[AUDIT 2](#_Toc16514888)

[Vocabulary 2](#_Toc16514889)

[Inshoring SOX functions 2](#_Toc16514890)

[RPA 2](#_Toc16514891)

[SKILLS 2](#_Toc16514892)

[AWS 2](#_Toc16514893)

[HADOOPP 2](#_Toc16514894)

[STREAM 2](#_Toc16514895)

[**SPARK** 2](#_Toc16514896)

[**Core** 2](#_Toc16514897)

[**SQL** 2](#_Toc16514898)

[**Streaming** 3](#_Toc16514899)

[Mlib Machine Learning Library 3](#_Toc16514900)

[GraphX 3](#_Toc16514901)

[APACHE KAFKA 3](#_Toc16514902)

[DJANGO 3](#_Toc16514903)

[SQL 4](#_Toc16514904)

[Oracle PL/SQL 4](#_Toc16514905)

[T-SQL 5](#_Toc16514906)

[SQL CLR 5](#_Toc16514907)

[POWER BI 5](#_Toc16514908)

[MONGODB 5](#_Toc16514909)

[Microservices 6](#_Toc16514910)

[API 7](#_Toc16514911)

[12 FACTORS 7](#_Toc16514912)

[REST 7](#_Toc16514913)

[DOCKER 7](#_Toc16514914)

[KUBERNETES 8](#_Toc16514915)

[PYTHON 8](#_Toc16514916)

[LUIGI 8](#_Toc16514917)

[DASK 8](#_Toc16514918)

[PANDAS 8](#_Toc16514919)

[SQL ALCHEMY 8](#_Toc16514920)

[R 8](#_Toc16514921)

[JAVASCRIPT 8](#_Toc16514922)

[SCIKIT-LEARN 8](#_Toc16514923)

[TENSORFLOW 8](#_Toc16514924)

[PYTORCH 8](#_Toc16514925)

[MACHINE LEARNING 8](#_Toc16514926)

[Regression 8](#_Toc16514927)

[Boost 8](#_Toc16514928)

[Deep Learning 8](#_Toc16514929)

[Feature Engineering 8](#_Toc16514930)

[EXHIBIT 9](#_Toc16514931)

[Audit Lifecycle 9](#_Toc16514932)

[AGILE AUDIT 9](#_Toc16514933)

[APPLICATION DEVELOPMENT AGILE 11](#_Toc16514934)

[NOVA Target State 12](#_Toc16514935)

[IDR Data Governance 13](#_Toc16514936)

[IDR Data Quality Lifecycle 13](#_Toc16514937)

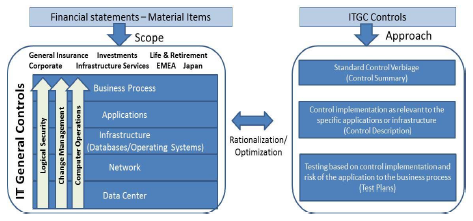
[IDR State Architecture 15](#_Toc16514938)

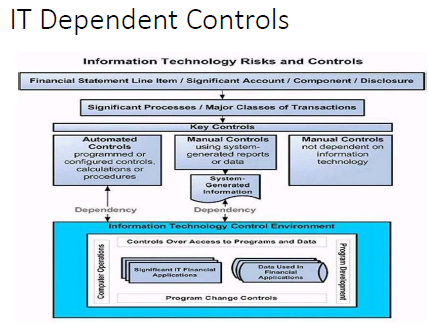
# AUDIT

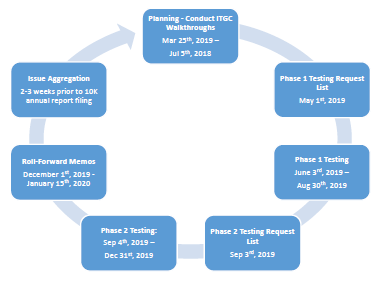
## Vocabulary

⬩**ELC** (Entity Level Control) ⬩**OSP** (Outside Service Providers) Oversight ⬩**LU**= Least (privileged) User Access ⬩**NTE**= nature, timing and extent ⬩**SSAE16 SOC1, SOC2** ⬩**Audit Writing 5C’s**: Criteria (what should be), Condition (the current state), Cause (the reason for the difference), Consequence (effect), Corrective action plans/recommendations.

## Inshoring SOX functions







## RPA

⬩Initiated from Accounts Payable (NJ), DBA <Tax, FIS Billing, FP&A Planning&Analysis, Comptrollers> ⬩Consultant: GENPACT ⬩Process 1: Batch creation + Monies moving ⬩Process 2: VOID/STOP Payment (Reversal) ⬩Systems AWD (Automated Work Distributor Imaging & Workflow), OASYS PrC (Fixed annuity Admin) ⬩ RPA: OPENSPAN PEGASYSTEMS

# SKILLS

## AWS

•**Elastic Compute Cloud (EC2)**, a service for provisioning computing resources on

demand •**Simple Storage Service (S3)**, online storage for opaque data •**Elastic Block Store (EBS)**, persistent disk-like storage for EC2 instances, in 2008 •**Elastic MapReduce (EMR)**, a service providing Hadoop-like clusters for running MapReduce (and later Apache Hive and Apache Pig) jobs, in 2009 •**Relational Database Service (RDS)**, a service for managing relational database server instances running in AWS, also in 2009

⬩**Instance types**: heavy compute capability, vast storage, economy, or simply general-purpose use ⬩**Availability zones** independent within a region, but faster interconnections ⬩**Temporary instance** can disappear after some time ⬩**Images** what instances are running: operating system type and version, the software packages that are available, and applications that are installed. These considerations are all bundled up into images ⬩**CIDR** (Classless Inter-Domain Routing) [notation](https://erikberg.com/notes/networks.html)

## HADOOPP

Key Features: ➊**HDFS** (Hadoop Distributed File System) adapted to work with huge or large scale bandwidth ➋**MapReduce**: Set up model for the processing of Big Data ➌**YARN**: resource scheduler or assistant for Hadoop resource management ➍**Hadoop Libraries**: enables third party programs to work with Hadoop

## STREAM

Emerging use cases for Spark and Kafka: •**ETL and data engineering**: Data preparation for all analytics •**AI and machine learning**: Massively scalable, parallel processing •**Business Intelligence**: Next-generation business intelligence with big and fast data •**Streaming apps**: Real-time processing of streaming data for the internet of things, artificial intelligence (AI) and natural language processing (NLP).

**SPARK**

**Core**

Spark Core is the foundation of the overall project. It provides distributed task dispatching, scheduling, and basic [I/O](https://en.wikipedia.org/wiki/I/O_interface) functionalities, exposed through an application programming interface (for [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)), and [R](https://en.wikipedia.org/wiki/R_(programming_language))) centered on the RDD [abstraction](https://en.wikipedia.org/wiki/Abstraction_(computer_science)) (the Java API is available for other JVM languages, but is also usable for some other non-JVM languages that can connect to the JVM, such as [Julia](https://en.wikipedia.org/wiki/Julia_(programming_language))[[16]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-16)). This interface mirrors a [functional](https://en.wikipedia.org/wiki/Functional_programming)/[higher-order](https://en.wikipedia.org/wiki/Higher-order_programming) model of programming: a "driver" program invokes parallel operations such as map, [filter](https://en.wikipedia.org/wiki/Filter_(computer_science)) or reduce on an RDD by passing a function to Spark, which then schedules the function's execution in parallel on the cluster.[[2]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-hc10-2) These operations, and additional ones such as [joins](https://en.wikipedia.org/wiki/Join_(database)), take RDDs as input and produce new RDDs. RDDs are [immutable](https://en.wikipedia.org/wiki/Immutable_object) and their operations are [lazy](https://en.wikipedia.org/wiki/Lazy_evaluation); fault-tolerance is achieved by keeping track of the "lineage" of each RDD (the sequence of operations that produced it) so that it can be reconstructed in the case of data loss. RDDs can contain any type of Python, Java, or Scala objects.

