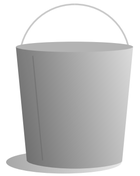
Data Strategies



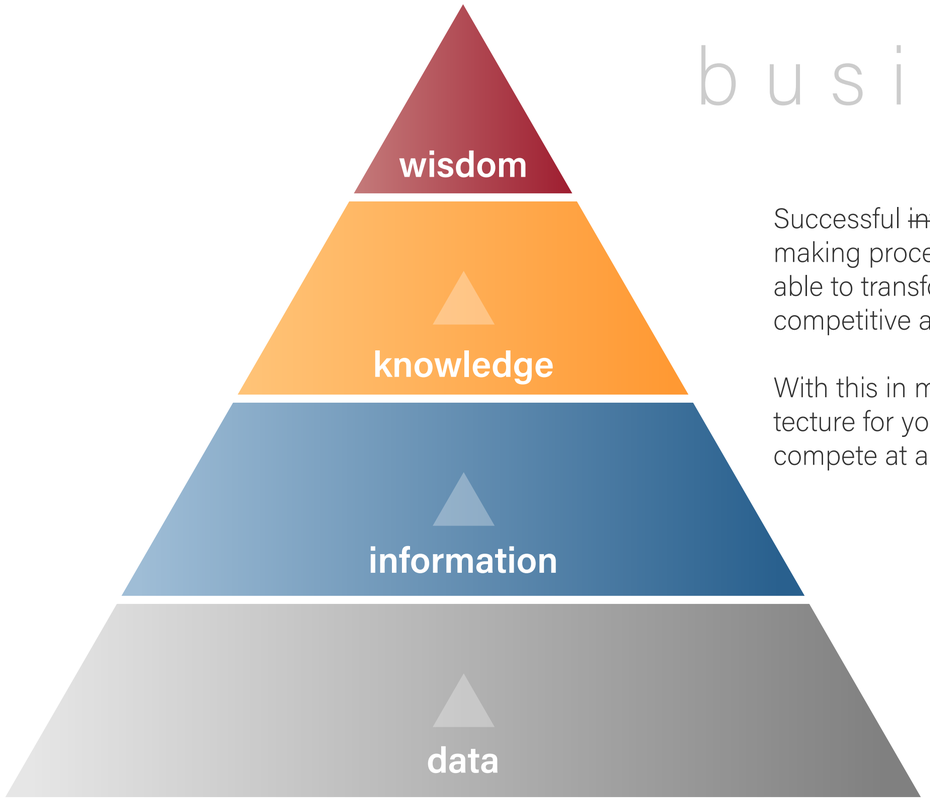
Defining architectural patterns and practices

What is Data?

Data

Data is a set of values of subjects with respect to qualitative or quantitative variables.

Where data fits?



business decisions

Successful ~~information technology~~ organizations leverage appropriate decision making processes. The accuracy, speed, and volume at which an organization is able to transform data to information to knowledge to wisdom is often aligned with competitive advantage.

With this in mind building an appropriate, scalable, performant, and reliable architecture for your data is imperitive to avoid disaster and to capture the opportunity to compete at a higher level.

Information

Information can be thought of as the resolution of uncertainty; it is that which answers the question of "what an entity is" and thus defines both its essence and nature of its characteristics.

Knowledge

Knowledge is a familiarity, awareness, or understanding of someone or something, such as facts, information, descriptions, or skills, which is acquired through experience or education by perceiving, discovering, or learning.

Wisdom

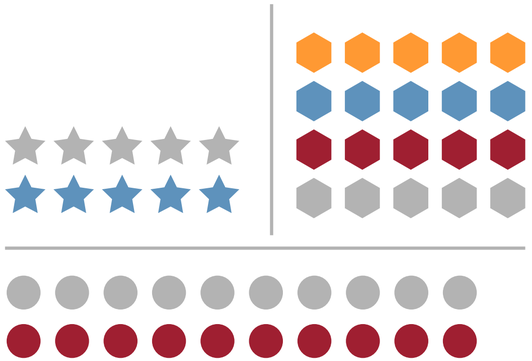
Wisdom, sapience, or sagacity is the ability to think and act using knowledge, experience, understanding, common sense and insight.

Why

Data strategies allow us to control data in an organized, consumable, efficient, and measureable fasion.

Without proper strategies it is easy to lose efficiency and control of your data management capabilities.

With strategies...



Why

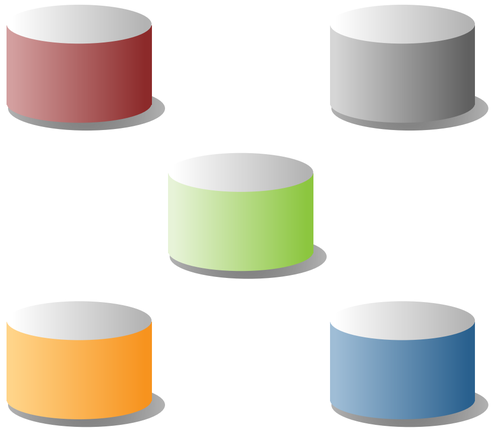
Without strategies...

data strategies

Avoiding disaster with appropriate data management strategies is imperitive to organizations of all sizes. Whether an organization is a start up or well established information based company, avoiding data management pitfalls is critical for success.

What are the types data?

Common Types



Process Data

Process data includes software, metadata, instrumentation, active systems logging, caching, and other internalized or externalized instructional or systems level data.

Referential Data

Referential data includes geographic, pricing, product, configuration vectors, and other validation and value list data.

Operational Data

Operational data includes financial, product, support, user, membership, and other business process data.

Temporal Data

Temporal data uses include DTO staging or temporary persistence of transactions or process data. Long running or resource intensive process flows and ad-hoc information extraction.

Reporting Data

Reporting data includes financial reporting, forcasting, summation of operational, transactional, and process data.

