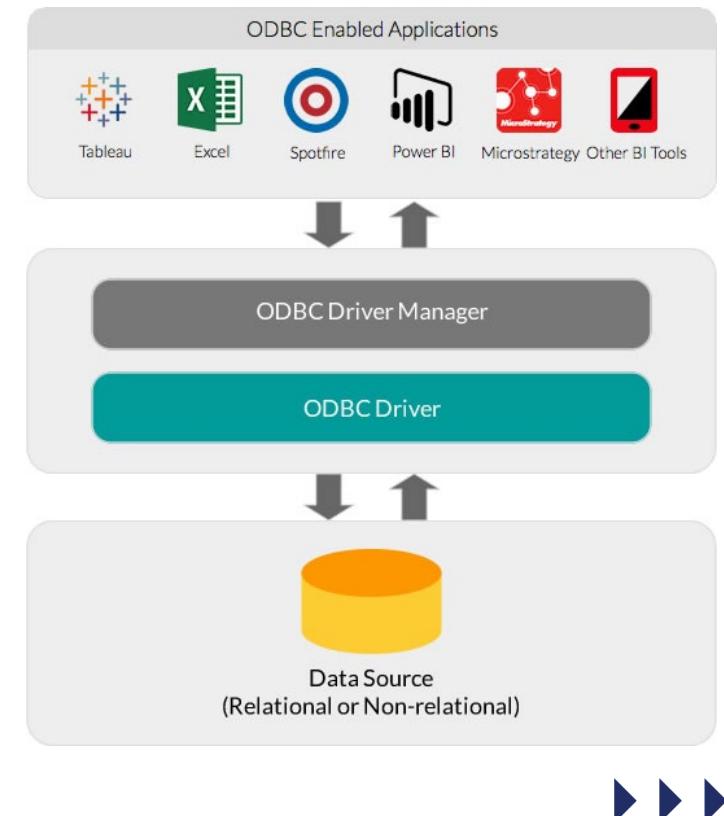


Image source: <https://www.simba.com/resources/odbc/>



ODBC vs mySQL Connector

- `pip install pyodbc`
 - Works for all database software installed on your computer (including MS Access)
 - Works for same bit version of Python (x86) and RDBMS (x86)
 - Or you need the database specific connector
- `pip install mysql-connector-python`
 - Specific to mySQL
 - Works when you have different version of Python (x86) and mySQL (x64)



Reading mySQL Data

- Using ODBC connection to read databases

```
import mysql.connector  
  
mydb = mysql.connector.connect(  
    host="localhost",  
    user="root",  
    password=open("mysql_pass.txt", "r").readline() #SAVE PASSWORD IN A FILE  
)  
  
cursor = mydb.cursor()  
cursor.execute('select * from products')  
  
for row in cursor.fetchall():  
    print (row)  
  
(1, 'apple', 3.5, 4.99, 100.0)  
(2, 'orange', 0.87, 1.49, 200.0)  
(3, 'milk', 2.68, 3.99, 300.0)  
(4, 'bread', 1.41, 1.99, 400.0)  
(5, 'beer', 9.9, 19.99, 100.0)  
(6, 'diaper', 18.21, 29.99, 50.0)
```

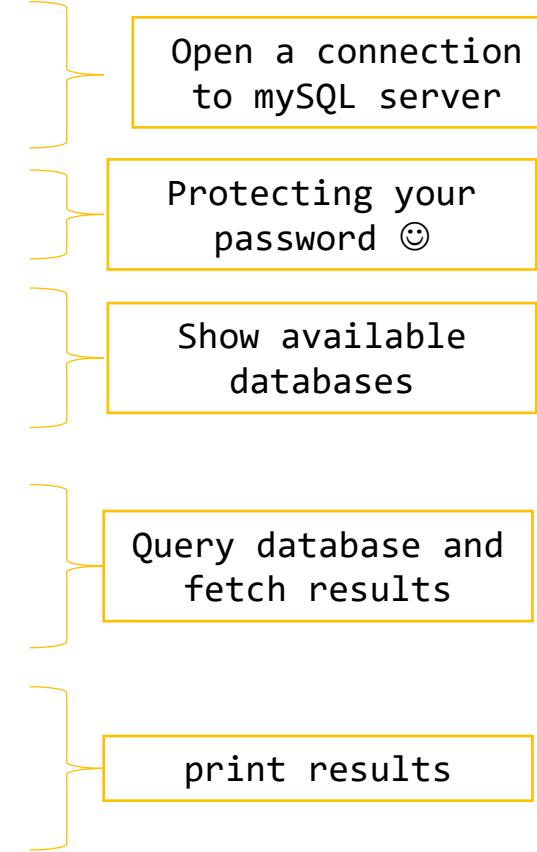
Mysql connector

SQL query to execute



Reading mySQL Data

```
• import mysql.connector  
• mydb = mysql.connector.connect(  
•     host="localhost",  
•     user="root",  
•     password=open("mysql_pass.txt", "r").readline()  
• )  
• mycursor = mydb.cursor()  
• mycursor.execute("SHOW databases")  
• print("AVAILABLE DATABASES:\n", mycursor.fetchall())  
  
• mycursor.execute("USE sakila")  
• mycursor.execute("SELECT * FROM customer")  
• myresult = mycursor.fetchall()  
  
• print("RESPONSE TO QUERY:")  
• for x in myresult:  
    print(x)
```



Writing Query output to CSV

```
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
```

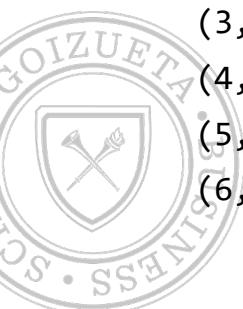
query data

```
fname = "query_out.csv"
fhandle = open(fname, "w", newline="")
csvfile = csv.writer(fhandle)
 csvfile.writerows(myresult)
fhandle.close()
```

Save to csv

```
(1, 'apple', 3.5, 4.99, 100.0)
(2, 'orange', 0.87, 1.49, 200.0)
(3, 'milk', 2.68, 3.99, 300.0)
(4, 'bread', 1.41, 1.99, 400.0)
(5, 'beer', 9.9, 19.99, 100.0)
(6, 'diaper', 18.21, 29.99, 50.0)
```

apple	3.50	4.99	922
orange	0.87	1.49	432
milk	2.68	3.99	761
bread	1.41	1.99	930
beer	9.90	19.99	883
diaper	18.21	29.99	364



Example - JOIN

- Extract shipping addresses for all customers and save the following columns: first name, last name, address, city, state, zip

```
SELECT first_name, last_name, line1, city, state, zip_code  
FROM customers c JOIN addresses a  
    ON c.customer_id = a.customer_id  
    AND c.shipping_address_id = a.address_id
```

\: allows string to be broken over multiple lines

```
query = 'SELECT first_name, last_name, line1, city, state, zip_code '\  
        'FROM customers c JOIN addresses a '\  
        'ON c.customer_id = a.customer_id '\  
        'AND c.shipping_address_id = a.address_id'  
getData(query)
```

Remember to leave a space at the end, else it will join the last word with the first word in next line



Example – MULTI-JOIN

- Extract customer name, item name ordered, item price, item discount and quantity from tables: orders, customers, order_items, and products

```
SELECT last_name, first_name, order_date, product_name, item_price, discount_amount, quantity
FROM customers c
    JOIN orders o ON c.customer_id = o.customer_id
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN products p ON oi.product_id = p.product_id
ORDER BY last_name, order_date, product_name
```

```
query = 'SELECT last_name, first_name, order_date, product_name, item_price, discount_amount,
quantity '\
        'FROM customers c '\
        'JOIN orders o ON c.customer_id = o.customer_id '\
        'JOIN order_items oi ON o.order_id = oi.order_id '\
        'JOIN products p ON oi.product_id = p.product_id '\
        'ORDER BY last_name, order_date, product_name'
getData(query)
```



Example – SUBQUERIES

- Extract which products have a list price that's greater than the average list price for all products?

```
SELECT product_name, list_price  
FROM products p  
WHERE list_price > (SELECT AVG(list_price) FROM products)  
ORDER BY list_price;
```

```
query = 'SELECT product_name, list_price '\  
'FROM products p '\  
'WHERE list_price > (SELECT AVG(list_price) FROM products) '\  
'ORDER BY list_price;'  
getData(query)
```



Additional aggregation functions:
<https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html>

Example – Data Manipulation

- Show the last 4 digits of card number used in orders and mask all previous digits with “X”

```
SELECT card_number,  
       LENGTH(card_number) AS card_number_length,  
       SUBSTRING(card_number, LENGTH(card_number)-3) AS last_four_digits,  
       CONCAT('XXXX-XXXX-XXXX-', SUBSTRING(card_number, LENGTH(card_number)-3)) AS  
formatted_number  
FROM orders;
```

```
query = 'SELECT card_number, '\  
       'LENGTH(card_number) AS card_number_length, '\  
       'SUBSTRING(card_number, LENGTH(card_number)-3) AS last_four_digits, '\  
       'CONCAT("XXXX-XXXX-XXXX-", SUBSTRING(card_number, LENGTH(card_number)-3)) AS  
formatted_number '\  
       'FROM orders;'  
getData(query)
```

Pass quotes as double quotes



SQL in Python

- SQL + Python/Pandas + ML = “Master of the Data Science”
- Why?
 - Bigger: Import/export/manipulate/analyze all kinds of data quickly
 - Faster: Use SQL engine for faster data process and storage (you can even connect to Spark, Hive, etc)
 - Cheaper: Massive volume of open-source libraries for data analysis and ODBC drivers are usually included for free!



SQL & Python

Google Cloud



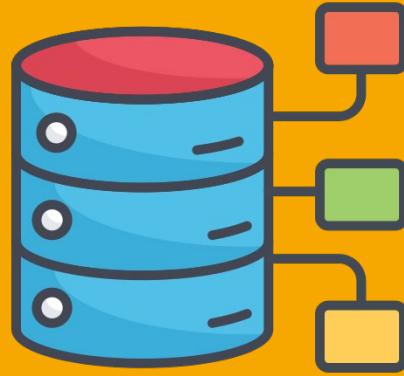
Rajiv Garg





SQL III & IV

Catchup



Data Warehousing

Need for Decision-Optimized Data Storage

- Business leaders need information to make plans, decisions, and assess results.
 - They need access to information such as:
 - What were the sales volumes by region and product category for the last 3 years?
 - Which of two new medications will result in the best outcomes (higher recovery rate and shorter hospital stay)?
- Executives and managers need information to:
 - get in-depth knowledge of their company's operations, review and monitor key performance indicators, keep track of how business factors change over time, and compare their company's performance relative to the competition and to industry benchmarks.
- Strategic information is not for running the day-to-day operations of the business.
 - Data needed for strategic decision making must be in a format suitable for easy analysis to spot trends.



Data Warehouses vs Transactional Systems

- Data captured by complex operational systems (OLTPs) is optimized to support well-defined transaction requirements.
 - Difficult to get needed information from data grounded in OLTPs.
- Data Warehouse (DW) / Business Intelligence (BI) systems almost never deal with one transaction at a time.
 - These systems are optimized for high-performance queries as users' questions often require hundreds of thousands of transactions be searched and compressed into an answer set.
- DW/BI systems typically demand that historical context be preserved to accurately evaluate the organization's performance over time.



Data Warehouse Architecture

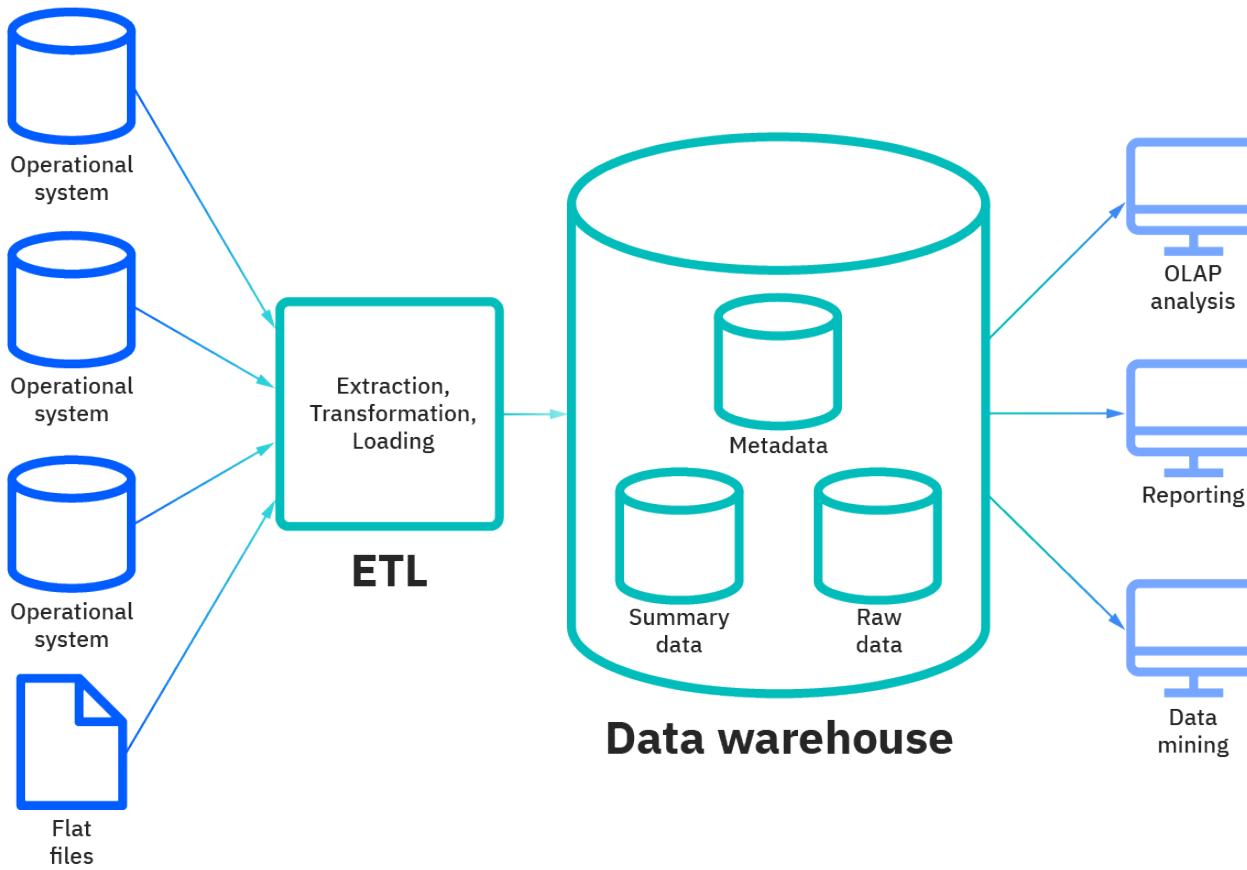


Image source: <https://www.ibm.com/cloud/learn/data-warehouse>



Operational vs. Informational Data

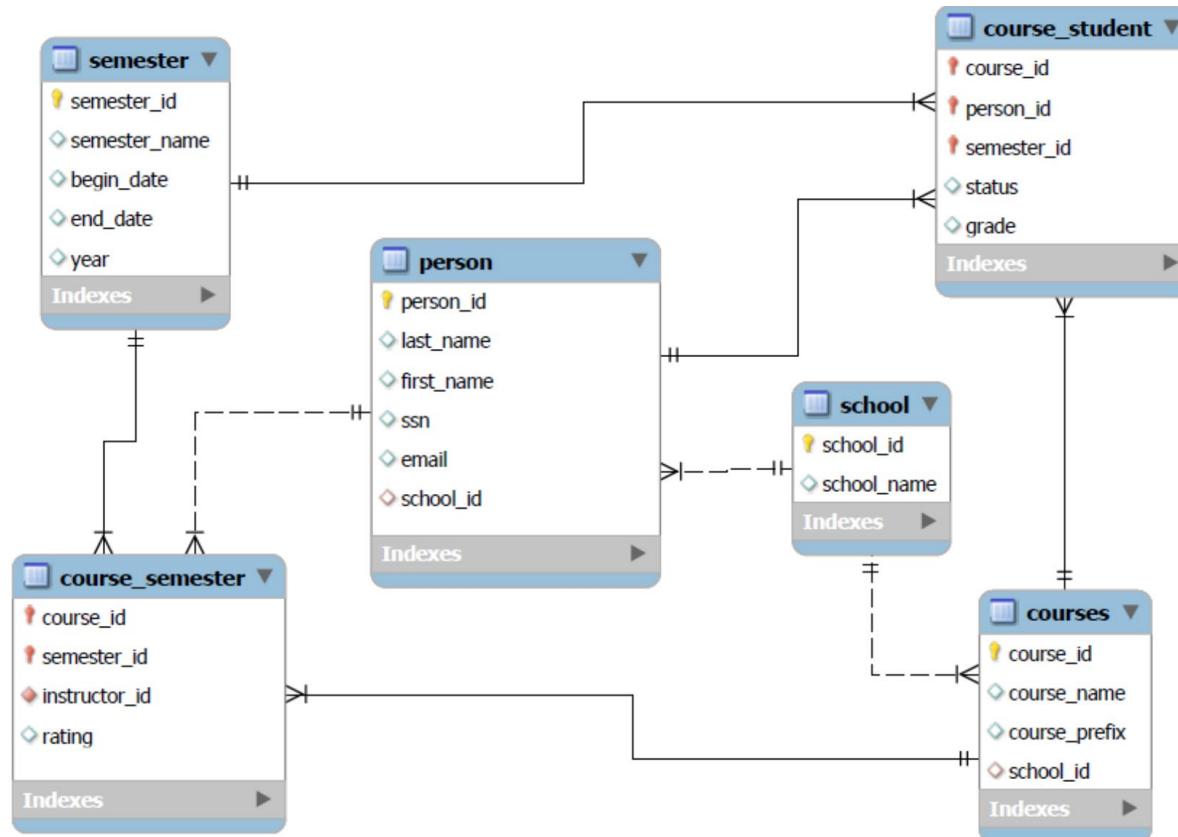
	Operational (OLTP)	Analytical (OLAP)
Data Content	Current Values	Historical, derived, summarized
Data Structure	Optimized for transactions (ER Models)	Optimized for complex queries (Star or Snowflake schema)
Data Volume	MB/GB of data	GB/TB/PB... of data
Access Frequency	High	Medium to low
Access Type	Read, update, delete	Read-only
Usage	Predictable, repetitive	Ad hoc, random, heuristic
Response Time	Sub-seconds	Several seconds to minutes
Users	Large number; operational & data workers	Relatively smaller number; data & knowledge workers



The Problem

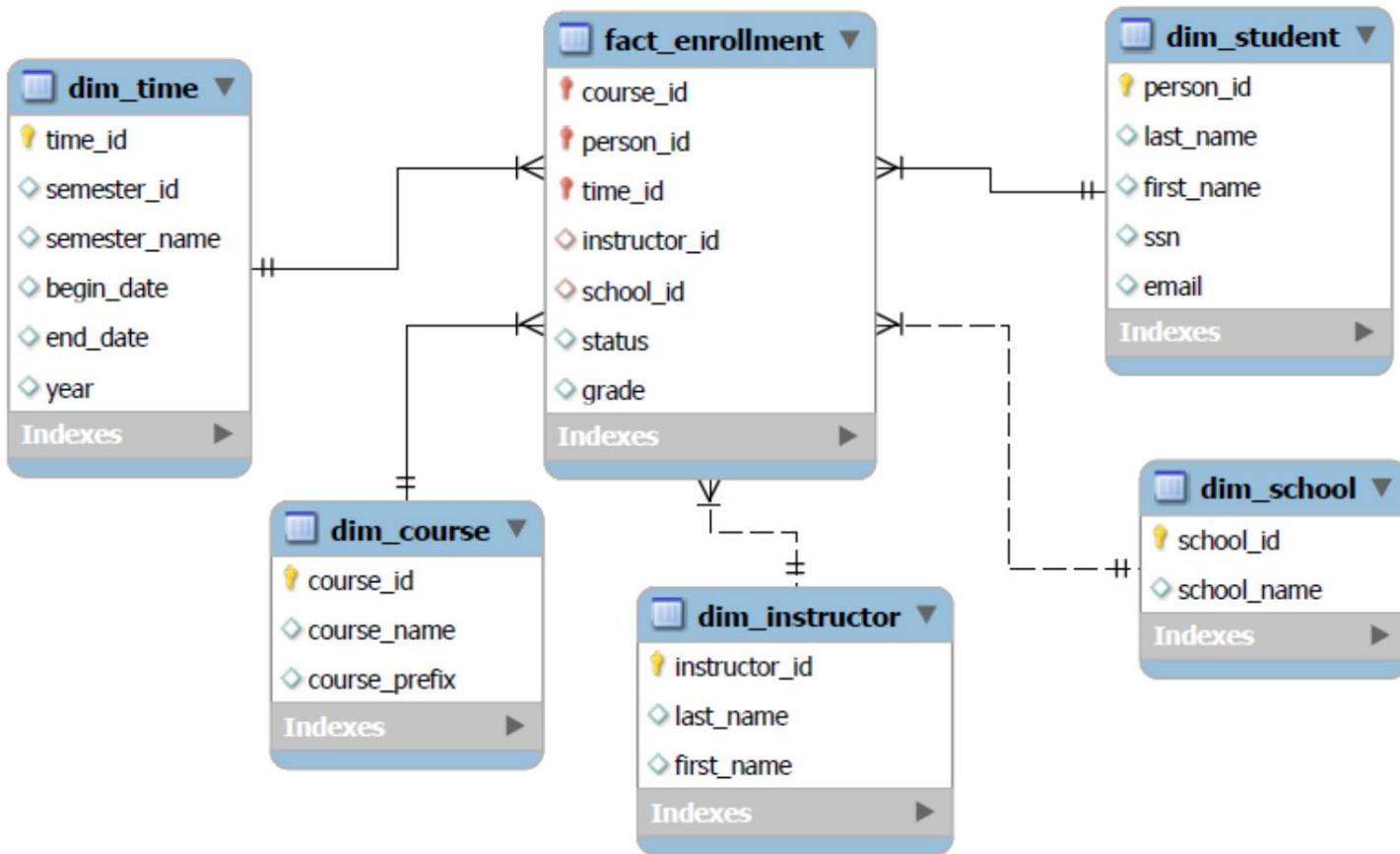
- Transactional models, while efficient for transaction processing, are not good for analytics

How do we determine the average grade in big data course in each semester?



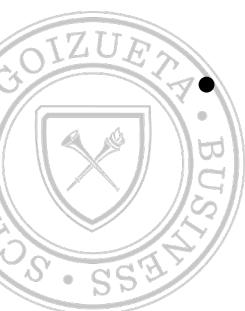
The Solution

- It would be nice to organize the data to be pulled out efficiently.

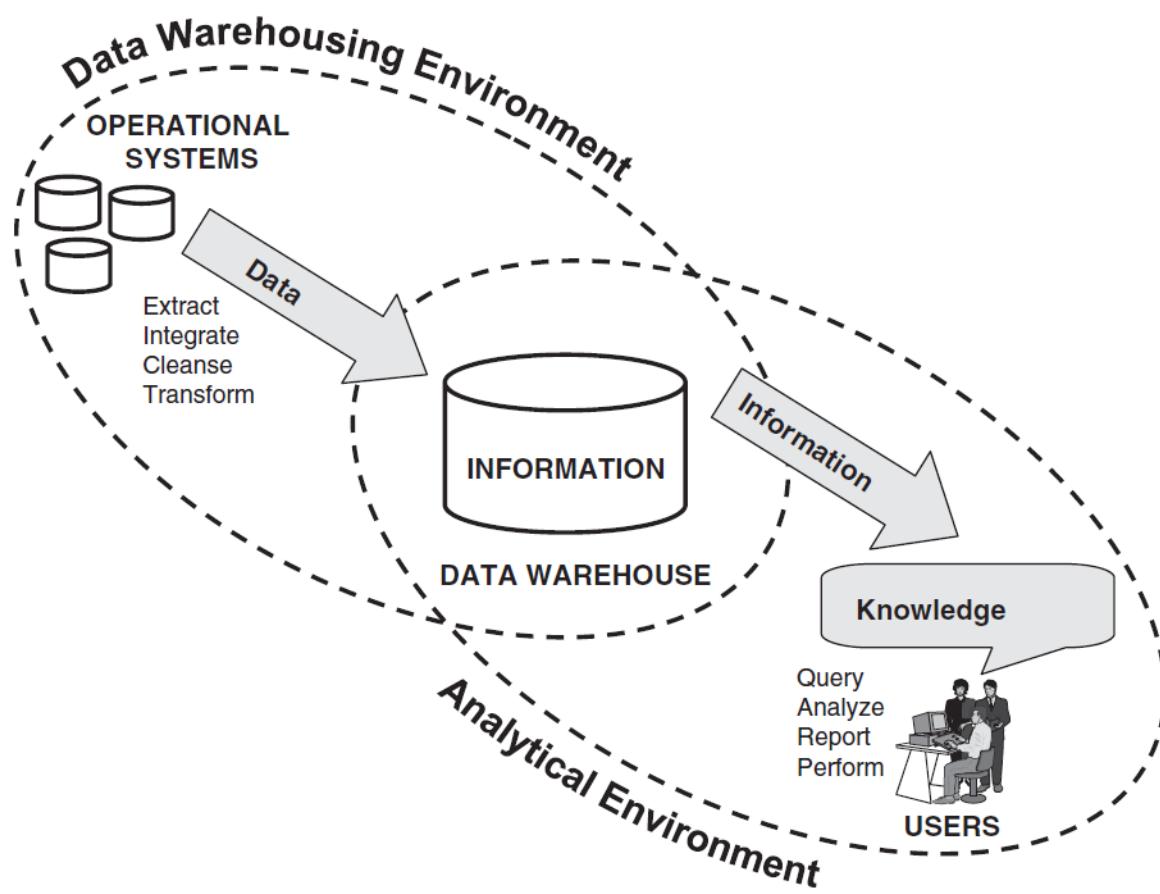


The concept of Data Warehouse

- “... a **subject-oriented, integrated, nonvolatile, and time variant collection of data** in support of management decisions.”
 - Managing the Data Warehouse, W. H. Inmon, John Wiley & Sons, December, 1996.
- “... a copy of transaction data specifically structured for query and analysis.”
 - The Data Warehouse Toolkit, R. Kimball, John Wiley & Sons, February, 1996.
- In summary: enterprise data that is extracted, transformed, and loaded into analytical tools for decision-making.



BI: Data Warehousing and Analytics



Examples of Uses of BI/DW

Industry	Use
Retail	Customer Loyalty
	Customer Service Effectiveness
Financial	Fraud Detection
	Profitability by product
Airlines	Route Profitability
	Customer Profitability
Manufacturing	Cost Reduction Opportunities
	Product Shipments
Non Profits	Giving Campaign Effectiveness
	Salvation Army Bell Ringer Effectiveness
Government	Manpower Planning
	School Academic Performance
Medical	Patient Risk for Disease
Technology/Consulting	Estimate future engagements



Let's Design!

- With access to normalized Sakila DB:
 - How would you identify demand by location (state) and time (year-month)?
 - What if you need the demand information by location and date frequently for your decisions?

```
SELECT city, year(rental_date),  
month(rental_date), COUNT(*)  
FROM customer c  
INNER JOIN address a  
ON c.address_id = a.address_id  
INNER JOIN city ct  
ON a.city_id = ct.city_id  
INNER JOIN rental r  
ON c.customer_id = r.customer_id  
GROUP BY city, year(rental_date),  
month(rental_date);
```

	city	year(rental_date)	month(rental_date)	COUNT(*)
▶	A Corua (La Corua)	2005	5	1
	A Corua (La Corua)	2005	6	4
	A Corua (La Corua)	2005	7	12
	A Corua (La Corua)	2005	8	11
	A Corua (La Corua)	2006	2	1
	Abha	2005	5	1
	Abha	2005	7	12
	Abha	2005	8	10
	Abha	2006	2	1
	Abu Dhabi	2005	5	3
	Abu Dhabi	2005	6	7
	Abu Dhabi	2005	7	13
	Abu Dhabi	2005	8	8



Dimensional Modeling

- Dimensional modeling is widely accepted as the preferred technique for presenting analytic data because it:
 - Delivers data that's understandable to a business user.
 - Delivers fast query performance.
- Dimension models are quite different from 3NF models which seek to remove data redundancies.
 - Normalized models might be too complicated for BI queries.
 - The complexity of users' unpredictable queries overwhelms the database optimizers.
 - A dimensional model contains the same information as a normalized model but packages the data in a format that delivers user understandability, query performance, and resilience to change.



Four-Step Dimensional Design Process

- Select the Business Process
 - e.g., taking orders, invoicing, receiving payments, handling service calls, registering students, etc.
 - Declare the Grain
 - specifying exactly what an individual fact table row represents.
 - Identify the Dimensions
 - “who, what, where, when, why, and how” associated with the event.
 - Identify the Facts
 - “What is the process measuring?”
- A Fact Table + Dimension Tables = A Dimensional Model



Fact Table

- The fact table in a dimensional model stores the performance measurements resulting from an organization's business process events.
 - A measurement event in the physical world has a one-to-one relationship to a single row in the corresponding fact table.
- The data on each row in a fact table is at a specific level of detail (i.e., grain)
 - e.g., one row per product sold on a sales transaction; usually the most finely grained data so that users can ask precise questions.
 - The most useful facts are numeric, and additive as BI applications rarely retrieve a single fact table row.
 - All fact tables have two or more foreign keys that connect to the dimension tables' primary keys.
 - The fact table generally has its own (multipart) primary key composed of a subset of the foreign keys.



Grain

- Grain conveys in business terms the level of detail associated with the fact table measurements.
 - It provides the answer to the question, “how do you describe a single row in the fact table?”
- Examples of grain declaration
 - One row per scan of an individual product on a customer’s sales transaction
 - One row per line item on a bill from a doctor
 - One row per individual boarding pass scanned at an airport gate
 - One row per daily snapshot of the inventory levels for each item in a warehouse
 - One row per bank account each month



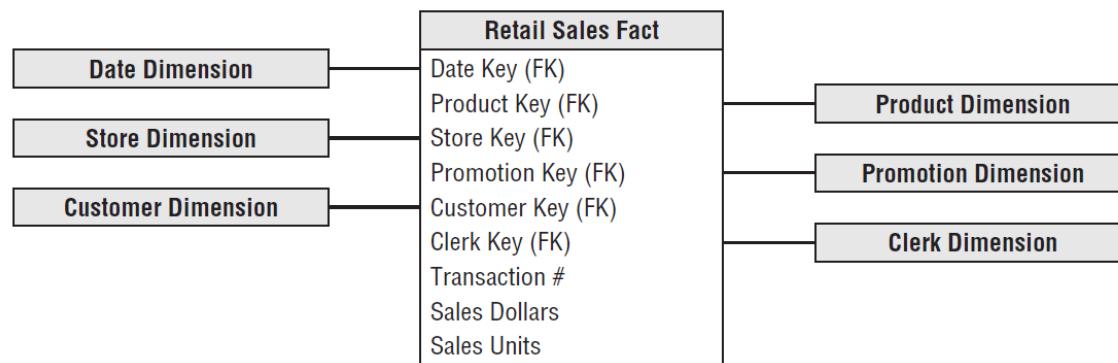
Dimension Tables

- Dimension tables (and attributes) serve as the primary source of query constraints, groupings, and report labels.
 - They describe the “who, what, where, when, how, and why” associated with the event.
 - Critical to making the DW/BI system usable and understandable.
- Dimension tables often have many columns or attributes but fewer rows than fact tables.
 - Each dimension is defined by a single primary key.
 - It is sometimes unclear whether a numeric data element is a fact or dimension attribute
 - Depends on whether the column is a measurement that takes on lots of values and participates in calculations (making it a fact) or
 - it is a discretely valued description that is more or less constant and participates in constraints and row labels (making it a dimensional attribute).
 - Often contain hierarchical relationships (city → state → region)



STAR Schema

- Each business process is represented by a dimensional model that consists of
 - a fact table containing the event's numeric measurements surrounded by
 - a collection of dimension tables that contain textual context that was true at the moment the event occurred.



Advantages of the STAR schema

- Simplicity
 - Easy for users to understand
- Optimizes navigation
- Most suitable for query processing
- Enables specific performance schemes
 - e.g., specialized index
- Extensibility
 - Easily accommodates change (but not that easily!)



Retail Case Study

- The business has 100 grocery stores in five states.
- Each store has a full complement of departments, including grocery, frozen foods, dairy, etc.
- Each store has approximately 60,000 SKUs.
- Data is collected at several interesting places in a grocery store.
 - Some of the most useful data is collected at the cash registers as customers purchase products.
 - Other data is captured at the store's back door where vendors make deliveries.
- Some of the most significant management decisions have to do with pricing and promotions.

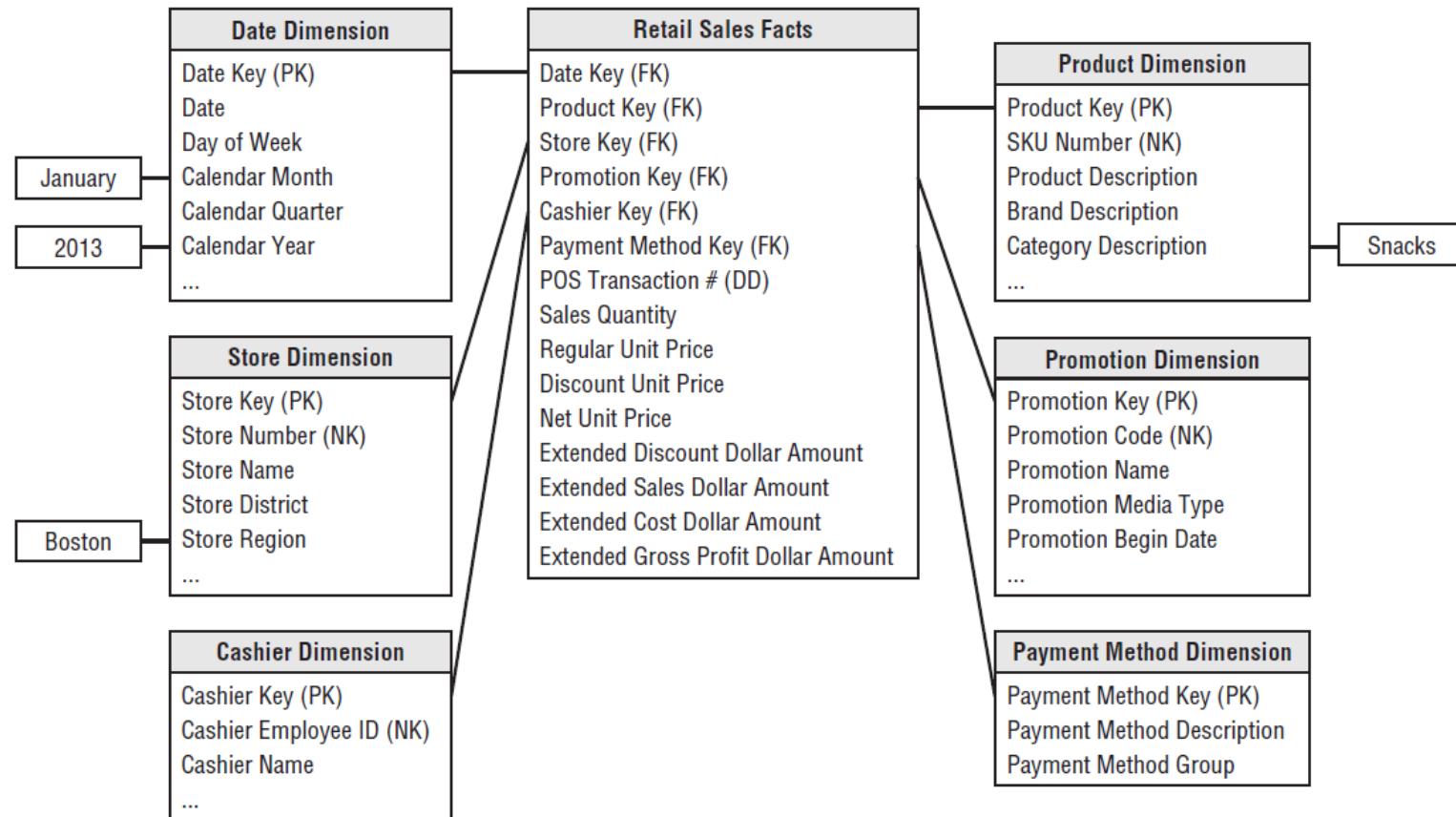


Four-Step Dimensional Design Process

- Select the Business Process:
 - The business process you're modeling is POS retail sales transactions.
 - Enables the business users to analyze which products are selling in which stores on which days under what promotional conditions in which transactions.
- Declare the Grain:
 - An individual product on a POS transaction.
- Identify the Dimensions:
 - Product, transaction, date, store, promotion, cashier, payment method, etc.
- Identify the Facts:
 - Sales quantity, per unit regular, discount, and net paid prices, and extended discount and sales dollar amounts.

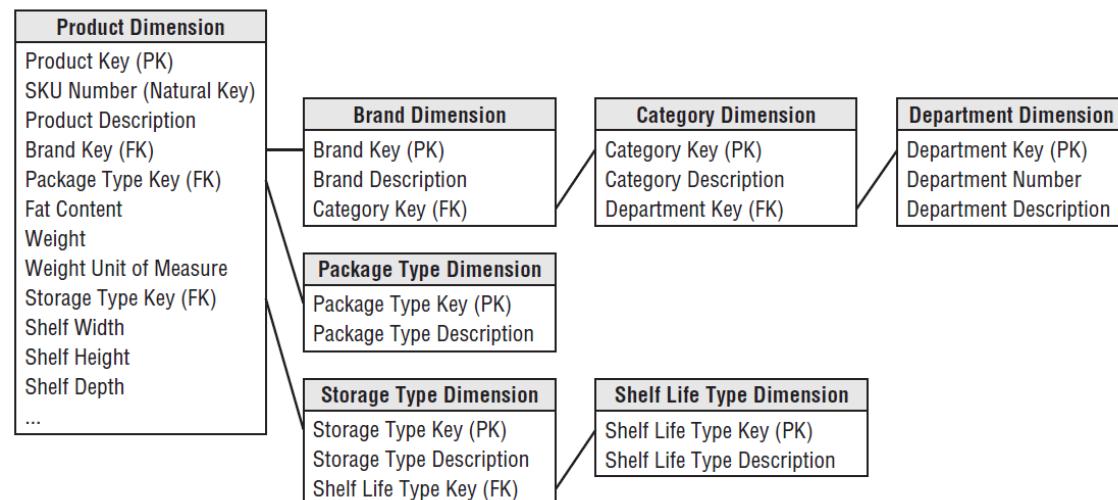


Retail Schema



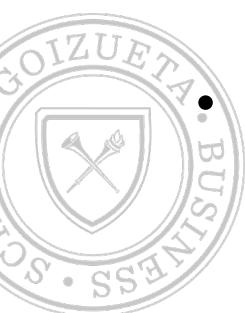
Snowflake product dimension

- Snowflaking is a method of normalizing the dimension tables that are in a STAR schema:
 - Pros: Small savings in storage space; normalized structures are easier to update and maintain
 - Cons: Less intuitive; degrades query performance; decreases ability to browse through the contents



Comparison of Schemas - Snowflake

- Star schema
 - De-normalized dimension tables
 - Faster load/query/analysis performance
 - Potentially more intuitive to users
- Snowflake
 - Partially normalized dimension tables
 - Potentially faster setup
 - Reduces the size of dimension table
 - Disadvantages
 - Increases presentation complexity for end users
 - Increases number of joins required to answer queries, resulting in performance degradation
- In most cases you can avoid snowflaking unless there is some operational reason to do so.



Benefits of Data Warehousing

- **Consolidation**
 - Single query engine can access multiple data sources concurrently
- **Isolation**
 - Access does not interfere with Operational OLTP data sets
- **History**
 - Can store and manage much larger data sets with longer retention than Operational OLTP, providing access to more historical data
- **Consistency**
 - A single data model regardless of the data sources' model
- **Performance**
 - Excellent query responsiveness, with no impact to production data sources
- **Value**
 - Access to better information for better business intelligence



Data Mart

- Simplified and faster data warehousing!
 - ETL data associated with a functional decision-making
 - Marketing
 - Supply chain
 - Each data mart can be created from:
 - Enterprise-wide Data Warehouse
 - Directly from OLTP sources
 - Combination of the DW and OLTP sources



Data Warehouse Architecture

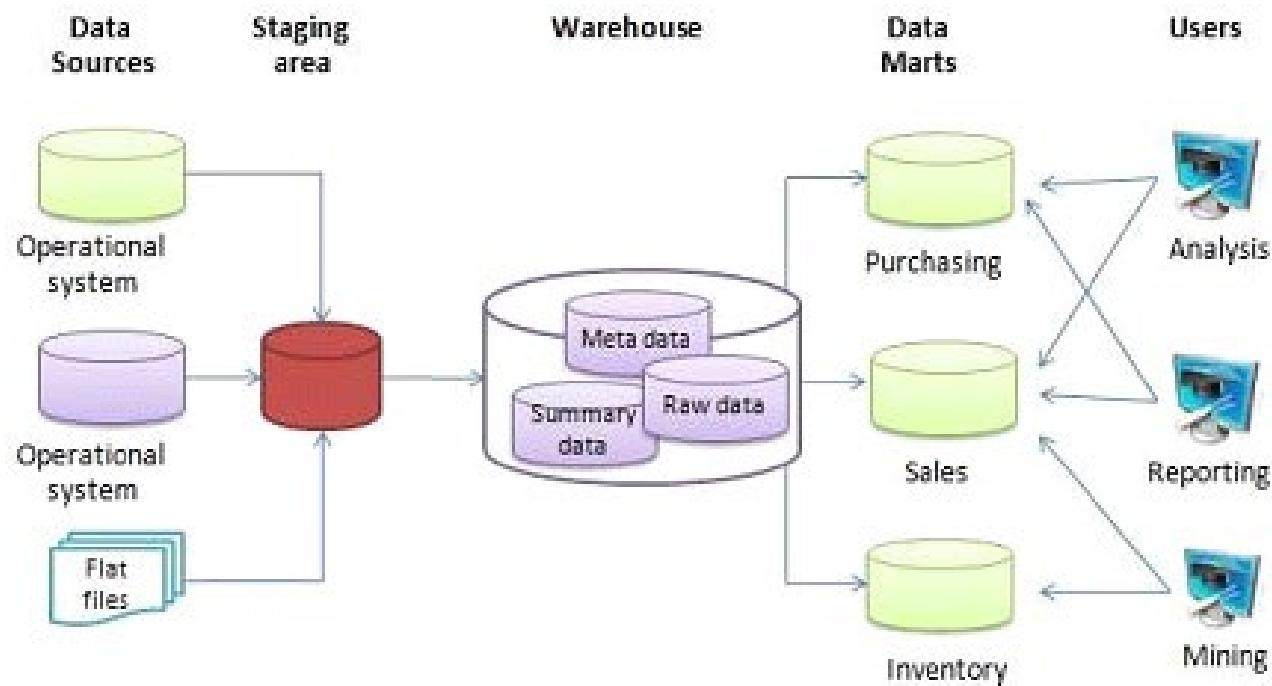
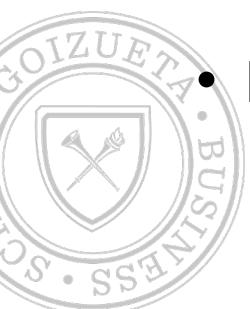


Image source: https://en.wikipedia.org/wiki/Data_warehouse



Data Warehouse Tools and Vendors

- Data Warehousing market includes hardware, database software, and tools
 - Multi-billion-dollar market segment
 - A maturing market
 - warehouses deployed in virtually every industry
- Database Servers
 - Oracle, Microsoft SQL Server, Sybase, IBM DB2, ...
- ETL Tools
 - Informatica PowerPlay, Ascential DataStage (IBM), Hyperion Application Link, Oracle PL/SQL, Microsoft Data Transformation Services, ...



Data Warehouse Market is Changing

- User-friendly tools for analysis, visualization
 - Tableau, Spotfire, SAS visual analytics, ...
 - Reporting Services, ...
- Advances in processing & storage technology
 - Parallel processing, Hadoop, NoSQL, in-memory database, caching, columnar databases
- The need for real-time streaming / analytics

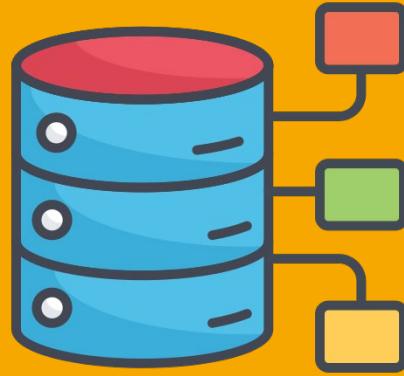


DW Additional Trainings

- AWS RedShift Lab: <https://zacks.one/aws-redshift-lab/>
- AWS RedShift Immersion Labs: <https://catalog.us-east-1.prod.workshops.aws/workshops/9f29cdba-66c0-445e-8cbb-28a092cb5ba7/en-US>
- Snowflake Hands-on Training: <https://www.snowflake.com/webinar/virtual-hands-on-labs/2022-09-21/>

Lab 2





Data Warehousing

catchup



NoSQL

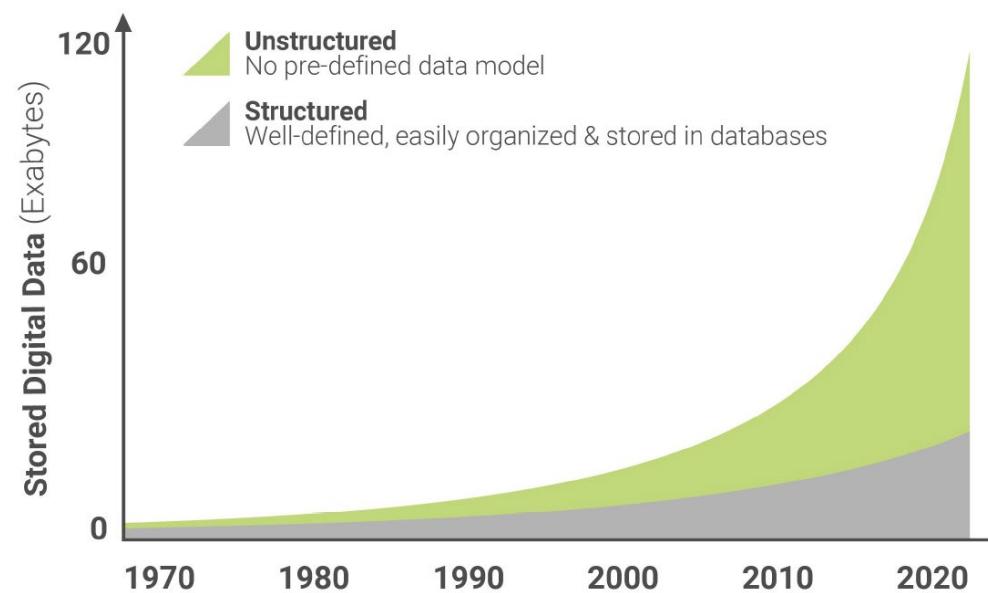
Data Management Systems

- For the past five decades RDBMS have been successfully:
 - Storing,
 - Serving, &
 - Processing data
- RDBMS have been widely adopted for:
 - online transaction processing (OLTP) and
 - online analytical processing (OLAP).
- But, in 21st century...



Something Changed!

- Companies such as Google, Amazon, Yahoo, Facebook and LinkedIn:
 - store, serve and process huge amount of data;
 - work with different types of data (often semi- or un-structured)



Source: <https://www.datanami.com/2019/01/14/from-oscar-to-ai-mining-visual-assets-for-fun-and-profit/>



Something Changed!

- RDBMS challenges:
 - expensive licensing
 - complex application logic
 - evolving data models
- Need for systems that:
 - could work with different kinds of data formats
 - do not require strict schema
 - are easily scalable



NoSQL

- The term NoSQL was coined in 1998 by Carlo Strozzi.
- NoSQL is generally interpreted as Not Only SQL, rather than NO SQL.
 - It is meant to convey that many applications need systems other than traditional relational SQL systems to augment data management needs.
- Most NoSQL systems are distributed databases that are:
 - Used for: storage and retrieval of “big data” (un/semi-structured),
 - To provide: high performance, availability, data replication, and scalability
 - As opposed to: immediate data consistency, powerful query languages, and structured data storage



Emergence of NoSQL systems

- In 2006, Google published BigTable paper.
- In 2007, Amazon presented DynamoDB.
- It didn't take long for all these ideas to diffuse to:
 - several open-source projects (Hbase, Cassandra) as well as
 - other companies (Facebook, Twitter, ...)
- An SFO meetup during Hadoop summit in 2009 used #NoSQL for getting all unstructured data engineers to come together
 - The name caught on like wildfire!

<https://hostingdata.co.uk/nosql-database/> lists more than 225 NoSQL databases.



NoSQL systems

- NoSQL is an accidental neologism with no prescriptive definition
- Attracted attention because of the need to support large volume of data on clusters
 - Relational databases are not designed to run efficiently on clusters
- Common characteristics of NoSQL systems are:
 - Running on clusters
 - Schema-less
 - Usually open-source
 - Usually used for online datasets



SQL vs NoSQL

SQL based databases		NoSQL
Data size	Gigabytes	Petabytes
Data storage	Data stored in a relational model (tables)	A variety of (non-relational) storage models; e.g., document, key-value, columnar, graph
Updates	Read and write multiple times	Usually write once, read many times
Data focus	Suited for structured data	Suited for unstructured data
Schemas and Flexibility	Each record conforms to fixed schema; schema-on-write	Schema-less, or dynamic schemas, or schema-on-read; each “row” does not have to contain data for each “column”
Scalability	Scaling is vertical (bigger server): Expensive and difficult	Scaling is horizontal (more nodes): Cheap and easy
ACID Compliancy	ACID compliant (Atomicity, Consistency, Isolation, Durability)	Many NoSQL solutions sacrifice ACID compliance for performance and scalability



Source: <http://dataconomy.com/sql-vs-nosql-need-know/>

CAP theorem for Databases

- Consistency
 - All nodes will have the same copies of replicated data item available for various transactions.
 - Availability
 - Each read/write request for a data item will either be processed successfully or receive a message that the operation cannot be completed.
 - Partition tolerance
 - System can continue operating if the network connecting the nodes has a fault that results in partitions.
- CAP theorem: it is not possible to guarantee all three of these
- ..at most two of them for any networked shared-data system.



NoSQL and the CAP theorem

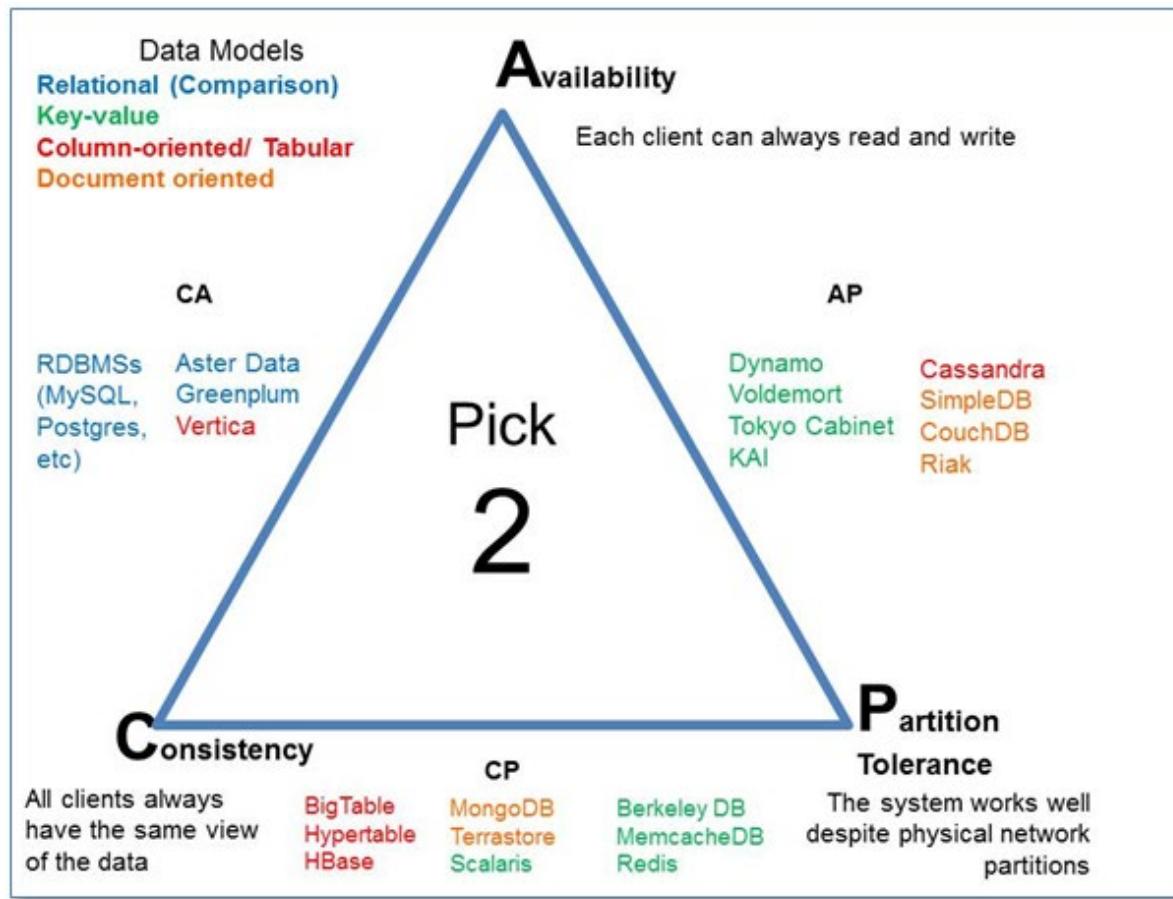
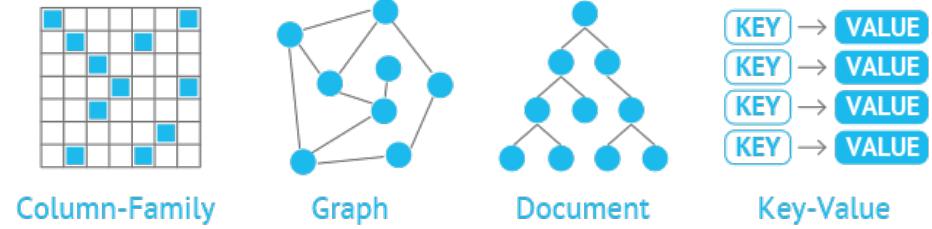


Image source: <https://www.mysoftkey.com/architecture/understanding-of-cap-theorem/>



NOSQL Databases



- **Key-value stores:**
 - a simple data model based on fast access by the key to the value associated with the key; the value can be a record or an object or a document or even have a more complex data structure.
- **Document-based:**
 - data is stored in the form of documents using well-known formats, such as JSON. Documents are accessible via their id - but can also be accessed rapidly using other indexes.
- **Column-based (or wide column):**
 - a table is partitioned by column into column families, where each column family is stored in its own files. They also allow versioning of data values.
- **Graph-based:**
 - Data is represented as graphs, and related nodes can be found by traversing the edges using path expressions.



NoSQL Key-Value Stores

- Simplest form of database management systems.
 - Can only store pairs of keys and values, as well as retrieve values when a key is known.
- Normally not adequate for complex applications.
 - But the simplicity makes such systems attractive - especially when applications mainly require lookups
- Many systems provide further extensions for seamless transition to document stores and wide column stores.



NoSQL Key-Value Stores

- Data model: (key, value) pairs
 - The key is a unique identifier associated with a data item and is used to locate data item rapidly.
 - The value is the data item itself, and it can have very different formats for different key-value storage systems.
 - In some cases, the value is just an array of bytes, and the application using the key-value store must interpret the structure of the data value.
 - In other cases, some standard formatted data is allowed;
 - for example, structured data rows (tuples) like relational data, or semi-structured data using JSON or some other self-describing data format.
- Basic Operations:
 - Insert(key,value), Fetch(key), Update(key), Delete(key)



Key-Value Stores: Databases

- Memcached - general-purpose distributed memory-caching system.
- Redis - most popular, similar to Memcached but more features
- Couchbase (previously Membase) - open-source, distributed multi-model NoSQL document-oriented database software package
- DynamoDB – Amazon’s Key-Value Database
 - Voldemort – open-source and based on many of the techniques proposed for DynamoDB (designed by LinkedIn)
 - Riak – based on Amazon’s DynamoDB (Erlang)
- Leveldb – fast and lightweight key/value database library by Google
- Oracle NoSQL Database





Redis

- Advanced key-value store.
 - Can contain strings, hashes, lists, sets, sorted sets, etc.
- Typically holds (caches) the whole dataset in memory.
 - Main difference from other systems
- Data persistence achieved by:
 - Periodic snapshot of data saved on drive or
 - or by appending each command to server log (rarely used).
- Use cases:
 - Latest item listings on website, real-time tracking, leader boards, queues, caching, etc. (place where fast lookups are important)



DynamoDB (Amazon) Overview



Amazon DynamoDB

- The basic data model in DynamoDB uses the concepts of tables, items, and attributes.
 - A table in DynamoDB does not have a schema; it holds a collection of self-describing items.
 - A table will hold a collection of items, and each item is a self-describing record (or object).
 - Each item will consist of a number of (attribute, value) pairs, and attribute values can be single-valued or multivalued.
- Each table has a partition key to rapidly locate the items in the table.
 - The partition key is the “key”, and the item is the “value” for the DynamoDB key-value store.
- Simple operations include get, put, and delete.



Example Data for Products

- ```
{'ProductCatalog': [{}{'PutRequest': {'Item': {'PageCount': {'N': '500'}, 'ISBN': {'S': '111-111111111'}, 'Price': {'N': '2'}, 'InPublication': {'BOOL': True}, 'ProductCategory': {'S': 'Book'}, 'Title': {'S': 'Book 101 Title'}, 'Id': {'N': '101'}, 'Dimensions': {'S': '8.5 x 11.0 x 0.5'}, 'Authors': {'L': [{"S": "Author1"}]}}, {}{'PutRequest': {'Item': {'PageCount': {'N': '600'}, 'ISBN': {'S': '222-2222222222'}, 'Price': {'N': '20'}, 'InPublication': {'BOOL': True}, 'ProductCategory': {'S': 'Book'}, 'Title': {'S': 'Book 102 Title'}, 'Id': {'N': '102'}, 'Dimensions': {'S': '8.5 x 11.0 x 0.8'}, 'Authors': {'L': [{"S": "Author1"}, {"S": "Author2"}]}}, {}{'PutRequest': {'Item': {'PageCount': {'N': '600'}, 'ISBN': {'S': '333-3333333333'}, 'Price': {'N': '2000'}, 'InPublication': {'BOOL': False}, 'ProductCategory': {'S': 'Book'}, 'Title': {'S': 'Book 103 Title'}, 'Id': {'N': '103'}, 'Dimensions': {'S': '8.5 x 11.0 x 1.5'}, 'Authors': {'L': [{"S": "Author1"}, {"S": "Author2"}]}}, {}{'PutRequest': {'Item': {'Description': {'S': '201 Description'}, 'Price': {'N': '100'}, 'ProductCategory': {'S': 'Bicycle'}, 'Brand': {'S': 'Mountain A'}, 'Title': {'S': '18-Bike-201'}, 'Id': {'N': '201'}, 'BicycleType': {'S': 'Road'}, 'Color': {'L': [{"S": "Red"}, {"S": "Black"}]}}, {}{'PutRequest': {'Item': {'Description': {'S': '202 Description'}, 'Price': {'N': '200'}, 'ProductCategory': {'S': 'Bicycle'}, 'Brand': {'S': 'Brand-Company A'}, 'Title': {'S': '21-Bike-202'}, 'Id': {'N': '202'}, 'BicycleType': {'S': 'Road'}, 'Color': {'L': [{"S": "Green"}, {"S": "Black"}]}}, {}{'PutRequest': {'Item': {'Description': {'S': '203 Description'}, 'Price': {'N': '300'}, 'ProductCategory': {'S': 'Bicycle'}, 'Brand': {'S': 'Brand-Company B'}, 'Title': {'S': '19-Bike-203'}, 'Id': {'N': '203'}, 'BicycleType': {'S': 'Road'}, 'Color': {'L': [{"S": "Red"}, {"S": "Green"}, {"S": "Black"}]}}, {}{'PutRequest': {'Item': {'Description': {'S': '204 Description'}, 'Price': {'N': '400'}, 'ProductCategory': {'S': 'Bicycle'}, 'Brand': {'S': 'Brand-Company B'}, 'Title': {'S': '18-Bike-204'}, 'Id': {'N': '204'}, 'BicycleType': {'S': 'Mountain'}, 'Color': {'L': [{"S": "Red"}]}}, {}{'PutRequest': {'Item': {'Description': {'S': '205 Description'}, 'Price': {'N': '500'}, 'ProductCategory': {'S': 'Bicycle'}, 'Brand': {'S': 'Brand-Company C'}, 'Title': {'S': '18-Bike-204'}, 'Id': {'N': '205'}, 'BicycleType': {'S': 'Hybrid'}, 'Color': {'L': [{"S": "Red"}, {"S": "Black"}]}}]}]
```



# Example Data for Products

| Id           | Authors                                     | BicycleType      | Brand             | Color                                        | Description              | Dimensions                | ISBN                    | InPublication   | PageCount    | Price         | ProductCategory  | Title             |
|--------------|---------------------------------------------|------------------|-------------------|----------------------------------------------|--------------------------|---------------------------|-------------------------|-----------------|--------------|---------------|------------------|-------------------|
| {'N': '101'} | {'L': [{"S': 'Author1'}]}                   | NaN              | NaN               | NaN                                          | NaN                      | {'S': '8.5 x 11.0 x 0.5'} | {'S': '111-1111111111'} | {'BOOL': True}  | {'N': '500'} | {'N': '2'}    | {'S': 'Book'}    | 'Book 101 Title'} |
| {'N': '102'} | {'L': [{"S': 'Author1"}, {"S': 'Author2"}]} | NaN              | NaN               | NaN                                          | NaN                      | {'S': '8.5 x 11.0 x 0.8'} | {'S': '222-2222222222'} | {'BOOL': True}  | {'N': '600'} | {'N': '20'}   | {'S': 'Book'}    | 'Book 102 Title'} |
| {'N': '103'} | {'L': [{"S': 'Author1"}, {"S': 'Author2"}]} | NaN              | NaN               | NaN                                          | NaN                      | {'S': '8.5 x 11.0 x 1.5'} | {'S': '333-3333333333'} | {'BOOL': False} | {'N': '600'} | {'N': '2000'} | {'S': 'Book'}    | 'Book 103 Title'} |
| {'N': '201'} | NaN                                         | {S': 'Road'}     | 'Mountain A'      | {'S': 'Red'}, {"S': 'Black'}}                | {'S': '201 Description'} | NaN                       | NaN                     | NaN             | NaN          | {'N': '100'}  | {'S': 'Bicycle'} | '18-Bike-201'     |
| {'N': '202'} | NaN                                         | {S': 'Road'}     | 'Brand-Company A' | {'S': 'Green'}, {"S': 'Black'}}              | {'S': '202 Description'} | NaN                       | NaN                     | NaN             | NaN          | {'N': '200'}  | {'S': 'Bicycle'} | '21-Bike-202'     |
| {'N': '203'} | NaN                                         | {S': 'Road'}     | 'Brand-Company B' | {'S': 'Red'}, {"S': 'Green"}, {"S': 'Bl...'} | {'S': '203 Description'} | NaN                       | NaN                     | NaN             | NaN          | {'N': '300'}  | {'S': 'Bicycle'} | '19-Bike-203'     |
| {'N': '204'} | NaN                                         | {S': 'Mountain'} | 'Brand-Company B' | {'L': [{"S': 'Red'}]}                        | {'S': '204 Description'} | NaN                       | NaN                     | NaN             | NaN          | {'N': '400'}  | {'S': 'Bicycle'} | '18-Bike-204'     |
| {'N': '205'} | NaN                                         | {S': 'Hybrid'}   | 'Brand-Company C' | {'L': [{"S': 'Red"}, {"S': 'Black'}]}        | {'S': '205 Description'} | NaN                       | NaN                     | NaN             | NaN          | {'N': '500'}  | {'S': 'Bicycle'} | '18-Bike-204'     |



# AWS DynamoDB

- AWS Console (DynamoDB UI)
- NoSQL Workbench
  - <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/workbench.html>
- AWS Console (with Cloud9 IDE with Python)
  - <https://aws.amazon.com/getting-started/hands-on/create-manage-nonrelational-database-dynamodb/>
- Local machine (with Python IDE)
  - Could also use local DynamoDB instance (for testing):  
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.DownloadingAndRunning.html>

Developer and API Guides:

<https://docs.aws.amazon.com/dynamodb/index.html>



# Try – AWS DynamoDB Console

- Startup:
  - Login to AWS console (AWS Academy)
  - Select DynamoDB
- Tasks:
  - Create table (music)
    - Select partition key (artist)
    - Select sort key (song)
    - Autoscaling
    - Create
  - Insert data
    - “Items” tab
    - Create item (repeat 4-5 times)
  - Query data
    - Select “query”
    - Search for title/artist/etc.

What is partition key?



# Try – Cloud9 IDE

- Startup:
  - Login to AWS console (AWS Academy)
  - Select Cloud9 IDE
    - Create environment
    - Get sample code
      - curl -sL https://s3.amazonaws.com/ddb-deep-dive/dynamodb.tar | tar -xv
  - Install Boto3 (AWS SDK)
    - sudo pip install boto3
    - (OR python -m pip install --user boto3)
- Tasks:
  - Create\_table (boto3: client.create\_table)
  - Insert\_items (boto3: table.batch\_writer(), batch.put\_item())
  - Get\_items (boto3: table.get\_item())
  - Query\_items (boto3: table.query())

Sample py code for dynamodb

Keytype: HASH, RANGE

S - String attribute  
N - Number attribute  
B - Binary attribute

Single item

Multiple items

Observe: we did not define any other attribute when creating table

# Try - Cloud9 IDE

- Get Item:
  - `resp = table.get_item(Key={"Author": "John Grisham", "Title": "The Rainmaker"})`
- Query Item:
  - `resp = table.query(KeyConditionExpression=Key('Author').eq('John Grisham'))`
  - `resp = table.query(KeyConditionExpression=Key('Author').eq('John Grisham') & Key('Title').between('A', 'U'))`
- Additional query options:
  - <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html>

Recall: that this is a Key-Value Data store. So key is the only thing indexed and you need it to query the data. What if you want to query on another attribute in “value”?



# Try – Cloud9 IDE

- Scan entire table:

| GameScores |                   |          |                       |      |        |     |
|------------|-------------------|----------|-----------------------|------|--------|-----|
| UserId     | GameTitle         | TopScore | TopScoreDateTime      | Wins | Losses |     |
| "101"      | "Galaxy Invaders" | 5842     | "2015-09-15:17:24:31" | 21   | 72     | ... |
| "101"      | "Meteor Blasters" | 1000     | "2015-10-22:23:18:01" | 12   | 3      | ... |
| "101"      | "Starship X"      | 24       | "2015-08-31:13:14:21" | 4    | 9      | ... |
| "102"      | "Alien Adventure" | 192      | "2015-07-12:11:07:56" | 32   | 192    | ... |
| "102"      | "Galaxy Invaders" | 0        | "2015-09-18:07:33:42" | 0    | 5      | ... |

- Or

- Secondary Index

|       |                   |      |                       |    |    |     |
|-------|-------------------|------|-----------------------|----|----|-----|
| "103" | "Attack Ships"    | 3    | "2015-10-19:01:13:24" | 1  | 8  | ... |
| "103" | "Galaxy Invaders" | 2317 | "2015-09-11:06:53:00" | 40 | 3  | ... |
| "103" | "Meteor Blasters" | 723  | "2015-10-19:01:13:24" | 22 | 12 | ... |
| "103" | "Starship X"      | 42   | "2015-07-11:06:53:00" | 4  | 19 | ... |

GameTitleIndex

| GameTitle         | TopScore | UserId | ... |
|-------------------|----------|--------|-----|
| "Alien Adventure" | 192      | "102"  | ... |
| "Attack Ships"    | 3        | "103"  | ... |
| "Galaxy Invaders" | 0        | "102"  | ... |
| "Galaxy Invaders" | 2317     | "103"  | ... |
| "Galaxy Invaders" | 5842     | "101"  | ... |
| "Meteor Blasters" | 723      | "103"  | ... |
| "Meteor Blasters" | 1000     | "101"  | ... |
| "Starship X"      | 24       | "101"  | ... |
| "Starship X"      | 42       | "103"  | ... |

Source: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html>



# Try – Cloud9 IDE

- Key-Value Pairs: searching/querying on keys
  - (secondary) Index: for search/query on other attributes
- Query with Index:
  - <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html>
- Tasks:
  - Create\_secondary\_index:
    - AttributeDefinitions (type: HASH)
    - GlobalSecondaryIndexUpdates (attribute type: HASH)
  - Wait for index creation: (table.global\_secondary\_indexes != 'ACTIVE')
  - Query\_with\_index: table.query(IndexName="CategoryIndex")



# Try – local Python

- Main change in accessing the database:
  - From:
    - `client = boto3.client('dynamodb', region_name='us-east-1')`
  - To:
    - `client = boto3.resource('dynamodb', aws_access_key_id='', aws_secret_access_key='', region_name='us-east-1')`
- Tasks:
  - Copy all py code from Cloud9
  - Get IAM for remote access
  - Download credentials (csv file) and add it to your Python code



# Key-Value Stores Summary

- Pros:
  - incredibly fast,
  - very scalable (able to scale horizontally),
  - efficient,
  - fault-tolerant,
  - simple model.
- Cons:
  - many data structures (objects) can't be easily modeled as key-value pairs.
- Well-suited for applications that have frequent small reads and writes along with simple data models.



# From Key-Value to Document Stores

- Key-value stores allow quick reads and updates, but we often want to query on other attributes as well
- If we have a JSON-type item, we can add indexes on attributes and allow querying on attributes as well
- Next class...





# NoSQL I

catchup

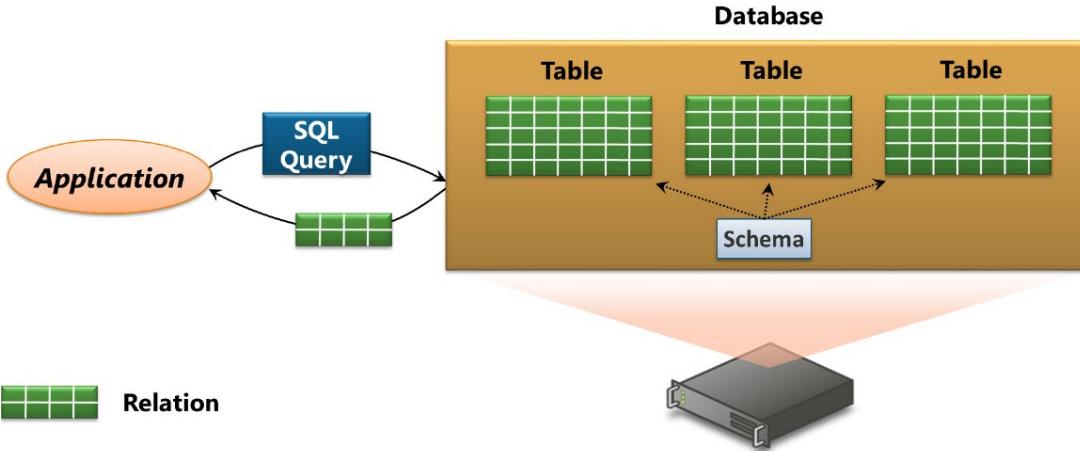


# NoSQL II

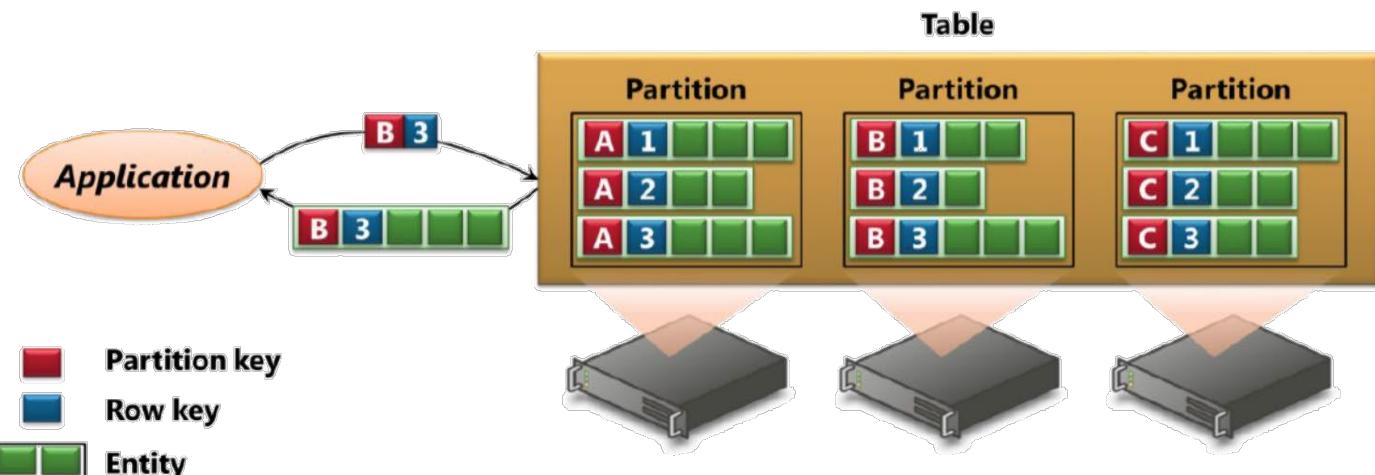
DocumentDB

# Key-value data stores (Azure view)

- Relational:



- Key-value:

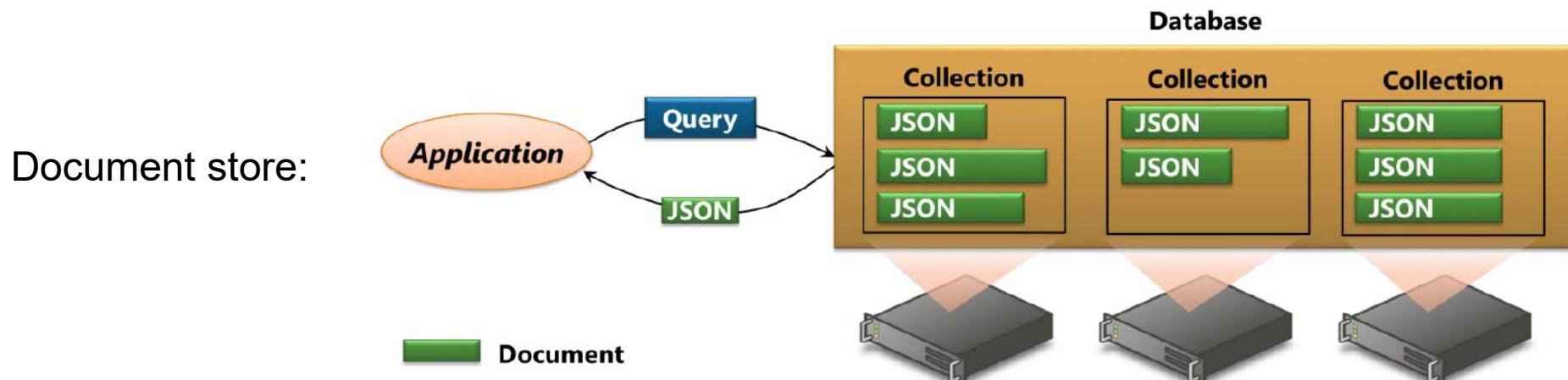


Source: [http://www.davidchappell.com/writing/white\\_papers/Windows\\_Azure\\_NoSQL\\_Technologies\\_v1.0--Chappell.pdf](http://www.davidchappell.com/writing/white_papers/Windows_Azure_NoSQL_Technologies_v1.0--Chappell.pdf)



# From Key-Value to Document Stores

- Key-value stores allow quick reads and updates, but we often want to query on other attributes as well
- If we have a JSON-type item, we can add indexes on attributes and allow querying on attributes as well

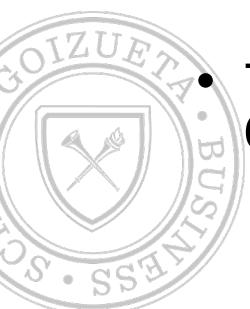


Source: [http://www.davidchappell.com/writing/white\\_papers/Windows\\_Azure\\_Nosql\\_Technologies\\_v1.0--Chappell.pdf](http://www.davidchappell.com/writing/white_papers/Windows_Azure_Nosql_Technologies_v1.0--Chappell.pdf)

# Document-Based NOSQL Systems

- Document-based or document-oriented NOSQL systems typically store data as collection of similar documents.
  - Documents can have different data elements (attributes), and
  - New documents can have new data elements that do not exist in any of the current documents in the collection.
- A popular language to specify documents in NOSQL systems is JSON (JavaScript Object Notation):

```
{Name:"John Doe",
Address:"Clifton Rd, Atlanta, GA"
Children: [Amber: "7", Barbara: "6", "Charlie: "3", "David: "1"]
}
```
- There are many document-based NOSQL systems, including MongoDB, CouchDB, and DocumentDB.