Besides the RDD-oriented functional style of programming, Spark provides two restricted forms of shared variables: *broadcast variables* reference read-only data that needs to be available on all nodes, while *accumulators* can be used to program reductions in an [imperative](https://en.wikipedia.org/wiki/Imperative_programming) style.[[2]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-hc10-2) A typical example of RDD-centric functional programming is the following Scala program that computes the frequencies of all words occurring in a set of text files and prints the most common ones. Each map, flatMap (a variant of map) and reduceByKey takes an [anonymous function](https://en.wikipedia.org/wiki/Anonymous_function) that performs a simple operation on a single data item (or a pair of items), and applies its argument to transform an RDD into a new RDD.

**val** conf **=** **new** **SparkConf**().setAppName("wiki\_test") // create a spark config object

**val** sc **=** **new** **SparkContext**(conf) // Create a spark context

**val** data **=** sc.textFile("/path/to/somedir") // Read files from "somedir" into an RDD of (filename, content) pairs.

**val** tokens **=** data.flatMap(**\_**.split(" ")) // Split each file into a list of tokens (words).

**val** wordFreq **=** tokens.map((**\_**, **1**)).reduceByKey(**\_** + **\_**) // Add a count of one to each token, then sum the counts per word type.

wordFreq.sortBy(s **=>** -s.\_2).map(x **=>** (x.\_2, x.\_1)).top(**10**) // Get the top 10 words. Swap word and count to sort by count.

**SQL**

Spark [SQL](https://en.wikipedia.org/wiki/SQL) is a component on top of Spark Core that introduced a data abstraction called DataFrames,[[a]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-18) which provides support for structured and [semi-structured data](https://en.wikipedia.org/wiki/Semi-structured_data). Spark SQL provides a [domain-specific language](https://en.wikipedia.org/wiki/Domain-specific_language) (DSL) to manipulate DataFrames in [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), or [Python](https://en.wikipedia.org/wiki/Python_(programming_language)). It also provides SQL language support, with [command-line interfaces](https://en.wikipedia.org/wiki/Command-line_interface) and [ODBC](https://en.wikipedia.org/wiki/Open_Database_Connectivity)/[JDBC](https://en.wikipedia.org/wiki/Java_Database_Connectivity) server. Although DataFrames lack the compile-time type-checking afforded by RDDs, as of Spark 2.0, the strongly typed DataSet is fully supported by Spark SQL as well.

**import** **org.apache.spark.sql.SparkSession**

**val** url **=** "jdbc:mysql://yourIP:yourPort/test?user=yourUsername;password=yourPassword" // URL for your database server.

**val** spark **=** **SparkSession**.builder().getOrCreate() // Create a Spark session object

**val** df **=** spark

.read

.format("jdbc")

.option("url", url)

.option("dbtable", "people")

.load()

df.printSchema() // Looks the schema of this DataFrame.

**val** countsByAge **=** df.groupBy("age").count() // Counts people by age

//or alternatively via SQL:

//df.createOrReplaceTempView("people")

//val countsByAge = spark.sql("SELECT age, count(\*) FROM people GROUP BY age")

**Streaming**

park Streaming uses Spark Core's fast scheduling capability to perform [streaming analytics](https://en.wikipedia.org/wiki/Event_stream_processing). It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, thus facilitating easy implementation of [lambda architecture](https://en.wikipedia.org/wiki/Lambda_architecture). However, this convenience comes with the penalty of latency equal to the mini-batch duration. Other streaming data engines that process event by event rather than in mini-batches include [Storm](https://en.wikipedia.org/wiki/Storm_(event_processor)) and the streaming component of [Flink](https://en.wikipedia.org/wiki/Apache_Flink).[[20]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-21) Spark Streaming has support built-in to consume from [Kafka](https://en.wikipedia.org/wiki/Apache_Kafka), [Flume](https://en.wikipedia.org/wiki/Apache_Flume), [Twitter](https://en.wikipedia.org/wiki/Twitter#Implementation), [ZeroMQ](https://en.wikipedia.org/wiki/ZeroMQ), [Kinesis](https://en.wikipedia.org/wiki/Amazon_Web_Services#Database), and [TCP/IP sockets](https://en.wikipedia.org/wiki/Network_socket).[[21]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-22) In Spark 2.x, a separate technology based on Datasets, called Structured Streaming, that has a higher-level interface is also provided to support streaming.[[22]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-23)

#### Mlib Machine Learning Library

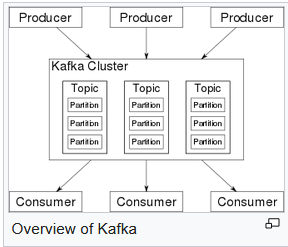
Spark MLlib is a [distributed](https://en.wikipedia.org/wiki/Distributed_computing) machine-learning framework on top of Spark Core that, due in large part to the distributed memory-based Spark architecture, is as much as nine times as fast as the disk-based implementation used by [Apache Mahout](https://en.wikipedia.org/wiki/Apache_Mahout) (according to benchmarks done by the MLlib developers against the [alternating least squares](https://en.wikipedia.org/wiki/Linear_regression) (ALS) implementations, and before Mahout itself gained a Spark interface), and [scales](https://en.wikipedia.org/wiki/Scale_(computing)) better than [Vowpal Wabbit](https://en.wikipedia.org/wiki/Vowpal_Wabbit).[[23]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-24) An overview of Spark MLlib is exist.[[24]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-25) Many common machine learning and statistical algorithms have been implemented and are shipped with MLlib which simplifies large scale machine learning [pipelines](https://en.wikipedia.org/wiki/Pipeline_(software)), including:

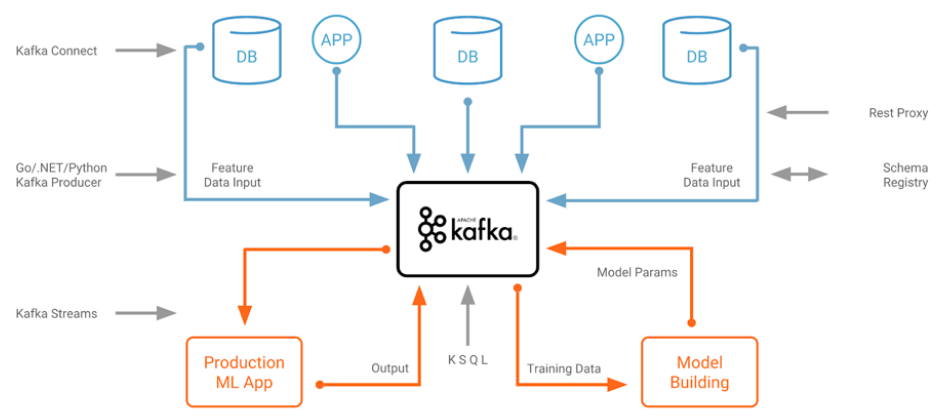
* [summary statistics](https://en.wikipedia.org/wiki/Summary_statistics), [correlations](https://en.wikipedia.org/wiki/Correlation_and_dependence), [stratified sampling](https://en.wikipedia.org/wiki/Stratified_sampling), [hypothesis testing](https://en.wikipedia.org/wiki/Hypothesis_testing), random data generation[[25]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-26)
* [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression](https://en.wikipedia.org/wiki/Regression_analysis): [support vector machines](https://en.wikipedia.org/wiki/Support_vector_machines), [logistic regression](https://en.wikipedia.org/wiki/Logistic_regression), [linear regression](https://en.wikipedia.org/wiki/Linear_regression), decision trees, [naive Bayes classification](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)
* [collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) techniques including alternating least squares (ALS)
* [cluster analysis methods](https://en.wikipedia.org/wiki/Cluster_analysis) including [k-means](https://en.wikipedia.org/wiki/K-means_clustering), and [latent Dirichlet allocation](https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation) (LDA)
* [dimensionality reduction techniques](https://en.wikipedia.org/wiki/Dimensionality_reduction) such as [singular value decomposition](https://en.wikipedia.org/wiki/Singular_value_decomposition) (SVD), and [principal component analysis](https://en.wikipedia.org/wiki/Principal_component_analysis) (PCA)
* [feature extraction](https://en.wikipedia.org/wiki/Feature_extraction) and [transformation](https://en.wikipedia.org/wiki/Data_transformation_(statistics)) functions
* [optimization](https://en.wikipedia.org/wiki/Optimization_(mathematics)) algorithms such as [stochastic gradient descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent), [limited-memory BFGS](https://en.wikipedia.org/wiki/Limited-memory_BFGS) (L-BFGS)