Transactional Data

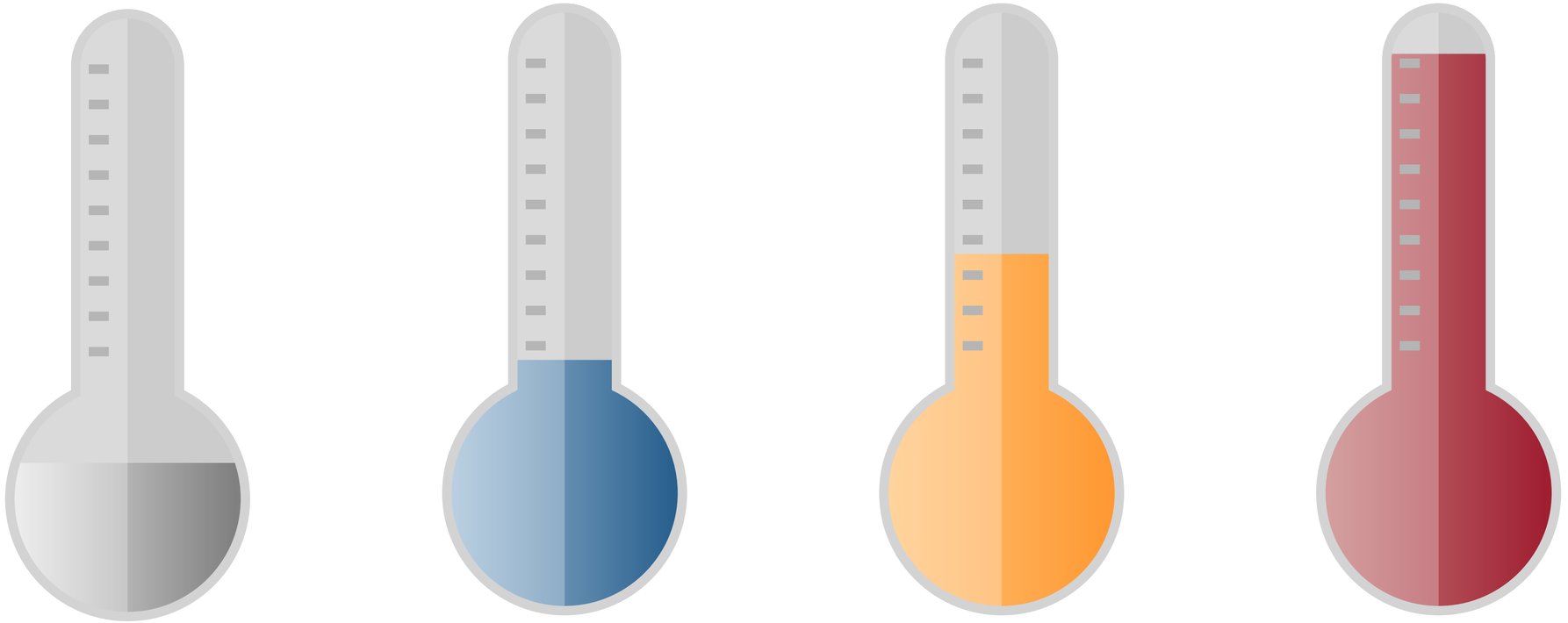
Transactional data includes financial, instrumentation, messaging, and other transactional records that represent business process transactions.

How to measure data?

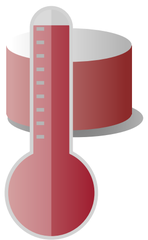
Just as data is a set of values of subject with respect to qualitative or quantitative variables, the set of data used to describe the frequency of use, access, storage, transmission, collection, distribution, relevence, size, speed, or any other important characteristic of itself can be converted to information.

Using this information and knowledge about an organization's data needs is the first step in developing appropriate data strategies.