# Document Store: Advantages

- Documents are independent units - parallel
- Maps more easily to program objects (JSON).
- Dynamic Schema:
  - Semi-structured data can be stored easily.
  - More flexible in dealing with changes and optional values:
    - No system down time need to make schema changes.
  - Costly migrations are avoided since the database does not need to know its information schema in advance.
- Improves querying capabilities of key-value databases with indexing and the ability to filter/select documents based on attribute values in the document.



# MongoDB Overview

- Read Performance - MongoDB employs a custom binary protocol (and format) storing documents in BSON (binary JSON).
- Provides speed-oriented operations like upserts (i.e., update or insert) and update-in-place mechanics in the database.
- Examples of commands:

```
db.createCollection("project", {capped:true, size:1310720, max:500})
```

```
db.createCollection("worker", {capped:true, size:5242880, max:2000})
```

```
db.project.insert({ _id: "P1", Pname: "ProductX", Plocation: "Bellaire" })
```

```
db.<collection_name>.remove(<condition>)
```

```
db.<collection_name>.find(<condition>)
```

capped: set a maximum size in the size field.

max: The maximum number of documents allowed in the capped collection.



TRY AWS DocumentDB (w MongoDB Shell):

<https://docs.aws.amazon.com/documentdb/latest/developerguide/get-started-guide.html>



# AWS DocumentDB (using MongoDB)

- Insert:

- db.restaurants.insert\_one(restaurant\_json)

- Find:

- results = db.restaurants.find({"promotedReviews.reviewer": reviewer})
  - results = db.restaurants.find({"address.location": location}).sort([("updatedAt", pymongo.DESCENDING)])

- Update:

- db.restaurants.update\_one(
    - {"name": name},
    - {
      - "\$push": {"promotedReviews": review},
      - "\$set": {"updatedAt": datetime.datetime.now().isoformat(),},
    - },
    - )

- Compound Index:

- db.restaurants.create\_index([("address.location", pymongo.ASCENDING), ("updatedAt", pymongo.DESCENDING)])



# Document-Based NOSQL: Summary

- Most popular of the NoSQL databases because of flexibility, performance, and ease of use.
  - Does not require a fixed, predefined schema:
    - Flexible with regards to attributes used by each document.
    - Documents can have embedded documents and lists of multiple values within a document.
  - Common use cases:
    - Back-end support for websites with high volumes of reads and writes
    - Managing data types with variable attributes, such as products
    - Tracking variable types of metadata
  - Easier integration with programming languages.
- Also available from cloud services such as Microsoft Azure, AWS DocumentDB and Cloudant's database.



# AWS DocumentDB Lab

- Try/Complete:
  - <https://aws.amazon.com/getting-started/hands-on/purpose-built-databases/documentdb/>
  - Create Cloud9 Environment
  - Create DocumentDB Database (EC2 Cluster)
    - Create a small/medium instance
  - Design Data Model and Load Data
    - Install pymongo using: python -m pip install --user pymongo
  - Connection update:
    - `client = pymongo.MongoClient(  
f"mongodb://{USER}:{PASSWORD}@{HOST}:27017/?tls=true&tlsCAFile=rds-combined-ca-  
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred")`
  - Develop Applications with DocumentDB – insert & query
  - Cleanup Resources



# AWS DocumentDB (load\_sample\_data.py)

```
client = pymongo.MongoClient(
 f"mongodb://{USER}:{PASSWORD}@{HOST}:27017/?ssl=true&ssl_ca_certs=rds-combined-ca-bundle.pem&replicaSet=rs0
)

db = client.restaurants

with open("scripts/restaurant_1.json", "r") as f:
 restaurant = json.load(f)
 db.restaurants.insert_one(restaurant) # Insert one document in DB

with open("scripts/restaurant_2.json", "r") as f:
 restaurant = json.load(f)
 db.restaurants.insert_one(restaurant) # Insert one document in DB

print("Documents loaded successfully.")

db.restaurants.create_index([("name", pymongo.DESCENDING)])

print("Index created successfully.")
```

port  
Username, password, and endpoint  
Open documentDB  
Insert one document in DB  
Insert one document in DB  
Create index (note that we did not create a column "name" in DB)



# AWS DocumentDB (add\_compound\_index.py)

```
db = client.restaurants
db.restaurants.create_index(
 [("address.location", pymongo.ASCENDING), ("updatedAt", pymongo.DESCENDING)]
)
print("Index created successfully.")
```

Open documentDB

Create “location” index

Create “updatedAt” index

Note: compound (multi-field) index



# AWS DocumentDB (add\_review\_to\_restaurant.py)

```
db = client.restaurants

def add_review_to_restaurant(name, review):
 db.restaurants.update_one(
 {"name": name},
 {
 "$push": {"promotedReviews": review},
 "$set": {"updatedAt": datetime.datetime.now().isoformat()},
 },
)
 return

add_review_to_restaurant(
 "The Vineyard",
 {
 "reviewer": "elated_eric",
 "rating": 5,
 "review": "Sooo good! Can't wait to come back.",
 },
)
```

Update one document

Add a new value to document

Update a value in document



# AWS DocumentDB (get\_recently\_updated\_by\_location.py)

```
db = client.restaurants

def get_recently_updated_by_location(location):
 results = db.restaurants.find({"address.location": location}).sort(
 [("updatedAt", pymongo.DESCENDING)])
)
return results

results = get_recently_updated_by_location("New York, NY")
for restaurant in results:
 print(f"Restaurant: {restaurant['name']}. Updated at {restaurant['updatedAt']}")
```

Find document by location



# HBase



September 28, 2023



# Column-based or Wide Column NOSQL Systems

- Wide column stores, also called extensible record stores, store data in records with an ability to hold very large numbers of dynamic columns.
  - Wide column stores share the characteristic of being schema-free like document stores, however the implementation is very different.
  - Since the column names as well as the record keys are not fixed, and since a record can have billions of columns, wide column stores can be seen as two-dimensional key-value stores.
- Examples: Google BigTable, Apache HBase, Facebook Cassandra

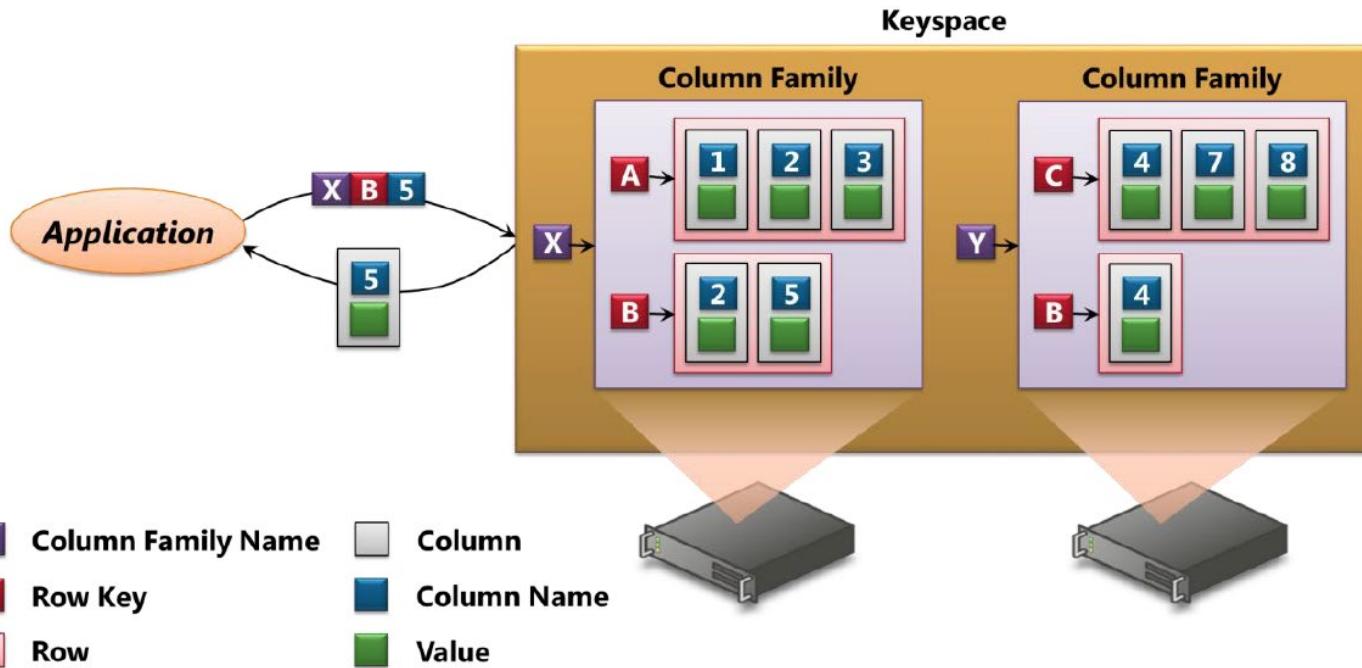


# Column-based NOSQL Systems

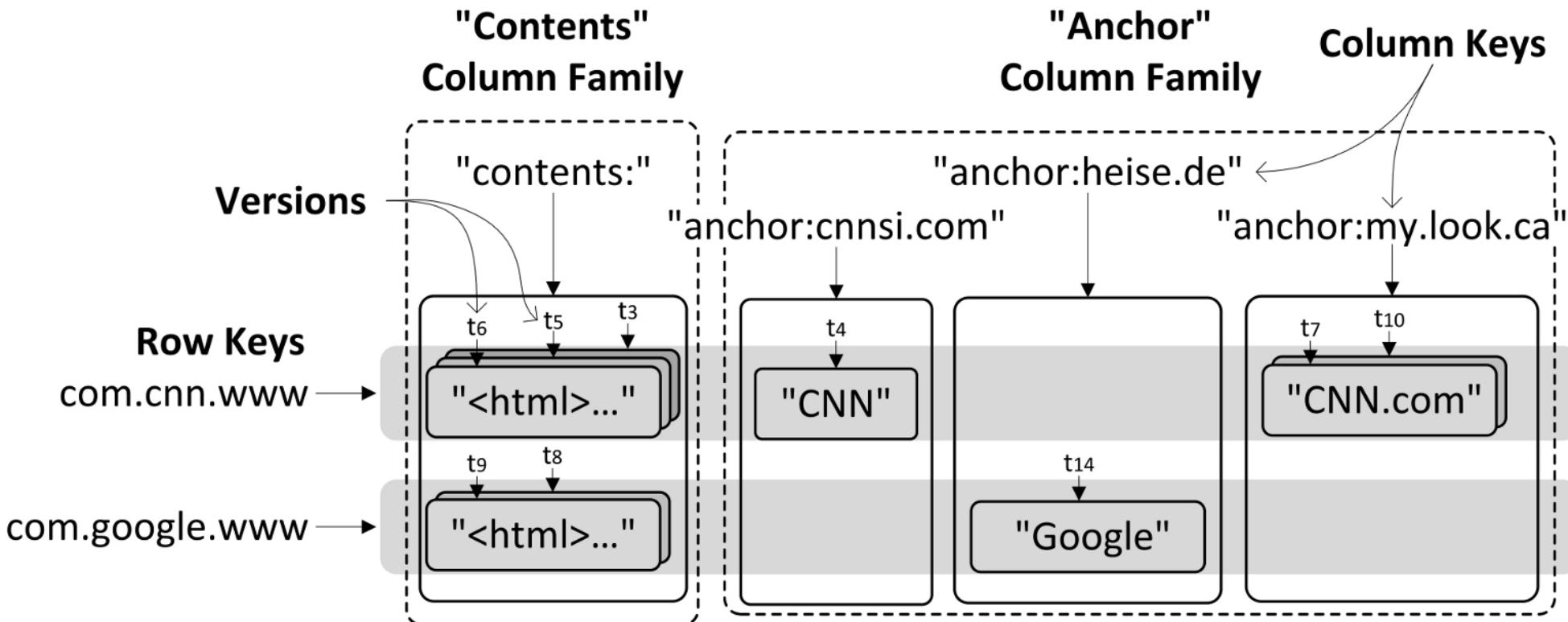
- Store data in columnar format
- Each storage block contains data from only one column
- Allow key-value pairs to be stored (and retrieved on key) in a massively parallel system
  - data model: families of attributes defined in a schema, new attributes can be added online
  - storing principle: big hashed distributed tables
- Properties:
  - partitioning (horizontally and/or vertically),
  - high availability,
  - completely transparent to application.



# Column-based NOSQL Systems



# Column-based NOSQL Systems



**Logical Model:**

Map<RowKey, Map<ColumnFamily, Map<ColumnQualifier, Map<Version, Data>>>



Additional reading: <https://research.google/pubs/pub27898/>



# Column-based NOSQL Systems

- More efficient than row (or document) store if retrievals access only some of the columns.
  - Common when multiple applications operate on top of same data.
  - Allow for better compression (since values in a column are similar).
- Inefficient if used in OLTP with many row inserts.



# Column-based NOSQL Systems: Examples

- BigTable - The Google distributed storage system for big data.
  - Used in many Google applications that require large amounts of data storage (e.g., Gmail). Provided as a service and integrated with big query.
- HBase - similar to BigTable
  - It typically uses HDFS (Hadoop Distributed File System) for data storage.
  - Can also use Amazon's Simple Storage System (aka S3) for data storage.
- Hypertable - also inspired by BigTable, used by Baidu
- Apache Cassandra - developed at Facebook for inbox search features, supports the Cassandra Query Language.



# Cassandra Overview



- Big-Table + DynamoDB:
  - Can be used as a distributed hash-table, with an "SQL-like" language, CQL (but no JOIN or SUM)
- Partition row stores into column families (called tables)
- Data can have expiration period (set on INSERT)
- Map/reduce possible with Apache Hadoop
- Rich data model (columns, composites, counters, secondary indexes, map, set, list)



# Cassandra Query Language (CQL)

- Example: create a customer/user table with addresses
  - RDBMS: recall sakila, each address has its own column
  - Wide-column: uses address family to add multiple addresses

```
CREATE TYPE address (
 street text,
 city text,
 zip int
);

CREATE TABLE user_profiles (
 login text PRIMARY KEY,
 first_name text,
 last_name text,
 email text,
 addresses map<text, address>
);

// Inserts a user with a home address
INSERT INTO user_profiles(login, first_name, last_name, email, addresses)
VALUES ('tsmith',
 'Tom',
 'Smith',
 'tsmith@gmail.com',
 { 'home': { street: '1021 West 4th St. #202',
 city: 'San Fransisco',
 zip: 94110 }});

// Adds a work address for our user
UPDATE user_profiles
 SET addresses = addresses
 + { 'work': { street: '3975 Freedom Circle Blvd',
 city: 'Santa Clara',
 zip: 95050 }}

WHERE login = 'tsmith';
```



# HBase Overview



- Apache HBase is the Hadoop database to use when you need real-time read/write access to your data.
- Automatic and configurable sharding of tables.
- Automatic failover support between RegionServers.
- Integrated with MapReduce.
- Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.
- Easy-to-use Java API for client access.



Try HBase on AWS: <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase.html>

# HBase Data Model Overview

- Table: HBase tables consist of multiple rows.
- Row: A row consists of a key and one or more columns.
  - Rows are sorted in alphabetical order using unique row keys.
- Column: Consists of the column family and column qualifier; e.g., “Info:email”
  - Each column family can be associated with many column qualifiers.
  - Column families group together related columns for storage purposes.
  - Column qualifiers are not specified as part of creating a table making the model a self-describing data model.
- Cell: Combination of row, column family, column qualifier, timestamp, and value. Holds a basic data item.
  - Multiple versions of a data item can be stored, with a timestamp associated with each version.



# How HBase stores data

- Logical representation of how values are stored in HBase:

| Row Key | Time stamp | Name Family |                 | Address Family |                    |
|---------|------------|-------------|-----------------|----------------|--------------------|
|         |            | first_name  | last_name       | number         | address            |
| row1    | t1         | <u>Bob</u>  | <u>Smith</u>    |                |                    |
|         | t5         |             |                 | 10             | First Lane         |
|         | t10        |             |                 | 30             | Other Lane         |
|         | t15        |             |                 | 7              | <u>Last Street</u> |
| row2    | t20        | <u>Mary</u> | Tompson         |                |                    |
|         | t22        |             |                 | 77             | One Street         |
|         | t30        |             | <u>Thompson</u> |                |                    |



# HBase Operations Overview

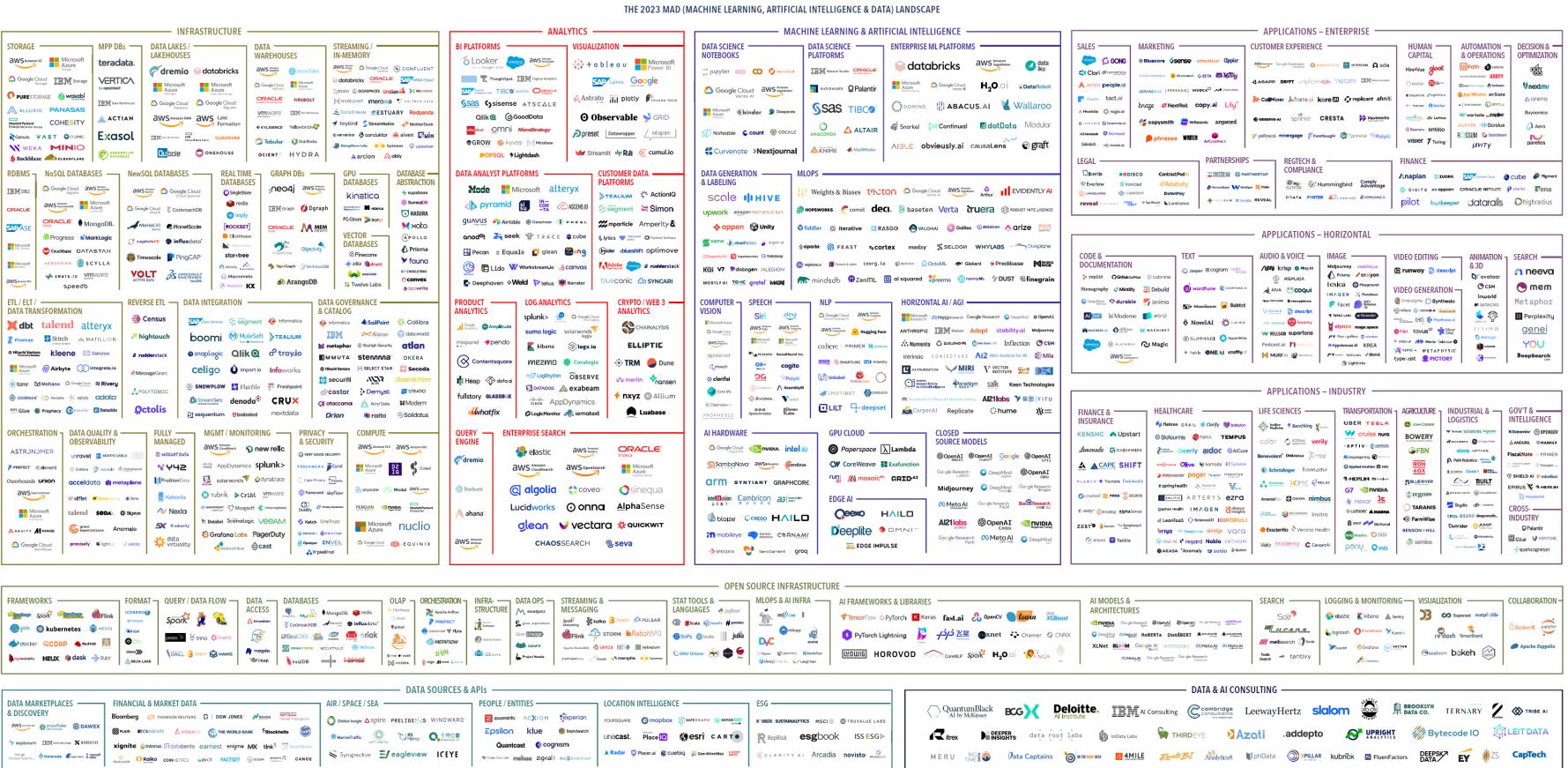
- Creating a table: create <tablename>, <column family>, <column family>, ...
  - create ‘EMPLOYEE’, ‘Name’, ‘Address’, ‘Details’
- Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
  - put ‘EMPLOYEE’, ‘row1’, ‘Name:Fname’, ‘John’
  - put ‘EMPLOYEE’, ‘row3’, ‘Details:Job’, ‘CEO’
- Reading Data (all data in a table): scan <tablename>
- Retrieve Data (one item): get <tablename>,<rowid>



# Course Content (technologies)



# Big Data Technologies



GMAT®是美国研究生管理学院协会（Graduate Management Admission Council, Inc.）的注册商标。

Digitized by srujanika@gmail.com

Volume 10 Number 1 March 2000

3.1.3.2.5. *Quaternions* 1 / 1

FIRSTMARK  
EARLY STAGE VENTURE CAPITAL

Source: <https://mad.firstmark.com/>

October 2, 2023

APACHE

**HBASE**



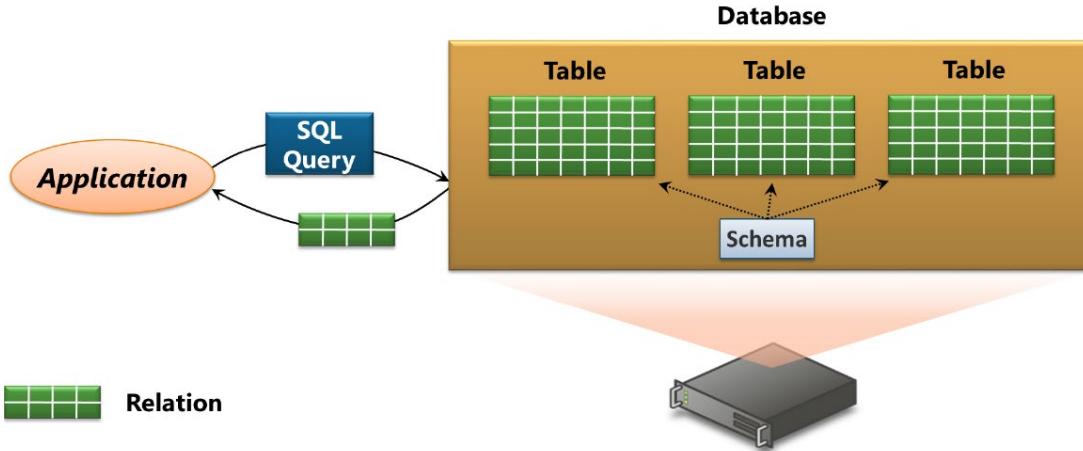
NoSQL III

HBase

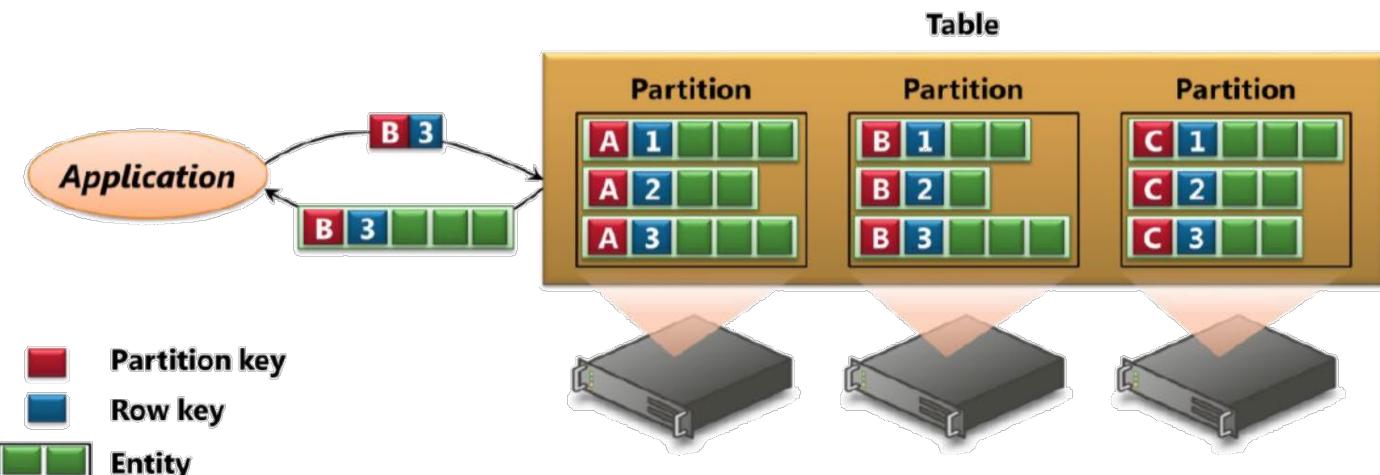
October 2, 2023

# Key-value data stores (Azure view)

- Relational:



- Key-value:

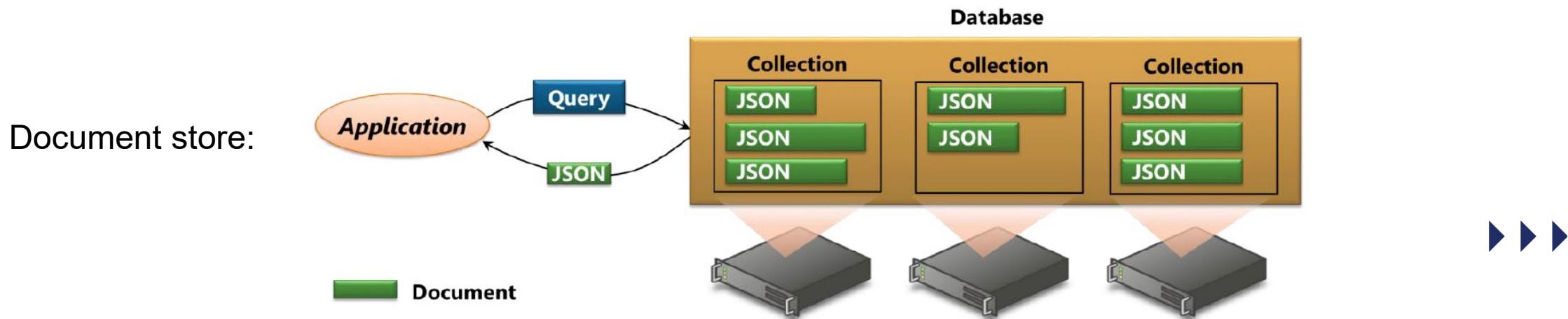


Source: [http://www.davidchappell.com/writing/white\\_papers/Windows\\_Azure\\_NoSQL\\_Technologies\\_v1.0--Chappell.pdf](http://www.davidchappell.com/writing/white_papers/Windows_Azure_NoSQL_Technologies_v1.0--Chappell.pdf)

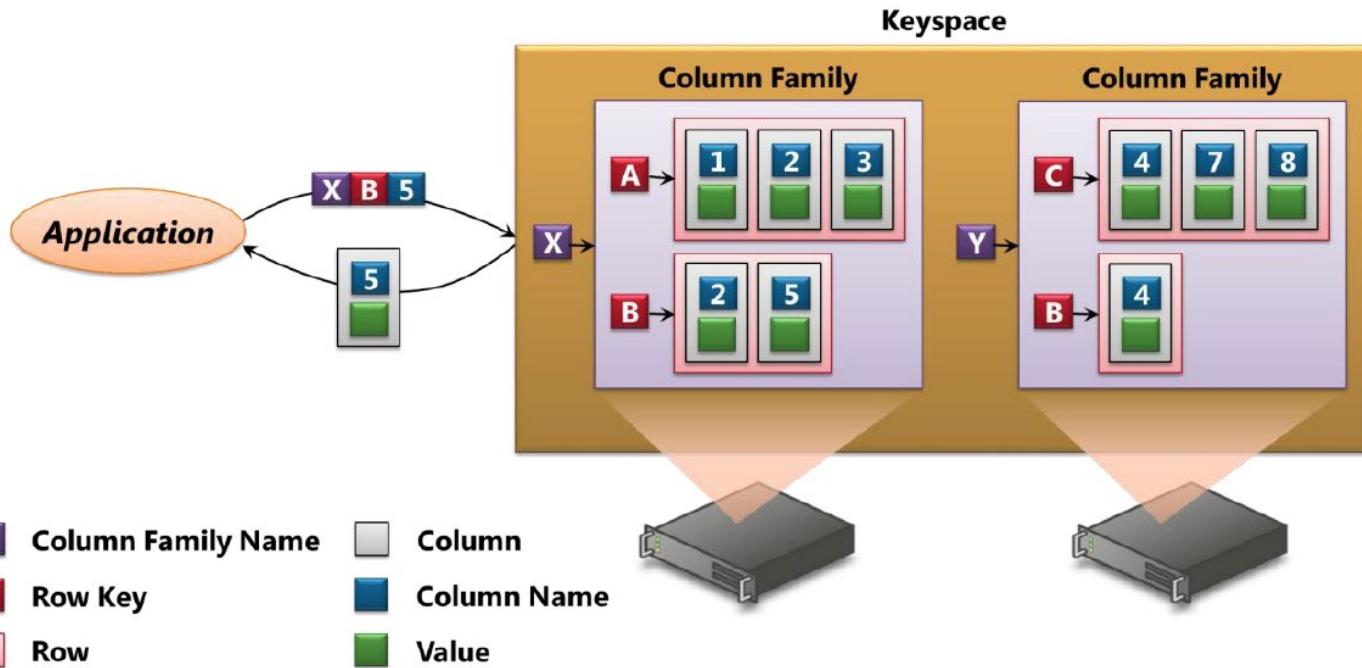


# From Key-Value to Document Stores

- Key-value stores allow quick reads and updates, but we often want to query on other attributes as well
- If we have a JSON-type item, we can add indexes on attributes and allow querying on attributes as well



# Column-based NOSQL Systems



# Apache HBase

- HBase is a NoSQL column-oriented distributed database.
- HBase is the Hadoop application to use when you require real-time read/write random access to very large datasets.
- Stores data in HDFS.
- HBase can scale (almost) linearly just by adding nodes.
  - It scales to support very high throughput for both reads and writes (e.g., millions of inserts or updates per second).
- A table can have many thousands of columns.
  - Handles sparse data well.
- Designed to store very large amounts of data
  - Petabytes+



# Apache HBase Characteristics

- Commodity hardware
  - Clusters are built on \$1,000-\$5,000 nodes rather than \$50,000 nodes. RDBMSs are I/O hungry, requiring more costly hardware.
- Fault tolerance
  - Lots of nodes means each is relatively insignificant. No need to worry about individual node downtime.
- Batch processing
  - MapReduce integration allows fully parallel, distributed jobs against your data with locality awareness.



# HBase data model

- Many concepts are similar, but use different terminology in HBase
  - The intersection of a row and column is called a cell
  - The primary key is called a row key and is required
- HBase stores data differently than a relational database
  - Column families group columns for separate storage and access
  - Columns within a family do not have to be declared in advance

| Row Key | contactinfo |       |         |          |          | annualbudget |          |        |
|---------|-------------|-------|---------|----------|----------|--------------|----------|--------|
|         | fname       | mname | lname   | mobile   | home     | movies       | concerts | travel |
| aj8462  | Alice       | Jane  | Ames    | 555-2182 | 555-3714 | 300          | 100      | 5000   |
| bb2988  | Bob         |       |         |          |          | 500          |          |        |
| cs8701  | Carol       | Sue   | Clemens | 555-2642 |          |              | 500      |        |
| dd7235  | Dan         |       | Davis   |          | 555-3421 |              | 375      | 1000   |
| ee5361  | Eve         | A.    | Ellis   | 555-3960 |          | 100          |          | 2500   |



# HBase region

- Region: basic building elements of HBase cluster. A unit of distribution and availability
  - Regions are split when grown too large
  - Max region size is a tuning parameter (default size is 256 MB)
    - Too low: prevents parallel scalability
    - Too high: makes things slow

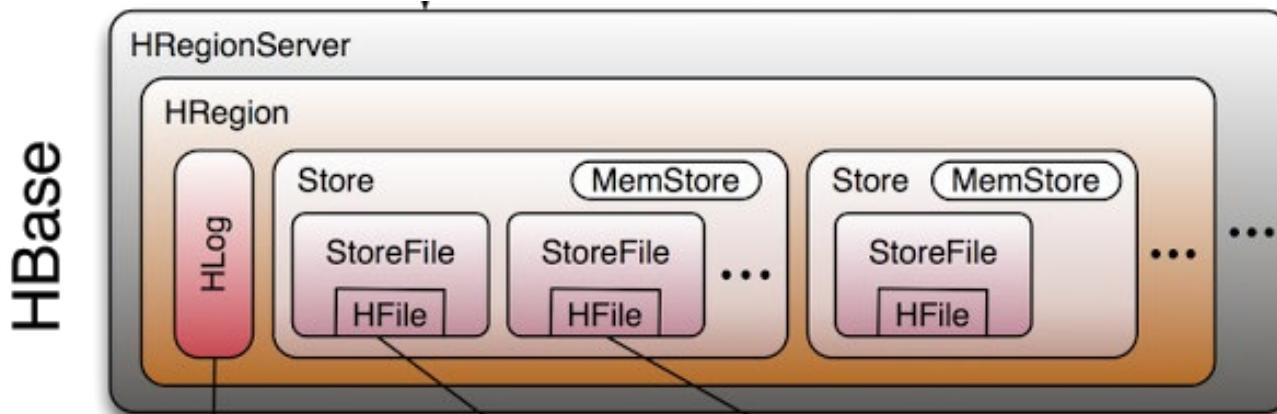


Image source: <http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html>

# HBase architecture

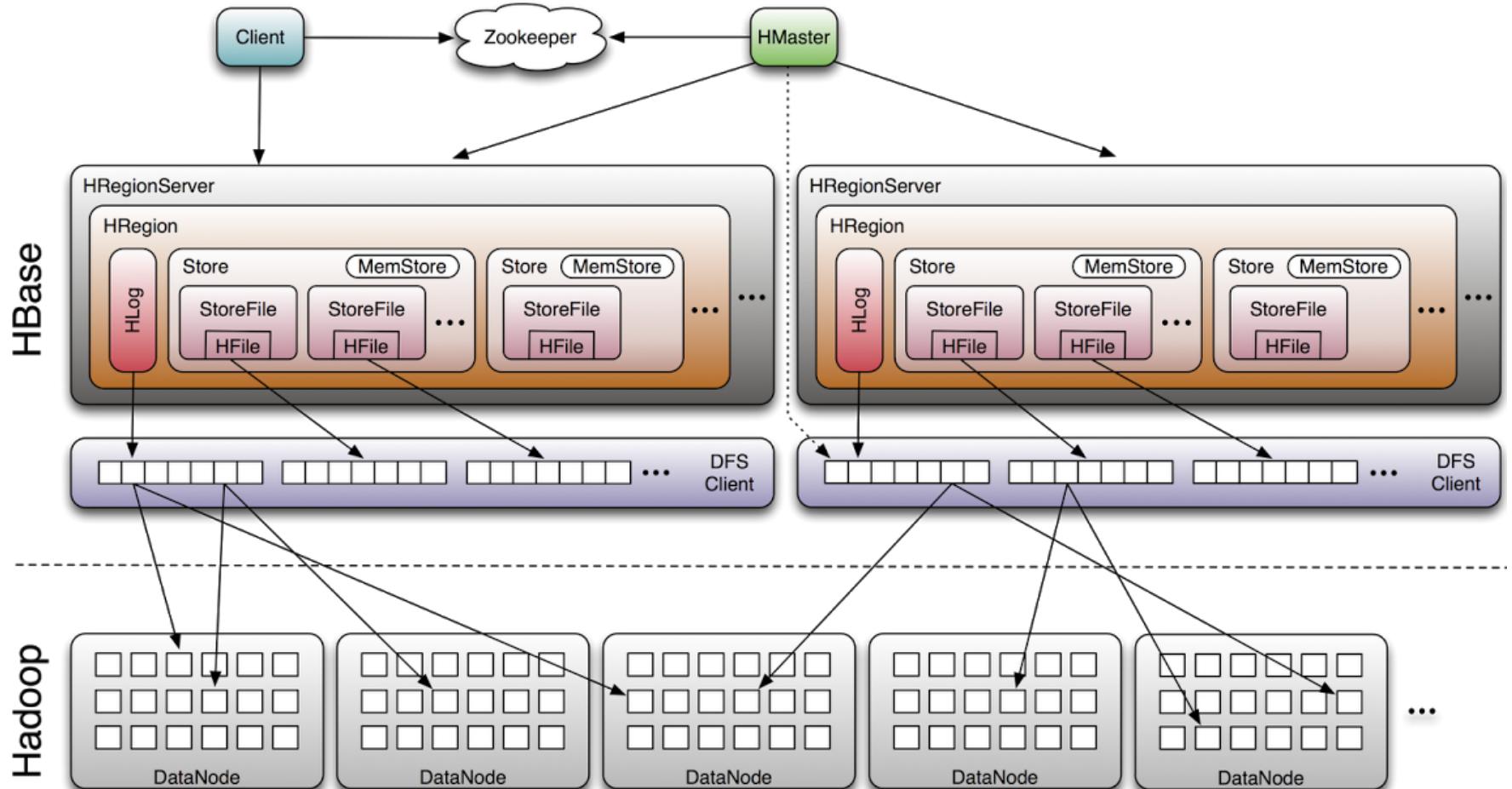


Image source: <http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html>

# HBase write-ahead log (WAL)

- Hregionserver sends the data to Hregion where it is stored in log and in volatile memory (before being written on disk)
- WAL (Write-ahead Log) allows for recovery of data if a Region crashes and is unable to save the data to disk.

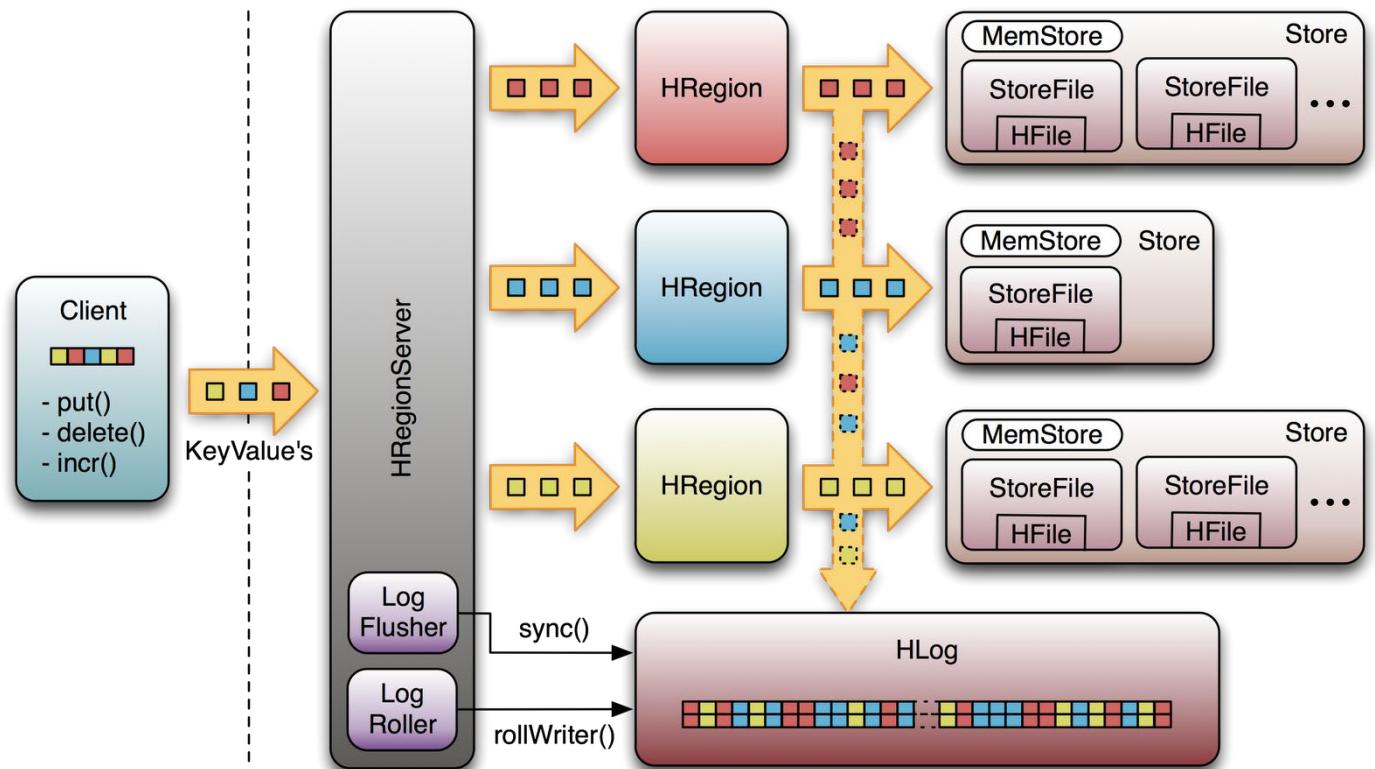


Image source: <http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html>

# HBase Data Model



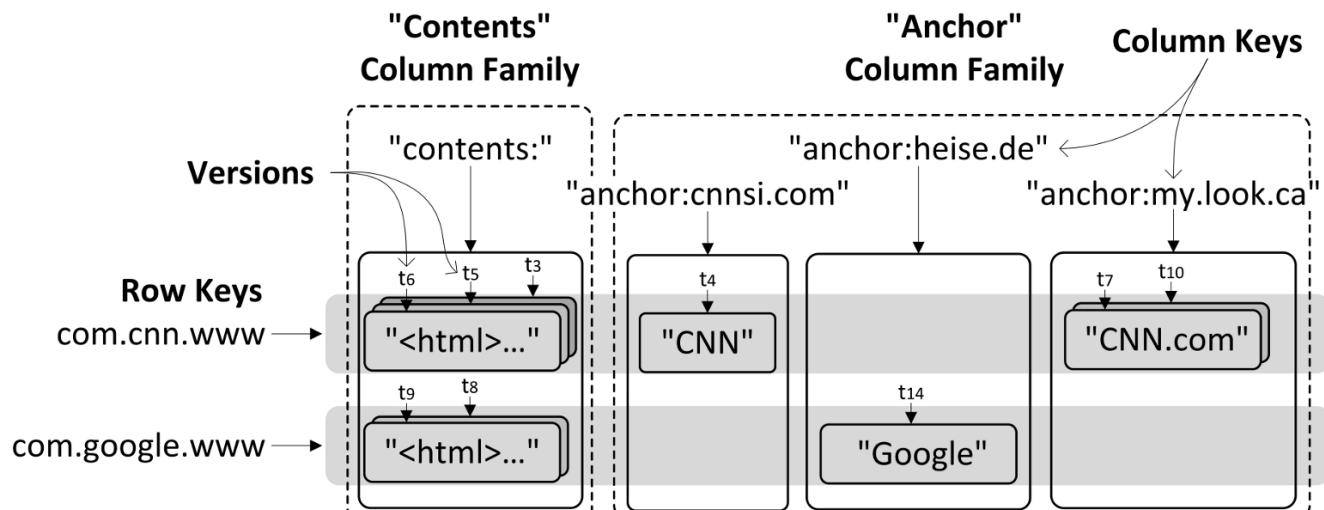
# HBase Data Model

- Table: HBase tables consist of multiple rows.
- Row: A row consists of a key and one or more columns.
  - Rows are sorted in alphabetical order using unique row keys.
- Column: Consists of the column family and column qualifier; e.g., “Info:email”
  - Each column family can be associated with many column qualifiers.
  - Column families group together related columns for storage purposes.
  - Column qualifiers are not specified when creating a table, making the model a self-describing data model.
- Cell: Combination of row, column family, column qualifier, timestamp, and value. Holds a basic data item.
  - Multiple versions of a data item can be stored, with a timestamp associated with each version.



# HBase Data Model

- A sparse, multi-dimensional, sorted map
  - {row, column, timestamp} -> cell
- Column = Column Family: Column Qualifier
- Rows are sorted lexicographically based on row key
- Region: contiguous set of sorted rows



Additional reading: <https://research.google/pubs/pub27898/>

# HBase data example

- Logical representation of how values are stored in HBase:

| Row Key | Time stamp | Name Family |                 | Address Family |                    |
|---------|------------|-------------|-----------------|----------------|--------------------|
|         |            | first_name  | last_name       | number         | address            |
| row1    | t1         | <u>Bob</u>  | <u>Smith</u>    |                |                    |
|         | t5         |             |                 | 10             | First Lane         |
|         | t10        |             |                 | 30             | Other Lane         |
|         | t15        |             |                 | 7              | <u>Last Street</u> |
| row2    | t20        | <u>Mary</u> | Tompson         |                |                    |
|         | t22        |             |                 | 77             | One Street         |
|         | t30        |             | <u>Thompson</u> |                |                    |



# Nested Data Representation

```
Table People (Name, Home, Office)
{
 101: {
 Timestamp: T403;
 Name: {First="Bob", Middle="D", Last="Smith"},
 Home: {Phone="555-1212", Email="bob.smith@fakemail.com"},
 Office: {Phone="666-1212", Email="bob.smith@fakemail.org"}
 },
 102: {
 Timestamp: T593;
 Name: { First="Mary", Last="Tompson" },
 Home: { Phone="555-1213" },
 Office: { Phone="666-1213" }
 },
 ...
}
```



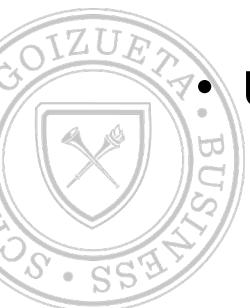
# HBase operators

- Operations are based on row keys
  - Single-row operations:
    - Put
    - Get
    - Scan
  - Multi-row operations:
    - Scan
    - MultiPut
- No built-in joins (uses MapReduce)



# Try HBase on AWS EMR

- Create EMR cluster:
  - <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase-create.html>
- Use PuTTY for SSH connection:
  - <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-ssh.html>
- Using shell:
  - <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase-connect.html>
  - <https://www.guru99.com/hbase-shell-general-commands.html>
- Using UI:
  - <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/hbase-web-ui.html>



# HBase Basic Operations

## » hbase shell

```
hbase(main):018:0>
```

- Launch hbase shell

## » status

```
1 active master, 0 backup masters, 1 servers, 0 dead, 3.0000 average load
```

- Get system status - number of servers in the cluster, active server count, and average load value

## » table\_help

```
Help for table-reference commands.
```

- Get information on various table shell commands available



# HBase Table Operations (create/put)

- » Creating a table: create <tablename>, <column family>, <column family>, ...
    - » create 'employee', 'name', 'address', 'other'
  - » Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
    - » put 'employee', 'row1', 'name:fname', 'Bob'
    - » put 'employee', 'row1', 'other:title', 'CIO'
    - » put 'employee', 'row3', 'name:fname', 'John'
    - » put 'employee', 'row3', 'other:title', 'CEO'
    - » put 'employee', 'row1', 'address:city', 'Atlanta'
    - » put 'employee', 'row3', 'address:city', 'Austin'
  - » Describe the column families in the table: describe <tablename>
    - » describe 'employee'
- Count number of row of data in a table: count '<tablename>'
  - count 'employee'



# HBase Table Operations (get/scan)

- Retrieve Data (one item): get <tablename>,<rowid>
  - » get 'employee', 'row1'
  - » get 'employee', 'row3'
  - » get 'employee', 'row1', {COLUMN => ['name', 'address']}
  - » get 'employee', 'row3', {COLUMN => ['other']}
  - » get 'employee', 'row1', {TIMERANGE => [166476700000, 166476710000]}
- Convert time: <https://www.epochconverter.com/>
- Reading Data (all data in a table): scan <tablename>
  - » scan 'employee'
  - » scan 'employee', {COLUMNS => 'other:title'}
  - » scan 'employee', {COLUMNS => ['address'], LIMIT => 5, STARTROW => 'row3'}
  - » scan 'employee', {TIMERANGE => [166476700000, 166476710000]}
  - » scan 'employee', {STARTROW => 'row3', ENDROW => 'row4'}



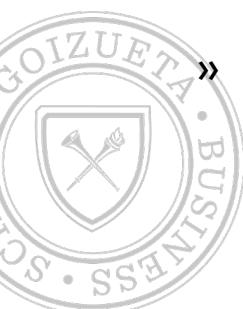
Try: <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase-create.html>

# HBase Table Operations (filters)

- When reading data from HBase using Get or Scan operations, you can use custom filters to return a subset of results to the client. Filters are generally used using the Java API but can be used from HBase Shell for testing and debugging purposes.
  - [https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/admin\\_hbase\\_filtering.html](https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/admin_hbase_filtering.html)
- Reading Data (all data in a table): scan <tablename> WITH FILTERS
  - » show\_filters
  - » Value Filters:
    - » scan 'employee', FILTER => "ValueFilter(=, 'binary:CEO')"
    - » scan 'employee', {FILTER=>"SingleColumnValueFilter ('name','fname',=,'regexstring:.\*Bo\*')"}
  - » Column Name Filters:
    - » scan 'employee', {FILTER => "FamilyFilter(=, 'regexstring:name')"}
    - » scan 'employee', {FILTER => "QualifierFilter(=, 'regexstring:title')"}

```
hbase(main):030:0> show_filters
DependentColumnFilter
KeyOnlyFilter
ColumnCountGetFilter
SingleColumnValueFilter
PrefixFilter
SingleColumnValueExcludeFilter
FirstKeyOnlyFilter
ColumnRangeFilter
TimestampsFilter
FamilyFilter
QualifierFilter
ColumnPrefixFilter
RowFilter
MultipleColumnPrefixFilter
InclusiveStopFilter
PageFilter
ValueFilter
ColumnPaginationFilter
```

[https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/admin\\_hbase\\_filtering.html](https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/admin_hbase_filtering.html)



# HBase Table Operations (delete)

- Deleting a column: delete <'tablename'>, <'row name'>, <'column name'>
  - delete 'employee', 'row1', 'other:title'
- Deleting a column family: deleteall <'tablename'>, <'rowname'>
  - deleteall 'employee', 'row1'
- Dropping an entire table: drop <'tablename'>
  - disable <'tablename'> (before dropping)
  - disable 'employee'
  - drop 'employee'



# HBase Table Operations (hue)

The screenshot shows the Hue HBase Browser interface. The URL in the browser bar is `ec2-54-165-110-73.compute-1.amazonaws.com:8888/hue/hbase/#Bigtop/employee`. The left sidebar lists databases: MySQL and Bigtop (0). A message says "Error loading databases.". The main area is titled "HBase Browser" and "Home - Bigtop / employee". It displays two rows of data:

| other: title | name: fname | address: city |
|--------------|-------------|---------------|
| CTO          | Bob2        | Atlanta       |

| other: title | name: fname | address: city |
|--------------|-------------|---------------|
| CEO          | John        | Austin        |

Below the tables, a message says "Fetched 10 entries starting from null in 0.112 seconds." At the bottom right are buttons for "Drop Rows", "Bulk Upload", and "New Row". Navigation arrows are at the bottom right.

Setup Hue: <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-web-interfaces.html>

# ACID Model

- Apache HBase is not an ACID compliant database. However, it does guarantee certain specific properties.
- **Atomicity:** an operation is atomic if it either completes entirely or not at all.
  - Atomic within a row - any put will either completely succeed or completely fail
- **Consistency:**
  - A GET is guaranteed to return a complete row that existed at some point in the table's history. (check timestamp!)
  - A SCAN must include all data written prior to the scan
    - May include updates since it started.
- **Isolation:** Not guaranteed outside a single row.
  - A scan is not a consistent view of a table. Scans do not exhibit snapshot isolation.

**Durability:** All successful writes have been made durable on disk.



Possible project idea:  
Develop a NoSQL database for a social media app

# HBase @Facebook

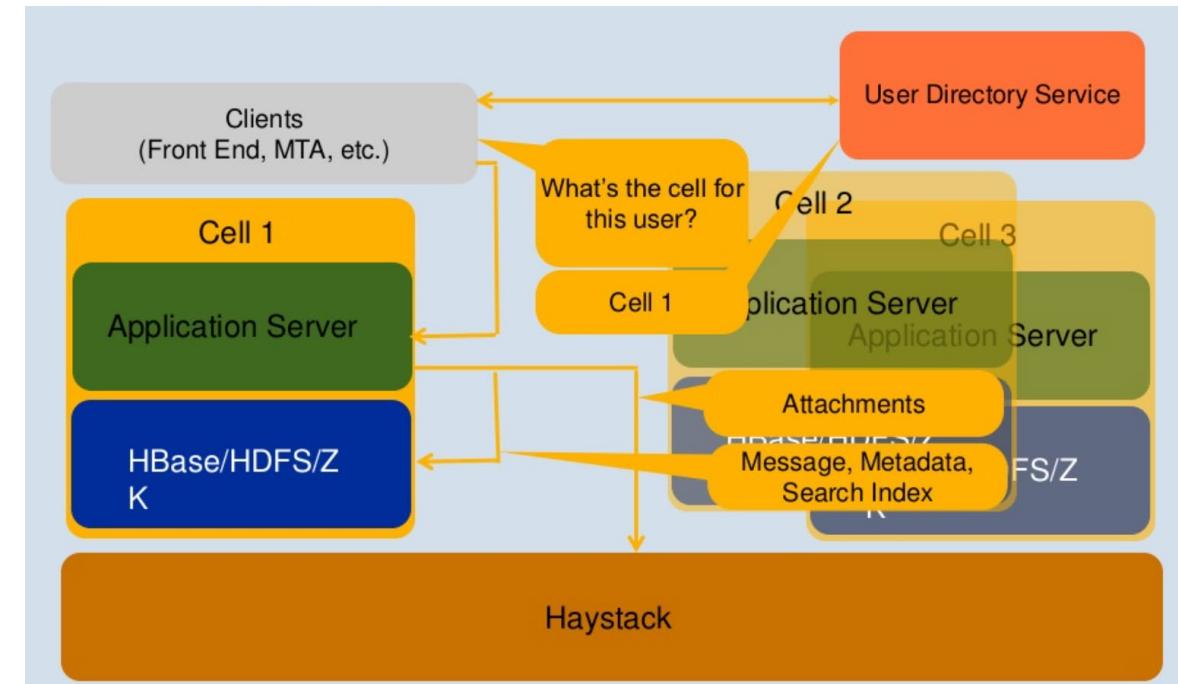
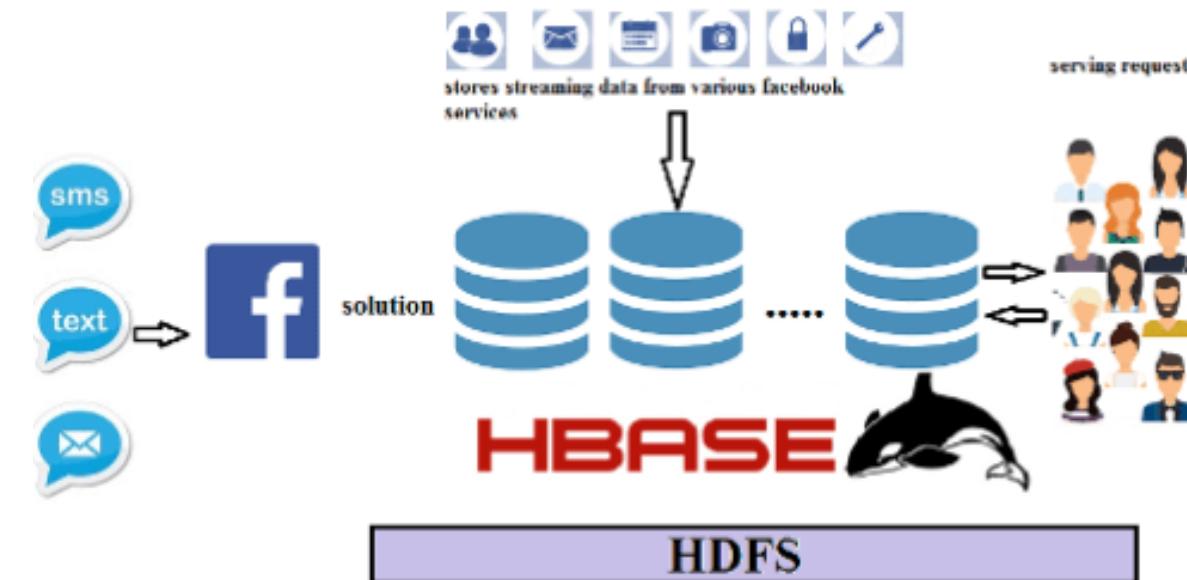


Image sources: <https://www.slideshare.net/brizzdotcom/facebook-messages-hbase>  
<https://beyondcorner.com/learn-apache-hbase/facebook-messenger-case-study-with-apache-hbase/>

# Strengths & Limitations of HBase

- Strengths:
  - Serving large amounts of data
  - Large number of client requests
  - Variable schema - rows may differ drastically
- Limitations:
  - Not a replacement for traditional RDBMS for transactions
    - Not good for transactional database
    - Not good for relational analytics for operations like ‘group by’, ‘join’, ‘where’, etc.



# HBase vs RDBMS

## HBase:

- Flexible schema, column-oriented
- Good with sparse & wide table
- Joins using MR (not optimized)
  - Tight integration with MR
- Data not normalized
- Scalable
- Not suitable for transactions
  - usually used for OLAP

## RDBMS

- Fixed schema
- Not good with sparse tables
- Optimized for table joins
- Data normalized
- Hard to scale
- Transactional (OLTP)



# Apache HBase: When Should I Use It?

- Use HBase if...
  - You need random reads
  - You need random writes
  - You need to do thousands of operations per second on terabytes of data
  - Your access patterns are simple and well-known
- Don't use HBase if...
  - You only append to your dataset and typically read the entire table
  - You primarily perform ad-hoc analytics (ill-defined access patterns)
  - Your data easily fits on one large node
    - Use an RDBMS in this case!



APACHE

**HBASE**



NoSQL III

CATCHUP

October 12, 2023

Possible project idea:  
Develop a NoSQL database for a social media app

# HBase @Facebook

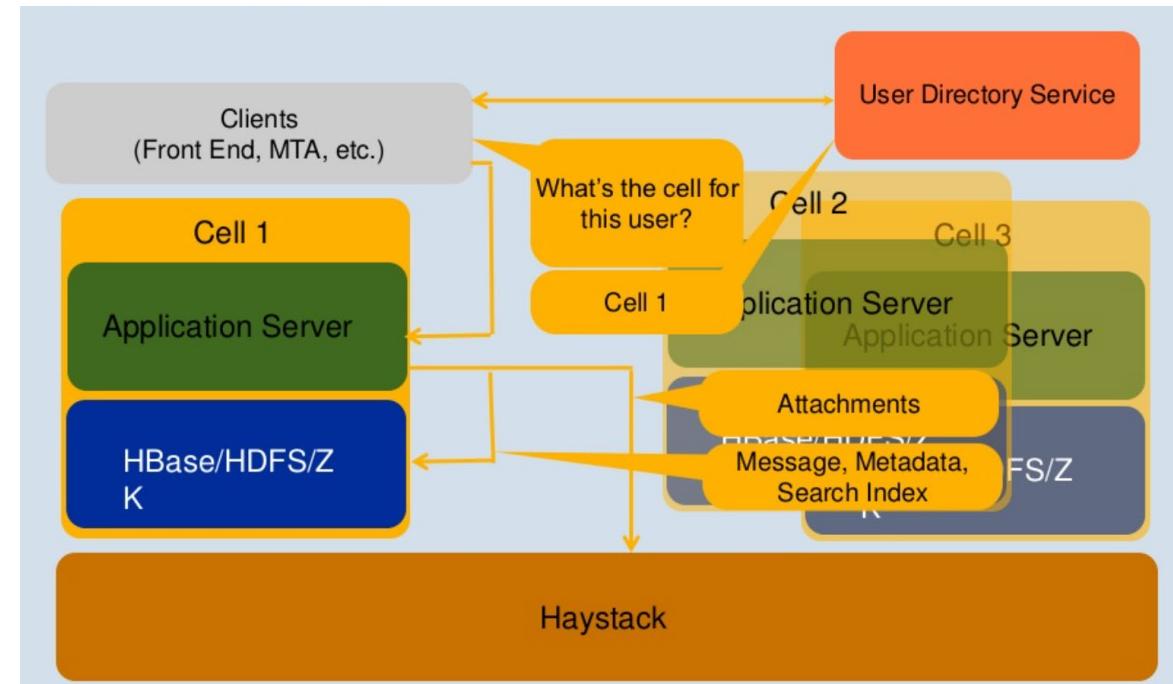
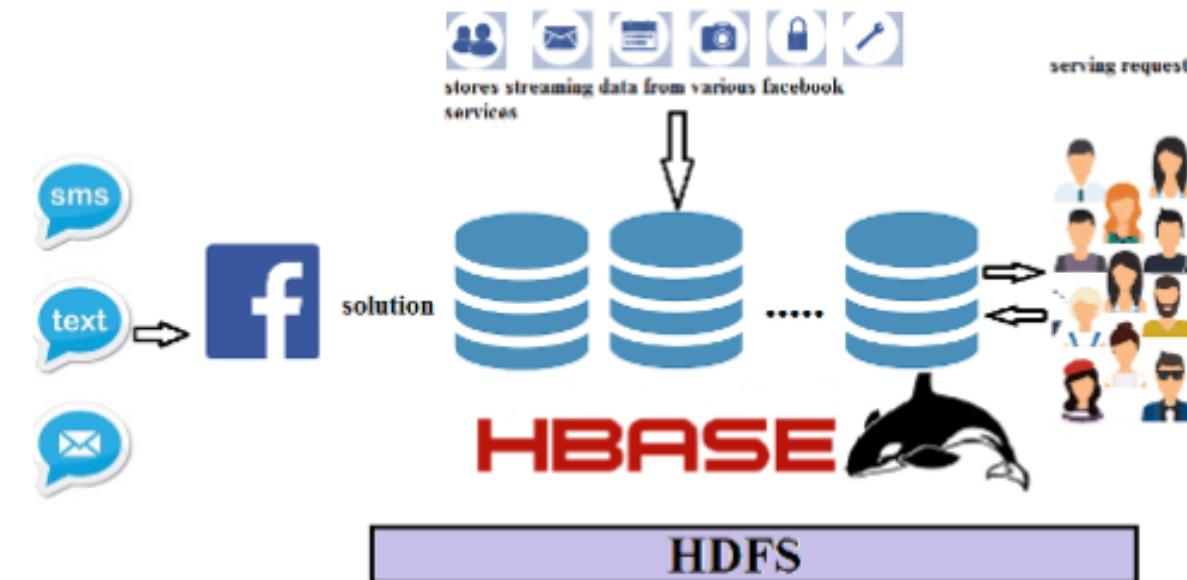


Image sources: <https://www.slideshare.net/brizzdotcom/facebook-messages-hbase>  
<https://beyondcorner.com/learn-apache-hbase/facebook-messenger-case-study-with-apache-hbase/>

# Strengths & Limitations of HBase

- Strengths:
  - Serving large amounts of data
  - Large number of client requests
  - Variable schema - rows may differ drastically
- Limitations:
  - Not a replacement for traditional RDBMS for transactions
    - Not good for transactional database
    - Not good for relational analytics for operations like ‘group by’, ‘join’, ‘where’, etc.



# HBase vs RDBMS

## HBase:

- Flexible schema, column-oriented
- Good with sparse & wide table
- Joins using MR (not optimized)
  - Tight integration with MR
- Data not normalized
- Scalable
- Not suitable for transactions
  - usually used for OLAP

## RDBMS

- Fixed schema
- Not good with sparse tables
- Optimized for table joins
- Data normalized
- Hard to scale
- Transactional (OLTP)



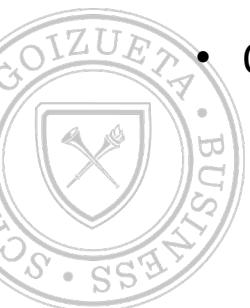
# Apache HBase: When Should I Use It?

- Use HBase if...
  - You need random reads
  - You need random writes
  - You need to do thousands of operations per second on terabytes of data
  - Your access patterns are simple and well-known
- Don't use HBase if...
  - You only append to your dataset and typically read the entire table
  - You primarily perform ad-hoc analytics (ill-defined access patterns)
  - Your data easily fits on one large node
    - Use an RDBMS in this case!