#### GraphX

GraphX is a distributed [graph-processing](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)) framework on top of Apache Spark. Because it is based on RDDs, which are immutable, graphs are immutable and thus GraphX is unsuitable for graphs that need to be updated, let alone in a transactional manner like a [graph database](https://en.wikipedia.org/wiki/Graph_database).[[26]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-27) GraphX provides two separate APIs for implementation of massively parallel algorithms (such as [PageRank](https://en.wikipedia.org/wiki/PageRank)): a [Pregel](https://en.wikipedia.org/wiki/Graph_database#Distributed_processing) abstraction, and a more general MapReduce-style API.[[27]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-28) Unlike its predecessor Bagel, which was formally deprecated in Spark 1.6, GraphX has full support for property graphs (graphs where properties can be attached to edges and vertices).[[28]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-29) GraphX can be viewed as being the Spark in-memory version of [Apache Giraph](https://en.wikipedia.org/wiki/Apache_Giraph), which utilized Hadoop disk-based MapReduce.[[29]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-30) Like Apache Spark, GraphX initially started as a research project at UC Berkeley's AMPLab and Databricks, and was later donated to the Apache Software Foundation and the Spark project.[[30]](https://en.wikipedia.org/wiki/Apache_Spark#cite_note-31)

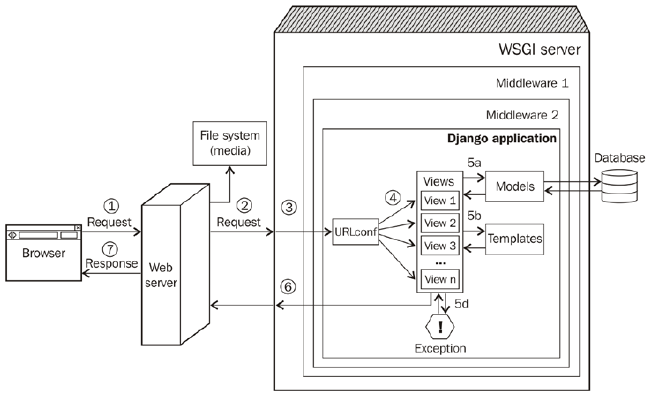
## APACHE KAFKA

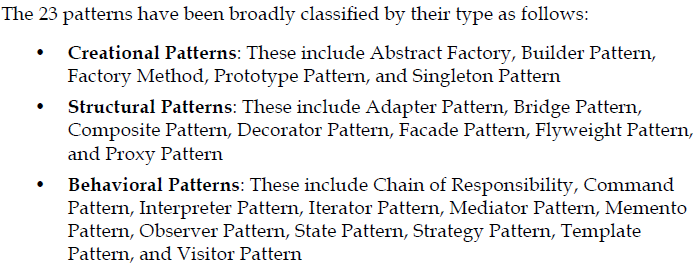


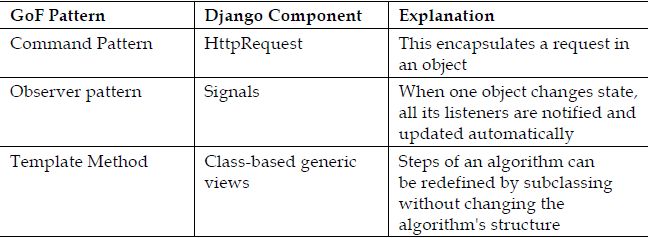


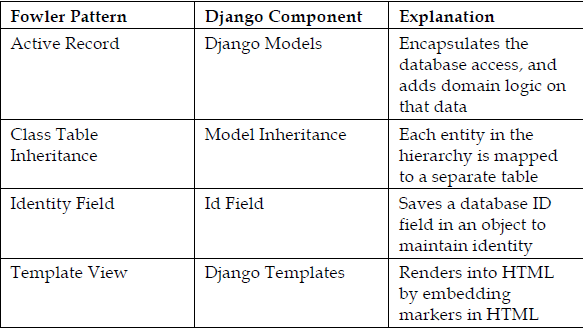
Kafka stores key-value messages that come from arbitrarily many processes called *producers*. The data can be partitioned into different "partitions" within different "topics". Within a partition, messages are strictly ordered by their offsets (the position of a message within a partition), and indexed and stored together with a timestamp. Other processes called "consumers" can read messages from partitions. For stream processing, Kafka offers the Streams API that allows writing Java applications that consume data from Kafka and write results back to Kafka. Apache Kafka also works with external stream processing systems such as [Apache Apex](https://en.wikipedia.org/wiki/Apache_Apex), [Apache Flink](https://en.wikipedia.org/wiki/Apache_Flink), [Apache Spark](https://en.wikipedia.org/wiki/Apache_Spark), and [Apache Storm](https://en.wikipedia.org/wiki/Apache_Storm). Kafka runs on a cluster of one or more servers (called brokers), and the partitions of all topics are distributed across the cluster nodes. Additionally, partitions are replicated to multiple brokers. This architecture allows Kafka to deliver massive streams of messages in a fault-tolerant fashion and has allowed it to replace some of the conventional messaging systems like [Java Message Service](https://en.wikipedia.org/wiki/Java_Message_Service) (JMS), [Advanced Message Queuing Protocol](https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol) (AMQP), etc. Since the 0.11.0.0 release, Kafka offers *transactional writes*, which provide exactly-once stream processing using the Streams API. Kafka supports two types of topics: ➊*Regular topics* can be configured with a retention time or a space bound. If there are records that are older than the specified retention time or if the space bound is exceeded for a partition, Kafka is allowed to delete old data to free storage space. By default, topics are configured with a retention time of 7 days, but it's also possible to store data indefinitely. ➋For *compacted topics*, records don't expire based on time or space bounds. Instead, Kafka treats later messages as updates to older message with the same key and guarantees never to delete the latest message per key. Users can delete messages entirely by writing a so-called tombstone message with null-value for a specific key. Four major APIs: ➊ **Producer API** Permits application to publish streams of records ➋ **Consumer API** Permits application to subscribe to topics and processes streams of records ➌ **Connector API** Executes the reusable producer and consumer APIs that can link the topics to the existing applications ➍ **Streams API** Converts the input streams to output and produces the result. The consumer and producer APIs build on top of the Kafka messaging protocol and offer a reference implementation for Kafka consumer and producer clients in Java. The underlying messaging protocol is a [binary protocol](https://en.wikipedia.org/wiki/Binary_protocol) that developers can use to write their own consumer or producer clients in any programming language. This unlocks Kafka from the [Java Virtual Machine](https://en.wikipedia.org/wiki/Java_Virtual_Machine) (JVM) eco-system. A list of available non-Java clients is maintained in the Apache Kafka wiki.

## DJANGO









**Structural patterns**: ⬩normalized models ⬩model mixins ⬩user profiles ⬩service objects

**Retrieval patterns**: ⬩property field ⬩custom model managers **View patterns**: ⬩acces controlled ⬩context enhancers ⬩services **Template patterns**: ⬩inheritance ⬩active link

## SQL

REPL (Read-Eval-Print-Loop) instructions

## Oracle PL/SQL

##### PL/SQL anonymous block

The basic unit of a PL/SQL (Procedural Language for SQL) source program is the block, which groups together related declarations and statements. A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END. These keywords divide the block into a declarative part, an executable part, and an exception-handling part. The declaration section is optional and may be used to define and initialize constants and variables. If a variable is not initialized then it defaults to [NULL](https://en.wikipedia.org/wiki/Null_(SQL)) value. The optional exception-handling part is used to handle run time errors. Only the executable part is required. A block can have a label.