Temperature of Data

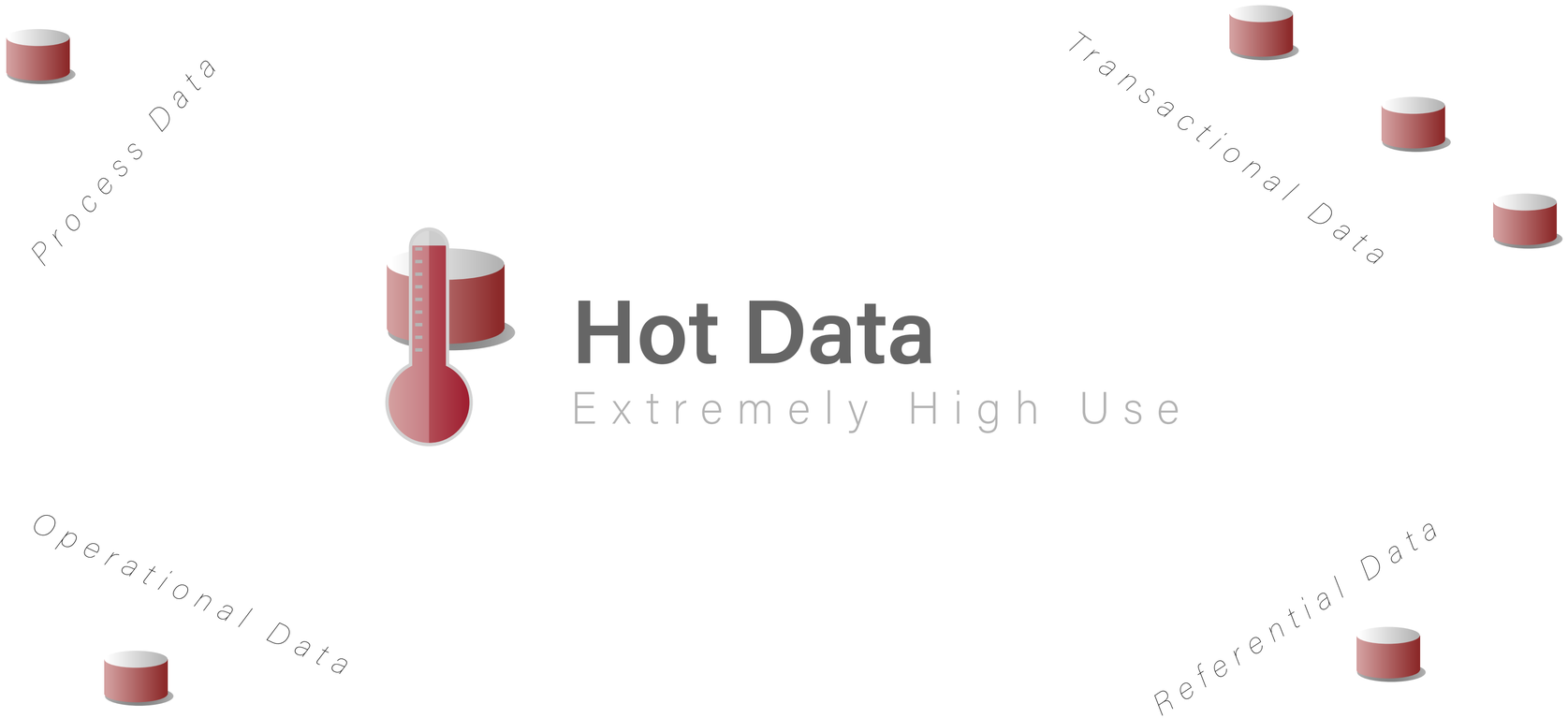


Frequency of Use



Hot Data

Extremely High Use


Process Data
Transactional Data
Hot Data
Extremely High Use
Operational Data
Referential Data

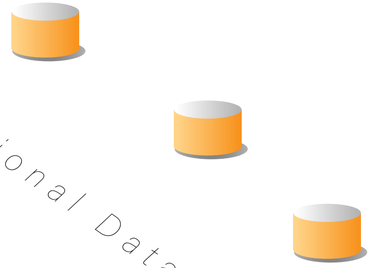
Warm Data

High Use

Temporal Data

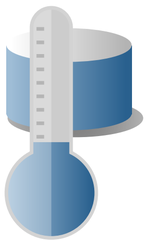
Warm Data

High Use



Transactional Data (Warm)

Reporting Data

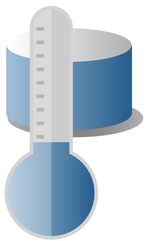


Cool Data

Low Use

Log Data

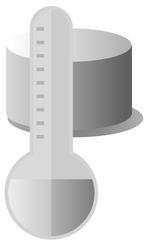
Transactional Data (Cool)



Cool Data

Low Use

Reporting Data

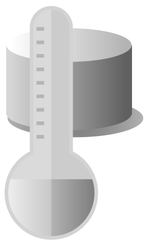


Frozen Data

Extremely Low Use

Log Data

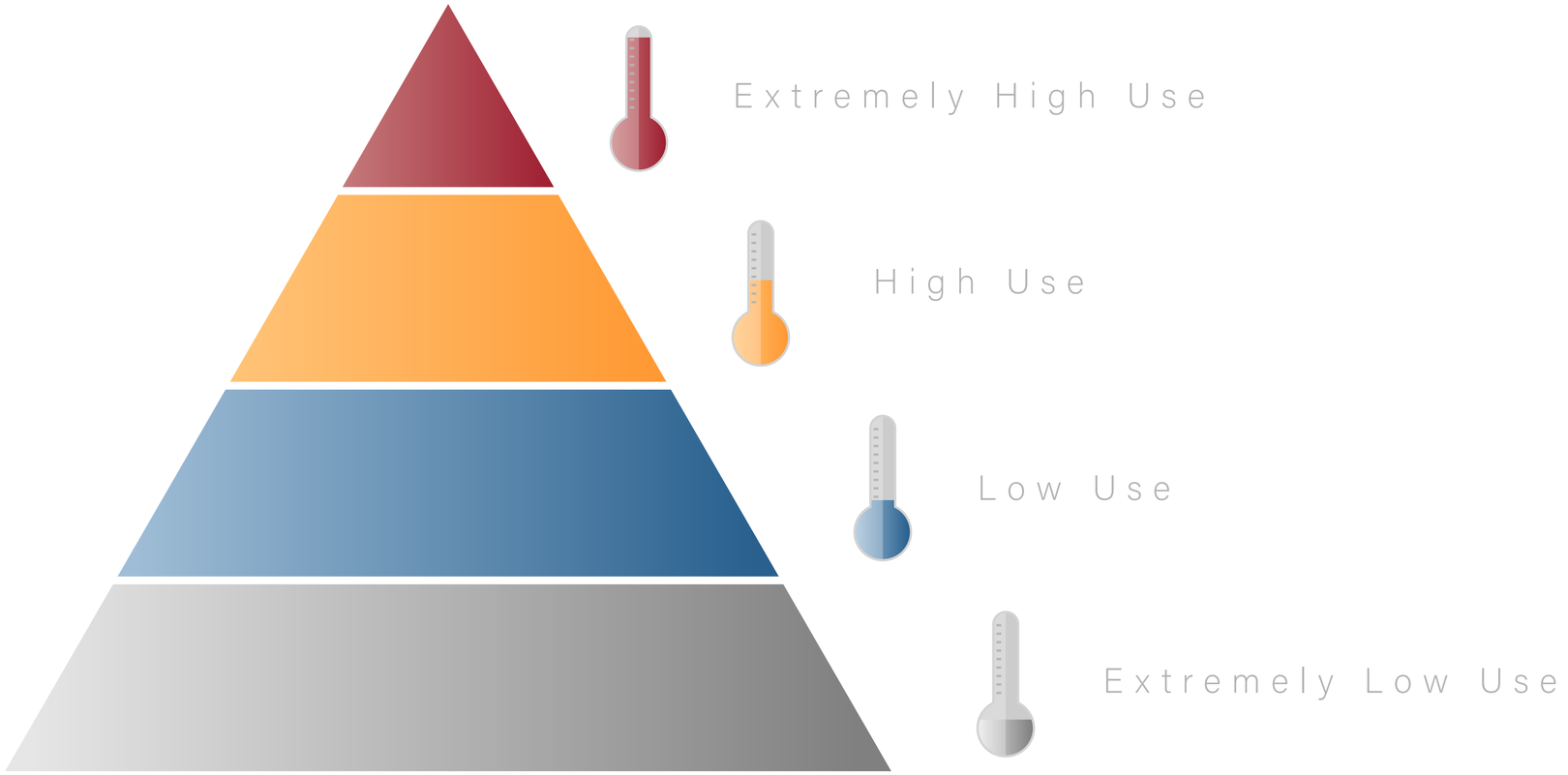
Transactional Data (Frozen)