# Additional Resources (HBase)

- HBase reference: <http://hbase.apache.org/book.html>
- HBase, querying data: <https://drill.apache.org/docs/querying-hbase/>
- HBase video tutorials:
  - <https://www.youtube.com/watch?v=Aky7R0P73D8> (by Think Big)
  - [https://www.youtube.com/watch?v=HLoH\\_PgrLk](https://www.youtube.com/watch?v=HLoH_PgrLk) (by O'Reilly)
- Sample shell commands: <https://www.guru99.com/hbase-shell-general-commands.html>
  - Linux tutorial (useful for Putty SSH): <https://ubuntu.com/tutorials/command-line-for-beginners>
- Other (we will do this post-midterm):
  - Hands-on workshop on Hive & Pig: <https://www.oracle.com/technetwork/topics/bigdata/articles/hive-and-pig-hol-1937050.pdf>





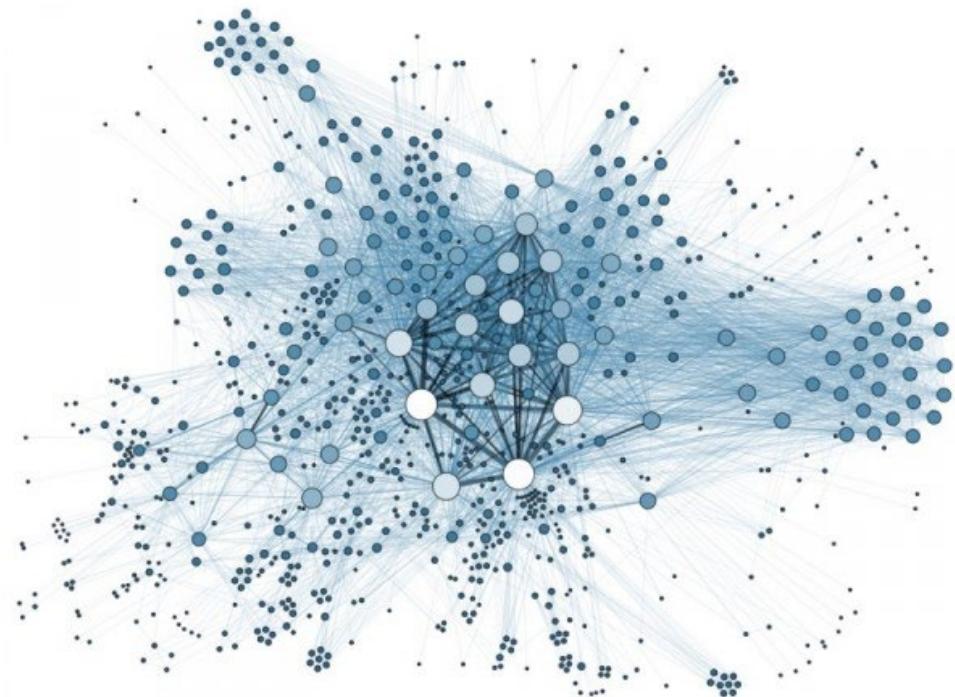
# neo4j

# NoSQL IV

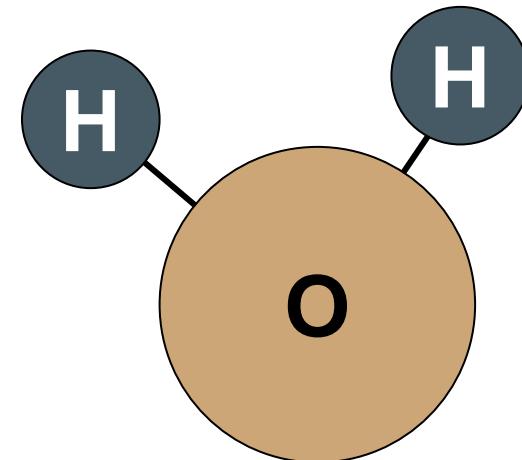
Neo4j

# What is a graph?

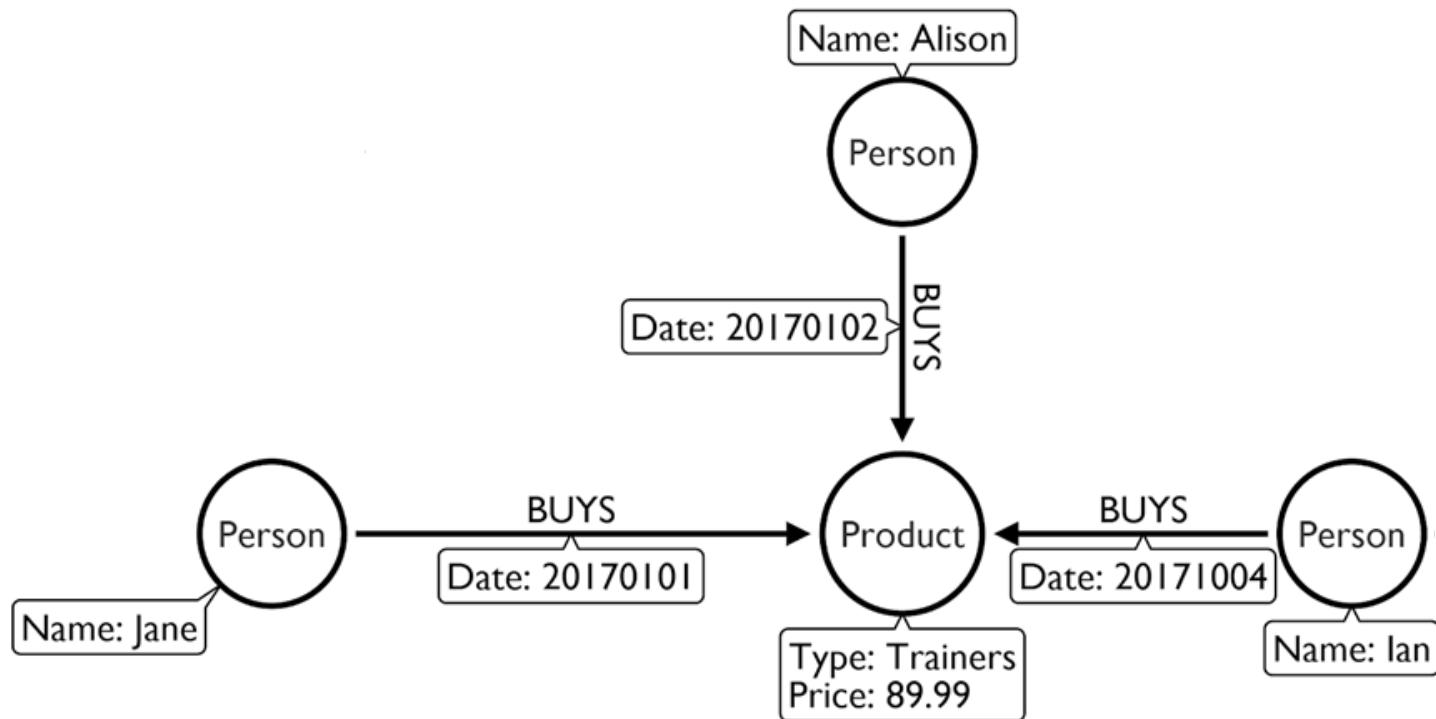
the Internet



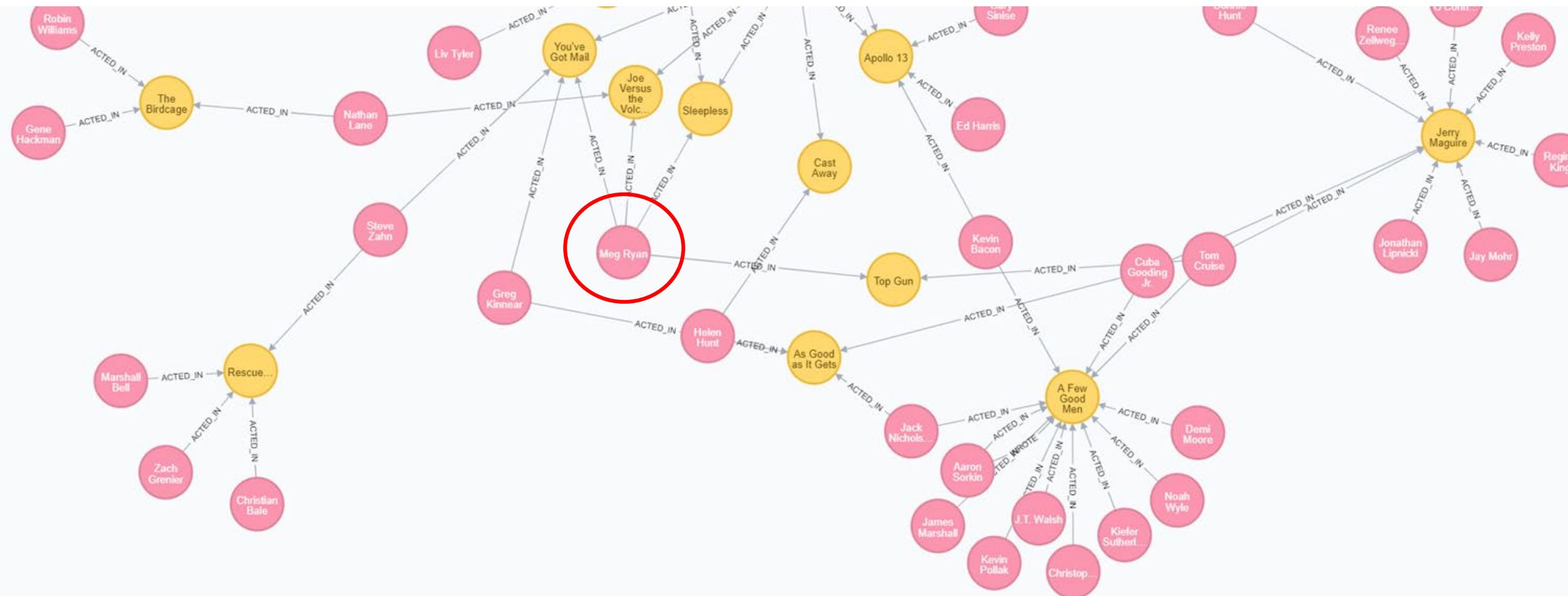
a water molecule



# Graph of purchases:

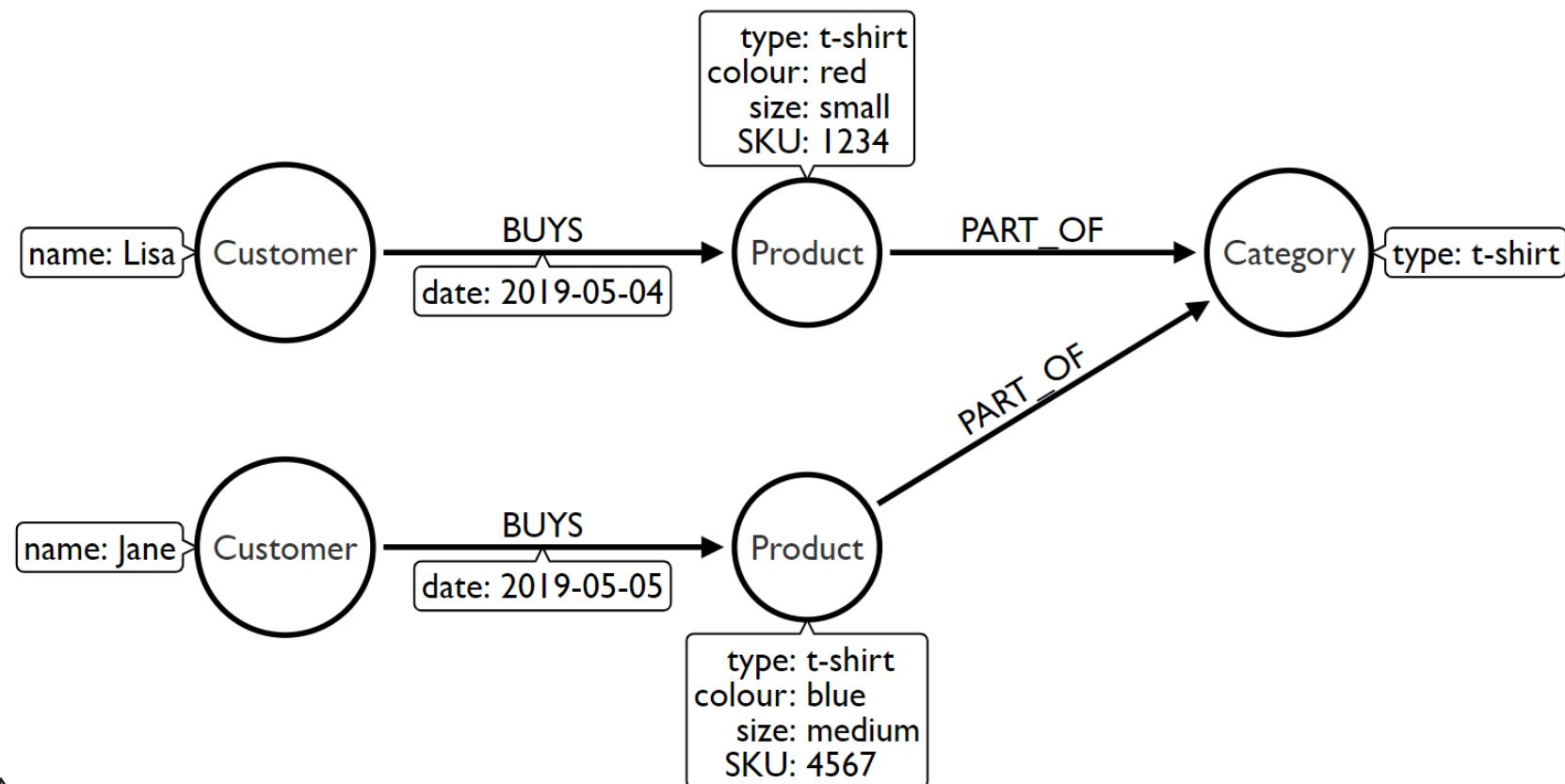


# Graph of movies:



# Graph scenario (1)

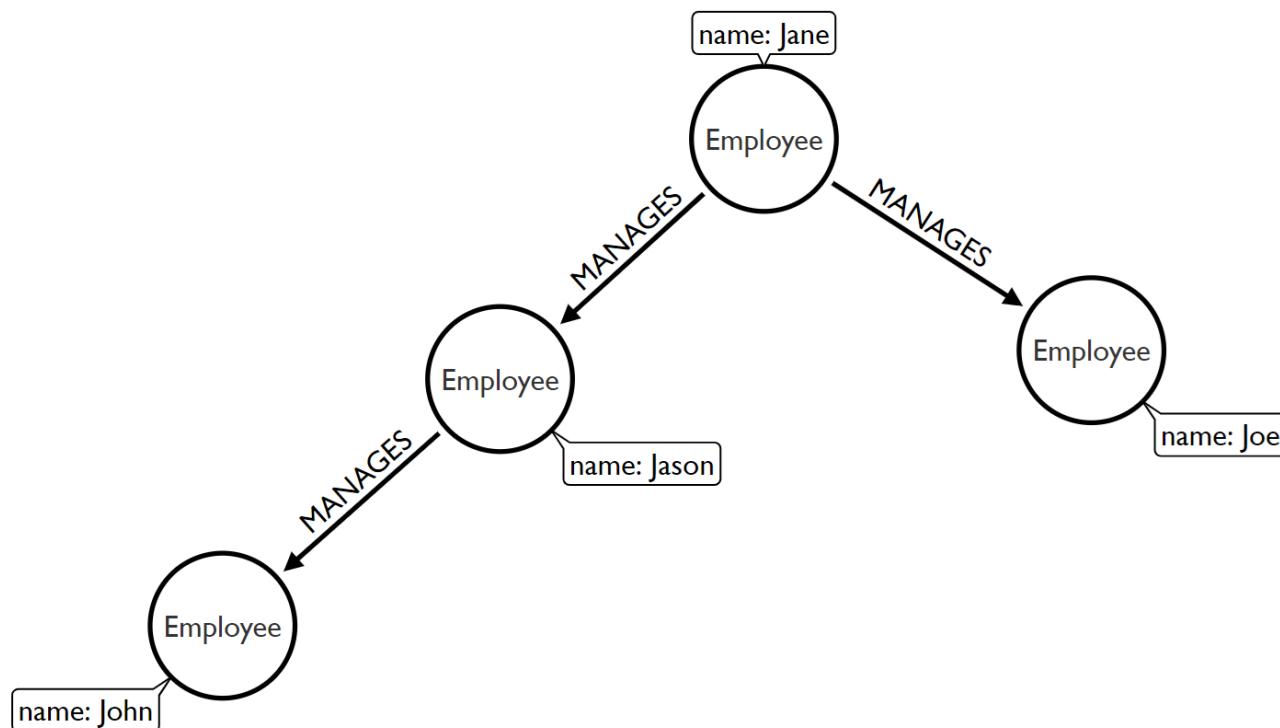
Understanding relationships between entities



- Recommendations
- Fraud detection
- Finding duplicates
- Data lineage

# Graph scenario (2)

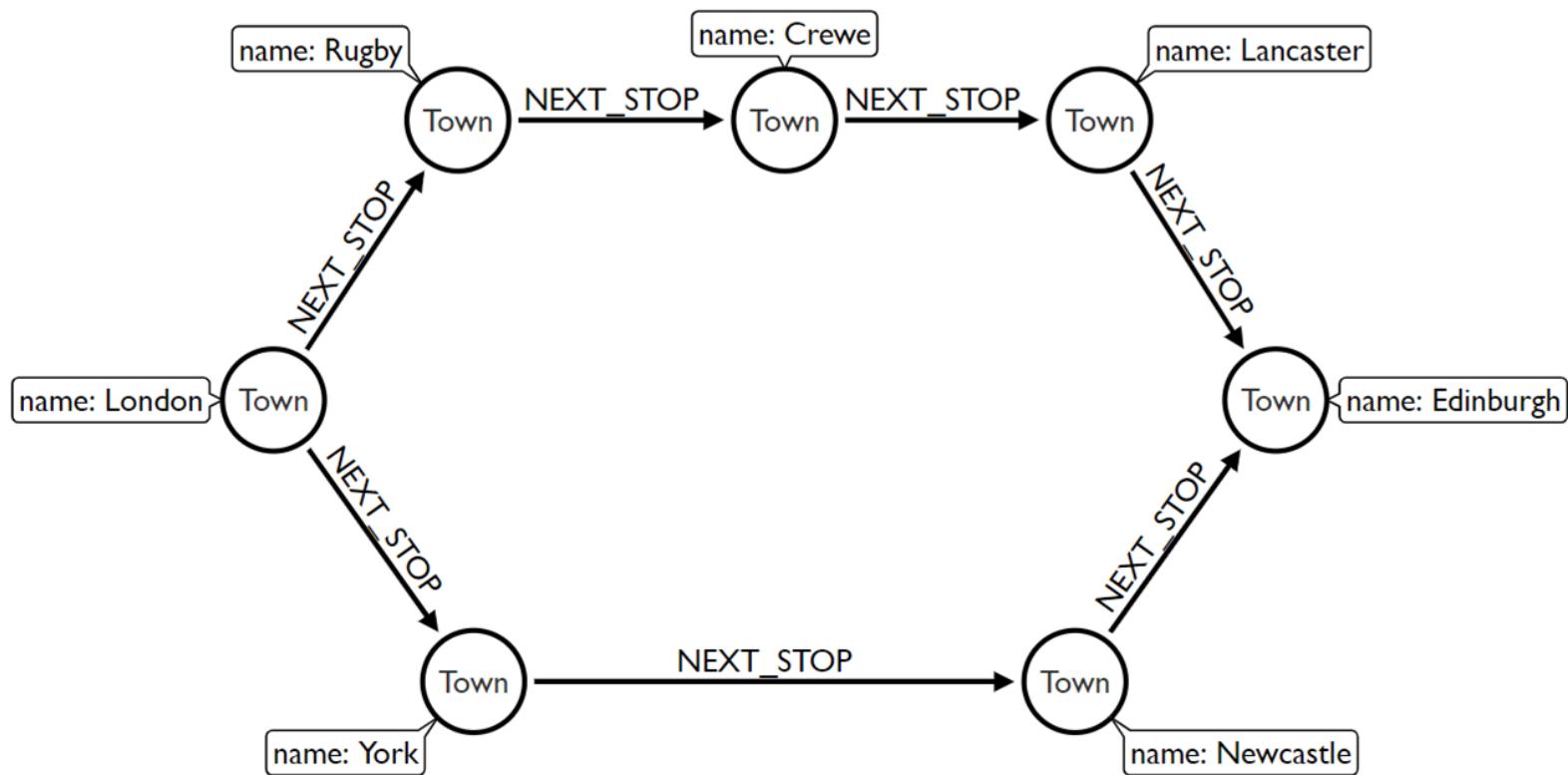
Self-referencing to the same type of entity



- Organizational hierarchies
- Access management
- Social influencers
- Friends of friends

# Graph scenario (3)

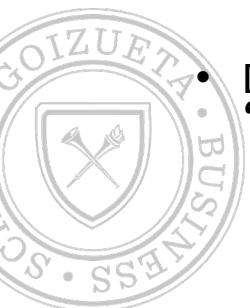
Discovering lots of different routes or paths?



- Logistics and routing
- Infrastructure management
- Dependency tracing

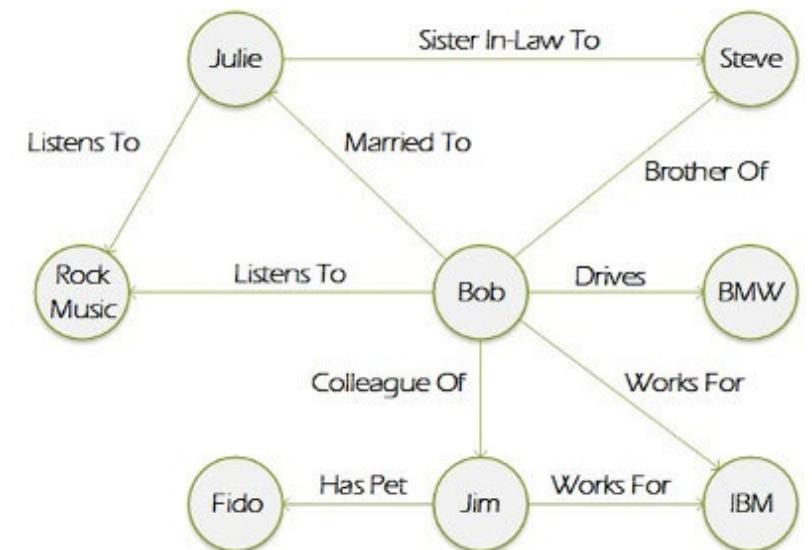
# NoSQL Graph Databases

- Often data have a “graph” format, i.e., form a network
  - “Friends on Facebook”, “Twitter Followers”, “LinkedIn connections”, “Supply chain network” etc.
- Common queries are of the type:
  - “How is X connected to Y?”
  - “How can I go from place A to place B?”
- Relational DBs can model graphs, but edges require frequent joins which are expensive.
- Key-value similarly cannot handle common graph queries well and require significant manual coding of such queries
- Document stores cannot store all the information for queries that require many “hops” in the network (e.g., “friends of friends”)



# NoSQL Graph Databases

- The data is represented as a graph, which is a collection of vertices (nodes) and edges.
  - Both nodes and edges can be labeled to indicate the types of entities and relationships they represent.
- It is generally possible to store data associated with both individual nodes and individual edges.
- Significantly different from the other three classes of NoSQL databases.
- Graph databases are based on the mathematical concept of graph theory.



# NoSQL Graph Databases: Examples

- Neo4j
  - Highly scalable open-source graph database that supports ACID
  - High availability
  - Well-documented REST interface
  - Most popular
- Titan – open-source graph store with support for ACID and eventual consistency
- OrientDB – Graph DB that supports Key-value DB and Document DB
  - Relationships are managed as in graph databases with direct connections between records



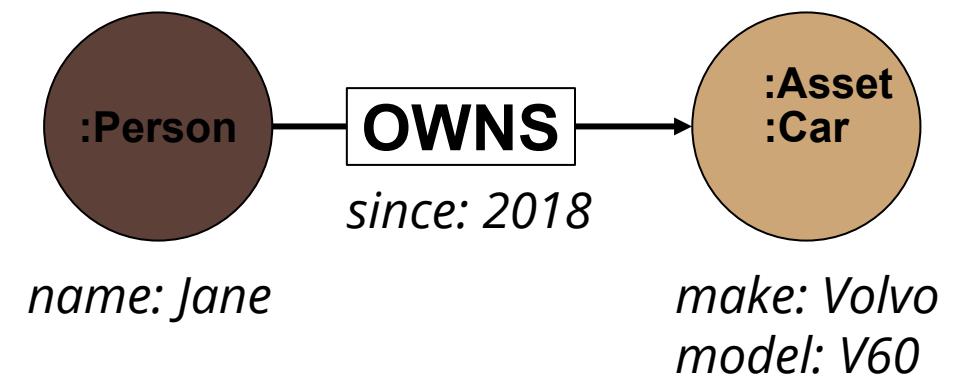
# Neo4j Overview

- Neo4j organizes data using the concepts of nodes (vertices) and relationships (edges).
- Both nodes and relationships can have properties, which store data items associated with nodes and relationships.
- Nodes can have (multiple) labels.
  - Nodes that have the same label are grouped into a collection for querying purposes.
- Relationships are directed.



# Neo4j - graph database

- Node (Vertex)
  - The main data element from which graphs are constructed
- Relationship (Edge)
  - A link between two nodes. Has:
    - Direction
    - Type
  - A node without relationships is permitted.  
A relationship without nodes is not
- Label
  - Define node category (optional)
  - Can have more than one
- Properties
  - Enrich a node or relationship
  - No need for nulls!



# Neo4j - Cypher Query Language

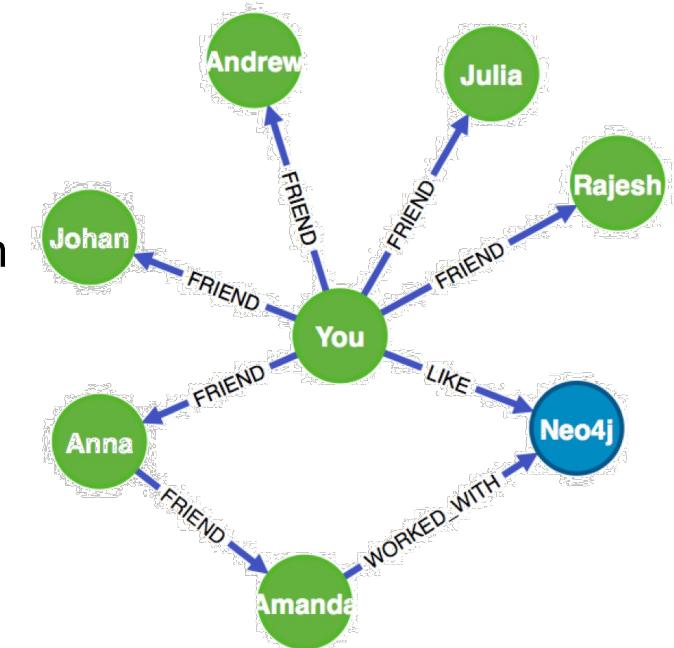
- Cypher is a declarative, SQL-inspired language for describing patterns in graphs
- Example:
  - Find someone in your network who can help you learn Neo4j

```
Defines variable "you"
MATCH (you {name:"You"})

Defines variable "expert"
MATCH (expert)-[:WORKED_WITH]->(db:Database
{name:"Neo4j"})

Defines variable "path", connecting "you" to "expert"
MATCH path = shortestPath((you)-[:FRIEND*..5]-(expert))

Returns the expert and the path
RETURN expert,path
```



# CypherQL - MATCH to retrieve nodes

```
//Match all nodes
MATCH (n)
RETURN n;
```

```
//Match all nodes with a Person label
MATCH (n:Person)
RETURN n;
```

```
//Match all nodes with a Person label and property name is "Tom Hanks"
MATCH (n:Person {name: "Tom Hanks"})
RETURN n;
```

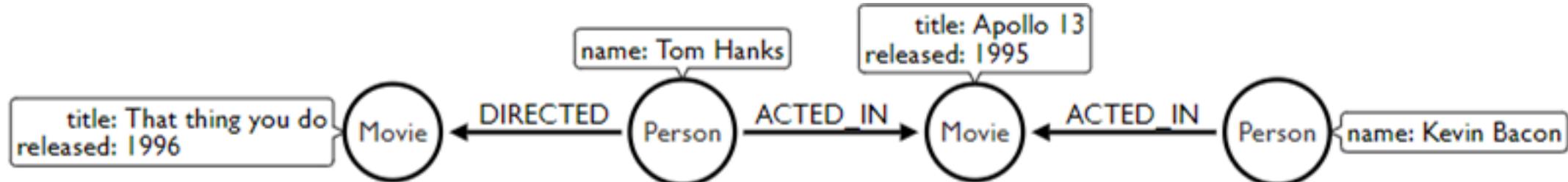


# CypherQL - MATCH and properties to retrieve nodes

```
//Return nodes with label Person and name property equals "Tom Hanks"
MATCH (p:Person)
WHERE p.name = "Tom Hanks"
RETURN p;
```

```
//Return nodes with label Movie, released property is between 1991 and 1999
MATCH (m:Movie)
WHERE m.released > 1990 AND m.released < 2000
RETURN m;
```

# CypherQL - Extending the MATCH



//Find all the movies Tom Hanks acted in

```
MATCH (:Person {name:"Tom Hanks"}) -[:ACTED_IN]-> (m:Movie)
RETURN m.title;
```

//Find all the movies Tom Hanks directed and order by latest movie

```
MATCH (:Person {name:"Tom Hanks"}) -[:DIRECTED]-> (m:Movie)
RETURN m.title, m.released ORDER BY m.released DESC;
```

//Find all of the co-actors Tom Hanks have worked with

```
MATCH (:Person {name:"Tom Hanks"}) --> (:Movie) <- [:ACTED_IN] - (coActor:Person)
RETURN coActor.name;
```

# CypherQL - CREATE

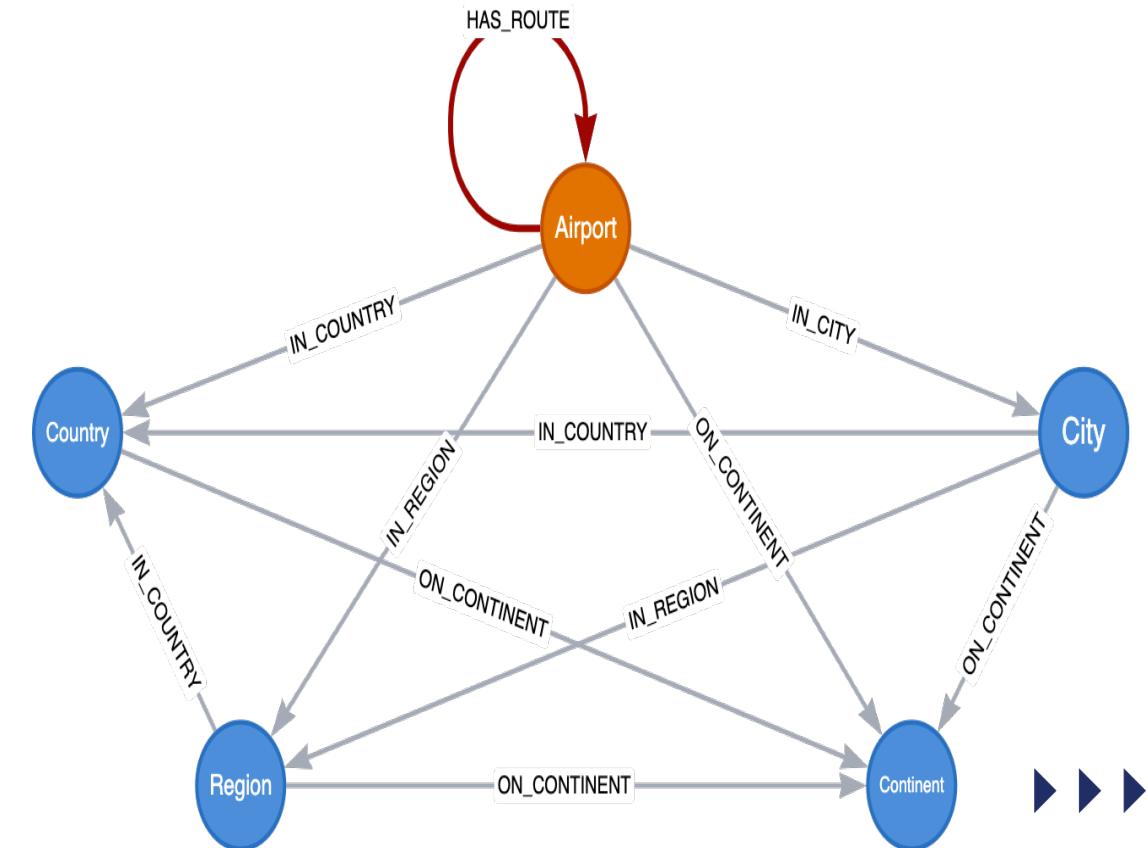
```
//Create a person node called "Tom Hanks"
CREATE (p:Person {name:"Tom Hanks"}) ;
```

```
//Create an ACTED_IN relationship between "Tom Hanks" and "Apollo 13"
MATCH (p:Person {name:"Tom Hanks"}) , (m:Movie {title:"Apollo 13"})
CREATE (p) -[:ACTED_IN] -> (m) ;
```

```
//Create the pattern of "Tom Hanks" ACTED_IN "Apollo 13"
//This will create the entire pattern, nodes and all!
CREATE (:Person {name:"Tom Hanks"}) -[:ACTED_IN] -> (:Movie {title:"Apollo 13"}) ;
```

# Neo4j Example: airplane routes

- <https://sandbox.neo4j.com/>
- 5 different node labels (Airport, City, Country, Continent, and Region)
- 5 different relationship types (:HAS\_ROUTE, :IN\_CITY, :IN\_COUNTRY, :IN\_REGION, and :ON\_CONTINENT).



# Data (CSV)

<https://github.com/krlawrence/graph/blob/master/sample-data/air-routes-latest-nodes.csv>

graph / sample-data / air-routes-latest-nodes.csv

 krlawrence August 2022 route updates #242 History

1 contributor

3750 lines (3750 sloc) | 417 KB

Q Search this file...

|    | -id | -label  | type:string | code:string | icao:string | desc:string                                                    |
|----|-----|---------|-------------|-------------|-------------|----------------------------------------------------------------|
| 2  | 0   | version | version     | 0.89        |             | Air Routes Data - Version: 0.89 Generated: 2022-08-29 14:10:18 |
| 3  | 1   | airport | airport     | ATL         | KATL        | Hartsfield - Jackson Atlanta International Airport             |
| 4  | 2   | airport | airport     | ANC         | PANC        | Anchorage Ted Stevens                                          |
| 5  | 3   | airport | airport     | AUS         | KAUS        | Austin Bergstrom International Airport                         |
| 6  | 4   | airport | airport     | BNA         | KBNA        | Nashville International Airport                                |
| 7  | 5   | airport | airport     | BOS         | KBOS        | Boston Logan                                                   |
| 8  | 6   | airport | airport     | BWI         | KBWI        | Baltimore/Washington International Airport                     |
| 9  | 7   | airport | airport     | DCA         | KDCA        | Ronald Reagan Washington National Airport                      |
| 10 | 8   | airport | airport     | DFW         | KDFW        | Dallas/Fort Worth International Airport                        |
| 11 | 9   | airport | airport     | FLL         | KFLL        | Fort Lauderdale/Hollywood International Airport                |
| 12 | 10  | airport | airport     | IAD         | KIAD        | Washington Dulles International Airport                        |
| 13 | 11  | airport | airport     | IAH         | KIAH        | George Bush Intercontinental                                   |
| 14 | 12  | airport | airport     | JFK         | KJFK        | New York John F. Kennedy International Airport                 |
| 15 | 13  | airport | airport     | LAX         | KLAX        | Los Angeles International Airport                              |
| 16 | 14  | airport | airport     | LGA         | KLGA        | New York La Guardia                                            |
| 17 | 15  | airport | airport     | MCO         | KMCO        | Orlando International Airport                                  |



# CypherQL - Create Nodes

- CREATE CONSTRAINT airports IF NOT EXISTS ON (a:Airport) ASSERT a.iata IS UNIQUE;
- CREATE CONSTRAINT cities IF NOT EXISTS ON (c:City) ASSERT c.name IS UNIQUE;
- CREATE CONSTRAINT regions IF NOT EXISTS ON (r:Region) ASSERT r.name IS UNIQUE;
- CREATE CONSTRAINT countries IF NOT EXISTS ON (c:Country) ASSERT c.code IS UNIQUE;
- CREATE CONSTRAINT continents IF NOT EXISTS ON (c:Continent) ASSERT c.code IS UNIQUE;
- CREATE INDEX locations IF NOT EXISTS FOR (air:Airport) ON (air.location);



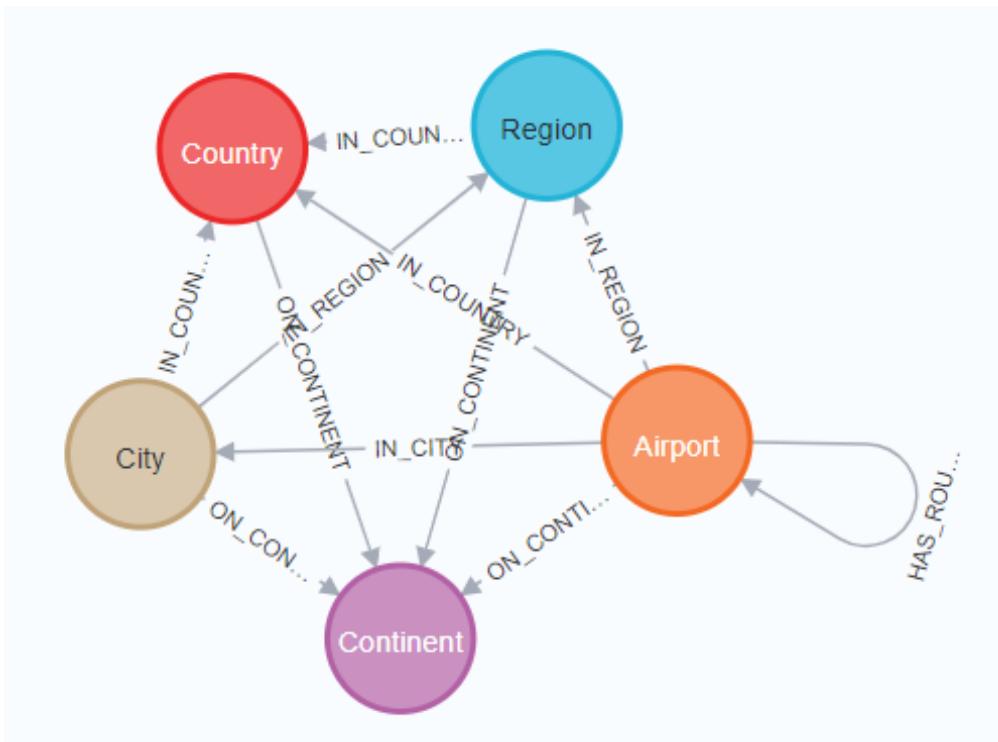
# CypherQL – Load Data

- WITH 'https://raw.githubusercontent.com/neo4j-graph-examples/graph-data-science2/main/data/airport-node-list.csv' AS url
- LOAD CSV WITH HEADERS FROM url AS row
- MERGE ...



# CypherQL – Schema Visualization

- `CALL db.schema.visualization()`



# CypherQL – Graph Data Science (PageRank)

- “PageRank algorithm measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The underlying assumption roughly speaking is that a page is only as important as the pages that link to it.”
- `CALL gds.pageRank.stream('routes')`
- `YIELD nodeId, score`
- `WITH gds.util.asNode(nodeId) AS n, score AS pageRank`
- `RETURN n.iata AS iata, n.descr AS description, pageRank`
- `ORDER BY pageRank DESC, iata ASC`



<https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/>

# CypherQL – Graph Data Science (Louvain)

- “Louvain method is an algorithm to detect communities in large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network.”
- ```
CALL gds.louvain.stream('routes')
YIELD nodeId, communityId
WITH gds.util.asNode(nodeId) AS n, communityId
RETURN
  communityId,
  SIZE(COLLECT(n)) AS numberofAirports,
  COLLECT(DISTINCT n.city) AS cities
ORDER BY numberofAirports DESC, communityId;
```



<https://neo4j.com/docs/graph-data-science/current/algorithms/louvain/>

CypherQL – Graph Data Science (Dijkstra)

- Dijkstra Shortest Path algorithm computes the shortest path between nodes.”

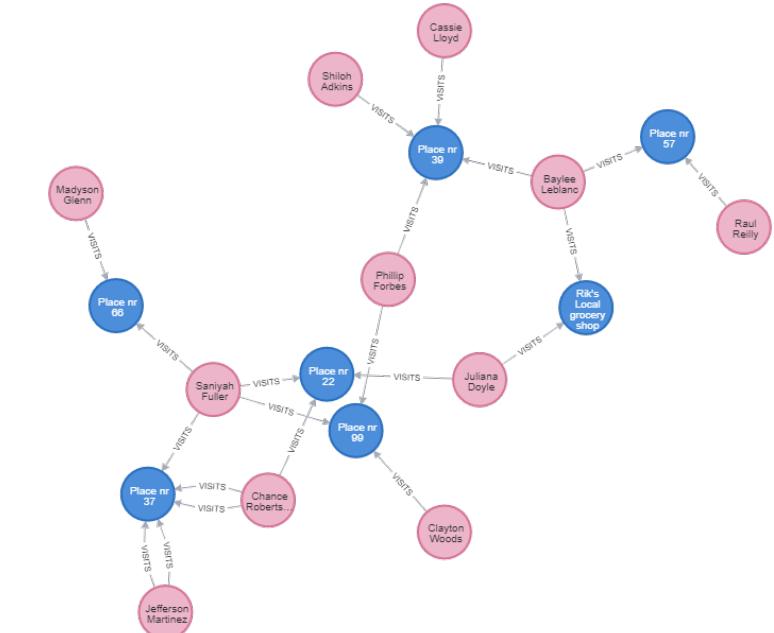
```
• MATCH (source:Airport {iata: 'ATL'}), (target:Airport {iata: 'KTM'})  
• CALL gds.shortestPath.dijkstra.stream('routes-weighted', {  
•   sourceNode: source,  
•   targetNode: target,  
•   relationshipWeightProperty: 'distance'  
• })  
• YIELD index, sourceNode, targetNode, nodeIds, path  
• RETURN  
•   index,  
•   gds.util.asNode(sourceNode).iata AS sourcenodeName,  
•   gds.util.asNode(targetNode).iata AS targetnodeName,  
•   [nodeId IN nodeIds | gds.util.asNode(nodeId).iata] AS nodeNames,  
•   nodes(path) as path  
ORDER BY index
```



<https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-source-target/>

Neo4j Example

- Visit: <https://neo4j.com/sandbox/>
- Select: pre-built data project (e.g. contact tracing)
- Open with browser
- Follow the steps
 - Alternatively: <https://guides.neo4j.com/sandbox/contact-tracing/index.html>
- Sample queries:
 - CALL db.schema.visualization() - visual representation of data structure
 - MATCH (p:Person) RETURN DISTINCT p.healthstatus, count(*);
 - MATCH (p:Person {healthstatus:"Sick"})-[v1:VISITS]->(pl:Place)
WITH p,v1,pl LIMIT 10
MATCH path = (p)-[v1]->(pl)<-[v2:VISITS]-(p2:Person {healthstatus:"Healthy"})
WITH path, apoc.coll.max([v1starttime.epochMillis, v2starttime.epochMillis]) as maxStart,
apoc.coll.min([v1endtime.epochMillis, v2endtime.epochMillis]) as minEnd
WHERE maxStart <= minEnd
RETURN path;



Which NoSQL Database to Choose?

- Key-value databases are generally useful for storing session information, user profiles, preferences, shopping cart data, etc.
- Document databases are generally useful for content management systems, blogging platforms, web analytics, real-time analytics, ecommerce-applications, etc.
- Column-based databases are generally useful for blogging platforms, maintaining counters, expiring usage, and heavy write volume such as log aggregation.
- Graph databases are well suited to problem spaces where we have connected data, such as social networks, spatial data, routing information for goods and money, etc.





neo4j

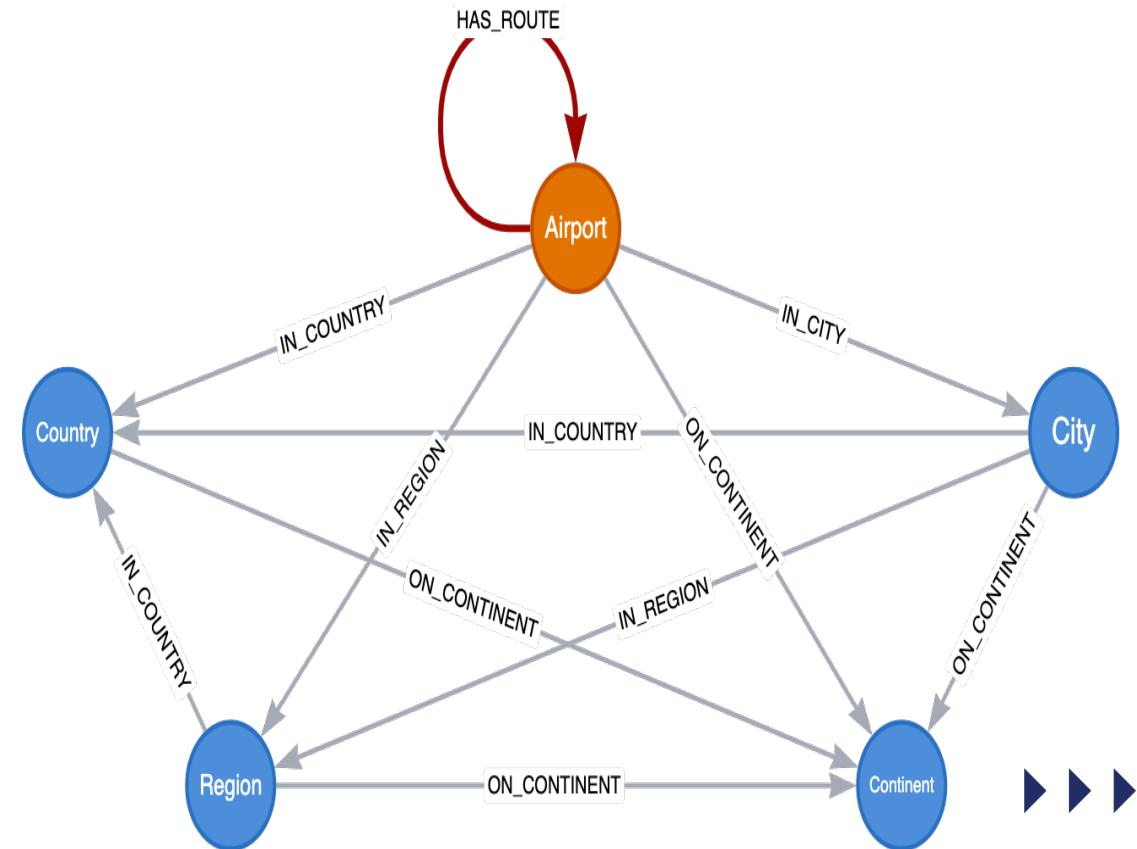
NoSQL IV

CATCHUP

October 16, 2023

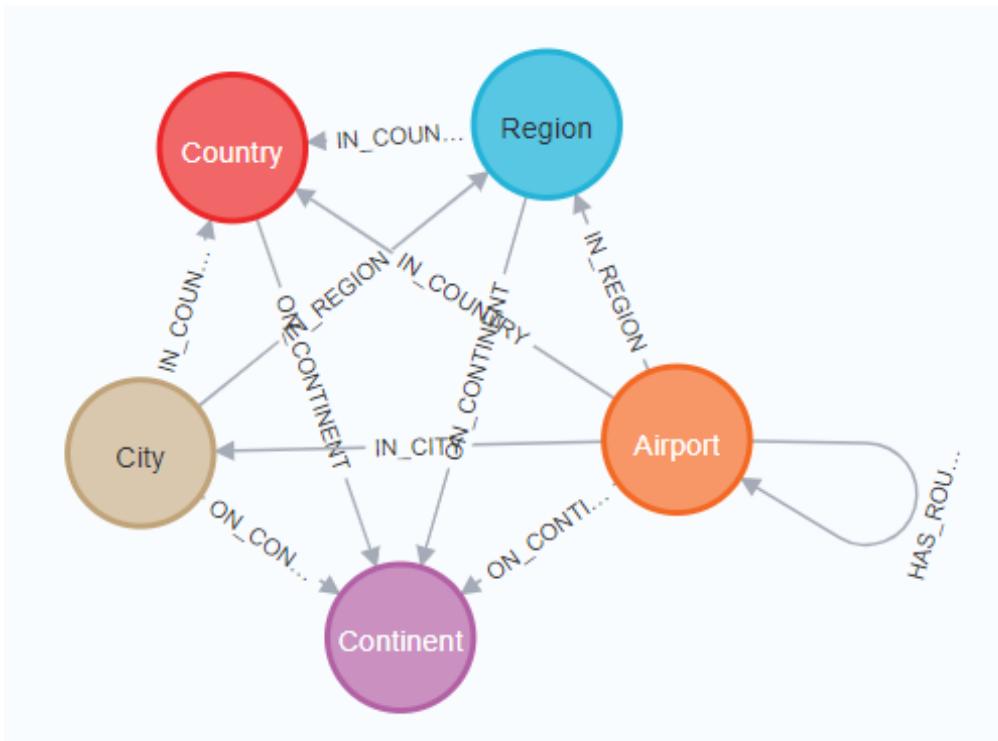
Neo4j Example: airplane routes

- <https://sandbox.neo4j.com/>
- 5 different node labels (Airport, City, Country, Continent, and Region)
- 5 different relationship types (:HAS_ROUTE, :IN_CITY, :IN_COUNTRY, :IN_REGION, and :ON_CONTINENT).
- Create data: <https://github.com/krlawrence/graph/blob/master/sample-data/Cypher/air-routes.cypher>



CypherQL – Schema Visualization

- `CALL db.schema.visualization()`



CypherQL – Graph Data Science (PageRank)

- “PageRank algorithm measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The underlying assumption roughly speaking is that a page is only as important as the pages that link to it.”

```
• CALL gds.pageRank.stream('routes')
• YIELD nodeId, score
• WITH gds.util.asNode(nodeId) AS n, score AS pageRank
• RETURN n.iata AS iata, n.descr AS description, pageRank
• ORDER BY pageRank DESC, iata ASC
```

iata	description	pageRank
"DFW"	"Dallas/Fort Worth International Airport"	11.97978261
"ORD"	"Chicago O'Hare International Airport"	11.16298818
"DEN"	"Denver International Airport"	10.99729934
"ATL"	"Hartsfield - Jackson Atlanta International Airport"	10.38994835
"IST"	"Istanbul International Airport"	8.425801218
"CDG"	"Paris Charles de Gaulle"	8.401469085
"IAH"	"George Bush Intercontinental"	8.341141085
"FRA"	"Frankfurt am Main"	8.203204539
"LAX"	"Los Angeles International Airport"	8.193558075

<https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/>



CypherQL – Graph Data Science (Louvain)

- “Louvain method is an algorithm to detect communities in large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network.”
- ```
CALL gds.louvain.stream('routes')
YIELD nodeId, communityId
WITH gds.util.asNode(nodeId) AS n, communityId
RETURN
 communityId,
 SIZE(COLLECT(n)) AS numberofAirports,
 COLLECT(DISTINCT n.city) AS cities
ORDER BY numberofAirports DESC, communityId;
```

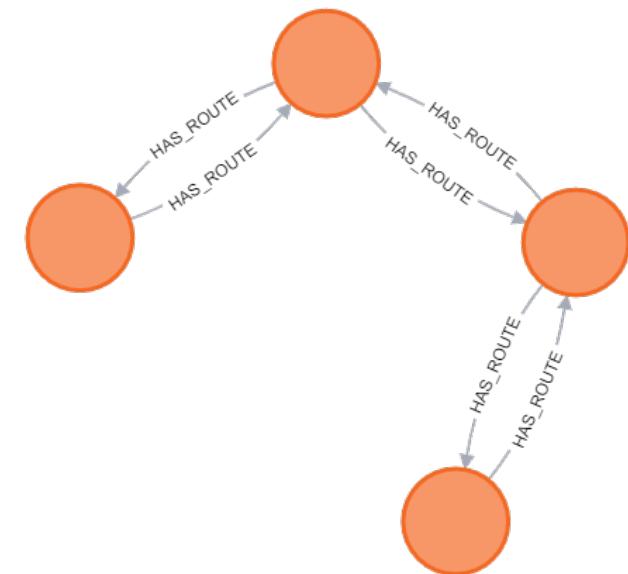


<https://neo4j.com/docs/graph-data-science/current/algorithms/louvain/>

# CypherQL – Graph Data Science (Dijkstra)

- Dijkstra Shortest Path algorithm computes the shortest path between nodes.

```
• MATCH (source:Airport {iata: 'ATL'}), (target:Airport {iata: 'KTM'})
• CALL gds.shortestPath.dijkstra.stream('routes-weighted', {
• sourceNode: source,
• targetNode: target,
• relationshipWeightProperty: 'distance'
• })
• YIELD index, sourceNode, targetNode, nodeIds, path
• RETURN
• index,
• gds.util.asNode(sourceNode).iata AS sourcenodeName,
• gds.util.asNode(targetNode).iata AS targetnodeName,
• [nodeId IN nodeIds | gds.util.asNode(nodeId).iata] AS nodeNames,
• nodes(path) AS path
• ORDER BY index
```

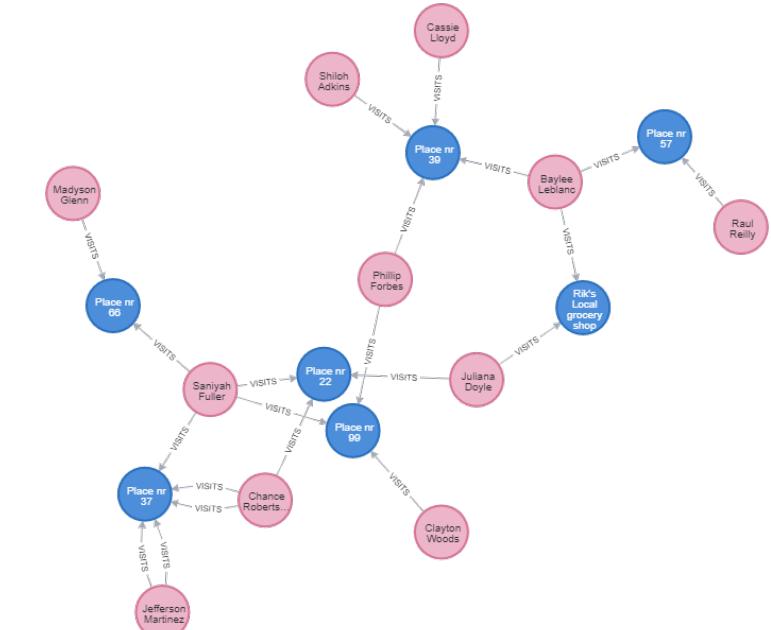


<https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-source-target/>



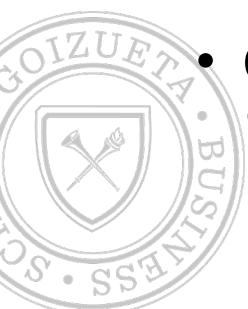
# Neo4j Example

- Visit: <https://neo4j.com/sandbox/>
- Select: pre-built data project (e.g. contact tracing)
- Open with browser
- Follow the steps
  - Alternatively: <https://guides.neo4j.com/sandbox/contact-tracing/index.html>
- Sample queries:
  - CALL db.schema.visualization() - visual representation of data structure
  - MATCH (p:Person) RETURN DISTINCT p.healthstatus, count(\*);
  - MATCH (p:Person {healthstatus:"Sick"})-[v1:VISITS]->(pl:Place)  
WITH p,v1,pl LIMIT 10  
MATCH path = (p)-[v1]->(pl)<-[v2:VISITS]-(p2:Person {healthstatus:"Healthy"})  
WITH path, apoc.coll.max([v1starttime.epochMillis, v2starttime.epochMillis]) as maxStart,  
apoc.coll.min([v1endtime.epochMillis, v2endtime.epochMillis]) as minEnd  
WHERE maxStart <= minEnd  
RETURN path;



# Which NoSQL Database to Choose?

- Key-value databases are generally useful for storing session information, user profiles, preferences, shopping cart data, etc.
- Document databases are generally useful for content management systems, blogging platforms, web analytics, real-time analytics, ecommerce-applications, etc.
- Column-based databases are generally useful for blogging platforms, maintaining counters, expiring usage, and heavy write volume such as log aggregation.
- Graph databases are well suited to problem spaces where we have connected data, such as social networks, spatial data, routing information for goods and money, etc.



# Things to try:

- DocumentDB on AWS:
  - <https://docs.aws.amazon.com/documentdb/latest/developerguide/get-started-guide.html>
  - Tutorial: [https://www.youtube.com/watch?v=Ild9ay9U\\_vY](https://www.youtube.com/watch?v=Ild9ay9U_vY)
- Hbase on AWS (Column-Oriented Database):
  - <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase.html>
- Neo4j Sandbox:
  - <https://neo4j.com/sandbox/>
  - Tutorial: <https://www.youtube.com/watch?v=fiQzHVY-kRQ>
  - Tutorial (AWS Neptune): <https://www.youtube.com/watch?v=rsAKj7sMbbQ>





# Hadoop I

Distributed File System

October 16, 2023

This Photo by Unknown Author is licensed under CC BY

# Big Data!

- LinkedIn boasts over **850 million members** in more than 200 countries.
  - <https://about.linkedin.com/>
- Facebook has over **2.9 billion active monthly users** worldwide today
  - <https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/>
- Twitter had an estimated **390 million user accounts**, and it was reported to have reached the rate of **500 million tweets per day** in 2022.
  - <https://www.omnicoreagency.com/twitter-statistics/>
- IoT, stock exchanges, satellites, sensors, etc. → datasets whose size exceeds the typical reach of an RDBMS to capture, store, manage, and analyze that data.



# Data Sources

- Science
  - Medical imaging, sensor data, genome sequencing, weather data, satellite feeds, etc.
- Industry
  - Financial, pharmaceutical, manufacturing, insurance, online, energy, retail data, etc.
- Legacy
  - Sales data, customer behavior, product databases, accounting data, etc.
- System Data
  - Log files, health and status feeds, activity streams, network messages, web analytics, intrusion detection, spam filters, etc.

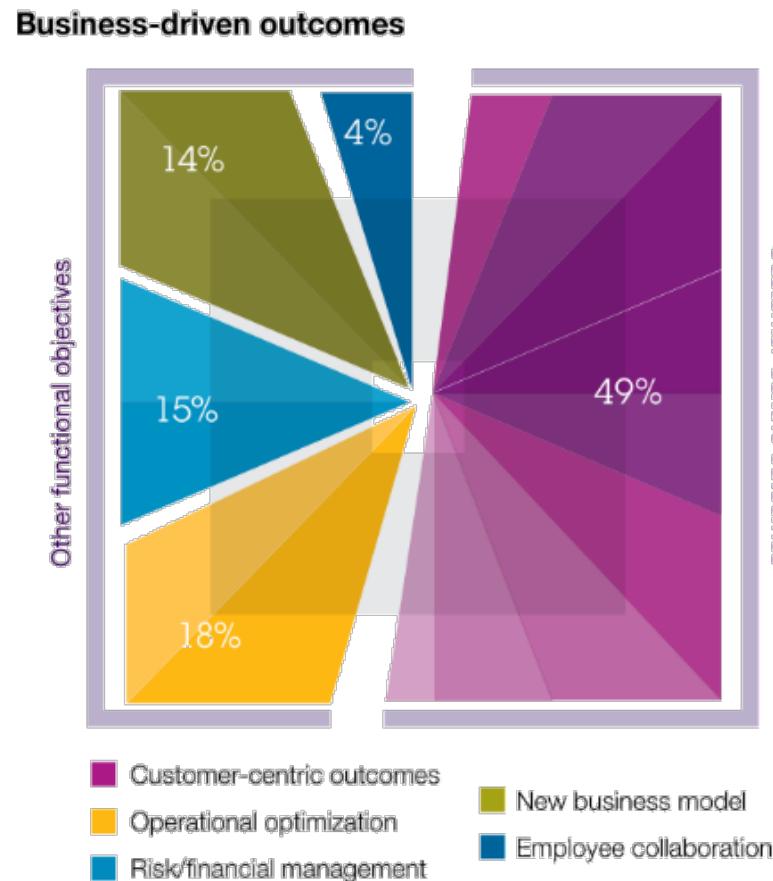


• ..



# Data Use

- Customer-centric activities are the top priority

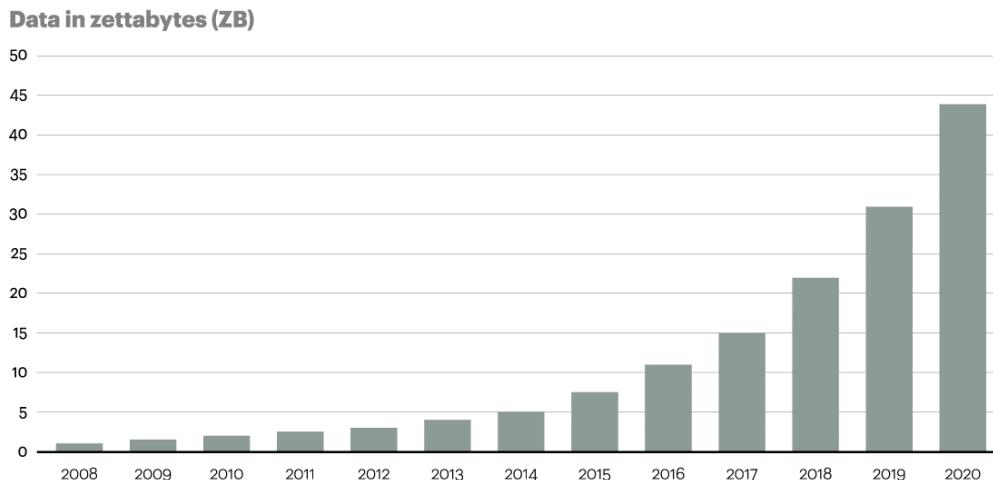


Source: Analytics: The real-world use of big data  
© 2012 IBM, IBM Institute for Business Value



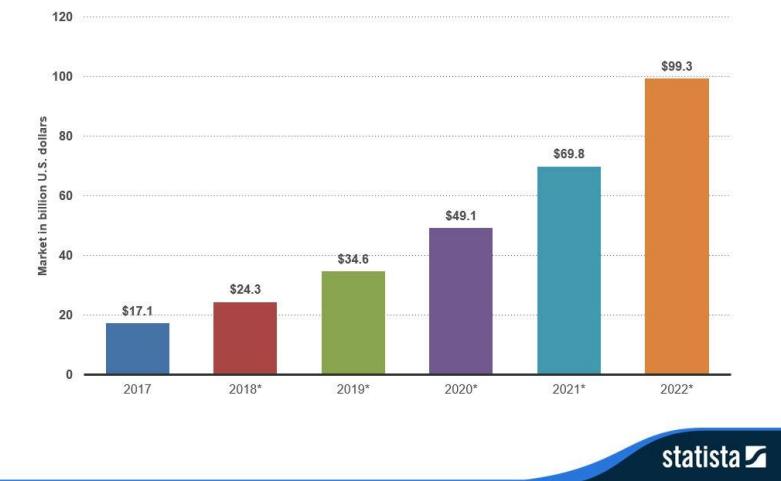
# Fast Growth

**Data is growing at a 40 percent compound annual rate, reaching nearly 45 ZB by 2020**



**Zettabytes: 1B TB**

Big Data and Hadoop Market Size Forecast Worldwide 2017-2022  
Size of Hadoop and Big Data Market Worldwide From 2017 To 2022  
(in billion U.S. dollars)



Source: <https://www.forbes.com/sites/louis columbus/2018/05/23/10-charts-that-will-change-your-perspective-of-big-datas-growth/#21e781882926>



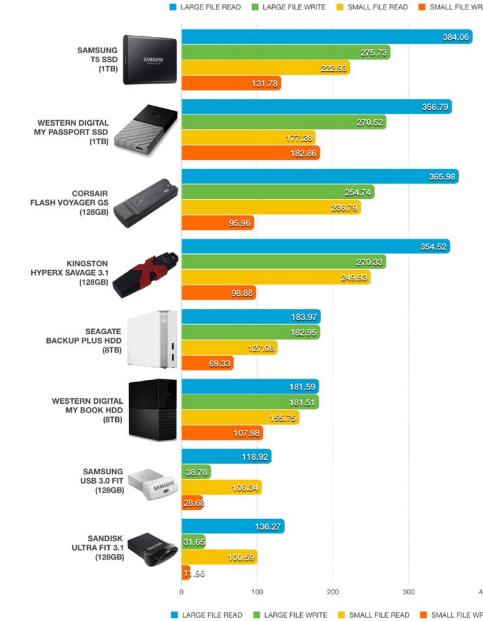
# Size and cost of storage

- Fortunately, the size and cost of storage have kept pace

| Year | Capacity (GB) | Cost per GB (USD) |
|------|---------------|-------------------|
| 1992 | 0.08          | \$3,827.20        |
| 1997 | 2.10          | \$157.00          |
| 2002 | 80.00         | \$3.74            |
| 2007 | 750.00        | \$0.35            |
| 2012 | 3,000.00      | \$0.05            |

- We can now retain what we previous discarded.





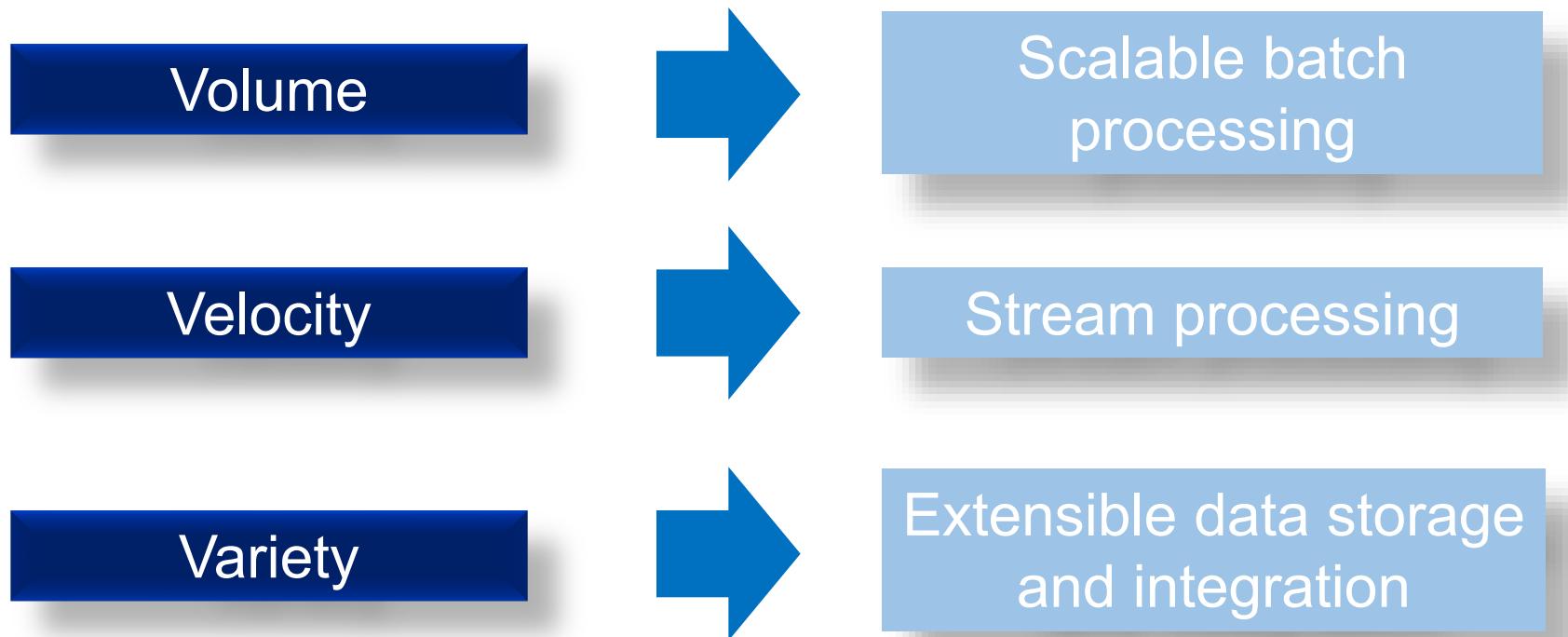
# Managing volume

- While storage capacity has increased, and storage costs have gone down; the transfer speed has fallen behind.
- How much time does it take to read a one Terabyte (1TB) of data into memory?
  - Suppose the reading speed is 100MB/s (approximate read speed of USB and conventional HDDs)
    - *It takes about 10,000sec = 2.8hrs*
  - Time to store/read the daily internet traffic of about 400 Petabytes (<https://www.internetlivestats.com/>)?
    - *It takes about 1.1 million hours = 126 years*
  - How do you solve this?
    - Parallel processing!



Source: <https://www.everythingusb.com/speed.html>

# Addressing the 3Vs:





# Where Did Hadoop Come From?

- Started with research at Google in 2003.
  - Google's objective was to index the entire World Wide Web
  - Google had reached the limits of scalability of RDBMS technology
  - Research led to a new approach to store and analyze large volume of data
    - Google File Systems (Ghemawat et al. 2003)
    - MapReduce: Simplified Data Processing on Large Clusters (Dean and Ghemawat 2008)
- Doug Cutting wrestling with many of the same problems in the implementation of his own open-source search engine “Lucene”.
  - Started open-source project based on Google’s research and created Hadoop in 2005.
  - Hadoop was named after his son’s toy elephant.
  - Hadoop became an open-source Apache project.



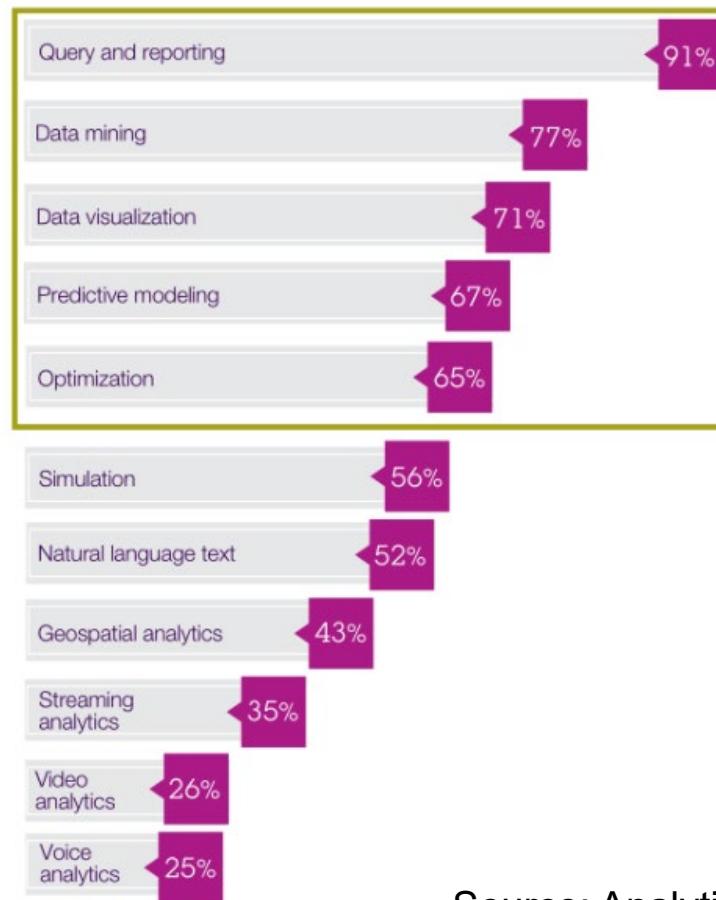


# What is Apache Hadoop?

- Apache Hadoop is a software framework for storing, processing, and analyzing “big data”
  - Distributed
  - Scalable
  - Fault-tolerant
  - Open source
- Industries that use Hadoop:
  - Finance
  - Healthcare
  - Manufacturing
  - Telecommunications
  - Nonprofit
  - ..



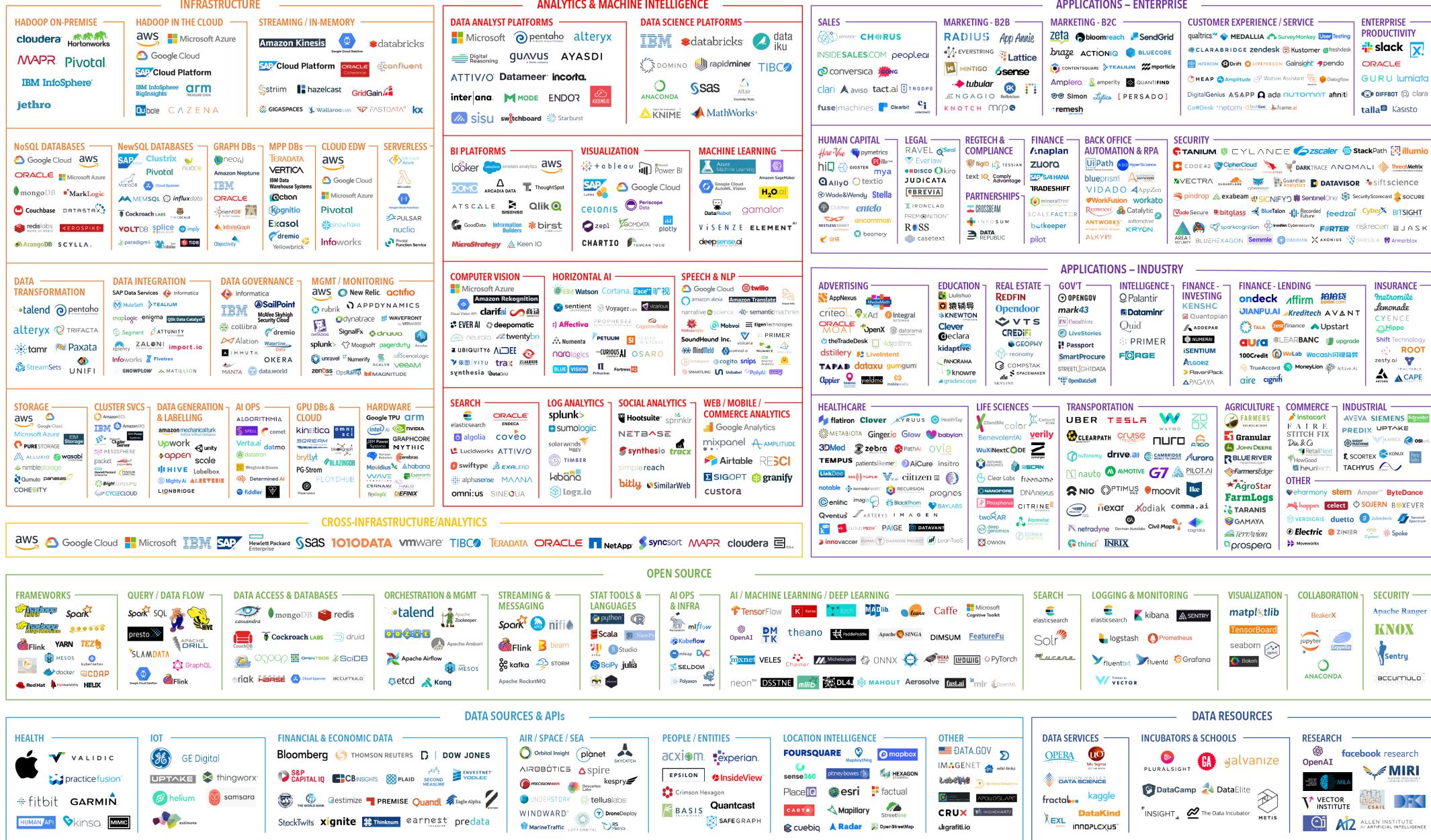
# Capabilities that organizations are utilizing



Source: Analytics: The real-world use of big data  
© 2012 IBM, IBM Institute for Business Value



DATA & AI LANDSCAPE 2019





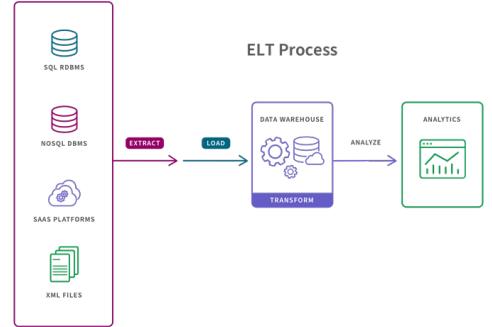
# Apache Hadoop

- A radical new approach to distributed computing
  - Distribute data when the data is being stored
  - Run computation where the data is stored
- Core Hadoop concepts:
  - Applications are written in high-level code
  - Nodes talk to each other as little as possible
  - Data is distributed in advance
    - Bring the computation to the data
  - Data is replicated for increased availability and reliability
  - Hadoop is scalable and fault-tolerant



# Hadoop Use Cases

- Extract, Transform, and Load (ETL)
  - Often now ELT: Extract, Load, then Transform
  - Once processed, data can be analyzed in Hadoop or moved to a DW
- Business Intelligence
  - Augments traditional data warehouses
- Predictive Analytics
  - Entire data sets can be used; no need to sample data
  - Statistical and machine learning models can run inside the Hadoop environment
- Low-Cost Data Storage
  - Fault tolerant architecture
  - Long-term storage of structured and unstructured data
  - Enterprise data hub



# Big Data Use Cases by Industry

- Security
  - Threat analysis and detection, security analytics, anti-virus, image recognition
- Social media
  - User demographics, brand perception
- Gaming
  - In-game analytics, usage analysis
- Web and mobile apps
  - Identify user trends, web indexing, log processing and analysis



# Big Data Use Cases by Industry

- Travel
  - Travel recommendations, dynamic pricing
- Airlines
  - Customer data mining, dynamic pricing
- Telcom
  - Analyze call records, operational metrics
- Government
  - Analyze public data sets, drive research
- Business Intelligence
  - Data warehousing, data visualization



# Common Types of Analysis with Hadoop

- Text mining
- Index building
- Graph creation and analysis
- Pattern recognition
- Collaborative filtering
- Prediction models
- Sentiment analysis
- Risk assessment
- ..



# Technology Strengths and Weaknesses

| Technology             | Strengths                                                                                                                                                     | Drawbacks                                                                                                                                                      |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Hadoop                 | <ul style="list-style-type: none"><li>• Scales to petabytes</li><li>• Import/export tools</li><li>• Flexible structure</li><li>• Commodity hardware</li></ul> | <ul style="list-style-type: none"><li>• Query speed</li><li>• No transaction support</li></ul>                                                                 |
| Relational Databases   | <ul style="list-style-type: none"><li>• Complex transactions</li><li>• 1000s of queries/second</li><li>• SQL</li></ul>                                        | <ul style="list-style-type: none"><li>• Data must fit into rows and columns</li><li>• Schema is costly to change</li><li>• Full table scans are slow</li></ul> |
| Data Warehouses        | <ul style="list-style-type: none"><li>• Reporting capabilities</li><li>• Up to 100s of terabytes</li></ul>                                                    | <ul style="list-style-type: none"><li>• Dimensions require pre-materialization</li></ul>                                                                       |
| File servers (SAN/NAS) | <ul style="list-style-type: none"><li>• Serving individual files</li><li>• Write caches</li></ul>                                                             | <ul style="list-style-type: none"><li>• Cost</li><li>• Reading large portions of the data saturates the network</li></ul>                                      |
| Backup systems (tape)  | <ul style="list-style-type: none"><li>• Cheap</li></ul>                                                                                                       | <ul style="list-style-type: none"><li>• Expensive to retrieve the data</li></ul>                                                                               |





# What is Apache Hadoop?

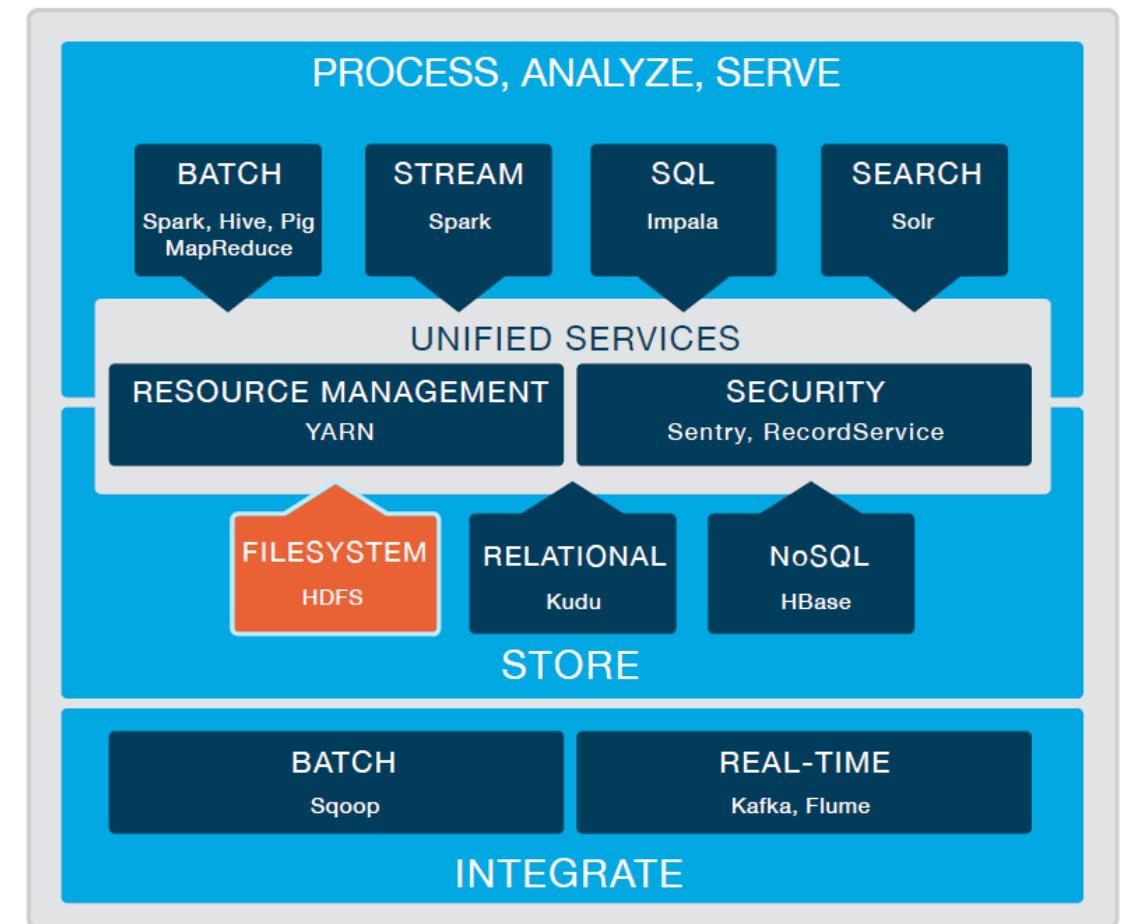
- Core Hadoop is a File System and a Processing Framework:
- The Hadoop Distributed File System (HDFS)
  - Any type of file can be stored in HDFS
  - Data is split into chunks and replicated as it is written
    - Provides resiliency and high availability
    - Handled automatically by Hadoop
- YARN (Yet Another Resource Negotiator)
  - Manages the processing resources of the Hadoop cluster
  - Schedules jobs
  - Runs processing frameworks
- MapReduce
  - A distributed processing framework

Core concept: distribute the data when initially stored in the system



# HDFS: Hadoop Distributed File System

- HDFS provides the storage layer for Hadoop data processing
- Provides inexpensive and reliable storage for massive amounts of data
- Other Hadoop components work with data in HDFS
  - Such as MapReduce, Hive, Impala, Pig, and Spark



# HDFS: Hadoop Distributed File System

- HDFS is the file system component of Hadoop and is designed to run on a cluster of commodity hardware.
  - HDFS is written in Java and sits on top of a native filesystem (e.g., EXT4)
- HDFS stores file system metadata and application data separately:
  - The metadata is stored on a dedicated server, called the NameNode;
  - The application data is stored on other servers, called DataNodes.
- All servers are fully connected and communicate with each other using TCP-based protocols.
- To make data durable, the file content is replicated on multiple DataNodes, as in the Google File System.
  - This increases reliability, multiplies the bandwidth for data transfer and enables colocation of computation with data.