**<<label>>** -- this is optional

**DECLARE**

-- this section is optional

number1 NUMBER(**2**);

number2 number1%TYPE := **17**; -- value default

text1 VARCHAR2(**12**) := 'Hello world';

text2 DATE := SYSDATE; -- current date and time

**BEGIN**

-- this section is mandatory, must contain at least one executable statement

**SELECT** street\_number

**INTO** number1

**FROM** address

**WHERE** **name** = 'INU';

**EXCEPTION**

-- this section is optional

**WHEN** OTHERS **THEN**

DBMS\_OUTPUT**.**PUT\_LINE('Error Code is ' || TO\_CHAR(sqlcode));

DBMS\_OUTPUT**.**PUT\_LINE('Error Message is ' || sqlerrm);

**END**;

The symbol := functions as an [assignment operator](https://en.wikipedia.org/wiki/Assignment_operator) to store a value in a variable.

Blocks can be nested – i.e., because a block is an executable statement, it can appear in another block wherever an executable statement is allowed. A block can be submitted to an interactive tool (such as SQL\*Plus) or embedded within an Oracle Precompiler or [OCI](https://en.wikipedia.org/wiki/Oracle_Call_Interface) program. The interactive tool or program runs the block once. The block is not stored in the database, and for that reason, it is called an anonymous block (even if it has a label).

##### Function

PL/SQL function is generally used to compute and return a single value. This returned value may be a single scalar value (such as a number, date or character string) or a single collection (such as a nested table or array). User-defined functions supplement the built-in functions provided by Oracle Corporation.

**CREATE** **OR** **REPLACE** **FUNCTION** <function\_name> [(**input**/output variable declarations)] **RETURN** return\_type

[AUTHID <**CURRENT\_USER** | **DEFINER**>] <**IS**|**AS**> -- heading part

amount number; -- declaration block

**BEGIN** -- executable part

<PL/SQL block **with** **return** **statement**>

**RETURN** <return\_value>;

[**Exception**

**none**]

**RETURN** <return\_value>;

**END**;

Pipe-lined table functions return collections[[4]](https://en.wikipedia.org/wiki/PL/SQL#cite_note-4) and take the form:

**CREATE** **OR** **REPLACE** **FUNCTION** <function\_name> [(**input**/output variable declarations)] **RETURN** return\_type

[AUTHID <**CURRENT\_USER** | **DEFINER**>] [<**AGGREGATE** | PIPELINED>] <**IS**|**USING**>

[declaration block]

**BEGIN**

<PL/SQL block **with** **return** **statement**>

PIPE **ROW** <**return** **type**>;

**RETURN**;

[**Exception**

**exception** block]

PIPE **ROW** <**return** **type**>;

**RETURN**;

**END**;

A function should only use the default IN type of parameter. The only out value from the function should be the value it returns.

##### Procedure

Like functions, procedures are named program units that can be invoked repeatedly. The primary difference is that **functions can be used in a SQL statement whereas procedures cannot**. Another difference is that the procedure can return multiple values whereas a function should only return a single value.

The procedure begins with a mandatory heading part to hold the procedure name and optionally the procedure parameter list. Next come the declarative, executable and exception-handling parts, as in the PL/SQL Anonymous Block.

**CREATE** **PROCEDURE** create\_email\_address ( -- Procedure heading part begins

name1 VARCHAR2,

name2 VARCHAR2,

company VARCHAR2,

email **OUT** VARCHAR2

) -- Procedure heading part ends

**AS**

-- Declarative part begins (optional)

error\_message VARCHAR2(**30**) := 'Email address is too long.';

**BEGIN** -- Executable part begins (mandatory)

email := name1 || '.' || name2 || '@' || company;

**EXCEPTION** -- Exception-handling part begins (optional)

**WHEN** VALUE\_ERROR **THEN**

DBMS\_OUTPUT**.**PUT\_LINE(error\_message);

**END** create\_email\_address;

The example above shows a standalone procedure - this type of procedure is created and stored in a database schema using the CREATE PROCEDURE statement. A procedure may also be created in a PL/SQL package - this is called a Package Procedure. A procedure created in a PL/SQL anonymous block is called a nested procedure. The standalone or package procedures, stored in the database, are referred to as "[stored procedures](https://en.wikipedia.org/wiki/Stored_procedure)". Procedures can have three types of parameters: IN, OUT and IN OUT.

1. An IN parameter is used as input only. An IN parameter is passed by reference, though it can be changed by the inactive program.
2. An OUT parameter is initially NULL. The program assigns the parameter value and that value is returned to the calling program.
3. An IN OUT parameter may or may not have an initial value. That initial value may or may not be modified by the called program. Any changes made to the parameter are returned to the calling program by default by copying but - with the NO-COPY hint - may be passed [by reference](https://en.wikipedia.org/wiki/Call_by_reference).

PL/SQL also supports external procedures via the Oracle database's standard ext-proc process. [[5]](https://en.wikipedia.org/wiki/PL/SQL#cite_note-5)

##### Package

Packages are groups of conceptually linked functions, procedures, variables, PL/SQL table and record TYPE statements, constants, cursors, etc. The use of packages promotes re-use of code. Packages are composed of the package specification and an optional package body. The specification is the interface to the application; it declares the types, variables, constants, exceptions, cursors, and subprograms available. The body fully defines cursors and subprograms, and so implements the specification. Two advantages of packages are:

1. Modular approach, encapsulation/hiding of business logic, security, performance improvement, re-usability. They support [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) features like function overloading and encapsulation.
2. Using package variables one can declare session level (scoped) variables since variables declared in the package specification have a session scope.

##### Trigger

A [database trigger](https://en.wikipedia.org/wiki/Database_trigger) is like a stored procedure that Oracle Database invokes automatically whenever a specified event occurs. It is a named PL/SQL unit that is stored in the database and can be invoked repeatedly. Unlike a stored procedure, you can enable and disable a trigger, but you cannot explicitly invoke it. While a trigger is enabled, the database automatically invokes it—that is, the trigger fires—whenever its triggering event occurs. While a trigger is disabled, it does not fire. You create a trigger with the CREATE TRIGGER statement. You specify the triggering event in terms of triggering statements, and the item they act on. The trigger is said to be created on or defined on the item—which is either a table, a view, a schema, or the database. You also specify the timing point, which determines whether the trigger fires before or after the triggering statement runs and whether it fires for each row that the triggering statement affects. If the trigger is created on a table or view, then the triggering event is composed of DML statements, and the trigger is called a DML trigger. If the trigger is created on a schema or the database, then the triggering event is composed of either DDL or database operation statements, and the trigger is called a system trigger. An INSTEAD OF trigger is either: A DML trigger created on a view or a system trigger defined on a CREATE statement. The database fires the INSTEAD OF trigger instead of running the triggering statement.

###### Purpose of triggers

⬩Generating derived column values automatically ⬩Enforcing referential integrity ⬩Event logging/ storing information on table access ⬩Auditing ⬩Synchronous replication of tables ⬩Security authorizations ⬩Preventing invalid transactions

##### Array handling

PL/SQL refers to [arrays](https://en.wikipedia.org/wiki/Array_data_type) as "collections". Three types: ➊ [Associative arrays](https://en.wikipedia.org/wiki/Associative_array) (Index-by tables) ➋Nested tables ➌Varrays (variable-size arrays). Programmers must specify an upper limit for varrays, but need not for index-by tables or for nested tables. The language includes several collection [methods](https://en.wikipedia.org/wiki/Method_(computer_science)) used to manipulate collection elements f. ex. FIRST, LAST, NEXT, PRIOR, EXTEND, TRIM, DELETE, etc. Index-by tables can be used to simulate associative arrays, as in this [example of a memo function for Ackermann's function in PL/SQL](https://en.wikipedia.org/wiki/Ackermann_function#cite_note-10).

###### Associative arrays (index-by tables)

Index-by tables can be indexed by numbers or strings. It parallels a [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) *map*, which comprises key-value pairs. There is only one dimension and is unbounded.

###### Nested tables

With nested tables, needs to understand what is nested. Here, a new type is created that may be composed of a number of components. That type can then be used to make a column in a table, and nested within that column are those components.