Frozen Data

Extremely Low Use

Reporting Data

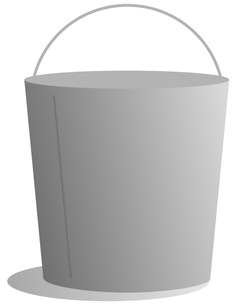


Typical Volume Distribution

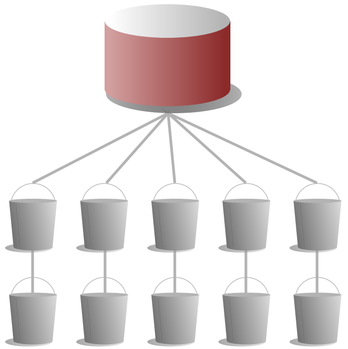
Common Strategies

Partitioning

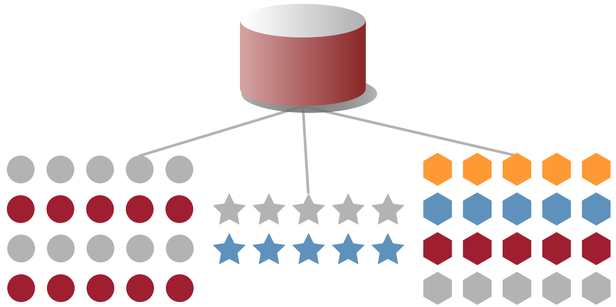
Buckets of Things



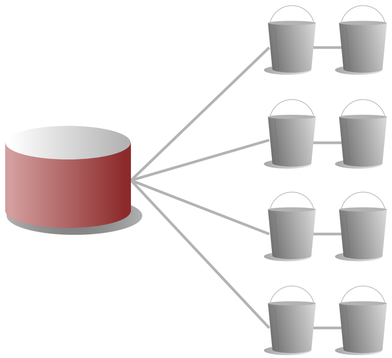
Vertical Partitioning



Functional Partitioning (Bounded Context)

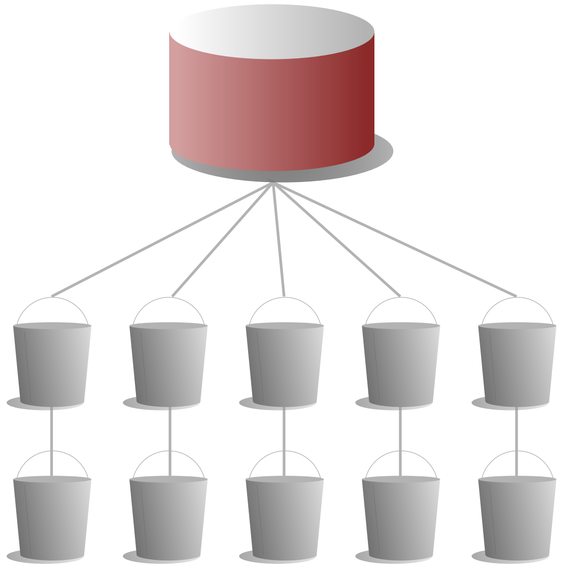


Horizontal Partitioning (Sharding)



Strategies

Vertical Partitioning

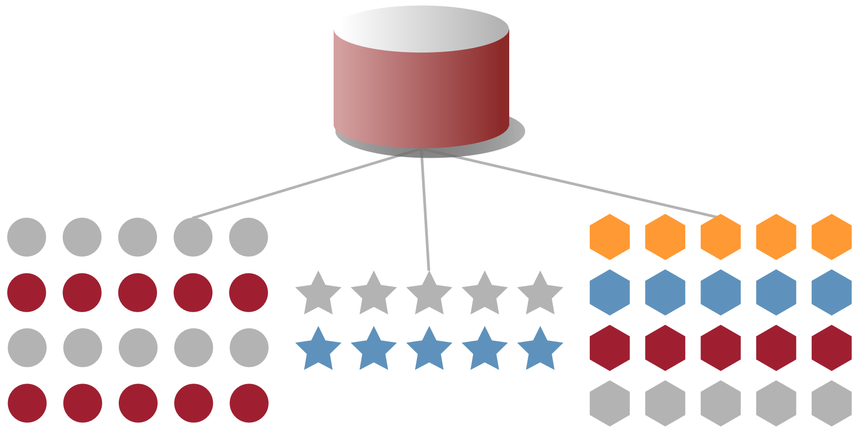


Vertical Partitioning Strategy

Vertical partitioning is a strategy of partitioning data into the same or seperate data stores based on functional, storage requirements, frequency of use or other logical factors. Typically one or more sets of values that comprise a portion of the data are split to seperate data areas and are typically referenced by key in a 1 to 1 relationship.

Strategies

Functional Partitioning (Bounded Context)

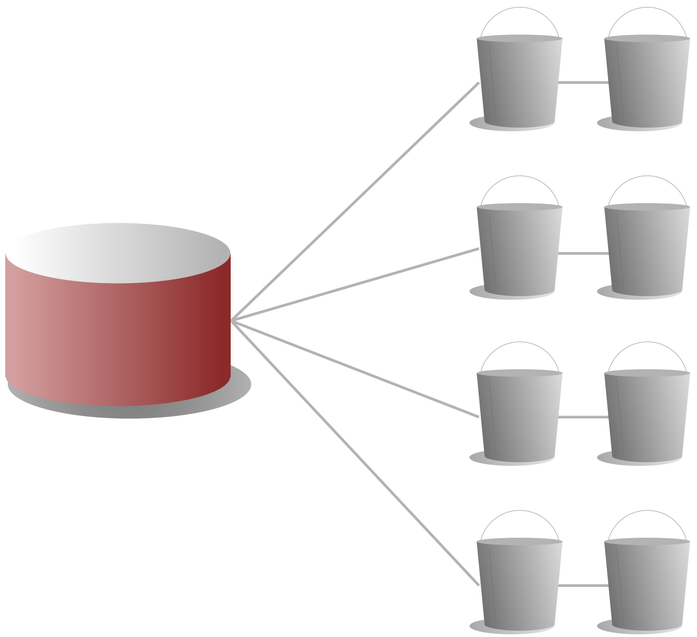


Functional Partitioning Strategy

Functional partitioning is a strategy of partitioning data by how it is used by each bounded context in the system. Functionally similiar data is typically partitioned together along organizational, departmental, data sources, read-write vs read-only, frequency of use, and volume concerns.

