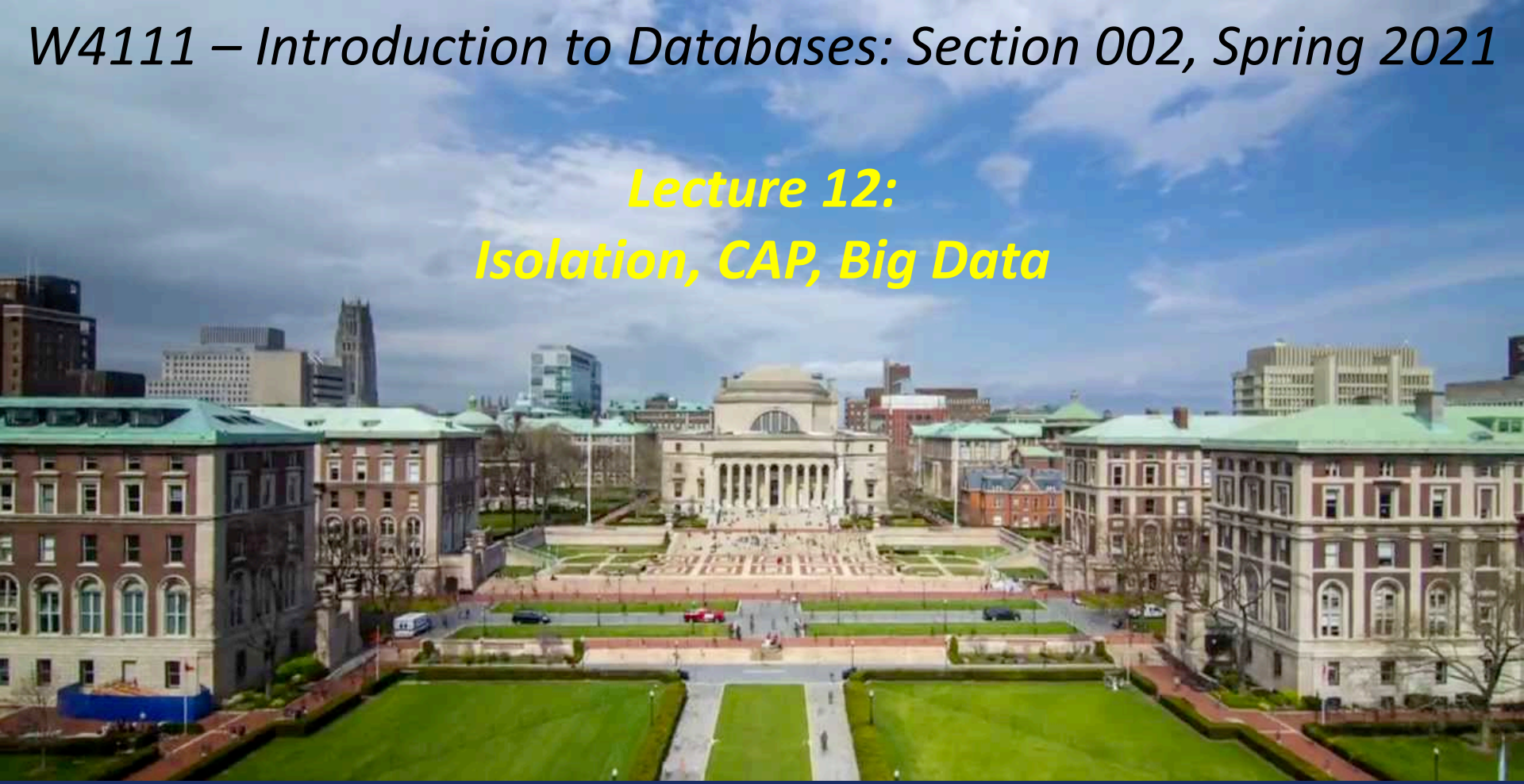
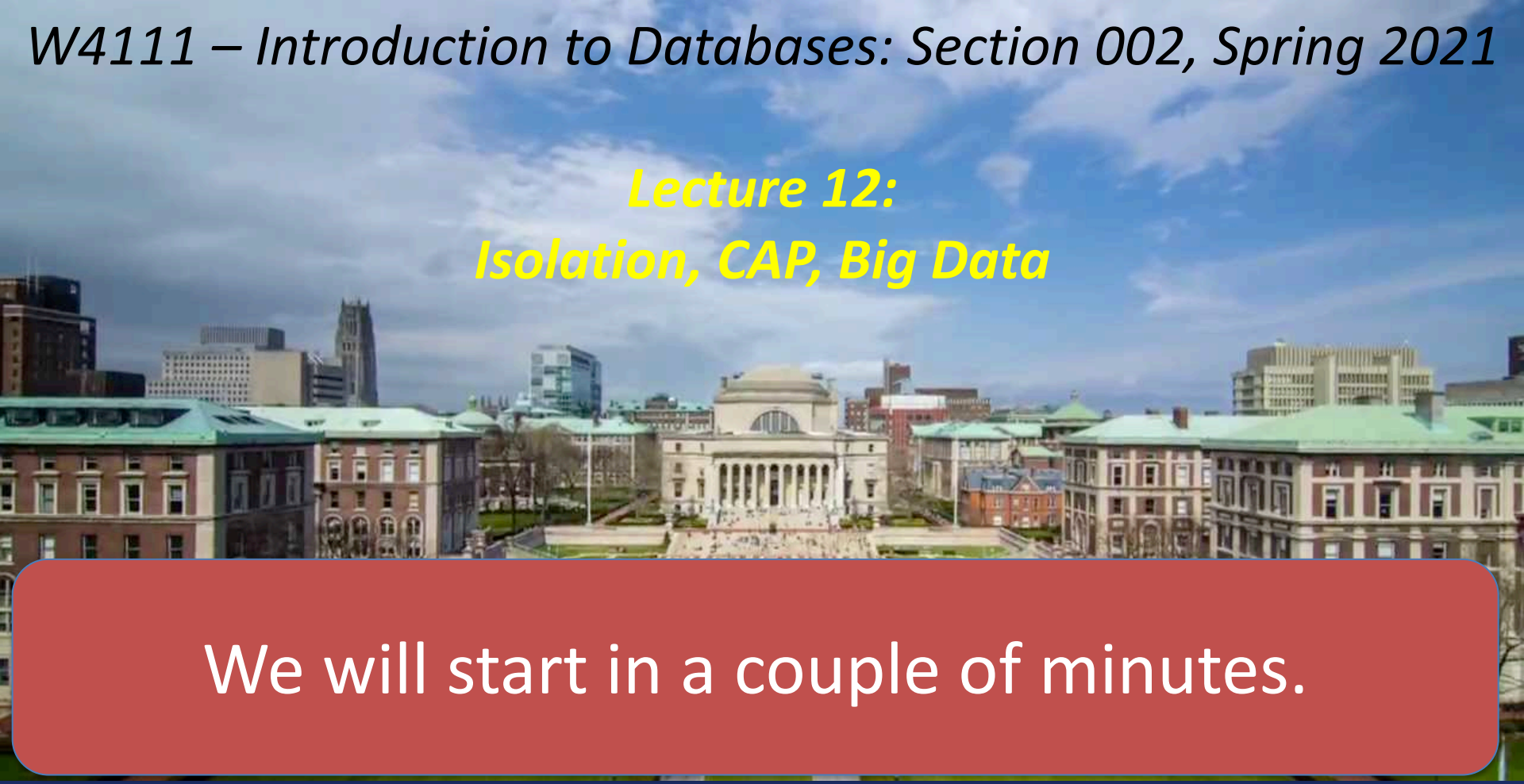
W4111 – Introduction to Databases: Section 002, Spring 2021



Lecture 12:
Isolation, CAP, Big Data

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

W4111 – Introduction to Databases: Section 002, Spring 2021



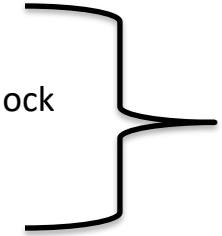
Lecture 12:
Isolation, CAP, Big Data

We will start in a couple of minutes.

Contents

Contents

- A Quick Completion of Transactions:
  - Isolation, Serializability, Aborts, Deadlock
  - CAP “Theorem”
  - ACID versus BASE
- A Quick Overview of Big Data:
  - ETL, Data Warehouse, Data Lake
  - Map-Reduce, Hadoop
  - Apache Spark
- Online Analytical Processing – OLAP
  - Start Schema; Facts and Dimensions
  - Cubes
  - Operations
  - Pivot Tables
- Discussion, End of Semester Roadmap



- There will be some written questions on the take home final exam.
- You can look up the answers.
- The purpose is to just reinforce basic concepts.

~~Transactions~~

Isolation

~~Recovery~~

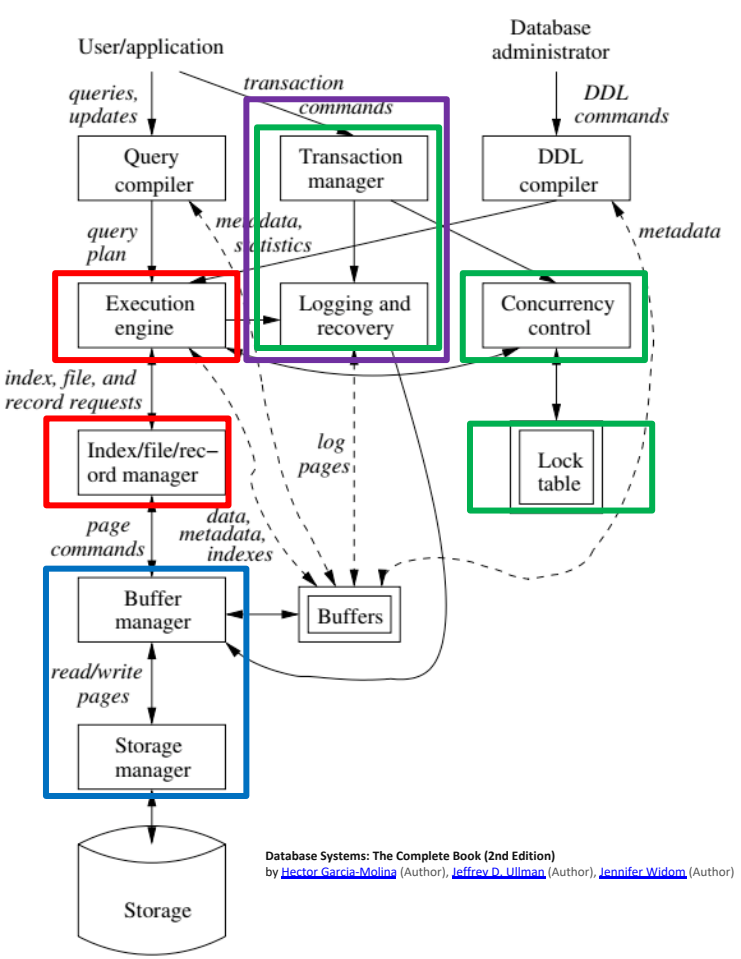
CAP, BASE

Scaling Models

Reminder

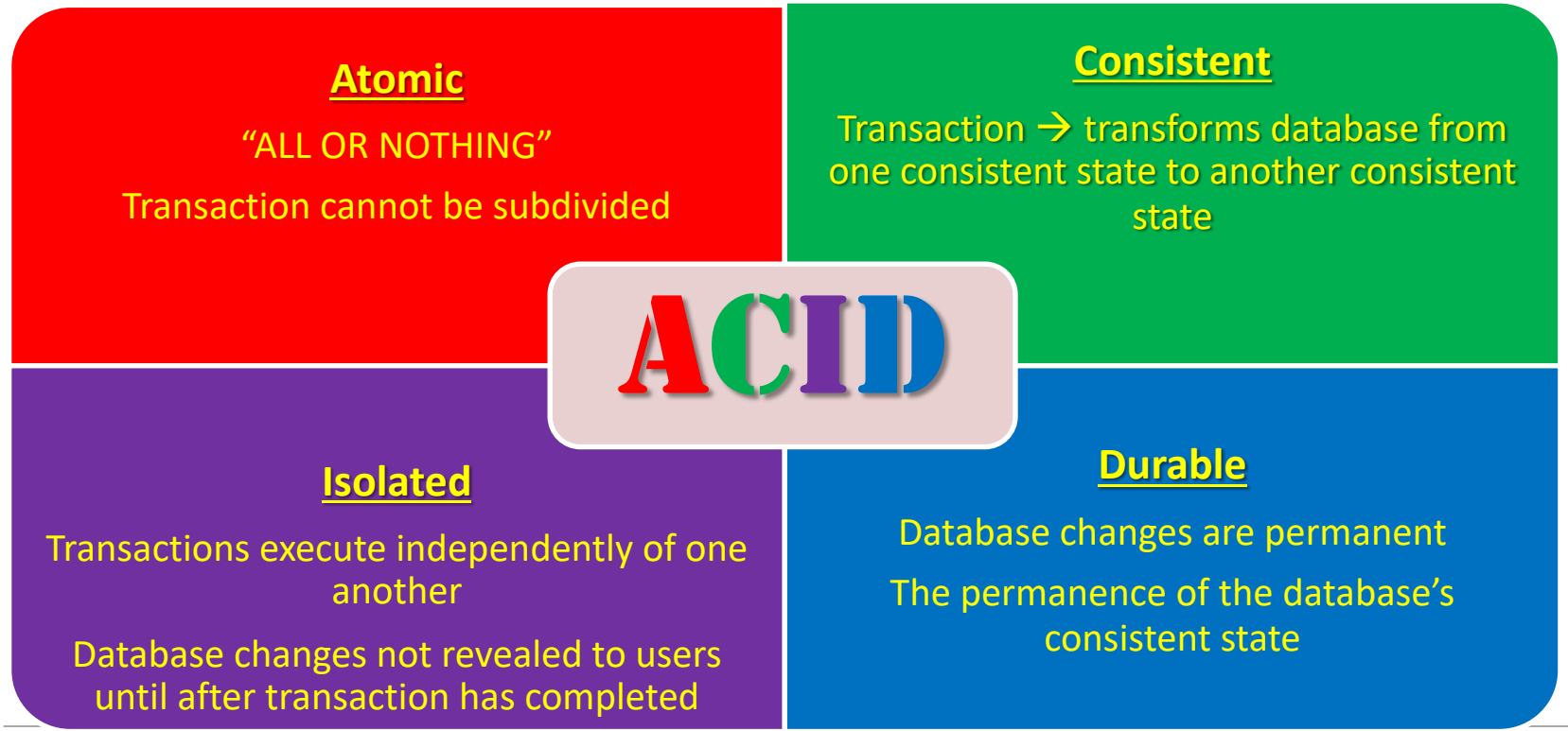
Data Management

- Find things quickly.
- Load/Save quickly.
- Control access (Isolation)
- Durability



Core Transaction Concept is ACID Properties

http://slideplayer.com/slide/9307681



ACID Properties (httpwww.tutorialspoint.com/dbms/dbms_transaction.htm)

A *transaction* is a very small unit of a program and it may contain several low-level tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** – This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** – The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** – The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** – In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Transaction Concept

Reminder

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.
- E.g., transaction to transfer $50 from account A to account B:

  BEGIN TRANSACTION {

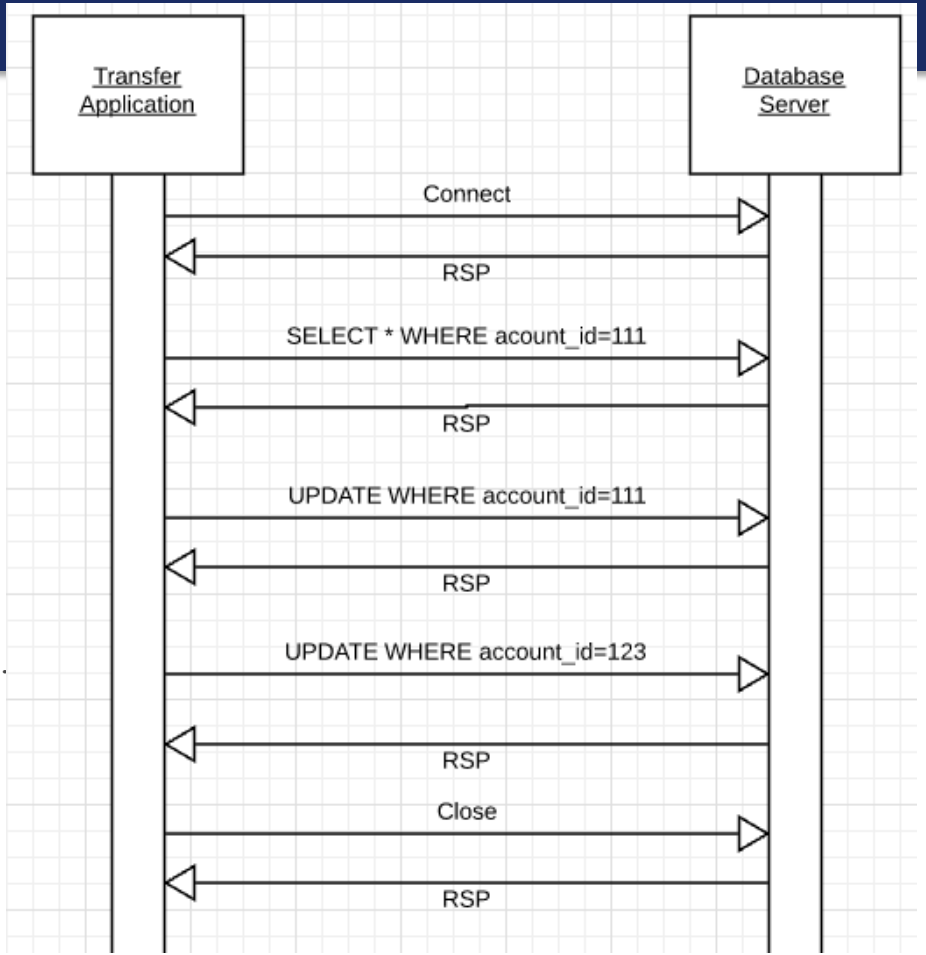
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)

  COMMIT or ROLLBACK }
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

Isolation

Isolation

- Transfer $50 from
  - account_id=111 to
  - account_id=123
- Requires 3 SQL statements
  - SELECT from 111 to check balance >= $50
  - UPDATE account_id=111
  - UPDATE account_id=123
- There are some interesting scenarios
  - Two different programs read the balance ($51)
  - And decide removing $50 is OK.
- DB constraints can prevent the conflict from happening, but …
  - There are more complex scenarios that constraints do not prevent.
  - Not ALL databases support constraints.
  - The "correct" execution should be that
    - One transaction responds "insufficient funds"
    - Before attempting transfer instead of after attempting.