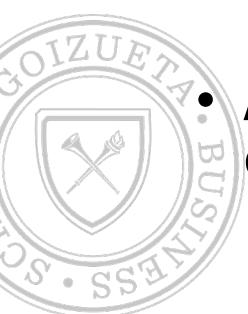
# HDFS Goals

- Hardware failure
  - Using commodity hardware and thousands of nodes, automatic detection and recovery from failures of hardware is important.
- Batch processing
  - High throughput is emphasized over low latency of data access. Full scans of files are typical.
- Large datasets
  - HDFS was designed to support huge files in the hundreds of gigabytes to terabytes range.
- Simple coherency model
  - HDFS applications need a ‘one writer and many reader’ access models for files. File content cannot be updated, but only appended. This model alleviates coherency issues among copies of data.

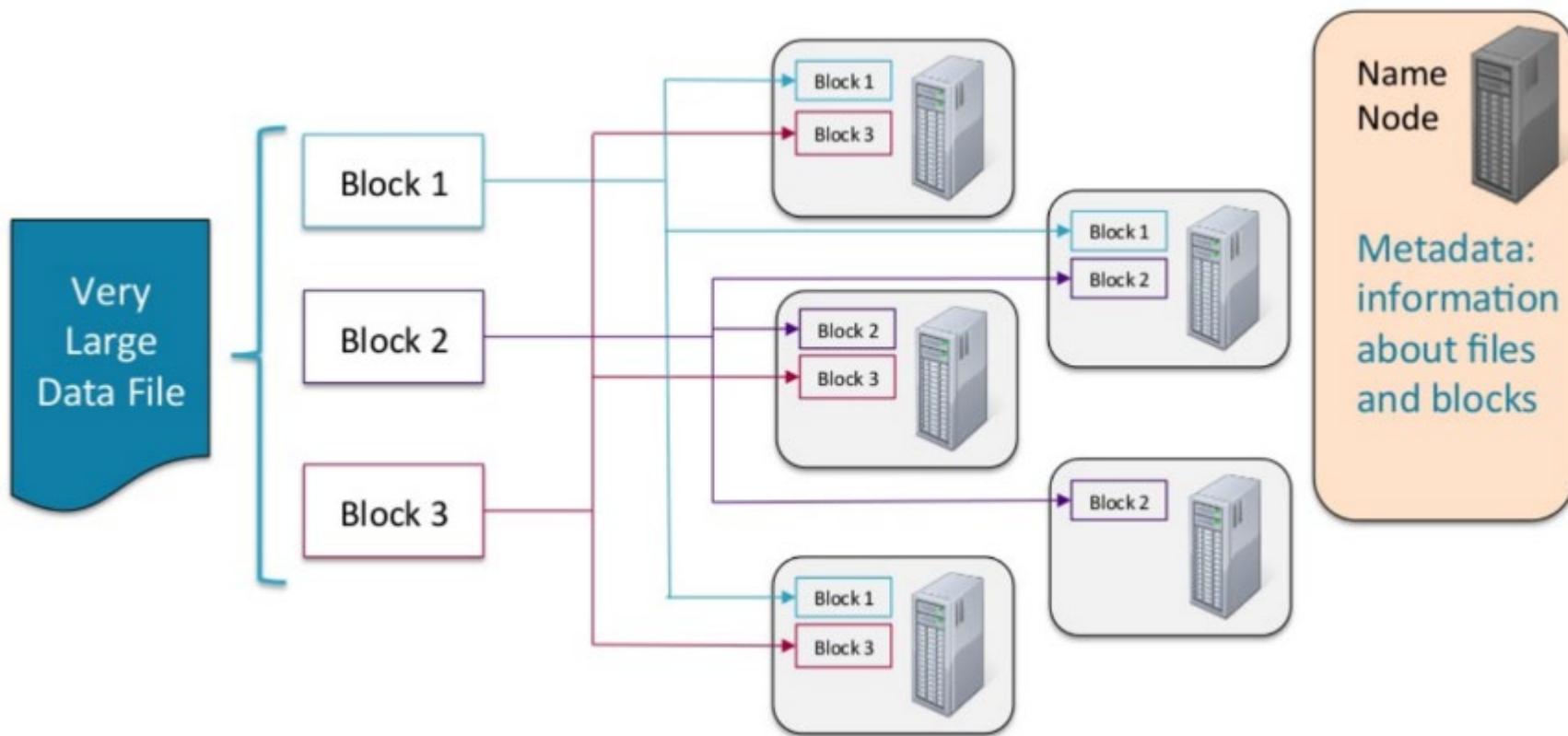


# Architecture of HDFS

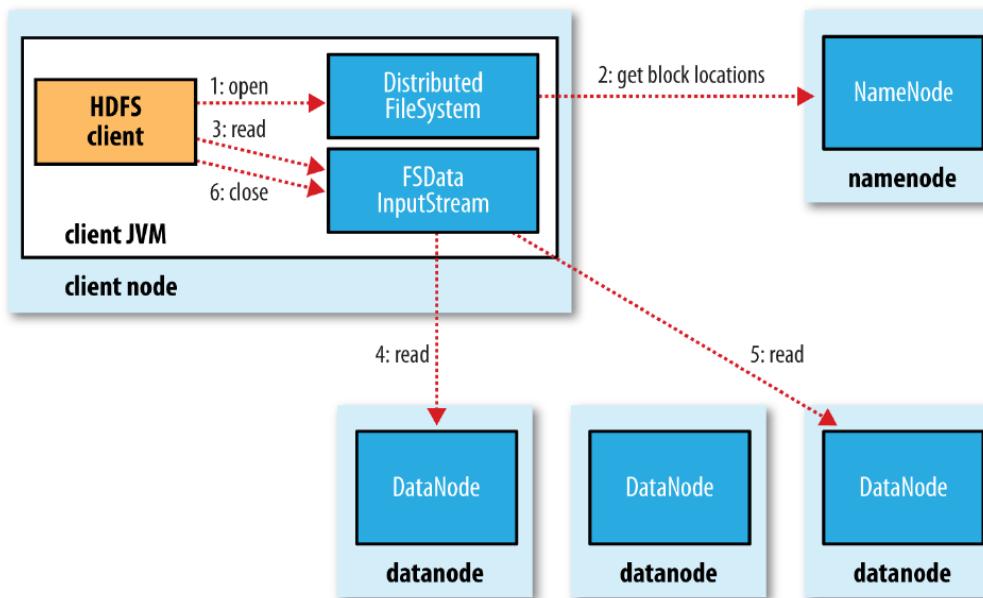
- HDFS has a master-worker architecture.
- The master server, called the NameNode, manages the file system storage area (or namespace).
  - Namenodes maintain inodes (index nodes) about files and directories with attributes like ownership, permissions, creation and access times.
  - Clients access the namespace through the Namenode.
- The workers, called DataNodes, manage the storage attached to the node that they run on.
  - They run on a cluster of commodity machines, usually one per machine.
  - DataNodes are responsible for serving read and write requests from clients.
  - DataNodes perform block creation, deletion, and replication operations as instructed by the NameNode.
- A cluster can have thousands of DataNodes and tens of thousands of HDFS clients simultaneously connected.



# How files are stored

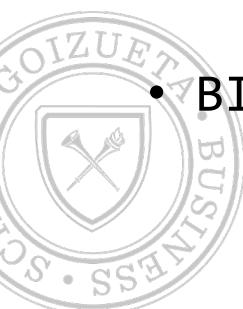


# Data Flow: File Read



# Getting data in and out of HDFS

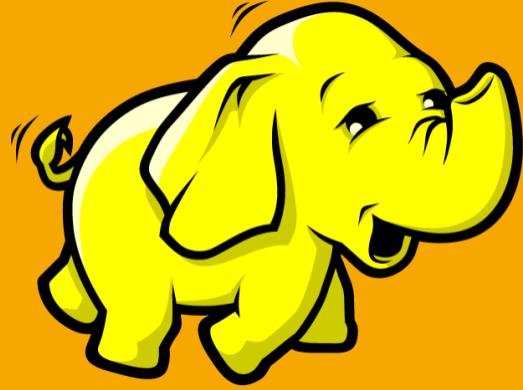
- Hadoop
  - Copies data between client (local) and HDFS (cluster)
  - API or command line
- Ecosystem Projects
  - Flume
    - Collects data from network sources (e.g., websites, system logs)
  - Sqoop
    - Transfers data between HDFS and RDBMS
- BI and ETL tools



# Hadoop Distributed File System (HDFS) Summary

- HDFS is the storage layer for Hadoop
- A filesystem which can store any type of data
- Provides inexpensive and reliable storage for massive amounts of data
  - Data is replicated across computers
- HDFS performs best with a ‘modest’ number of larger files
  - Millions, rather than billions of files
  - Each file is typically 100MB or more
- Files in HDFS are ‘write once’
  - Appends are permitted
  - No random writes are allowed

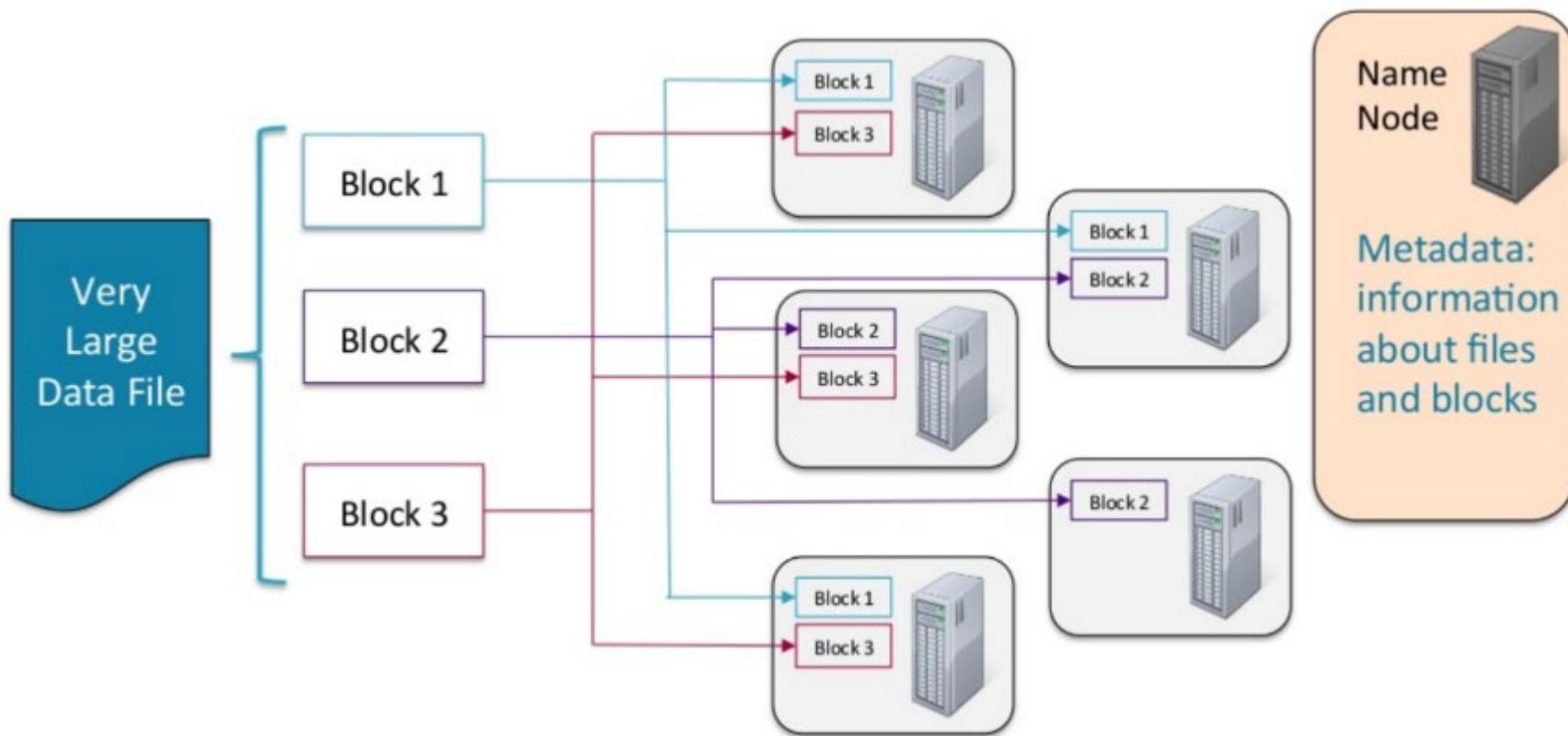




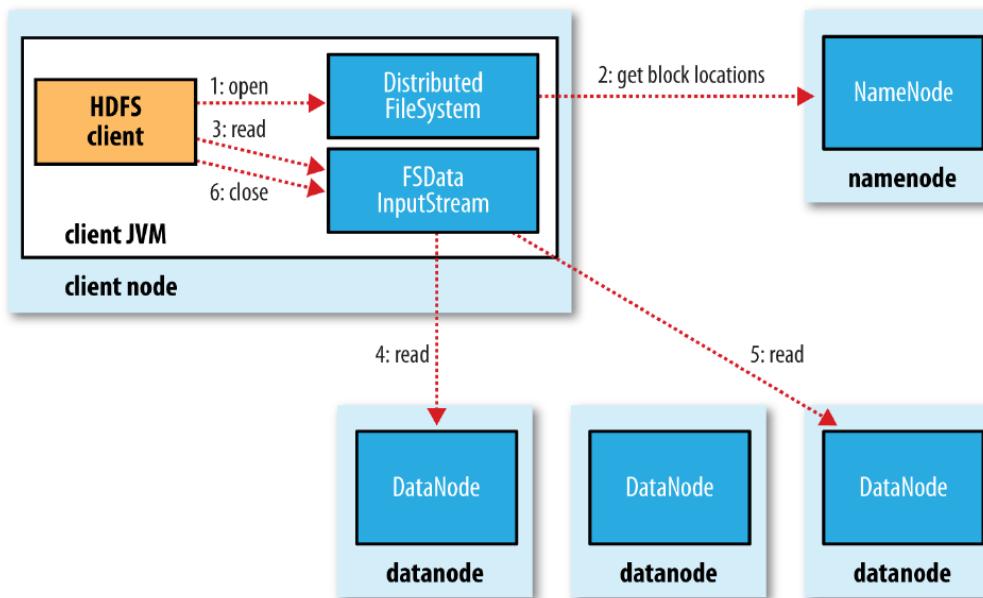
# Hadoop I

CATCHUP

# How files are stored

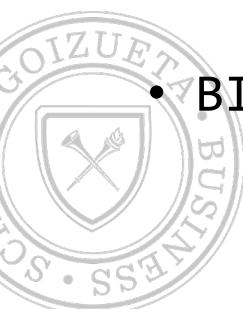


# Data Flow: File Read



# Getting data in and out of HDFS

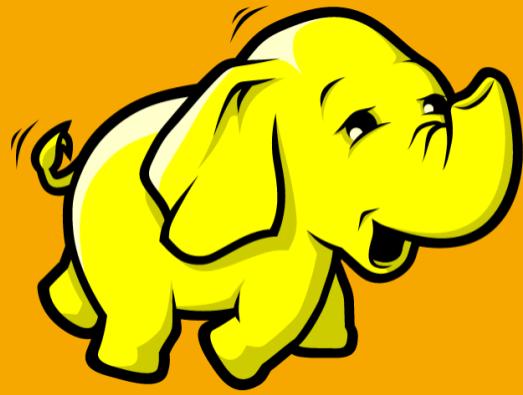
- Hadoop
  - Copies data between client (local) and HDFS (cluster)
  - API or command line
- Ecosystem Projects
  - Flume
    - Collects data from network sources (e.g., websites, system logs)
  - Sqoop
    - Transfers data between HDFS and RDBMS
- BI and ETL tools



# Hadoop Distributed File System (HDFS) Summary

- HDFS is the storage layer for Hadoop
- A filesystem which can store any type of data
- Provides inexpensive and reliable storage for massive amounts of data
  - Data is replicated across computers
- HDFS performs best with a ‘modest’ number of larger files
  - Millions, rather than billions of files
  - Each file is typically 100MB or more
- Files in HDFS are ‘write once’
  - Appends are permitted
  - No random writes are allowed





# Hadoop II

MapReduce & Yarn

# MapReduce

- MapReduce is not a language; it's a programming model.
  - Neither platform- nor language-specific.
  - The programming paradigm has existed as far back as the language LISP.
  - MapReduce code in Hadoop is typically written in Java.
- Programs are written in a functional style of map and reduce tasks
  - which are automatically parallelized and executed on large clusters of commodity hardware.
  - many computations can be expressed as applying a map operation to each logical “record” that produces a set of intermediate key-value pairs and then applying a “reduce” operation to all the values that share the same key.
  - Record-oriented data processing (key and value).
  - Facilitates task distribution across multiple nodes.



# MapReduce

- The runtime system handles many of the messy engineering aspects of parallelization, fault tolerance, data distribution, load balancing, and management of task communication.
- Originally, Hadoop only supported MapReduce as a processing framework.
  - Now, Spark can co-exist with MapReduce.
  - Each framework competes for compute and memory resources on the nodes.
  - YARN (Yet Another Resource Negotiator) was developed to allocate resources to different frameworks based on demand and system administrator settings.

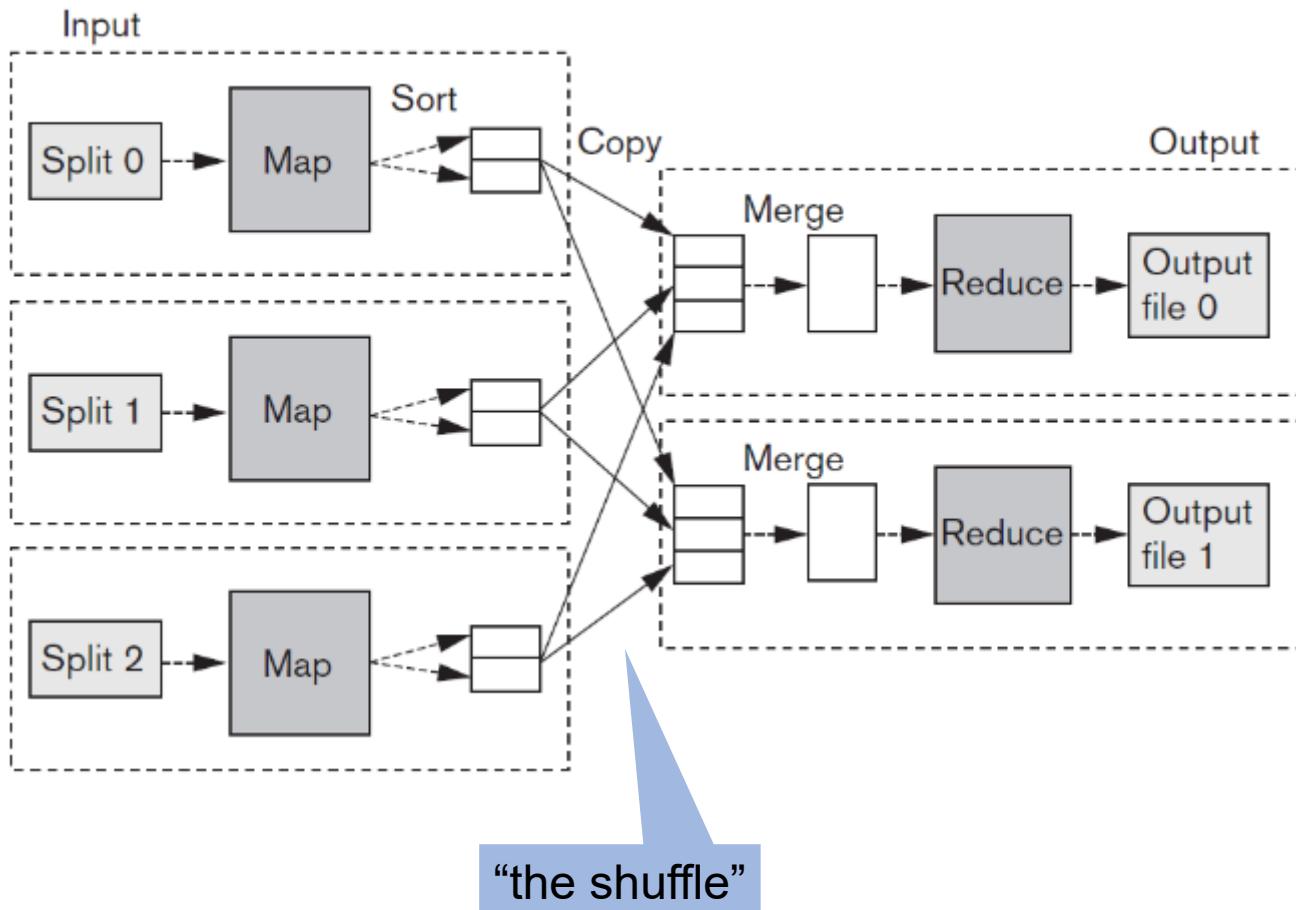


# The MapReduce Programming Model

- The map and reduce functions have the following general form:
  - `map[K1,V1]` which is `(key, value) : List[K2,V2]`
  - `reduce(K2, List[V2]) : List[K3,V3]`
- Map is a generic function that takes a key of type K1 and a value of type V1 and returns a list of key-value pairs of type K2 and V2.
- Reduce is a generic function that takes a key of type K2 and a list of values of type V2 and returns pairs of type (K3,V3).

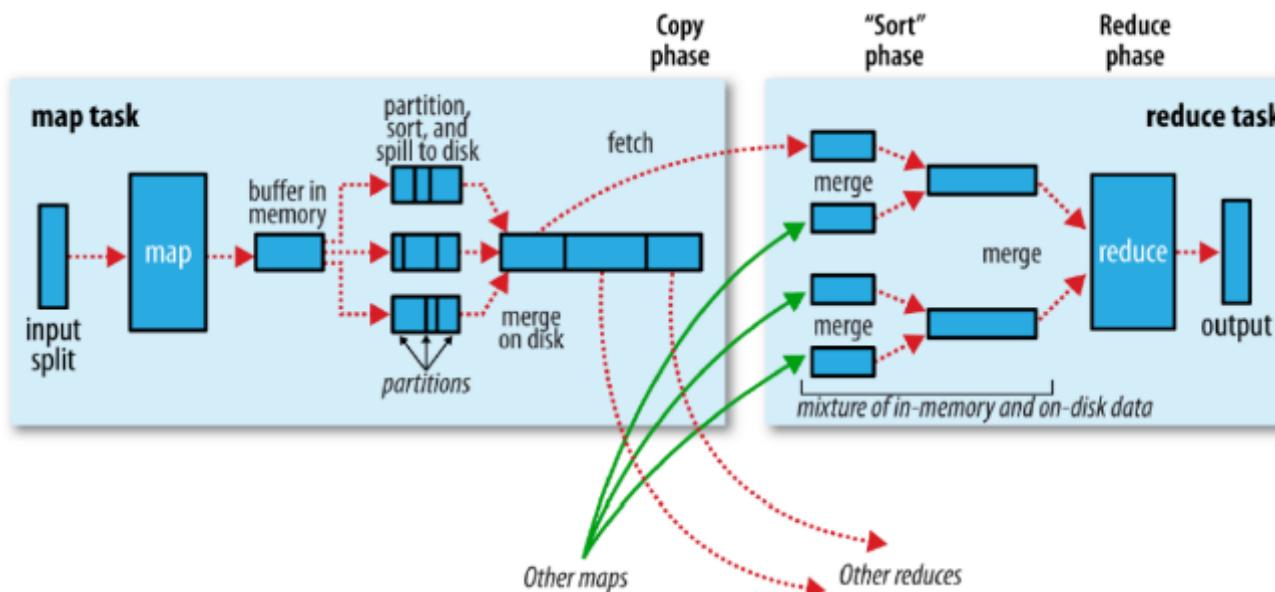


# Overview of MapReduce Execution



# Shuffle and Sort

- MapReduce makes the guarantee that the input to every reducer is sorted by key.
- The process by which the system performs the sort (and transfers the map outputs to the reducers as inputs) is known as the shuffle.

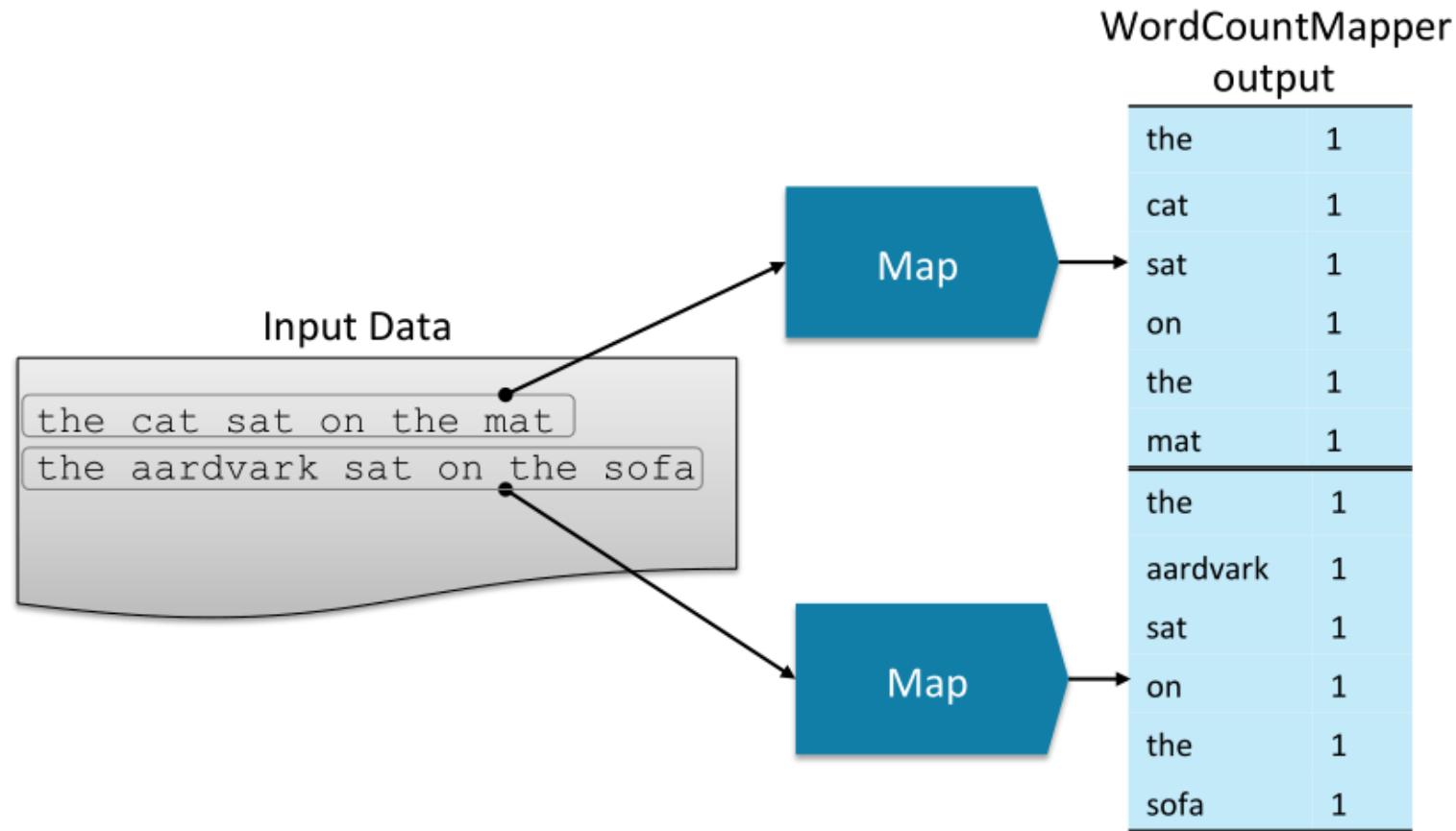


# MapReduce Concepts Summary

- Each Mapper processes a single input split from HDFS
  - Often, a single HDFS block
- Hadoop passes one record at a time to the developer's Mapper code
- Each record has a key and a value
- The Mapper writes intermediate data to the local disk
- During the shuffle and sort phase, all the values associated with the same intermediate key are transferred to the same Reducer
  - The developer specifies the number of Reducers
- Reducer is passed each key and a list of all its values
  - Keys are passed in sorted order
- Output from the Reducers is written to HDFS



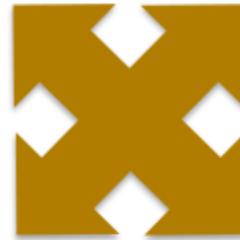
# The MapReduce Programming Model: Example (wordcount)



# The MapReduce Programming Model: Example (wordcount)

Mapper Output

|          |   |
|----------|---|
| the      | 1 |
| cat      | 1 |
| sat      | 1 |
| on       | 1 |
| the      | 1 |
| mat      | 1 |
| the      | 1 |
| aardvark | 1 |
| sat      | 1 |
| on       | 1 |
| the      | 1 |
| sofa     | 1 |

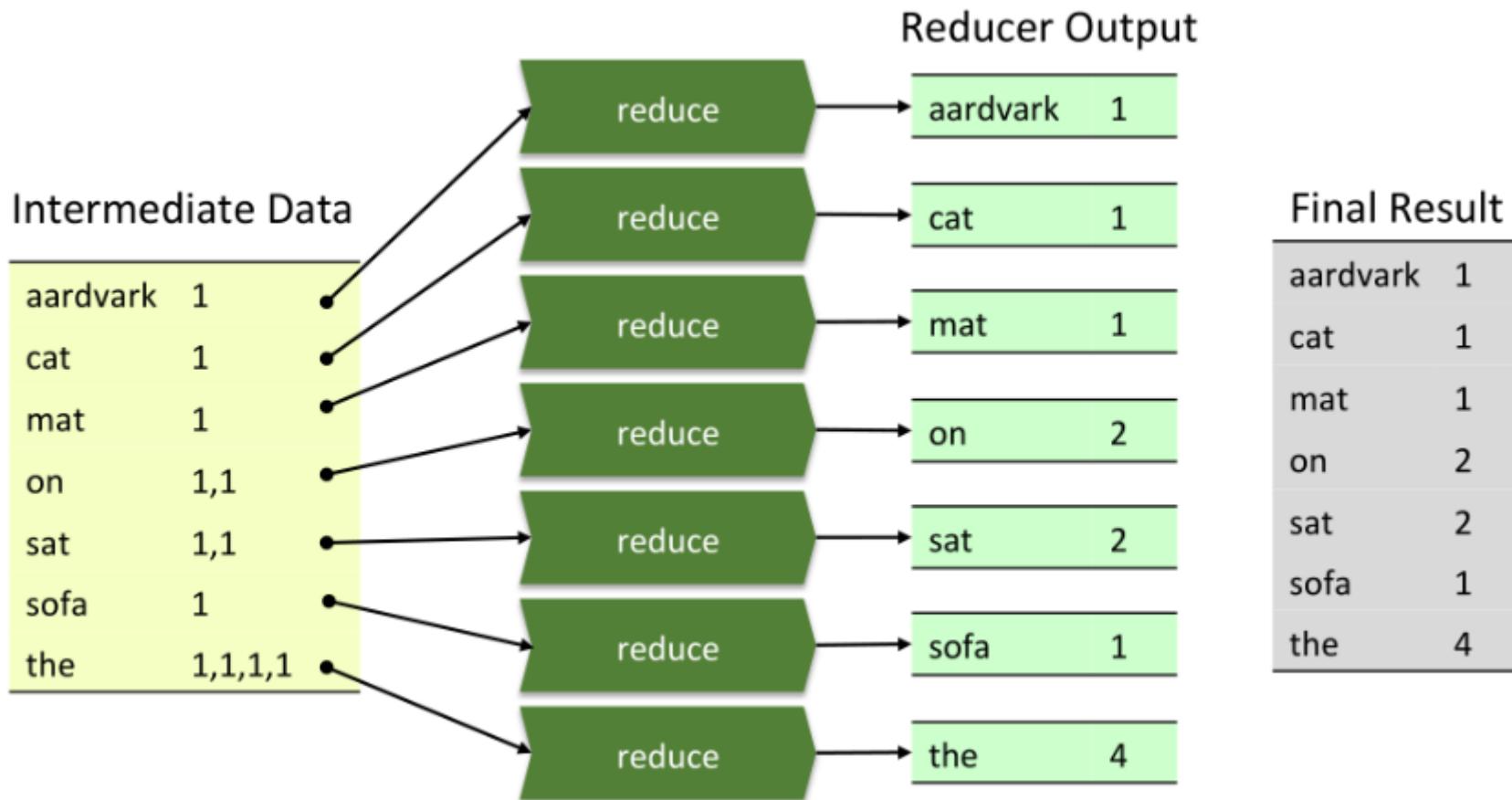


Intermediate Data

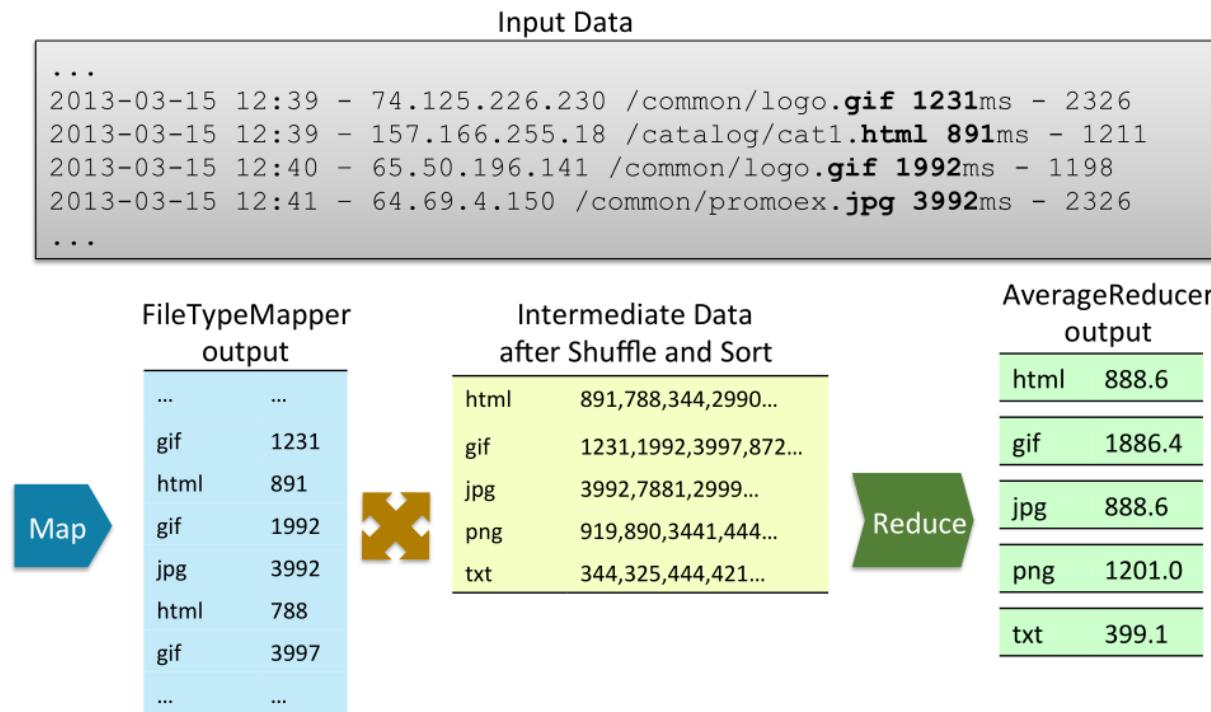
|          |         |
|----------|---------|
| aardvark | 1       |
| cat      | 1       |
| mat      | 1       |
| on       | 1,1     |
| sat      | 1,1     |
| sofa     | 1       |
| the      | 1,1,1,1 |



# The MapReduce Programming Model: Example (wordcount)



# The MapReduce Programming Model: Example (file speed)



# The MapReduce Web UI in Hadoop

- [http://resource-manager-host:8088//](http://resource-manager-host:8088/)

The screenshot shows the Hadoop MapReduce Web UI interface. At the top left is the Hadoop logo. To its right is the title "All Applications". In the top right corner, it says "Logged in as: dr.who". On the far left, there's a sidebar with a tree icon and sections for "Cluster Metrics", "User Metrics for dr.who", and a table showing application details. The main area displays three completed applications:

| ID                             | User     | Name            | Application Type | Queue         | StartTime                     | FinishTime                    | State    | FinalStatus | Progress               | Tracking UI       |
|--------------------------------|----------|-----------------|------------------|---------------|-------------------------------|-------------------------------|----------|-------------|------------------------|-------------------|
| application_1410450250506_0003 | ec2-user | Max temperature | MAPREDUCE        | root.ec2-user | Fri, 12 Sep 2014 10:38:11 GMT | N/A                           | RUNNING  | UNDEFINED   | <input type="button"/> | ApplicationMaster |
| application_1410450250506_0002 | ec2-user | Max temperature | MAPREDUCE        | root.ec2-user | Fri, 12 Sep 2014 10:27:23     | Fri, 12 Sep 2014 10:34:36 GMT | FINISHED | SUCCEEDED   | <input type="button"/> | History           |
| application_1410450250506_0001 | ec2-user | distcp          | MAPREDUCE        | root.ec2-user | Fri, 12 Sep 2014 08:47:09 GMT | Fri, 12 Sep 2014 08:52:56 GMT | FINISHED | SUCCEEDED   | <input type="button"/> | History           |

At the bottom of the main area, it says "Showing 1 to 3 of 3 entries".



# The MapReduce Web UI in Hadoop



Logged in as: dr.who

## MapReduce Job job\_1410450250506\_0003

Job Overview

|           |                              |
|-----------|------------------------------|
| Job Name: | Max temperature              |
| State:    | RUNNING                      |
| Uberized: | false                        |
| Started:  | Fri Sep 12 06:38:24 EDT 2014 |
| Elapsed:  | 6mins, 25sec                 |

ApplicationMaster

| Attempt Number | Start Time                   | Node                            | Logs |
|----------------|------------------------------|---------------------------------|------|
| 1              | Fri Sep 12 06:38:19 EDT 2014 | ip-10-1-1-172.ec2.internal:8042 | logs |

Task Type

| Progress | Total | Pending | Running | Complete |
|----------|-------|---------|---------|----------|
| Map      | 101   | 25      | 14      | 62       |
| Reduce   | 8     | 8       | 0       | 0        |

Attempt Type

| New     | Running | Failed | Killed | Successful |
|---------|---------|--------|--------|------------|
| Maps    | 25      | 14     | 0      | 62         |
| Reduces | 8       | 0      | 0      | 0          |



# MRjob (local)

- Python library: mrjob
  - Mapper()
  - Combiner()
  - Reducer()
- Note: Part of next assignment

```
from mrjob.job import MRJob
class Count(MRJob):
 """ The below mapper() function takes key value
 argument and generates the output in tuple format .
 The mapper below splits text and generates a count
 value for each word i.e. 1 """
 def mapper(self, _, line):
 for word in line.split():
 yield(word, 1)

 """ The below reducer() is aggregates result
 according to key and produces output in a
 key-value format with its total count"""
 def reducer(self, word, counts):
 yield(word, sum(counts))

 """the following lines executes the mrjob"""
if __name__ == '__main__':
 Count.run()
```

```
C:\Users\rgarg28\OneDrive - Emory University\Teaching\py_code_2021\bigdata_mr>python countword.py data.txt
No configs found; falling back on auto-configuration
No configs specified for inline runner
Creating temp directory C:\Users\rgarg28\AppData\Local\Temp\countword.RGARG28.20211020.162533.105751
Running step 1 of 1...
job output is in C:\Users\rgarg28\AppData\Local\Temp\countword.RGARG28.20211020.162533.105751\output
Streaming final output from C:\Users\rgarg28\AppData\Local\Temp\countword.RGARG28.20211020.162533.105751\output...
"map" 2
"not" 2
"or" 3
"reduce" 2
"to" 4
Removing temp directory C:\Users\rgarg28\AppData\Local\Temp\countword.RGARG28.20211020.162533.105751...
```



# MRjob (AWS EMR)

- Create EMR Cluster
- Launch PuTTy
- CLI:
  - pip install mrjob
  - vi mrjob\_word\_count.py (:wq to save and exit)
    - Copy code from: <https://mrjob.readthedocs.io/en/latest/guides/quickstart.html#installation>
  - vi rjp.txt (:wq to save and exit)
    - Copy text from any source
  - python mrjob\_word\_count.py rjp.txt
  - python word\_count.py -r hadoop rjp.txt
  - On browser with proxy: explore the MR jobs (link to the task will show up in your python window)

The screenshot shows the Hadoop MapReduce Job Overview page for job `job_1666153234481_0002`. The job name is `streamjob146753446408940400.jar`, user name is `hadoop`, queue is `default`, state is `SUCCEEDED`, and it was submitted on `Wed Oct 19 04:45:53 UTC 2022`. The job started at `Wed Oct 19 04:46:01 UTC 2022` and finished at `Wed Oct 19 04:46:32 UTC 2022`. The total elapsed time was `31sec`. Diagnostics show average map time of `1sec`, average shuffle time of `6sec`, average merge time of `0sec`, and average reduce time of `0sec`. The ApplicationMaster table lists one attempt starting at `Wed Oct 19 04:45:55 UTC 2022` on node `ip-172-31-42-247.ec2.internal:8042`. Task statistics show 8 maps and 3 reduces attempted, all successful.

| Task Type | Total | Complete |
|-----------|-------|----------|
| Maps      | 8     | 8        |
| Reduces   | 3     | 3        |

| Attempt Type | Maps | Reduces | Failed | Killed | Successful |
|--------------|------|---------|--------|--------|------------|
| Maps         | 0    | 0       | 0      | 0      | 8          |
| Reduces      | 0    | 0       | 0      | 0      | 3          |

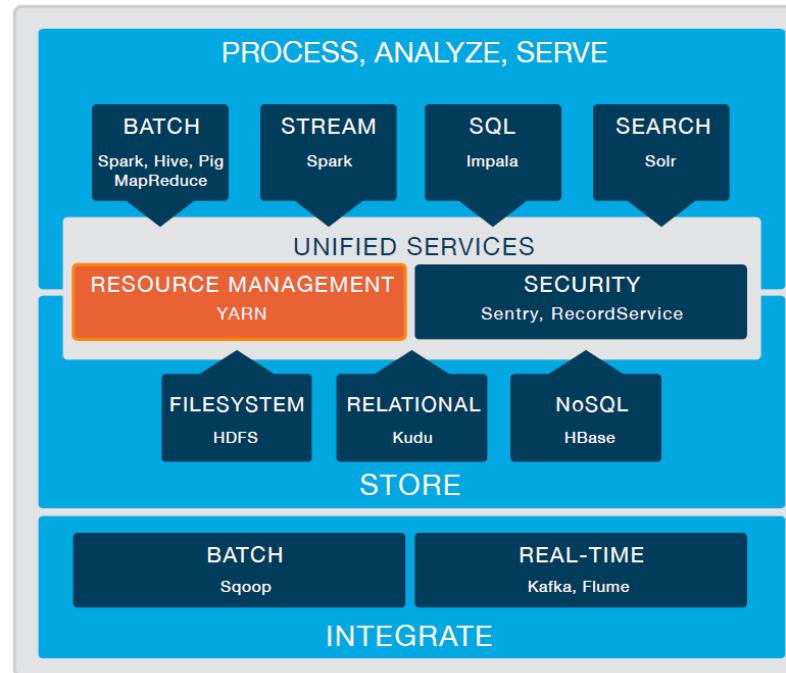


# YARN



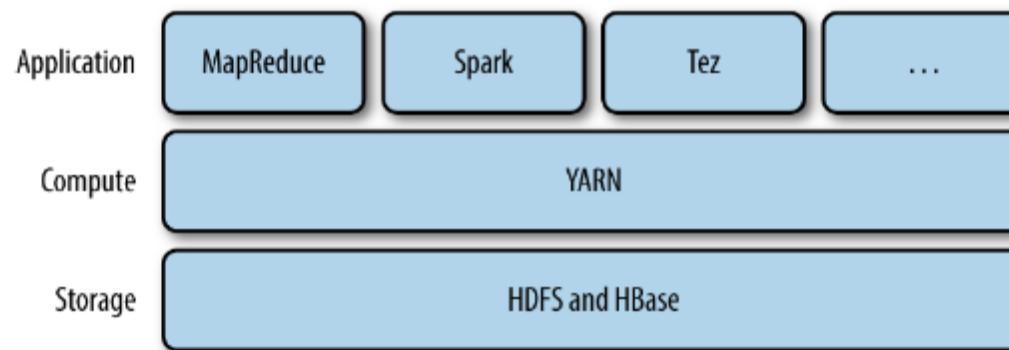
# Workload Management: YARN

- Many Hadoop tools work with data in a Hadoop cluster
- Distributing and monitoring work across the cluster requires workload management



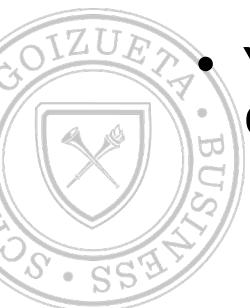
# YARN

- Apache YARN (Yet Another Resource Negotiator) is Hadoop's cluster resource management system.
- YARN was introduced in Hadoop 2 to improve the MapReduce implementation,
  - but it is general enough to support other distributed computing paradigms as well.
- Three schedulers are available in YARN: the FIFO, Capacity, and Fair Schedulers.



# Why YARN?

- YARN allows us to run diverse workloads on the same Hadoop cluster
  - Jobs using different frameworks will probably have different resource profiles
- Examples:
  - A MapReduce job or an Impala query that scans a large table
    - Likely heavily disk-bound
    - Requires little memory
  - A Spark job executing an iterative machine learning algorithm
    - Will probably attempt to store the entire dataset in memory
    - May use spurts of CPU to perform complex computations
- YARN allows you to share cluster memory and CPU resources dynamically between processing frameworks
  - MapReduce, Impala, Spark, and others



# Cloud Deployment

- Hadoop can be deployed on the cloud
  - Amazon Web Services, Microsoft Azure, etc.
- Benefits of deploying Hadoop in the cloud include:
  - Flexible deployment
    - Bypass prolonged infrastructure selection/procurement process
  - Fast ramp-up/ramp-down
    - Easily alter the cluster size based on the current workload
  - Cost savings
    - Deploying in the cloud eliminates the need for dedicated, on-premise computing resources
      - A cost comparison is required to examine the best solution



# Capacity planning

- Basing your cluster growth on storage capacity is often a good method to use
- Example:
  - Data grows by approximately 3TB per week/40TB per quarter
  - Hadoop replicates 3 times = 120TB
  - Extra space required for temporary data while running jobs (~30%)=160TB
  - Assuming machines with 12 x 3TB hard drives
    - 4-5 new machines per quarter
    - Two years of data = 1.3PB requires approximately 36 machines



# Why do you need Hadoop?

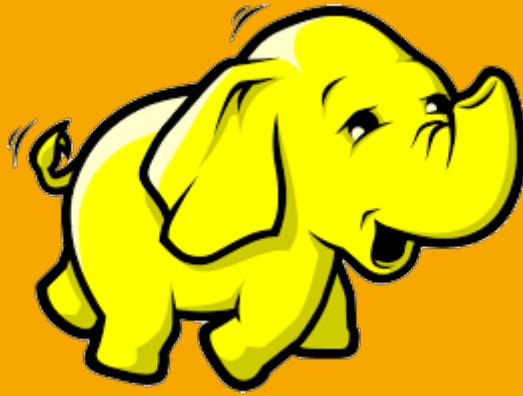
- More data is coming
  - IoT, sensor data, streaming
- More data means bigger questions
- More data means better answers
- Hadoop easily scales to store and handle all of your data
- Hadoop is cost effective
  - Typically provides a significant cost-per-terabyte saving over traditional legacy systems
- Hadoop integrates with your existing datacenter components



# Try

- EMR Tutorial
  - <https://www.youtube.com/watch?v=jylp2atrZjc>





# Hadoop III

Hadoop Ecosystem

# The Hadoop Ecosystem

- Many tools integrate with Hadoop for:
  - Data analysis
  - Database integration
  - Workflow management
  - ..
- These are part of the ‘Hadoop ecosystem’
  - Many are also open-source Apache projects
  - Examples: Pig, Hive, Impala, HBase, Flume, Sqoop, Oozie, Mahout, ..



# The Hadoop Ecosystem

- Hadoop ecosystem has a set of related projects that provide additional functionality on top of HDFS and MapReduce.
- HBase: A column-oriented key-value store that uses HDFS as its underlying store. HBase is patterned after BigTable from Google but is implemented using Hadoop and HDFS.
- Pig: Provides a dataflow language. A script written in PigScript translates into a directed acyclic graph (DAG) of MR jobs.
- Hive: Provides an SQL interface on top of MapReduce.
- Oozie: A service for scheduling and running workflows of Jobs; individual steps can be Hive queries, Pig scripts, etc.
- Sqoop: A library and a runtime environment for efficiently moving data between relational databases and HDFS.

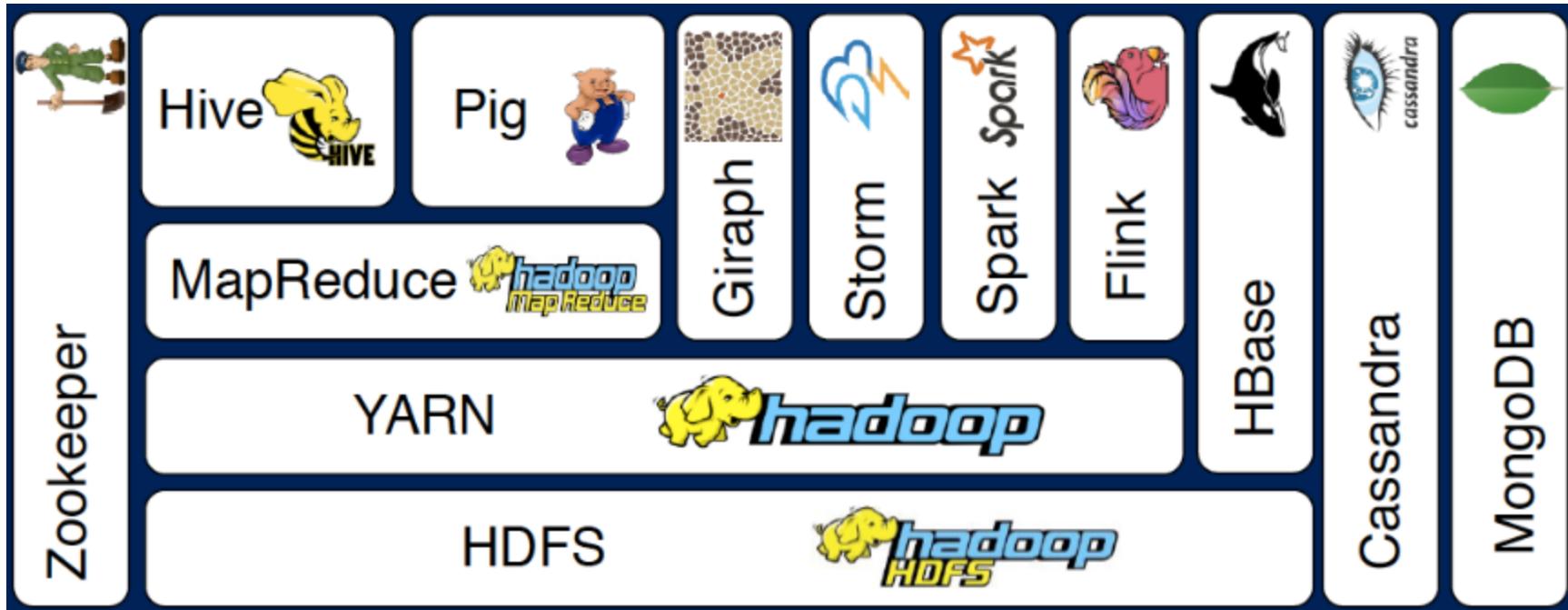


# The Hadoop Ecosystem

- **Spark**: In-memory and stream processing framework
- **Impala**: SQL query engine designed for BI workloads
- **Parquet**: Very efficient columnar data storage format
- **Flume**, **Kafka**: Streaming data ingestion
- **Solr**: Powerful text search functionality
- **Hue**: Web-based user interface for Hadoop
- **Sentry**: Authorization tool, providing security for Hadoop
- **Falcon**: Data governance engine for data management
- **Giraph**, **Storm**, etc.



# Apache Hadoop Ecosystem



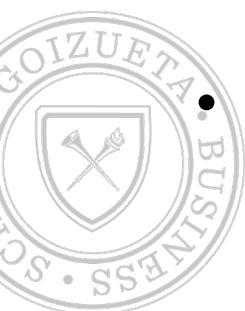
# Data Storage

The Hadoop Ecosystem

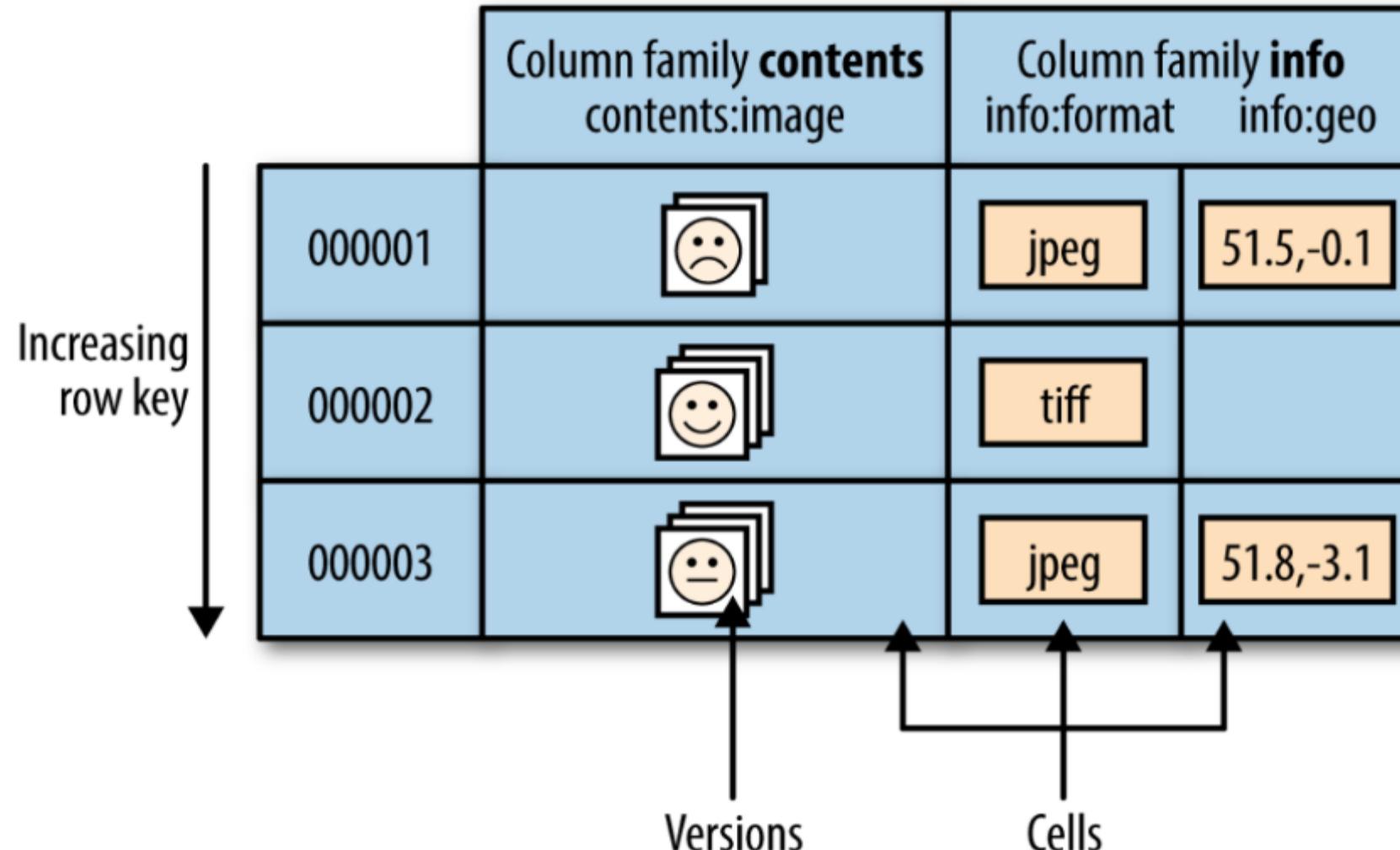


# Apache HBase

- HBase is a NoSQL column-oriented distributed database.
- HBase is the Hadoop application to use when you require real-time read/write random access to very large datasets.
- Stores data in HDFS.
- HBase can scale (almost) linearly just by adding nodes.
  - It scales to support very high throughput for both reads and writes (e.g., millions of inserts or updates per second).
- A table can have many thousands of columns.
  - Handles sparse data well.
- Designed to store very large amounts of data
  - Petabytes+



# The HBase data model



# The HBase data model

- Many concepts are familiar, but use different terminology in HBase
  - The intersection of a row and column is called a cell
  - The primary key is called a row key and is required
- HBase stores data differently than a relational database
  - Column families group columns for separate storage and access
  - Columns within a family do not have to be declared in advance

| Row Key | contactinfo |       |         |          |          | annualbudget |          |        |
|---------|-------------|-------|---------|----------|----------|--------------|----------|--------|
|         | fname       | mname | lname   | mobile   | home     | movies       | concerts | travel |
| aj8462  | Alice       | Jane  | Ames    | 555-2182 | 555-3714 | 300          | 100      | 5000   |
| bb2988  | Bob         |       |         |          |          | 500          |          |        |
| cs8701  | Carol       | Sue   | Clemens | 555-2642 |          |              | 500      |        |
| dd7235  | Dan         |       | Davis   |          | 555-3421 |              | 375      | 1000   |
| ee5361  | Eve         | A.    | Ellis   | 555-3960 |          | 100          |          | 2500   |



# Data Ingestion

The Hadoop Ecosystem



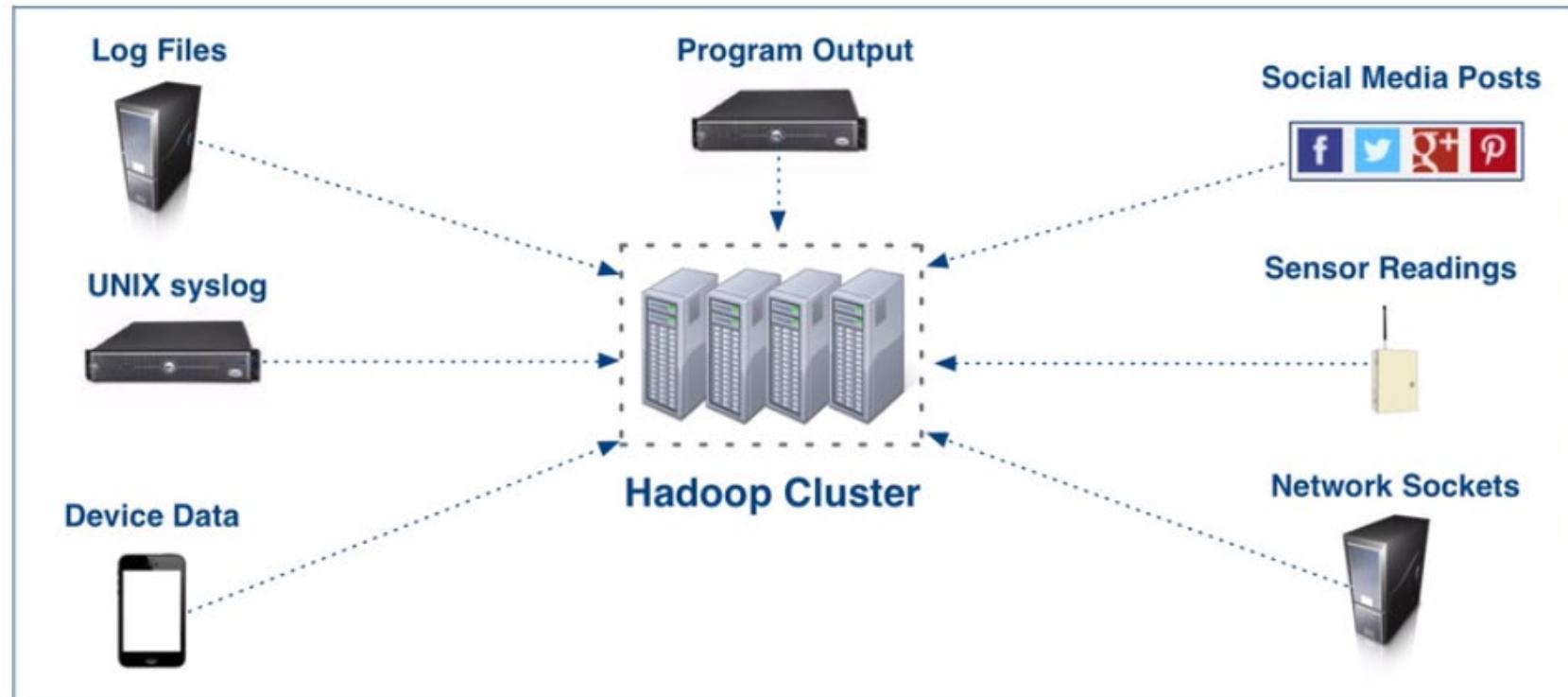


# Apache Flume

- Apache Flume is designed for high-volume event-based data ingestion into Hadoop.
  - e.g., using Flume to move log events from (remote) log files into new aggregated files in HDFS.
  - Now widely used for collection of any type of event data
  - Supports aggregating from many sources into HDFS
- Flume is
  - Distributed
  - Reliable and available
  - Horizontally scalable
  - Extensible



# Apache Flume





# Apache Flume

- To use Flume, we need to run a Flume agent, which is a long-lived Java process that runs sources and sinks, connected by channels.
  - A source in Flume produces events and delivers them to the channel, which stores the events until they are forwarded to the sink.
  - Flume uses separate transactions to guarantee delivery from the source to the channel and from the channel to the sink.
  - Using Solr (or Elasticsearch) as search, near-real-time indexing can be achieved.

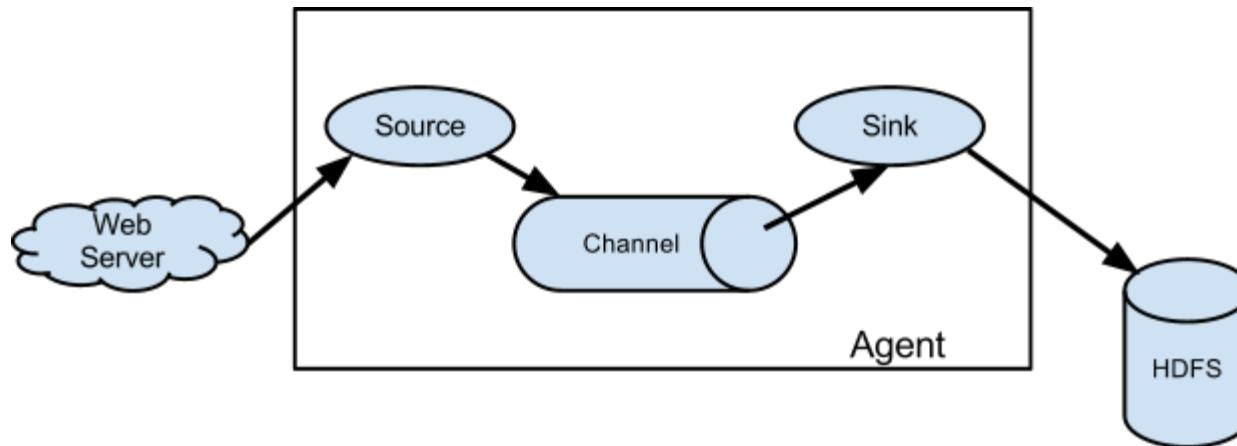


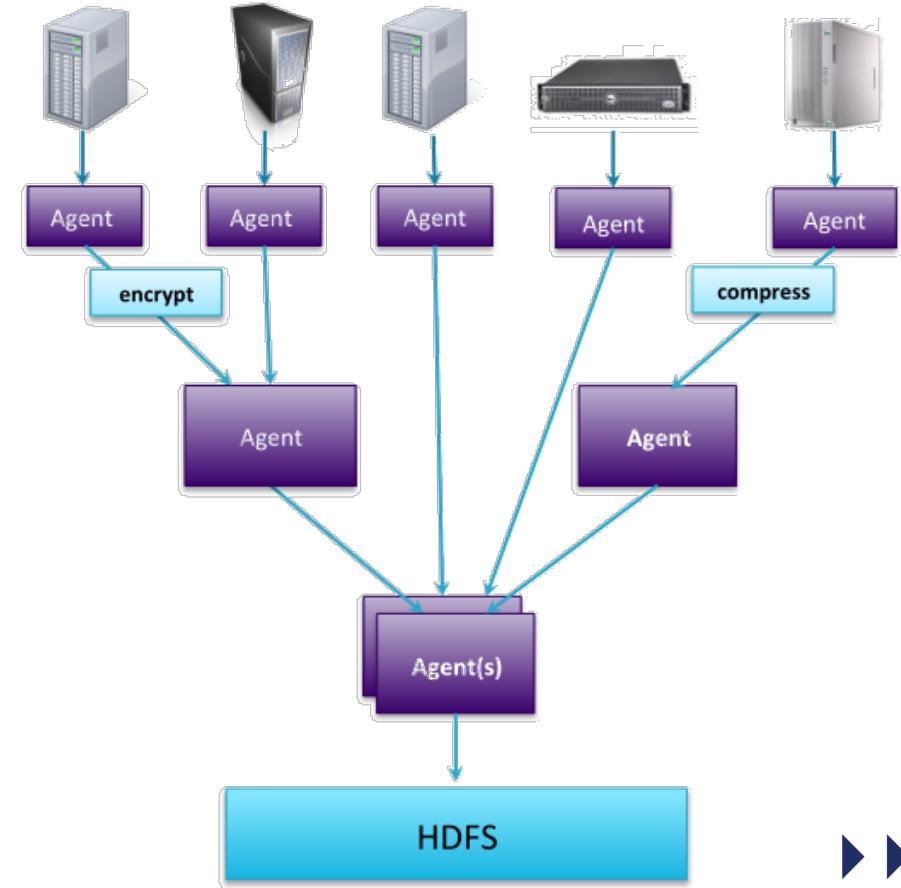
Image source: <https://flume.apache.org/FlumeUserGuide.html>



# Apache Flume: Pipeline Example



- Collect data as it is produced
  - Files, syslogs, stdout or custom source
- Process in place
  - e.g., encrypt, compress
- Pre-process data before storing
  - e.g., transform, scrub, enrich
- Write in parallel
  - Scalable throughput
- Store in any format
  - Text, compressed, binary, or custom sink



# Apache Kafka

- Both Flume and Kafka are tools for ingesting event data into Hadoop as that data is being generated
  - Log files, sensor data, streaming data from social media, etc.
  - They are ideal for aggregating event data from many sources in a centralized locations, such as HDFS.
  - Also, vital for applications such as fraud prevention, threat detection, etc.
- Flume is typically easier to configure, but Kafka provides more functionality
  - Flume generally provides a path from a data source to HDFS or to a streaming framework such as Spark
  - Kafka uses a 'Publish/Subscribe' model allowing data to be consumed by many different systems, including writing to HDFS



# Characteristics of Kafka

- Scalable
  - Kafka is a distributed system that supports multiple nodes
- Fault-tolerant
  - Data is persisted to disk and can be replicated throughout the cluster
- High throughput
  - Each broker can process hundreds of thousands of messages per second
- Low latency
  - Data is delivered in a fraction of a second
- Flexible
  - Decouples the production of data from its consumption





# Apache Sqoop

- Sqoop (SQL to Hadoop) rapidly moves data between RDBMSs and HDFS
  - Import tables (or partial tables) from an RDBMS into HDFS
  - Export data from HDFS to a database table
- The processing can be done with MapReduce programs or other higher-level tools such as Hive.
- Example command:
  - `sqoop import --connect jdbc:mysql://localhost/databasename --table tablename -m 1`
- It's also possible to use Sqoop to move data from a database into Hbase.





# Apache Sqoop

- Supports JDBC, ODBC, and several specific databases
  - Custom connectors for MySQL, Postgres, Teradata, Oracle, etc.
- Recent versions of Sqoop support Avro-based serialization and schema generation as well,
  - allowing you to use Sqoop in your project without integrating with generated code.
- Not open source, but free to use
- Further reading:
  - Apache Sqoop Cookbook by Kathleen Ting and Jarek Jarcec Cecho (O'Reilly, 2013).





# HUE User Interface

The Hadoop Ecosystem



# Apache Hue

- Hue (Hadoop User Experience) provides a Web front-end to a Hadoop
  - Upload data
  - Browse data
  - Query table in Impala and Hive
  - Search
  - ..
- Provides access control for the cluster by requiring users to log in before they can use the system
- Overall, Hue makes Hadoop easier to use
- Try:
  - <https://demo.gethue.com/>
  - Or on AWS EMR instance



# Possible Hue Error

- sudo vim /etc/hue/conf/hue.ini
- Search (type): /webhdfs\_url (press ENTER key)
- Press INSERT key and change port 14000 to 9870 (press ESC key)
- Save and quit: “:wq”
- sudo systemctl status hue
- sudo systemctl stop hue
- sudo systemctl start hue



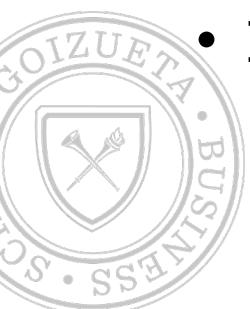
# Data Processing & Analysis

The Hadoop Ecosystem



# Hive and Pig: High Level Data Languages

- The motivation: MapReduce is powerful but hard to master
- The solution: **Hive** and **Pig**
  - Languages for querying and manipulating data
  - Leverage existing skillsets
    - Data analysts who use SQL
    - Programmers who use scripting languages
  - Open-source Apache projects
    - Hive initially developed at Facebook
    - Pig Initially developed at Yahoo!
- Interpreter runs on a client machine
  - Turns queries into MapReduce jobs
  - Submits jobs to the cluster



# Apache Pig



- Apache Pig raises the level of abstraction for processing large datasets.
- Pig was designed to bridge the gap between declarative-style interfaces, such as SQL, and low-level procedural-style programming style required by MapReduce.
  - With Pig, the data structures are much richer, typically being multivalued and nested, and the transformations you can apply to the data are much more powerful, including joins.
- Pig is made up of two pieces:
  - The language used to express data flows, called Pig Latin.
  - The execution environment to run Pig Latin programs.
- A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input.
  - Under the covers, Pig turns the transformations into a series of MR jobs





# Pig Latin Example

- Find the category with high pagerank pages:
- SQL example:

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10**6
```

- Equivalent Pig Latin example:

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)> 10**6;
output = FOREACH big_groups GENERATE category, AVG(good_urls.pagerank);
DUMP output;
```



# Apache Pig vs Databases

- Pig Latin is a data flow programming language, whereas SQL is a declarative programming language.
- RDBMSs operate with tightly predefined schemas. Pig lets you define a schema at runtime.
- Pig supports complex, nested data structures. SQL operates on flatter data structures.
- RDBMSs have several features (e.g., indexes) to support online, low-latency queries that are absent in Pig.
- Hive sits between Pig and conventional RDBMSs.





# Apache Hive

- Hive was developed at Facebook with an intent to:
  - provide a high-level interface to Hadoop using SQL-like queries and support the processing of aggregate analytical queries (data warehousing).
- Hive went beyond Pig Latin providing a layer that makes Hadoop look like a RDBMS with DDL, metadata repository, JDBC/ODBC access, and an SQL compiler.
- The Hive query language HiveQL includes a subset of SQL that includes all types of joins, Group By operations, as well as useful functions related to primitive and complex data types.