Strategies

Horizontal Partitioning (Sharding)



Lookup Strategy

Shard stategy that groups data using a shard key broad groups of related data. For example a muti-tenant system may create a shard per tenant or client.

Range Strategy

Range stategy group and shard similiar data based on qualitative or quantitave data properties. Typical sharding vectors include object state, date and/or datetime, geo-graphic region, and any other values that can fall within a range specification.

Hash Strategy

Hash strategy uses hashing algorithims typically designed to equally distribute the mapping of data to the appropriate shard. Commonly used to avoid hot spots in data distribution.

Common Partitioning Scenarios

Every organization and every DBA, Developer, and IT professional uses data partitioning strategies of some kind every day even without realizing it.

As a DBA do you keep every SQL script you use for ad-hoc management tasks stored within every database. Most likely you store those in a management database, source control system, network file share, or at least on a local hard drive.

As a developer you don't keep all configuration settings, HTML, javascript, or even the code you write in a single data store. Most logging frameworks write log files using a horizontal sharding strategy of once per day.

These are all naturally occuring examples of partitioning and data strategies.

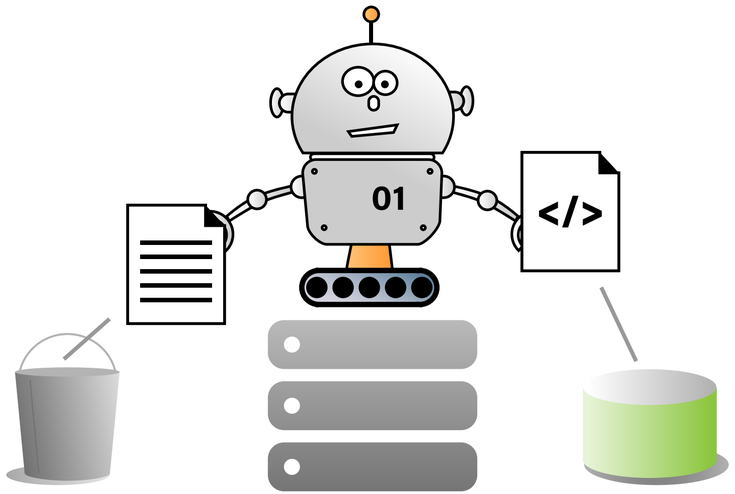
Process Data

Functional, Vertical, and Horizontal Candidates

Process data takes many forms and is a good candidate for a functional partitioning that requires distributing, sharing, persisting, or transmitting the data within one or more systems and applications.

Internal system data, instrumentation data. logging, queue, and other types of cached or buffer data can often benefit from horizontal partitioning strategies.

Instrumentation, debugging, or logging data occassionally can be vertically partitioned to allow fast access.



Process Data

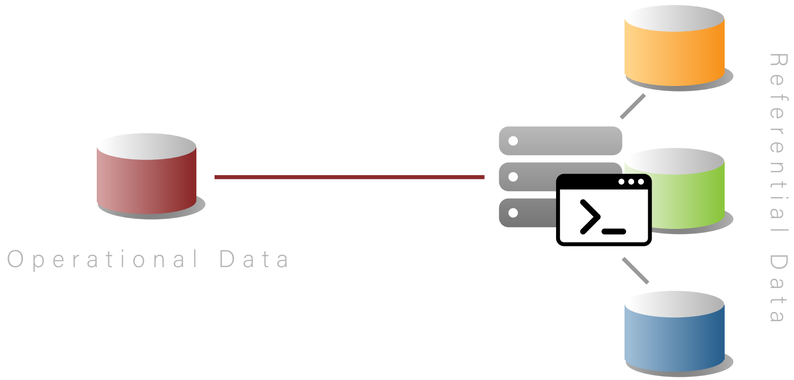
Referential Data

Functional, Vertical, and Horizontal Candidates

High read / low write data with low to medium storage needs is a good candidate for a functional strategy that could distribute the data directly within application memory.

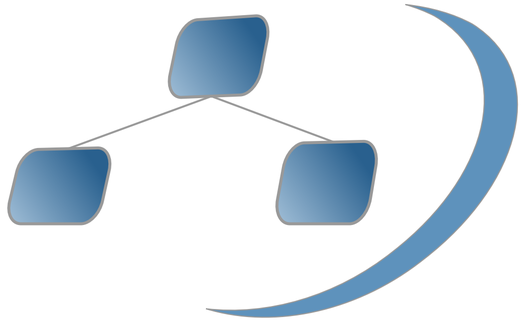
High read / low write data with medium to high storage needs could be stored in a read optimized sharded data store to allow fast retrieval without impacting other data processing.

High read data that contains large text or serialized/binary data can be vertically partitioned to allow fast retrieval using distributed caching, storage, and/or CDN type mechanisms.



Transactional Data

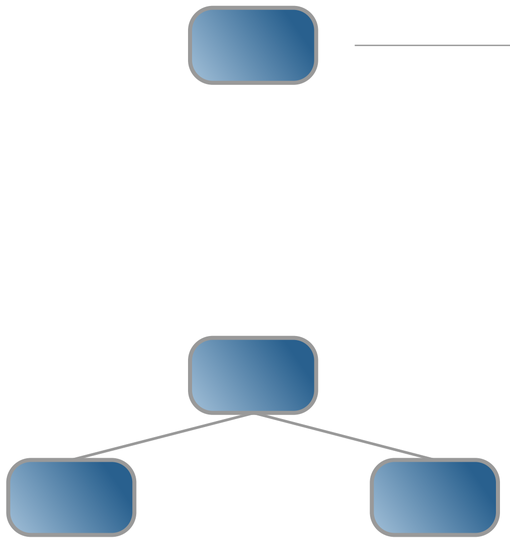
Sliding Windows Data



Database Normalization

“Normalize till it hurts, Denormalize till it works!”