###### Varrays (variable-size arrays)

In Varrays, "variable" in "variable-size arrays" doesn't apply to the size of the array in the way you might think that it would. The size the array is declared with is in fact fixed. The number of elements in the array is variable up to the declared size. Arguably then, variable-sized arrays aren't that variable in size.

##### Cursors

A [cursor](https://en.wikipedia.org/wiki/Cursor_(databases)) is a mechanism, pointer to a private SQL area that stores information coming from a SELECT or data manipulation language (DML) statement (INSERT, UPDATE, DELETE, or MERGE). A [cursor](https://en.wikipedia.org/wiki/Cursor_(databases)) holds the rows (one or more) returned by a SQL statement. The set of rows the [cursor](https://en.wikipedia.org/wiki/Cursor_(databases)) holds is referred to as the active set.[[7]](https://en.wikipedia.org/wiki/PL/SQL#cite_note-7) A [cursor](https://en.wikipedia.org/wiki/Cursor_(databases)) can be explicit or implicit. In a FOR loop, an explicit cursor shall be used if the query will be reused, otherwise an implicit cursor is preferred. If using a cursor inside a loop, use a FETCH is recommended when needing to bulk collect or when needing dynamic SQL.

## T-SQL

**DECLARE** @var1 NVARCHAR(**30**);

**SET** @var1 = 'Some Name';

**SELECT** @var1 = Name

**FROM** Sales.Store

**WHERE** CustomerID = **100**;

## SQL CLR

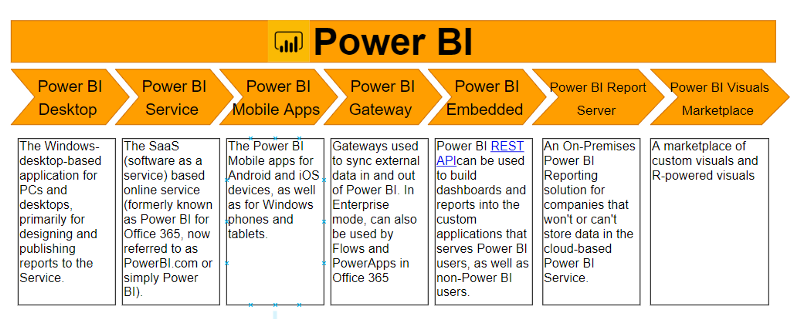
**SQL CLR** or **SQLCLR** ([SQL](https://en.wikipedia.org/wiki/SQL) [Common Language Runtime](https://en.wikipedia.org/wiki/Common_Language_Runtime)) is technology for hosting of the Microsoft .NET common language runtime engine within SQL Server. The SQLCLR allows [managed code](https://en.wikipedia.org/wiki/Managed_code) to be hosted by, and run in, the [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server) environment.

This technology, introduced in 2005, allow users for example to create the following types of managed code objects in SQL Server in .NET languages such as [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) or [VB.NET](https://en.wikipedia.org/wiki/VB.NET).

* [Stored procedures](https://en.wikipedia.org/wiki/Stored_procedure) (SPs) which are analogous to *procedures* or *void functions* in procedural languages like VB or C,
* [Triggers](https://en.wikipedia.org/wiki/Database_trigger) which are stored procedures that fire in response to [Data Manipulation Language](https://en.wikipedia.org/wiki/Data_Manipulation_Language) (DML) or [Data Definition Language](https://en.wikipedia.org/wiki/Data_Definition_Language) (DDL) events,
* [User-defined functions](https://en.wikipedia.org/wiki/User-defined_function) (UDFs) which are analogous to functions in procedural languages,
* [User-defined aggregates](https://en.wikipedia.org/w/index.php?title=User-defined_aggregate&action=edit&redlink=1) (UDAs) which allow developers to create custom aggregates that act on sets of data instead of one row at a time,
* [User-defined types](https://en.wikipedia.org/wiki/User-defined_type) (UDTs) that allow users to create simple or complex data types which can be serialized and deserialized within the database.

The SQL CLR relies on the creation, deployment, and registration of [CLI assemblies](https://en.wikipedia.org/wiki/Assembly_(CLI)), which are physically stored in managed code dynamic load libraries (DLLs). These assemblies may contain CLI namespaces, classes, functions and properties.

## POWER BI



## MONGODB

##### Terminology and Concepts

The following table presents the various SQL terminology and concepts and the corresponding MongoDB terminology and concepts.

|  |  |
| --- | --- |
| SQL Terms/Concepts | MongoDB Terms/Concepts |
| database | [database](https://docs.mongodb.com/manual/reference/glossary/#term-database) |
| table | [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection) |
| row | [document](https://docs.mongodb.com/manual/reference/glossary/#term-document) or [BSON](https://docs.mongodb.com/manual/reference/glossary/#term-bson) document |
| column | [field](https://docs.mongodb.com/manual/reference/glossary/#term-field) |
| index | [index](https://docs.mongodb.com/manual/reference/glossary/#term-index) |
| table joins | [$lookup](https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup), embedded documents |
| primary key  Specify any unique column or column combination as primary key. | [primary key](https://docs.mongodb.com/manual/reference/glossary/#term-primary-key)  In MongoDB, the primary key is automatically set to the [\_id](https://docs.mongodb.com/manual/reference/glossary/#term-id) field. |
| aggregation (e.g. group by) | aggregation pipeline  See the [SQL to Aggregation Mapping Chart](https://docs.mongodb.com/manual/reference/sql-aggregation-comparison/). |
| transactions | [transactions](https://docs.mongodb.com/manual/core/transactions/)  For many scenarios, the [denormalized data model (embedded documents and arrays)](https://docs.mongodb.com/manual/core/data-model-design/#data-modeling-embedding) will continue to be optimal for your data and use cases instead of multi-document transactions. That is, for many scenarios, modeling your data appropriately will minimize the need for multi-document transactions. |

##### Executables

The following table presents some database executables and the corresponding MongoDB executables. This table is *not* meant to be exhaustive.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | MongoDB | MySQL | Oracle | Informix | DB2 |
| Database Server | [mongod](https://docs.mongodb.com/manual/reference/program/mongod/#bin.mongod) | mysqld | oracle | IDS | DB2 Server |
| Database Client | [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) | mysql | sqlplus | DB-Access | DB2 Client |

##### Examples

The following table presents the various SQL statements and the corresponding MongoDB statements. The examples in the table assume the following conditions:

* The SQL examples assume a table named people.
* The MongoDB examples assume a collection named people that contain documents of the following prototype:

{

\_id: ObjectId("509a8fb2f3f4948bd2f983a0"),

user\_id: "abc123",

age: 55,

status: 'A'

}

###### Create and Alter

The following table presents the various SQL statements related to table-level actions and the corresponding MongoDB statements.

|  |  |
| --- | --- |
| SQL Schema Statements | MongoDB Schema Statements |
| CREATE TABLE people (  id MEDIUMINT NOT NULL  AUTO\_INCREMENT,  user\_id Varchar(30),  age Number,  status char(1),  PRIMARY KEY (id)  ) | Implicitly created on first [insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/#db.collection.insertOne) or [insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/#db.collection.insertMany) operation. The primary key \_id is automatically added if \_id field is not specified.  db.people.insertOne( {  user\_id: "abc123",  age: 55,  status: "A"  } )  However, you can also explicitly create a collection:  db.createCollection("people") |
| ALTER TABLE people  ADD join\_date DATETIME | Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.  However, at the document level, [updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany) operations can add fields to existing documents using the [$set](https://docs.mongodb.com/manual/reference/operator/update/set/#up._S_set) operator.  db.people.updateMany(  { },  { $set: { join\_date: new Date() } }  ) |
| ALTER TABLE people  DROP COLUMN join\_date | Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.  However, at the document level, [updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany) operations can remove fields from documents using the [$unset](https://docs.mongodb.com/manual/reference/operator/update/unset/#up._S_unset) operator.  db.people.updateMany(  { },  { $unset: { "join\_date": "" } }  ) |
| CREATE INDEX idx\_user\_id\_asc  ON people(user\_id | db.people.createIndex( { user\_id: 1 } ) |
| CREATE INDEX  idx\_user\_id\_asc\_age\_desc  ON people(user\_id, age DESC) | db.people.createIndex( { user\_id: 1, age: -1 } ) |
| DROP TABLE people | db.people.drop() |