Isolation

- Consider two **simultaneous** transfer transactions T1 and T2.
- There are two equally ***correct*** executions
  1. T1 transfers, T2 responds "insufficient funds" and does not attempt transfer
  2. T2 transfers, T1 responds "insufficient funds" and does not attempt transfer
- Each correct simultaneous execution is equivalent to a serial (sequential) execution schedule
  - (1) Execute T1, Execute T2
  - (2) Execute T2, Execute T1
- NOTE:
  - We are focusing on ***correctness*** not
  - ***Fairness:***
    - We do not care which transaction was actually submitted first.
    - And probably do not know due to networking, etc.

Serializability

"In concurrency control of databases, transaction processing (transaction management), and various transactional applications (e.g., transactional memory[3] and software transactional memory), both centralized and distributed, a transaction schedule is **serializable** if its outcome (e.g., the resulting database state) is equal to the outcome of its transactions executed serially, i.e. without overlapping in time. Transactions are normally executed concurrently (they overlap), since this is the most efficient way. Serializability is the major correctness criterion for concurrent transactions' executions. It is considered the highest level of isolation between transactions, and plays an essential role in concurrency control. As such it is supported in all general purpose database systems." (https://en.wikipedia.org/wiki/Serializability)

Schedule

18.1.1 Schedules

A *schedule* is a sequence of the important actions taken by one or more transactions. When studying concurrency control, the important read and write actions take place in the main-memory buffers, not the disk. That is, a database element A that is brought to a buffer by some transaction T may be read or written in that buffer not only by T but by other transactions that access A.

T_1	T_2
READ(A,t)	READ(A,s)
t := t+100	s := s*2
WRITE(A,t)	WRITE(A,s)
READ(B,t)	READ(B,s)
t := t+100	s := s*2
WRITE(B,t)	WRITE(B,s)

Figure 18.2: Two transactions

Garcia-Molina et al.

Serializable

- Assume there are three
  - concurrently executing transactions
  - T1, T2 and T3
- The transaction manager
  - Enables concurrent execution
  - But schedules individual operations
  - To ensure that the final DB state
  - Is *equivalent* to one of the following schedules
    - T1, T2, T3
    - T1, T3, T2
    - T2, T1, T3
    - T2, T3, T1
    - T3, T1, T2
    - T3, T2, T1

  Concurrent execution was *serializable.*

18.1.2 Serial Schedules

A schedule is *serial* if its actions consist of all the actions of one transaction, then all the actions of another transaction, and so on. No mixing of the actions is allowed.

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Figure 18.3: Serial schedule in which T_1 precedes T_2

Serializability (en.wikipedia.org/wiki/Serializability)

- **Serializability** is used to keep the data in the data item in a consistent state. Serializability is a property of a transaction schedule (history). It relates to the *isolation* property of a database transaction.
- **Serializability** of a schedule means equivalence (in the outcome, the database state, data values) to a *serial schedule* (i.e., sequential with no transaction overlap in time) with the same transactions. It is the major criterion for the correctness of concurrent transactions' schedule, and thus supported in all general purpose database systems.
- **The rationale behind serializability** is the following:
  - If each transaction is correct by itself, i.e., meets certain integrity conditions,
  - then a schedule that comprises any *serial* execution of these transactions is correct (its transactions still meet their conditions):
    - "Serial" means that transactions do not overlap in time and cannot interfere with each other, i.e, complete *isolation* between each other exists.
    - Any order of the transactions is legitimate, (…)
    - As a result, a schedule that comprises any execution (not necessarily serial) that is equivalent (in its outcome) to any serial execution of these transactions, is correct.

Locking and Isolation

- Strict Two-phase Locking (Strict 2PL) Protocol:
  - Each transaction must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.
  - All locks held by a transaction are released when the transaction completes
- (Non-strict) 2PL Variant: Transaction can release locks anytime, but cannot acquire locks after releasing any lock.
- If a transaction holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Strict 2PL allows only serializable schedules.
- Additionally, it simplifies transaction aborts.
- (Non-strict) 2PL also allows only serializable schedules, but involves more complex abort processing (cascading aborts).

Aborts

- If a transaction T_i is aborted, all its actions have to be undone.
  Not only that, if T_j reads an object last written by T_i, T_j must be aborted as well!
- Most systems avoid such *cascading aborts* by
  - Releasing a transaction's locks only at commit time.
  - If T_i writes an object, T_j can read this only after T_i commits.
- In order to *undo* the actions of an aborted transaction,
  - The DBMS maintains a *log* in which every write is recorded. Records contain:
    (Transaction ID, Block, Record Location, Before Value, After Value, ...)
  - Abort *undoes* the write by applying the *Before Value.*
  - This mechanism is also used to recover from system crashes:
    - All active transactions at the time of the crash are aborted when the system comes back up.
    - Any committed, changed values not *written to disk,* are *redone* using after image.

Some Other Concepts

- Deadlock:
  - The transaction manager blocks a transaction if it requests a lock the conflicts with an existing lock held by another transaction.
  - Assume T1 holds an S(A) share lock on A and T2 requests an exclusive lock X(A). → T1 *waits for* T2. T1 cannot progress until T2 releases the lock, commits or rollsback.
  - The *wait for* graph is a graph: Transactions are nodes.
    - There is an edge (T_i) → (T_j) if T_i cannot progress until T_j release a lock.
    - There is a *deadlock* if there is a cycle in the wait for graph.
- Cascading Abort: Suppose that:
  - T_j writes A and releases the lock before commit.
  - T_i reads (A), reads (B) and then writes C = A + B. The value of C depends on the uncommitted value of A.
  - If T_j aborts/rolls back, the value of C depends on an invalid value of A. ➔ T_i must also abort/rollback.
  - This is called a *cascading abort.*

MySQL (Locking) Isolation

13.3.6 SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION
    transaction_characteristic [, transaction_characteristic] ...

transaction_characteristic:
    ISOLATION LEVEL level
  | READ WRITE
  | READ ONLY

level:
     REPEATABLE READ
   | READ COMMITTED
   | READ UNCOMMITTED
   | SERIALIZABLE
```

- There are many application scenarios in which *serializable* is unnecessary.
- It is "overkill."
- Unnecessary use of serializable may result in
- lower concurrency and poor performance.

Scope of Transaction Characteristics

You can set transaction characteristics globally, for the current session, or for the next transaction:

- With the `GLOBAL` keyword, the statement applies globally for all subsequent sessions. Existing sessions are unaffected.
- With the `SESSION` keyword, the statement applies to all subsequent transactions performed within the current session.
- Without any `SESSION` or `GLOBAL` keyword, the statement applies to the next (not started) transaction performed within the current session. Subsequent transactions revert to using the `SESSION` isolation level.

Isolation Levels (https://en.wikipedia.org/wiki/Isolation_(database_systems)

Not all transaction use cases require 2PL and serializable execution. Databases support a a set of levels.

- **Serializable**
  - With a lock-based concurrency control DBMS implementation, serializability requires read and write locks (acquired on selected data) to be released at the end of the transaction. Also *range-locks* must be acquired when a SELECT query uses a ranged *WHERE* clause, especially to avoid the ***phantom reads*** phenomenon.
  - *The execution of concurrent SQL-transactions at isolation level SERIALIZABLE is guaranteed to be serializable. A serializable execution is defined to be an execution of the operations of concurrently executing SQL-transactions that produces the same effect as some serial execution of those same SQL-transactions. A serial execution is one in which each SQL-transaction executes to completion before the next SQL-transaction begins.*
- **Repeatable reads**
  - In this isolation level, a lock-based concurrency control DBMS implementation keeps read and write locks (acquired on selected data) until the end of the transaction. However, *range-locks* are not managed, so ***phantom reads*** can occur.
  - Write skew is possible at this isolation level, a phenomenon where two writes are allowed to the same column(s) in a table by two different writers (who have previously read the columns they are updating), resulting in the column having data that is a mix of the two transactions.[3][4]
- **Read committed**
  - In this isolation level, a lock-based concurrency control DBMS implementation keeps write locks (acquired on selected data) until the end of the transaction, but read locks are released as soon as the SELECT operation is performed (so the ***non-repeatable reads phenomenon*** can occur in this isolation level). As in the previous level, *range-locks* are not managed.
  - Putting it in simpler words, read committed is an isolation level that guarantees that any data read is committed at the moment it is read. It simply restricts the reader from seeing any intermediate, uncommitted, 'dirty' read. It makes no promise whatsoever that if the transaction re-issues the read, it will find the same data; data is free to change after it is read.
- **Read uncommitted**
  - This is the *lowest* isolation level. In this level, ***dirty reads*** are allowed, so one transaction may see *not-yet-committed* changes made by other transactions

In Databases, Cursors Define *Isolation*

- We have talked about ACID transactions

Isolation level	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	may occur	may occur	may occur
Read Committed	-	may occur	may occur
Repeatable Read	-	-	may occur
Serializable	-	-	-