Picking the correct mix of normalized and denormalized data in a relational database is important. The characteristics of how data is used, the volume, and it’s relationships with other data must be considered heavily in designing RDBMS models. Simply defining unnormalized, normalized, or denormalized data haphazardly because it is initially the easiest or even the most elegant or complex approach can have severe future consequences.



```
/* ================ Unnormalized Form ================ */

SELECT [Id]
     ,[Shape]
     ,[Color]
FROM [dbo].[NoStrategy]

/* ================ Normalized Form ================ */


SELECT [Id]
     ,[ShapeId] -- maps to [Id] from [Strategy01Shapes] table
     ,[ColorId] -- maps to [Id] from [Strategy01Colors] table
  FROM [dbo].[Strategy01]
GO

SELECT [Id]
     ,[Color]
  FROM [dbo].[Strategy01Colors]
GO

SELECT [Id]
     ,[Shape]
  FROM [dbo].[Strategy01Shapes]
GO
```

```
/* ================ Unnormalized Form ================ */

SELECT [Id]
      ,[Shape]
      ,[Color]
FROM [dbo].[NoStrategy]
WHERE [Color] = 'orange'
AND [Shape] = 'hexagon'
```

Id	Shape	Color
1	hexagon	orange
2	hexagon	orange
3	hexagon	orange
4	HEXAGON	Orange
5	hexagon	orange

```
/* ================ Normalized Form ================ */

SELECT a.[Id]
      ,b.[Color]
      ,c.[Shape]
FROM [dbo].[Strategy01] as a
JOIN [dbo].[Strategy01Colors] as b
ON a.[ColorId] = b.[Id]
JOIN [dbo].[Strategy01Shapes] as c
ON a.[ShapeId] = c.[Id]
WHERE b.[Color] = 'orange'
AND c.[Shape] = 'hexagon'
```

Id	Color	Shape
11	orange	hexagon
12	orange	hexagon
13	orange	hexagon
14	orange	hexagon
15	orange	hexagon

```
/* ================ Denormalized Indexed View / Materialized View ================ */

SET NUMERIC_ROUNDABORT OFF;
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT,
   QUOTED_IDENTIFIER, ANSI_NULLS ON;

IF OBJECT_ID ('dbo.vStrategy01', 'view') IS NOT NULL
   DROP VIEW [dbo].[vStrategy01];
GO
CREATE VIEW [dbo].[vStrategy01]
   WITH SCHEMABINDING
   AS
    SELECT a.[Id]
      ,b.[Color]
      ,c.[Shape]
    FROM [dbo].[Strategy01] as a
    JOIN [dbo].[Strategy01Colors] as b
    ON a.[ColorId] = b.[Id]
    JOIN [dbo].[Strategy01Shapes] as c
    ON a.[ShapeId] = c.[Id]
GO

CREATE UNIQUE CLUSTERED INDEX IDX_vStrategy01
   ON [dbo].[vStrategy01] ([Id],[Color],[Shape]);
GO
```

```
SELECT [Id]
      ,[Color]
      ,[Shape]
FROM [dbo].[vStrategy01]
```

Unnormalized Form

Id	Shape	Color
1	hexagon	orange
2	hexagon	orange
3	hexagon	orange
4	HEXAGON	Orange
5	hexagon	orange
6	hexagon	blue
7	hexagon	blue
8	hexagon	blue
9	hexagon	blue
10	hexagon	blue

Normalized Form

Id	ShapeId	ColorId
1	1	4
2	1	4
3	1	4
4	1	4
5	1	4
6	1	2
7	1	2
8	1	2
9	1	2
10	1	2

Id	Color
1	orange
2	blue
3	red
4	grey

Id	Shape
1	star
2	hexagon
3	circle

Denormalized Form

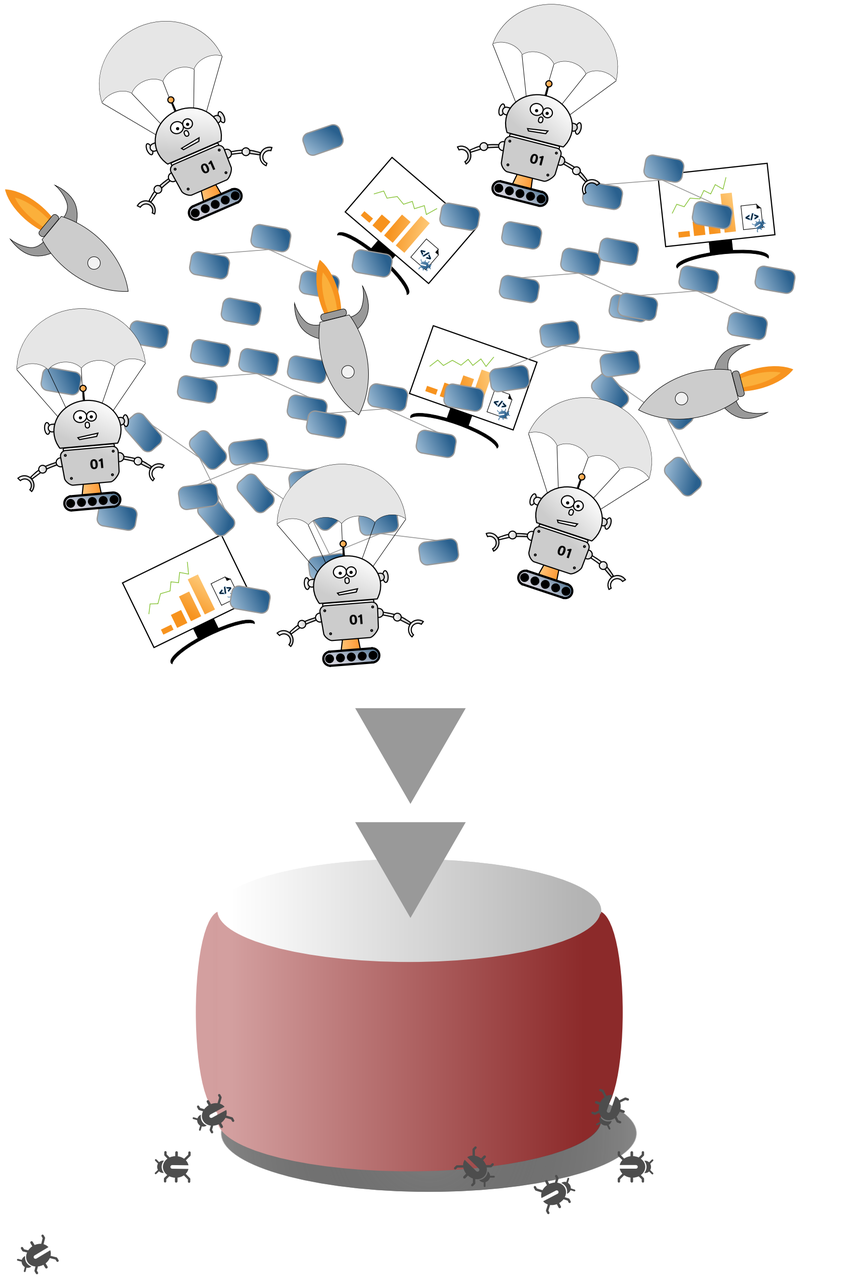
Id	Color	Shape
1	grey	star
2	grey	star
3	grey	star
4	grey	star
5	grey	star