# Apache Hive

- In normal use, Hive runs on your workstation and converts your SQL query into a series of jobs for execution on a Hadoop cluster.
  - The Hive interpreter uses MapReduce, Spark, or Tez to process the data.
- Hive organizes data into tables, which provide a means for attaching structure to data stored in HDFS.
- Metadata, such as table schemas, is stored in a database called the metastore.
- The shell is the primary way that we will interact with Hive, by issuing commands in HiveQL. ▶▶▶



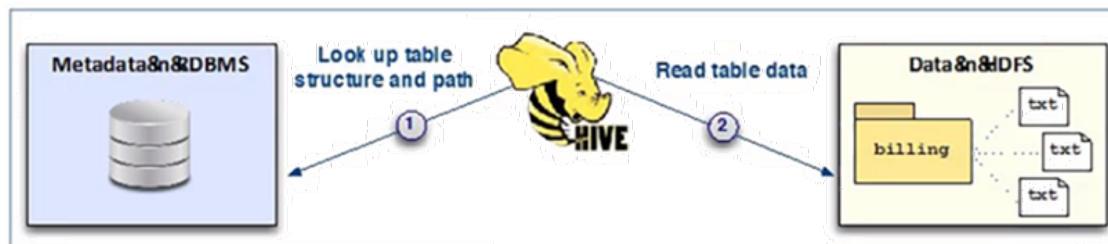


# How Hive Works

- A Hive table is created in HDFS with associated metadata
- Tables are created by describing pre-existing data in HDFS

```
CREATE TABLE products (
 id INT,
 name STRING,
 price INT
);
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

- Metadata (table structure and path to data) is stored in an RDBMS





# Why should I use Hive?

- Data can be loaded before the table is defined
  - Schema-on-Read
  - You do not need to know the data's structure prior to loading it
- Does not require a developer who knows Java, Scala, Python or other traditional programming languages
  - Anyone who knows SQL can process and analyze data on the cluster
- Well suited for dealing with structured data, or data which can have a structure applied to it



# Hive is NOT an RDBMS

- Note that Hive is not an RDBMS
  - Results take several seconds, minutes, or even hours to be produced
  - Not possible to modify the data using HiveQL
    - UPDATE and DELETE are not supported



# Impala



- Impala is a high-performance SQL engine
  - Runs on Hadoop clusters
  - Does not rely on MapReduce
  - Massively-parallel processing (MPP)
  - Inspired by Google's Dremel project
  - Very low latency - typically measured in milliseconds
  - Faster than Hive
- Impala supports a dialect of SQL very similar to Hive's
- Impala was developed by Cloudera
  - 100% open source, released under the Apache software license
- Other prominent open-source alternatives include Presto from Facebook, Apache Drill, and Spark SQL.



# What is Impala?



- Like Hive, Impala allows users to query data in HDFS using an SQL-like language
- Unlike Hive, Impala does not turn queries into MapReduce jobs
  - Impala queries run on an additional set of daemons that run on the Hadoop cluster
    - Often referred to as 'Impala Servers'
  - Impala queries run significantly faster than Hive queries
    - Tests show improvements of 10x to 50x or more
  - Impala also supports many simultaneous users much more efficiently than Hive
- Impala uses the same shared Metastore that Hive uses
  - Tables created in Hive are visible in Impala (and vice versa)



# Impala



- Impala bypasses MapReduce and directly runs queries on the datanodes.
  - Thus, YARN doesn't work for Impala
  - Long Lived Application Master (Llama) provides scheduling and resource negotiation interface with YARN.

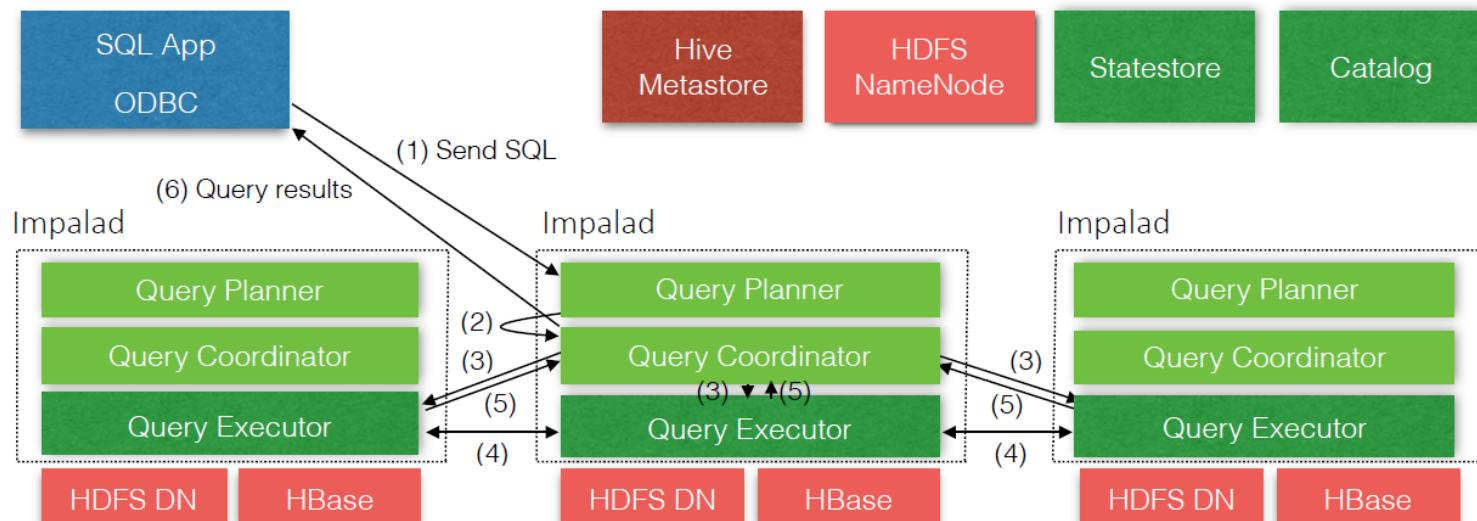


Figure 1: Impala is a distributed query processing system for the Hadoop ecosystem. This figure also shows the flow during query processing.

Source: [http://cidrdb.org/cidr2015/Papers/CIDR15\\_Paper28.pdf](http://cidrdb.org/cidr2015/Papers/CIDR15_Paper28.pdf)



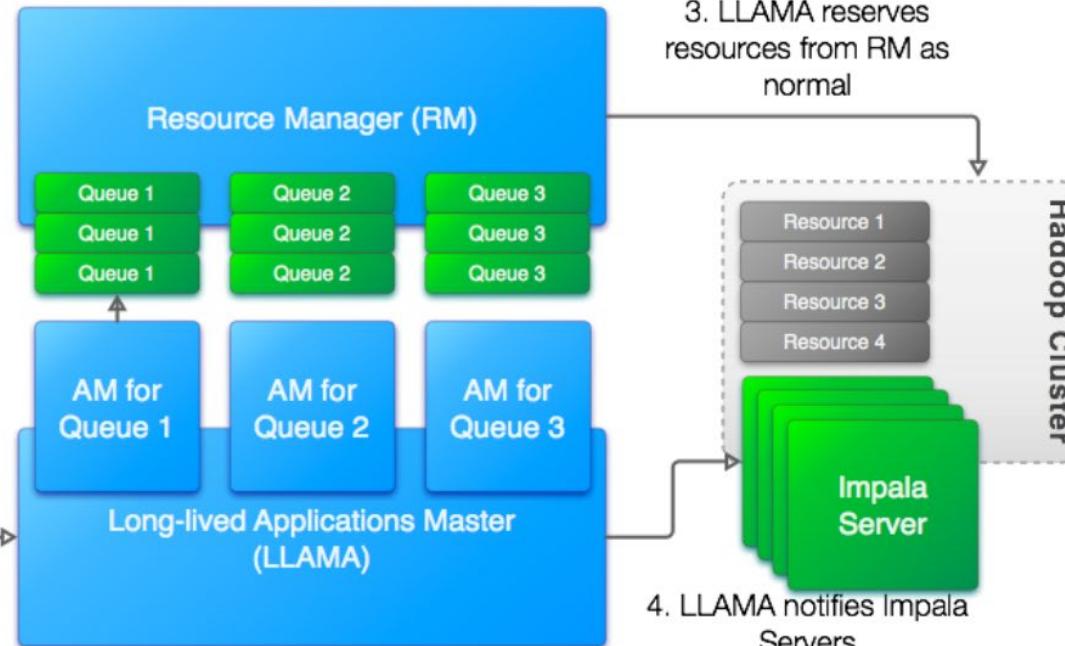
# Impala



## Llama's role in Impala resource acquisition

1. LLAMA registers with RM at start time, before any queries are issued, creating one AM per queue

2. Impala Server asks LLAMA for resources for a query



Source: @mattjacobs (Cloudera)



# Data Processing

The Hadoop Ecosystem



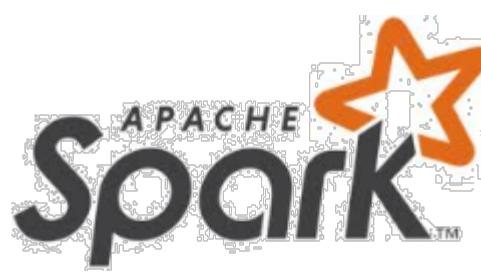


# Apache Spark

- Apache Spark is a large-scale data processing engine.
- Spark does not use MapReduce as an execution engine; instead, it uses its own distributed runtime.
- Supports a wide range of workloads:
  - Machine learning, interactive analysis, batch applications, iterative algorithms, business intelligence, etc.
- Spark is best known for its ability to keep large working datasets in memory between jobs.
- Spark Streaming provides the ability to process data as that data is being generated.
  - Typically in conjunction with Flume or Kafka.



# Apache Spark



- The Apache Spark project includes modules for:
  - machine learning ([MLlib](#)),
  - graph processing ([GraphX](#)),
  - stream processing ([Spark Streaming](#)), and
  - SQL ([Spark SQL](#)).
- A Spark job is made up of an arbitrary directed acyclic graph (DAG) of stages which are split into tasks by the Spark runtime and are executed in parallel across the cluster.



# Apache Spark and other frameworks

- Different engines have different characteristics.
- Spark is more efficient than MapReduce if there is a lot of intermediate data to be passed between jobs, because:
  - It doesn't store data to disks
  - It can rebuild data in a cluster by using DAG tracking of the workflows – thus achieving fault tolerance
- Spark code can be written in Python, Scala, or Java.
- Spark is well-suited to iterative processing algorithms as well as interactive analysis
  - e.g., several machine learning applications.

Spark Streaming provides real-time data processing features.



# Data Exploration

The Hadoop Ecosystem



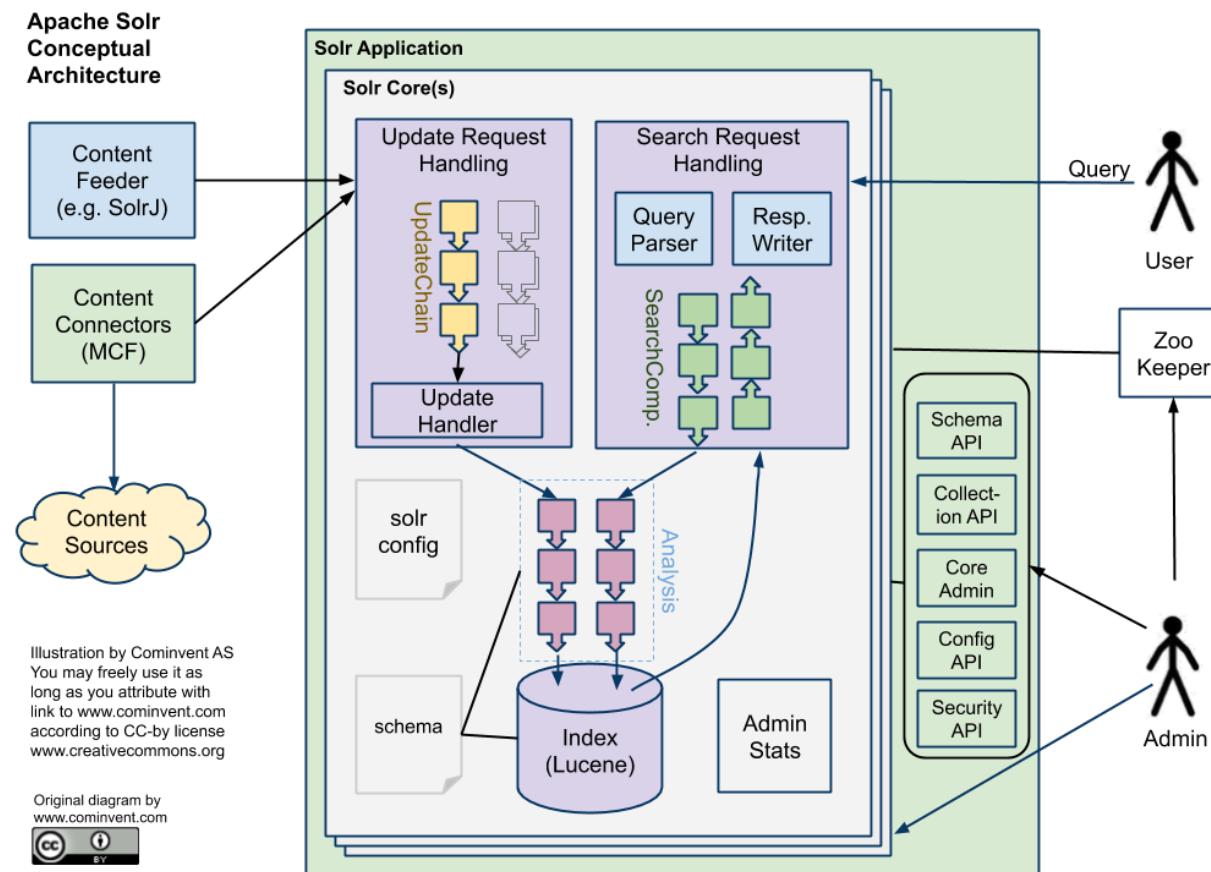


# Cloudera Search and Apache Solr

- Search is an interactive full-text search for data in your Hadoop cluster
- Supports real-time and batch indexing
- Allows non-technical users to access your data
- Cloudera Search (open-source) enhanced [Apache Solr](#)
  - Integrations with HDFS, MapReduce, Hbase, Flume, Kafka, ..
  - Support for file formats widely used with Hadoop
  - Dynamic Web-based dashboard Search interface with Hue
  - Apache Sentry based security



# Apache Solr



# Cloudera Search: Why Should I Use It?

- Databases are often used to analyze data
  - Search is typically used to discover data
- Databases are designed to join tables based on a key
  - Search is intended for queries on denormalized (flat) data sets
- Databases are optimized to find and sort by specific values
  - Search can match based on specific values, term variants, or ranges
  - Search results are usually sorted by relevance
- As with a database, Cloudera Search is primarily a back-end tool
  - End users usually interact with it through user interfaces you create
  - APIs are available for application development in multiple languages



# Data Security

The Hadoop Ecosystem



# Apache Sentry



- Apache Sentry is a plugin that provides fine-grained access control (authorization) to various Hadoop ecosystem components
  - Impala
  - Hive
  - Cloudera Search
  - The HDFS command line
- In conjunction with Kerberos authentication, Sentry authorization provides a complete cluster security solution



# Apache Sentry: Why Should I Use It?



- Hadoop has long supported Kerberos authentication
  - “Prove you are who you say you are”
- Typically, a production cluster also requires authorization controls
  - “This person is allowed to do only these things”
  - This is especially true for clusters in regulated environments such as financial services, healthcare, etc.
- Sentry allows an administrator to grant fine-grained access rights to individuals
  - For example, permission to view only a certain column in a given Hive table
- Sentry is a key component of a secure Hadoop cluster



# Workflow Management

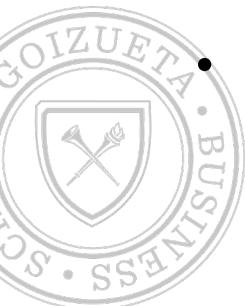
The Hadoop Ecosystem





# Apache Oozie

- Apache Oozie is a system for running workflows of dependent jobs. It is composed of two main parts:
  - a workflow engine that stores and runs workflows composed of different types of Hadoop jobs (MapReduce, Pig, Hive, and so on), and
  - a coordinator engine that runs workflow jobs based on predefined schedules and data availability.
    - A workflow is a DAG of action nodes and control-flow nodes.
- Oozie has been designed to scale, and it can manage the timely execution of thousands of workflows, each composed of possibly dozens of constituent jobs.
- Example:
  - `oozie job -config max-temp-workflow.properties -run`



# Other Tools

The Hadoop Ecosystem





# Apache Mahout

- Mahout is a Machine Learning library written in Java
- Used for
  - Collaborative filtering (recommendations)
  - Clustering (finding naturally occurring “groupings” in data)
  - Classification (determining whether new data fits a category)
- Why should you use Hadoop for Machine Learning?
  - “It’s not who has the best algorithms that wins. It’s who has the most data.”



Note: Mahout is not available in EMR 6.x

- <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-600-release.html>





# Apache Avro

- Apache Avro is a language-neutral data serialization system.
- The project was created by Doug Cutting (the creator of Hadoop) to address the major downside of Hadoop Writables: lack of language portability.
  - Having a data format that can be processed by many languages (currently C, C++, C#, Java, JavaScript, Perl, PHP, Python, and Ruby) makes it easier to share datasets with a wider audience.





# Apache Avro

- Avro data is described using a language-independent schema.
  - You can read and write data that conforms to a given schema even if your code has not seen that particular schema before.
- Avro schemas are usually written in JSON, and data is usually encoded using a binary format.
- Avro specifies an object container format for sequences of objects, similar to Hadoop's sequence file.
- An Avro datafile has a metadata section where the schema is stored, which makes the file self-describing.
- Avro datafiles support compression and are splittable, which is crucial for a MapReduce, Pig, Hive, Spark, etc.



# Apache Parquet

- Apache Parquet is a columnar storage format that can efficiently store nested data.
- Columnar formats are attractive since they enable greater efficiency, in terms of both file size and query performance.
- A key strength of Parquet is its ability to store data that has a deeply nested structure in true columnar fashion.
  - Schemas with several levels of nesting are common in real-world systems.
  - The technique was introduced by Google in Dremel; Parquet was created by engineers at Twitter and Cloudera.
- Most of the time Parquet files are processed using higher-level tools like Pig, Hive, or Impala.



# Data Format Summary

| Feature                   | Text | Sequence Files | Avro | Parquet |
|---------------------------|------|----------------|------|---------|
| Supported by many tools   | ✓    |                | ✓    | ✓       |
| Good performance at scale |      | ✓              | ✓    | ✓       |
| Binary format             |      | ✓              | ✓    | ✓       |
| Embedded schema           |      |                | ✓    | ✓       |
| Columnar organization     |      |                |      | ✓       |



# Apache ZooKeeper



- Apache ZooKeeper is a highly available, high-performance coordination service.
- ZooKeeper gives you a set of tools to build distributed applications that can safely handle partial failures.
  - Partial failures: when we don't even know if an operation failed.
- ZooKeeper is, at its core, a stripped-down filesystem that exposes a few simple operations and some extra abstractions, such as ordering and notifications.
  - It doesn't have files and directories, but a unified concept of a node, called a znode, that acts both as a container of data (like a file) and a container of other znodes (like a directory).

• More information: <http://zookeeper.apache.org/doc/r3.7.0/zookeeperOver.html>



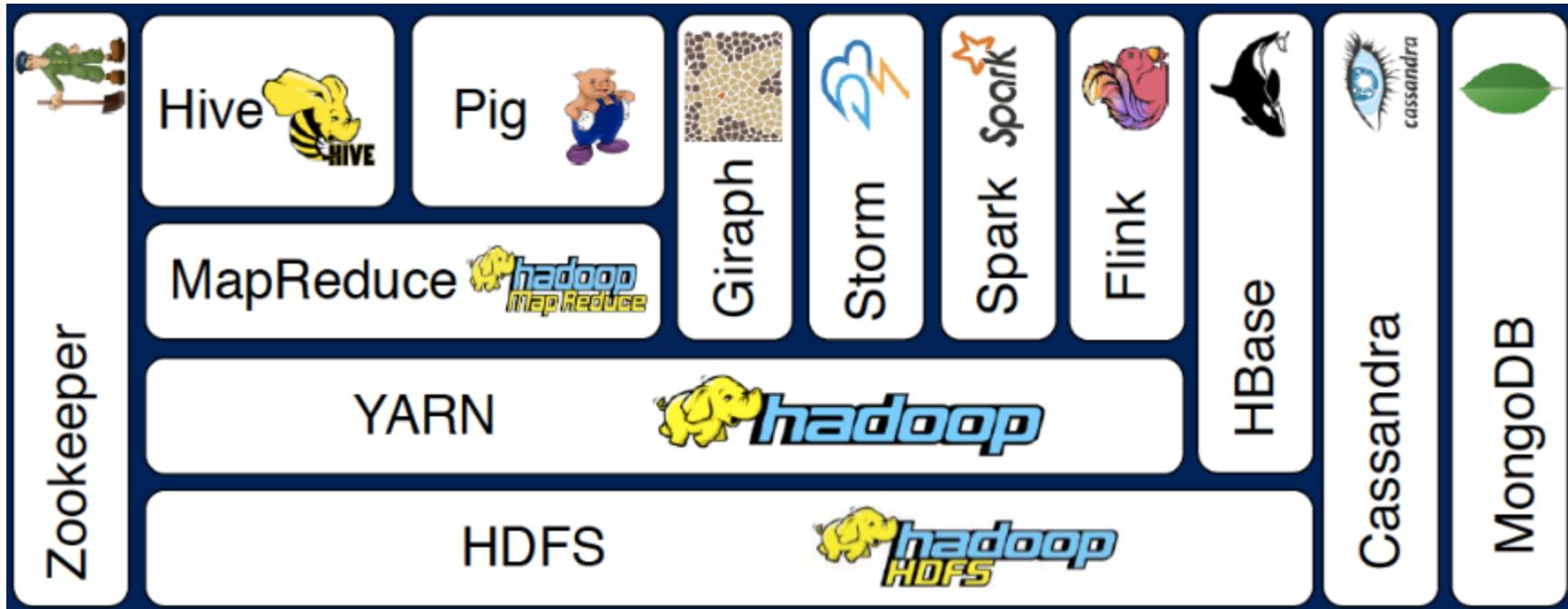
# Apache ZooKeeper



- The ZooKeeper primitives are a rich set of building blocks that can be used to build a large class of coordination data structures and protocols.
  - Examples include distributed queues, distributed locks, and leader election among a group of peers.
- ZooKeeper facilitates loosely coupled interactions:
  - ZooKeeper interactions support participants that do not need to know about one another.
- ZooKeeper provides an open source, shared repository of implementations and recipes of common coordination patterns.



# Layers of Apache Hadoop Ecosystem Tools



# Challenges Faced by Big Data Technologies

- Heterogeneity of information
  - The cost of doing a clean job of integration to bring all data into a single structure is prohibitive for most applications;
  - Proper interpretation of data analysis results requires large amounts of metadata.
- Privacy and confidentiality
  - There is widespread concern about personal information being compromised.
  - Regulations and laws are not always available (except HIPAA).
- Need for visualization and better human interfaces
- Inconsistent and incomplete information
  - There is an inherent uncertainty about data collected from regular users using normal devices when such data comes in multiple forms;
    - e.g., images, rates of speed, direction of travel.



# Why do you need Hadoop?

- More data is coming
  - IoT, sensor data, streaming
- More data means bigger questions
- More data means better answers
- Hadoop easily scales to store and handle all of your data
- Hadoop is cost effective
  - Typically provides a significant cost-per-terabyte saving over traditional legacy systems
- Hadoop integrates with your existing datacenter components



# Why do you need Hadoop? (contd...)

- Go from surviving to innovating
- Turn cost centers into revenue generators
- Hadoop lets you exploit the data you have
- Hadoop lets you exploit data you have been throwing away
- Hadoop is the foundation for the applications of the future
- Answer questions that you previously could not ask





# Hadoop IIIa

Oozie

# The Motivation for Oozie

- Many problems cannot be solved with a single MapReduce job
- Instead, a **workflow** of jobs must be created
- Simple workflow:
  - Run Job A
  - Use output of Job A as input to Job B
  - Use output of Job B as input to Job C
  - Output of Job C is the final required output
- Easy if the workflow is linear like above
  - Can be created as standard Driver code



# The Motivation for Oozie

- If the workflow is more complex, Driver code becomes much more difficult to maintain
- Example: running multiple jobs in parallel, using the output from all of those jobs as the input to the next job
- Example: including Hive or Pig jobs as part of the workflow



# What is Oozie?

- Oozie is a ‘workflow engine’
- Runs on a server
  - Typically outside the cluster
- Runs workflows of Hadoop jobs
  - Including Pig, Hive, Sqoop jobs
  - Submits those jobs to the cluster based on a workflow definition
- Workflow definitions are submitted via HTTP
- Jobs can be run at specific times
  - One-off or recurring jobs
- Jobs can be run when data is present in a directory



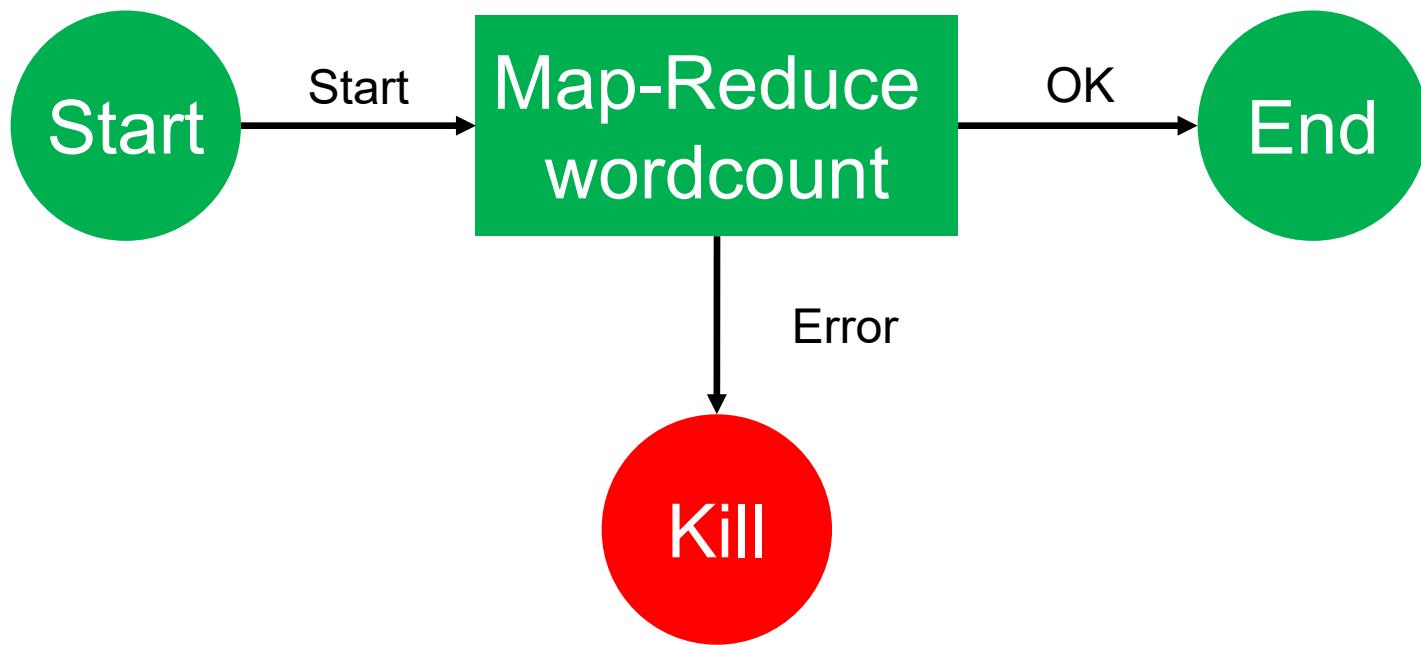
# Oozie Workflow Basics

- Oozie workflows are written in XML
- Workflow is a collection of actions
  - MapReduce jobs, Pig jobs, Hive jobs, etc.
- A workflow consists of control flow nodes and action nodes
- Control flow nodes define the beginning and end of a workflow
  - They provide methods to determine the workflow execution path
    - Example: Run multiple jobs simultaneously
- Action nodes trigger the execution of a processing task
  - e.g., a MapReduce job, a Hive query, a Sqoop data import job, etc.



# Oozie Example

- Simple example workflow for WordCount:



Source: [https://oozie.apache.org/docs/3.1.3-incubating/DG\\_Overview.html](https://oozie.apache.org/docs/3.1.3-incubating/DG_Overview.html)



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end'/>
 <error to='kill'/>
 </action>
 <kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
 </kill/>
 <end name='end' />
</workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">

 A workflow is wrapped in the workflow-app
 entity

 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end' />
 <error to='kill' />
</action>
<kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
</kill>
 <end name='end' />
</workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end' />
 <error to='kill' />
</action>
<kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
</kill/>
<end name='end' />
</workflow-app>
```

The start node is the control node which tells Oozie which workflow node should be run first. There must be one start node in an Oozie workflow. In our example, we are telling Oozie to start by transitioning to the wordcount workflow node.



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount' />
 <action name='wordcount'>
 <map-reduce>
 <configuration>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end' />
 <error to='kill' />
 </action>
 <kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
 </kill/>
 <end name='end' />
</workflow-app>
```

The wordcount action node defines a map-reduce action – a standard Java MapReduce job.



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <end to='end' />
</workflow-app>
```

Within the action, we define the job's properties.



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
```

We specify what to do if the action ends successfully, and what to do if it fails. In this example, if the job is successful we go to the end node. If it fails we go to the kill node.

```
 </map-reduce>
 <ok to='end'/>
 <error to='kill'/>
 </action>
 <kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
 </kill/>
 <end name='end' />
 </workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
```

If the workflow reaches a kill node, it will kill all running actions and then terminate with an error. A workflow can have zero or more kill nodes.

```
</action>
<kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
</kill/>
<end name='end' />
</workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
```

Every workflow must have an end node. This indicates that the workflow has completed successfully.

```
</kill/>
<end name='end' />
</workflow-app>
```

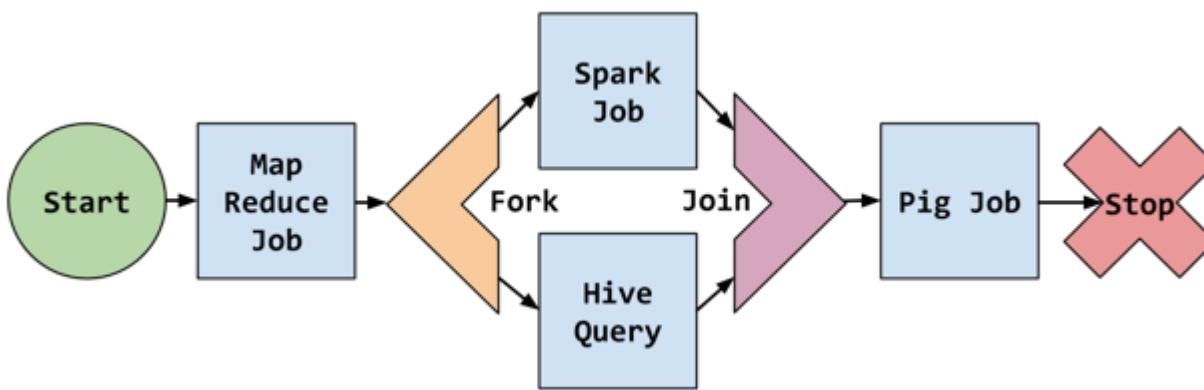


# Other Oozie Control Nodes

- A decision control node allows Oozie to determine the workflow execution path based on some criteria
  - Similar to a switch-case statement
- **fork** and **join** control nodes split one execution path into multiple execution paths which run concurrently
  - fork splits the execution path
  - join waits for all concurrent execution paths to complete before proceeding
  - fork and join are used in pairs



# Oozie Directed Acyclic Graphs (DAGs)



Source: <https://aws.amazon.com/blogs/big-data/use-apache-oozie-workflows-to-automate-apache-spark-jobs-and-more-on-amazon-emr/>



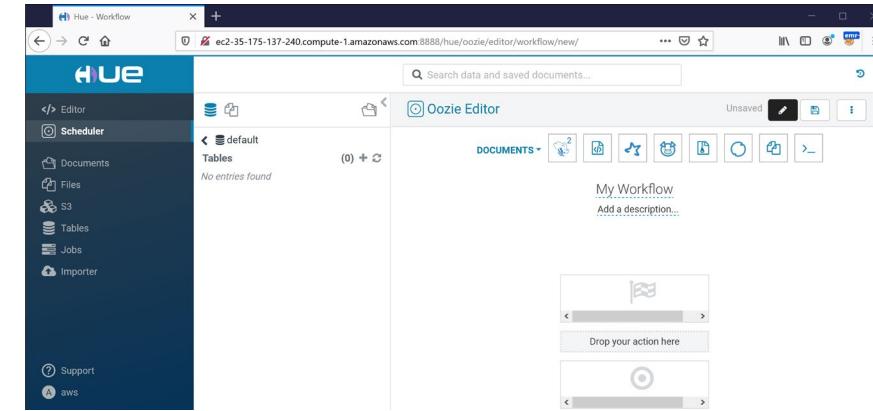
# Oozie Workflow Action Nodes

Node Name	Description
map-reduce	Runs either a Java MapReduce or Streaming job
fs	Create directories, move or delete files or directories
java	Runs the main() method in the specified Java class as a single-Map, Map-only job on the cluster
pig	Runs a Pig script
hive	Runs a Hive query
sqoop	Runs a Sqoop job
email	Sends an email message



# Try

- EMR Tutorial:
  - <https://www.youtube.com/watch?v=jy1p2atrZjc>
- EMR/Hive Lab on AWS:
  - <https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html>
- Hue on AWS:
  - <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/accessing-hue.html>
- Hue sandbox:
  - <https://demo.gethue.com/hue/accounts/login>



# Hadoop III

catchup

# Data Security

The Hadoop Ecosystem



# Apache Sentry



- Apache Sentry is a plugin that provides fine-grained access control (authorization) to various Hadoop ecosystem components
  - Impala
  - Hive
  - Cloudera Search
  - The HDFS command line
- In conjunction with Kerberos authentication, Sentry authorization provides a complete cluster security solution



# Apache Sentry: Why Should I Use It?



- Hadoop has long supported Kerberos authentication
  - “Prove you are who you say you are”
- Typically, a production cluster also requires authorization controls
  - “This person is allowed to do only these things”
  - This is especially true for clusters in regulated environments such as financial services, healthcare, etc.
- Sentry allows an administrator to grant fine-grained access rights to individuals
  - For example, permission to view only a certain column in a given Hive table
- Sentry is a key component of a secure Hadoop cluster



# Other Tools

The Hadoop Ecosystem





# Apache Mahout

- Mahout is a Machine Learning library written in Java
- Used for
  - Collaborative filtering (recommendations)
  - Clustering (finding naturally occurring “groupings” in data)
  - Classification (determining whether new data fits a category)
- Why should you use Hadoop for Machine Learning?
  - “It’s not who has the best algorithms that wins. It’s who has the most data.”





# Apache Mahout

- Get the MovieLens data
  - wget <http://files.grouplens.org/datasets/movielens/ml-1m.zip>
  - unzip ml-1m.zip
- Convert ratings.dat, trade “::” for “,”, and take only the first three columns:
  - cat ml-1m/ratings.dat | sed 's/::/,/g' | cut -f1-3 -d , > ratings.csv
- Put ratings file into HDFS:
  - hadoop fs -put ratings.csv /ratings.csv
- Run the recommender job:
  - mahout recommenditembased --input /ratings.csv --output recommendations --numRecommendations 10 --outputPathForSimilarityMatrix similarity-matrix --similarityClassname SIMILARITY\_COSINE
- Look for the results in the part-files containing the recommendations:
  - hadoop fs -ls recommendations
  - hadoop fs -cat recommendations/part-r-00000 | head

```
12/10/24 05:51:03 INFO MahoutDriver: Program took 594270 ms (Minutes: 9.937833333333334)
[hadoop@ip-172-31-35-8 ~]$ hadoop fs -ls recommendations
Found 4 items
-rw-r--r-- 1 hadoop hadoop 0 2022-10-26 05:18 recommendations/_SUCCESS
-rw-r--r-- 1 hadoop hadoop 197023 2022-10-26 05:18 recommendations/part-r-00000
-rw-r--r-- 1 hadoop hadoop 197883 2022-10-26 05:18 recommendations/part-r-00001
-rw-r--r-- 1 hadoop hadoop 197669 2022-10-26 05:18 recommendations/part-r-00002
[hadoop@ip-172-31-35-8 ~]$ hadoop fs -cat recommendations/part-r-00000
[32745.0, 0.24615.0, 0.11119.0, 0.29681.0, 0.37975.0, 0.24145.0, 0.59415.0, 0.15175.0, 0.30355.0, 0.26415.0]
[3035.0, 0.24851.0, 0.10015.0, 0.18715.0, 0.21015.0, 0.30345.0, 0.22981.0, 0.59415.0, 0.25721.0, 0.23755.0]
[1320.0, 0.11245.0, 0.31015.0, 0.26355.0, 0.14495.0, 0.19615.0, 0.1815.0, 0.15175.0, 0.191215.0, 0.12015.0]
[22431.0, 0.14495.0, 0.29615.0, 0.36291.0, 0.31681.0, 0.37351.0, 0.26901.0, 0.36171.0, 0.18091.0, 0.133715.0, 0.124015.0]
[22431.0, 0.14495.0, 0.29615.0, 0.36291.0, 0.31681.0, 0.37351.0, 0.26901.0, 0.36171.0, 0.18091.0, 0.133715.0, 0.124015.0]
[35941.0, 0.252947.0, 0.39514.0, 0.518176.0, 0.318914.0, 0.4741864.0, 0.398214.0, 0.4755986.0, 0.331615.0, 0.458021.0, 0.374514.0, 0.453851.0, 0.348414.0, 0.451155]
[37,945.0, 0.405654.0, 0.394814.0, 0.3768.0, 0.351214.0, 0.4271555]
[18 [2112.0, 0.132015.0, 0.303515.0, 0.188415.0, 0.25235.0, 0.351015.0, 0.211015.0, 0.18715.0, 0.314815.0]
[21 [22431.0, 0.14495.0, 0.29615.0, 0.36291.0, 0.31681.0, 0.37351.0, 0.26901.0, 0.36171.0, 0.18091.0, 0.133715.0, 0.124015.0]
[24 [22431.0, 0.246015.0, 0.246515.0, 0.237015.0, 0.316815.0, 0.14495.0, 0.303515.0, 0.211215.0, 0.529515.0, 0.264115.0]
[27 [22431.0, 0.14495.0, 0.29615.0, 0.36291.0, 0.31681.0, 0.123315.0, 0.59415.0, 0.303515.0, 0.221515.0, 0.297115.0]
[30 [1957.0, 0.347115.0, 0.123115.0, 0.195815.0, 0.253915.0, 0.221515.0, 0.301914.0, 0.7629395.0, 0.133947.0, 0.71485.0, 0.263014.0, 0.9445]
cat: Unable to write to output stream.
[hadoop@ip-172-31-35-8 ~]$
```



Source: <https://aws.amazon.com/blogs/big-data/building-a-recommender-with-apache-mahout-on-amazon-elastic-mapreduce-emr/>



# Apache Avro

- Apache Avro is a language-neutral data serialization system.
- The project was created by Doug Cutting (the creator of Hadoop) to address the major downside of Hadoop Writables: lack of language portability.
  - Having a data format that can be processed by many languages (currently C, C++, C#, Java, JavaScript, Perl, PHP, Python, and Ruby) makes it easier to share datasets with a wider audience.





# Apache Avro

- Avro data is described using a language-independent schema.
  - You can read and write data that conforms to a given schema even if your code has not seen that particular schema before.
- Avro schemas are usually written in JSON, and data is usually encoded using a binary format.
- Avro specifies an object container format for sequences of objects, similar to Hadoop's sequence file.
- An Avro datafile has a metadata section where the schema is stored, which makes the file self-describing.
- Avro datafiles support compression and are splittable, which is crucial for a MapReduce, Pig, Hive, Spark, etc.



# Apache Parquet

- Apache Parquet is a columnar storage format that can efficiently store nested data.
- Columnar formats are attractive since they enable greater efficiency, in terms of both file size and query performance.
- A key strength of Parquet is its ability to store data that has a deeply nested structure in true columnar fashion.
  - Schemas with several levels of nesting are common in real-world systems.
  - The technique was introduced by Google in Dremel; Parquet was created by engineers at Twitter and Cloudera.
- Most of the time Parquet files are processed using higher-level tools like Pig, Hive, or Impala.



# Data Format Summary

Feature	Text	Sequence Files	Avro	Parquet
Supported by many tools	✓		✓	✓
Good performance at scale		✓	✓	✓
Binary format		✓	✓	✓
Embedded schema			✓	✓
Columnar organization				✓



# Apache ZooKeeper



- Apache ZooKeeper is a highly available, high-performance coordination service.
- ZooKeeper gives you a set of tools to build distributed applications that can safely handle partial failures.
  - Partial failures: when we don't even know if an operation failed.
- ZooKeeper is, at its core, a stripped-down filesystem that exposes a few simple operations and some extra abstractions, such as ordering and notifications.
  - It doesn't have files and directories, but a unified concept of a node, called a znode, that acts both as a container of data (like a file) and a container of other znodes (like a directory).

• More information: <http://zookeeper.apache.org/doc/r3.7.0/zookeeperOver.html>



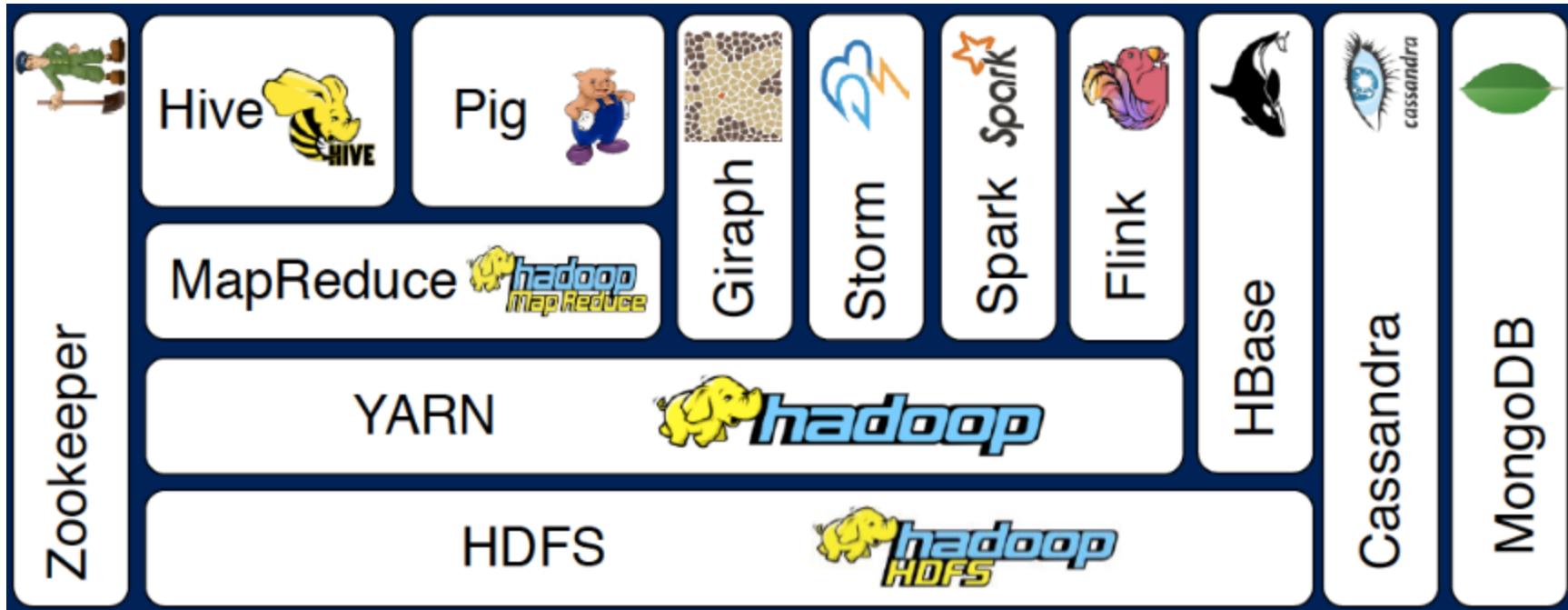
# Apache ZooKeeper



- The ZooKeeper primitives are a rich set of building blocks that can be used to build a large class of coordination data structures and protocols.
  - Examples include distributed queues, distributed locks, and leader election among a group of peers.
- ZooKeeper facilitates loosely coupled interactions:
  - ZooKeeper interactions support participants that do not need to know about one another.
- ZooKeeper provides an open source, shared repository of implementations and recipes of common coordination patterns.



# Layers of Apache Hadoop Ecosystem Tools



# Challenges Faced by Big Data Technologies

- Heterogeneity of information
  - The cost of doing a clean job of integration to bring all data into a single structure is prohibitive for most applications;
  - Proper interpretation of data analysis results requires large amounts of metadata.
- Privacy and confidentiality
  - There is widespread concern about personal information being compromised.
  - Regulations and laws are not always available (except HIPAA).
- Need for visualization and better human interfaces
- Inconsistent and incomplete information
  - There is an inherent uncertainty about data collected from regular users using normal devices when such data comes in multiple forms;
    - e.g., images, rates of speed, direction of travel.



# Why do you need Hadoop?

- More data is coming
  - IoT, sensor data, streaming
- More data means bigger questions
- More data means better answers
- Hadoop easily scales to store and handle all of your data
- Hadoop is cost effective
  - Typically provides a significant cost-per-terabyte saving over traditional legacy systems
- Hadoop integrates with your existing datacenter components



# Why do you need Hadoop? (contd...)

- Go from surviving to innovating
- Turn cost centers into revenue generators
- Hadoop lets you exploit the data you have
- Hadoop lets you exploit data you have been throwing away
- Hadoop is the foundation for the applications of the future
- Answer questions that you previously could not ask

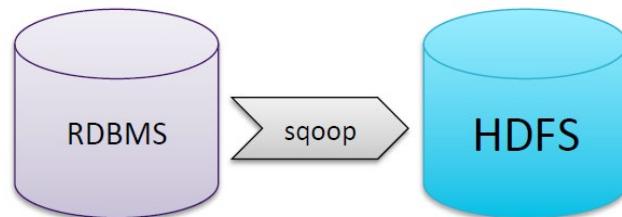


# Hadoop IV

Sqoop & Flume

# Importing Data From an RDBMS to HDFS

- Typical scenario: data stored in an RDBMS is needed in a MapReduce job
  - Lookup tables
  - Legacy data
- Possible to read directly from an RDBMS in your Mapper
  - Can lead to the equivalent of a distributed denial of service (DDoS) attack on your RDBMS
    - In practice – don't do it!
- Better solution: use **Sqoop** to import the data into HDFS beforehand





# Sqoop: SQL to Hadoop

- Sqoop: open-source tool originally written at Cloudera
  - Now a top-level Apache Software Foundation project
- Imports tables from an RDBMS into HDFS
  - Just one table
  - All tables in a database
  - Just portions of a table
    - Sqoop supports WHERE clauses
- Uses MapReduce to import the data
  - ‘Throttles’ the number of Mappers to avoid DDoS scenarios
- Uses a JDBC interface
  - Should work with virtually any JDBC-compatible database



# Sqoop Features

- Imports a single table, or all tables in a database
- Can specify which rows to import
  - via a WHERE clause
- Can specify which columns to import
- Can provide an arbitrary SELECT statement
- Sqoop can automatically create a Hive table based on the imported data
- Support incremental imports of data
- Can export data from HDFS to a database table



# Sqoop: Basic Syntax

- Standard syntax:

```
$ sqoop tool-name [tool-options]
```

- Tools include:

```
import
import-all-tables
list-tables
```

- Options include:

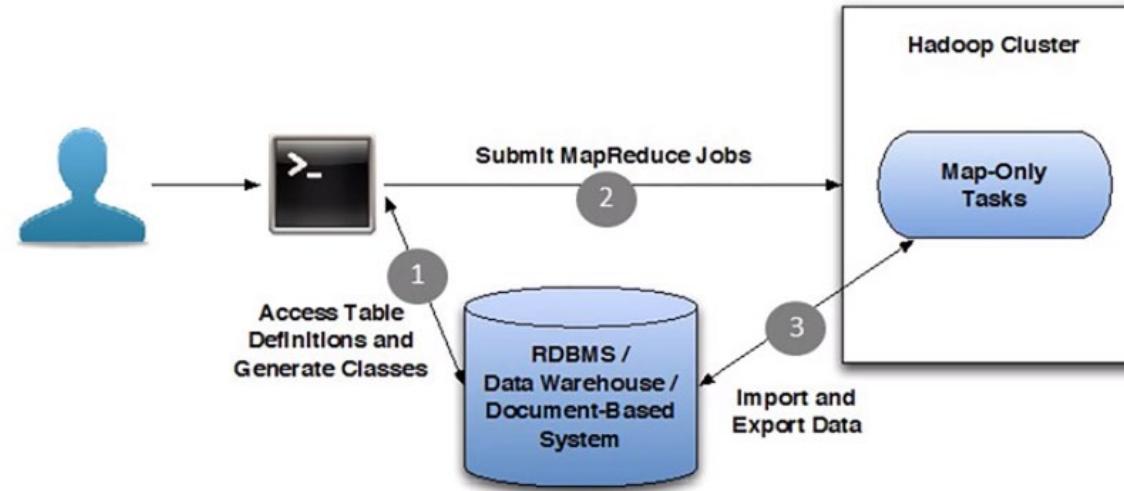
```
--connect
--username
--password
```

Instead of:  
"--password YOUR\_PASS"  
use: -P (it will prompt you  
for a password)



# Overview of the Import Process

- Imports are performed using Hadoop MapReduce jobs
- Sqoop begins by examining the table to be imported
  - Determines the primary key, if possible
  - Runs a boundary query to see how many records will be imported
  - Divides result of boundary query by the number of tasks (mappers)
    - Uses this to configure tasks so that they will have equal loads



# Sqoop: Example

- Example: import a table called employees from a database called ‘personnel’ in a MySQL RDBMS

```
$ sqoop import --username fred --password derf \
 --connect jdbc:mysql://database.example.com/personnel \
 --table employees
```

- Example: as above, but only records with an ID greater than 1000

```
$ sqoop import --username fred --password derf \
 --connect jdbc:mysql://database.example.com/personnel \
 --table employees \
 --where "id > 1000"
```



# Importing an Entire Database with Sqoop

- Import all tables from the database (fields will be tab-delimited)

```
$ sqoop import-all-tables \
 --connect jdbc:mysql://localhost/company \
 --username twheeler --password bigsecret \
 --fields-terminated-by '\t' \
 --warehouse-dir /mydata
```



# Importing Partial Tables with Sqoop

- Importing only specified columns from accounts table

```
sqoop import --table accounts \
 --connect jdbc:mysql://dbhost/loudacre \
 --username dbuser --password pw \
 --columns "id,first_name,last_name,state"
```

- Import only matching rows from accounts table

```
sqoop import --table accounts \
 --connect jdbc:mysql://dbhost/loudacre \
 --username dbuser --password pw \
 --where "state='CA'"
```



# Using a Free-Form Query

- You can also import the results of a query, rather than a single table
- Supply a complete SQL query using the --query option
  - You must add the literal WHERE \$CONDITIONS token
  - Use --split-by to identify field used to divide work among mappers
  - The --target-dir option is required for free-form queries

```
sqoop import \
 --connect jdbc:mysql://dbhost/loudacre \
 --username dbuser --password pw \
 --target-dir /data/loudacre/payable \
 --split-by accounts.id \
 --query 'SELECT accounts.id, first_name, last_name, bill_amount
FROM accounts JOIN invoices ON
 (accounts.id = invoices.cust_id) WHERE $CONDITIONS'
```



# Using a Free-Form Query with WHERE

- The --where option is ignored in a free-form query
  - You must specify your criteria using AND following the WHERE clause

```
sqoop import \
 --connect jdbc:mysql://dbhost/loudacre \
 --username dbuser --password pw \
 --target-dir /data/loudacre/payable \
 --split-by accounts.id \
 --query 'SELECT accounts.id, first_name, last_name, bill_amount
FROM accounts JOIN invoices ON
 (accounts.id = invoices.cust_id) WHERE $CONDITIONS AND bill_amount
>= 40'
```



# Incremental Imports with Sqoop

- What if new records are added to the database?
  - Could re-import all records, but this is inefficient
- Sqoop's incremental *append* mode imports only new records
  - Based on value of last record in specified column

```
$ sqoop import \
 --connect jdbc:mysql://localhost/company \
 --username twalker --password bigsecret \
 --warehouse-dir /mydata \
 --table orders \
 --incremental append \
 --check-column order_id \
 --last-value 6713821
```



# Handling Modifications With Incremental Imports

- What if existing records are also modified in the database?
  - Incremental append mode doesn't handle this
- Sqoop's “lastmodified” append mode adds and updates records
  - Caveat: You must maintain a timestamp column in your table
- Beneath the hood:
  - Sqoop runs two standalone MapReduce jobs
    - One that imports the changed rows into a temporary directory in HDFS
    - The other that merges the changed rows with the records previously imported into a new set of up-to-date records

```
$ sqoop import \
 --connect jdbc:mysql://localhost/company \
 --username twheeler \
 --password bigsecret \
 --warehouse-dir /mydata \
 --incremental lastmodified \
 --check-column last_update_date \
 --last-value "2013-06-12 03:15:59"
```



# Sqoop: Other Options

- Sqoop can take data from HDFS and insert it into an already-existing table in an RDBMS with the command

```
$ sqoop export [options]
```

- For general Sqoop help:

```
$ sqoop help
```

- For help on a particular command:

```
$ sqoop help command
```



# Possible Driver Error

- ERROR sqoop.Sqoop: Got exception running Sqoop: java.lang.RuntimeException: Could not load db driver class: com.mysql.jdbc.Driver
  - Solution: Copy driver file in Sqoop folder
    - bash -c "wget -O mysql-connector-j-8.0.33.jar https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.0.33/mysql-connector-j-8.0.33.jar;sudo mv mysql-connector-j-8.0.33.jar /usr/lib/sqoop/lib/"
    - cd /usr/lib/sqoop/lib/
    - chmod 755 mysql-connector-j-8.0.33.jar
- ERROR tool.ImportTool: Import failed: java.io.IOException: Generating splits for a textual index column allowed only in case of "-Dorg.apache.sqoop.splitter.allow\_text\_splitter=true" property passed as a parameter
  - Solution: pass the parameter:
    - -D org.apache.sqoop.splitter.allow\_text\_splitter=true
    - You don't need it if number of mappers is 1 (-m 1) or if the split is on numeric values



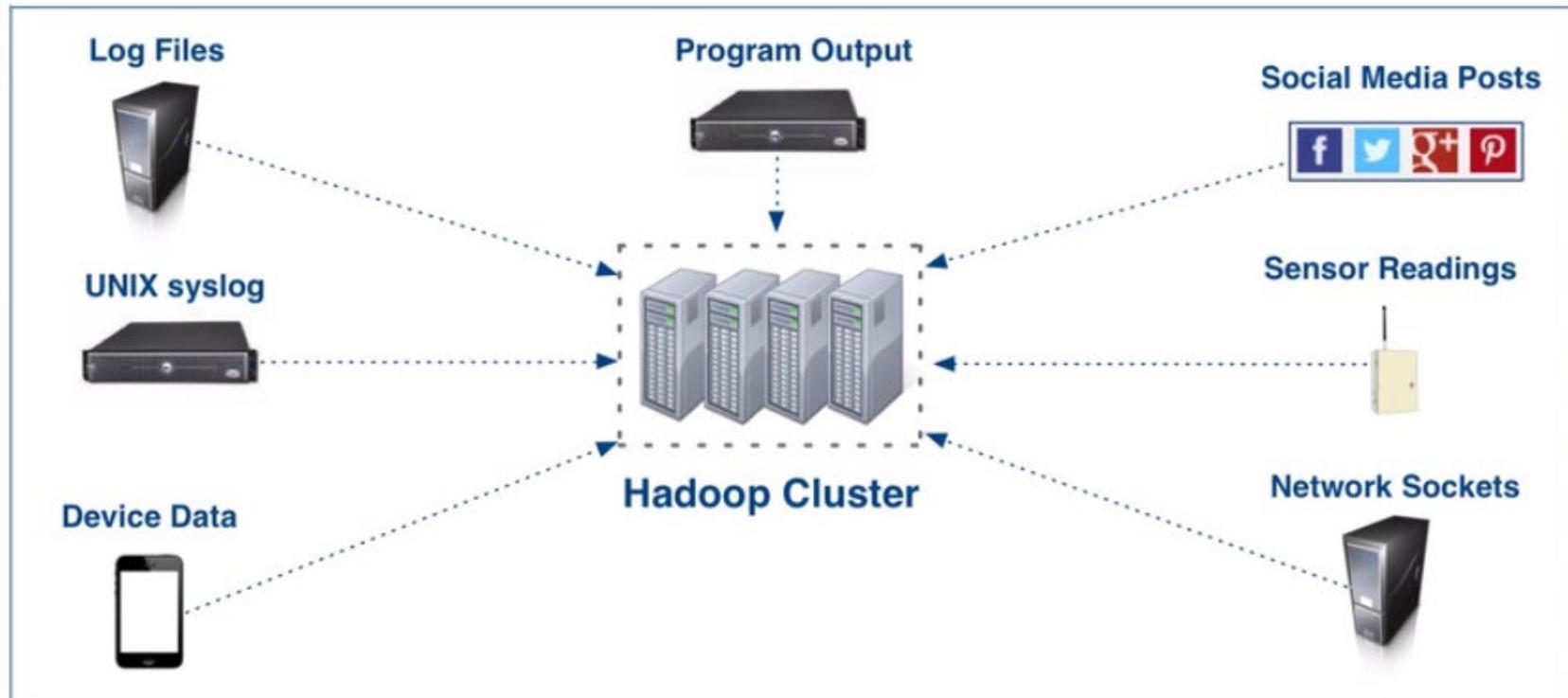


# Apache Flume

- Apache Flume is designed for high-volume ingestion into Hadoop of event-based data.
  - e.g., using Flume to move log events from (remote) log files into new aggregated files in HDFS.
  - Now widely used for collection of any type of event data
  - Supports aggregating from many sources into HDFS
- Flume is
  - Distributed
  - Reliable and available
  - Horizontally scalable
  - Extensible



# Apache Flume





# Apache Flume

- To use Flume, we need to run a Flume agent, which is a long-lived Java process that runs sources and sinks, connected by channels.
  - A source in Flume produces events and delivers them to the channel, which stores the events until they are forwarded to the sink.
  - Flume uses separate transactions to guarantee delivery from the source to the channel and from the channel to the sink.
  - Using Solr (or Elasticsearch) as search, near-real-time indexing can be achieved.

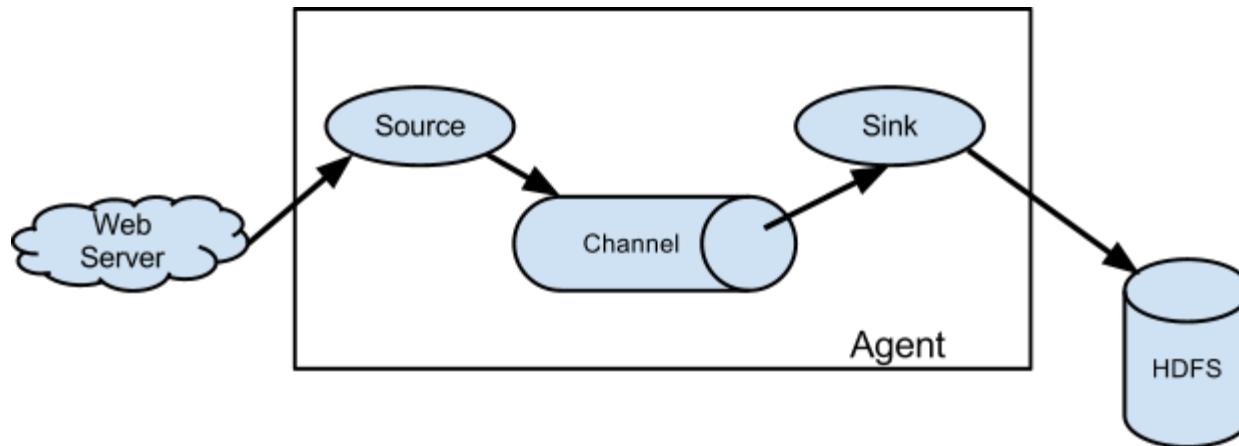


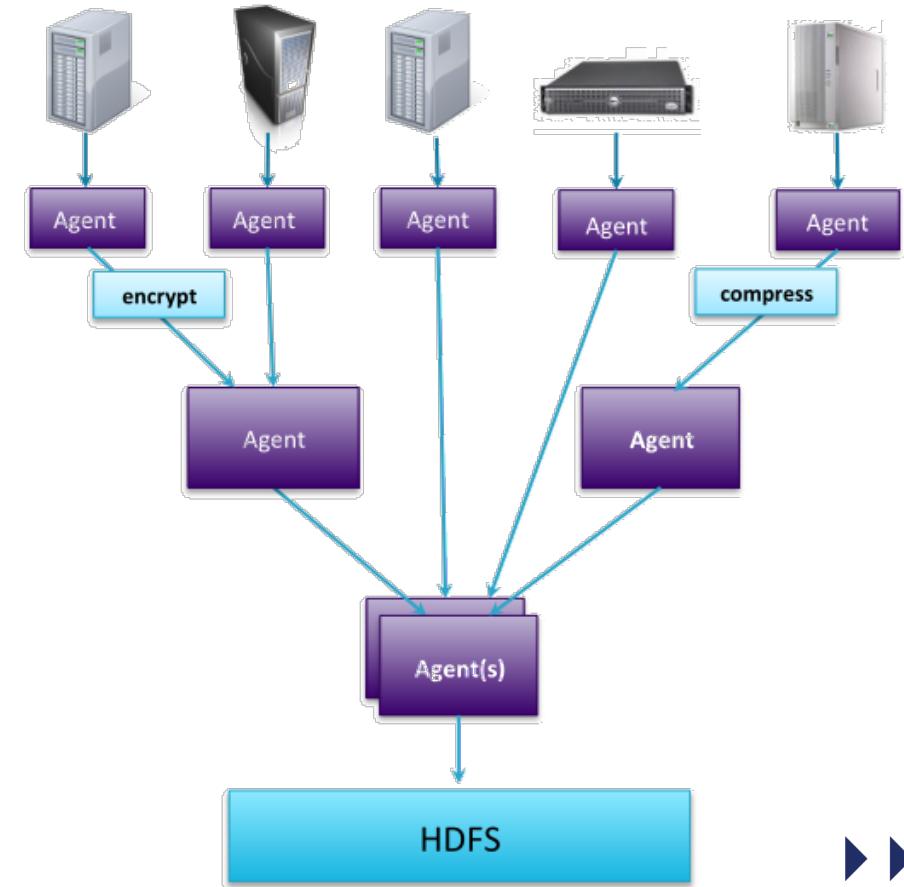
Image source: <https://flume.apache.org/FlumeUserGuide.html>



# Apache Flume: Pipeline Example



- Collect data as it is produced
  - Files, syslogs, stdout or custom source
- Process in place
  - e.g., encrypt, compress
- Pre-process data before storing
  - e.g., transform, scrub, enrich
- Write in parallel
  - Scalable throughput
- Store in any format
  - Text, compressed, binary, or custom sink

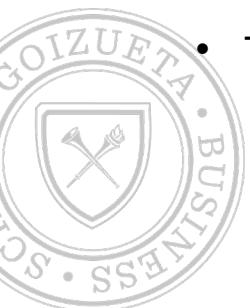
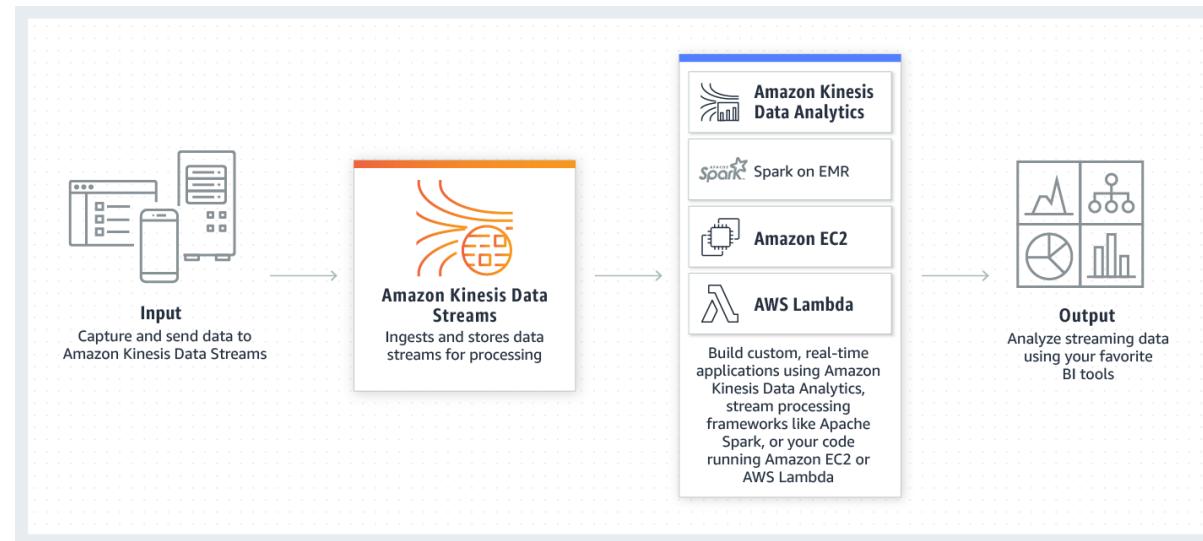


# AWS Kinesis

- Similar to Flume and Kafka, AWS Kinesis collects, processes, and analyze real-time, streaming data.
  - Flume pushes data (into sinks), but Kafka and Kinesis requires a “consumer” to pull data. Thus, Flume is used more for log data.
  - Kafka is fast and free but requires implementation for an enterprise-class solution. Kinesis is as an out-of-the-box streaming data solution similar to Kafka. Kinesis has shards, Kafka has partitions.

- Try:

- <https://docs.aws.amazon.com/streams/latest/dev/examples.html>
- <https://docs.aws.amazon.com/kinesisanalytics/latest/dev/app-hotspots-detection.html>





# Hadoop IIIa

Oozie

# The Motivation for Oozie

- Many problems cannot be solved with a single MapReduce job
- Instead, a **workflow** of jobs must be created
- Simple workflow:
  - Run Job A
  - Use output of Job A as input to Job B
  - Use output of Job B as input to Job C
  - Output of Job C is the final required output
- Easy if the workflow is linear like above
  - Can be created as standard Driver code



# The Motivation for Oozie

- If the workflow is more complex, Driver code becomes much more difficult to maintain
- Example: running multiple jobs in parallel, using the output from all of those jobs as the input to the next job
- Example: including Hive or Pig jobs as part of the workflow



# What is Oozie?

- Oozie is a ‘workflow engine’
- Runs on a server
  - Typically outside the cluster
- Runs workflows of Hadoop jobs
  - Including Pig, Hive, Sqoop jobs
  - Submits those jobs to the cluster based on a workflow definition
- Workflow definitions are submitted via HTTP
- Jobs can be run at specific times
  - One-off or recurring jobs
- Jobs can be run when data is present in a directory



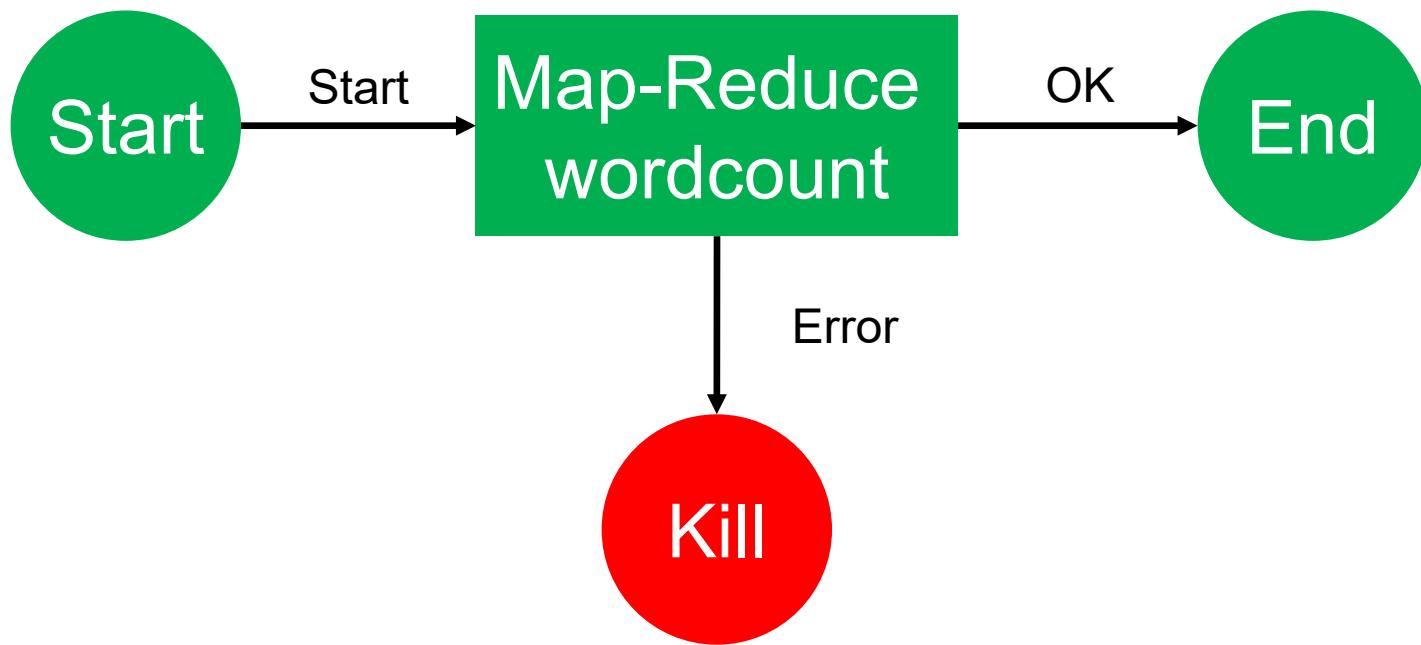
# Oozie Workflow Basics

- Oozie workflows are written in XML
- Workflow is a collection of actions
  - MapReduce jobs, Pig jobs, Hive jobs, etc.
- A workflow consists of control flow nodes and action nodes
- Control flow nodes define the beginning and end of a workflow
  - They provide methods to determine the workflow execution path
    - Example: Run multiple jobs simultaneously
- Action nodes trigger the execution of a processing task
  - e.g., a MapReduce job, a Hive query, a Sqoop data import job, etc.



# Oozie Example

- Simple example workflow for WordCount:



Source: [https://oozie.apache.org/docs/3.1.3-incubating/DG\\_Overview.html](https://oozie.apache.org/docs/3.1.3-incubating/DG_Overview.html)



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end'/>
 <error to='kill'/>
 </action>
 <kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
 </kill/>
 <end name='end' />
</workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">

 A workflow is wrapped in the workflow-app
 entity

 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end' />
 <error to='kill' />
</action>
<kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
</kill>
 <end name='end' />
</workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end' />
 <error to='kill' />
</action>
<kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
</kill/>
<end name='end' />
</workflow-app>
```

The start node is the control node which tells Oozie which workflow node should be run first. There must be one start node in an Oozie workflow. In our example, we are telling Oozie to start by transitioning to the wordcount workflow node.



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount' />
 <action name='wordcount'>
 <map-reduce>
 <configuration>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <ok to='end' />
 <error to='kill' />
 </action>
 <kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
 </kill/>
 <end name='end' />
</workflow-app>
```

The wordcount action node defines a map-reduce action – a standard Java MapReduce job.



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
 <name>mapred.output.dir</name>
 <value>${outputDir}</value>
 </property>
 </configuration>
 </map-reduce>
 <end to='end' />
</workflow-app>
```

Within the action, we define the job's properties.



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
```

We specify what to do if the action ends successfully, and what to do if it fails. In this example, if the job is successful we go to the end node. If it fails we go to the kill node.

```
 </map-reduce>
 <ok to='end'/>
 <error to='kill'/>
 </action>
 <kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
 </kill/>
 <end name='end' />
 </workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
```

If the workflow reaches a kill node, it will kill all running actions and then terminate with an error. A workflow can have zero or more kill nodes.

```
</action>
<kill name='kill'>
 <message>Something went wrong: ${wf:errorCode('wordcount')}</message>
</kill/>
<end name='end' />
</workflow-app>
```



# Oozie Example

```
<workflow-app name='wordcount-wf' xmlns="uri:oozie:workflow:0.1">
 <start to='wordcount'/>
 <action name='wordcount'>
 <map-reduce>
 <job-tracker>${jobTracker}</job-tracker>
 <name-node>${nameNode}</name-node>
 <configuration>
 <property>
 <name>mapred.mapper.class</name>
 <value>org.myorg.WordCount.Map</value>
 </property>
 <property>
 <name>mapred.reducer.class</name>
 <value>org.myorg.WordCount.Reduce</value>
 </property>
 <property>
 <name>mapred.input.dir</name>
 <value>${inputDir}</value>
 </property>
 <property>
```

Every workflow must have an end node. This indicates that the workflow has completed successfully.