For more information on the methods and operators used, see:

|  |  |  |
| --- | --- | --- |
| [db.collection.insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/#db.collection.insertOne)  [db.collection.insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/#db.collection.insertMany)  [db.createCollection()](https://docs.mongodb.com/manual/reference/method/db.createCollection/#db.createCollection) | [db.collection.updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany)  [db.collection.createIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/#db.collection.createIndex)  [db.collection.drop()](https://docs.mongodb.com/manual/reference/method/db.collection.drop/#db.collection.drop) | [$set](https://docs.mongodb.com/manual/reference/operator/update/set/#up._S_set)  [$unset](https://docs.mongodb.com/manual/reference/operator/update/unset/#up._S_unset) |

See also: ⬩[Databases and Collections](https://docs.mongodb.com/manual/core/databases-and-collections/) ⬩[Documents](https://docs.mongodb.com/manual/core/document/) ⬩[Indexes](https://docs.mongodb.com/manual/indexes/) ⬩[Data Modeling Concepts](https://docs.mongodb.com/manual/core/data-models/).

###### Insert

The following table presents the various SQL statements related to inserting records into tables and the corresponding MongoDB statements.

|  |  |
| --- | --- |
| SQL INSERT Statements | MongoDB insertOne() Statements |
| INSERT INTO people(user\_id,  age,  status)  VALUES ("bcd001",  45,  "A") | db.people.insertOne(  { user\_id: "bcd001", age: 45, status: "A" }  ) |

For more information, see [db.collection.insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/#db.collection.insertOne).

See also ⬩[Insert Documents](https://docs.mongodb.com/manual/tutorial/insert-documents/) ⬩[db.collection.insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/#db.collection.insertMany) ⬩[Databases and Collections](https://docs.mongodb.com/manual/core/databases-and-collections/) ⬩[Documents](https://docs.mongodb.com/manual/core/document/)

###### Select

The following table presents the various SQL statements related to reading records from tables and the corresponding MongoDB statements. **Note:** The [find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find) method always includes the \_id field in the returned documents unless specifically excluded through [projection](https://docs.mongodb.com/manual/tutorial/project-fields-from-query-results/#projection). Some of the SQL queries below may include an \_id field to reflect this, even if the field is not included in the corresponding [find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find) query.

|  |  |
| --- | --- |
| SQL SELECT Statements | MongoDB find() Statements |
| SELECT \*  FROM people | db.people.find() |
| SELECT id,  user\_id,  status  FROM people | db.people.find(  { },  { user\_id: 1, status: 1 }  ) |
| SELECT user\_id, status  FROM people | db.people.find(  { },  { user\_id: 1, status: 1, \_id: 0 }  ) |
| SELECT \*  FROM people  WHERE status = "A" | db.people.find(  { status: "A" }  ) |
| SELECT user\_id, status  FROM people  WHERE status = "A" | db.people.find(  { status: "A" },  { user\_id: 1, status: 1, \_id: 0 }  ) |
| SELECT \*  FROM people  WHERE status != "A" | db.people.find(  { status: { $ne: "A" } }  ) |
| SELECT \*  FROM people  WHERE status = "A"  AND age = 50 | db.people.find(  { status: "A",  age: 50 }  ) |
| SELECT \*  FROM people  WHERE status = "A"  OR age = 50 | db.people.find(  { $or: [ { status: "A" } , { age: 50 } ] }  ) |
| SELECT \*  FROM people  WHERE age > 25 | db.people.find(  { age: { $gt: 25 } }  ) |
| SELECT \*  FROM people  WHERE age < 25 | db.people.find(  { age: { $lt: 25 } }  ) |
| SELECT \*  FROM people  WHERE age > 25  AND age <= 50 | db.people.find(  { age: { $gt: 25, $lte: 50 } }  ) |
| SELECT \*  FROM people  WHERE user\_id like "%bc%" | db.people.find( { user\_id: /bc/ } )  -or-  db.people.find( { user\_id: { $regex: /bc/ } } ) |
| SELECT \*  FROM people  WHERE user\_id like "bc%" | db.people.find( { user\_id: /^bc/ } )  -or-  db.people.find( { user\_id: { $regex: /^bc/ } } ) |
| SELECT \*  FROM people  WHERE status = "A"  ORDER BY user\_id ASC | db.people.find( { status: "A" } ).sort( { user\_id: 1 } ) |
| SELECT \*  FROM people  WHERE status = "A"  ORDER BY user\_id DESC | db.people.find( { status: "A" } ).sort( { user\_id: -1 } ) |
| SELECT COUNT(\*)  FROM people | db.people.count()  *or*  db.people.find().count() |
| SELECT COUNT(user\_id)  FROM people | db.people.count( { user\_id: { $exists: true } } )  *or*  db.people.find( { user\_id: { $exists: true } } ).count() |
| SELECT COUNT(\*)  FROM people  WHERE age > 30 | db.people.count( { age: { $gt: 30 } } )  *or*  db.people.find( { age: { $gt: 30 } } ).count() |
| SELECT DISTINCT(status)  FROM people | db.people.aggregate( [ { $group : { \_id : "$status" } } ] )  or, for distinct value sets that do not exceed the [BSON size limit](https://docs.mongodb.com/manual/reference/limits/#limit-bson-document-size)  db.people.distinct( "status" ) |
| SELECT \*  FROM people  LIMIT 1 | db.people.findOne()  *or*  db.people.find().limit(1) |
| SELECT \*  FROM people  LIMIT 5  SKIP 10 | db.people.find().limit(5).skip(10) |
| EXPLAIN SELECT \*  FROM people  WHERE status = "A" | db.people.find( { status: "A" } ).explain() |

For more information on the methods and operators used, see

|  |  |
| --- | --- |
| [db.collection.find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/#db.collection.find)  [db.collection.distinct()](https://docs.mongodb.com/manual/reference/method/db.collection.distinct/#db.collection.distinct)  [db.collection.findOne()](https://docs.mongodb.com/manual/reference/method/db.collection.findOne/#db.collection.findOne)  [limit()](https://docs.mongodb.com/manual/reference/method/cursor.limit/#cursor.limit)  [skip()](https://docs.mongodb.com/manual/reference/method/cursor.skip/#cursor.skip)  [explain()](https://docs.mongodb.com/manual/reference/method/cursor.explain/#cursor.explain)  [sort()](https://docs.mongodb.com/manual/reference/method/cursor.sort/#cursor.sort)  [count()](https://docs.mongodb.com/manual/reference/method/cursor.count/#cursor.count) | [$ne](https://docs.mongodb.com/manual/reference/operator/query/ne/#op._S_ne)  [$and](https://docs.mongodb.com/manual/reference/operator/query/and/#op._S_and)  [$or](https://docs.mongodb.com/manual/reference/operator/query/or/#op._S_or)  [$gt](https://docs.mongodb.com/manual/reference/operator/query/gt/#op._S_gt)  [$lt](https://docs.mongodb.com/manual/reference/operator/query/lt/#op._S_lt)  [$exists](https://docs.mongodb.com/manual/reference/operator/query/exists/#op._S_exists)  [$lte](https://docs.mongodb.com/manual/reference/operator/query/lte/#op._S_lte)  [$regex](https://docs.mongodb.com/manual/reference/operator/query/regex/#op._S_regex) |

See also: ⬩[Query Documents](https://docs.mongodb.com/manual/tutorial/query-documents/) ⬩[Query and Projection Operators](https://docs.mongodb.com/manual/reference/operator/query/) ⬩[mongo Shell Methods](https://docs.mongodb.com/manual/reference/method/)

###### Update Records

The following table presents the various SQL statements related to updating existing records in tables and the corresponding MongoDB statements.