The Database is King!

One database to rule them all strategy.

One of the most dangerous pitfalls experienced by organizations is the belief by many that an RDBMS database is king. Simply dumping all data and business logic into a single database responsible for all facets of data management sets an organization on a certain path to doom. As new requirements, new data, and new business logic are added the complexity and fragility of all systems increases. Digging an organization out from that pit of doom becomes more difficult with time and organizational success.

Using a single authoritative database model to satisfy all of an enterprises' data architecture can result in a non-performant, "jack of all trades, master of none" solution.



Avoiding Apocolypse

One single monolithic database without proper data strategies is a sure fire recipe for an enterprise wide disaster.

Partitioning Can Reduce Impact

A Hardware Failure Comparison

Object-Relational Impedance Mismatch

Object-relational impedence mismatch is the observation of structural and technical challenges associated with inter-process translation of object-oriented programming languages and relational database management systems. In a mathmatical sense objects represent each other in the form of a graph whereas relational database management systems use relational algebra to define heterogeneous tuples (rows) of data.

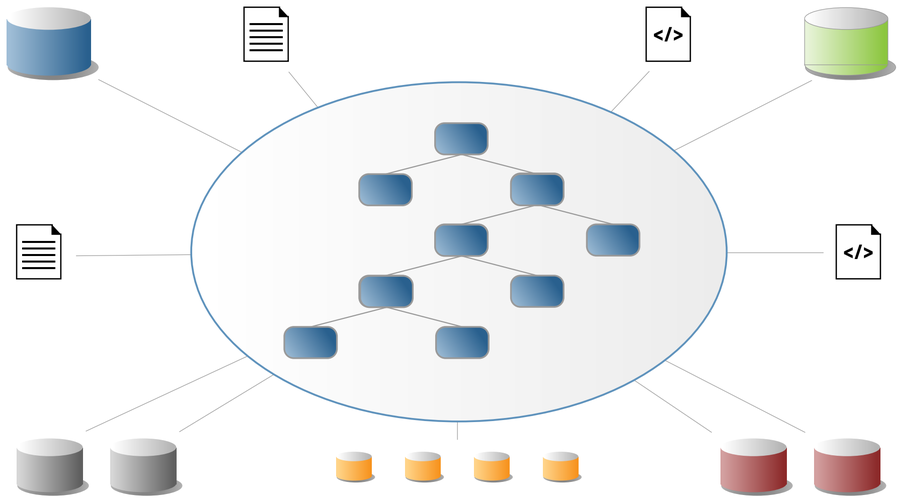
SQL (Structured Query Language) was never intended to provide a robust object model interface language and is inefficient at providing an organization's entire canonical model needs. SQL was designed to effectively store, query, and sort large sets of data and does this extremely well. Attempting to include object-oriented functionality and business logic in relational database systems is bad practice and will polute the database schema, degrade performance, increase complexity, and will increase the potential for errors.