```
</kill/>
<end name='end' />
</workflow-app>
```

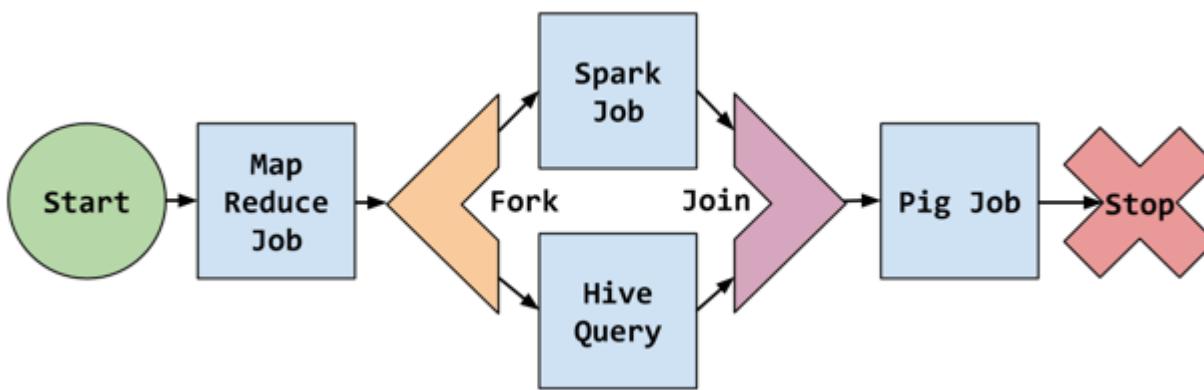


# Other Oozie Control Nodes

- A decision control node allows Oozie to determine the workflow execution path based on some criteria
  - Similar to a switch-case statement
- **fork** and **join** control nodes split one execution path into multiple execution paths which run concurrently
  - fork splits the execution path
  - join waits for all concurrent execution paths to complete before proceeding
  - fork and join are used in pairs



# Oozie Directed Acyclic Graphs (DAGs)



Source: <https://aws.amazon.com/blogs/big-data/use-apache-oozie-workflows-to-automate-apache-spark-jobs-and-more-on-amazon-emr/>



# Oozie Workflow Action Nodes

Node Name	Description
map-reduce	Runs either a Java MapReduce or Streaming job
fs	Create directories, move or delete files or directories
java	Runs the main() method in the specified Java class as a single-Map, Map-only job on the cluster
pig	Runs a Pig script
hive	Runs a Hive query
sqoop	Runs a Sqoop job
email	Sends an email message





# Hadoop IIIa

Oozie

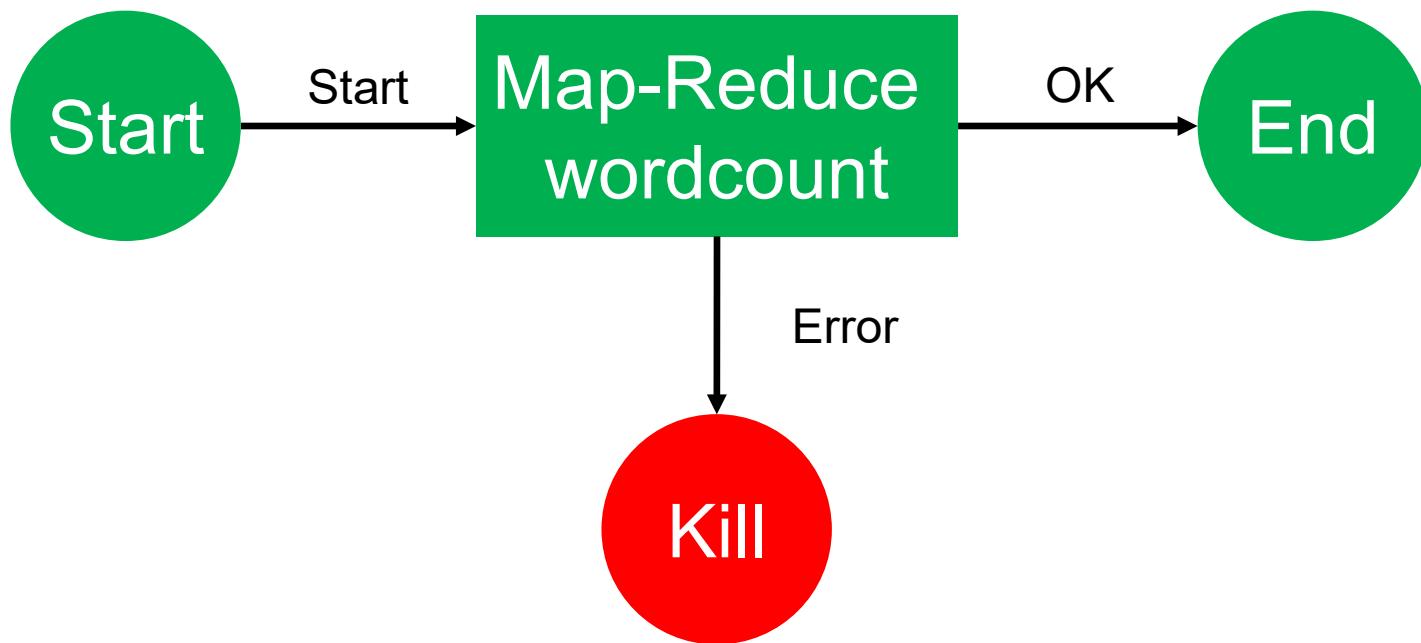
# What is Oozie?

- Oozie is a ‘workflow engine’
- Runs on a server
  - Typically outside the cluster
- Runs workflows of Hadoop jobs
  - Including Pig, Hive, Sqoop jobs
  - Submits those jobs to the cluster based on a workflow definition
- Workflow definitions are submitted via HTTP
- Jobs can be run at specific times
  - One-off or recurring jobs
- Jobs can be run when data is present in a directory



# Oozie Example

- Simple example workflow for WordCount:



Source: [https://oozie.apache.org/docs/3.1.3-incubating/DG\\_Overview.html](https://oozie.apache.org/docs/3.1.3-incubating/DG_Overview.html)



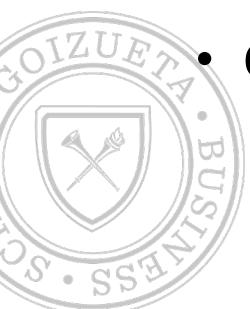
# Oozie Workflow Action Nodes

Node Name	Description
map-reduce	Runs either a Java MapReduce or Streaming job
fs	Create directories, move or delete files or directories
java	Runs the main() method in the specified Java class as a single-Map, Map-only job on the cluster
pig	Runs a Pig script
hive	Runs a Hive query
sqoop	Runs a Sqoop job
email	Sends an email message



# Example

- Hue (1):
  - Create Sqoop script
    - sqoop import --connect jdbc:mysql://isom671.cqxikovybdnm.us-east-2.rds.amazonaws.com/happiness --username admin --password isom671db --table y2015 --target-dir /user/hadoop/happy/ -m 1
  - Create Oozie workflow
    - Add sqoop script
- Hue (2):
  - Create Hive Query (1): SELECT MAX(salary) as maxsal from jobs;
  - Create Hive Query (2): SELECT \* from jobs where salary>=100000;
  - Create Hive Query (3): SELECT \* from jobs where salary<100000;
  - Create Oozie Workflow
- CLI:
  - <https://aws.amazon.com/blogs/big-data/use-apache-oozie-workflows-to-automate-apache-spark-jobs-and-more-on-amazon-emr/>





# Hive I

Data Processing in Hadoop



# Overview of Apache Hive

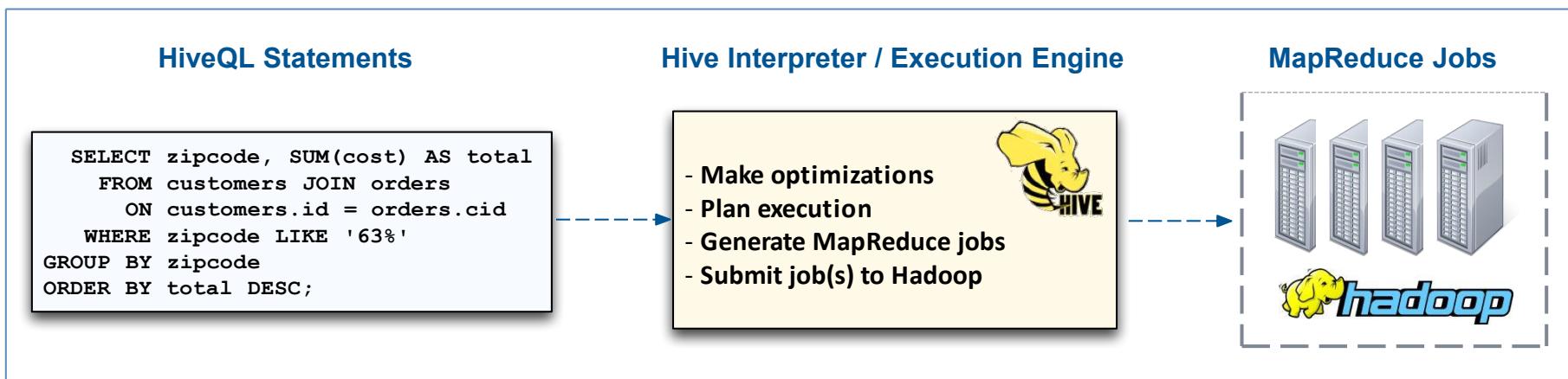
- Apache Hive is a high-level abstraction on top of MapReduce
  - Hive offers data warehousing infrastructure for Hadoop
  - Uses a SQL-like language called HiveQL
  - Generates MapReduce jobs that run on the Hadoop cluster
    - Automatically runs the jobs, and displays the results to the user
  - Originally developed by Facebook for data warehousing
    - Now an open-source Apache project

```
SELECT zipcode, SUM(cost) AS total
 FROM customers
 JOIN orders
 ON customers.cust_id = orders.cust_id
 WHERE zipcode LIKE '63%'
 GROUP BY zipcode
 ORDER BY total DESC;
```

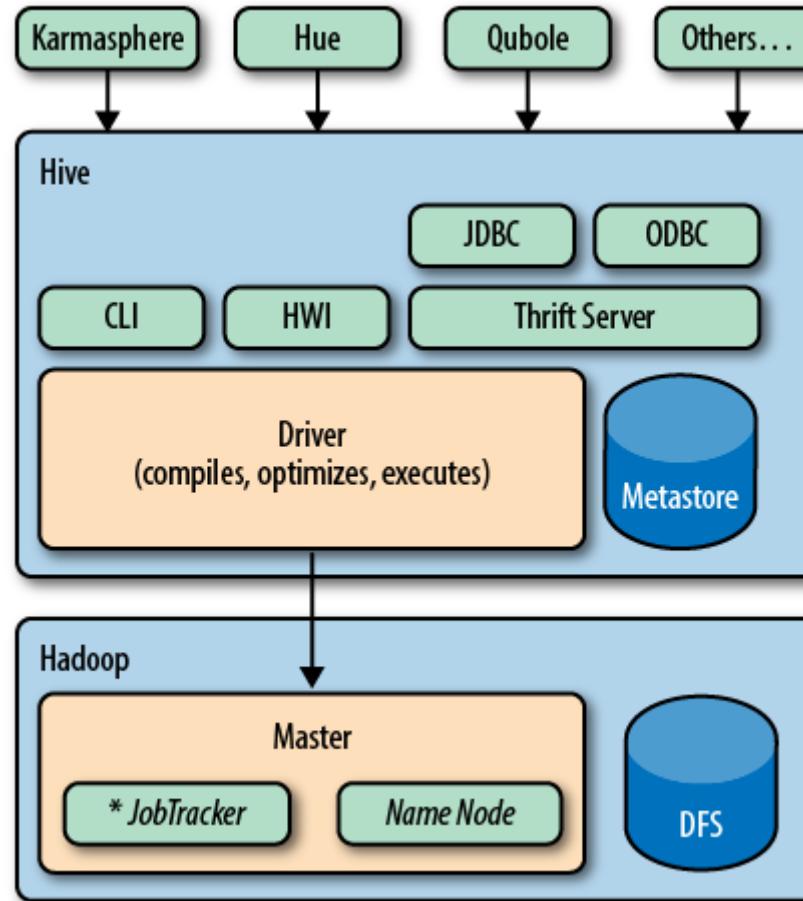


# High-Level Overview for Hive Users

- Hive runs on the client machine
  - Turns HiveQL queries into MapReduce jobs
  - Submits those jobs to the cluster



# Hive Modules



Source: <https://learning.oreilly.com/library/view/programming-hive/9781449326944/ch01.html>



# Why Use Apache Hive?

- More productive than writing MapReduce directly
  - Five lines of HiveQL might be equivalent to 100 lines or more of Java MR
- Brings large-scale data analysis to a broader audience
  - No software development experience required
  - Leverage existing knowledge of SQL
- Offers interoperability with other systems
  - Extensible through Java and external scripts
  - Many business intelligence (BI) tools support Hive
- Caveat Emptor
  - Remember that Hive generates MapReduce jobs, making it ultimately a batch processing platform, not a real-time / interactive platform



# Word Count Example

- Python code for word-count is about 20 lines
- Java code for word-count is about 80 lines
- Hive query for word-count is 7 lines:

```
CREATE TABLE FILES (line STRING);
LOAD DATA INPATH '/user/hadoop/docs' OVERWRITE INTO TABLE FILES;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, ' ')) AS word FROM FILES) w
GROUP BY word
ORDER BY word;
```

*You can specify the data to be loaded directly from S3 bucket (the data will be lost in S3)*

*You can use s3-dist-copy:  
s3-dist-cp --src=s3://s3distcp-source/input-data --dest=hdfs://output-folder1*

‘docs’ is a document full of text



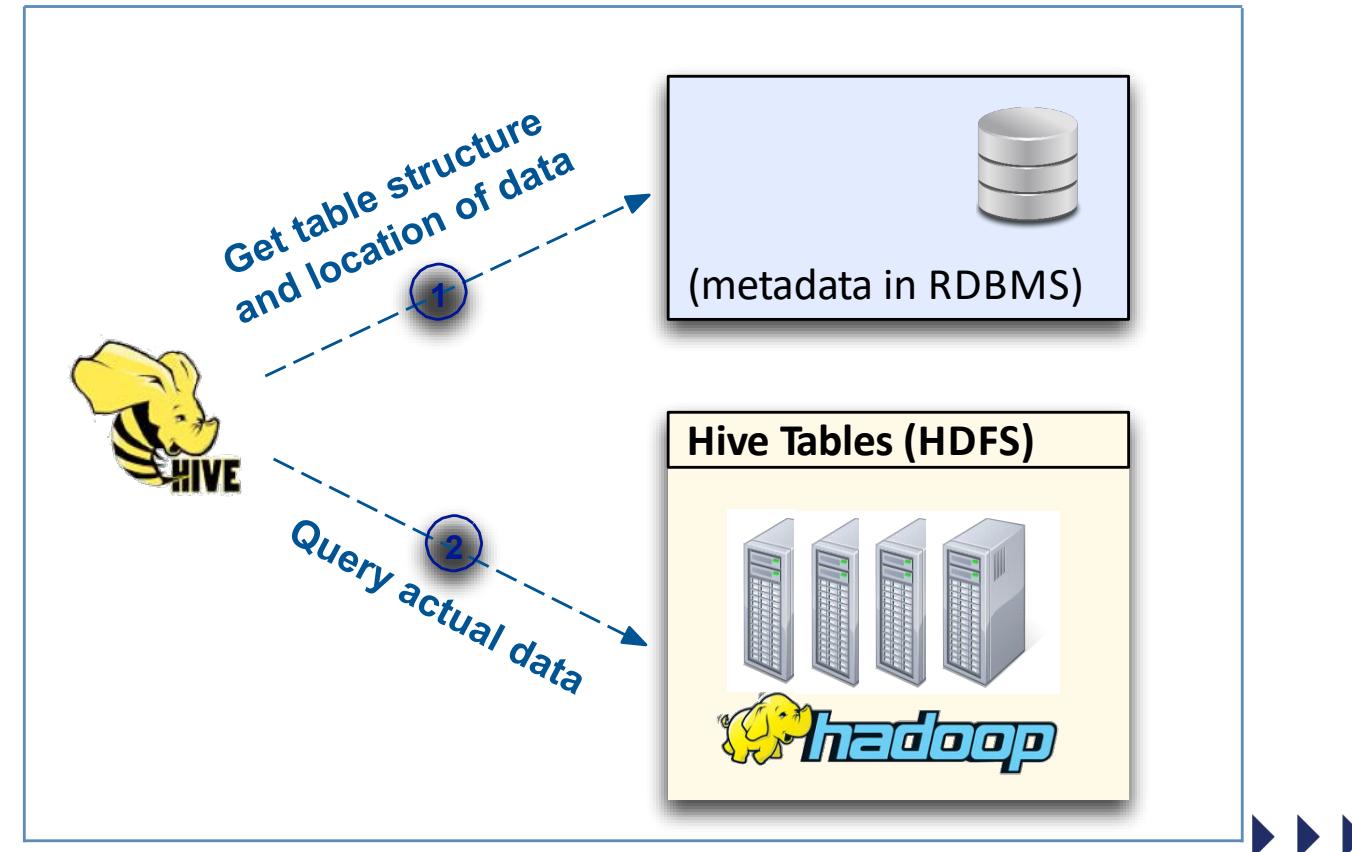
# How Hive Loads and Stores Data

- Hive's queries operate on tables, just like in an RDBMS
  - A table is simply an HDFS directory containing one or more files
  - Default path: /user/hive/warehouse/<table\_name>
  - Hive supports many formats for data storage and retrieval
    - Unlike an RDBMS which must conform to rigorous schema constraints
- How does Hive know the structure and location of tables?
  - These are specified when tables are created
  - This metadata is stored in Hive's metastore
    - Contained in an RDBMS such as MySQL
      - That's right, Hive (unlike Pig) requires a real RDBMS to store table metadata



# How Hive Loads and Stores Data

- Hive consults the metastore to determine data format and location
  - The query itself operates on data stored on a filesystem (typically HDFS)



# Using the Hive Shell

- You can execute HiveQL statements in the Hive Shell
  - This interactive tool is similar to the MySQL shell
- Run the `hive` command to start the Hive shell
  - The Hive shell will display its `hive>` prompt
  - Each statement must be terminated with a semicolon
  - Use the `quit` command to exit the Hive shell

```
$ hive
hive> SELECT cust_id, fname, lname
 FROM customers WHERE zipcode=20525;
1000000 Quentin Shepard
1000001 Brandon Louis
1000002 Marilyn Ham
hive> quit;
$
```



# Accessing Hive from the Command Line

- You can also execute a file containing HiveQL code using the -f option

```
$ hive -f myquery.hql
```

- Or use HiveQL directly from the command line using the -e option

```
$ hive -e 'SELECT * FROM users'
```

- Use the -S (silent) option to suppress informational messages

- Can also be used with -e or -f options

```
$ hive -S
```



# Interacting with the Operating System and HDFS

- Use ! to execute system commands from within Hive
  - Neither pipes nor globs (wildcards) are supported
  - ```
hive> ! date;
```
 - ```
Mon May 20 16:44:35 PDT 2017
```
- Prefix HDFS commands with dfs to use them from within Hive
  - ```
hive> dfs -mkdir /reports/sales/2017;
```



Hive's Data Definition Language (DDL)

Creating Databases and Tables



Rajiv Garg



Creating Databases in Hive

- Hive databases are simply namespaces
 - Helps to organize your tables
- To create a new database:
 - `hive> CREATE DATABASE dualcore;`
 - Adds the database definition to the metastore
 - Creates a storage directory in HDFS; e.g.,
`/user/hive/warehouse/dualcore.db`
- To conditionally create a new database
 - Avoids error in case database already exists (useful for scripting)
 - `hive> CREATE DATABASE IF NOT EXISTS dualcore;`
- Hive's Data Definition Language (DDL):
 - <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL>



Example Table Definition

- The following example creates a new table named jobs
 - Data stored as text with four comma-separated fields per line

```
CREATE TABLE jobs
  (id INT,
   title STRING,
   salary INT,
   posted TIMESTAMP
  )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

- Example of corresponding record for the table above
- 1,Data Analyst,100000,2013-06-21 15:52:03



Creating Tables with Complex Column Types

- Complex columns are typed
 - Arrays have a single type
 - Maps have a key type and a value type
 - Structs have a type for each attribute
- These types are specified with angle brackets

```
CREATE TABLE stores
(store_id SMALLINT,
departments ARRAY<STRING>,
staff MAP<STRING, STRING>,
address STRUCT<street:STRING,
           city:STRING,
           state:STRING,
           zipcode:STRING>
);
```



Loading Data From HDFS Files

- To load data, simply add files to the table's directory in HDFS

- Can be done directly using hadoop fs commands
 - This example loads data from HDFS into Hive's sales table

```
$ hadoop fs -mv sales.txt /user/hive/warehouse/sales/
```

- Alternatively, use Hive's **LOAD DATA INPATH** command

- Done from within the Hive shell (or a Hive script)
 - This moves data within HDFS, just like the command above
 - *Destroying the original in the process*
 - Source can be either a file or directory

```
hive> LOAD DATA INPATH 'sales.txt' INTO TABLE sales;
```



Loading Data From Files

- Add the `LOCAL` keyword to load data from the local disk

- Copies a local file (or directory) to the table's directory in HDFS

```
hive> LOAD DATA LOCAL INPATH '/home/bob/sales.txt'  
      INTO TABLE sales;
```

- This is equivalent to the following hadoop fs -put command

```
$ hadoop fs -put /home/bob/sales.txt \  
  /user/hive/warehouse/sales
```



Loading Data From Files

- Add the **OVERWRITE** keyword to delete all existing records before import
 - Removes all files within the table's directory
 - Then moves the new files into that directory

```
hive> LOAD DATA INPATH '/depts/finance/salesdata'  
      OVERWRITE INTO TABLE sales;
```



Appending Selected Records to a Table

- Another way to populate a table is through a query

- Use `INSERT INTO` to add results to an existing Hive table

```
INSERT INTO TABLE accounts_copy  
    SELECT * FROM accounts;
```

- Specify a `WHERE` clause to control which records are appended

```
INSERT INTO TABLE loyal_customers  
    SELECT * FROM accounts  
    WHERE YEAR(acct_create_dt) = 2008  
        AND acct_close_dt IS NULL;
```



Loading Data From a Relational Database

- Sqoop has built-in support for importing data into Hive
- Just add the *--hive-import* option to your Sqoop command

- Creates the table in Hive (metastore)
- Imports data from RDBMS to table's directory in HDFS

```
$ sqoop import \  
  --connect jdbc:mysql://localhost/dualcore \  
  --username training \  
  --password training \  
  --fields-terminated-by '\t' \  
  --table employees \  
  --hive-import
```





Hive I (catchup)

Data Processing in Hadoop

Loading Data From a Relational Database

- Sqoop has built-in support for importing data into Hive
- Just add the *--hive-import* option to your Sqoop command

- Creates the table in Hive (metastore)
- Imports data from RDBMS to table's directory in HDFS

```
$ sqoop import \  
  --connect jdbc:mysql://localhost/dualcore \  
  --username training \  
  --password training \  
  --fields-terminated-by '\t' \  
  --table employees \  
  --hive-import
```





Hive II

Data Processing in Hadoop

Removing a Database

- Removing a database is similar to creating it

- Just replace CREATE with `DROP`

```
hive> DROP DATABASE dualcore;
```

```
hive> DROP DATABASE IF EXISTS dualcore;
```

- These commands will fail if the database contains tables

- Add the `CASCADE` keyword to force removal

- Caution: this command removes data in HDFS!

```
hive> DROP DATABASE dualcore CASCADE;
```



Removing a Table

- Syntax is similar to database removal

```
hive> DROP TABLE customers;
```

```
hive> DROP TABLE IF EXISTS customers;
```

- **Caution:** These commands can remove data in HDFS!

- Hive does not have a rollback or undo feature



Renaming Tables and Columns

- Use `ALTER TABLE` to modify a table's metadata
 - This command does not change data in HDFS

- Rename an existing table

```
hive> ALTER TABLE customers RENAME TO clients;
```

- Rename a column by specifying its old and new names
 - Type must be specified even though it is unchanged

```
hive> ALTER TABLE clients  
      CHANGE fname first_name STRING;
```

Old Name New Name Type



Modifying and Adding Columns

- You can also modify a column's type
 - The old and new column names will be the same
 - You must ensure data in HDFS conforms to the new type

```
hive> ALTER TABLE jobs CHANGE salary salary BIGINT;
```

| Old Name | New Name | Type |
|----------|----------|------|
|----------|----------|------|

- Add new columns to a table
 - Appended to the end of any existing columns
 - Existing data will have NULL values for new columns

```
hive> ALTER TABLE jobs  
ADD COLUMNS (city STRING, bonus INT);
```



Data Analysis with Hive

Introduction to HiveQL



Rajiv Garg



An Introduction To HiveQL

- HiveQL is Hive's query language
 - Based on a subset of SQL-92, plus Hive-specific extensions (found in MySQL and Oracle SQL dialects)
- Some limitations compared to 'standard' SQL
 - Some features are not supported
 - e.g. Updating or deleting individual records
 - Others are only partially implemented
 - Include joins on non-equality conditions
 - Correlated subqueries in a WHERE clause
 - <http://stackoverflow.com/questions/10710075/hive-column-as-a-subquery-select>
 - Views: Only non-materialized views are currently supported
- HiveQL also has some features not offered in SQL



HiveQL Basics

- Hive keywords are not case-sensitive
 - Though they are often capitalized by convention
- Statements are terminated by a semicolon
 - A statement may span multiple lines
- Comments begin with -- (double hyphen)
 - Only supported in Hive scripts
 - There are no multi-line comments in Hive

```
$ cat myscript.hql
```

```
SELECT cust_id, fname, lname FROM customers  
WHERE zipcode='60601';    -- downtown Chicago
```



Selecting Data from Hive Tables

- The **SELECT** statement retrieves data from Hive tables
 - Can specify an ordered list of individual columns

```
hive> SELECT cust_id, fname, lname FROM customers;
```

- An asterisk matches all columns in the table

```
hive> SELECT * FROM customers;
```



Limiting and Sorting Query Results

- The `LIMIT` clause sets the maximum number of rows returned

```
hive> SELECT fname, lname FROM customers LIMIT 10;
```

- Caution: no guarantee regarding which 10 results are returned

- Use `ORDER BY` for top-N queries
- The field(s) you `ORDER BY` must be selected

```
hive> SELECT cust_id, fname, lname FROM customers  
ORDER BY cust_id DESC LIMIT 10;
```



Using a WHERE Clause to Restrict Results

- WHERE clauses restrict rows to those matching specified criteria

- String comparisons are case-sensitive

```
hive> SELECT * FROM orders WHERE order_id=1287;
```

```
hive> SELECT * FROM customers WHERE state  
      IN ('CA', 'OR', 'WA', 'NV', 'AZ');
```

- You can combine expressions using AND or OR

```
hive> SELECT * FROM customers  
      WHERE fname LIKE 'Ann%'  
      AND (city='Seattle' OR city='Portland');
```



Table Aliases

- Table aliases can help simplify complex queries

```
hive> SELECT o.order_date, c.fname, c.lname  
      FROM customers c JOIN orders o  
      ON c.cust_id = o.cust_id  
      WHERE c.zipcode='94306';
```

- Note: Using **AS** to specify table aliases is not supported



Combining Query Results with UNION ALL

- Unifies output from SELECTs into a single result set
 - The name, order, and types of columns in each query must match
 - Hive only supports UNION ALL

```
SELECT emp_id, fname, lname, salary  
      FROM employees  
     WHERE state='CA' AND salary > 75000  
UNION ALL  
SELECT emp_id, fname, lname, salary  
      FROM employees  
     WHERE state != 'CA' AND salary > 50000;
```

- UNION ALL can also be used with subqueries



Subqueries in Hive

- Hive only supports subqueries in the FROM clause of the SELECT statement

- It does not support correlated subqueries

```
SELECT prod_id, brand, name
FROM (SELECT *
      FROM products
      WHERE (price - cost) / price > 0.65
      ORDER BY price DESC
      LIMIT 10) high_profits
WHERE price > 1000
ORDER BY brand, name;
```

- Hive allows arbitrary levels of subqueries

- Each subquery must be named (like high_profits above)



HiveQL

Relational Data Analysis



Rajiv Garg



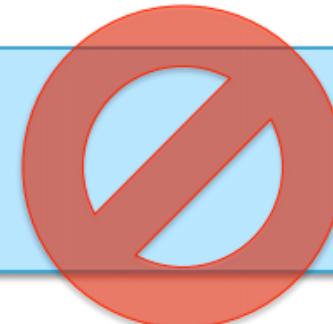
Join Syntax

- Hive requires the following syntax for joins

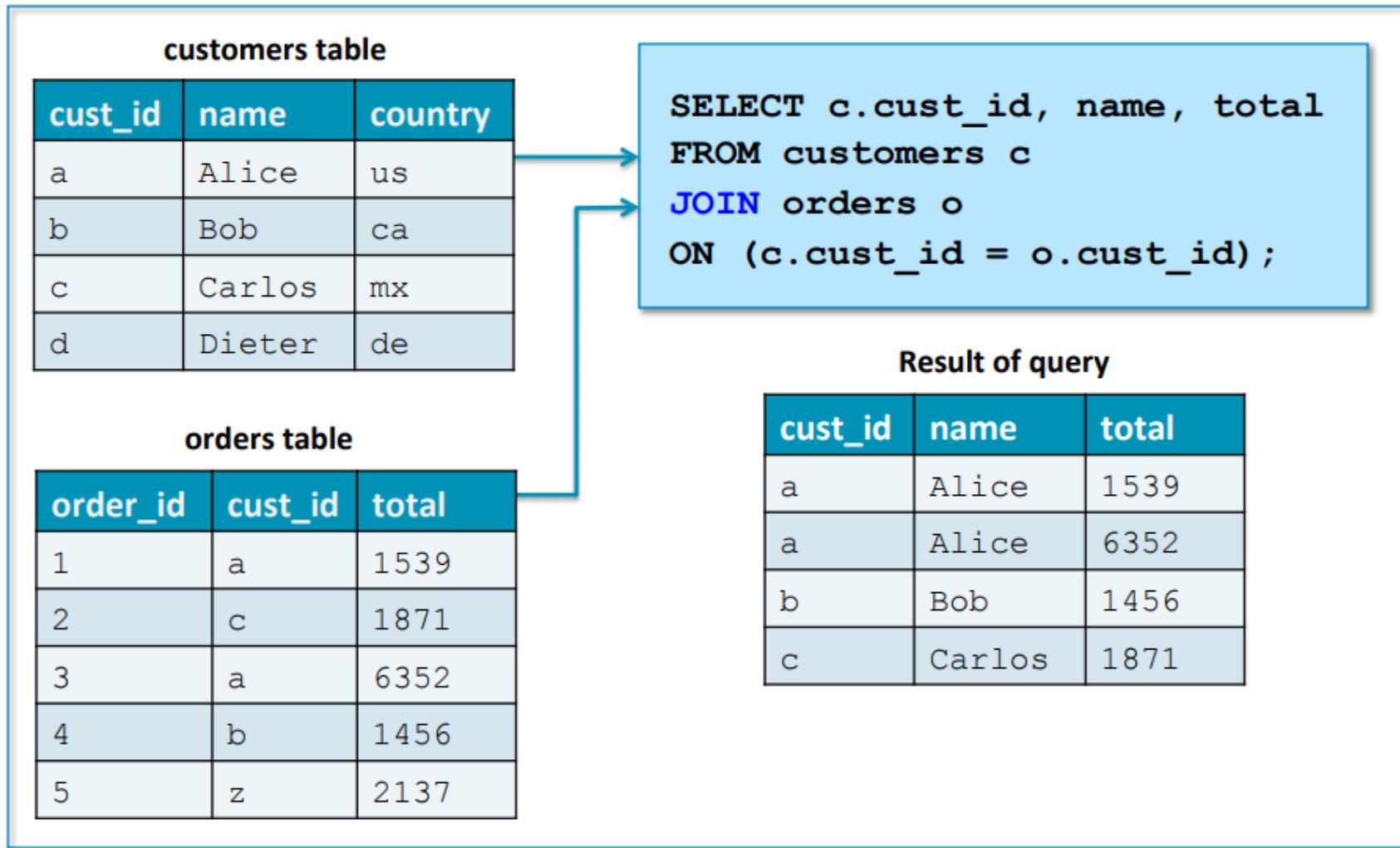
```
SELECT c.cust_id, name, total  
FROM customers c  
JOIN orders o ON (c.cust_id = o.cust_id);
```

- The above example is an inner join
 - Can replace JOIN with another type (e.g. RIGHT OUTER JOIN)
- The following join syntax is not valid in Hive

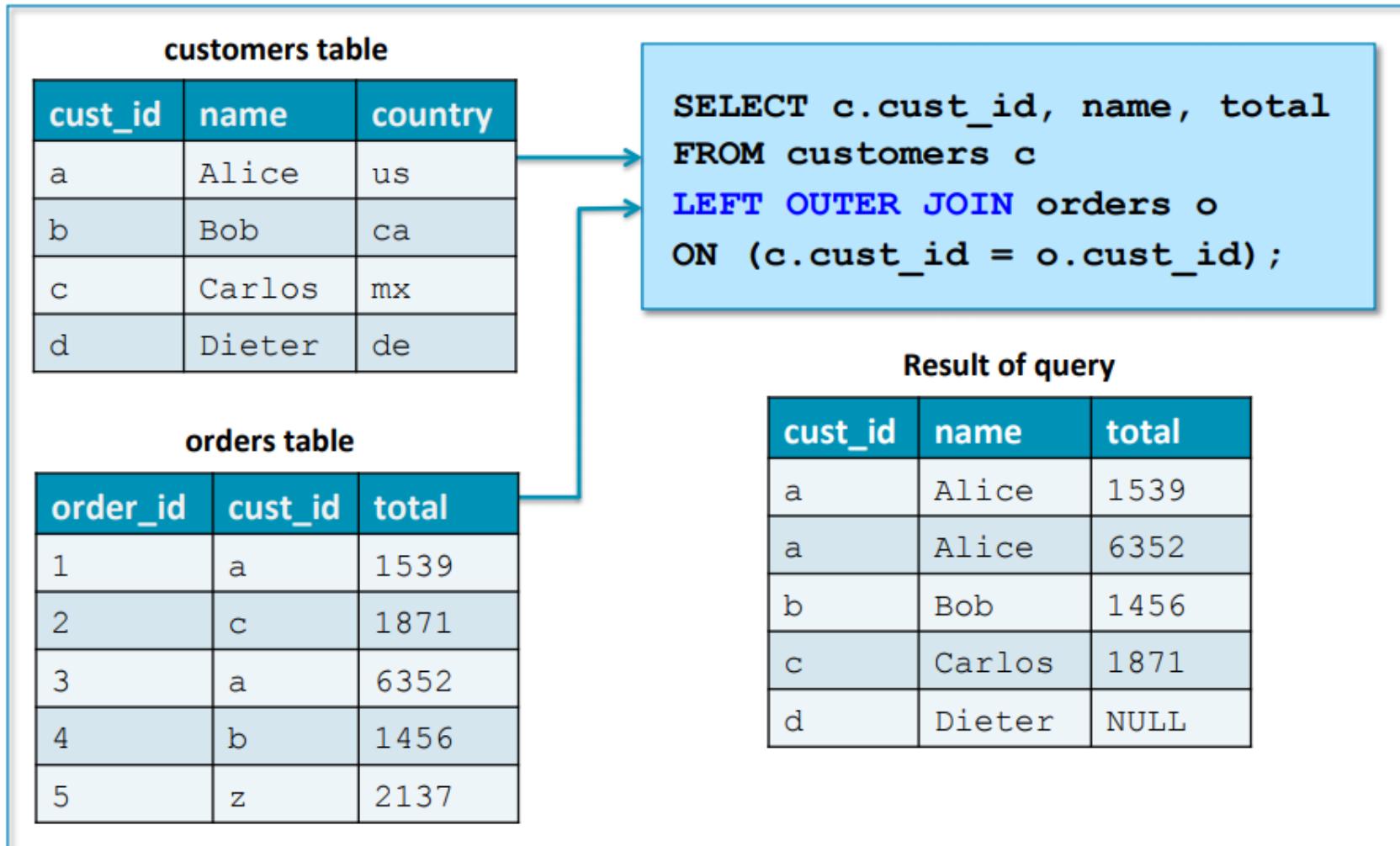
```
SELECT c.cust_id, name, total  
FROM customers c, orders o  
WHERE (c.cust_id = o.cust_id);
```



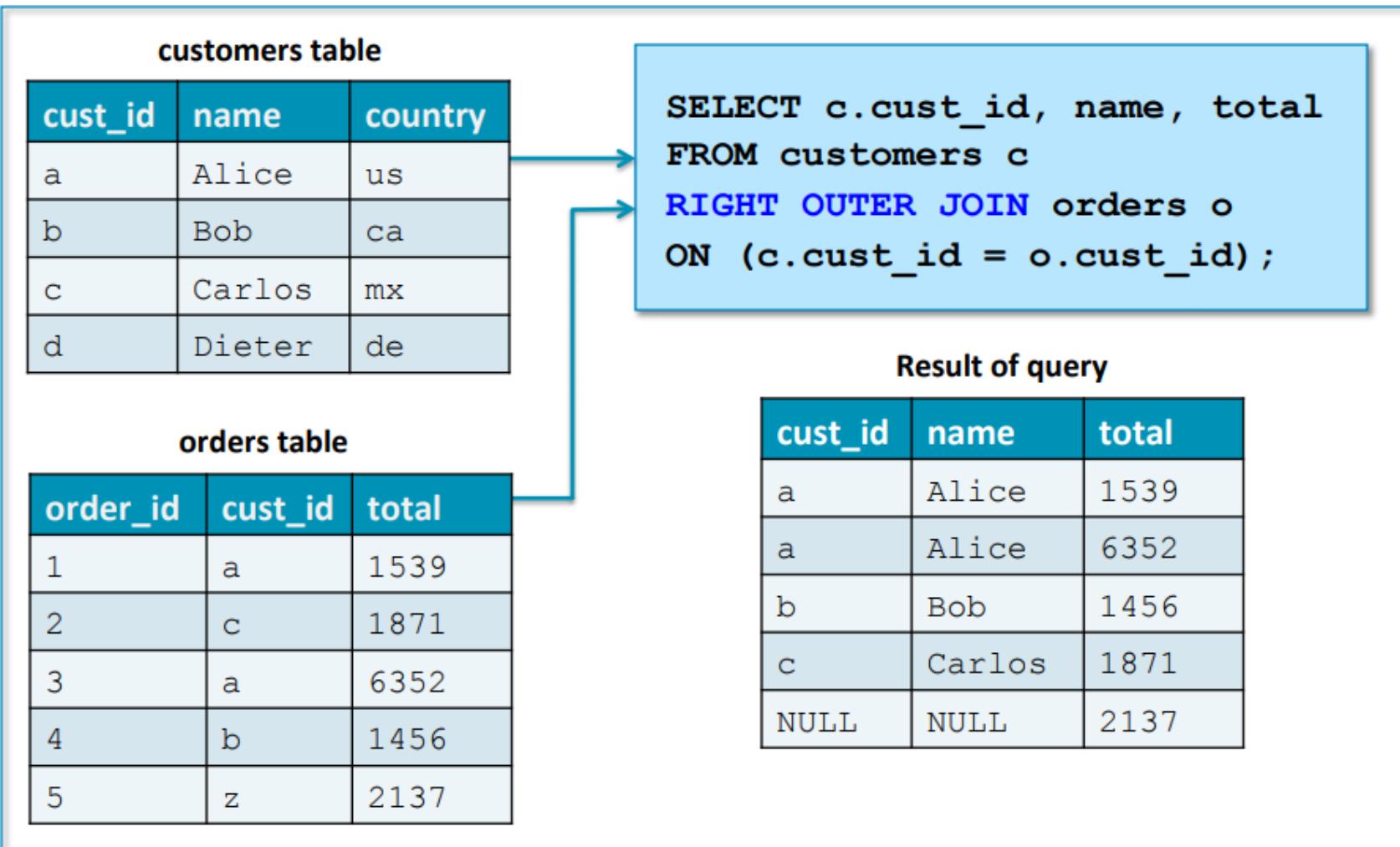
Inner Join Example



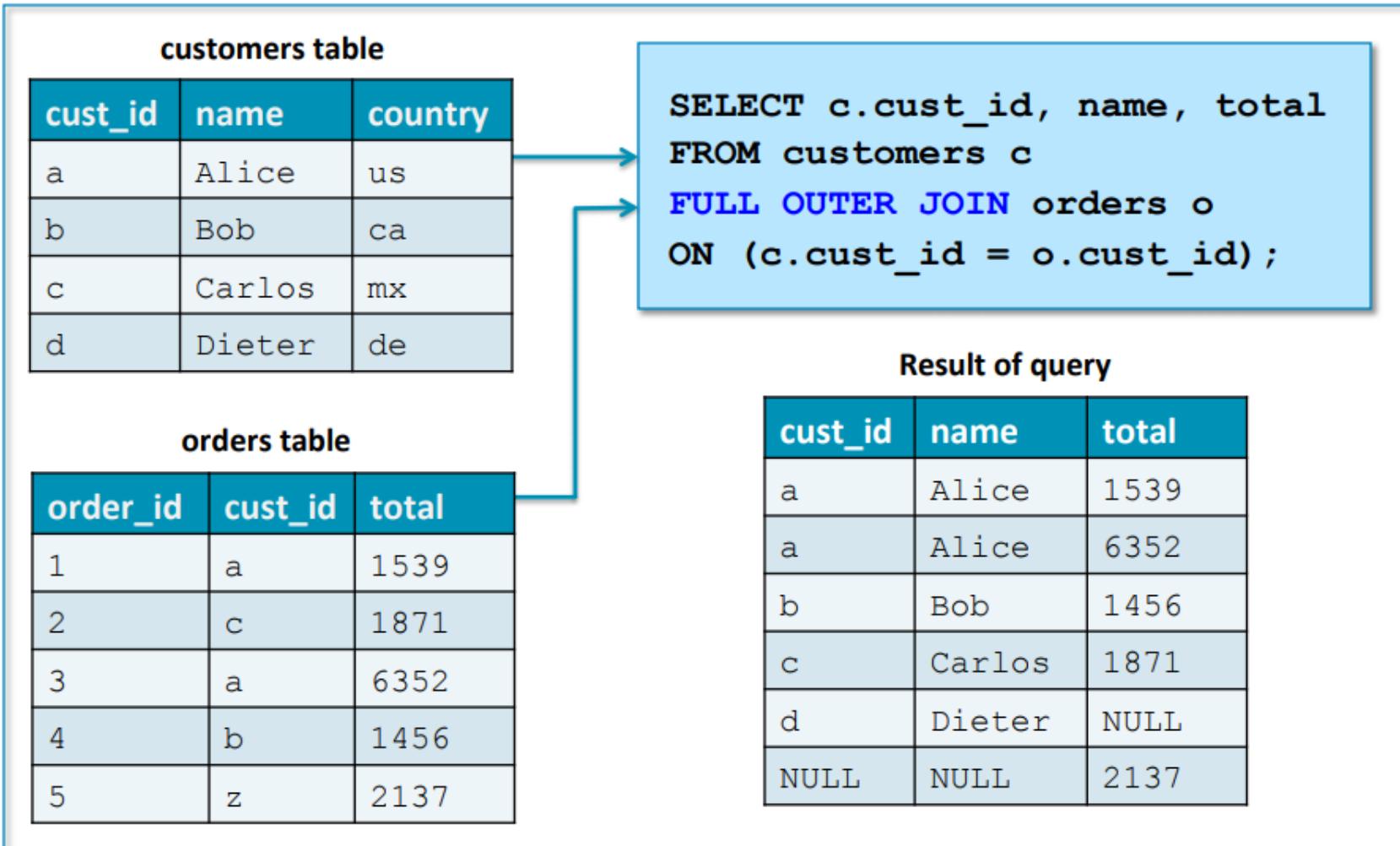
Left Outer Join Example



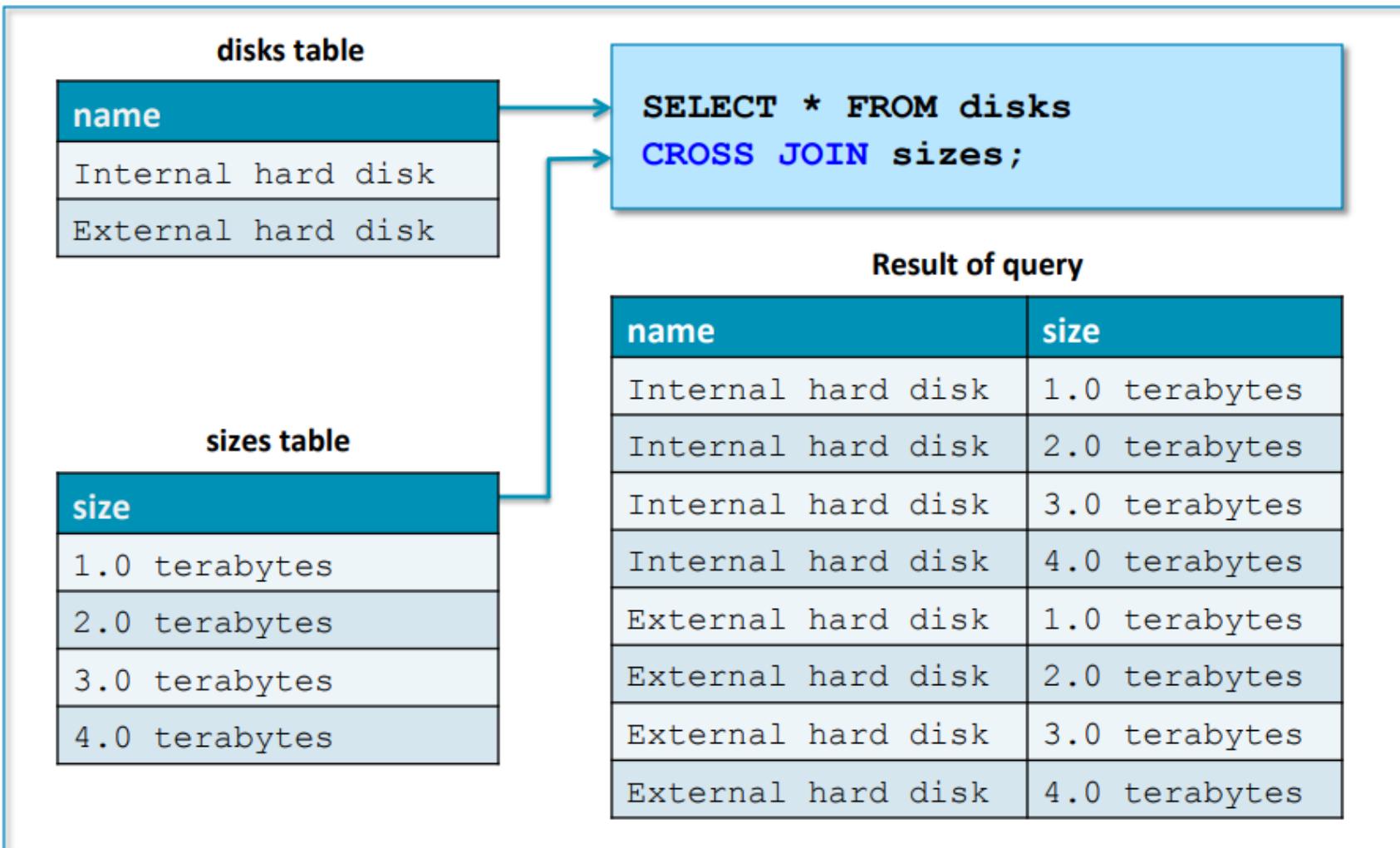
Right Outer Join Example



Full Outer Join Example



Cross Join Example



Left Semi Joins

- A less common type of join is the LEFT SEMI JOIN
 - They are a special (and efficient) type of inner join
 - They behave more like a filter than a join
- Left semi joins include additional criteria in the ON clause
 - Only unique records that match these criteria are returned
 - Fields listed in SELECT are limited to the left-side table

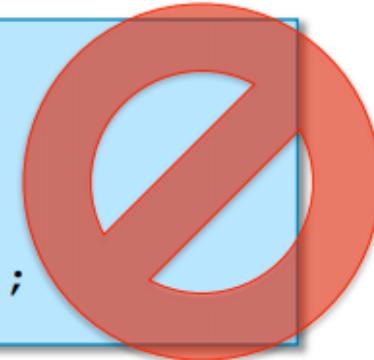
```
SELECT c.cust_id  
FROM customers c  
LEFT SEMI JOIN orders o  
ON (c.cust_id = o.cust_id  
    AND YEAR(o.order_date) = '2012');
```



Left Semi Joins

- Hive does not support IN/EXISTS subqueries

```
SELECT c.cust_id FROM customers c
WHERE o.cust_id IN
  (SELECT o.cust_id FROM orders o
   WHERE YEAR(o.order_date) = '2012'));
```



- Using a LEFT SEMI JOIN is a common workaround

```
SELECT c.cust_id
FROM customers c
LEFT SEMI JOIN orders o
ON (c.cust_id = o.cust_id
  AND YEAR(o.order_date) = '2012');
```



HiveQL

Common Built-In Functions



Rajiv Garg



Hive Functions

- Hive offers dozens of built-in functions
 - Many are identical to those found in SQL
 - Others are Hive-specific
- Example function invocation
 - Function names are not case-sensitive
 - ```
hive> SELECT CONCAT(fname, ' ', lname) AS fullname
 FROM customers;
```
- To see information about a function
  - ```
hive> DESCRIBE FUNCTION UPPER;
      UPPER(str) - Returns str with all characters changed to uppercase
```



Example Built-in Functions

- These functions operate on numeric values

| Function Description | Example Invocation | Input | Output |
|-----------------------------------|-----------------------|--------|----------|
| Rounds to specified # of decimals | ROUND(total_price, 2) | 23.492 | 23.49 |
| Returns nearest integer above | CEIL(total_price) | 23.492 | 24 |
| Returns nearest integer below | FLOOR(total_price) | 23.492 | 23 |
| Return absolute value | ABS(temperature) | -49 | 49 |
| Returns square root | SQRT(area) | 64 | 8 |
| Returns a random number | RAND() | | 0.584977 |



Example Built-in Functions

- These functions operate on timestamp values

| Function Description | Example Invocation | Input | Output |
|---------------------------------|-----------------------------|---------------------------|------------------------|
| Convert to UNIX format | UNIX_TIMESTAMP(order_dt) | 2013-06-14
16:51:05 | 1371243065 |
| Convert to string format | FROM_UNIXTIME(mod_time) | 1371243065 | 2013-06-14
16:51:05 |
| Extract date portion | TO_DATE(order_dt) | 2013-06-14
16:51:05 | 2013-06-14 |
| Extract year portion | YEAR(order_dt) | 2013-06-14
16:51:05 | 2013 |
| Returns # of days between dates | DATEDIFF(order_dt, ship_dt) | 2013-06-14,
2013-06-17 | 3 |



Example Built-in Functions

- Here are some other useful functions

| Function Description | Example Invocation | Input | Output |
|------------------------------|----------------------------|-------|--------|
| Converts to uppercase | UPPER(fname) | Bob | BOB |
| Extract portion of string | SUBSTRING(name, 0, 2) | Alice | Al |
| Selectively return value | IF(price > 1000, 'A', 'B') | 1500 | A |
| Convert to another type | CAST(weight as INT) | 3.581 | 3 |
| Returns size of array or map | SIZE(array_field) | N/A | 6 |



HiveQL

Aggregation and Windowing

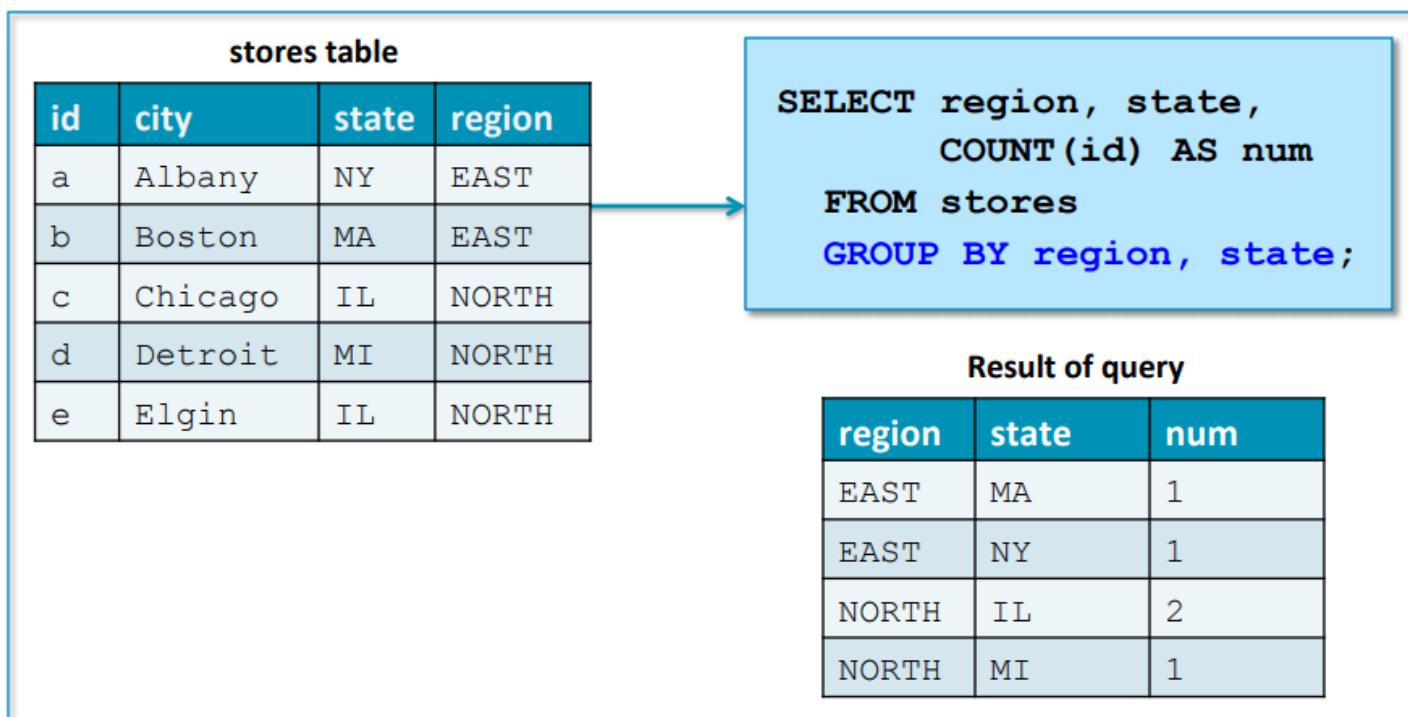


Rajiv Garg



Record Grouping and Aggregate Functions

- GROUP BY groups selected data by one or more columns
 - Caution: Columns not part of aggregation must be listed in GROUP BY



Built-in Aggregate Functions

- Hive offers many aggregate functions, including

| Function Description | Example Invocation |
|---|------------------------|
| Count all rows | COUNT (*) |
| Count all rows where field is not null | COUNT (fname) |
| Count all rows where field is unique and not null | COUNT (DISTINCT fname) |
| Returns the largest value | MAX (salary) |
| Returns the smallest value | MIN (salary) |
| Adds all supplied values and returns result | SUM (price) |
| Returns the average of all supplied values | AVG (salary) |



Window Aggregation

- Standard aggregation groups sets of rows together into single rows
 - Individual rows are not preserved in the output
- Windowing applies a function over sets of rows without combining them
 - Individual rows are preserved



Windows

- A window defines a set of rows in a table
- `OVER(window-specification)` specifies the windows over which to apply an aggregation or windowing function
- Example: `OVER(PARTITION BY brand)`

| prod_id | brand | name | price |
|---------|----------|-----------------|-------|
| 1 | Dualcore | USB Card Reader | 18.39 |
| 2 | Dualcore | HDMI Cable | 11.99 |
| 3 | Dualcore | VGA Cable | 1.99 |
| 4 | Gigabux | 6-cell Battery | 40.50 |
| 5 | Gigabux | 8-cell Battery | 50.50 |
| 6 | Gigabux | Wall Charger | 20.00 |
| 7 | Gigabux | Auto Charger | 20.00 |

The diagram illustrates the concept of windows in a database query. It shows a table of product data with four columns: prod_id, brand, name, and price. The data is partitioned into two windows based on the brand. The first window, labeled 'Window', contains rows 1, 2, and 3, all belonging to the 'Dualcore' brand. The second window, also labeled 'Window', contains rows 4, 5, 6, and 7, all belonging to the 'Gigabux' brand.



Example: RANK and ROW_NUMBER

- Question: Rank the products by price within each brand

```
SELECT prod_id, brand, price,  
       RANK() OVER(PARTITION BY brand ORDER BY price) AS rank,  
       ROW_NUMBER() OVER(PARTITION BY brand ORDER BY price) AS n  
FROM products;
```

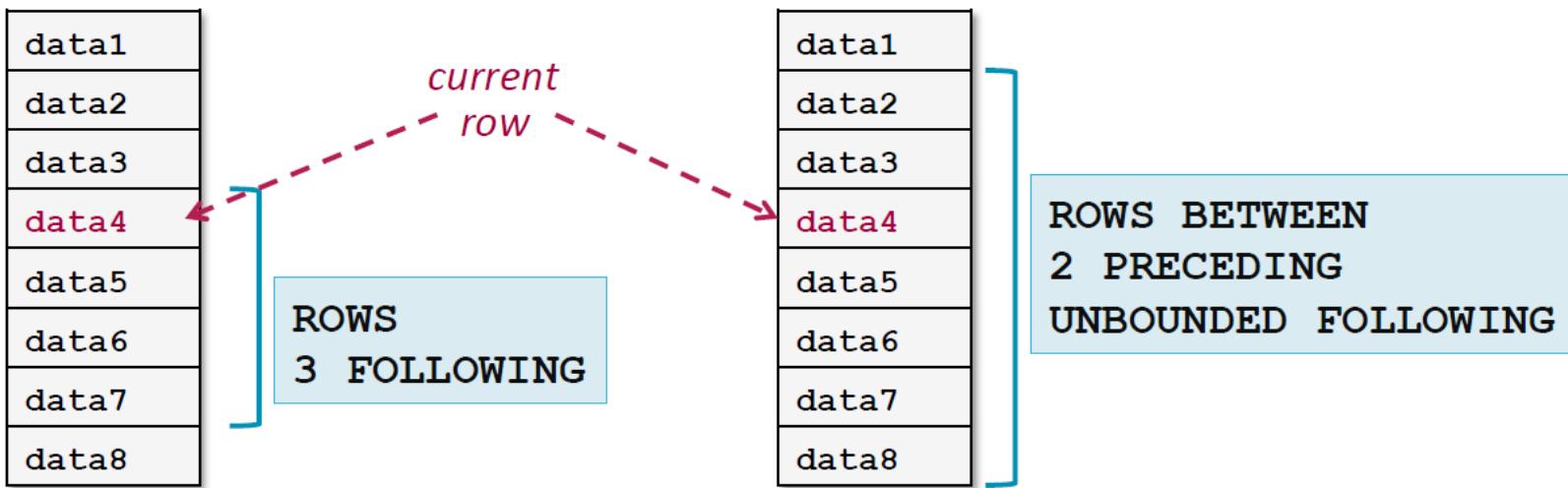
| products table | | | |
|----------------|----------|-----------------|-------|
| prod_id | brand | name | price |
| 1 | Dualcore | USB Card Reader | 18.39 |
| 2 | Dualcore | HDMI Cable | 11.99 |
| 3 | Dualcore | VGA Cable | 1.99 |
| 4 | Gigabux | 6-cell Battery | 40.50 |
| 5 | Gigabux | 8-cell Battery | 50.50 |
| 6 | Gigabux | Wall Charger | 20.00 |
| 7 | Gigabux | Auto Charger | 20.00 |

| query results | | | | |
|---------------|----------|-------|------|---|
| prod_id | brand | price | rank | n |
| 3 | Dualcore | 1.99 | 1 | 1 |
| 2 | Dualcore | 11.99 | 2 | 2 |
| 1 | Dualcore | 18.39 | 3 | 3 |
| 6 | Gigabux | 20.00 | 1 | 1 |
| 7 | Gigabux | 20.00 | 1 | 2 |
| 4 | Gigabux | 40.50 | 3 | 3 |
| 5 | Gigabux | 50.50 | 4 | 4 |



Sliding Windows

- There are three optional parts of a window specification
 - Partitioning – PARTITION BY
 - Ordering – ORDER BY (ASC or DESC)
 - Frame boundaries – ROWS or RANGE
- Frame boundaries specify a sliding window relative to the current row



Example: Time-Based Sliding Window

- Question: What is the per-website average visit count for the week ending today?

```
SELECT display_date, display_site, n,
       AVG(n) OVER
          (PARTITION BY display_site ORDER BY display_date
           ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS wavg
  FROM (
    SELECT display_date, display_site, COUNT(display_date) AS n
      FROM ads GROUP BY display_date, display_site) ads
 ORDER BY display_date, display_site;
```

| display_date | display_site | n | wavg |
|--------------|--------------|---|------|
| 2016-05-01 | audiophile | 1 | 1 |
| 2016-05-01 | photosite | 2 | 2 |
| 2016-05-02 | audiophile | 3 | 2 |
| 2016-05-02 | dvdreview | 1 | 1 |
| 2016-05-03 | audiophile | 1 | 1.66 |



Hive

Storing Query Results



Rajiv Garg



Creating Tables Based On Existing Data

- Hive supports creating a table based on a SELECT statement
 - Often known as ‘Create Table As Select’ (CTAS)

```
CREATE TABLE ny_customers AS  
    SELECT cust_id, fname, lname FROM customers  
    WHERE state = 'NY';
```

- Column definitions are derived from the existing table
- Column names are inherited from the existing names
 - Use aliases in the SELECT statement to specify new names
- The newly created table is automatically added to the Hive metastore



Saving Query Output to a Table

- SELECT statements display their results on screen
- To send results to a Hive table instead, use **INSERT OVERWRITE TABLE**
 - Destination table (with the same schema) must already exist
 - Existing contents will be deleted
- **INSERT INTO TABLE** adds records without first deleting existing data
 - In other words, appends rather than overwrites

```
hive> INSERT OVERWRITE TABLE ny_customers  
      SELECT * FROM customers  
      WHERE state = 'NY';
```

- **INSERT INTO TABLE** adds records without first deleting existing data
 - In other words, appends rather than overwrites

```
hive> INSERT INTO TABLE ny_customers  
      SELECT * FROM customers  
      WHERE state = 'NJ' OR state = 'CT';
```



Writing Output to a Filesystem

- You can save output to a file in HDFS

```
hive> INSERT OVERWRITE DIRECTORY '/dualcore/ny/'  
      SELECT * FROM customers  
      WHERE state = 'NY';
```

- Add **LOCAL** to store results to local disk instead

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/home/bob/ny/'  
      SELECT * FROM customers  
      WHERE state = 'NY';
```

- Both produce text files delimited by Ctrl-A characters, regardless of the delimiter used in the table itself



Hive

Simplifying Queries with Views



Rajiv Garg



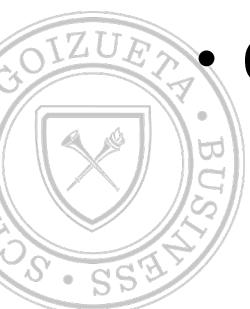
Creating Views

- Views in Hive are conceptually like a table, but backed by a query
 - You cannot directly add data to a view

```
CREATE VIEW order_info AS
SELECT o.order_id, order_date, p.prod_id, brand, name
FROM orders o
JOIN order_details d
ON (o.order_id = d.order_id)
JOIN products p
ON (d.prod_id = p.prod_id);
```

- Our query is now greatly simplified

```
hive> SELECT * FROM order_info WHERE order_id=6584288;
```



MySQL & Python

SQL & Excel/Python

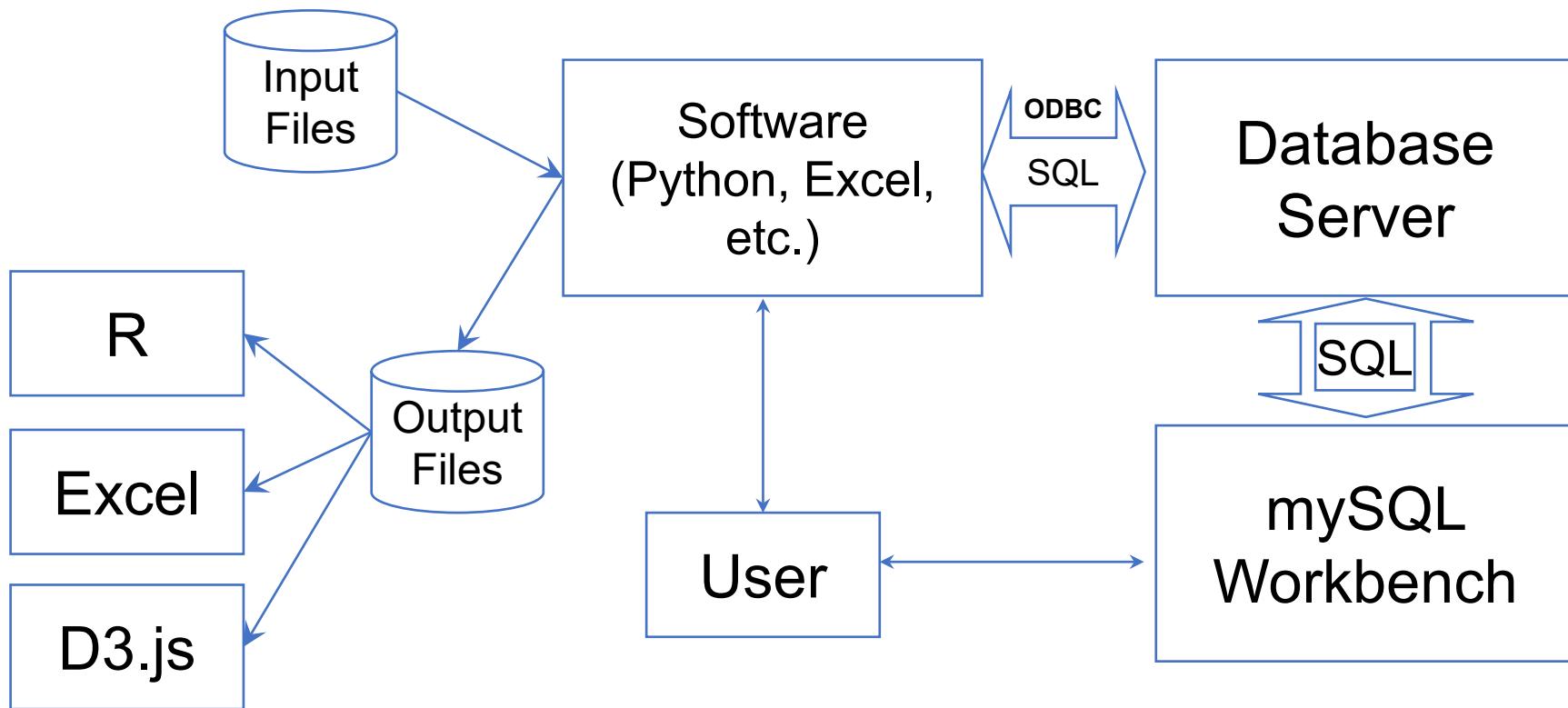


Image source: Charles Severance (py4e.com)



Accessing Databases

- Install ODBC connector:
 - <https://dev.mysql.com/downloads/connector/odbc/>
- OR
- pip install pyodbc:
 - <https://pypi.org/project/pyodbc/>
- ODBC: Open Database Connectivity
 - Microsoft introduced the ODBC standard in 1992.
 - ODBC is a specification for a database API that is independent of any one DBMS or operating system.
 - “The functions in the ODBC API are implemented by developers of DBMS-specific drivers. Applications call the functions in these drivers to access data in a DBMS-independent manner.”
 - <https://docs.microsoft.com/en-us/sql/odbc/reference/what-is-odbc>

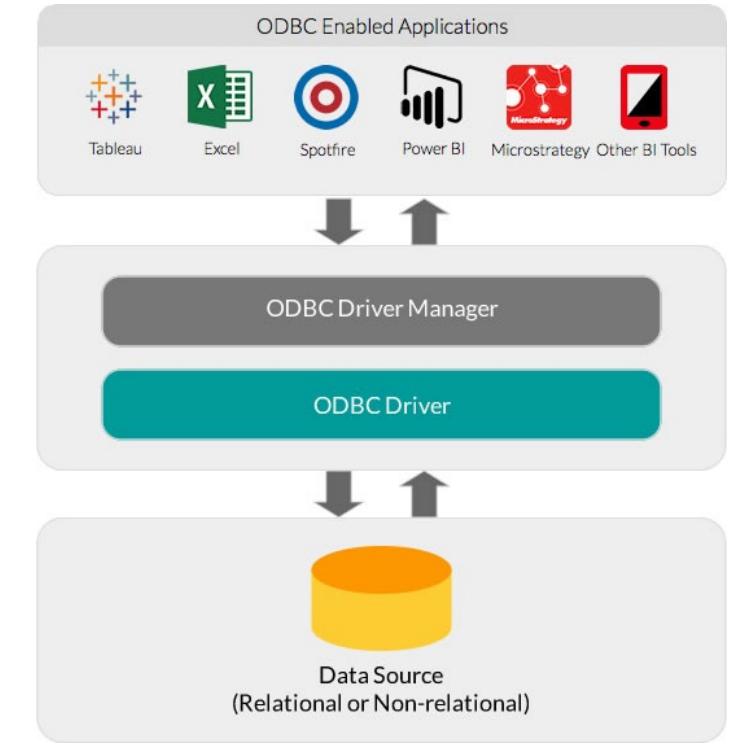


Image source: <https://www.simba.com/resources/odbc/>



mySQL Connector

- `pip install mysql-connector-python`
 - Specific to mySQL
 - Works when you have different version of Python (x86) and mySQL (x64)



Reading mySQL Data

- Using ODBC connection to read databases

```
import mysql.connector  
  
mydb = mysql.connector.connect(  
    host="localhost",  
    user="root",  
    password=open("mysql_pass.txt", "r").readline() #SAVE PASSWORD IN A FILE  
)  
  
cursor = conn.cursor()  
cursor.execute('select * from products')  
  
for row in cursor.fetchall():  
    print (row)  
  
(1, 'apple', 3.5, 4.99, 100.0)  
(2, 'orange', 0.87, 1.49, 200.0)  
(3, 'milk', 2.68, 3.99, 300.0)  
(4, 'bread', 1.41, 1.99, 400.0)  
(5, 'beer', 9.9, 19.99, 100.0)  
(6, 'diaper', 18.21, 29.99, 50.0)
```

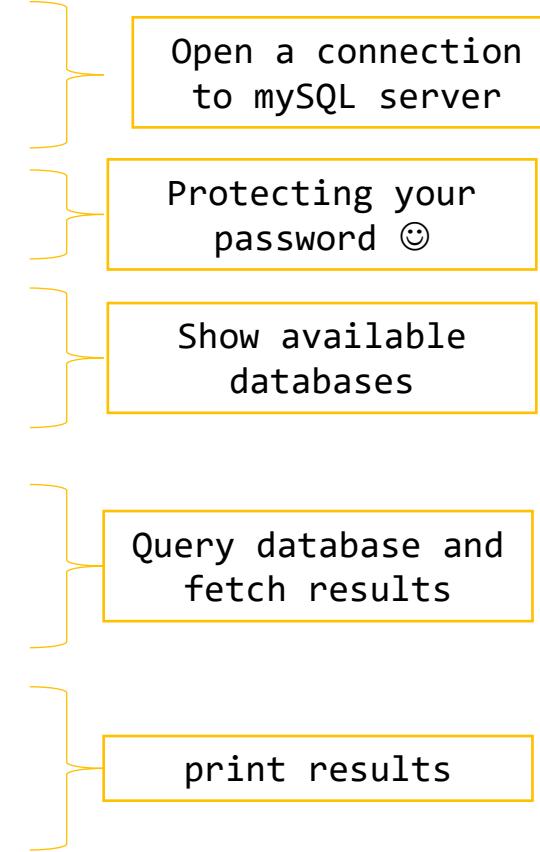
Mysql connector

SQL query to execute



Reading mySQL Data

```
• import mysql.connector  
• mydb = mysql.connector.connect(  
•     host="localhost",  
•     user="root",  
•     password=open("mysql_pass.txt", "r").readline()  
• )  
• mycursor = mydb.cursor()  
• mycursor.execute("SHOW databases")  
• print("AVAILABLE DATABASES:\n", mycursor.fetchall())  
  
• mycursor.execute("USE sakila")  
• mycursor.execute("SELECT * FROM customer")  
• myresult = mycursor.fetchall()  
  
• print("RESPONSE TO QUERY:")  
• for x in myresult:  
    print(x)
```

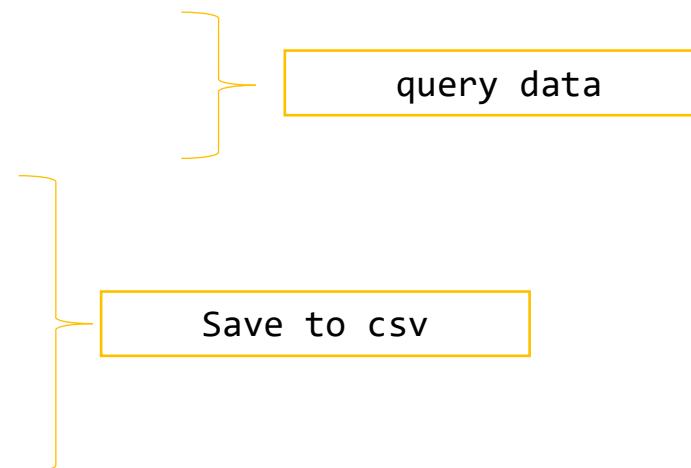


Writing Query output to CSV

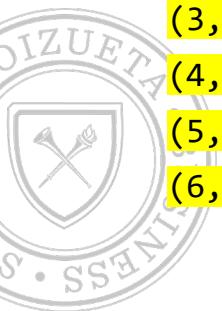
```
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
```

```
fname = "query_out.csv"
fhandle = open(fname, "w", newline="")
csvfile = csv.writer(fhandle)
 csvfile.writerows(myresult)
fhandle.close()
```

```
(1, 'apple', 3.5, 4.99, 100.0)
(2, 'orange', 0.87, 1.49, 200.0)
(3, 'milk', 2.68, 3.99, 300.0)
(4, 'bread', 1.41, 1.99, 400.0)
(5, 'beer', 9.9, 19.99, 100.0)
(6, 'diaper', 18.21, 29.99, 50.0)
```



| | | | |
|--------|-------|-------|-----|
| apple | 3.50 | 4.99 | 922 |
| orange | 0.87 | 1.49 | 432 |
| milk | 2.68 | 3.99 | 761 |
| bread | 1.41 | 1.99 | 930 |
| beer | 9.90 | 19.99 | 883 |
| diaper | 18.21 | 29.99 | 364 |



Example - JOIN

- Extract shipping addresses for all customers and save the following columns: first name, last name, address, city, state, zip

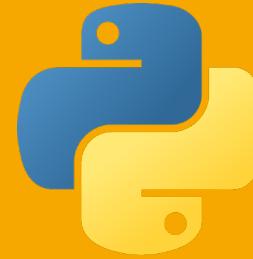
```
SELECT first_name, last_name, line1, city, state, zip_code  
FROM customers c JOIN addresses a  
    ON c.customer_id = a.customer_id  
    AND c.shipping_address_id = a.address_id
```