```
InitialContext ctx = new InitialContext();
DataSource ds = (DataSource)
ctx.lookup("jdbc/MyBase");
Connection con = ds.getConnection();
DatabaseMetaData dbmd = con.getMetaData();
if (dbmd.supportsTransactionIsolationLevel(TRANSACTION_SERIALIZABLE)
{ Connection.setTransactionIsolation(TRANSACTION_SERIALIZABLE); }
```

- Isolation
  - Determines what happens when two or more threads are manipulating the data at the same time.
  - And is defined relative to where cursors are and what they have touched.
  - Because the cursor movement determines *what you are reading or have read.*
- *But, … Cursors are client conversation state and cannot be used in REST.*

CAP, BASE

CAP Theorem

- Consistency

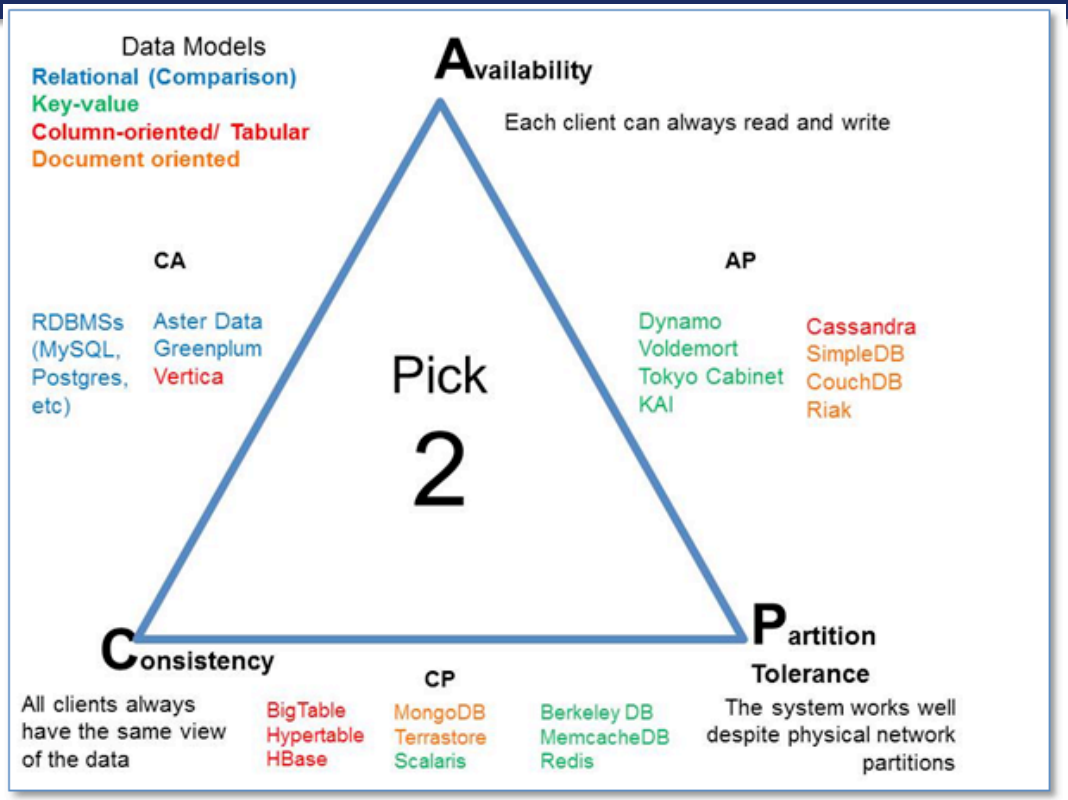
  Every read receives the most recent write or an error.

- Availability

  Every request receives a (non-error) response – without guarantee that it contains the most recent write.

- Partition Tolerance

  The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



Consistency Models

- **STRONG CONSISTENCY:** Strong consistency is a consistency model where all subsequent accesses to a distributed system will always return the updated value after the update.
- **WEAK CONSISTENCY:** It is a consistency model used in distributed computing where subsequent accesses might not always be returning the updated value. There might be inconsistent responses.
- **EVENTUAL CONSISTENCY:** Eventual consistency is a special type of weak consistency method which informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

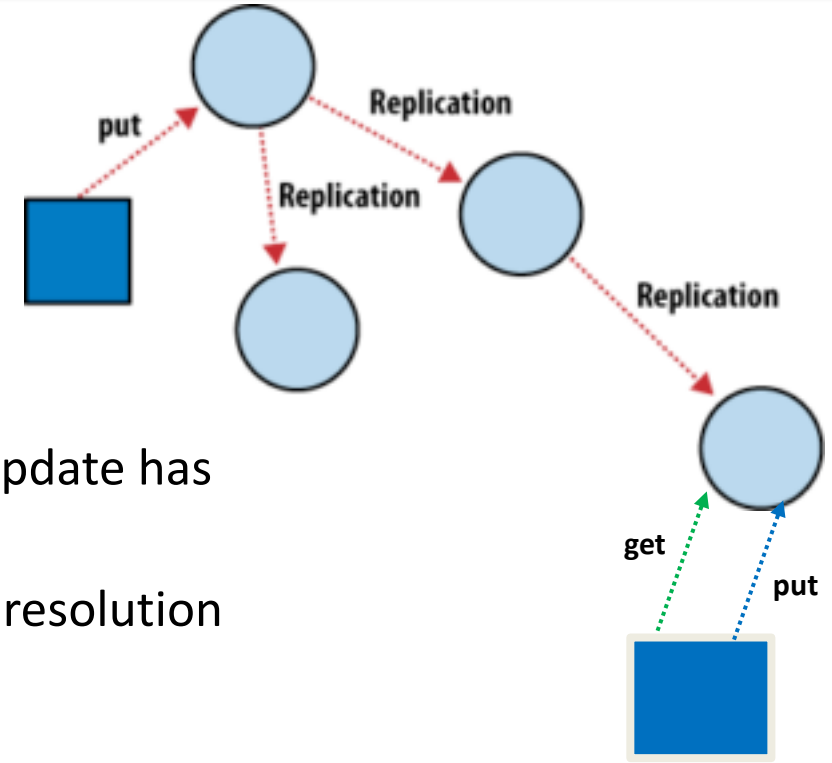
BASE

Eventually-consistent services are often classified as providing BASE (Basically Available, Soft state, Eventual consistency) semantics, in contrast to traditional ACID (Atomicity, Consistency, Isolation, Durability) guarantees. Rough definitions of each term in BASE:

- **Basically Available**: basic reading and writing operations are available as much as possible (using all nodes of a database cluster), but without any kind of consistency guarantees (the write may not persist after conflicts are reconciled, the read may not get the latest write)
- **Soft state**: without consistency guarantees, after some amount of time, we only have some probability of knowing the state, since it may not yet have converged
- **Eventually consistent**: If the system is functioning and we wait long enough after any given set of inputs, we will eventually be able to know what the state of the database is, and so any further reads will be consistent with our expectations

Eventual Consistency

- Availability and scalability via
  - Multiple, replicated data stores.
  - Read goes to “any” replica.
  - PUT/POST/DELETE
    - Goes to any replica
    - Change propagate asynchronously
- GET may not see the latest value if the update has not propagated to the replica.
- There are several algorithms for conflict resolution
  - Detect and handle in application.
  - Clock/change vectors/version numbers
  - … …



ACID – BASE (Simplistic Comparison)

ACID (relational)	BASE (NoSQL)
Strong consistency	Weak consistency
Isolation	Last write wins (Or other strategy)
Transaction	Program managed
Robust database	Simple database
Simple code (SQL)	Complex code
Available and consistent	Available and partition-tolerant
Scale-up (limited)	Scale-out (unlimited)
Shared (disk, mem, proc etc.)	Nothing shared (parallellizable)

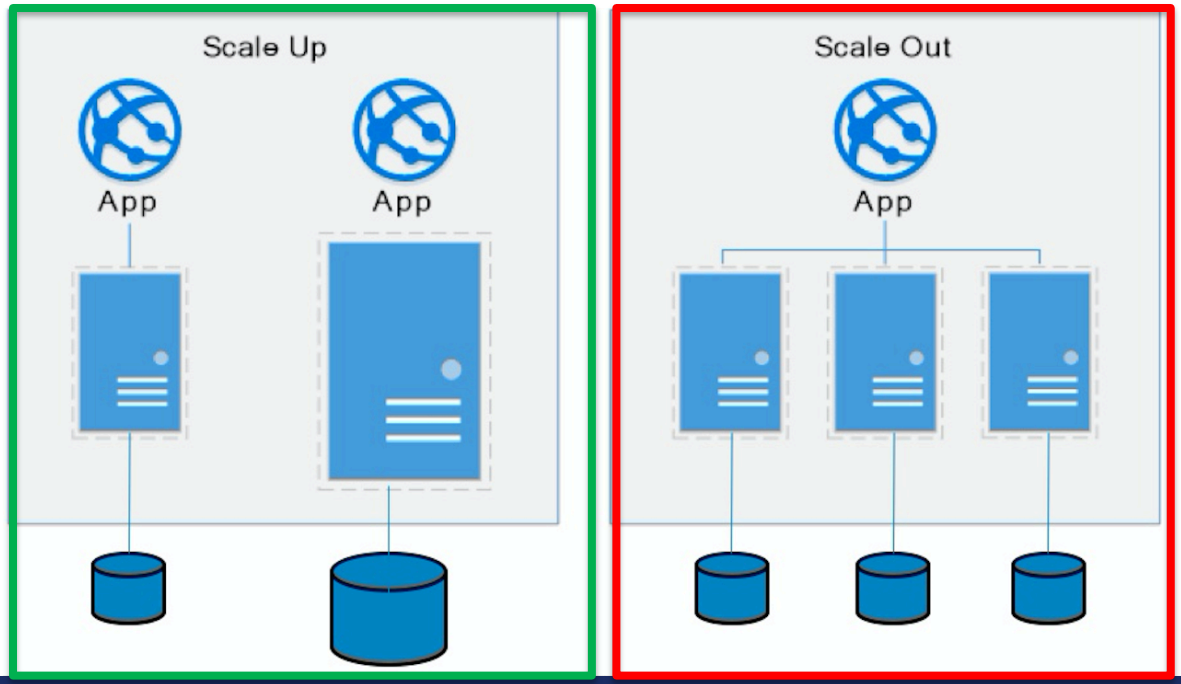
Scalability, Availability

Approaches to Scalability

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

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

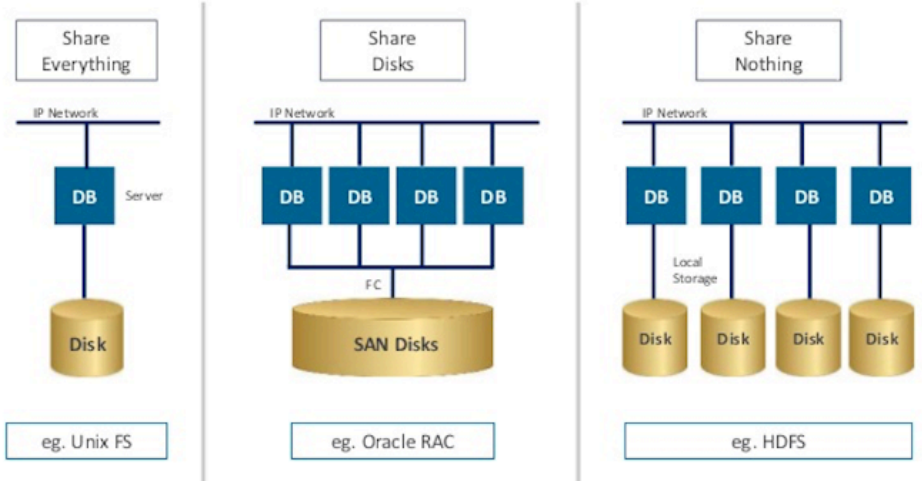
Add another system.



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

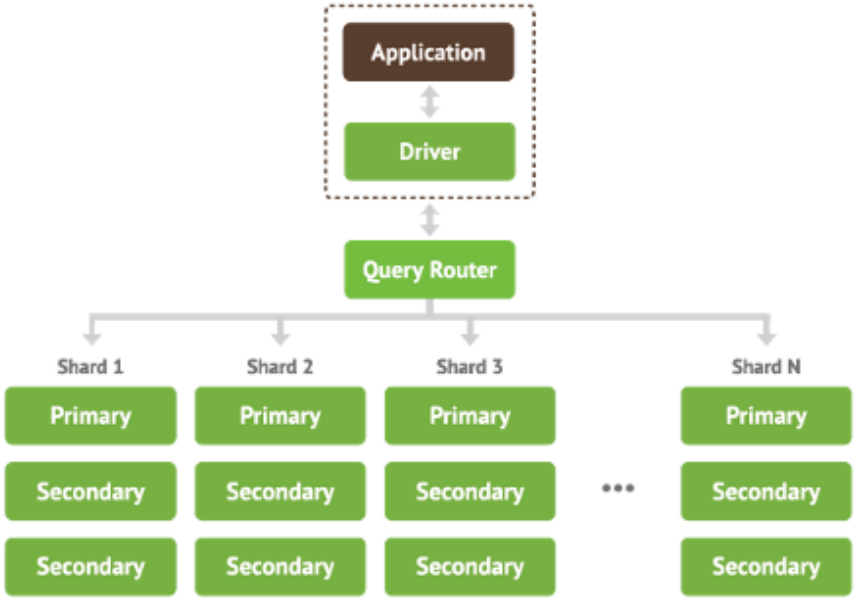
Disk Architecture for Scale-Out

- Share disks:
  - Is basically scale-up for data/disks. You can use NAS, SAN and RAID.
  - Isolation/Integrity requires distributed locking to control access from multiple database servers.
- Share nothing:
  - Is basically scale-out for disks.
  - Data is partitioned into *shards* based on a function f() applied to a key.
  - Can improve availability, at the code consistency, with data replication.
  - There is a router that sends requests to the proper shard based on the function.