|  |  |
| --- | --- |
| SQL Update Statements | MongoDB updateMany() Statements |
| UPDATE people  SET status = "C"  WHERE age > 25 | db.people.updateMany(  { age: { $gt: 25 } },  { $set: { status: "C" } }  ) |
| UPDATE people  SET age = age + 3  WHERE status = "A" | db.people.updateMany(  { status: "A" } ,  { $inc: { age: 3 } }  ) |

For more information on the method and operators used in the examples:

⬩[db.collection.updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany) ⬩[$gt](https://docs.mongodb.com/manual/reference/operator/query/gt/#op._S_gt) ⬩[$set](https://docs.mongodb.com/manual/reference/operator/update/set/#up._S_set) ⬩[$inc](https://docs.mongodb.com/manual/reference/operator/update/inc/#up._S_inc) See also: ⬩[Update Documents](https://docs.mongodb.com/manual/tutorial/update-documents/) ⬩[Update Operators](https://docs.mongodb.com/manual/reference/operator/update/) ⬩[db.collection.updateOne()](https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne) ⬩[db.collection.replaceOne()](https://docs.mongodb.com/manual/reference/method/db.collection.replaceOne/#db.collection.replaceOne)

###### Delete Records

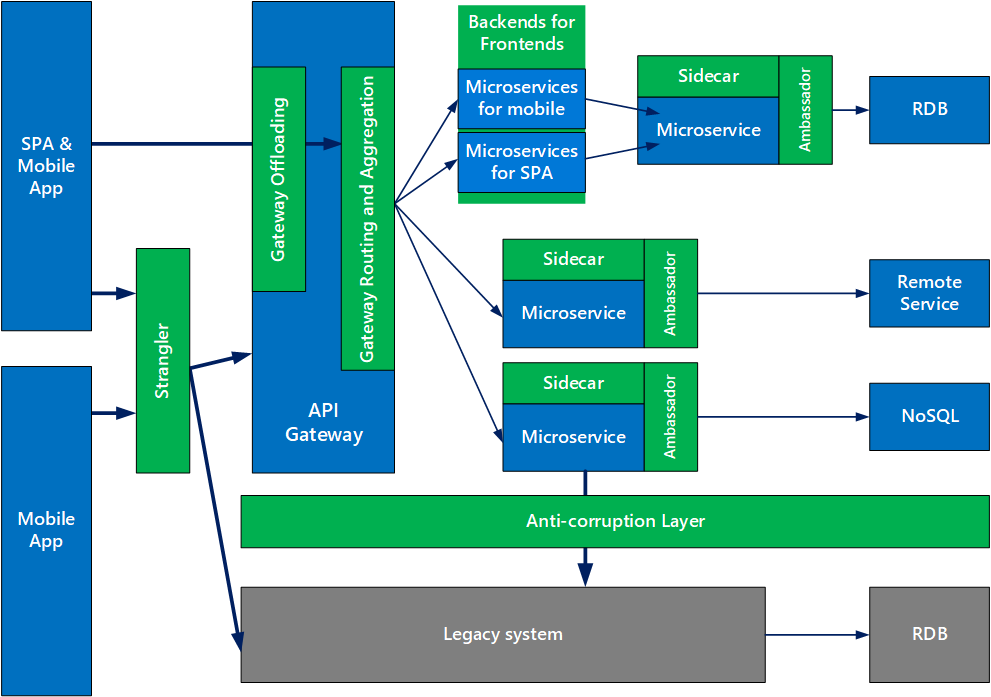
The following table presents the various SQL statements related to deleting records from tables and the corresponding MongoDB statements.

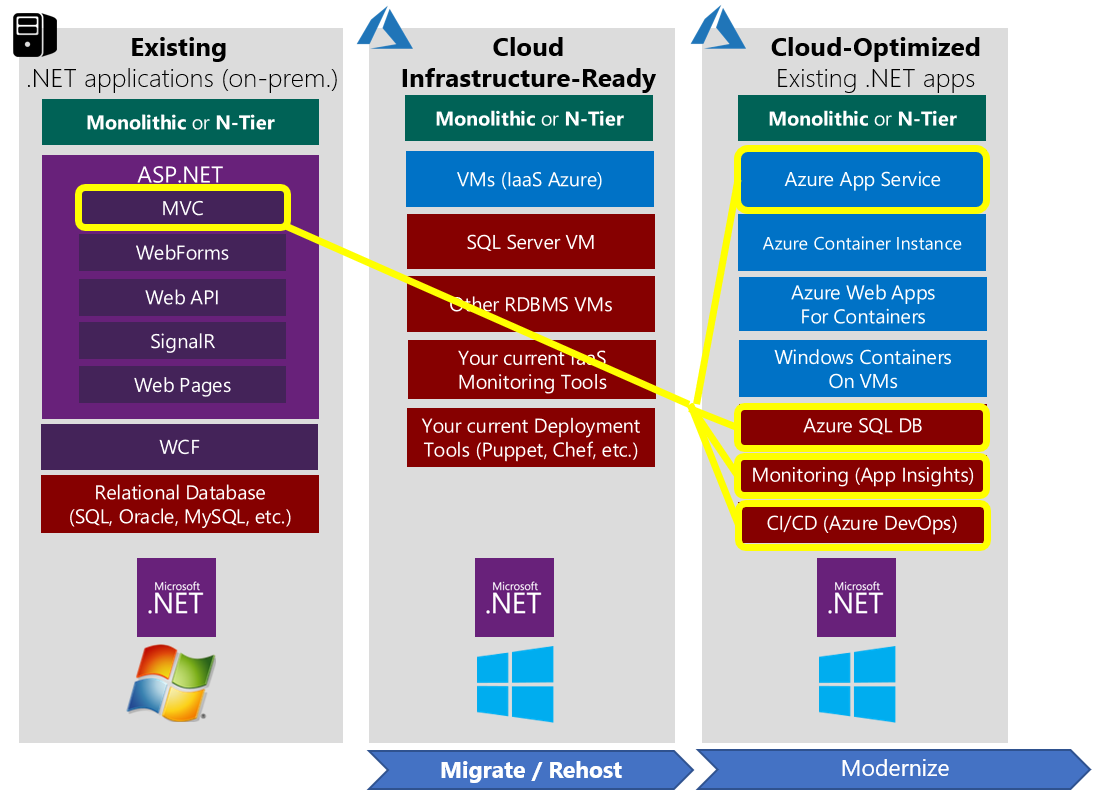
|  |  |
| --- | --- |
| SQL Delete Statements | MongoDB deleteMany() Statements |
| DELETE FROM people  WHERE status = "D" | db.people.deleteMany( { status: "D" } ) |
| DELETE FROM people | db.people.deleteMany({}) |