```
query = 'SELECT first_name, last_name, line1, city, state, zip_code '\  
       'FROM customers c JOIN addresses a '\  
       'ON c.customer_id = a.customer_id '\  
       'AND c.shipping_address_id = a.address_id'  
getData(query)
```

\: allows string to be broken over multiple lines

Remember to leave a space at the end, else it will join the last word with the first word in next line

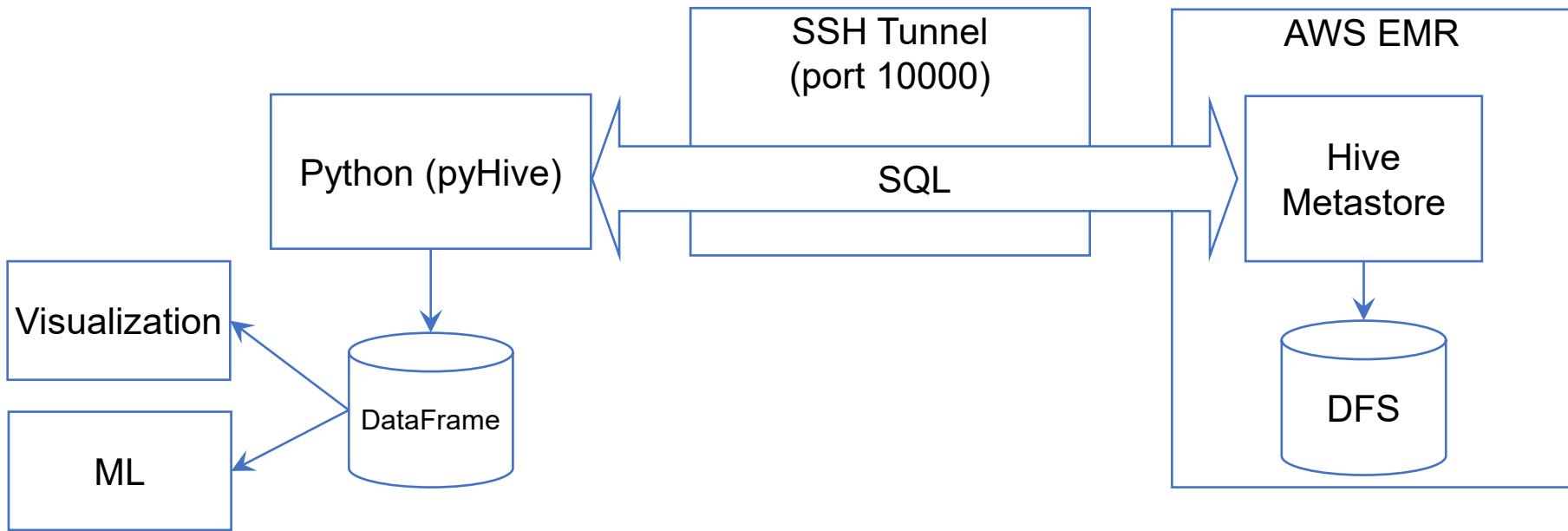




Hive + Python

Data Processing in Hadoop

AWS EMR Hive + pyHive



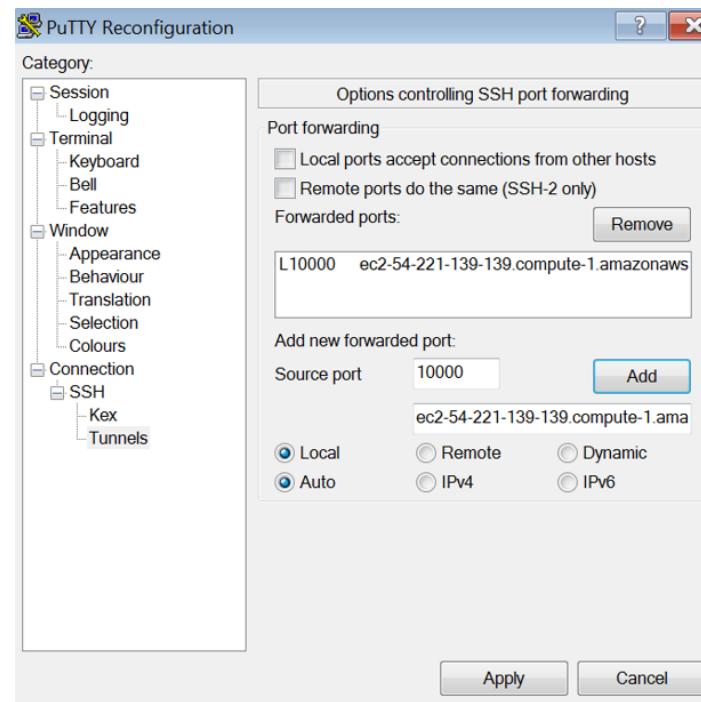
Install Python Libraries

- PyHive is a collection of Python database API for Hive: <https://pypi.org/project/PyHive/>
 - `pip install pyhive`
- Apache Thrift software framework, for scalable cross-language services development:
<https://pypi.org/project/thrift/>
 - `pip install thrift`
- Simple Authentication Security Layer (SASL) provides developers of applications and shared libraries with mechanisms for authentication, data integrity-checking, and encryption:
<https://pypi.org/project/sasl/>
 - `pip install sasl`
 - NOTE: usually doesn't work well on newer (3.x, x≤10) versions of Python. You need the pre-compiled build wheel:
<https://www.lfd.uci.edu/~gohlke/pythonlibs/#sasl>
 - `pip install thrift_sasl`



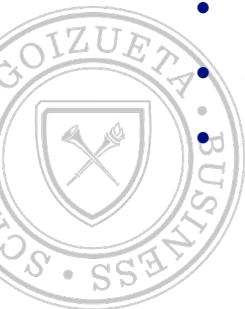
SSH Setup

- Setup SSH connection to EMR with tunnel for port 10000
 - <https://aws.amazon.com/blogs/big-data/using-amazon-emr-with-sql-workbench-and-other-bi-tools/>



Connect to Hive

```
• import pandas as pd  
• from pyhive import hive  
  
• HIVE_URL = "localhost"  
• HIVE_PORT = "10000"  
• HIVE_USER = "hadoop"  
• conn = hive.Connection(host=HIVE_URL,  
                         port=HIVE_PORT,  
                         username=HIVE_USER)  
• cur = conn.cursor()  
  
• cur.execute("SELECT * FROM divs LIMIT 5")  
• df = pd.DataFrame(cur.fetchall())  
• conn.close()
```

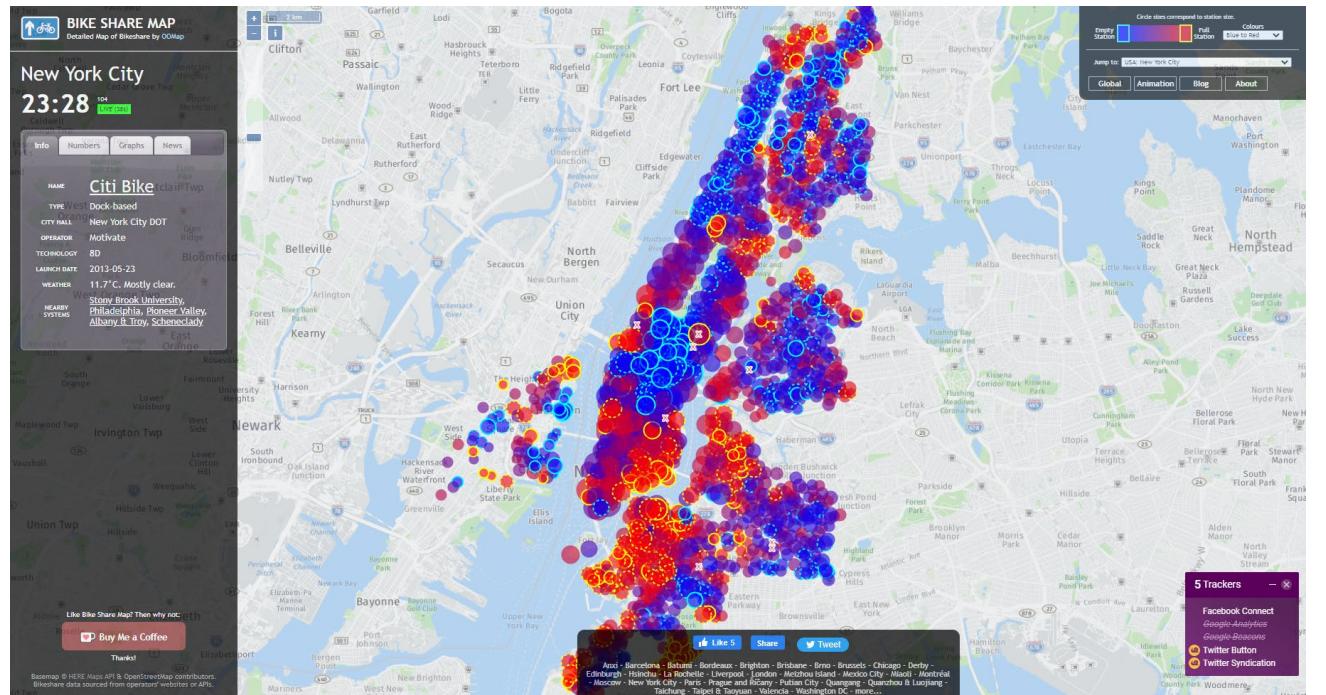


Hive+Python+Analytics



NYC Citi Bike

- “Citi Bike is the nation's largest bike share program, with 20,000 bikes and over 1,300 stations across Manhattan, Brooklyn, Queens, the Bronx and Jersey City. It was designed for quick trips with convenience in mind, and it's a fun and affordable way to get around town.” <https://ride.citibikenyc.com/>
- Data source:
 - <https://ride.citibikenyc.com/system-data>
 - Ride_id, station_info (start/stop), time_info (start/stop), ride type, and customer type
- Map:
 - <https://bikesharemap.com/newyork/>



Citibike Data

- Data: <https://s3.amazonaws.com/tripdata/201306-citibike-tripdata.zip>
- Insights:
 - Gender distribution
 - Age distribution
 - Membership distribution
 - Trip durations
 - Popular starting stations
 - Popular ending stations



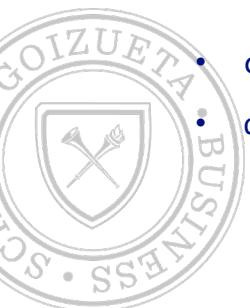
Citibike (load data in mySQL)

- create schema citibike;
- use citibike;
- CREATE TABLE tripdata202209 (
 - `ride_id` text,
 - `rideable_type` text,
 - `started_at` text,
 - `ended_at` text,
 - `start_station_name` text,
 - `start_station_id` double,
 - `end_station_name` text,
 - `end_station_id` double,
 - `start_lat` double,
 - `start_lng` double,
 - `end_lat` double,
 - `end_lng` double,
 - `member_casual` text);
- LOAD DATA LOCAL INFILE 'D:/citibike/202209-citibike-tripdata.csv'
- into table tripdata202209
- fields terminated by ","
- optionally enclosed by '\"'
- lines terminated by "\r\n"
- ignore 1 lines;



Citibike (SQL-Pandas)

- Read data from mySQL Database into Pandas
- `import mysql.connector`
- `conn = mysql.connector.connect(`
- `host="isom352.cqxikovybdnm.us-east-2.rds.amazonaws.com",`
- `database="citibike",`
- `user="admin",`
- `password="ISOM352db"`
- `)`
- `cursor = conn.cursor()`
- `cursor.execute('select * from tripdata202209 limit 10;')`
- `df = pd.DataFrame(cursor.fetchall())`



Citibike (HIVE-QL)

```
• import pandas as pd  
• from pyhive import hive  
  
• HIVE_URL = "localhost"  
• HIVE_PORT = "10000"  
• HIVE_USER = "hadoop"  
• conn = hive.Connection(host=HIVE_URL, port=HIVE_PORT, username=HIVE_USER)  
• cursor = conn.cursor()  
  
• cursor.execute('select * from tripdata202209 limit 10;')  
• df = pd.DataFrame(cursor.fetchall())
```



Visualizations - Plotly

- Plotly creates, manipulates and renders graphical figures (i.e. charts, plots, maps and diagrams) represented by data structures.
- Figures can be represented in Python as dictionaries and are serialized as text in JavaScript Object Notation (JSON) before being passed to Plotly.js (underlying graphing engine).
- To use plotly:
 - pip install plotly
 - import plotly.express as px



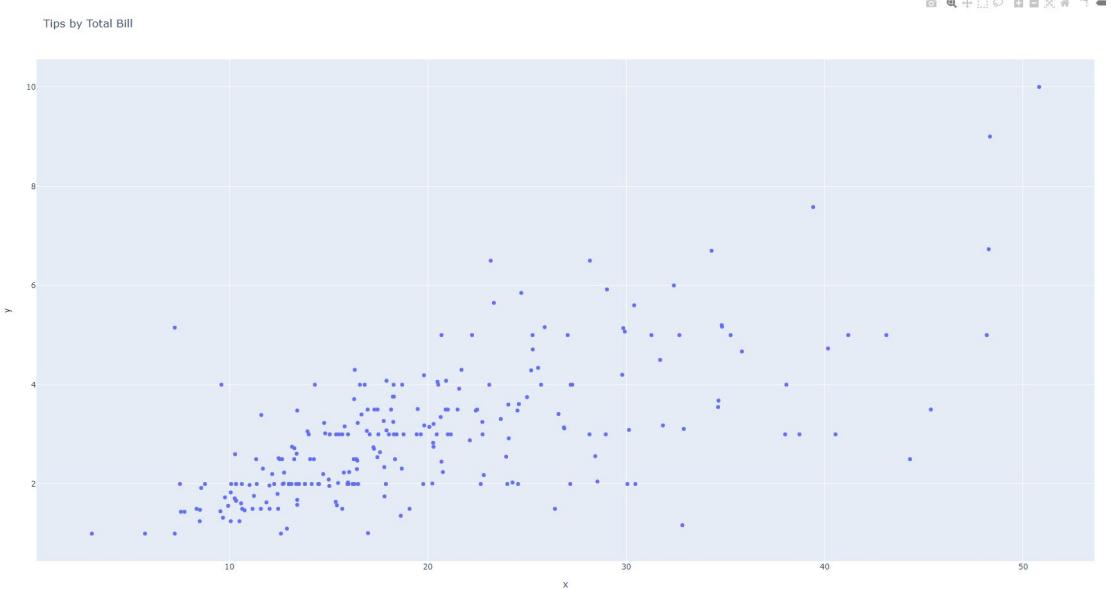
Plotly IO

- Plotly shows interactive graphs in browser. Many IDEs dont support the interactive graphs and need explicit command to show graphs in the browser.
- `import plotly.io as pio`
- `pio.renderers.default='browser'`
- `fig.show()`
- `pio.renderers.default='svg'`



Plotly - Scatter Plot

```
df = sns.load_dataset('tips')
fig = px.scatter(x=df['total_bill'], y=df['tip'])
print(fig)
fig.show()
```



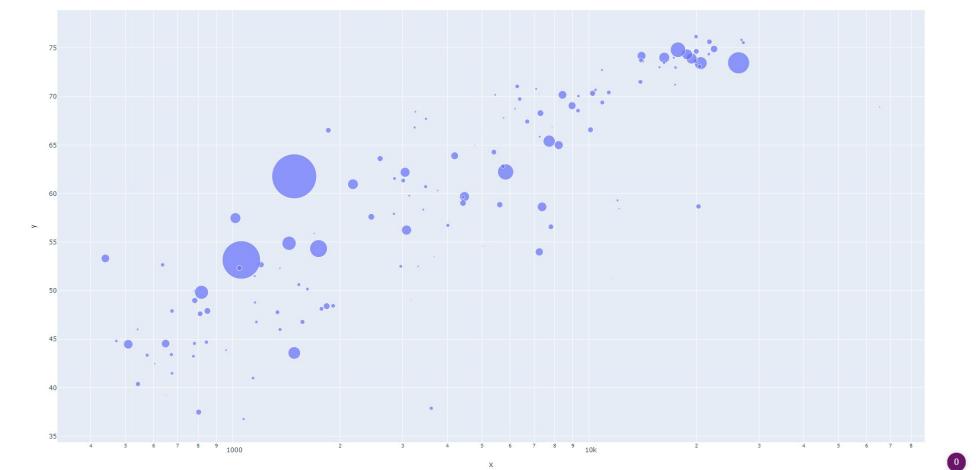
```
type(fig)
<class 'plotly.graph_objs._figure.Figure'>

Figure({
    'data': [{ 'hovertemplate':
      'x=%{x}<br>y=%{y}</extra></extra>',
      'legendgroup': '',
      'marker': { 'color': '#636efa', 'symbol':
      'circle'}, 'mode': 'markers',
      'name': '',
      'orientation': 'v',
      'showlegend': False,
      'type': 'scatter',
      'x': array([16.99, 10.34, 21.01, ..., 22.67,
      17.82, 18.78]), 'xaxis': 'x',
      'y': array([1.01, 1.66, 3.5 , ..., 2. ,
      1.75, 3. ]), 'yaxis': 'y'}],
    'layout': { 'legend': { 'tracegroupgap': 0},
      'template': '...', 'title': { 'text': 'Tips by Total Bill'},
      'xaxis': { 'anchor': 'y', 'domain': [0.0, 1.0], 'title': { 'text': 'x'}},
      'yaxis': { 'anchor': 'x', 'domain': [0.0, 1.0], 'title': { 'text': 'y'}}})
```



Plotly - Bubble

- df = pd.read_csv("../datasets/gapminder.tsv", sep='\t')
- dfmean = df.groupby('country')[['gdpPercap', 'lifeExp', 'pop']].mean().reset_index().sort_values('gdpPercap', ascending = False)
- fig = px.scatter(x=dfmean["gdpPercap"], y=dfmean["lifeExp"], size=dfmean["pop"], hover_name=dfmean["country"], log_x=True, size_max=60)



Scatter plot with bubble size represented by 'population'

Recall: log-scale



Plotly - Bubble

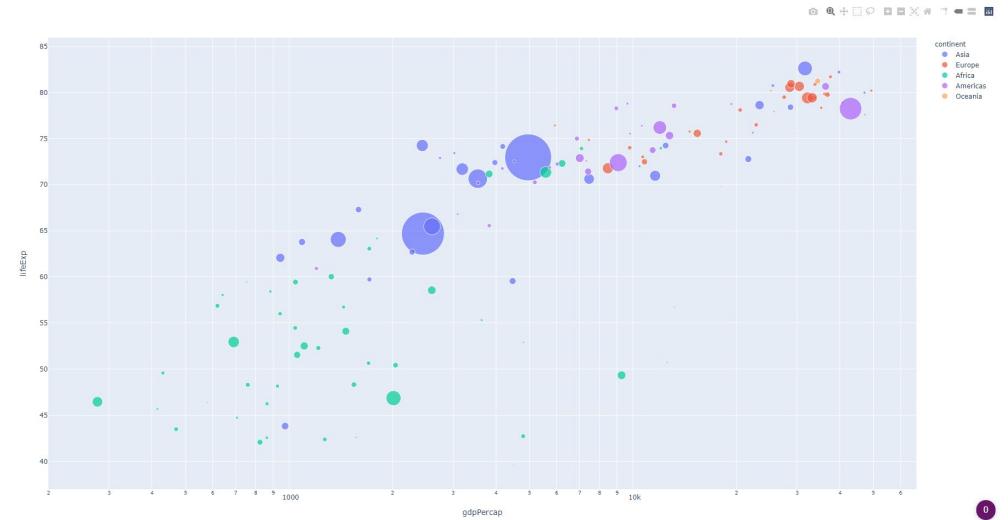
```
df = px.data.gapminder()
```

```
fig = px.scatter(df.query("year==2007"),
x="gdpPercap", y="lifeExp", size="pop",
color="continent", hover_name="country",
log_x=True, size_max=60)
```

```
fig.show()
```

```
fig.write_html("gapminder.html")
```

df.query selects data for a provided condition.
Equivalent to:
df = df[df['year']==2007]

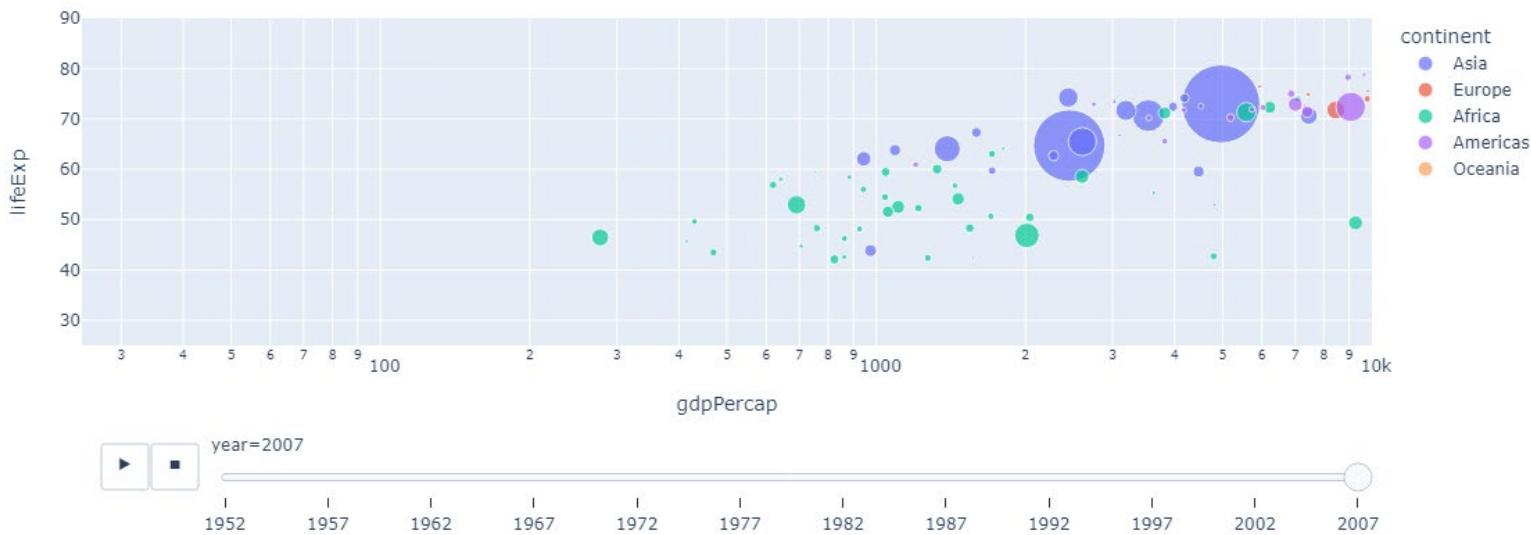


Save the plot as an html file for embedding on webpages, ppt's, etc.



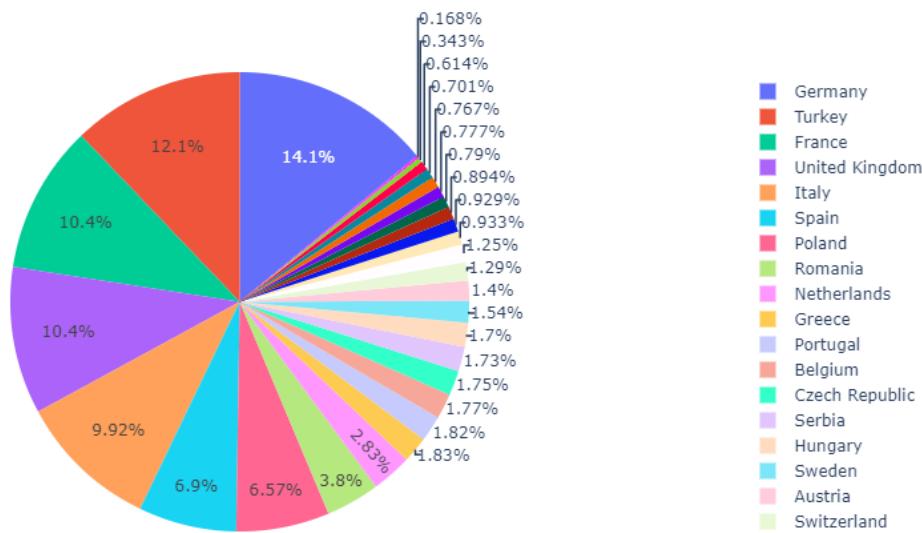
Plotly - Animation Frame

- <https://plotly.com/python/animations/>
- ```
fig = px.scatter(df, x='gdpPercap', y='lifeExp', color='continent', size='pop', size_max=40,
 hover_name='country', log_x=True, animation_frame='year', animation_group='country',
 range_x=[25, 10000], range_y=[25,90])
```
- `fig.show()`



# Plotly - Pie chart

- df = px.data.gapminder().query("year == 2007").query("continent == 'Europe'")
- fig = px.pie(df, values='pop', names='country')
- fig.show()



More at: <https://plotly.com/python/pie-charts/>



# Plotly Graphs

- px.line()
- px.area()
- px.bar()
- px.funnel()
- px.imshow()
- px.pie()
- px.scatter\_3d()
- px.choropleth()
- ...



# Data Analysis

- Statistical Analysis (statmodels):
  - Linear regression
  - Logistic regression
  - Poisson regression
- Machine Learning (sklearn):
  - Unsupervised learning
    - Clustering (kMean)
    - Dimensionality reduction (PCA)
    - Anomaly detection (IsolationForest)
  - Supervised learning
    - Classification



# Example: Tips Data

- Linear Regression

```
import pandas as pd
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm

tips = sns.load_dataset('tips')
print(tips.head())
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4



# Example: Tips Data (ols)

- Linear Regression

```
model = smf.ols('tip ~ total_bill', data=tips)
results = model.fit()
print(results.summary())
```

Is this good?

What do these coefficients mean?

OLS Regression Results							
Dep. Variable:	tip	R-squared:	0.457				
Model:	OLS	Adj. R-squared:	0.454				
Method:	Least Squares	F-statistic:	203.4				
Date:	Tue, 03 Nov 2020	Prob (F-statistic):	6.69e-34				
Time:	00:29:30	Log-Likelihood:	-350.54				
No. Observations:	244	AIC:	705.1				
Df Residuals:	242	BIC:	712.1				
Df Model:	1						
Covariance Type:	nonrobust						
	coef	std err	t	P> t	[0.025	0.975]	
Intercept	0.9203	0.160	5.761	0.000	0.606	1.235	▶▶▶
total_bill	0.1050	0.007	14.260	0.000	0.091	0.120	

# Unsupervised Learning

- Clustering (kMean)
- Dimensionality reduction (PCA)
- Anomaly detection (IsolationForest)

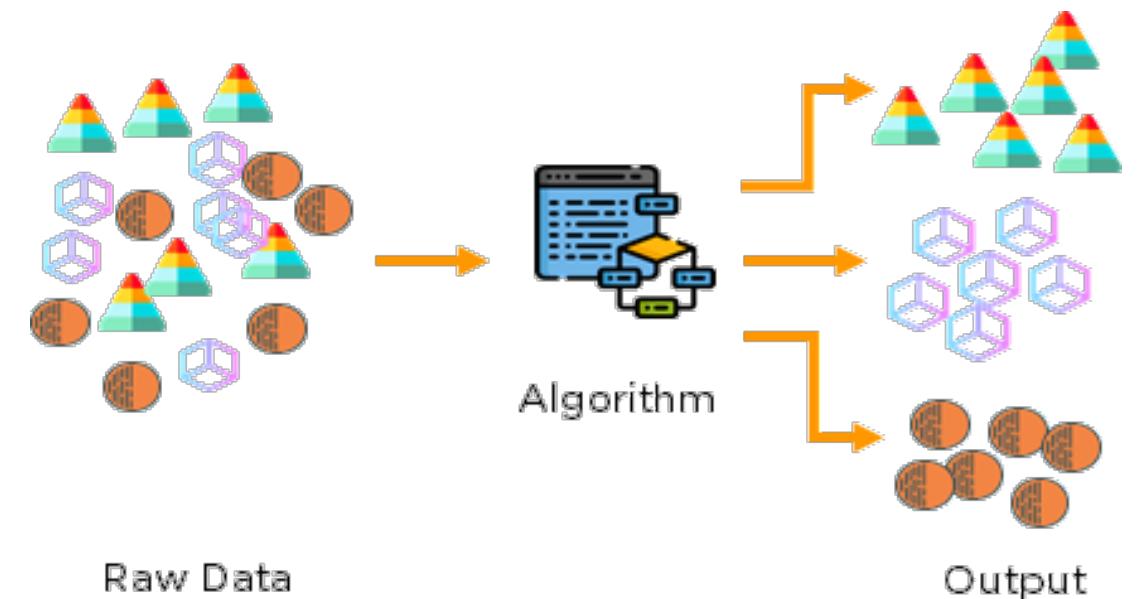


Image sources:

<https://medium.com/analytics-vidhya/beginners-guide-to-unsupervised-learning-76a575c4e942>

# Wine – Clustering (KMeans, k=3)

Considering only 2 features and 3 clusters:

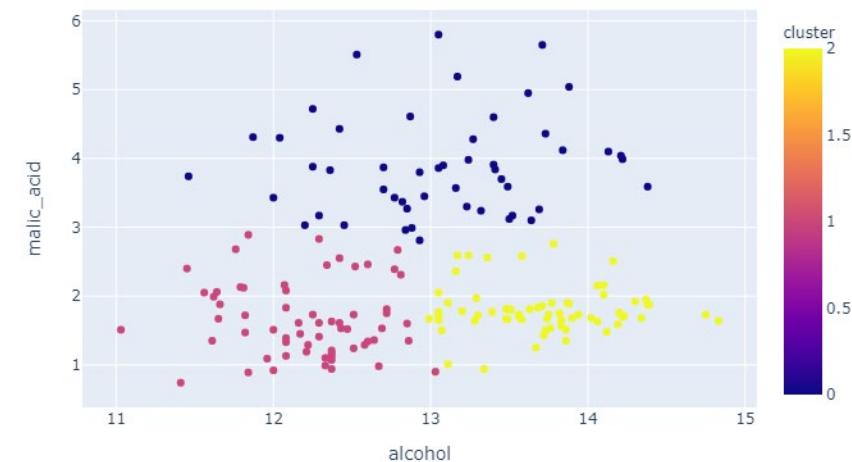
```
from sklearn.cluster import KMeans

#estimate 3 clusters using KMeans based on two variables: 'alcohol','malic_acid'
km1 = KMeans(n_clusters=3, random_state=42).fit(df[['alcohol','malic_acid']])

#add the estimate cluster labels to the dataframe
clusters1 = pd.DataFrame(km1.labels_, columns=['cluster'])
df1 = pd.concat([df[['alcohol','malic_acid']], clusters1], axis=1)

#plot the two variables and predicted clusters
fig1 = px.scatter(df1, 'alcohol', 'malic_acid', color='cluster')
fig1.show()
```

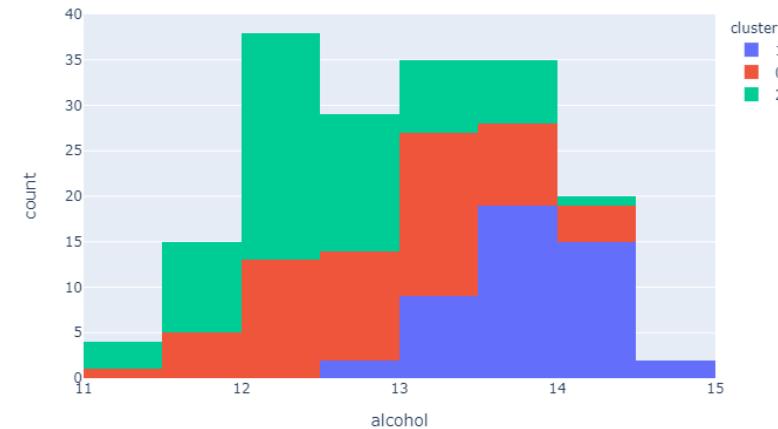
Providing a random seed returns same result every time.



# Clustering - KMeans

Considering ALL features and 3 clusters:

- `kmeans = KMeans(n_clusters=3, random_state=42).fit(df)`
- `clusters = pd.DataFrame(kmeans.labels_, columns=['cluster'])`
- `df2 = pd.concat([df, clusters], axis=1)`
- `fig = px.histogram(df2, 'alcohol', color='cluster')`
- `fig.show()`



*plotting these clusters for all 13 features is hard (13-dimensions!)*



# PCA – dimensionality reduction

- Principal Component Analysis (PCA)

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=2).fit(df)
```

```
pca_trans = pca.transform(df)
```

Project the data on to 2 components

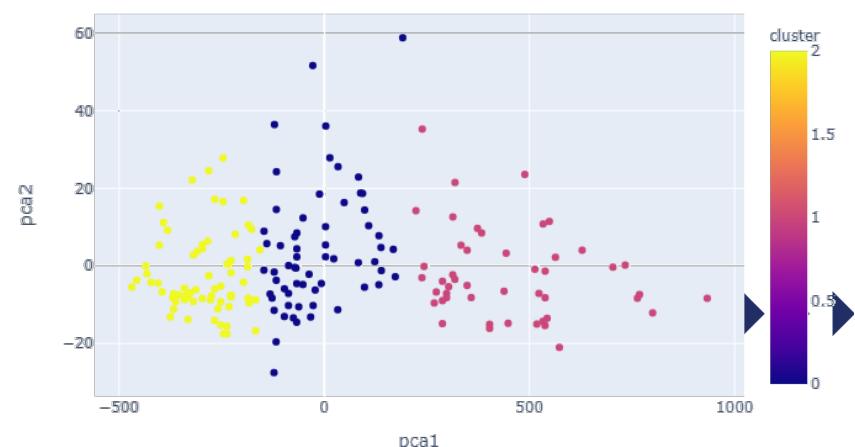
Transform the data to have 2 dimensions

```
pca_trans_df = pd.DataFrame(pca_trans, columns=['pca1', 'pca2'])
```

```
df3 = pd.concat([df2['cluster'], pca_trans_df], axis=1)
```

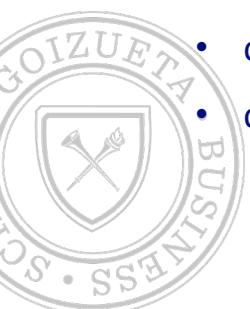
```
fig = px.scatter(df3, 'pca1', 'pca2', color='cluster')
```

```
fig.show()
```



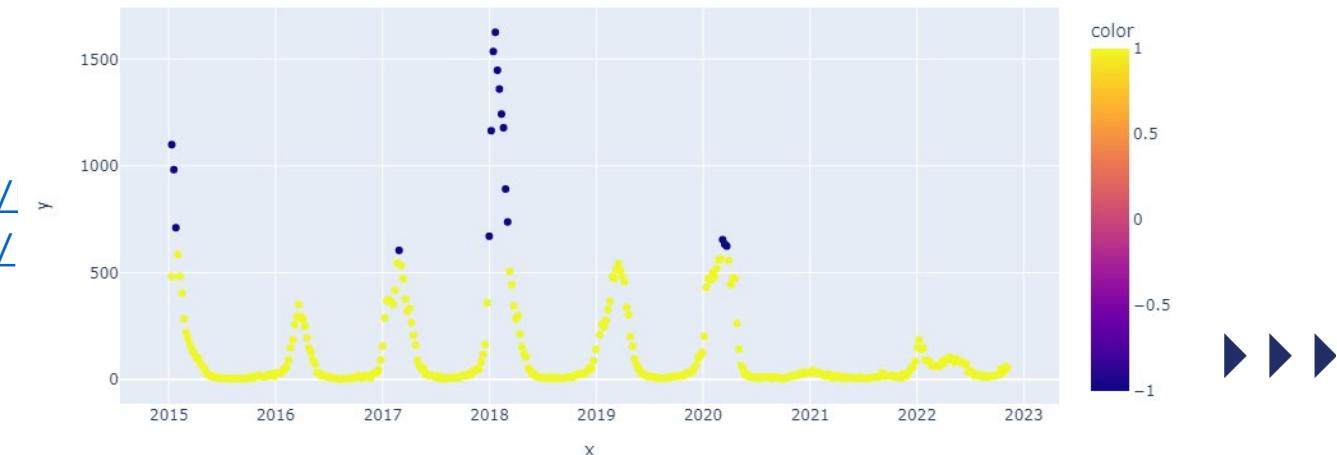
# Anomaly Detection (IsolationForest)

- Find anomaly Flu related mortality
- ```
dfflu = pd.read_csv('datasets/NCHSData44.csv')
```
- ```
from sklearn.ensemble import IsolationForest
```
- ```
#contamination parameter specifies the percent of anomalies in the data
```
- ```
model = IsolationForest(random_state=42, contamination=0.04).fit(dfflu[['Influenza Deaths']])
```
- ```
#add the anomaly score and anomaly prediction in dataframe
```
- ```
dfflu['scores']=model.decision_function(dfflu[['Influenza Deaths']])
```
- ```
dfflu['anomaly']=model.predict(dfflu[['Influenza Deaths']])
```
- ```
dfflu[['Year','Week','Influenza Deaths', 'scores', 'anomaly']].head(20)
```



# Anomaly Detection (IsolationForest)

- #display only anomalous data
- `dfflu[['Year', 'Week', 'Influenza Deaths', 'scores', 'anomaly']].loc[dfflu['anomaly']==-1]`
- `fig = px.scatter(x=dfflu.date, y=dfflu['Influenza Deaths'], color=dfflu['anomaly'])`
- `fig.show()`
- Read:
  - <https://www.ncbi.nlm.nih.gov/pmc/articles/>
  - <https://surveillance.shinyapps.io/fluvview/>
  - <https://www.cdc.gov/flu/weekly/index.htm>



# Supervised Learning

- Linear prediction
- Classification
  - kNN
  - ANN
  - SVM

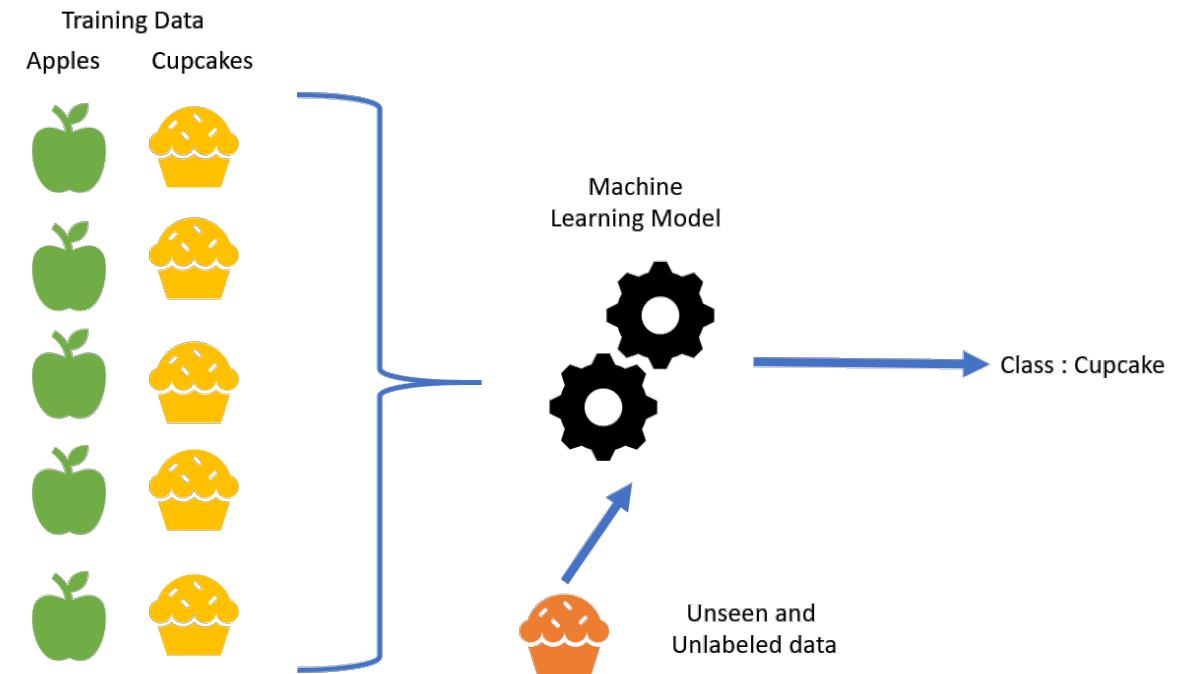


Image source:  
<https://medium.com/@zahraelhamraoui1997/types-of-machine-learning-112467f8c2e>



# kNN – Wine Data

- Split the data into training (80%) and testing (20%) set:

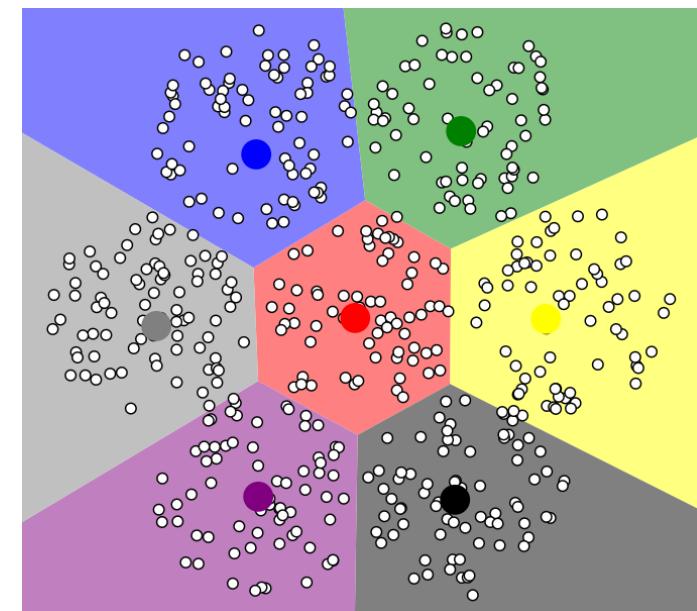
```
• from sklearn.model_selection import train_test_split
• wine = datasets.load_wine()
• data_train, data_test, label_train, label_test = train_test_split(wine['data'], wine['target'], test_size=0.2)
```

- Train the model:

```
• from sklearn.neighbors import KNeighborsClassifier
• knn = KNeighborsClassifier(n_neighbors=6)
• knn.fit(data_train, label_train)
• label_predict = knn.predict(data_test)
```

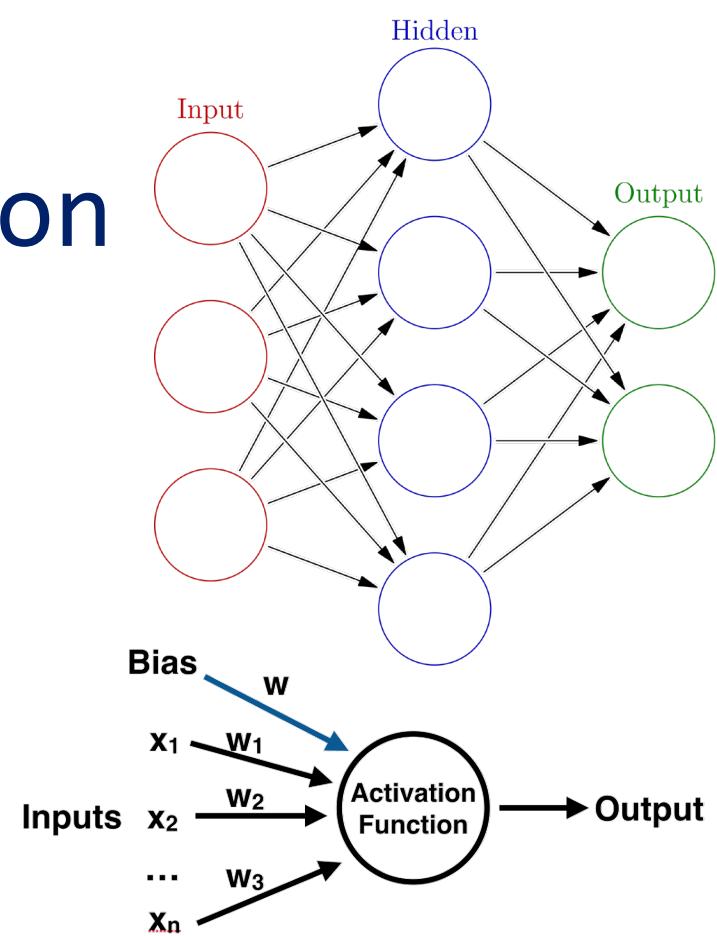
- Estimate the accuracy:

```
• from sklearn import metrics
metrics.accuracy_score(label_test, label_predict)
```



# ANN – Breast Cancer Prediction

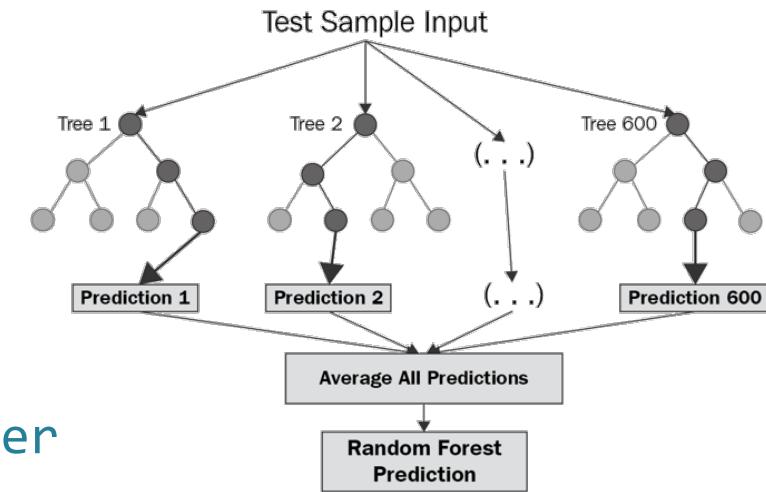
- Data:
  - `bc = datasets.load_breast_cancer()`
  - `data_train, data_test, label_train, label_test = train_test_split(bc.data,bc.target,test_size=0.2)`
- Training (Multi-layer Perceptron classifier):
  - `from sklearn.neural_network import MLPClassifier`
  - `mlp = MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=500)`
  - `mlp.fit(data_train,label_train)`
- Prediction:
  - `label_predict = mlp.predict(data_test)`
  - `print(metrics.accuracy_score(label_test, label_predict))`
  - `print(metrics.confusion_matrix(label_test, label_predict))`



# RF – Breast Cancer Prediction

- Random Forest Classifier

- ```
from sklearn.ensemble import RandomForestClassifier
```
- ```
model = RandomForestClassifier(max_depth=2).fit(data_train,label_train)
```
- ```
Y_pred = model.predict(data_test)
```
- ```
print(metrics.accuracy_score(label_test, label_predict))
```
- ```
print(metrics.confusion_matrix(label_test, label_predict))
```
- **0.921**
- **$\begin{bmatrix} 41 & 6 \\ 3 & 64 \end{bmatrix}$**

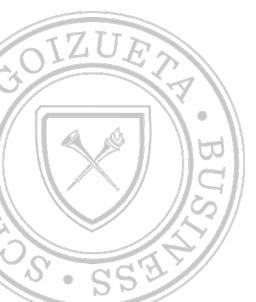


Machine bias

- ML is exciting but can introduce machine bias. Be cautious!
- Machine bias:
 - There's software used across the country to predict future criminals. And it's biased against blacks.
- Why?
 - Prediction is only as good as historical data
 - Supervised learning may introduce human biases - higher weight threshold to certain race



Bernard Parker, left, was rated high risk; Dylan Fugett was rated low risk. Fugett was rated low risk after being arrested with cocaine and marijuana. He was arrested three times on drug charges after that.



SQL in Python

- MySQL + NoSQL + Python + ML = “Grandmaster of the Data Science” ☺
- Why?
 - Bigger: Import/export/manipulate/analyze all kinds of data quickly
 - Faster: Use SQL engine for faster data process and storage (you can even connect to Spark, Hive, etc)
 - Cheaper: Massive volume of open-source libraries for data analysis and ODBC drivers are usually included for free!





Pig I

Data Processing in Hadoop

Apache Pig

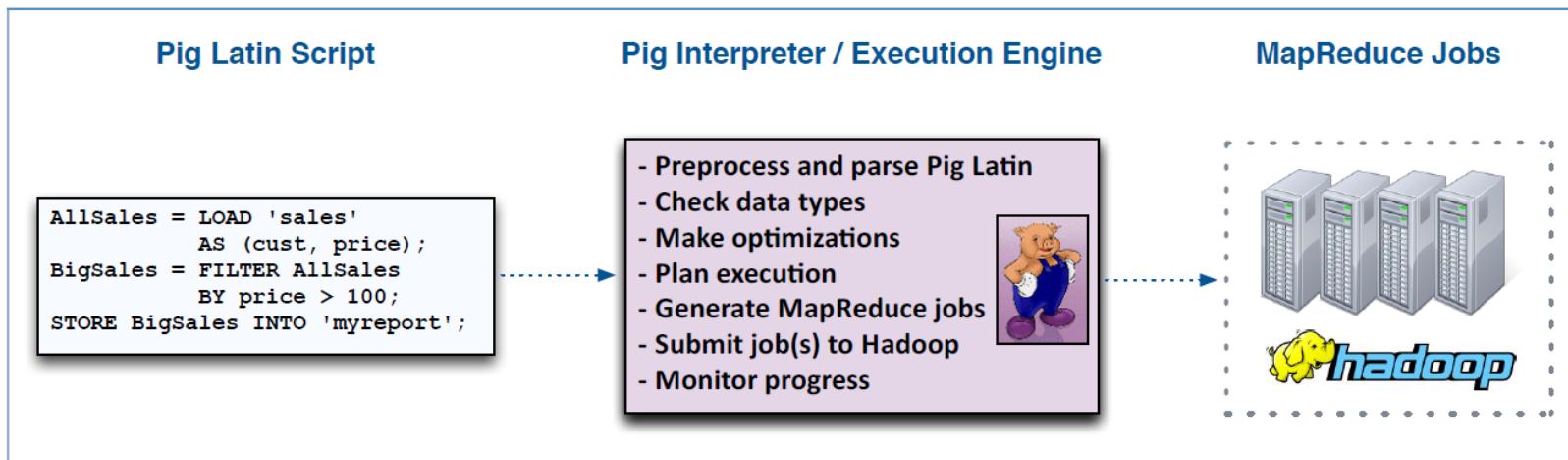


- Apache Pig is a platform for data analysis and processing on Hadoop
 - It offers an alternative to writing (low-level) MapReduce code directly
- Pig was designed to bridge the gap between declarative-style interfaces, such as SQL, and low-level procedural-style programming style required by MapReduce
- Originally developed as a research project at Yahoo
 - Goals: flexibility, productivity, and maintainability
 - Officially released in 2008
 - Now an open-source Apache project



The Anatomy of Pig

- Main components of Pig:
 - The data flow language (Pig Latin)
 - The interactive shell where you can type Pig Latin statements (Grunt)
 - The Pig interpreter and execution engine



Pig Features

- Pig is an alternative to writing low-level MapReduce code
 - A 10-15 line Pig script might do the work of a 200 line MapReduce job
- Many features enable sophisticated analysis and processing
 - HDFS manipulation
 - UNIX shell commands
 - Relational operations
 - Positional references for fields
 - Common mathematical functions (e.g., abs, min, max, sum, etc.)
 - Support for custom functions and data formats
 - Complex data structures like maps and nested data constructs



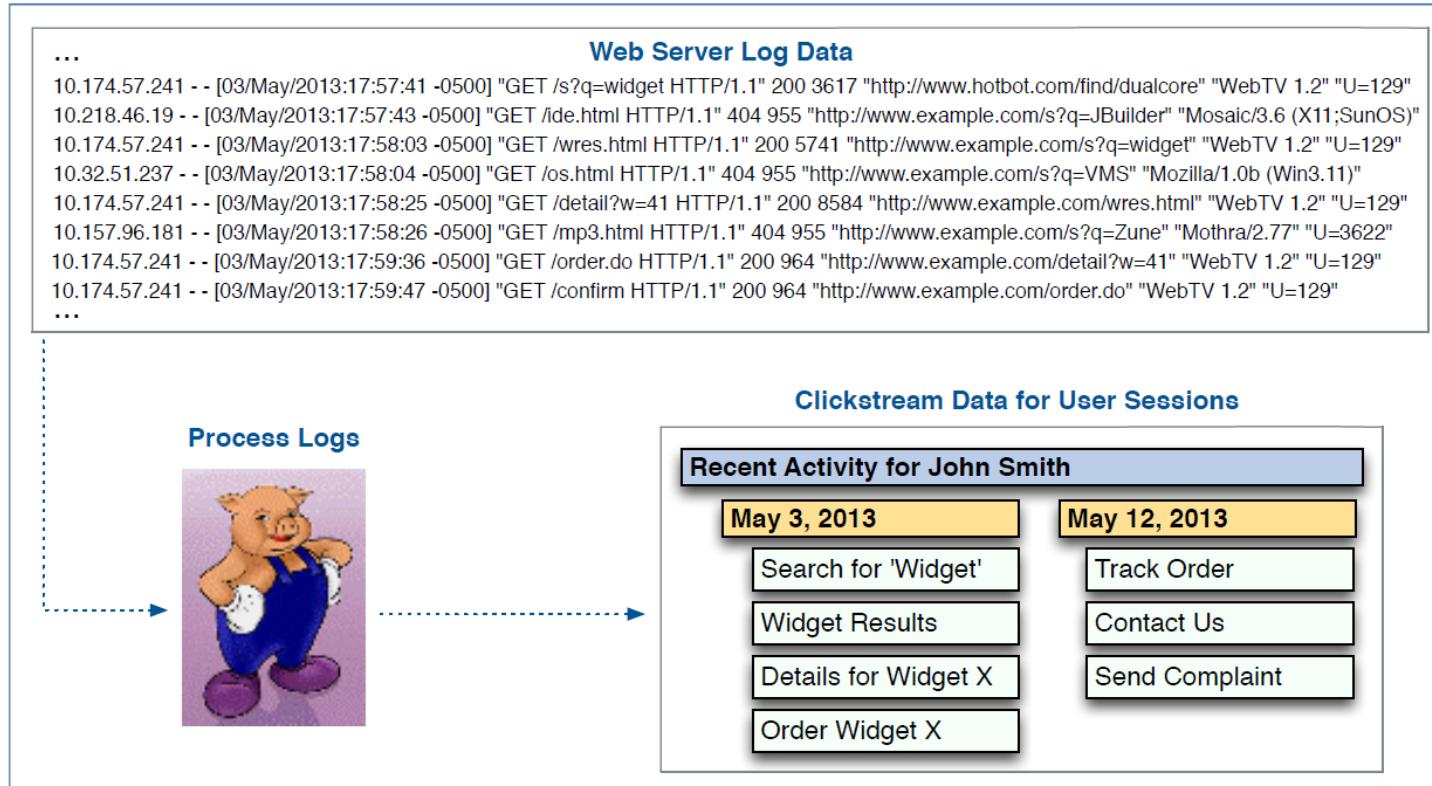
How Are Organizations Using Pig?

- Many organizations use Pig for data analysis
 - Finding relevant records in a massive data set
 - Querying multiple data sets
 - Calculating values from input data
 - Currently in action:
 - 70% of production cluster jobs at Yahoo use Pig
 - Also used by Twitter, LinkedIn, Ebay, AOL, ...
- Pig is also frequently used for data processing
 - Reorganizing an existing data set
 - Joining data from multiple sources to produce a new data set



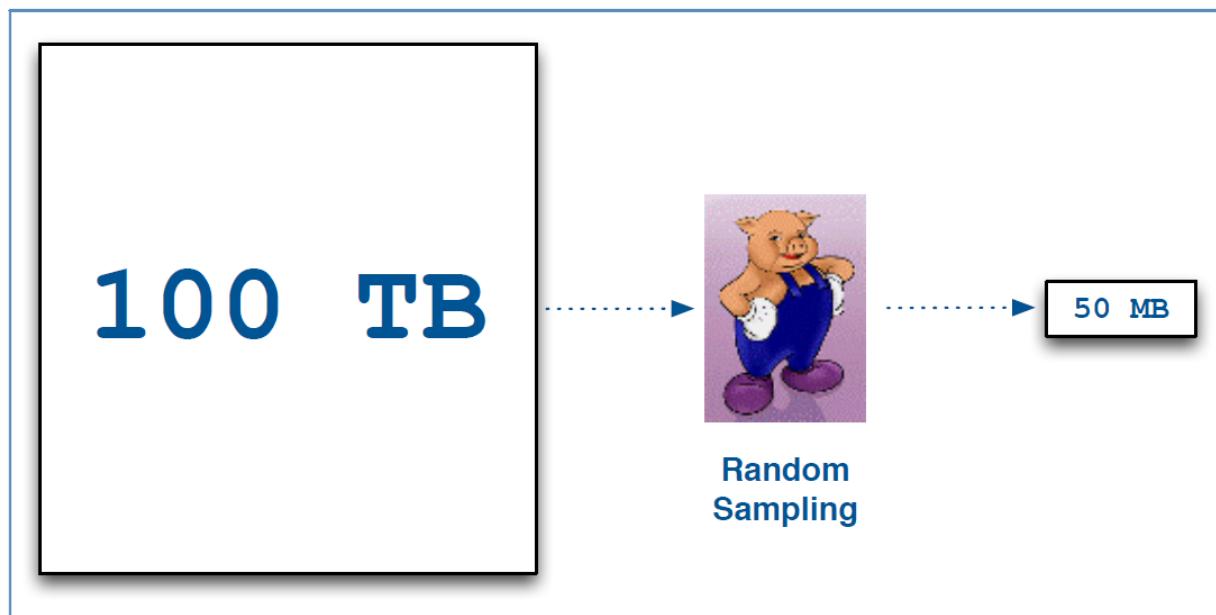
Use Case: Web Log

- Pig can help you extract valuable information from Web server log files



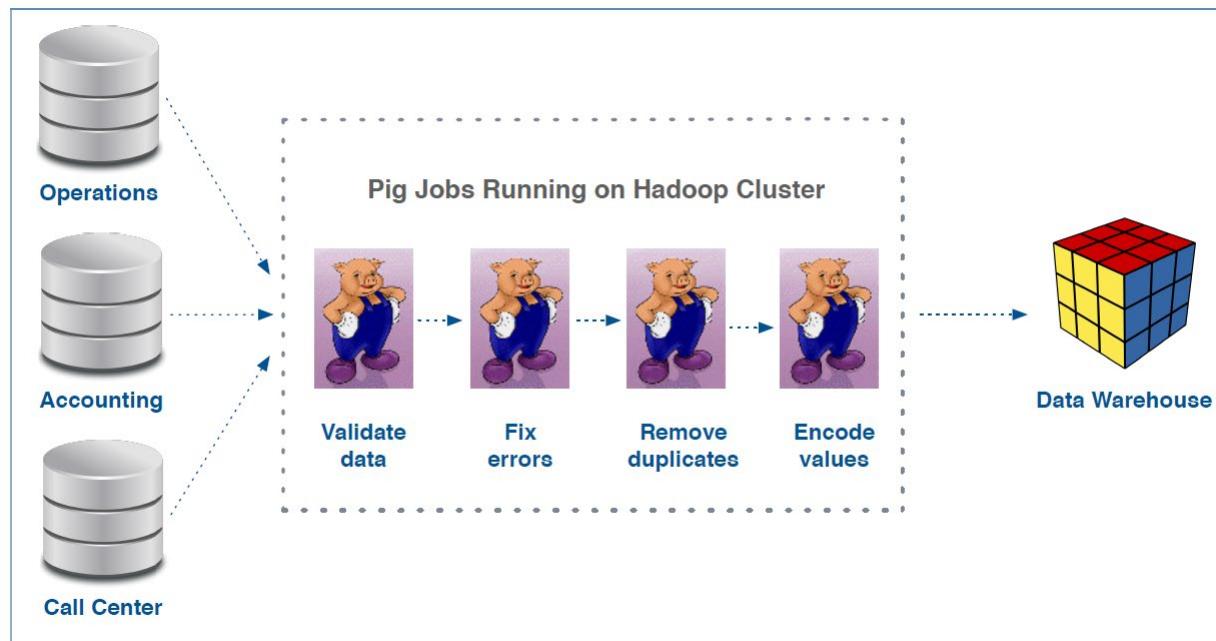
Use Case: Data Sampling

- Sampling can help you explore a representative portion of a large data set
 - Allows you to examine this portion with tools that do not scale well
 - Supports faster iterations during development of analysis jobs



Use Case: ETL Processing

- Pig is also widely used for Extract, Transform, and Load (ETL) processing
 - With the benefits of parallelism that batch jobs ETL often lack



Using Pig Interactively

- You can use Pig interactively, via the Grunt shell
 - Pig interprets each Pig Latin statement as you type it
 - Execution is delayed until output is required
 - Very useful for ad hoc data inspection
- Example of how to start, use, and exit Grunt

```
$ pig
grunt> allsales = LOAD 'sales' AS (name, price);
grunt> bigsales = FILTER allsales BY price > 100;
grunt> STORE bigsales INTO 'myreport';
grunt> quit;
```

- You can also execute a Pig Latin statement from the UNIX shell via the -e option



Interacting with HDFS via Pig

- You can manipulate HDFS with Pig, via the fs command

```
grunt> fs -mkdir sales/;  
grunt> fs -put europe.txt sales/;  
grunt> allsales = LOAD 'sales' AS (name, price);  
grunt> bigsales = FILTER allsales BY price > 100;  
grunt> STORE bigsales INTO 'myreport';  
grunt> fs -getmerge myreport/ bigsales.txt;
```



Running Pig Scripts

- A Pig script is simply Pig Latin code stored in a text file
 - By convention, these files have the .pig extension
- You can run a Pig script from within the Grunt shell via the run command
 - This is useful for automation and batch execution
- It is common to run a Pig script directly from the UNIX shell

```
$ pig salesreport.pig
```



MapReduce and Local Modes for Pig

- As described earlier, Pig turns Pig Latin into MapReduce jobs
 - Pig submits those jobs for execution on the Hadoop cluster
- It is also possible to run Pig in ‘local mode’ using the -x flag
 - This runs MapReduce jobs on the local machine instead of the cluster
 - Local mode uses the local filesystem instead of HDFS
 - Can be helpful for testing before deploying a job to production

```
$ pig -x local          -- interactive  
$ pig -x local salesreport.pig -- batch
```

- One useful option is -c (check), which validates the syntax of your code



Pig Latin Overview

- Pig Latin is a data flow language
 - The flow of data is expressed as a sequence of statements
- The following is a script to load, filter, and store data

```
allsales = LOAD 'sales' AS (name, price);
bigsales = FILTER allsales BY price > 999; -- in US cents
/*
 * Save the filtered results into a new
 * directory, below my home directory.
 */
STORE bigsales INTO 'myreport';
```

- Pig Latin keywords are reserved
 - You cannot use them to name things
- Identifiers are the names assigned to fields and other data structures
- Two types of comments (i.e., -- for single line and /* */ for multi-line comments)



Pig Latin Grammar: Identifiers

- Identifiers must conform to Pig's naming rules
- An identifier must always begin with a letter
 - This may only be followed by letters, numbers, or underscores
 - Identifiers are case-sensitive

| | | | | |
|---------|---|---------|---------|--------|
| Valid | x | q1 | q1_2013 | MyData |
| Invalid | 4 | price\$ | profit% | _sale |



Case-Sensitivity in Pig Latin

- Whether case is significant in Pig Latin depends on context
- Keywords are not case-sensitive
 - Neither are operators (such as AND, OR, or IS NULL)
- Identifiers and paths (shown here in red text) are case-sensitive
 - So are function names (such as SUM or COUNT) and constants

```
allsales = LOAD 'sales' AS (name, price);
bigsales = FILTER allsales BY price > 999;
STORE bigsales INTO 'myreport';
```



Common Operators in Pig Latin

- Many commonly-used operators in Pig Latin are familiar to SQL users
 - Notable difference: Pig Latin uses == and != for comparison

| Arithmetic | Comparison | Null | Boolean |
|------------|------------|-------------|---------|
| + | == | IS NULL | AND |
| - | != | IS NOT NULL | OR |
| * | < | | NOT |
| / | > | | |
| % | <= | | |
| | >= | | |



Basic Data Loading in Pig

- Pig's default loading function is called PigStorage
 - The name of the function is implicit when calling LOAD
 - You don't have to invoke the PigStorage function as part of the LOAD command (it's implied)
 - PigStorage assumes text format with tab-separated columns
- Consider the following file in HDFS called sales
 - The two fields are separated by tab characters

Alice 2999
Bob 3625
Carlos 2764

- This example loads data from the above file
- ```
allsales = LOAD 'sales' AS (name, price);
```



# Data Sources: File and Directories

- The previous example loads data from a file named sales

```
allsales = LOAD 'sales' AS (name, price);
```
- Since this is not an absolute path, it is relative to your home directory (or Grunt shell)
  - Your home directory in HDFS is typically /user/youruserid/
  - Can also specify an absolute path (e.g., /dept/sales/2017/q4)
- The path can also refer to a directory
  - In this case, Pig will recursively load all files in that directory
  - Unix style file patterns (“globs”) are also supported:

```
allsales = LOAD 'sales_200[5-9]' AS (name, price);
```



# Using Alternate Column Delimiters

- You can specify an alternate single-character delimiter as an argument to PigStorage
- This example shows how to load comma-delimited data
  - Note that this is a single statement

```
allsales = LOAD 'sales.csv' USING PigStorage(',') AS (name, price);
```

- Or to load pipe-delimited data without specifying column names

```
allsales = LOAD 'sales.txt' USING PigStorage('|');
```



# Data Output in Pig

- The command used to handle output depends on its destination
  - DUMP: sends output to the screen
  - STORE: sends output to disk (HDFS)
- Example of DUMP output, using data from the file shown earlier

- The parentheses and commas indicate tuples with multiple fields

```
(Alice,2999,us)
(Bob,3625,ca)
(Carlos,2764,mx)
(Dieter,1749,de)
(Étienne,2368,fr)
(Franco,5637,it)
```



# Storing Data with Pig

- The **STORE** command is used to store data to HDFS
  - Similar to LOAD, but writes data instead of reading it
  - The output path is the name of a directory
  - The directory must not yet exist (as in any MapReduce job)
- As with LOAD, the use of PigStorage is implicit
  - The field delimiter also has a default value (tab)

```
STORE bigsales INTO 'myreport';
```

- You may also specify an alternate delimiter

```
STORE bigsales INTO 'myreport' USING PigStorage(',');
```



# Viewing the Schema

- The `DESCRIBE` command shows the structure of the data, including names and types
  - Just echos the definition of fields declared when the bag was loaded
    - Particularly useful representing relations containing nested data sets
- The following Grunt session shows an example

```
grunt> allsales = LOAD 'sales' AS (name:chararray, price:int);
grunt> DESCRIBE allsales;

allsales: {name: chararray,price: int}
```



# Eliminating Duplicates

- DISTINCT eliminates duplicate records in a bag
- All fields must be equal to be considered a duplicate

```
unique_records = DISTINCT all_alices;
```

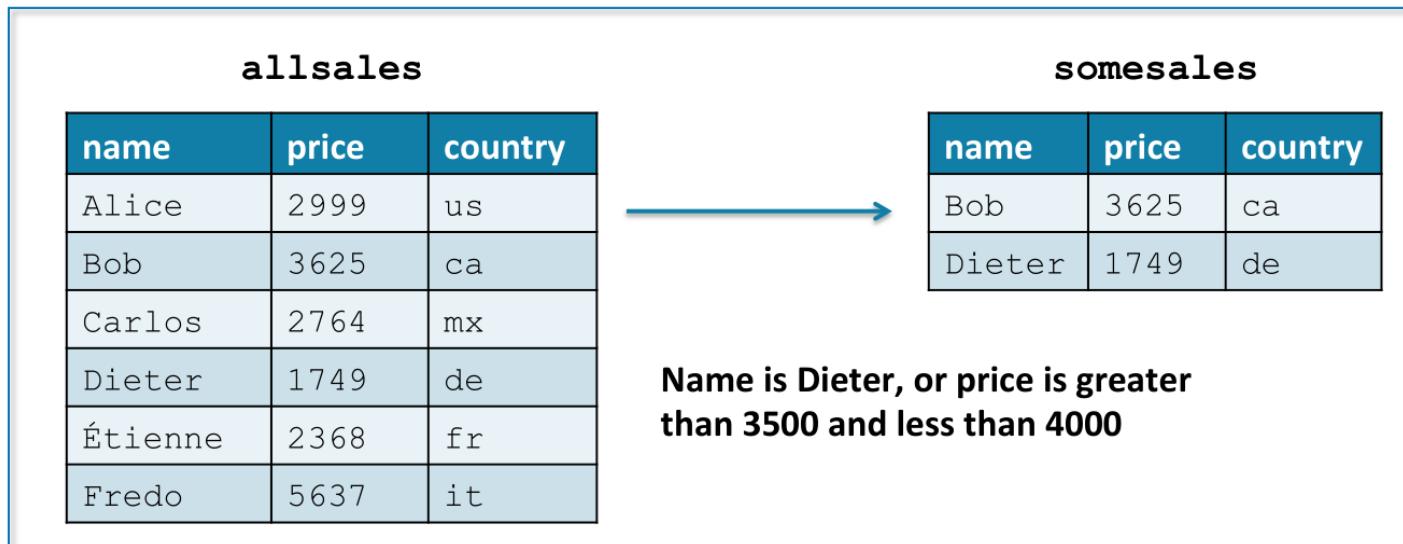
all_alices			unique_records		
firstname	lastname	country	firstname	lastname	country
Alice	Smith	us	Alice	Smith	us
Alice	Jones	us	Alice	Jones	us
<b>Alice</b>	<b>Brown</b>	<b>us</b>	<b>Alice</b>	<b>Brown</b>	<b>us</b>
<b>Alice</b>	<b>Brown</b>	<b>us</b>	Alice	Brown	ca
Alice	Brown	ca			



# Filtering Tuples

- The **FILTER** keyword extracts tuples matching the specified criteria
- You can combine criteria with AND and OR

```
somesales = FILTER allsales BY name == 'Dieter' OR (price > 3500 AND price < 4000);
```



# How Pig Handles Invalid Data

- When encountering invalid data, Pig substitutes NULL for the value
  - For example, an int field containing the value Q4
- The **IS NULL** and **IS NOT NULL** operators test for null values
  - Note that NULL is not the same as the empty string “ ”
- You can use these operators to filter out bad records

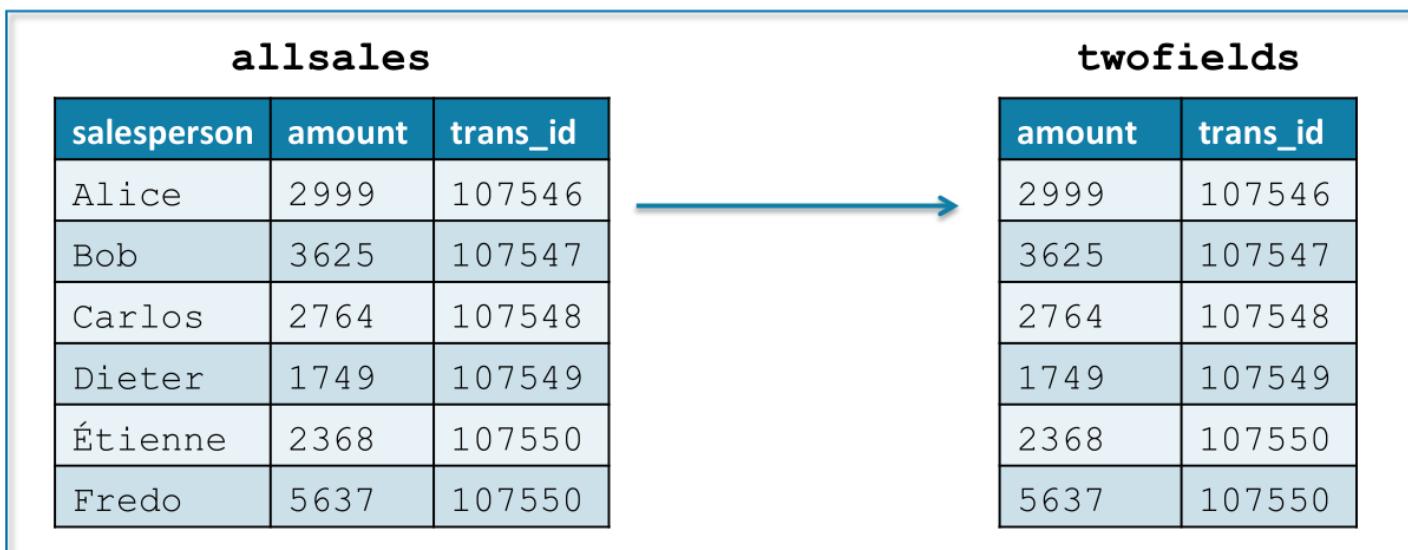
```
hasprices = FILTER Records BY price IS NOT NULL;
```



# Field Selection in Pig Latin

- Filtering extracts rows, but sometimes we need to extract columns

- This is done in Pig Latin using the FOREACH and GENERATE keywords
- ```
twofields = FOREACH allsales GENERATE amount, trans_id;
```



Generating New Fields in Pig Latin

- The FOREACH and GENERATE keywords can also be used to create fields

- For example, you could create a new field based on price

```
t = FOREACH allsales GENERATE price * 0.07;
```

- It is possible to name such fields

```
t = FOREACH allsales GENERATE price * 0.07 AS tax;
```

- And you can also specify the data type

```
t = FOREACH allsales GENERATE price * 0.07 AS tax:float;
```



Regular Expressions

- The FOREACH and REGEX_EXTRACT keywords can be used to parse fields
 - REGEXP_EXTRACT(STRING data, STRING regex pattern, INT index)
 - For example, you could extract a field (IP address) from “IP:port”

```
ipAddrs = FOREACH ipdata GENERATE REGEX_EXTRACT(data, '(.*):(.*)', 1);
```
 - you could extract a field (IP port) from “IP:port”

```
pPorts = FOREACH ipdata GENERATE REGEX_EXTRACT(data, '(.*):(.*)', 2);
```
- Sample data:
 - 192.168.1.1:1433
 - 192.168.1.2:9020



Regular Expressions (example)

- Data (<https://www.kaggle.com/datasets/kimjmin/apache-web-log>):
 - 14.49.42.25 - - [12/May/2022:01:24:44 +0000] "GET /articles/ppp-over-ssh/ HTTP/1.1" 200 18586 "-" "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2b1) Gecko/20091014 Firefox/3.6b1 GTB5"
 - 14.49.42.25 - - [12/May/2022:01:24:15 +0000] "GET /articles/openldap-with-saslauthd/ HTTP/1.1" 200 12700 "-" "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2b1) Gecko/20091014 Firefox/3.6b1 GTB5"
- Pig:
 - loglines = LOAD 's3://my671/weblog-sample.log' AS line;
 - logs = FOREACH loglines GENERATE FLATTEN(REGEX_EXTRACT_ALL(line,'^(.* - - (.*) "(.*") (.*) (.*) "-" (.*)\$')) as (ip,timestamp,access,status,bytes,device);
 - logs10 = limit logs 10;
 - dump logs10;
- Output (description: <https://httpd.apache.org/docs/2.4/logs.html>):
 - (14.49.42.25,[12/May/2022:01:24:44 +0000],GET /articles/ppp-over-ssh/ HTTP/1.1,200,18586,"Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2b1) Gecko/20091014 Firefox/3.6b1 GTB5")
 - (14.49.42.25,[12/May/2022:01:24:15 +0000],GET /articles/openldap-with-saslauthd/ HTTP/1.1,200,12700,"Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2b1) Gecko/20091014 Firefox/3.6b1 GTB5")



Controlling Sort Order

- Use ORDER...BY to sort the records in a bag in ascending order
 - Add DESC to sort in descending order instead
 - Take care to specify a schema - data type affects how data is sorted!