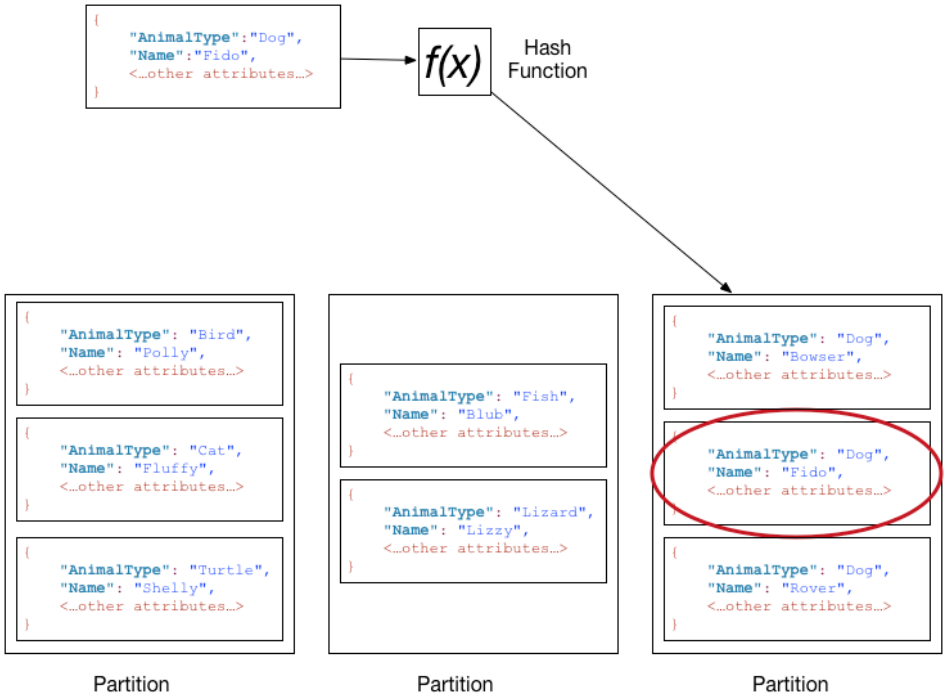


Shared Nothing, Scale-Out

MongoDB Sharding



DynamoDB Partitioning



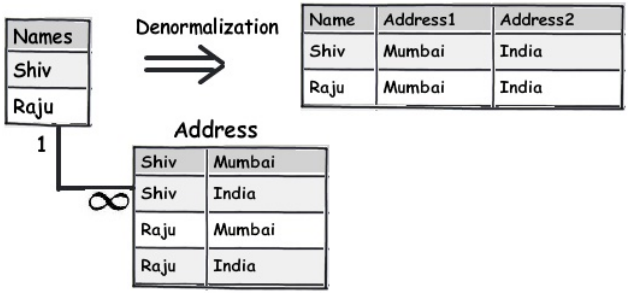
Decision Support

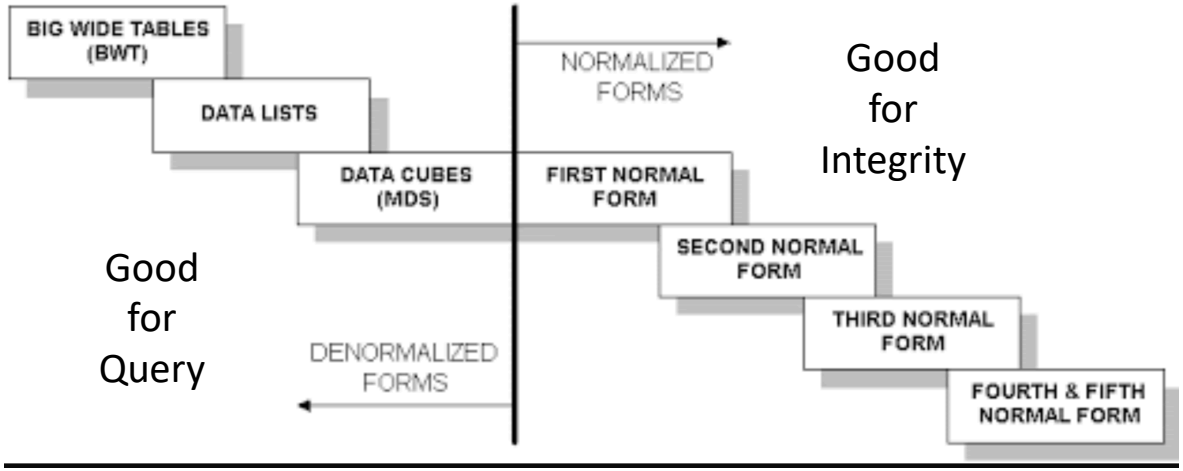
Concepts

Schema Design Tradeoffs

Data Warehouse, Data Lake, ETL

Wide Flat Tables





- Improve query performance by precomputing and saving:
  - JOINs
  - Aggregation
  - Derived/computed columns
- One of the primary strength of the relational model is maintaining "integrity" when applications create, update and delete data. This relies on:
  - The core capabilities of the relational model, e.g. constraints.
  - A well-design database (We will cover a formal definition – "normalization" in more detail later.
- Data models that are well designed for integrity are very bad for read only analysis queries.
  We will build and analyze wide flat tables as part of the analysis tasks in HW3, HW4 as projects.

Chapter 20: Data Analysis

- Decision Support Systems
- Data Warehousing
- Data Mining
- Classification
- ~~Association Rules~~
- ~~Clustering~~

Decision Support Systems

- **Decision-support systems** are used to make business decisions, often based on data collected by on-line transaction-processing systems.
- Examples of business decisions:
  - What items to stock?
  - What insurance premium to change?
  - To whom to send advertisements?
- Examples of data used for making decisions
  - Retail sales transaction details
  - Customer profiles (income, age, gender, etc.)

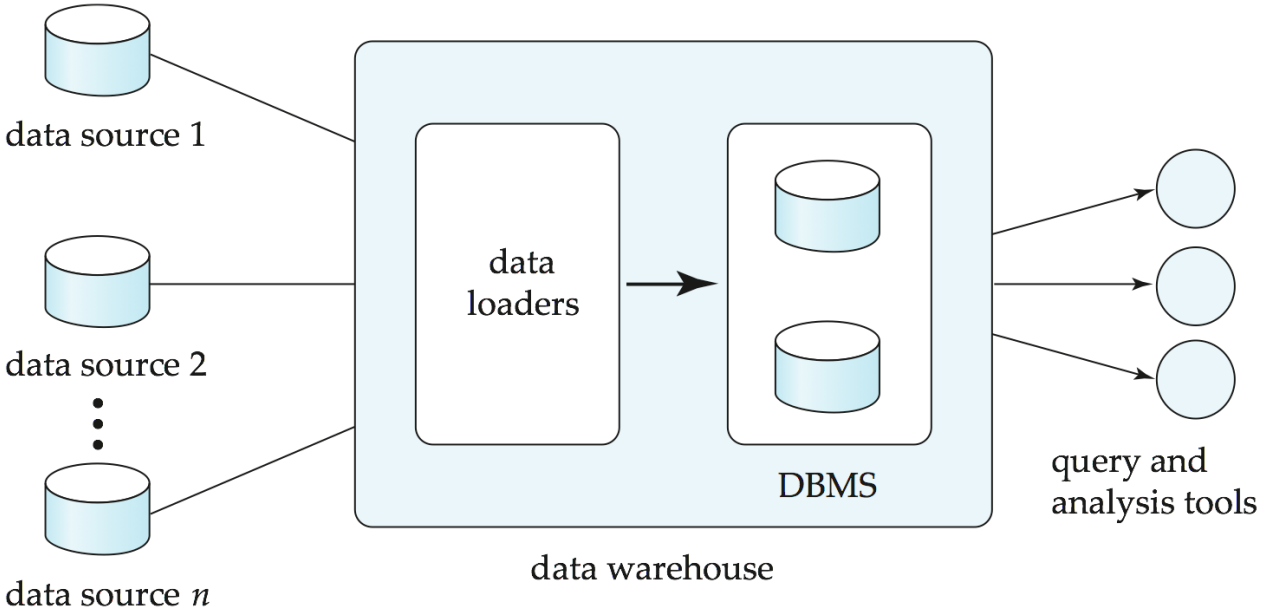
Decision-Support Systems: Overview

- **Data analysis** tasks are simplified by specialized tools and SQL extensions
  - Example tasks
    - For each product category and each region, what were the total sales in the last quarter and how do they compare with the same quarter last year
    - As above, for each product category and each customer category
- **Statistical analysis** packages (e.g., : S++) can be interfaced with databases
  - Statistical analysis is a large field, but not covered here
- **Data mining** seeks to discover knowledge automatically in the form of statistical rules and patterns from large databases.
- A **data warehouse** archives information gathered from multiple sources, and stores it under a unified schema, at a single site.
  - Important for large businesses that generate data from multiple divisions, possibly at multiple sites
  - Data may also be purchased externally

Data Warehousing

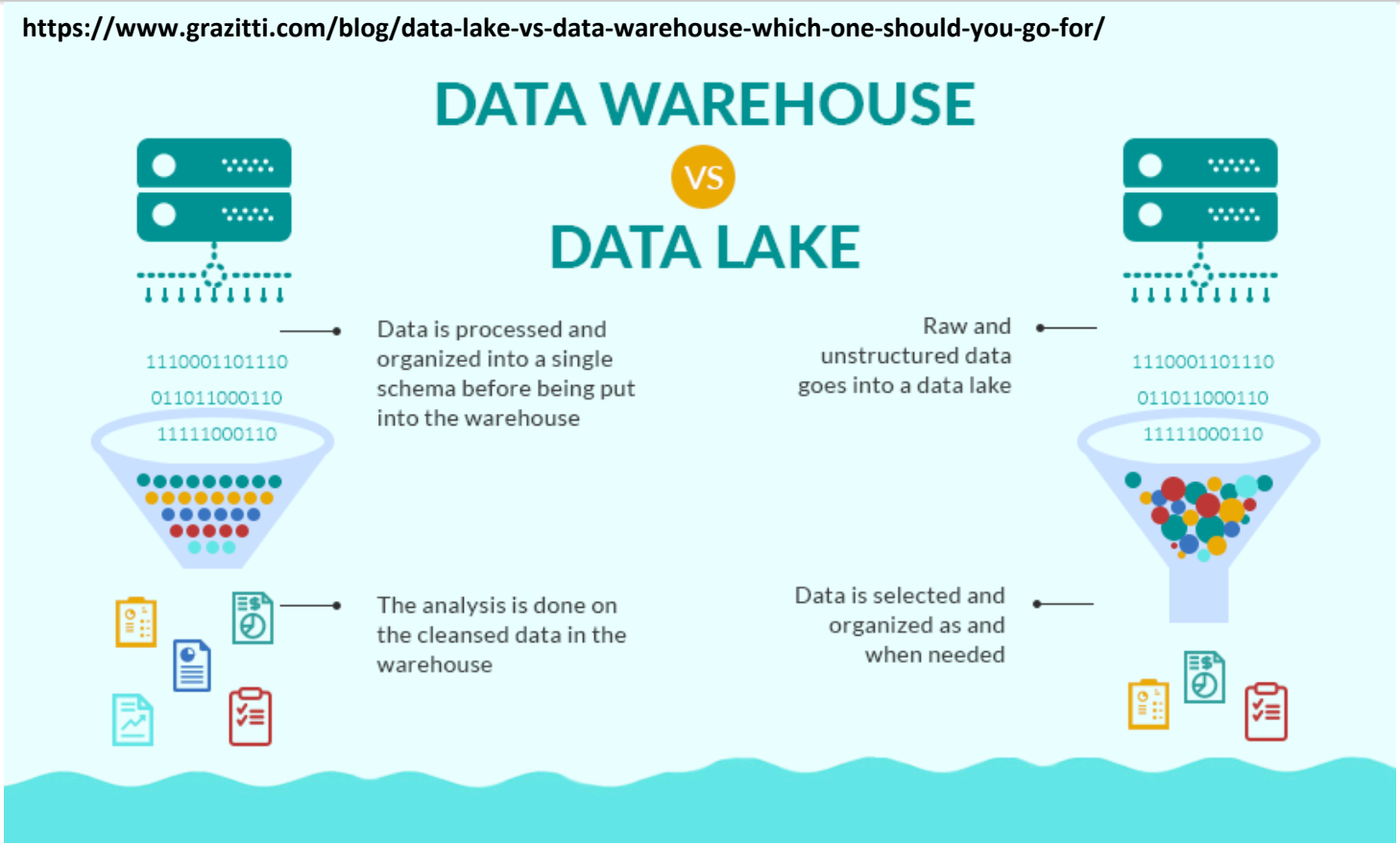
- Data sources often store only current data, not historical data
- Corporate decision making requires a unified view of all organizational data, including historical data
- A **data warehouse** is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site
  - Greatly simplifies querying, permits study of historical trends
  - Shifts decision support query load away from transaction processing systems

Data Warehousing



Data Warehouse vs Data Lake

https://www.grazitti.com/blog/data-lake-vs-data-warehouse-which-one-should-you-go-for/



Simplistic: Data Warehouse vs Data Lake

https://panoply.io/data-warehouse-guide/data-warehouse-vs-data-lake/

DATA WAREHOUSE	vs.	DATA LAKE
structured, processed	DATA	structured / semi-structured / unstructured, raw
schema-on-write	PROCESSING	schema-on-read
expensive for large data volumes	STORAGE	designed for low-cost storage
less agile, fixed configuration	AGILITY	highly agile, configure and reconfigure as needed
mature	SECURITY	maturing
business professionals	USERS	data scientists et. al.

Design Issues

- *When and how to gather data*
  - **Source driven architecture**: data sources transmit new information to warehouse, either continuously or periodically (e.g., at night)
  - **Destination driven architecture**: warehouse periodically requests new information from data sources
  - Keeping warehouse exactly synchronized with data sources (e.g., using two-phase commit) is too expensive
    - Usually OK to have slightly out-of-date data at warehouse
    - Data/updates are periodically downloaded form online transaction processing (OLTP) systems.
- *What schema to use*
  - Schema integration

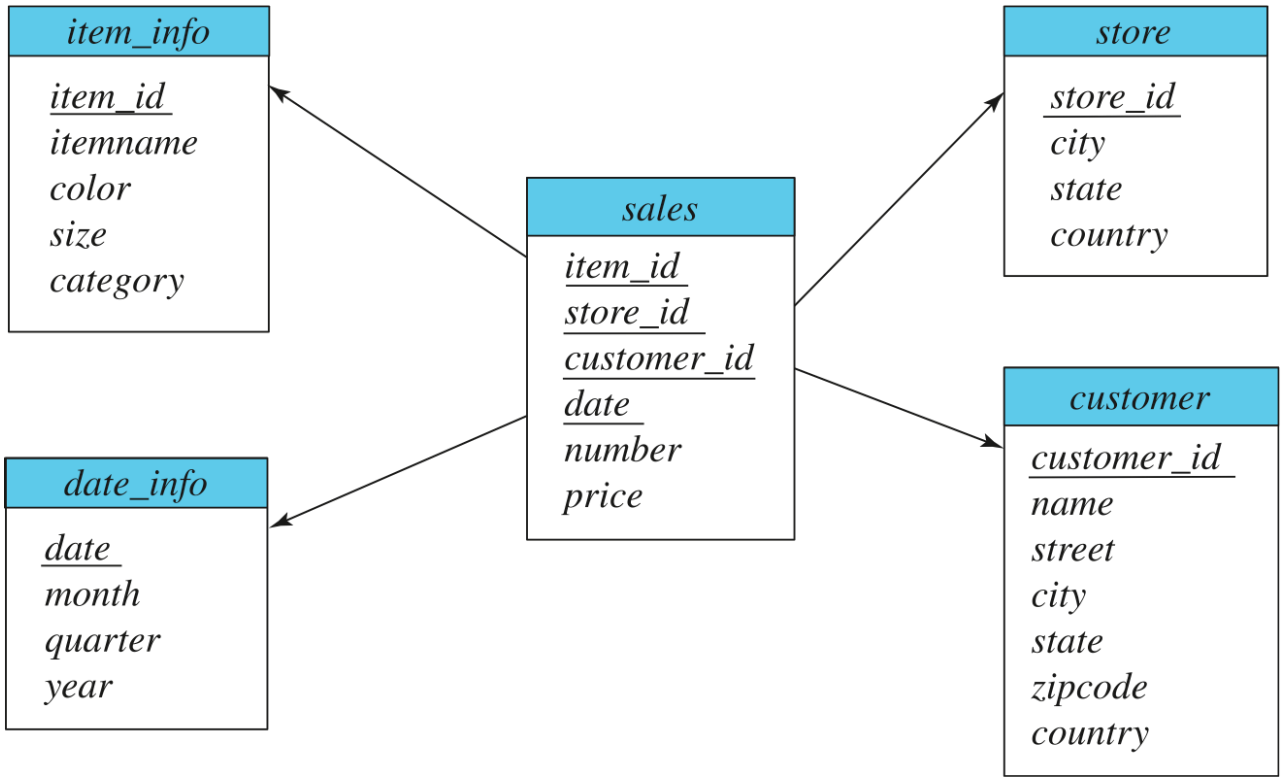
More Warehouse Design Issues

- *Data cleansing*