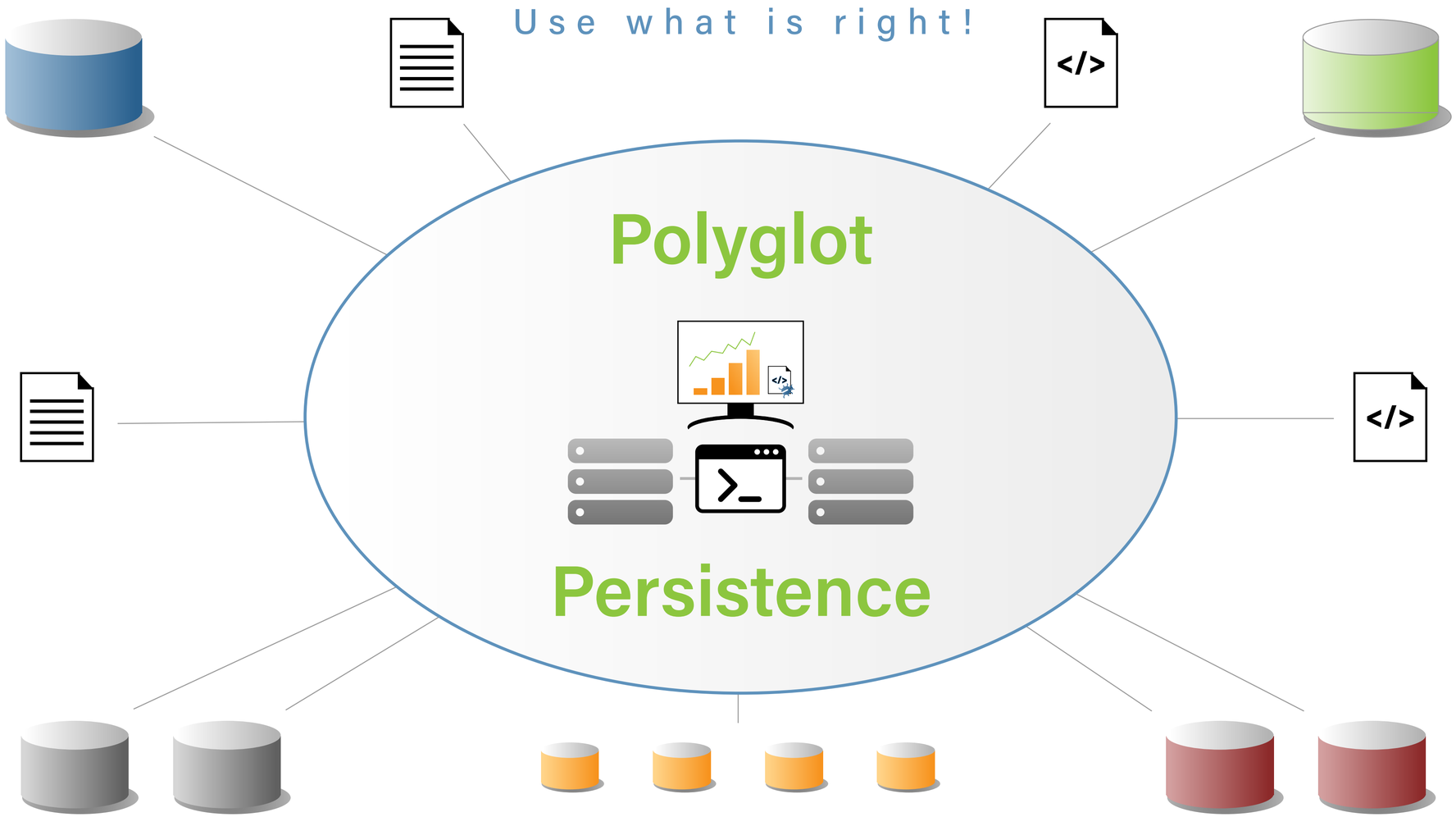
Enterprise Design Pattern

Building an approriate canonical data model

A canonical data model is a standardized enterprise systems data model design pattern. This pattern attempts to define all common core data entities in an application and data storage independent way. It provides a common and consistent approach to manage data in a more flexible, organized, efficient, and reliable way.

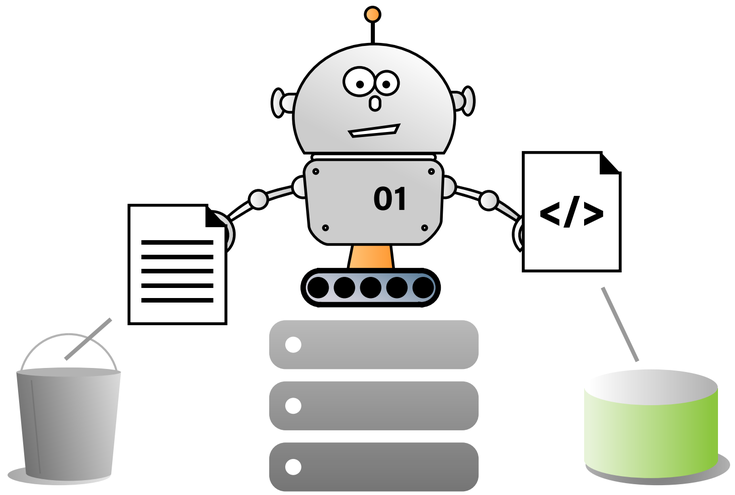
In this way object-relational impedent mismatch is handled unseen and with clearer definition and fewer required implementations thus reducing complexity and potential for error. It also eliminates direct coupling and dependencies that typically cause a condition of frozen or fragile code where any minor extension or change risks destroying an entire organization's code base and data systems.

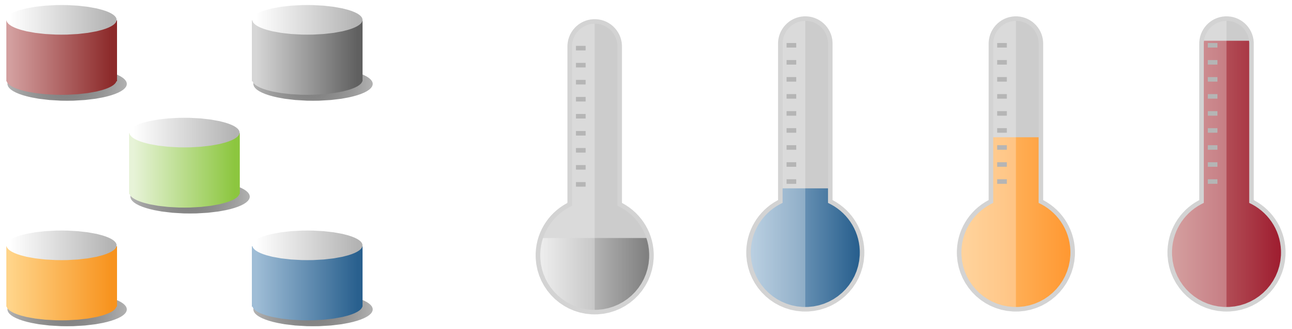


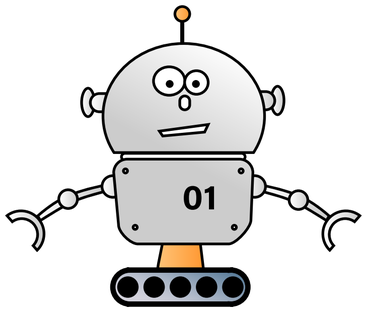

Use what is right!
Polyglot
Persistence

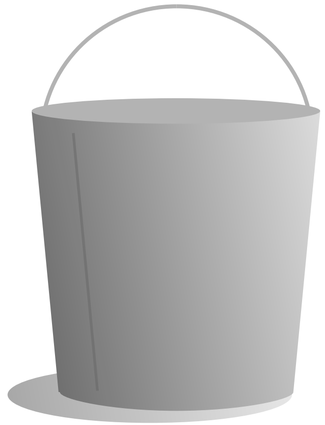
Know Your Data!

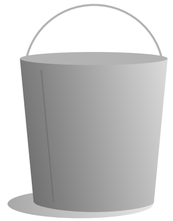
Effectively implementing an appropriate polyglot persistence strategy requires that you understand the types, temperature, and how your data is used.











Size your Strategies