```
sortedsales = ORDER allsales BY country DESC;
```

| allsales | | | sortedsales | | |
|----------|-------|---------|-------------|-------|---------|
| name | price | country | name | price | country |
| Alice | 29.99 | us | Alice | 29.99 | us |
| Bob | 36.25 | ca | Carlos | 27.64 | mx |
| Carlos | 27.64 | mx | Fredo | 56.37 | it |
| Dieter | 17.49 | de | Étienne | 23.68 | fr |
| Étienne | 23.68 | fr | Dieter | 17.49 | de |
| Fredo | 56.37 | it | Bob | 36.25 | ca |



Limiting Results

- As in SQL, you can use `LIMIT` to reduce the number of output records

```
somesales = LIMIT allsales 10;
```

- Useful for sampling a data set
- Beware! Record ordering is random unless specified with `ORDER BY`

- And can vary from one command to the next
 - Without `ORDER/BY` running the same Pig command twice in succession will yield the same records, but possibly in different orders
 - Use `ORDER BY` and `LIMIT` together to find top-N results

```
sortedsales = ORDER allsales BY price DESC;  
top_five = LIMIT sortedsales 5;
```



Grouping Records By a Field

- Sometimes you need to group records by a given field
 - For example, so you can calculate commissions for each employee

Alice 729

Bob 3999

Alice 27999

Carol 32999

Carol 4999

- Use **GROUP BY** to do this in Pig Latin

- The new relation has one record per unique value in the specified field

```
grunt> byname = GROUP sales BY name;
```



Grouping Records By a Field

- The new relation always contains two fields

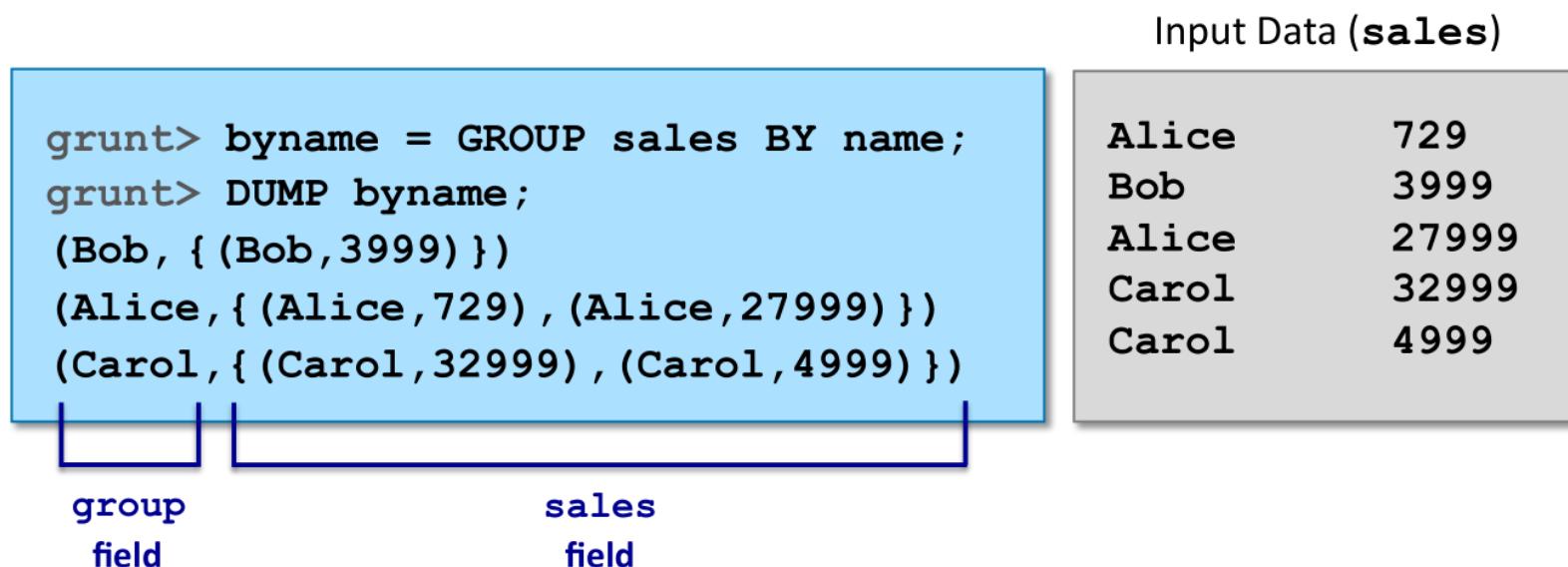
```
grunt> byname = GROUP sales BY name;  
grunt> DESCRIBE byname;  
byname: {group: chararray,sales: {(name:  
chararray,price: int)}}
```

- The first field is literally named group in all cases
 - No matter what the field you grouped by is actually called
 - Contains the value from the field specified in GROUP BY
- The second field is named after the relation specified in GROUP BY
 - It's a bag containing one tuple for each corresponding value



Grouping Records By a Field

- The example below shows the data after grouping



Using GROUP BY to Aggregate Data

- Aggregate functions create one output value from multiple input values
 - For example, to calculate total sales by employee
 - Usually applied to grouped data

```
grunt> byname = GROUP sales BY name;
grunt> DUMP byname;
(Bob,{{Bob,3999}})
(Alice,{{Alice,729},{Alice,27999}})
(Carol,{{Carol,32999},{Carol,4999}})

grunt> totals = FOREACH byname GENERATE
          group, SUM(sales.price);
grunt> dump totals;
(Bob,3999)
(Alice,28728)
(Carol,37998)
```



Grouping Everything Into a Single Record

- We just saw that GROUP BY creates one record for each unique value
- GROUP ALL puts all data into one record

```
grunt> grouped = GROUP sales ALL;  
grunt> DUMP grouped;  
(all,{{(Alice,729),(Bob,3999),(Alice,27999),  
(Carol,32999),(Carol,4999)}})
```



Using GROUP ALL to Aggregate Data

- Use GROUP ALL when you need to aggregate one or more columns
 - For example, to calculate total sales for all employees

```
grunt> grouped = GROUP sales ALL;  
grunt> DUMP grouped;  
(all,{{(Alice,729),(Bob,3999),(Alice,27999),(Carol,32999),  
(Carol,4999)})  
  
grunt> totals = FOREACH grouped GENERATE SUM(sales.price);  
grunt> dump totals;  
(70725)
```



Using SAMPLE to Create a Smaller Data Set

- Your code might process terabytes of data in production
 - However, it is convenient to test with smaller amounts during development
- Use **SAMPLE** to choose a random set of records from a data set
- This example selects about 5% of records from `bigdata`
 - Stores them in a new directory called `mysample`

```
everything = LOAD 'bigdata';
subset = SAMPLE everything 0.05;
STORE subset INTO 'mysample';
```



Intelligent Sampling with ILLUSTRATE

- Sometimes a random sample may lack data needed for testing
 - For example, matching records in two data sets for a JOIN operation
- Pig's **ILLUSTRATE** keyword can do more intelligent sampling
 - Pig will examine the code to determine what data is needed
 - It picks a few records that properly exercise the code
- You should specify a schema when using ILLUSTRATE
 - Pig will generate records when yours don't suffice





Pig II

Data Processing in Hadoop

Multi-Dataset Operations

In Pig



Grouping Multiple Relations

- We previously learned about the GROUP operator
 - Groups values in a relation based on the specified field(s)
- The GROUP operator can also group multiple relations
 - In this case, using the synonymous COGROUP operator is preferred
grouped = COGROUP stores BY store_id, salespeople BY store_id;
- This collects values from both data sets into a new relation
 - As before, the new relation is keyed by a field named group
 - This group field is associated with one bag for each input
(group, {bag of records}, {bag of records})

↑ ↑ ↑
store_id records from stores records from salespeople



Example of COGROUP

Stores

| | |
|---|-----------|
| A | Anchorage |
| B | Boston |
| C | Chicago |
| D | Dallas |
| E | Edmonton |
| F | Fargo |

Salespeople

| | | |
|----|---------|---|
| 1 | Alice | B |
| 2 | Bob | D |
| 3 | Carlos | F |
| 4 | Dieter | A |
| 5 | Étienne | F |
| 6 | Fredo | C |
| 7 | George | D |
| 8 | Hannah | B |
| 9 | Irina | C |
| 10 | Jack | |

```
grunt> grouped = COGROUP stores BY store_id,  
salespeople BY store_id;  
  
grunt> DUMP grouped;  
(A, { (A,Anchorage) }, {(4,Dieter,A)})  
(B, { (B,Boston) }, {(1,Alice,B), (8,Hannah,B)})  
(C, { (C,Chicago) }, {(6,Fredo,C), (9,Irina,C)})  
(D, { (D,Dallas) }, {(2,Bob,D), (7,George,D)})  
(E, { (E,Edmonton) }, {})  
(F, { (F,Fargo) }, {(3,Carlos,F), (5,Étienne,F)})  
(, {}, {(10,Jack,)})
```



Join Overview

- The **COGROUP** operator creates a nested data structure
- Pig Latin's **JOIN** operator creates a flat data structure
 - Similar to joins in a relational database
 - Supports only “equi-joins” (the foreign key is present in all input data sets)
- A **JOIN** is similar to doing a **COGROUP** followed by a **FLATTEN**
 - Though they handle null values differently
 - COGROUP performs the equivalent of a full outer join
 - JOIN performs the equivalent of an inner join (filtering out non-null values)



Key Fields

- Like COGROUP, joins rely on a field shared by each relation

```
joined = JOIN stores BY store_id, salespeople BY store_id;
```

- Joins can also use multiple fields as the key

- e.g. a name might not be unique, nor phone_number, but when combined they might provide a way to uniquely identify a record

```
joined = JOIN customers BY (name, phone_number),  
accounts BY (name, phone_number);
```



Inner Joins

- The default JOIN in Pig Latin is an inner join

```
joined = JOIN stores BY store_id, salespeople BY store_id;
```
- An inner join outputs records only when the key is found in all inputs
 - In the above example, stores that have at least one salesperson
- You can do an inner join on multiple relations in a single statement
 - But you must use the same key to join them



Inner Join Example

stores

| | |
|---|-----------|
| A | Anchorage |
| B | Boston |
| C | Chicago |
| D | Dallas |
| E | Edmonton |
| F | Fargo |

salespeople

| | | |
|----|---------|---|
| 1 | Alice | B |
| 2 | Bob | D |
| 3 | Carlos | F |
| 4 | Dieter | A |
| 5 | Étienne | F |
| 6 | Fredo | C |
| 7 | George | D |
| 8 | Hannah | B |
| 9 | Irina | C |
| 10 | Jack | |

```
grunt> joined = JOIN stores BY store_id,  
salespeople BY store_id;
```

```
grunt> DUMP joined;  
(A,Anchorage,4,Dieter,A)  
(B,Boston,1,Alice,B)  
(B,Boston,8,Hannah,B)  
(C,Chicago,6,Fredo,C)  
(C,Chicago,9,Irina,C)  
(D,Dallas,2,Bob,D)  
(D,Dallas,7,George,D)  
(F,Fargo,3,Carlos,F)  
(F,Fargo,5,Étienne,F)
```



Eliminating Duplicate Fields

- We can use FOREACH...GENERATE to retain just the fields we need
 - However, it is now slightly more complex to reference fields
 - We must fully-qualify any fields with names that are not unique

```
grunt> DESCRIBE joined;
joined: {stores::store_id: chararray,stores::name:
chararray,salespeople::person_id: int,salespeople::name:
chararray,salespeople::store_id: chararray}

grunt> cleaned = FOREACH joined GENERATE stores::store_id,
stores::name, person_id, salespeople::name;

grunt> DUMP cleaned;
(A,Anchorage,4,Dieter)
(B,Boston,1,Alice)
(B,Boston,8,Hannah)
... (additional records omitted for brevity) ...
```



Outer Joins

- Pig Latin allows you to specify the type of join following the field name
 - Inner joins do not specify a join type
`joined = JOIN relation1 BY field [LEFT|RIGHT|FULL] OUTER,
relation2 BY field;`
- An outer join does not require the key to be found in both inputs
 - Supports non-equi-joins
- Outer joins require Pig to know the schema for at least one relation
 - Which relation requires schema depends on the join type
 - Full outer joins require schema for both relations



Left Outer Join Example



| stores | |
|--------|-----------|
| A | Anchorage |
| B | Boston |
| C | Chicago |
| D | Dallas |
| E | Edmonton |
| F | Fargo |

| salespeople | | |
|-------------|---------|---|
| 1 | Alice | B |
| 2 | Bob | D |
| 3 | Carlos | F |
| 4 | Dieter | A |
| 5 | Étienne | F |
| 6 | Fredo | C |
| 7 | George | D |
| 8 | Hannah | B |
| 9 | Irina | C |
| 10 | Jack | |

- Result contains *all* records from the relation specified on the left, but only *matching* records from the one specified on the right

```
grunt> joined = JOIN stores BY store_id  
LEFT OUTER, salespeople BY store_id;
```

```
grunt> DUMP joined;  
(A,Anchorage,4,Dieter,A)  
(B,Boston,1,Alice,B)  
(B,Boston,8,Hannah,B)  
(C,Chicago,6,Fredo,C)  
(C,Chicago,9,Irina,C)  
(D,Dallas,2,Bob,D)  
(D,Dallas,7,George,D)  
(E,Edmonton,,,)  
(F,Fargo,3,Carlos,F)  
(F,Fargo,5,Étienne,F)
```



Right Outer Join Example



| stores | |
|--------|-----------|
| A | Anchorage |
| B | Boston |
| C | Chicago |
| D | Dallas |
| E | Edmonton |
| F | Fargo |

| salespeople | | |
|-------------|---------|---|
| 1 | Alice | B |
| 2 | Bob | D |
| 3 | Carlos | F |
| 4 | Dieter | A |
| 5 | Étienne | F |
| 6 | Fredo | C |
| 7 | George | D |
| 8 | Hannah | B |
| 9 | Irina | C |
| 10 | Jack | |

- Result contains *all* records from the relation specified on the right, but only *matching* records from the one specified on the left

```
grunt> joined = JOIN stores BY store_id  
RIGHT OUTER, salespeople BY store_id;
```

```
grunt> DUMP joined;  
(A,Anchorage,4,Dieter,A)  
(B,Boston,1,Alice,B)  
(B,Boston,8,Hannah,B)  
(C,Chicago,6,Fredo,C)  
(C,Chicago,9,Irina,C)  
(D,Dallas,2,Bob,D)  
(D,Dallas,7,George,D)  
(F,Fargo,3,Carlos,F)  
(F,Fargo,5,Étienne,F)  
(,,10,Jack,)
```



Full Outer Join Example



| stores | |
|--------|-----------|
| A | Anchorage |
| B | Boston |
| C | Chicago |
| D | Dallas |
| E | Edmonton |
| F | Fargo |

| salespeople | | |
|-------------|---------|---|
| 1 | Alice | B |
| 2 | Bob | D |
| 3 | Carlos | F |
| 4 | Dieter | A |
| 5 | Étienne | F |
| 6 | Fredo | C |
| 7 | George | D |
| 8 | Hannah | B |
| 9 | Irina | C |
| 10 | Jack | |

- Result contains *all* records where there is a match in *either* relation

```
grunt> joined = JOIN stores BY store_id  
      FULL OUTER, salespeople BY store_id;
```

```
grunt> DUMP joined;  
(A,Anchorage,4,Dieter,A)  
(B,Boston,1,Alice,B)  
(B,Boston,8,Hannah,B)  
(C,Chicago,6,Fredo,C)  
(C,Chicago,9,Irina,C)  
(D,Dallas,2,Bob,D)  
(D,Dallas,7,George,D)  
(E,Edmonton,,,)  
(F,Fargo,3,Carlos,F)  
(F,Fargo,5,Étienne,F)  
(,,10,Jack,)
```



Crossing Data Sets

- **JOIN** finds records in one relation that match records in another
- Pig's **CROSS** operator creates the cross product of both relations
 - Combines all records in both tables regardless of matching
 - Joins every tuple in stores with every tuple in salespeople
 - In other words, all possible combinations of records
crossed = CROSS stores, salespeople;
- Careful: This can generate huge amounts of data!



Cross Product Example

| stores | | |
|--------|-----------|--|
| A | Anchorage | |
| B | Boston | |
| D | Dallas | |

| salespeople | | |
|-------------|--------|---|
| 1 | Alice | B |
| 2 | Bob | D |
| 8 | Hannah | B |
| 10 | Jack | |

- Generates every possible combination of records in the stores and salespeople relations

```
grunt> crossed = CROSS stores, salespeople;  
  
grunt> DUMP crossed;  
(A,Anchorage,1,Alice,B)  
(A,Anchorage,2,Bob,D)  
(A,Anchorage,8,Hannah,B)  
(A,Anchorage,10,Jack,)  
(B,Boston,1,Alice,B)  
(B,Boston,2,Bob,D)  
(B,Boston,8,Hannah,B)  
(B,Boston,10,Jack,)  
(D,Dallas,1,Alice,B)  
(D,Dallas,2,Bob,D)  
(D,Dallas,8,Hannah,B)  
(D,Dallas,10,Jack,)
```



Concatenating Data Sets

- We have explored several techniques for combining data sets
 - They have had one thing in common: they combine horizontally
 - Combining rows to create more columns
- The **UNION** operator combines records vertically
 - It adds data from input relations into a new single relation
 - Pig does not require these inputs to have the same schema
 - It does not eliminate duplicate records nor preserve order
- This is helpful for incorporating new data into your processing

```
both = UNION june_items, july_items;
```

Similar to UNION ALL in SQL



UNION Example

| june | |
|---------|-------|
| Adapter | 549 |
| Battery | 349 |
| Cable | 799 |
| DVD | 1999 |
| HDTV | 79999 |

| july | |
|------|-------|
| Fax | 17999 |
| GPS | 24999 |
| HDTV | 65999 |
| Ink | 3999 |

- Concatenates all records from june and july

```
grunt> both = UNION june_items, july_items;  
  
grunt> DUMP both;  
(Fax,17999)  
(GPS,24999)  
(HDTV,65999)  
(Ink,3999)  
(Adapter,549)  
(Battery,349)  
(Cable,799)  
(DVD, 1999)  
(HDTV,79999)
```



Splitting Data Sets

- You have learned several ways to combine data sets into a single relation
- Sometimes you need to split a data set into multiple relations
 - Server logs by date range
 - Customer lists by region
 - Product lists by vendor
- Pig Latin supports this with the **SPLIT** operator

```
SPLIT relation INTO relationA IF expression1,  
        relationB IF expression2,  
        relationC IF expression3, ...  
        relationZ OTHERWISE;
```

- Expressions need not be mutually exclusive



Split Example

- E.g., split customers into groups for rewards program, based on lifetime value

customers

| | |
|---------|-------|
| Annette | 9700 |
| Bruce | 23500 |
| Charles | 17800 |
| Dustin | 21250 |
| Eva | 8500 |
| Felix | 9300 |
| Glynn | 27800 |
| Henry | 8900 |
| Ian | 43800 |
| Jeff | 29100 |
| Kai | 34000 |
| Laura | 7800 |
| Mirko | 24200 |

```
grunt> SPLIT customers INTO
      gold_program IF ltv >= 25000,
      silver_program IF ltv >= 10000
      AND ltv < 25000;
```

```
grunt> DUMP gold_program;
(Glynn,27800)
(Ian,43800)
(Jeff,29100)
(Kai,34000)
```

```
grunt> DUMP silver_program;
(Bruce,23500)
(Charles,17800)
(Dustin,21250)
(Mirko,24200)
```



Pig Parameters



Specifying Parameters in Script

- Instead of hardcoding values, Pig allows you to use parameters

- These are replaced with specified values at runtime

```
allsales = LOAD '$INPUT' AS (name, price);  
bigsales = FILTER allsales BY price > $MINPRICE;  
  
bigsales_name = FILTER bigsales BY name == '$NAME';  
STORE bigsales_name INTO '$NAME';
```

- Then specify the values on the command line

```
$ pig -p INPUT=sales -p MINPRICE=999 \  
-p NAME='Jo Anne' reporter.pig
```



Specifying Parameter Values

- You can also specify parameter values in a text file

- An alternative to typing each one on the command line

```
INPUT=sales
```

```
MINPRICE=999
```

```
# comments look like this
```

```
NAME='Alice'
```

- Use -m filename option to tell Pig which file contains the values

- Parameter values can be defined with the output of a shell command

- For example, to set MONTH to the current month:

```
MONTH=`date +'%m'`      # returns 03 for March, 05 for May
```





Spark

Introduction

Data Access Speed & Bottlenecks

- Data transfer speeds within a node:
 - CPU to RAM: 10 GB/sec
 - CPU to HDD: 0.1 GB/sec
 - CPU to SSD: 0.6 GB/sec
- Data transfer speeds between nodes:
 - 0.1 GB/sec to 1 GB/sec
- What data transfers exist on Hadoop?



MapReduce

- MapReduce: original scalable processing engine
 - Disk-based data processing framework (HDFS files)
 - Persists intermediate results to disk
 - Best for ETL-like workloads (batch processing)
 - Data is reloaded from the disk with every query (slower I/O)
 - Not appropriate for iterative or stream processing workloads
- Enters Spark!
 - Memory based data processing framework → keeps intermediate results in memory
 - Leverages distributed memory
 - Remembers operations applied to a dataset
 - Data-locality-based computation results in high performance
 - Best for both iterative (or stream processing) and batch workloads



Spark

- Originally developed at the University of California, Berkeley's AMPLab
- Spark is a unified engine designed for massive parallel processing (MPP) on-premise in data centers
- Spark offers in-memory storage for intermediate computations
 - Making it much faster than Hadoop MR
- Spark offers 4 key characteristics:
 - Speed
 - Simplicity (ease of use)
 - Modularity
 - Extensibility



Spark - Speed

- Price and performance of CPUs and memory.
 - Commodity servers are cheap, with hundreds of gigabytes of memory, and multiple cores
- Iterative algorithms
 - Spark offers simplified data flow
 - Avoids materializing data on HDFS after each iteration
 - Intermediate results stored in memory (and limited disk I/O) gives Spark a huge performance boost.
- Spark builds its query computations as a directed acyclic graph (DAG);
 - Its DAG scheduler and query optimizer construct an efficient computational graph that can usually be decomposed into tasks that are executed in parallel across workers on the cluster.



Spark - Speed

- Example: Word Count

```

def mapper(text):
    myWordFreq1 = {}
    myList = text.lower().replace(",","").split()
    myList.sort()
    for word in myList:
        if word in myWordFreq1:
            myWordFreq1[word] = myWordFreq1[word] + 1
        else:
            myWordFreq1[word] = 1
    return myWordFreq1

def reducer(maps):
    myWordFreq1 = {}
    for myList in maps:
        for word in myList:
            if word in myWordFreq1:
                myWordFreq1[word] = myWordFreq1[word] + myList[word]
            else:
                myWordFreq1[word] = myList[word]
    return myWordFreq1

```

```

lines = sc.textFile("hamlet.txt")
counts = lines.flatMap(lambda line: line.split(" "))
                  .map(lambda word: (word, 1))
                  .reduceByKey(lambda x, y: x + y)

```

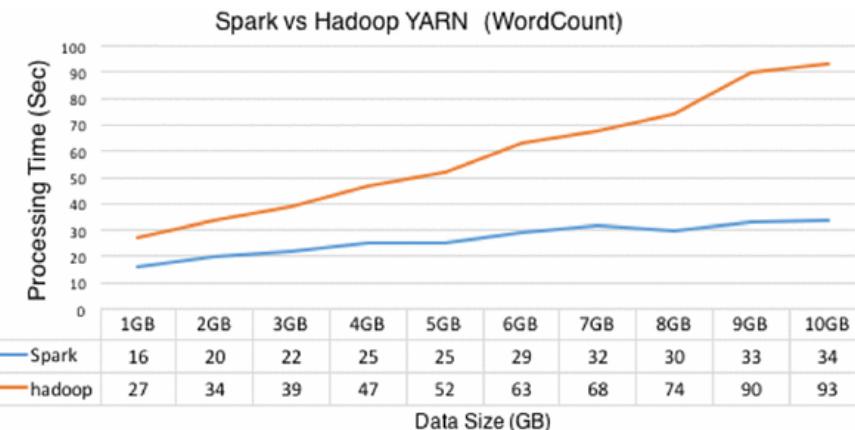


Image source: <https://link.springer.com/article/10.1007%2Fs10586-017-0846-z>



Spark - Simplicity

- Resilient Distributed Dataset (RDD)
 - Spark achieves simplicity by providing a fundamental abstraction of a simple logical data structure (RDD) upon which all other higher-level structured data abstractions, such as DataFrames and Datasets, are constructed.
- Programming model comprising of actions and operations
 - By providing a set of transformations and actions as operations, Spark offers a simple programming model that you can use to build big data applications in familiar languages (like R, Python, Java, etc.).



Spark - Modularity

- Spark operations can be applied across many types of workloads and expressed in any of the supported programming languages: Scala, Java, Python, SQL, and R.
- Spark offers unified libraries with well-documented APIs that include the following modules as core components: Spark SQL, Spark Structured Streaming, Spark MLlib, and GraphX, combining all the workloads running under one engine.
- You can write a single Spark application that can do it all
 - No need for distinct engines for disparate workloads
 - No need to learn separate APIs.
 - With Spark, you get a unified processing engine for your workloads.



Spark - Extensibility

- Spark focuses on its fast, parallel computation engine rather than on storage.
 - Unlike Hadoop, Spark decouples storage and compute.
 - You can process all data in memory.
 - You can use Spark to read data stored in diverse sources
 - e.g. Apache Hadoop, Apache Cassandra, Apache HBase, MongoDB, Apache Hive, RDBMSs, and more
 - Spark's DataFrameReaders and DataFrameWriters can also be extended to read data from other sources, such as Apache Kafka, Kinesis, Azure Storage, and Amazon S3.



Spark Modules

- “Spark offers four distinct components as libraries for diverse workloads: Spark SQL, Spark MLlib, Spark Structured Streaming, and GraphX.”
- “Each of these components is separate from Spark’s core fault-tolerant engine.”
 - “You use APIs to write your Spark application, and Spark converts this into a DAG that is executed by the core engine.”

Spark SQL and
DataFrames +
Datasets

Spark Streaming
(Structured
Streaming)

Machine Learning
MLlib

Graph
Processing
Graph X

Spark Core and Spark SQL Engine

Scala

SQL

Python

Java

R



Spark – SQL

- SQL:
 - Works well with structured data.
 - You can read data stored in an RDBMS table or from file formats with structured data (CSV, text, JSON, Avro, ORC, Parquet, etc.)
 - You can construct permanent or temporary tables in Spark.
 - When using Spark's Structured APIs in Java, Python, Scala, or R, you can combine SQL-like queries to query the data just read into a Spark DataFrame.
 - Spark SQL is ANSI SQL:2003-compliant, and it also functions as a pure SQL engine.

```
// In Scala
// Read data off Amazon S3 bucket into a Spark DataFrame
spark.read.json("s3://apache_spark/data/committers.json")
  .createOrReplaceTempView("committers")
// Issue a SQL query and return the result as a Spark DataFrame
val results = spark.sql("""SELECT name, org, module, release, num_commits
FROM committers WHERE module = 'mllib' AND num_commits > 10
ORDER BY num_commits DESC""")
```



Spark – Streaming

- Spark views a stream as a continually growing table, with new rows of data appended at the end.
 - Developers can merely treat this as a structured table and issue queries against it as they would a static table.
- Underneath the Structured Streaming model, the Spark SQL core engine handles all aspects of fault tolerance and late-data semantics, allowing developers to focus on writing streaming applications with relative ease.

```
# In Python
# Read a stream from a local host
from pyspark.sql.functions import explode, split
lines = (spark
    .readStream
    .format("socket")
    .option("host", "localhost")
    .option("port", 9999)
    .load())

# Perform transformation
# Split the lines into words
words = lines.select(explode(split(lines.value, " ")).alias("word"))

# Generate running word count
word_counts = words.groupBy("word").count()

# Write out to the stream to Kafka
query = (word_counts
    .writeStream
    .format("kafka")
    .option("topic", "output"))
```



Spark – MLlib

- MLlib provides many popular machine learning algorithms built on high-level DataFrame-based APIs to build models.
 - These APIs allow you to extract or transform features, build pipelines (for training and evaluating), and persist models (for saving and reloading them) during deployment.

```
# In Python
from pyspark.ml.classification import LogisticRegression
...
training = spark.read.csv("s3://...")
test = spark.read.csv("s3://...")

# Load training data
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Predict
lrModel.transform(test)
...
```

Image source: <https://www.oreilly.com/library/view/learning-spark-2nd/9781492050032/ch01.html>

More information: <https://spark.apache.org/docs/latest/ml-guide.html>



Spark – GraphX

- GraphX is a library for manipulating graphs
 - e.g., social network graphs, routes and connection points, or network topology graphs
- Performs graph-parallel computations.
 - Offers standard graph algorithms for analysis, connections, and traversals
 - Available algorithms include PageRank, Connected Components, and Triangle Counting.

```
// In Scala
val graph = Graph(vertices, edges)
messages = spark.textFile("hdfs://...")
val graph2 = graph.joinVertices(messages) {
    (id, vertex, msg) => ...
}
```



Programming Components

Spark



Spark - Components

- Driver
 - SparkContext
- Resilient Distributed Datasets (RDDs)
- Data Operations
 - Transformations
 - Actions



SparkContext

- `SparkContext`: establishes connections with spark cluster
 - Every Spark application requires a spark context: the main entry point to the Spark API
 - Spark Shell provides a preconfigured Spark Context called “`sc`”

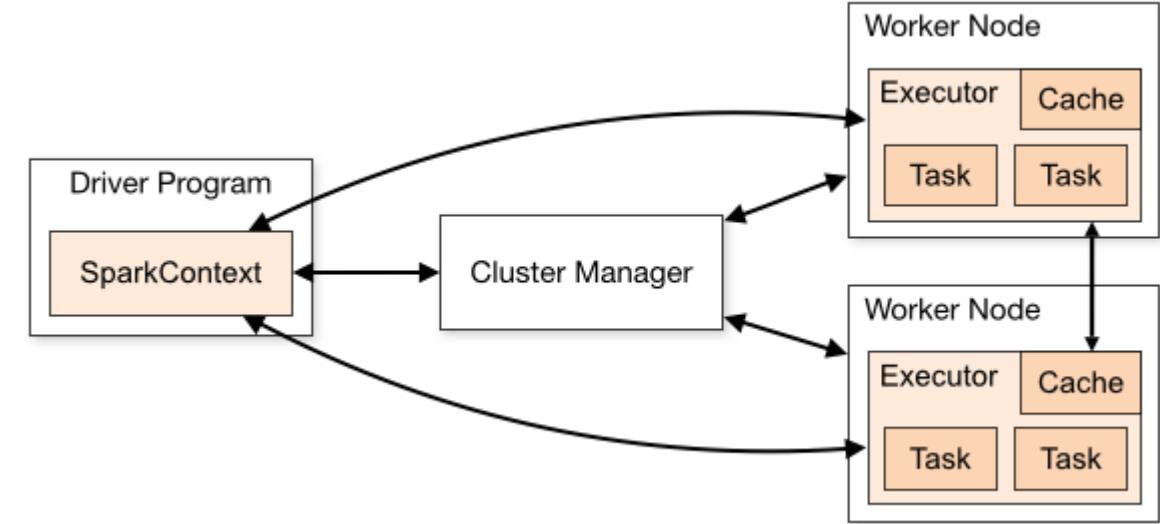


Image source: <https://spark.apache.org/docs/latest/cluster-overview.html>

RDDs

- RDD (Resilient Distributed Dataset) is a fundamental unit of data in Spark: An Immutable collection of objects that can be operated on “in parallel”
- Resilient Distributed Datasets (RDDs)
 - Collections of objects spread across a cluster
 - Stored in RAM or on Disk
 - Fall back to disk when dataset does not fit in memory
 - Built through parallel transformations
 - Automatically rebuilt on failure
- RDDs can be created from:
 - File(s)
 - Data in RAM
 - Data in another RDD



Operations - Transformation

- Driver applications comprises of transformations on distributed datasets
 - Collections of objects spread across a Memory caching layer(cluster) that stores data in a distributed, fault-tolerant cache
 - Built through parallel transformations (map, filter, group-by, join, etc.)
- Transformations (e.g., map, filter, groupBy)
 - Set of operations on an RDD that define how data should be transformed
 - Application of a transformation to an RDD yields a new RDD
 - because RDDs are immutable
 - Examples: map(), filter(), groupByKey(), sortByKey(), etc.



Operations - Actions

- Actions (e.g., count, collect, save)
 - Store data to an external data source (e.g., HDFS)
 - Fetch data from an RDD (post-transformation chain)
 - Return results to driver
- Some common actions
 - count() - return the number of elements
 - take(n) - return an array of the first n elements
 - collect() - return an array of all elements
 - saveAsTextFile(file) - save to text file(s)



Spark Application Anatomy

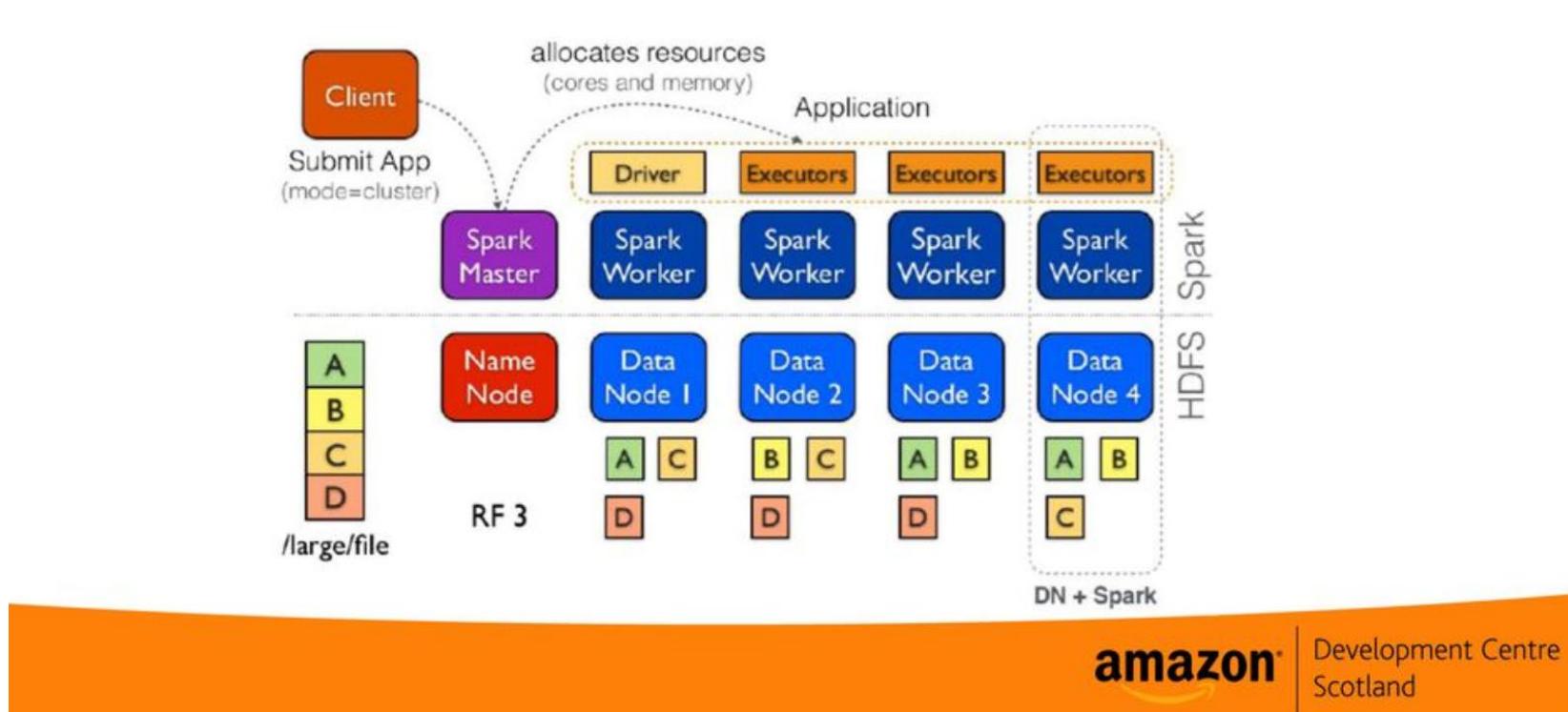


Image Source: <https://slideplayer.com/slide/11824985/>



Full-text search of Wikipedia

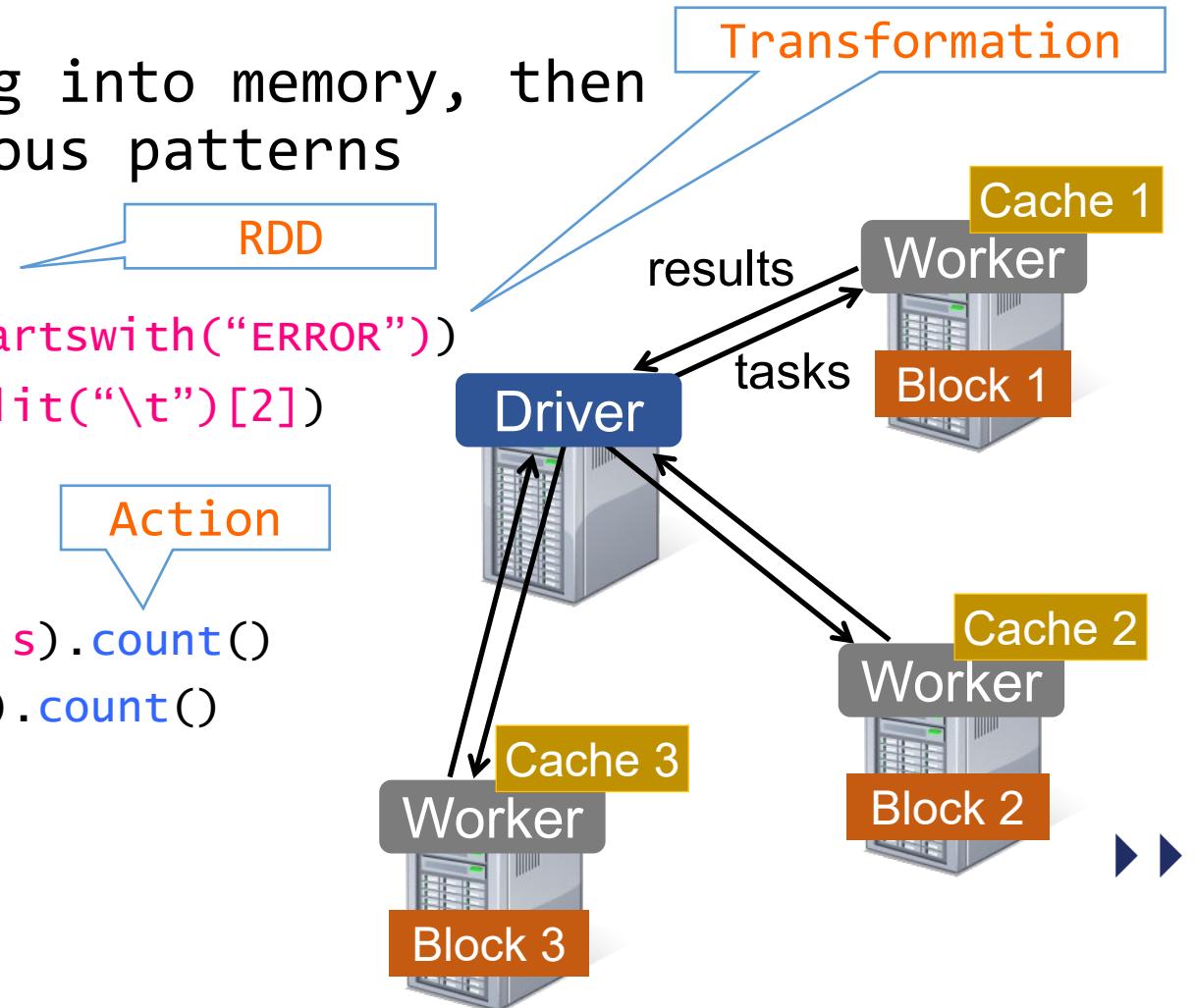
- 60GB on 20 EC2 machine
- 0.5 sec vs. 20s for on-disk

Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()  
messages.filter(lambda s: "php" in s).count()  
...
```



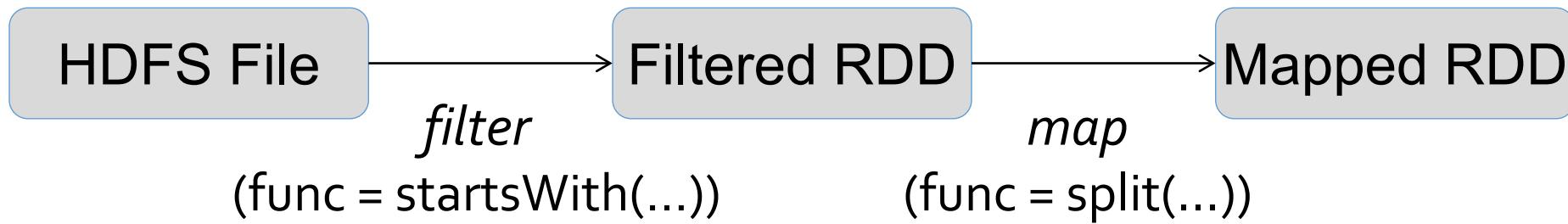
Source: Patrick Wendell @Databricks



Fault Recovery

- RDDs track lineage information that can be used to efficiently recompute lost data

```
msgs = textFile.filter(lambda s: s.startswith("ERROR"))
                 .map(lambda s: s.split("\t")[2])
```



PySpark

Python



Spark – create RDD

Creating an RDD:

```
# Turn a Python collection into an RDD
nums = sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
tripdata = sc.textFile("directory/*.txt")
tripdata = sc.textFile("hdfs://namenode:9000/path/tripdata")
tripdata = sc.textFile("s3://aws-path/data/tripdata.csv")

# Use existing Hadoop InputFormat (Java/Scala only)
text = sc.hadoopFile(keyClass, valClass, inputFmt, conf)
```



Spark – basic actions

Basic Actions

```
nums = sc.parallelize([1, 2, 3])  
  
# Retrieve RDD contents as a local collection  
nums.collect() # => [1, 2, 3]  
  
# Return first K elements  
nums.take(2) # => [1, 2]  
  
# Count number of elements  
nums.count() # => 3
```



Spark – functions (lambda)

Basic Transformations:

```
nums = sc.parallelize([1, 2, 3])

# Merge elements with an associative function
nums.reduce(lambda x, y: x + y) # => 6

# Pass each element through a function
squares = nums.map(lambda x: x*x) // {1, 4, 9}

# Selectively keep elements passing
even = squares.filter(lambda x: x % 2 == 0) // {4}

# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```



Spark – key-value

Spark's “distributed reduce” transformations operate on RDDs of key-value pairs. Some Key-Value Operations:

```
pets = sc.parallelize([('cat', 1), ('dog', 1), ('cat', 2)])
```

```
pets.reduceByKey(lambda x, y: x + y)      # => {('cat', 3), ('dog', 1)}
```

```
pets.groupByKey()      # => {('cat', [1, 2]), ('dog', [1])}
```

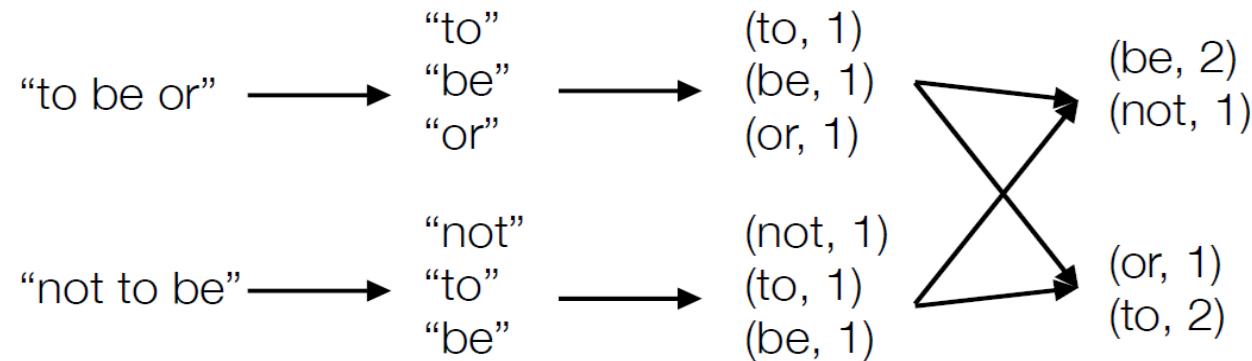
```
pets.sortByKey()      # => {('cat', 1), ('cat', 2), ('dog', 1)}
```



Spark – word count

```

lines = sc.textFile('/user/hadoop/hamlet.txt')
words = lines.flatMap(lambda line: line.split(' '))
wordfreq = words.map(lambda word: (word, 1))
counts = wordfreq.reduceByKey(lambda x, y: x + y)
counts.take(10)
    
```





Spark I

catchup

November 13, 2023



Spark II

Programming

November 13, 2023

Spark – programming

RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API that offers transformations and actions.

DataFrame is an immutable distributed collection of data. Unlike RDD, data is organized into named columns, like a table in a relational database.

Designed to make large dataset processing easier, DataFrame allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction.

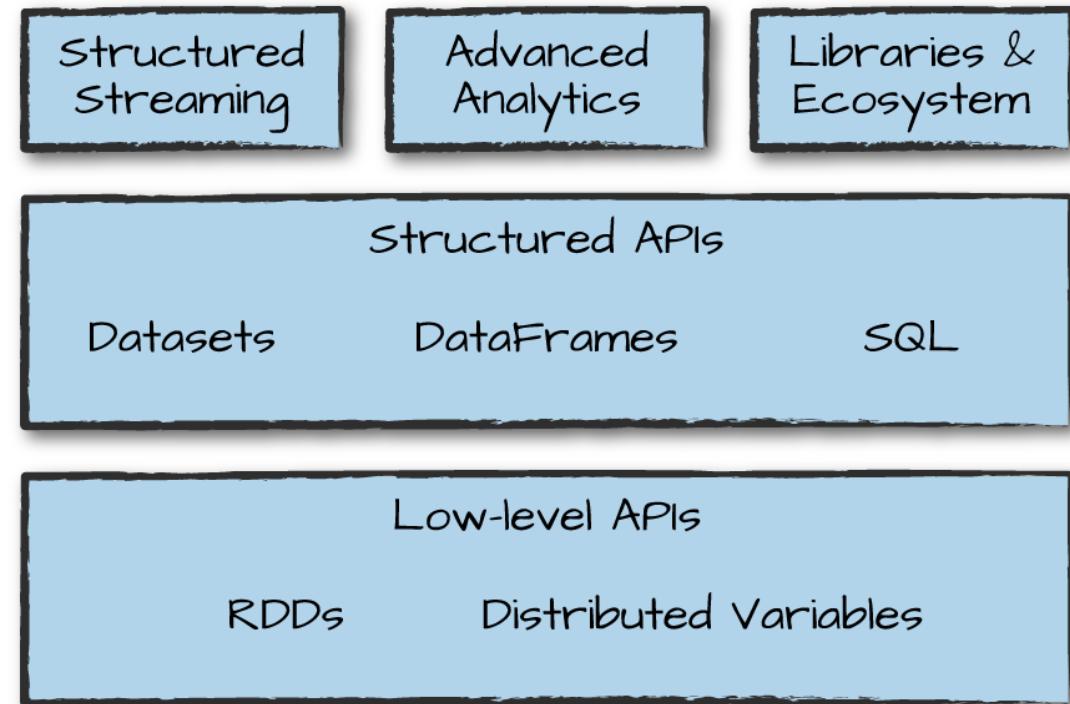


Image source: <https://www.oreilly.com/library/view/spark-the-definitive/9781491912201/ch03.html>



PySpark

Low-level API



Rajiv Garg



SparkContext

- SparkContext is usually available by default in spark shell and pyspark.
- Creating SparkContext:
 - `conf = SparkConf().setAppName(appName).setMaster(master)`
 - SparkConf sets configuration for a Spark application. Used to set various Spark parameters as key-value pairs.
 - appName parameter is a name for your application to show on the cluster UI.
 - master is a Spark, Mesos or YARN cluster URL, or “local” to run in local mode.
 - `sc = SparkContext(conf=conf)`



Spark - Low-level API

- Transformations:
 - `map(func)`
 - `filter(func)`
 - `flatMap(func)`
 - `groupByKey([numPartitions])`
 - `reduceByKey(func, [numPartitions])`
 - `sortByKey([ascending], [numPartitions])`
 - `repartition(numPartitions)`
- More: <https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations>



Spark - Low-level API

- Actions:
 - collect()
 - count()
 - take(n)
 - saveAsTextFile(path)
- More: <https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions>



Spark – create RDD

Creating an RDD:

```
# Turn a Python collection into an RDD
nums = sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
tripdata = sc.textFile("directory/*.txt")
tripdata = sc.textFile("hdfs://namenode:9000/path/tripdata")
tripdata = sc.textFile("s3://aws-path/data/tripdata.csv")

# Use existing Hadoop InputFormat (Java/Scala only)
text = sc.hadoopFile(keyClass, valClass, inputFmt, conf)
```

sc: first thing a Spark program must do is to create a SparkContext object, which tells Spark how to access a cluster.

Parallelize: elements of a collection are copied to form RDD that can be operated on in parallel.



Spark – basic actions

Basic Actions

```
nums = sc.parallelize([1, 2, 3])  
  
# Retrieve RDD contents as a local collection  
nums.collect() # => [1, 2, 3]  
  
# Return first K elements  
nums.take(2) # => [1, 2]  
  
# Count number of elements  
nums.count() # => 3
```



Spark – functions (lambda)

Basic Transformations:

```

nums = sc.parallelize([1, 2, 3])

# Merge elements with an associative function
nums.reduce(lambda x, y: x + y) # => 6

# Pass each element through a function
squares = nums.map(lambda x: x*x) // {1, 4, 9}

# Selectively keep elements passing
even = squares.filter(lambda x: x % 2 == 0) // {4}

# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")

```

Spark core doesn't execute the transformation operations until an action is called (lazy evaluation)



Spark – key-value

Spark's “distributed reduce” transformations operate on RDDs of key-value pairs. Some Key-Value Operations:

```
pets = sc.parallelize([('cat', 1), ('dog', 1), ('cat', 2)])
```

```
pets.reduceByKey(lambda x, y: x + y)      # => {('cat', 3), ('dog', 1)}
```

```
pets.groupByKey()      # => {('cat', [1, 2]), ('dog', [1])}
```

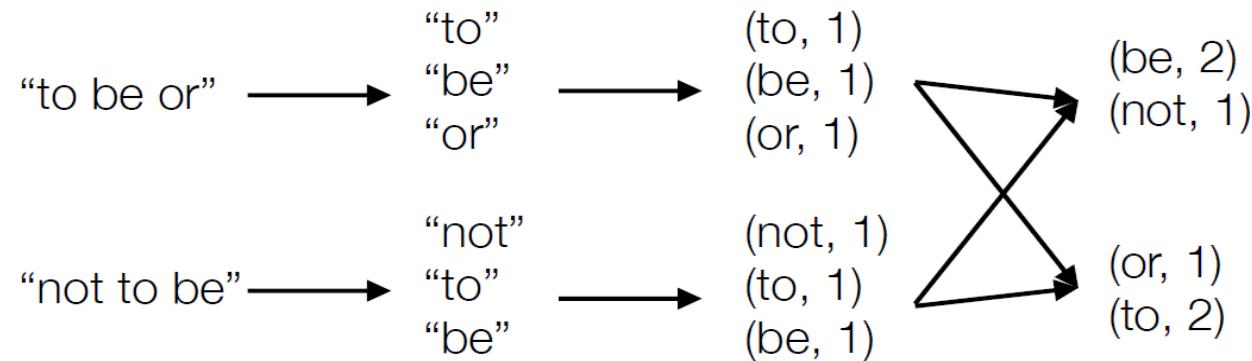
```
pets.sortByKey()      # => {('cat', 1), ('cat', 2), ('dog', 1)}
```



Spark – word count

```

lines = sc.textFile('/user/hadoop/hamlet.txt')
words = lines.flatMap(lambda line: line.split(' '))
wordfreq = words.map(lambda word: (word, 1))
counts = wordfreq.reduceByKey(lambda x, y: x + y)
counts.take(10)
    
```

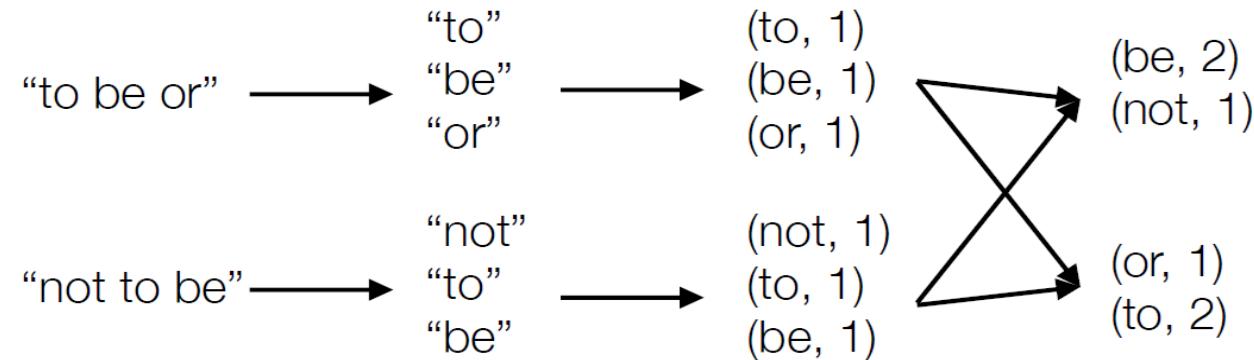


Example – word count

```

lines = sc.textFile("hamlet.txt")
counts = lines.flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda x, y: x + y)

```



Example – word search

```
from pyspark.sql.functions import col  
from pyspark.sql import Row  
  
lines = sc.textFile('s3://isom671data/hamlet.txt')  
df = lines.map(lambda r: Row(r)).toDF(["line"])  
hamlet = df.filter(col("line").like("%Hamlet%"))  
hamlet.count()
```



PySpark

Structured API



Rajiv Garg



DataFrame

- Same as the DataFrame in Python and R
 - Represents the data in a tabular form
 - Can be read using structured API

```
flightdata = spark.read.option("inferSchema",  
"true").option("header",  
"true").csv("s3://isom671data/flightdata/2015-  
summary.csv")
```

No data read/loaded – just peeked into first few rows to understand schema

```
flightdata.take(3)
```

```
In [23]: flightdata = spark.read.option("inferSchema", "true").option("header", "true").csv("s3://isom671data/flightdata/2015-summa  
flightdata.take(3)
```

```
[Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Romania', count=15), Row(DEST_COUNTRY_NAME='United States',  
ORIGIN_COUNTRY_NAME='Croatia', count=1), Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Ireland', count=34  
4)]
```

```
In [24]: fd = sc.textFile('s3://isom671data/flightdata/2015-summary.csv')  
fd.take(3)
```

```
['DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count', 'United States,Romania,15', 'United States,Croatia,1']
```



Structured API Execution

- Code execution process on Spark cluster:
 - Execute a DataFrame code
 - If code valid, create logical plan for execution
 - Convert the logical plan to physical plan while optimizing
 - Check for different execution strategies for cost (CPU, I/O, network)
 - Executes the best physical plan

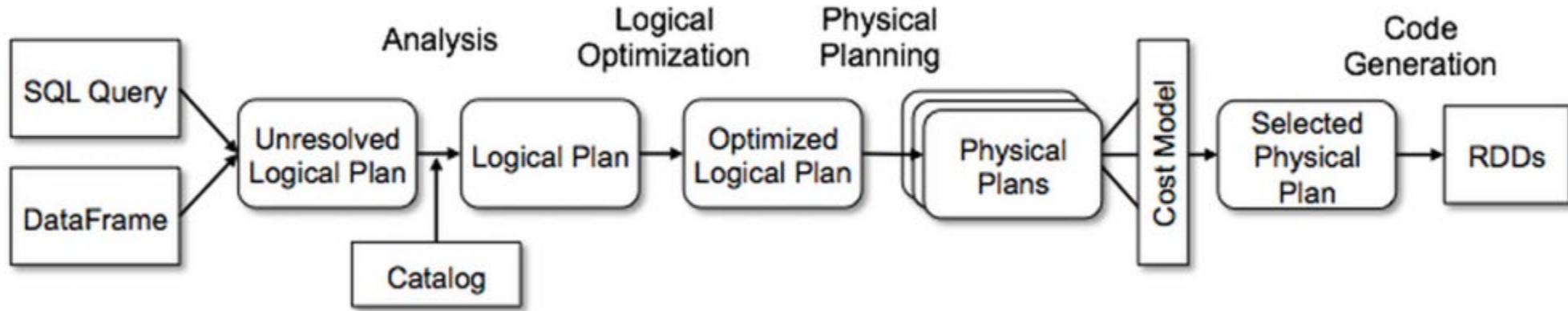


Image source: <https://databricks.com/glossary/catalyst-optimizer>



Logical/Physical Plan (.explain)

- `flightdata = spark.read.option("inferSchema", "true").option("header", "true").csv(" s3://isom671data/flightdata/2015-summary.csv ")`
- `flightdata.sort("count").explain()`

```
== Physical Plan ==
AdaptiveSparkPlan(isFinalPlan=false)
+- Sort [count#81 ASC NULLS FIRST], true, 0
  +- Exchange rangepartitioning(count#81 ASC NULLS FIRST, 1000)
    +- FileScan csv [DEST_COUNTRY_NAME#79,ORIGIN_COUNTRY_NAME#80,count#81] Batched: false, Format:
CSV, Location: InMemoryFileIndex[s3://isom671data/flightdata/2015-summary.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<DEST_COUNTRY_NAME:string,ORIGIN_COUNTRY_NAME:string,count:int>
```



DataFrame – infer schema

- ```
flightdata = spark.read.option("inferSchema", "true").option("header", "true").csv(" s3://isom671data/flightdata/2015-summary.csv ")
```
- ```
flightdata.printSchema()
```

```
root
|-- DEST_COUNTRY_NAME: string (nullable = true)
|-- ORIGIN_COUNTRY_NAME: string (nullable = true)
|-- count: integer (nullable = true)
```
- Python Data Types:
 - https://spark.apache.org/docs/2.4.4/sql-reference.html#tab_python_0



DataFrame – manual schema

```
from pyspark.sql.types import StructField, StructType, StringType, LongType  
myManualSchema = StructType([  
    StructField("DEST_COUNTRY_NAME", StringType(), True),  
    StructField("ORIGIN_COUNTRY_NAME", StringType(), True),  
    StructField("count", LongType(), True) ])
```

Boolean flag in StructField specifies if a column can contain null/missing values

```
df = spark.read.format("csv").schema(myManualSchema).load("s3://isom671data/flightdata/2015-  
summary.csv")  
df.printSchema()
```

```
root  
|-- DEST_COUNTRY_NAME: string (nullable = true)  
|-- ORIGIN_COUNTRY_NAME: string (nullable = true)  
|-- count: long (nullable = true)
```



DataFrame – text files

- Spark SQL API enables:
 - `spark.read().text("file_name")` to read a file or directory of text files into a Spark DataFrame,
 - `dataframe.write().text("path")` to write to a text file
- `df1 = spark.read.text(path)`
Read a file with each line as a row
- `df2 = spark.read.text(path, lineSep=",")`
Read with different line delimiter
- `df3 = spark.read.text(path, wholertext=True)`
Read input file as a single row
- `df1.write.text("output")`
- `df1.write.text("output_compressed", compression="gzip")`



DataFrame – csv files

- Spark SQL API enables:
 - `spark.read().text("file_name")` to read a file or directory of text files into a Spark DataFrame,
 - `dataframe.write().text("path")` to write to a text file
 - `df = spark.read.csv(path)` Read csv file with each line as a row
 - `df2 = spark.read.option("delimiter", ";").csv(path)` Read csv file with different delimiter
 - `df3 = spark.read.option("delimiter", ";").option("header", True).csv(path)` Read csv file with header as columns
 - `df4 = spark.read.options(delimiter=";", header=True).csv(path)` Read - multiple options
 - `df5 = spark.read.csv(folderPath)` Read ALL csv files in a folder
- `df1.write.csv("output")` ▶▶▶

More data sources (JSON, Hive, Parquet):
<https://spark.apache.org/docs/latest/sql-data-sources.html>

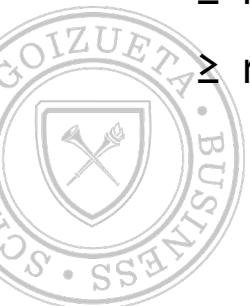


DataFrame - Columns & Rows

```
> from pyspark.sql.functions import col  
> col("someColumnName")  
  
> from pyspark.sql import Row  
> myRow = Row("USA", "Mexico", 2)  
> myRow[0]  
USA  
  
> myDf = spark.createDataFrame([myRow], myManualSchema)  
> myDf.show()
```

You can add more columns to dataframe using df.withColumn()

You can drop columns in a dataframe using df.drop().columns



More on Row and Col:

<https://sparkbyexamples.com/pyspark/pyspark-row-using-rdd-dataframe/>
<https://sparkbyexamples.com/pyspark/pyspark-column-functions/>

DataFrame - Columns & Rows

- Adding rows:
 - columns = ['DEST_COUNTRY', 'ORIGIN_COUNTRY', 'count']
 - vals = [('USA', 'Canada', 5), ('Canada', 'USA', 3)]
- df2 = spark.createDataFrame(vals, columns)
- newRow = spark.createDataFrame([('USA', 'Mexico', 4)], columns)
- appended = df2.union(newRow)
- appended.show()

| DEST_COUNTRY | ORIGIN_COUNTRY | count |
|--------------|----------------|-------|
| USA | Canada | 5 |
| Canada | USA | 3 |
| USA | Mexico | 4 |



DataFrame – Expressions

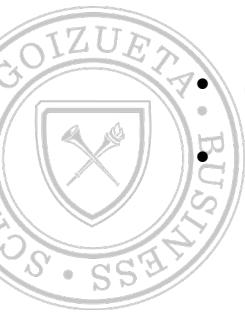
- `from pyspark.sql.functions import expr`
- `expr("(((someCol + 5) * 200) - 6) < otherCol")`
- `data = [("James","M"),("Michael","F"),("Jen","")]`
- `df = spark.createDataFrame(data = data, schema = ["name","gender"])`
- `df.withColumn("NG",expr(" name || ',' || gender")).show()`
- `df2=df.withColumn("gender", expr("CASE WHEN gender = 'M' THEN 'Male' " +
"WHEN gender = 'F' THEN 'Female' ELSE 'unknown' END")).show()`

| name | gender | NG |
|---------|--------|-----------|
| James | M | James,M |
| Michael | F | Michael,F |
| Jen | | Jen, |

| name | gender |
|---------|---------|
| James | Male |
| Michael | Female |
| Jen | unknown |

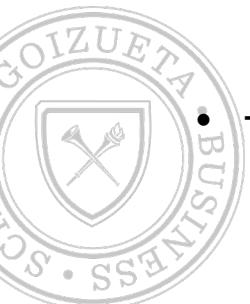


More: <https://sparkbyexamples.com/pyspark/pyspark-sql-expr-expression-function/>



DataFrame – Select & SelectExpr

- Spark select() function is used to select columns from the DataFrame.
 - PySpark select() is a transformation function and returns a new DataFrame
- ```
flightdata = spark.read.option("inferSchema", "true").option("header", "true").csv("s3://isom671data/flightdata/2015-summary.csv")
```
- ```
flightdata.select(expr("DEST_COUNTRY_NAME AS destination")).show(2)
```
- ```
flightdata.selectExpr("*", "(DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME) as withinCountry").show(5)
```
- ```
flightdata.selectExpr("avg(count)", "count(distinct(DEST_COUNTRY_NAME))").show()
```



More: <https://sparkbyexamples.com/pyspark/select-columns-from-pyspark-dataframe/>

Word Count – Structured API

- `from pyspark.sql.functions import split, explode`
 - `shakespeareDF = spark.read.text('/user/hadoop/hamlet.txt')`
 - `shakeWordsSplitDF = (shakespeareDF.select(split(shakespeareDF.value, '\s+').alias('split')))`
 - `shakeWordsSingleDF = (shakeWordsSplitDF.select(explode(shakeWordsSplitDF.split).alias('word')))`
 - `shakeWordsDFCount = shakeWordsSingleDF.count()`
- ```
print(shakeWordsDFCount)
```



More info: <https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html>



# Partitions

- Spark creates partitions of data based on the size of data and available nodes/resources.
    - You can have multiple partitions on the same node
    - You need to repartition if volume of data is changing significantly
  - `df.rdd.getNumPartitions()`
  - `df.repartition(5)`
  - `df.repartition(col("DEST_COUNTRY_NAME"))`
- `df.repartition(5, col("DEST_COUNTRY_NAME")).coalesce(2)`
- You can define number of partitions or partitions based on a column
- Coalesce reduces the number of partitions
- ▶▶▶



# PySpark

Example - PageRank



Rajiv Garg



# Online Search

- How does online search work?
  - Index (spiders)
  - Store (map-reduce)
  - Query (Google DFS)
  - Present (PageRank)



# PageRank

- Simple Rank:

- Where:
- $R(u)$ : rank of page  $u$
- $R(v)$ : rank of page  $v$  (element of  $B_u$ )
- $B_u$ : Back-links
- $N_v$ : Forward-links

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

- Generalizing:

- $R = C^*A^*R$
- Where:
- $R$  is a Eigen vector (rank matrix)
- $C$  is the eigenvalue

A is the square matrix of links weights between pages

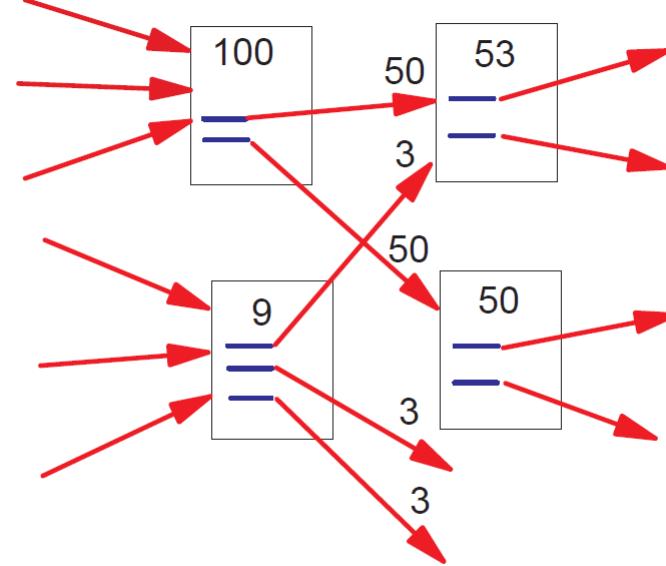


Figure 2: Simplified PageRank Calculation



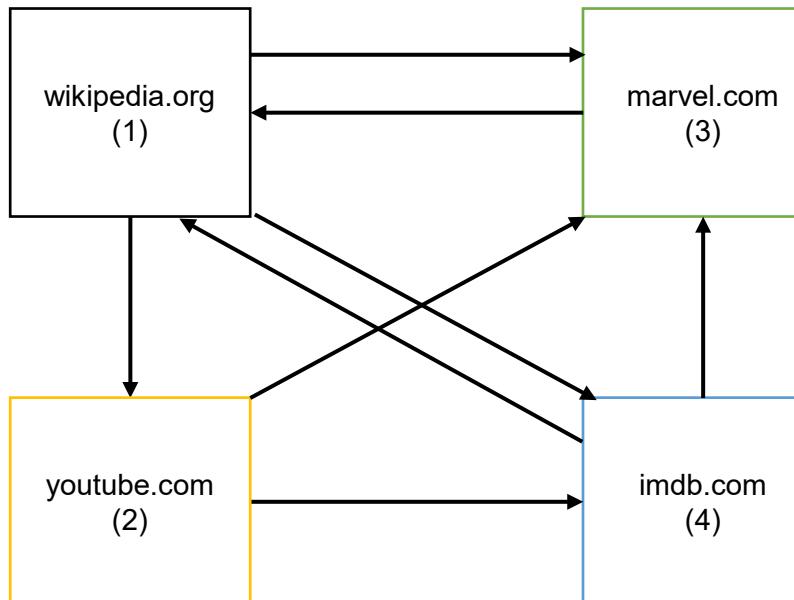
# PageRank



You**Tube**



**WIKIPEDIA**  
The Free Encyclopedia



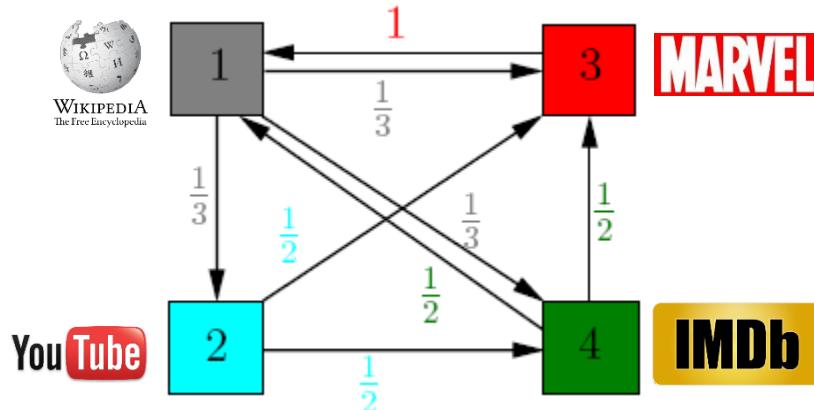
**MARVEL**

**IMDb**



# PageRank

- Each link is assigned a value (based on out-links)



$$\begin{cases} x_1 = 1 \cdot x_3 + \frac{1}{2} \cdot x_4 \\ x_2 = \frac{1}{3} \cdot x_1 \\ x_3 = \frac{1}{3} \cdot x_1 + \frac{1}{2} \cdot x_2 + \frac{1}{2} \cdot x_4 \\ x_4 = \frac{1}{3} \cdot x_1 + \frac{1}{2} \cdot x_2 \end{cases}$$

x1	x2	x3	x4
1	1	1	1
1.50	0.33	1.33	0.83
1.75	0.50	1.08	0.67
1.42	0.58	1.17	0.83
1.58	0.47	1.18	0.76
1.56	0.53	1.15	0.76
1.53	0.52	1.17	0.78
1.56	0.51	1.16	0.77
1.55	0.52	1.16	0.77
1.55	0.52	1.16	0.78
1.55	0.52	1.16	0.77
1.55	0.52	1.16	0.77

- In simplest terms, value of each page is (solution to the simultaneous equations based on back-links):

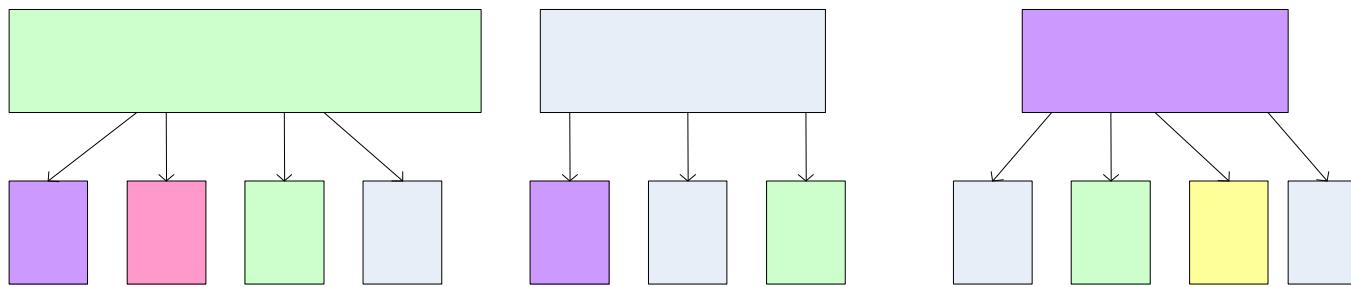


Image source:

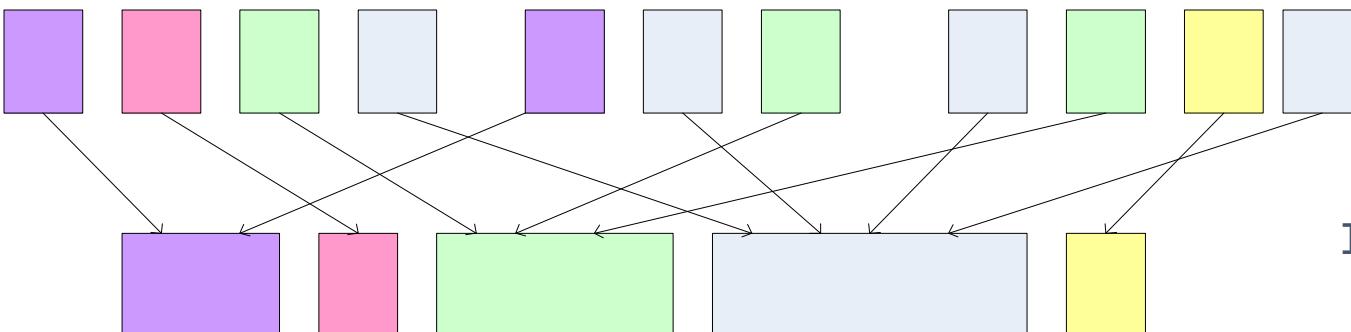
<http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html>

# PageRank using MapReduce

Map: distribute PageRank “credit” to link targets



Reduce: gather up PageRank “credit” from multiple sources to compute new PageRank value



Iterate until convergence



Source of Image: Lin 2008



# PageRank in Spark

**Get/Process Data:**

```
lines = sc.textFile("s3://isom671data/spark/pagerank_urls.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))
```

```
def computeContribs(urls, rank):
 num_urls = len(urls)
 for url in urls:
 yield (url, rank / num_urls)
```

Recall: map vs flatMap

Recall: reduceByKey

**Iteratively compute ranks:**

```
for iteration in range(10):
 # Calculates URL contributions to the rank of other URLs.
 contribs = links.join(ranks).flatMap(lambda url_urls_rank:
 computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))

 # Re-calculates URL ranks based on neighbor contributions.
 ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank: rank * 0.85 + 0.15)
```

```
ranks.collect()
```