  - E.g., correct mistakes in addresses (misspellings, zip code errors)
  - **Merge** address lists from different sources and **purge** duplicates
- *How to propagate updates*
  - Warehouse schema may be a (materialized) view of schema from data sources
- *What data to summarize*
  - Raw data may be too large to store on-line
  - Aggregate values (totals/subtotals) often suffice
  - Queries on raw data can often be transformed by query optimizer to use aggregate values

Warehouse Schemas

- Dimension values are usually encoded using small integers and mapped to full values via dimension tables
- Resultant schema is called a **star schema**
  - More complicated schema structures
    - **Snowflake schema**: multiple levels of dimension tables
    - **Constellation**: multiple fact tables

Data Warehouse Schema

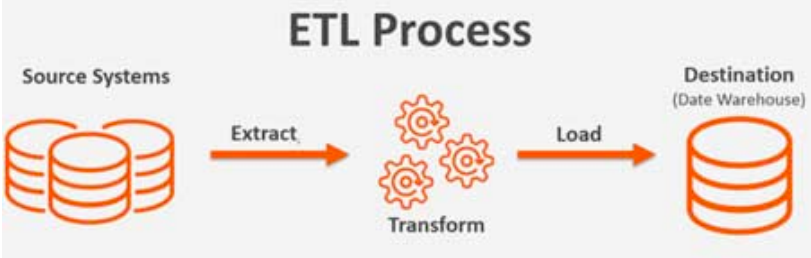


Overview (Cont.)

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

ETL Concepts

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



Extract

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

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

Transform

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

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

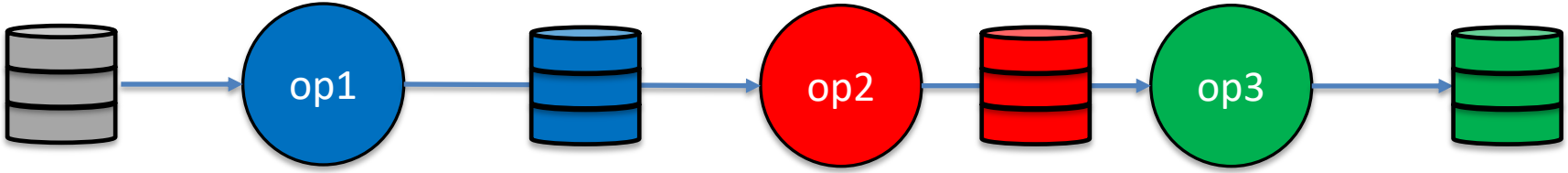
Load

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

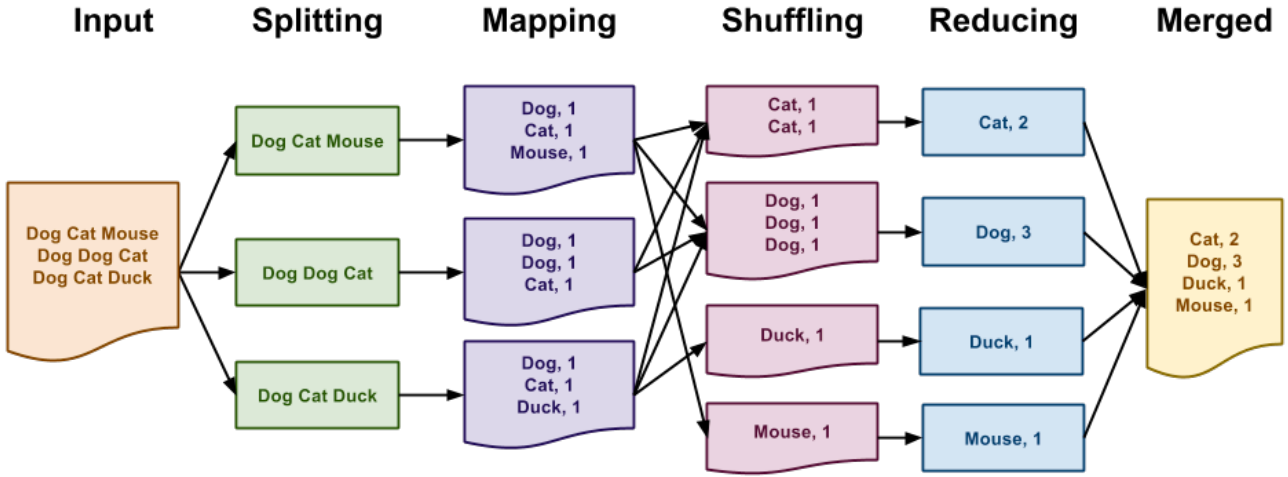
MapReduce, Spark (and ETL)

MapReduce

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



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



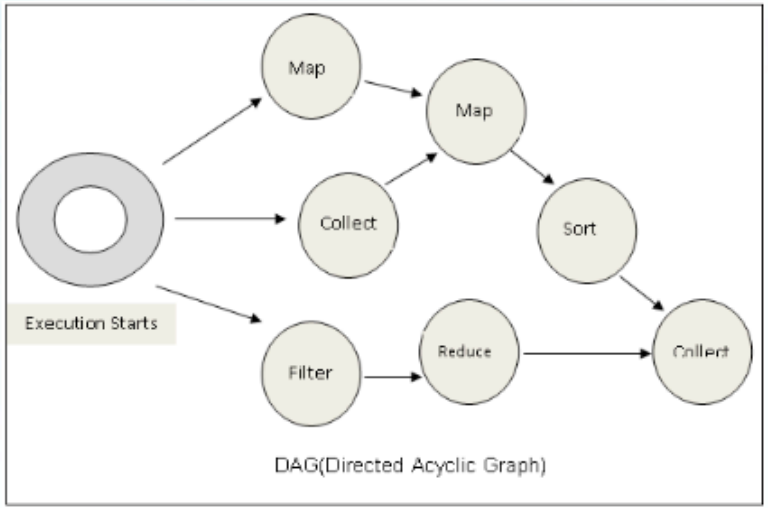
What if we want more complex "operators?"

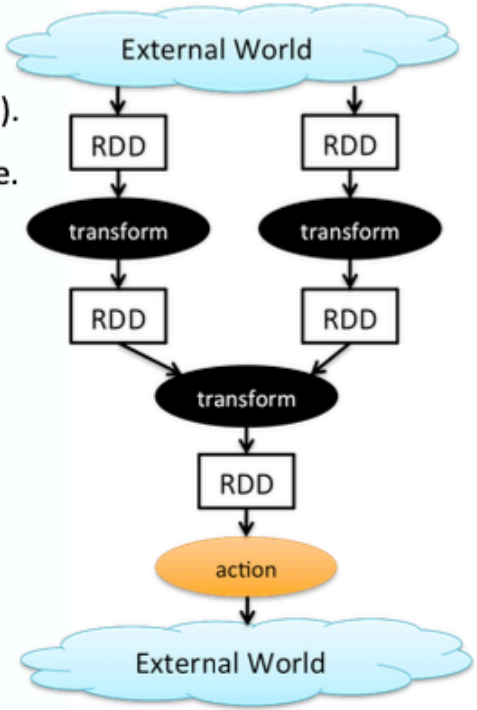
Algebraic Operations

- Current generation execution engines
  - natively support algebraic operations such as joins, aggregation, etc. natively.
  - Allow users to create their own algebraic operators
  - Support trees of algebraic operators that can be executed on multiple nodes in parallel
- E.g. Apache Tez, Spark
  - Tex provides low level API; Hive on Tez compiles SQL to Tez
  - Spark provides more user-friendly API

- In the relational model,
  - The are relations.
  - A fixed set of operators that produce relations from relations (closed).
  - Are declarative languages and compute the execution graph/plan.
- The Spark/... ... engines:
  - Enable programmers to develop and add new operators.
  - Operators convert reliable distributed datasets/data frames to new RDDs/data frames.
  - Data engineer explicitly defines the execution graph (data flow).

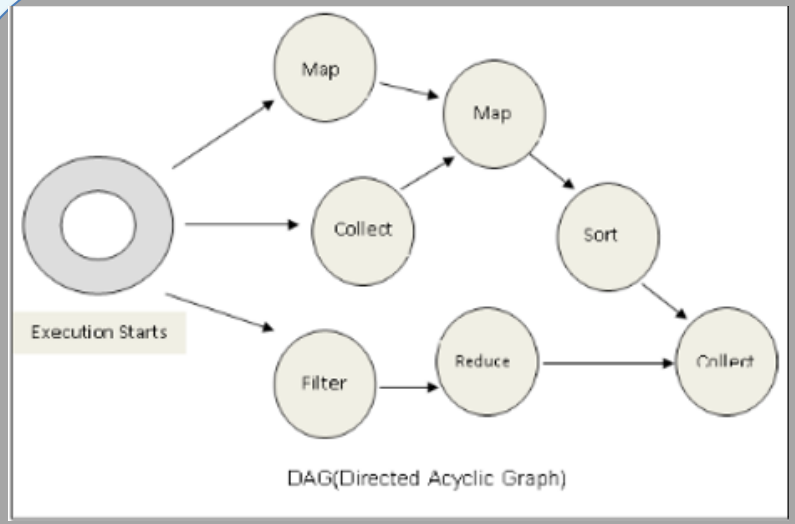
- All jobs in spark comprise a series of operators and run on a set of data.
- All the operators in a job are used to construct a DAG (Directed Acyclic Graph).
- The DAG is optimized by rearranging and combining operators where possible.





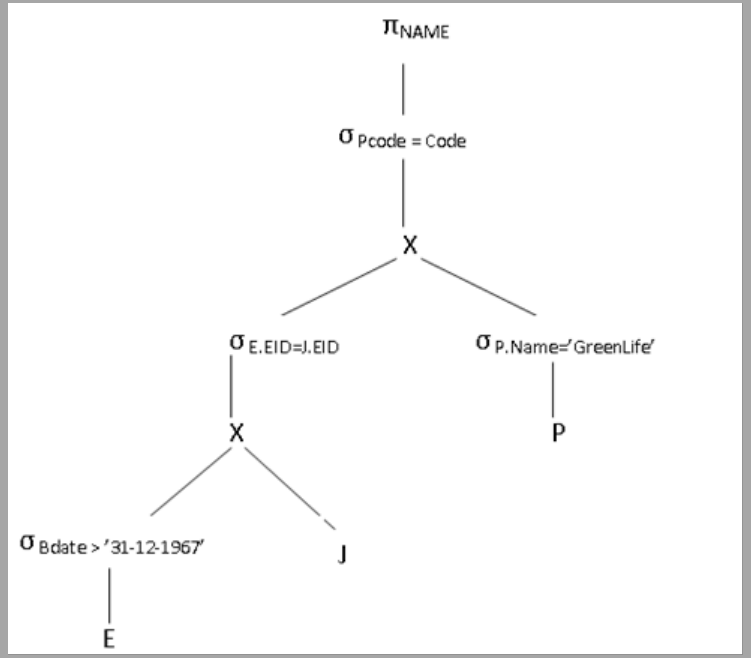
www.edureka.co/apache-spark-scala-training

Spark vs SQL



Conceptually similar to query evaluation graph, but ...

- You explicitly define the graph.
- You can develop your own operators.
- You can define levels of parallelism that the infrastructure manages.



SQL is declarative, and the query engine transparently produces the execution plan.

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

Algebraic Operations in Spark

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

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

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

DataFrame

edureka!

- Inspired by DataFrames in R and Python (Pandas).
- DataFrames API is designed to make big data processing on tabular data easier.
- DataFrame is a distributed collection of data organized into named columns.
- Provides operations to filter, group, or compute aggregates, and can be used with Spark SQL.
- Can be constructed from structured data files, existing RDDs, tables in Hive, or external databases.

DataFrame features

edureka!

- Ability to scale from KBs to PBs
- Support for a wide array of data formats and storage systems
- State-of-the-art optimization and code generation through the spark SQL catalyst optimizer
- Seamless integration with all big data tooling and infrastructure via spark
- APIs for Python, Java, Scala, and R

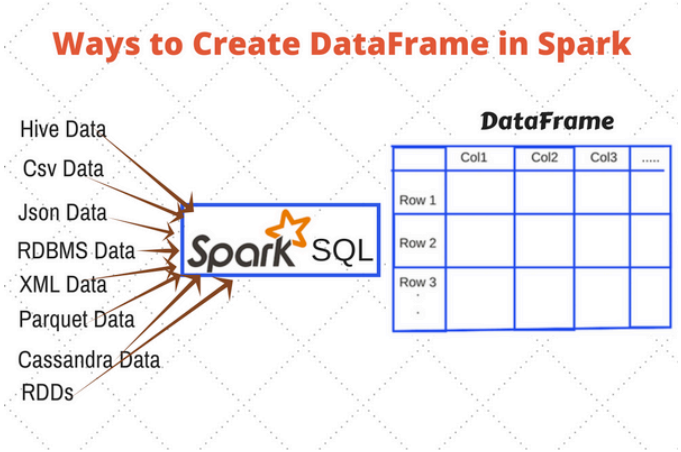
1. DataFrame in PySpark: Overview

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

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

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

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



Conceptual Example

- Remember that really unpleasant set of queries and stored procedure we had to write for the midterm? You had to convert from … …

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

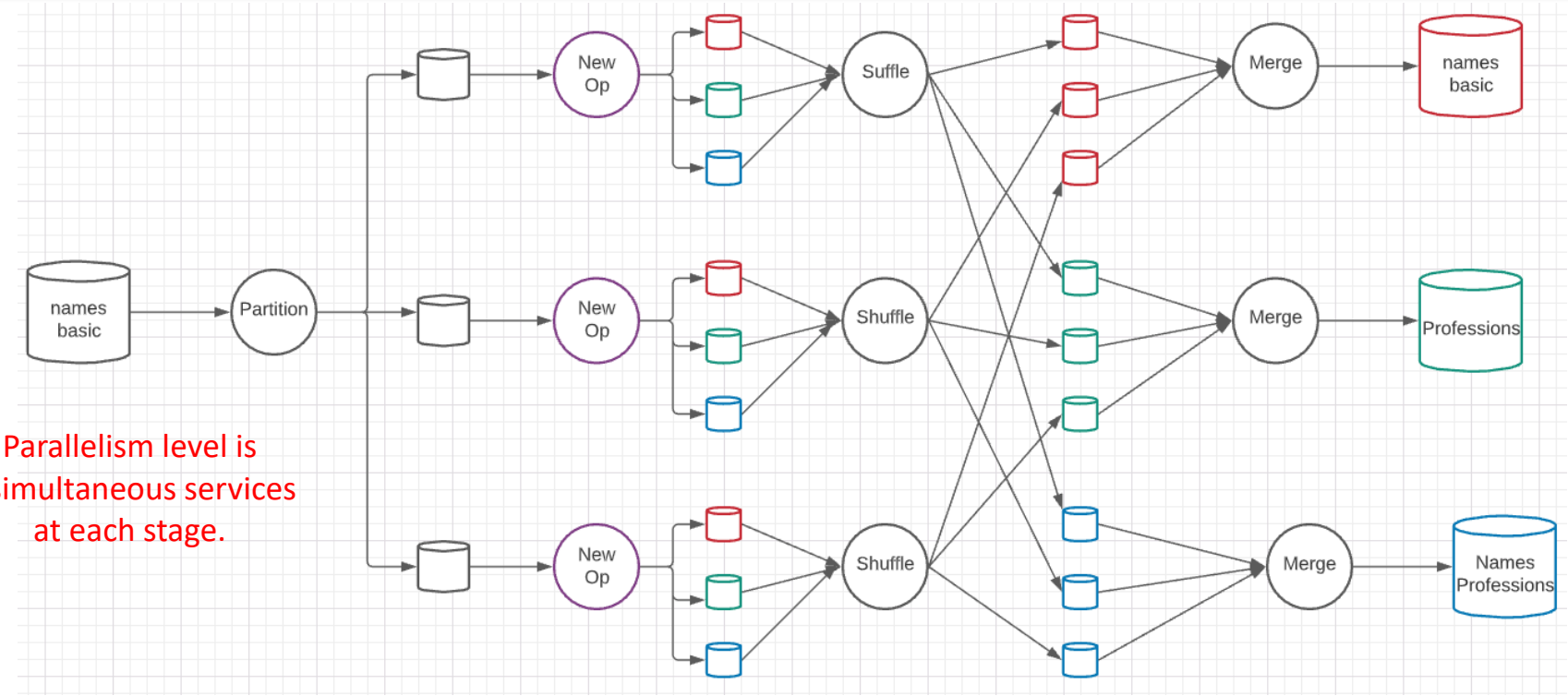
- To

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

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

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

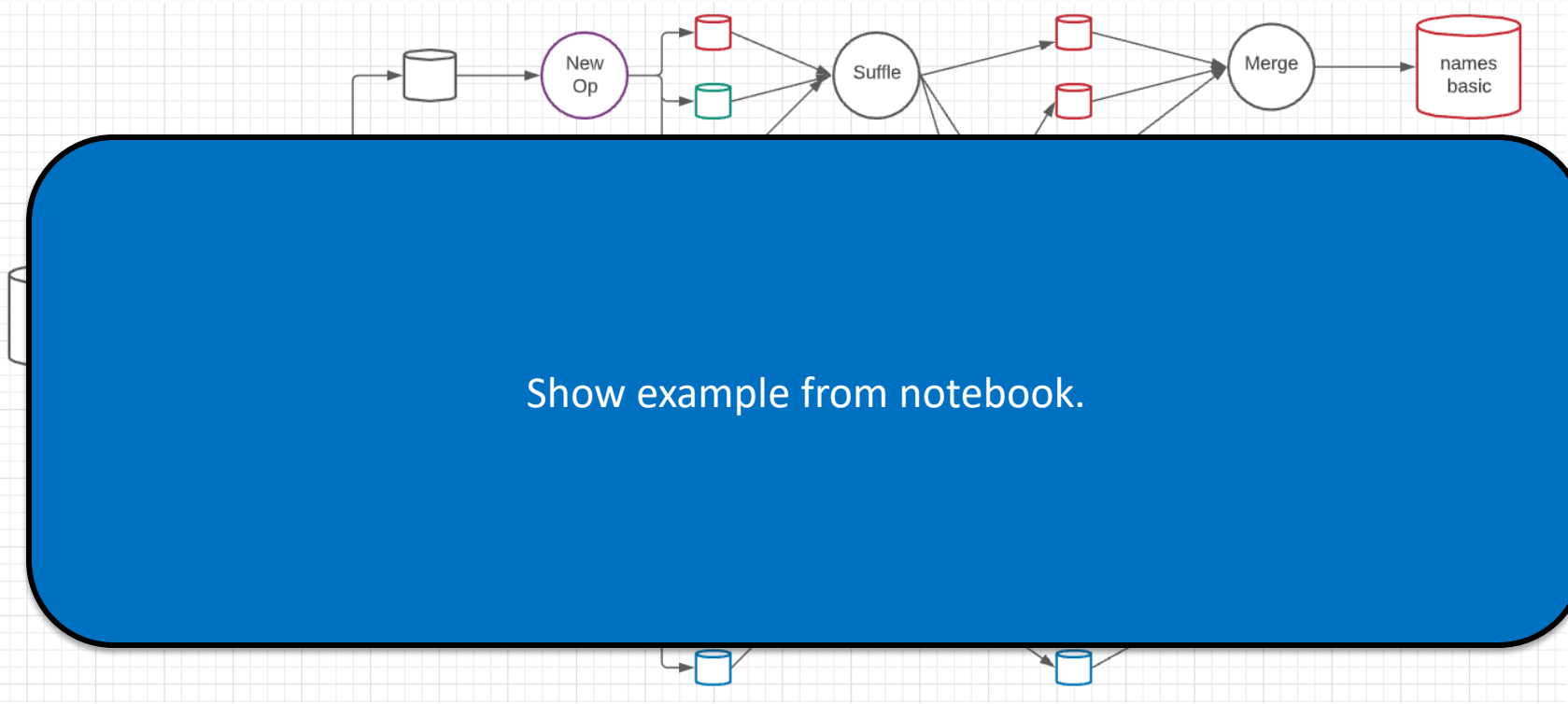
A New Algebraic Operator



Parallelism level is 3 simultaneous services at each stage.

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

A New Algebraic Operator



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

OLAP

Data Analysis and OLAP

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

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

Example sales relation

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

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

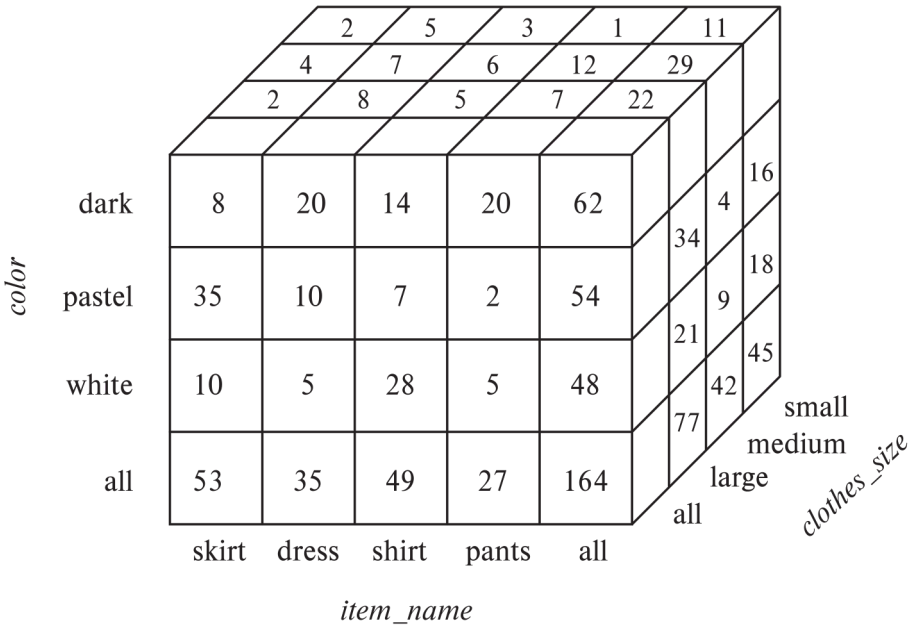
clothes_size **all**

item_name	*color* dark	pastel	white	total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pants	20	2	5	27
total	62	54	48	164

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

Data Cube

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



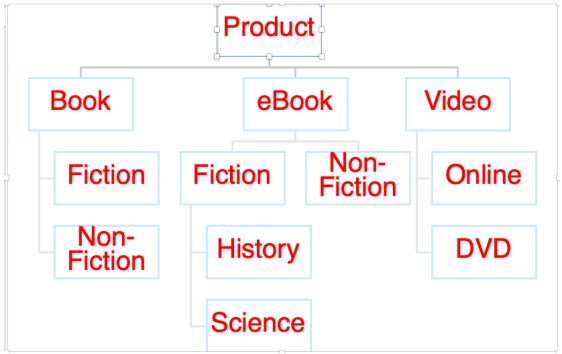
Online Analytical Processing Operations

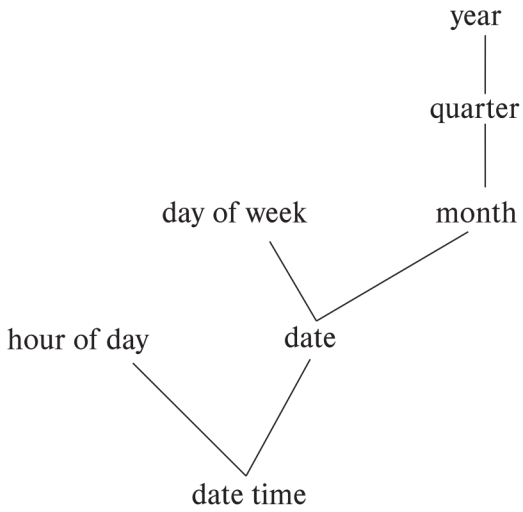
- **Pivoting:** changing the dimensions used in a cross-tab
  - E.g., moving colors to column names
- **Slicing:** creating a cross-tab for fixed values only
  - E.g., fixing color to white and size to small
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.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

Hierarchies on Dimensions

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

Another dimension could be ...
Product category.

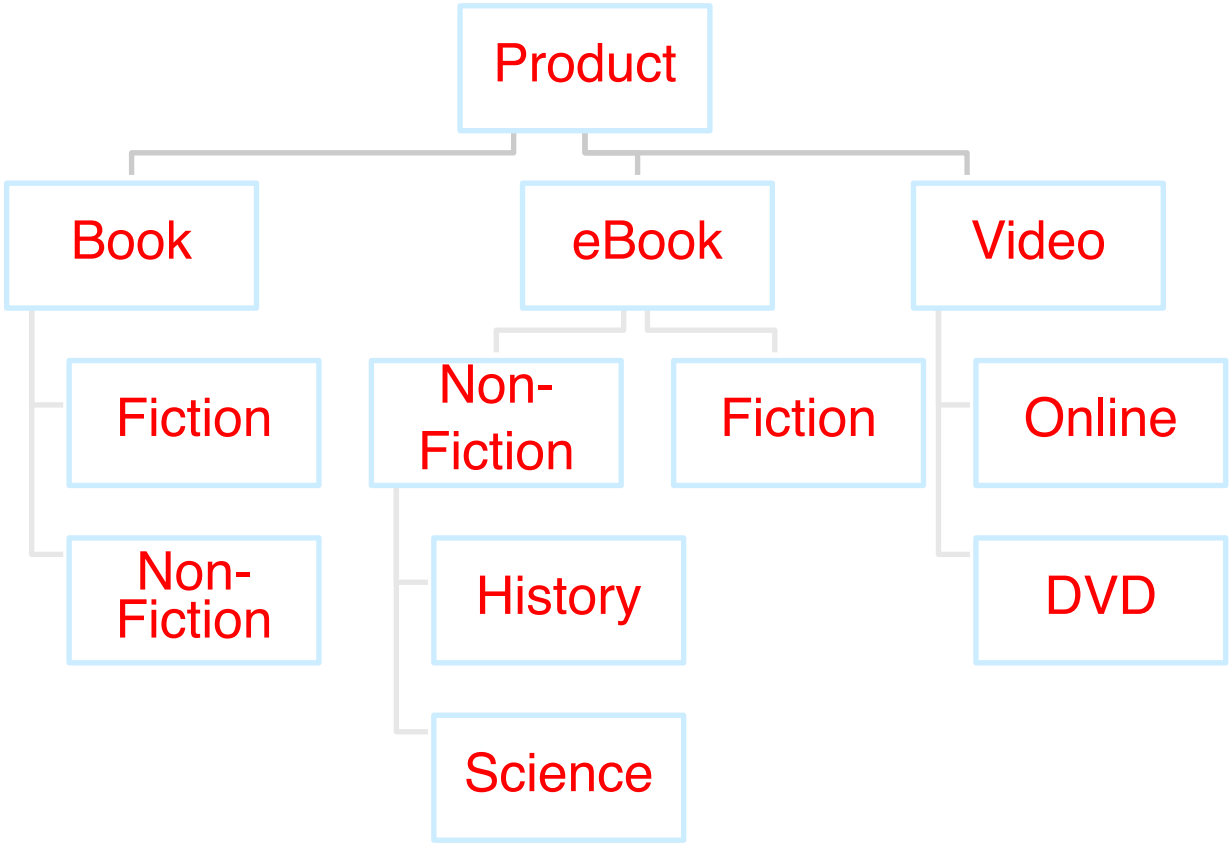




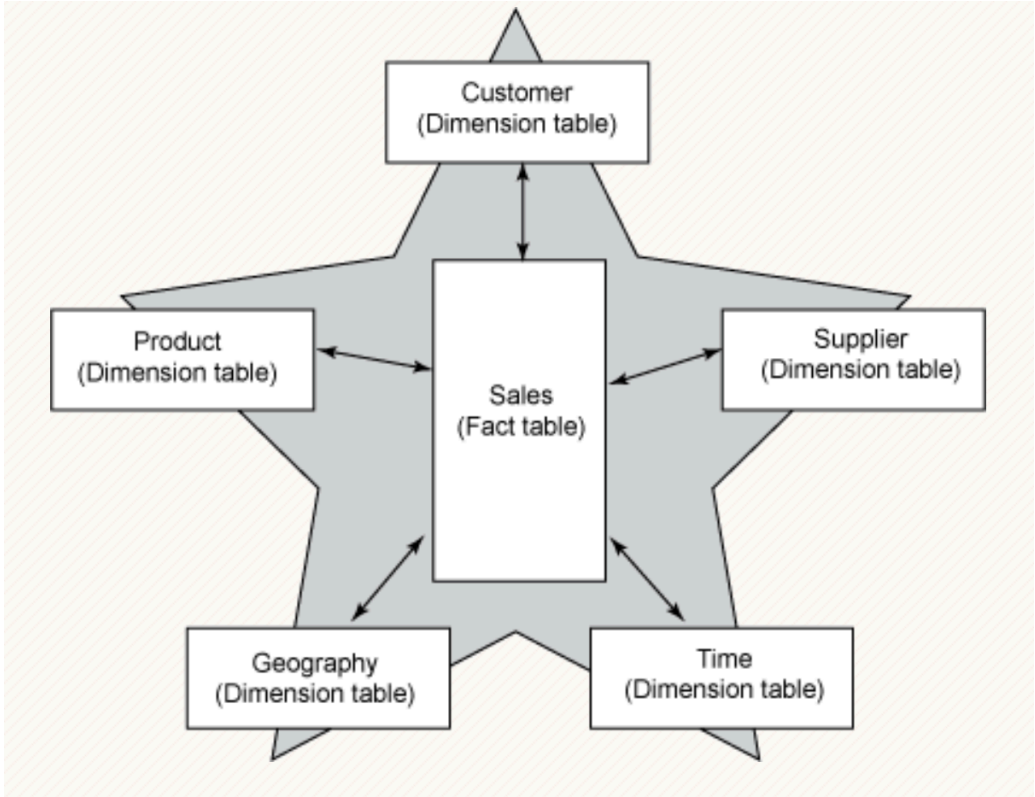
(a) time hierarchy

(b) location hierarchy

Another Dimension Example – Product Categories



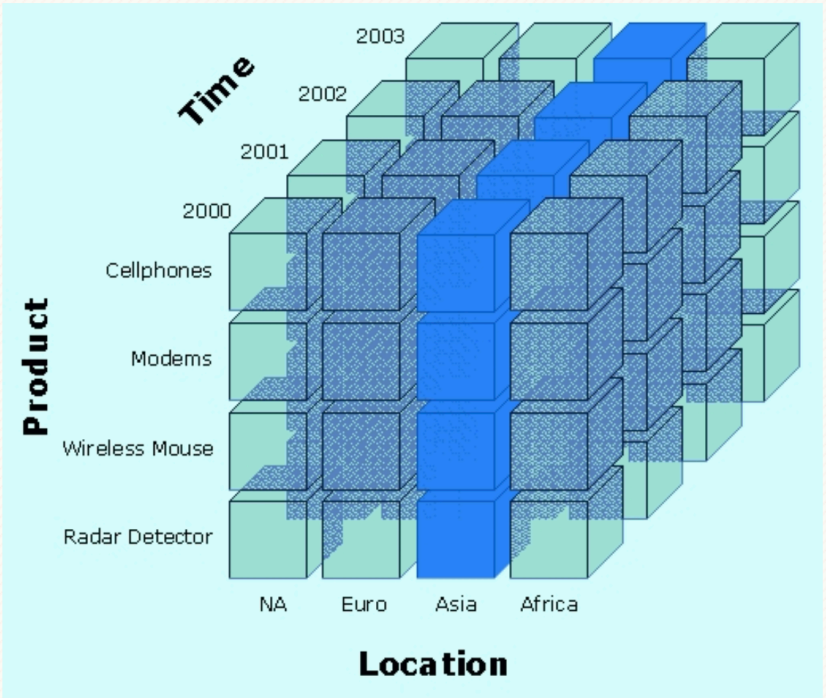
Facts and Dimensions



Slice

Slice:

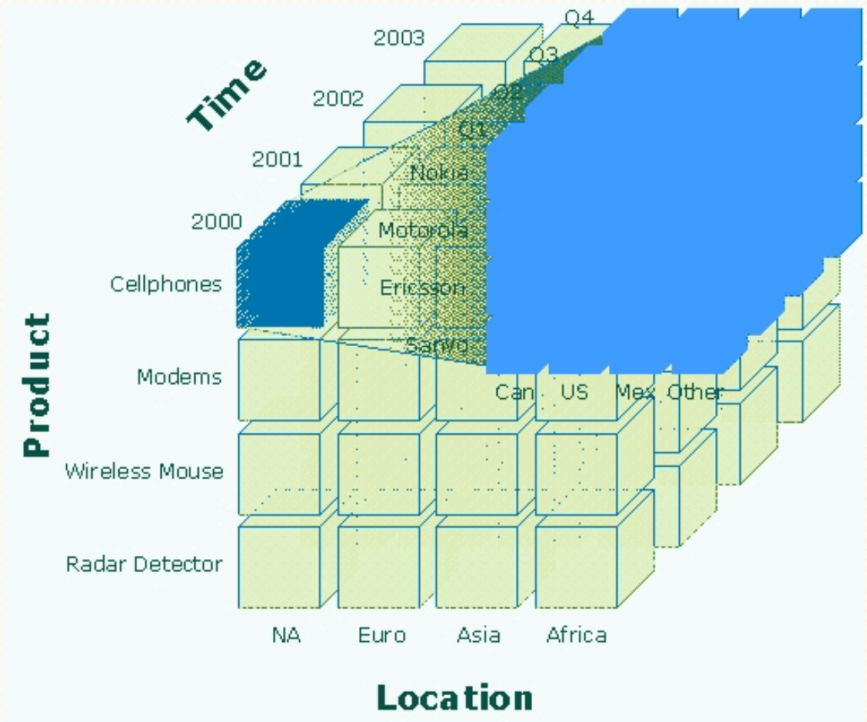
A slice is a subset of a multi-dimensional array corresponding to a single value for one or more members of the dimensions not in the subset.



Dice

Dice:

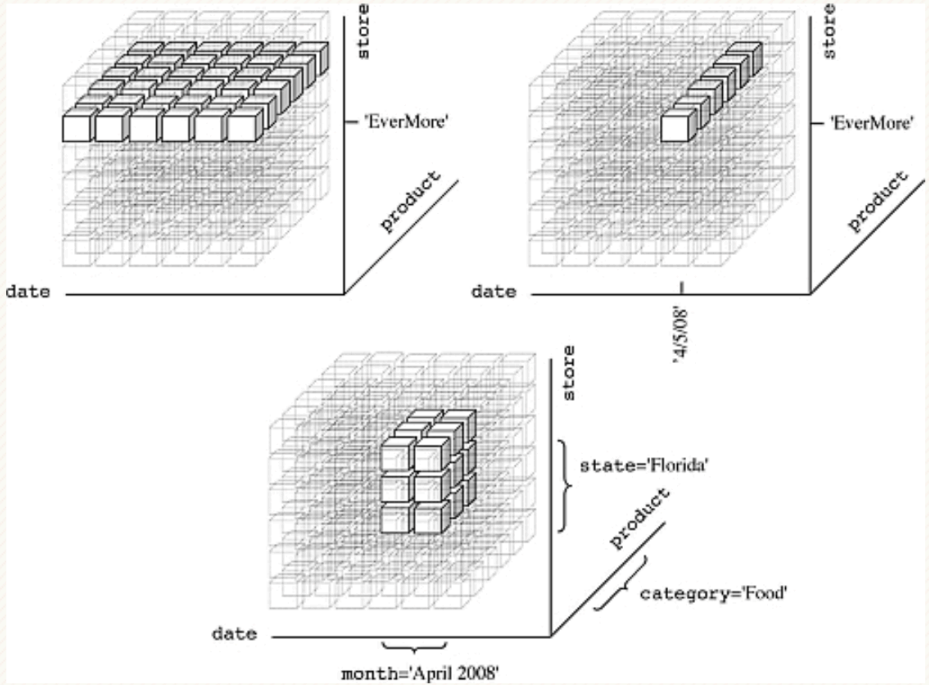
The dice operation is a slice on more than two dimensions of a data cube (or more than two consecutive slices).



Drilling

Drill Down/Up:

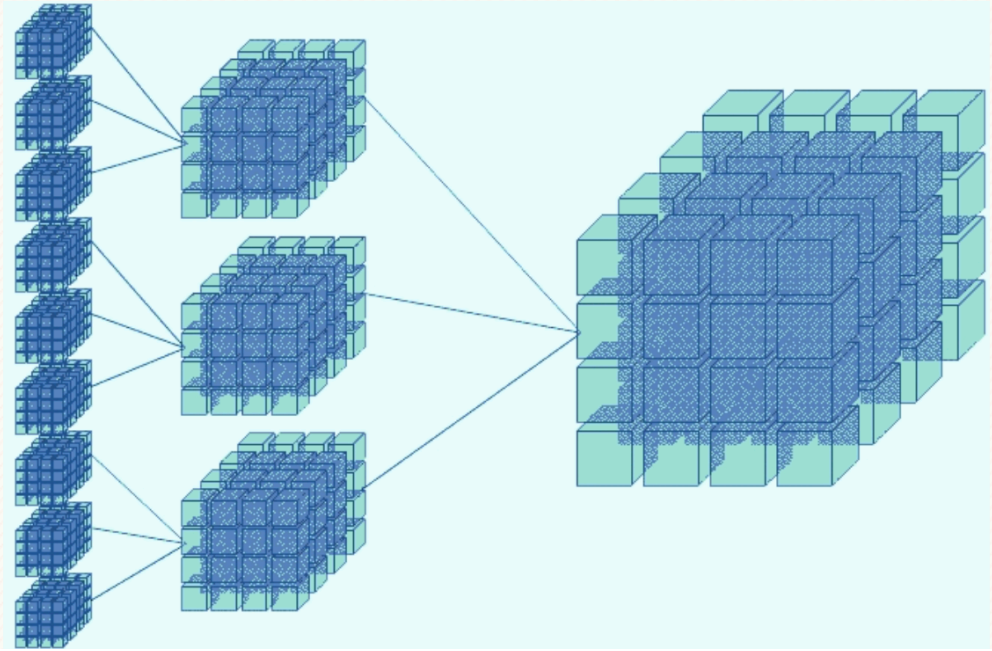
Drilling down or up is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down).



Roll-up

Roll-up:

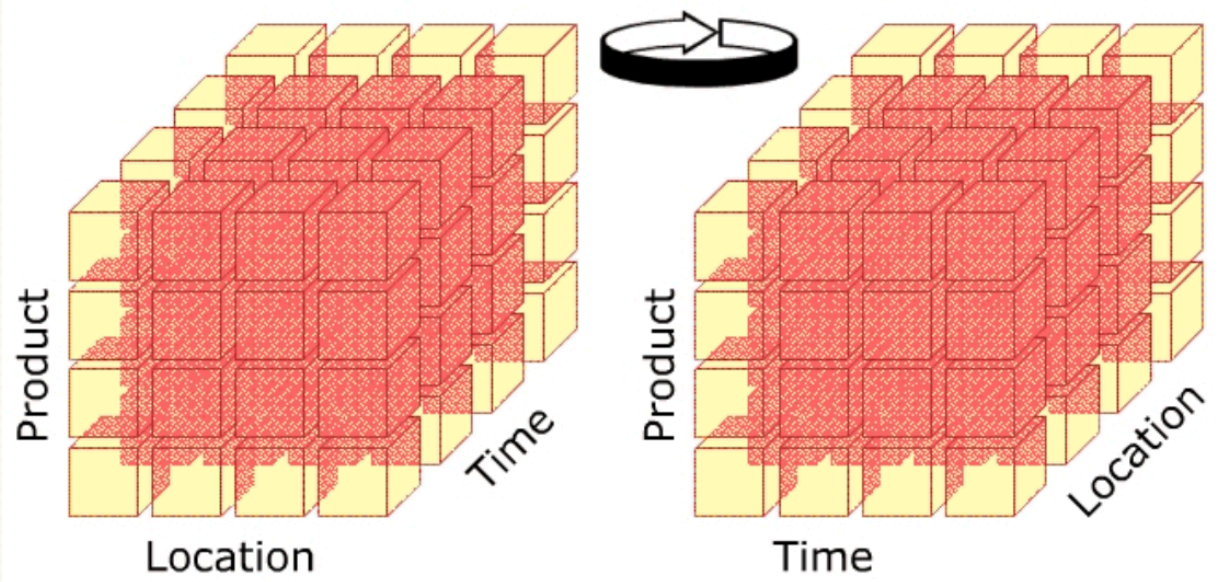
(Aggregate, Consolidate) A roll-up involves computing all of the data relationships for one or more dimensions. To do this, a computational relationship or formula might be defined.



Pivot

Pivot:

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



Cross Tabulation With Hierarchy

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

clothes_size: **all**

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

Relational Representation of Cross-tabs

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

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

Data Mining

Data Mining

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

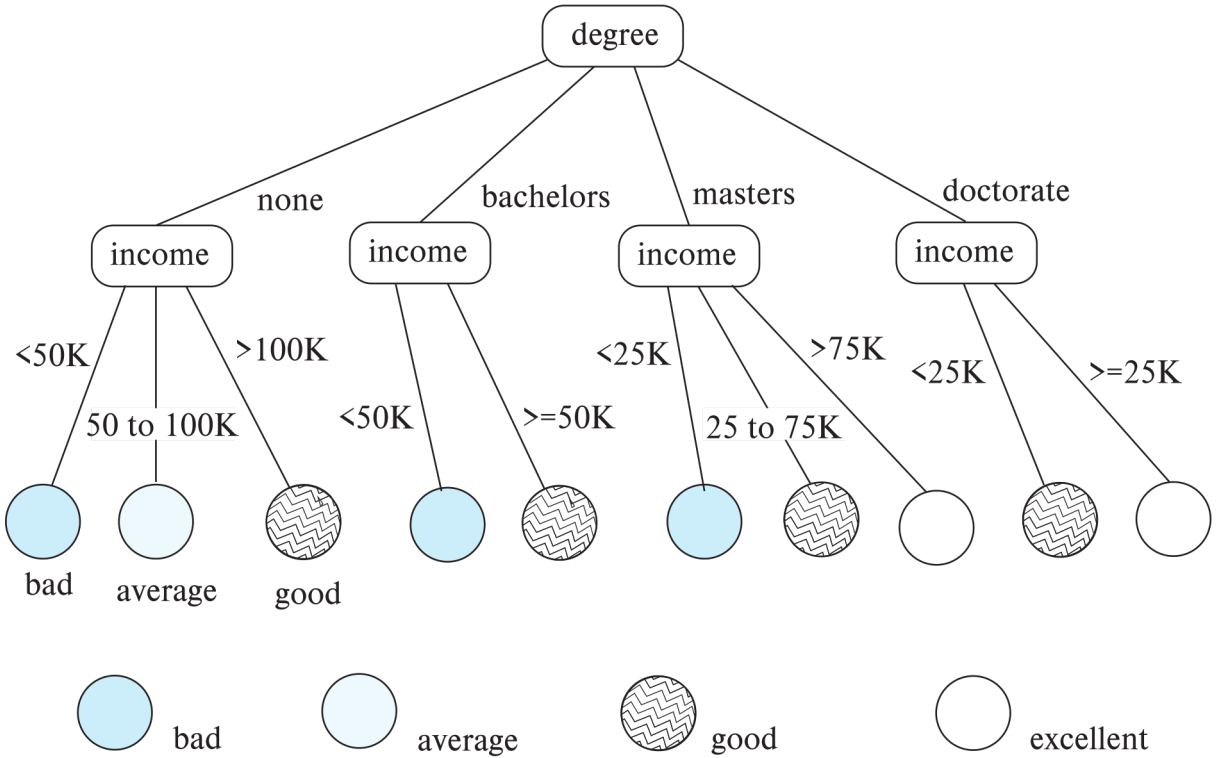
Types of Data Mining Tasks

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

Data Mining (Cont.)

- **Descriptive Patterns**
  - **Associations**
    - Find books that are often bought by “similar” customers. If a new such customer buys one such book, suggest the others too.
    - Associations may be used as a first step in detecting **causation**
      - E.g., association between exposure to chemical X and cancer,
  - **Clusters**
    - E.g., typhoid cases were clustered in an area surrounding a contaminated well
    - Detection of clusters remains important in detecting epidemics

Decision Tree Classifiers



Decision Trees

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

Bayesian Classifiers

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

$$p(c_j \mid d) = \frac{p(d \mid c_j)\, p(c_j)}{p(d)}$$

where

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

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

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

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

Naïve Bayesian Classifiers

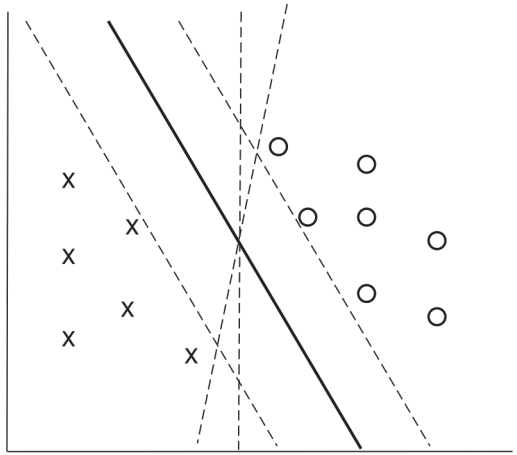
- Bayesian classifiers require
  - computation of $p(d \mid c_j)$
  - precomputation of $p(c_j)$
  - $p(d)$ can be ignored since it is the same for all classes
- To simplify the task, **naïve Bayesian classifiers** assume attributes have independent distributions, and thereby estimate

  $$p(d \mid c_j) = p(d_1 \mid c_j) * p(d_2 \mid c_j) * \ldots * (p(d_n \mid c_j)$$

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

Support Vector Machine Classifiers

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

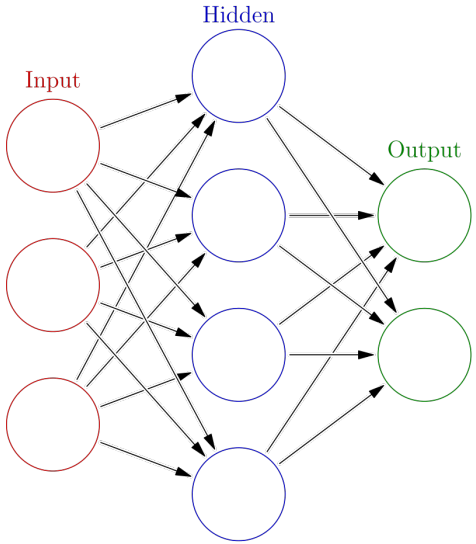


Support Vector Machine

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

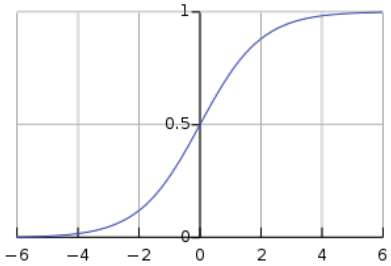
Neural Network Classifiers

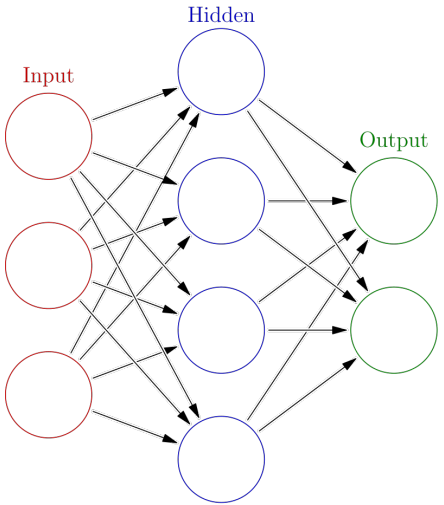
- Neural network has multiple layers
  - Each layer acts as input to next later
- First layer has input nodes, which are assigned values from input attributes
- Each node combines values of its inputs using some weight function to compute its value
  - Weights are associated with edges
- For classification,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



Neural Network Classifiers

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





Neural Networks (Cont.)

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

Regression

- Regression deals with the prediction of a value, rather than a class.
  - Given values for a set of variables, X_1, X_2, …, X_n, we wish to predict the value of a variable Y.
- One way is to infer coefficients a_0, a_1, a_1, …, a_n such that

$$Y = a_0 + a_1 * X_1 + a_2 * X_2 + \ldots + a_n * X_n$$

- Finding such a linear polynomial is called **linear regression**.
  - In general, the process of finding a curve that fits the data is also called **curve fitting**.
- The fit may only be approximate
  - because of noise in the data, or
  - because the relationship is not exactly a polynomial
- Regression aims to find coefficients that give the best possible fit.

Association Rules

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

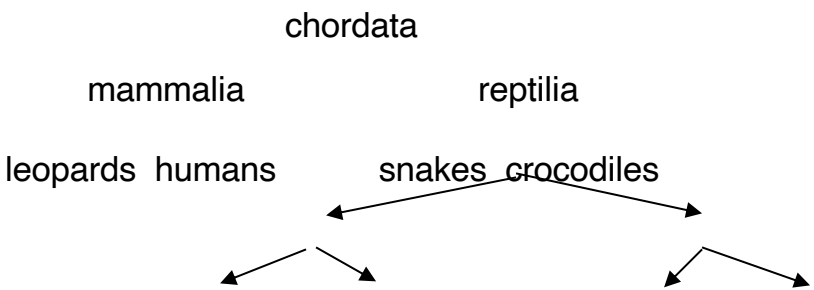
  bread $\Rightarrow$ *milk* *DB-Concepts, OS-Concepts* $\Rightarrow$ Networks
  - Left hand side: **antecedent**, right hand side: **consequent**
  - An association rule must have an associated **population**; the population consists of a set of **instances**
    - E.g. each transaction (sale) at a shop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

Association Rules (Cont.)

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

Clustering

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



Clustering and Collaborative Filtering

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

Other Types of Mining

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

Worked Example
Classic Models

Denormalization Example

Classic Models (Cars) Database

http://www.mysqltutorial.org/mysql-sample-database.aspx

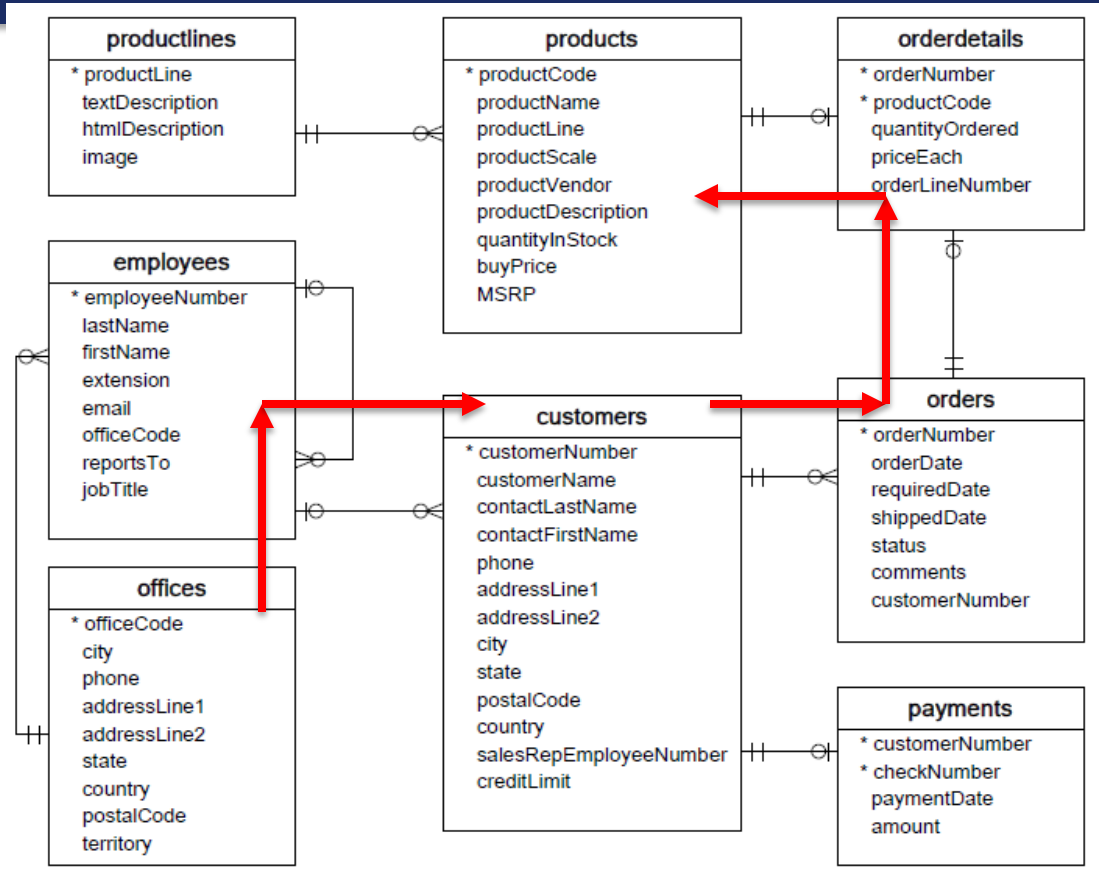
The classic models (cars) database we have seen has the following tables:

- **Customers**: stores customer's data.
- **Products**: stores a list of scale model cars.
- **ProductLines**: stores a list of product line categories.
- **Orders**: stores sales orders placed by customers.
- **OrderDetails**: stores sales order line items for each sales order.
- **Payments**: stores payments made by customers based on their accounts.
- **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
- **Offices**: stores sales office data.

Normalized Schema

- The normalization helps prevent update errors:
  - I cannot create an order without a valid customer no.
  - An order cannot have an invalid product ID.
  - etc.
- But some interesting data insight questions require complicated joins,

"Please show me the total number of instances of products from vendor XXX that employees in office YYY have sold."



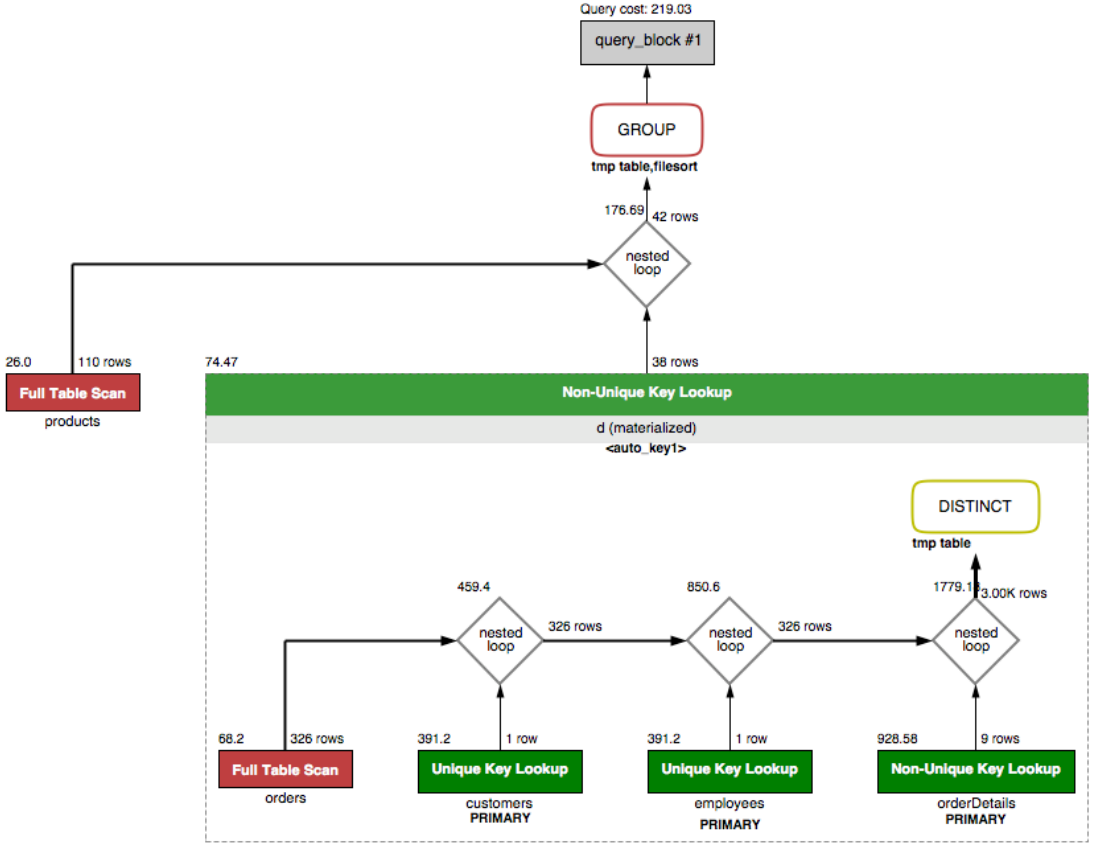
Potential Query

```
select officeCode, productVendor, productCode, sum(quantityOrdered) from
   (select employeeNumber, officeCode, customerNumber, orderNumber, orderLineNumber,
      quantityOrdered, productCode, productVendor
      from (select  *   from
         (
            SELECT distinct * FROM
               (
                  select * from
                     (select * from employees join customers on
                        salesRepEmployeeNumber=employees.employeeNumber) as a
                     join
                     orders using(customerNumber)
               ) as b
            join
               orderDetails using(orderNumber)) as d) as e
         join
            products using(productCode)
   where officeCode=3 and productVendor='Motor City Art Classics') as f
group by officeCode, productCode;
```

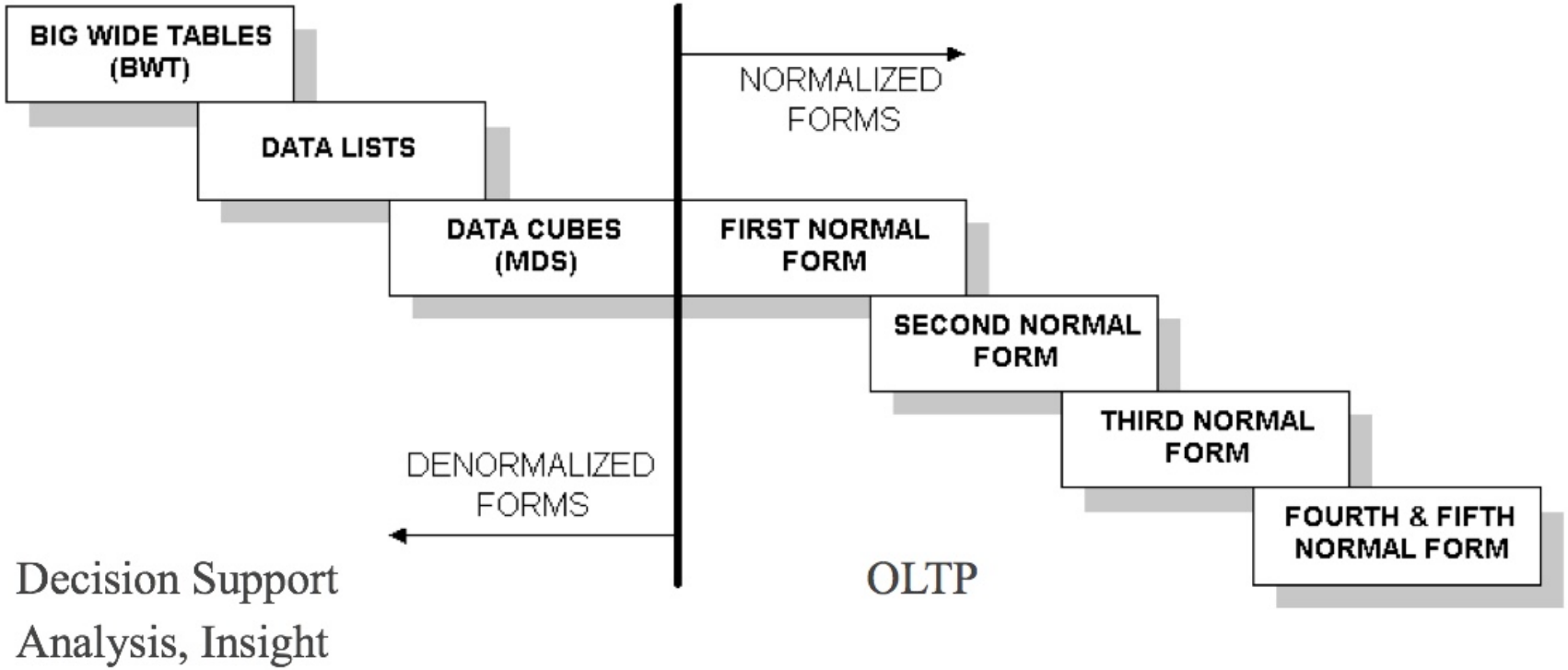
Execution Plan

- I am not completely sure that query is correct.

- I am sure of two things, however:
  - I do not want to write queries like this over and over for each question.
  - The DB engine really, really does not like running queries like this one.



Decision Support versus OLTP



Some Relational OLAP (ROLAP)

Fact Table

customerNumb...	orderNumber	status	quantityOrder...	orderLineNumb...	priceEach	MSRP	buyPrice	productCo...	alesRepEmployeeNumb...	geo_dimension_id	shipped_dimen...	ordered_dimension_...	product_dimension...
363	10100	Shipped	30	3	136.00	170.00	86.70	S18_1749	216	78	3	1	89
363	10100	Shipped	50	2	55.09	60.54	33.30	S18_2248	216	78	3	1	90
363	10100	Shipped	22	4	75.46	92.03	43.26	S18_4409	216	78	3	1	99
363	10100	Shipped	49	1	35.29	41.03	21.75	S24_3969	216	78	3	1	107
128	10101	Shipped	25	4	108.06	127.13	58.48	S18_2325	504	7	266	2	91
128	10101	Shipped	26	1	167.06	168.75	72.56	S18_2795	504	7	266	2	92
128	10101	Shipped	45	3	32.53	33.19	22.57	S24_1937	504	7	266	2	102
128	10101	Shipped	46	2	44.35	44.80	20.61	S24_2022	504	7	266	2	103
181	10102	Shipped	39	2	95.55	102.74	60.62	S18_1342	286	9	267	3	87
181	10102	Shipped	41	1	43.13	53.91	24.26	S18_1367	286	9	267	3	88
121	10103	Shipped	26	11	214.30	214.30	98.58	S10_1949	504	4	268	4	1
121	10103	Shipped	42	4	119.67	147.74	103.42	S10_4962	504	4	268	4	3
121	10103	Shipped	27	8	121.64	136.67	77.90	S12_1666	504	4	268	4	76
121	10103	Shipped	35	10	94.50	116.67	58.33	S18_1097	504	4	268	4	78
121	10103	Shipped	22	2	58.34	60.77	24.92	S18_2432	504	4	268	4	80
121	10103	Shipped	27	12	92.19	101.31	60.78	S18_2949	504	4	268	4	93

Field	Type
customerNumber	int(11)
orderNumber	int(11)
status	varchar(15)
quantityOrdered	int(11)
orderLineNumber	smallint(6)
priceEach	decimal(10,2)
MSRP	decimal(10,2)
buyPrice	decimal(10,2)
productCode	varchar(15)
salesRepEmployeeNumber	int(11)
geo_dimension_id	int(11)
shipped_dimension_id	int(11)
ordered_dimension_id	int(11)
product_dimension_id	int(11)

- Fact
  - What was sold.
  - To whom.
  - For how much.
  - And when.
- Dimensions tell you how to position in space, time, company structure, your product inventory, … …

Facts

Orders

orderNumber	orderDate	customerNum...
10223	2004-02-20	114
10361	2004-12-17	282
10263	2004-06-28	175
10388	2005-03-03	462
10309	2004-10-15	121
10134	2003-07-01	250
10285	2004-08-27	286
10201	2003-12-01	129
10168	2003-10-28	161
10107	2003-02-24	131
10375	2005-02-03	119
10251	2004-05-18	328
10275	2004-07-23	119
10121	2003-05-07	353
10237	2004-04-05	181
10329	2004-11-15	131
10211	2004-01-15	406
10341	2004-11-24	382

OrdersDetails

productCode	quantityOrder	priceEach
S10_1678	37	80.39
S10_1678	20	92.83
S10_1678	34	89.00
S10_1678	42	80.39
S10_1678	41	94.74
S10_1678	41	90.92
S10_1678	36	95.70
S10_1678	22	82.30
S10_1678	36	94.74
S10_1678	30	81.35
S10_1678	21	76.56
S10_1678	59	93.79
S10_1678	45	81.35
S10_1678	34	86.13
S10_1678	23	91.87
S10_1678	42	80.39
S10_1678	41	90.92
S10_1678	41	84.22

Products

MSRP	buyPrice
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81
95.70	48.81

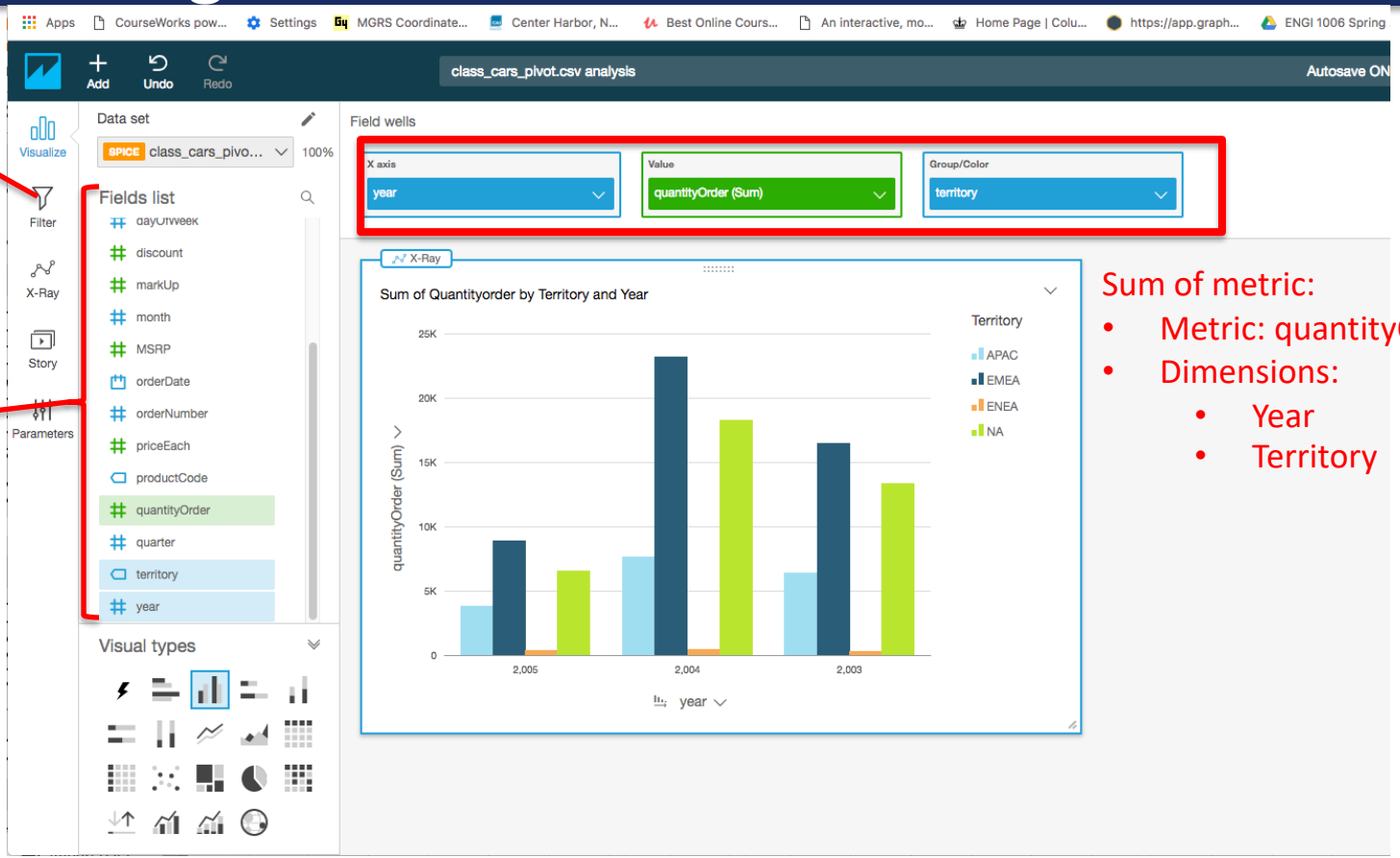
Computed

discount	markUp
0.1866	0.6556
0.0323	0.9015
0.0787	0.8195
0.1866	0.6556
0.0106	0.9424
0.0550	0.8605
0.0000	0.9629
0.1580	0.6761
0.0106	0.9424
0.1721	0.6761
0.2482	0.5737
0.0213	0.9219
0.1721	0.6761
0.1161	0.7580
0.0435	0.8810
0.1866	0.6556
0.0550	0.8605
0.1306	0.7171

AWS QuickSight: Online BI Service

Filter

Facts and Dimensions



Sum of metric:

- Metric: quantityOrdered
- Dimensions:
  - Year
  - Territory

Dimension Examples

Facts and derived facts

Date Dimensions

Geo Dimension

X11 fx

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	orderNumbe	orderDate	custome	productCode	quant	priceEach	MSRP	buyPrice	discount	markUp	revenue	Unit Profit	Profit	year	quarter	month	dayOfWe	day	city	country	territory
2	10223	2/20/04	114	S10_1678	37	80.39	95.7	48.81	0.19	0.66	$2,974.43	31.58	$1,168.4	2004	1	2	6	2	Melbourne	Australia	APAC
3	10361	12/17/04	282	S10_1678	20	92.83	95.7	48.81	0.03	0.90	$1,856.60	44.02	$880.4	2004	4	12	6	1	Chatswood	Australia	APAC
4	10263	6/28/04	175	S10_1678	34	89	95.7	48.81	0.08	0.82	$3,026.00	40.19	$1,366.4	2004	2	6	2	2	San Rafael	USA	NA
5	10388	3/3/05	462	S10_1678	42	80.39	95.7	48.81	0.19	0.66	$3,376.38	31.58	$1,326.3	2005	1	3	5		New Bedforc	USA	NA
6	10309	10/15/04	121	S10_1678	41	94.74	95.7	48.81	0.01	0.94	$3,884.34	45.93	$1,883.1	2004	3	10	6	1	Stavern	Norway	EMEA
7	10134	7/1/03	250	S10_1678	41	90.92	95.7	48.81	0.06	0.86	$3,727.72	42.11	$1,726.5	2003	2	7	3		Paris	France	EMEA
8	10285	8/27/04	286	S10_1678	36	95.7	95.7	48.81	0.00	0.96	$3,445.20	46.89	$1,688.0	2004	3	8	6	2	Cambridge	USA	NA
9	10201	12/1/03	129	S10_1678	22	82.3	95.7	48.81	0.16	0.68	$1,810.60	33.49	$736.7	2003	4	12	2		San Francisc	USA	NA
10	10168	10/28/03	161	S10_1678	36	94.74	95.7	48.81	0.01	0.94	$3,410.64	45.93	$1,653.4	2003	3	10	3	2	San Francisc	USA	NA
11	10375	2/3/05	119	S10_1678	21	76.56	95.7	48.81	0.25	0.57	$1,607.76	27.75	$582.7	2005	1	2	5		Nantes	France	EMEA
12	10107	2/24/03	131	S10_1678	30	81.35	95.7	48.81	0.17	0.68	$2,440.50	32.54	$976.2	2003	1	2	2	2	New York	USA	NA
13	10251	5/18/04	328	S10_1678	59	93.79	95.7	48.81	0.02	0.92	$5,533.61	44.98	$2,653.8	2004	2	5	3	1	Newark	USA	NA
14	10121	5/7/03	353	S10_1678	34	86.13	95.7	48.81	0.12	0.76	$2,928.42	37.32	$1,268.8	2003	2	5	4		Reims	France	EMEA
15	10275	7/23/04	119	S10_1678	45	81.35	95.7	48.81	0.17	0.68	$3,660.75	32.54	$1,464.3	2004	2	7	6	2	Nantes	France	EMEA
16	10237	4/5/04	181	S10_1678	23	91.87	95.7	48.81	0.04	0.88	$2,113.01	43.06	$990.3	2004	2	4	2		New York	USA	NA
17	10329	11/15/04	131	S10_1678	42	80.39	95.7	48.81	0.19	0.66	$3,376.38	31.58	$1,326.3	2004	3	11	2	1	New York	USA	NA
18	10211	1/15/04	406	S10_1678	41	90.92	95.7	48.81	0.06	0.86	$3,727.72	42.11	$1,726.5	2004	1	1	5	1	Paris	France	EMEA
19	10341	11/24/04	382	S10_1678	41	84.22	95.7	48.81	0.13	0.72	$3,453.02	35.41	$1,451.8	2004	3	11	4	2	Salzburg	Austria	EMEA
20	10417	5/13/05	141	S10_1678	66	79.43	95.7	48.81	0.20	0.64	$5,242.38	30.62	$2,020.9	2005	2	5	6	1	Madrid	Spain	EMEA
21	10354	12/4/04	323	S10_1678	42	84.22	95.7	48.81	0.13	0.72	$3,537.24	35.41	$1,487.2	2004	4	12	7		Auckland	New Zealand	APAC
22	10188	11/18/03	167	S10_1678	48	95.7	95.7	48.81	0.00	0.96	$4,593.60	46.89	$2,250.7	2003	3	11	3	1	Bergen	Norway	EMEA
23	10145	8/25/03	205	S10_1678	45	76.56	95.7	48.81	0.25	0.57	$3,445.20	27.75	$1,248.7	2003	3	8	2	2	Pasadena	USA	NA
24	10399	4/1/05	496	S10_1678	40	77.52	95.7	48.81	0.23	0.59	$3,100.80	28.71	$1,148.4	2005	2	4	6		Auckland	New Zealand	APAC
25	10403	4/8/05	201	S10_1678	24	85.17	95.7	48.81	0.13	0.74	$2,044.08	36.36	$872.6	2005	2	4	6		Liverpool	UK	ENEA
26	10318	11/2/04	157	S10_1678	46	84.22	95.7	48.81	0.13	0.72	$3,874.12	35.41	$1,628.8	2004	3	11	3		Allentown	USA	NA
27	10180	11/11/03	171	S10_1678	29	76.56	95.7	48.81	0.25	0.57	$2,220.24	27.75	$804.7	2003	3	11	3	1	Lille	France	EMEA
28	10159	10/10/03	321	S10_1678	49	81.35	95.7	48.81	0.17	0.68	$3,986.15	32.54	$1,594.4	2003	3	10	6	1	San Francisc	USA	NA
29	10299	9/30/04	186	S10_1678	23	76.56	95.7	48.81	0.25	0.57	$1,760.88	27.75	$638.2	2004	3	9	5	3	Helsinki	Finland	EMEA
30	10270	7/19/04	282	S10_1949	21	171.44	214.3	98.58	0.25	0.74	$3,600.24	72.86	$1,530.0	2004	2	7	2	1	Chatswood	Australia	APAC
31	10391	3/9/05	276	S10_1949	24	195.91	214.3	98.58	0.10	0.97	$4,689.24	96.43	$2,314.2	2005	1	3	4		North Sydne	Australia	APAC
32	10250	6/15/04	209	S10_1949	22	177.87	214.3	98.58	0.20	0.80	$5,691.84	79.30	$2,537.2	2004	2	6	3	15	Minato-ku	Japan	APAC