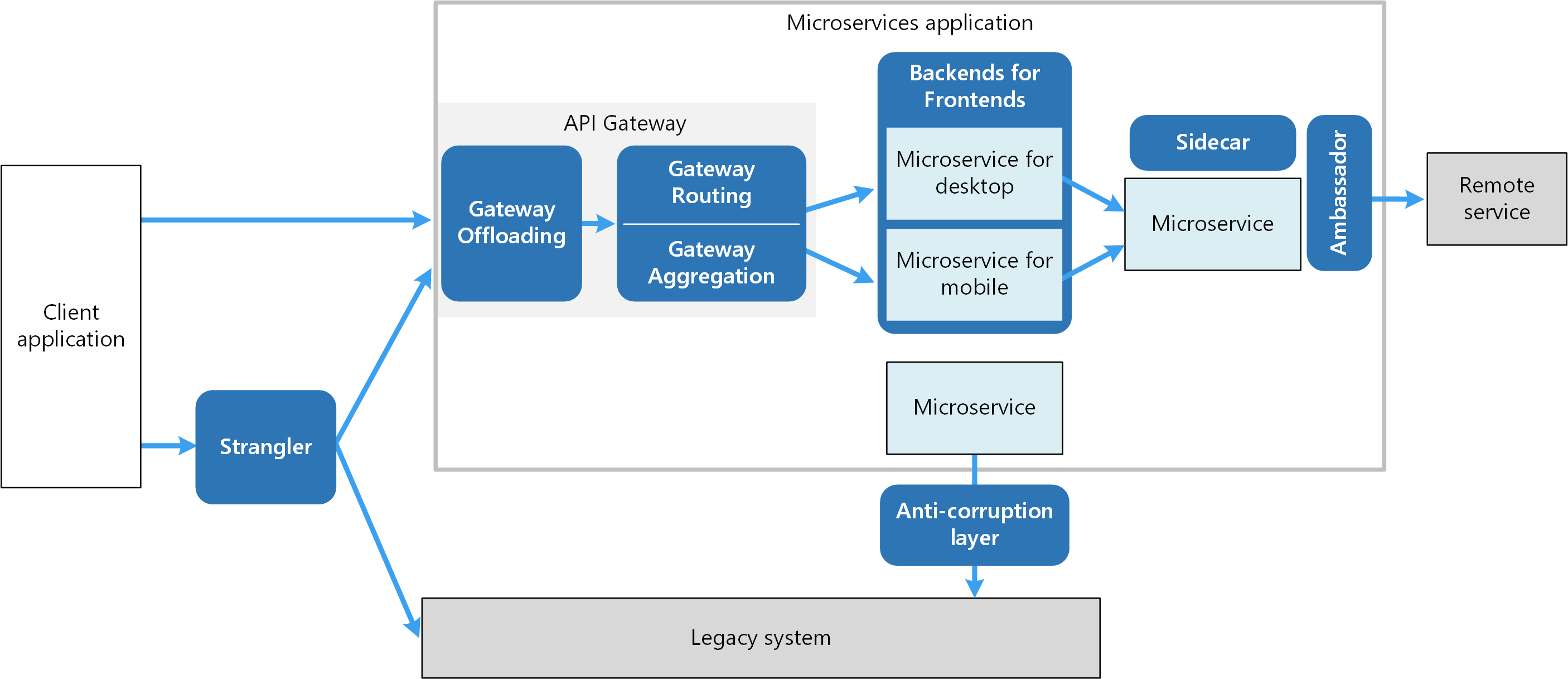
For more information, see [db.collection.deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/#db.collection.deleteMany). See also: ⬩[Delete Documents](https://docs.mongodb.com/manual/tutorial/remove-documents/) ⬩

[db.collection.deleteOne()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/#db.collection.deleteOne)

## Microservices







* ⬩[**Ambassador**](https://docs.microsoft.com/azure/architecture/patterns/ambassador) to offload common client connectivity tasks such as monitoring, logging, routing, and security (such as TLS) in a language agnostic way ⬩[**Anti-corruption layer**](https://docs.microsoft.com/azure/architecture/patterns/anti-corruption-layer) implements a façade between new and legacy applications, to ensure that the design of a new application is not limited by dependencies on legacy systems ⬩[**Backends for Front-ends**](https://docs.microsoft.com/azure/architecture/patterns/backends-for-frontends) creates separate backend services for different types of clients (desktop, mobile) ⇨ A single backend service doesn’t need to handle the conflicting requirements of various client types. This pattern can help keep each microservice simple, by separating client-specific concerns ⬩[**Bulkhead**](https://docs.microsoft.com/azure/architecture/patterns/bulkhead) isolates critical resources (connection pool, memory, CPU) for each workload or service. A single workload (or service) can’t consume all of the resources, starving others. This pattern increases the resiliency of the system by preventing cascading failures caused by one service ⬩[**Gateway Aggregation**](https://docs.microsoft.com/azure/architecture/patterns/gateway-aggregation) aggregates requests to multiple individual microservices into a single request, reducing chattiness between consumers and services ⬩[**Gateway Offloading**](https://docs.microsoft.com/azure/architecture/patterns/gateway-offloading) enables each microservice to offload shared service functionality, such as the use of SSL certificates, to an API gateway ⬩[**Gateway Routing**](https://docs.microsoft.com/azure/architecture/patterns/gateway-routing) routes requests to multiple microservices using a single endpoint, so that consumers don't need to manage many separate endpoints ⬩[**Sidecar**](https://docs.microsoft.com/azure/architecture/patterns/sidecar) deploys helper components of an application as a separate container or process to provide isolation and encapsulation ⬩[**Strangler**](https://docs.microsoft.com/azure/architecture/patterns/strangler) supports incremental migration by gradually replacing specific pieces of functionality with new services.

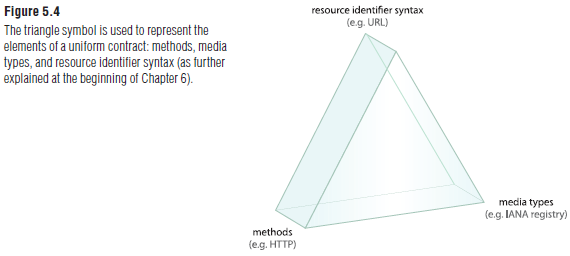
#### API

#### 12 FACTORS

Cloud-friendly applications embrace elastic scalability, ephemeral filesystems, statelessness, and treating everything as a service ⇨ Can scale and deploy rapidly ⬩**Codebase** One codebase tracked in revision control, many deploys ⬩**Dependencies**. Explicitly declare and isolate dependencies ⬩**Configuration**: Store configuration in the environment ⬩**Backing Services**: Treat backing services as attached resources ⬩**Build, release, run:** Separate build and run stages ⬩**Processes**: Execute app as one or more stateless processes ⬩**Port binding:** Export services via port binding ⬩**Concurrency**: Scale out via the process model

⬩**Disposability**: Maximize robustness with fast startup and graceful shutdown ⬩**Dev/prod parity:** Keep development, staging, and production as similar as possible ⬩**Logs**: Treat logs as event streams ⬩**Admin processes**: Run admin/management tasks as one-off processes

#### REST

REST constraints are design rules that are applied to establish the distinct characteristics

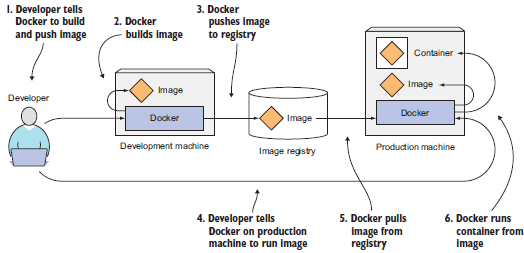
of the REST architectural style. Each constraint is a pre-determined design decision that can have both positive and negative impacts. The intent is for the positives of each constraint to balance out the negatives to produce an overall architecture that resembles the best features of the Web. Formal REST constraints: ⬩**Client-Server** {393} requires that a service offer one or more capabilities and listen for requests on these capabilities. A consumer invokes a capability by sending the corresponding request message, and the service either rejects the request or performs the requested task before sending a response message back to the consumer. Exceptions that prevent the task from proceeding are raised back to the consumer, and the consumer is responsible for taking corrective action ⬩**Stateless** {395} The communication between service consumer (client) and service (server) must be stateless between requests. This means that each request from a service consumer should contain all the necessary information for the service to understand the meaning of the request, and all session state data should then be returned to the service consumer at the end of each request. Statelessness is one of the primary influences over service contract design in REST-style architecture. It imposes significant restrictions on the kinds of communication allowed between services and their consumers in order to achieve its design goals. The application of the Cache {398} and Layered System {404} constraints helps to compensate for limitations resulting from Stateless {395} ⬩**Cache** {398} Response messages from the service to its consumers are explicitly labeled as cacheable or non-cacheable. This way, the service, the consumer, or one of the intermediary middleware components can cache the response for reuse in later requests. The Cache {398} constraint builds upon Client-Server {393} and Stateless {395} with a requirement that responses are implicitly or explicitly labeled as cacheable or noncacheable. Requests are passed through a cache component, which may reuse previous responses to partially or completely eliminate some interactions over the network. This form of elimination can improve efficiency and scalability, and can further improve user-perceived performance by reducing the average latency during a series of interactions. However, a common reason for incorporating caching as a native part of a REST architecture is as a counterbalance to some of the negative impacts of applying the Stateless {395} constraint ⬩**Interface/Uniform Contract** {400} The Interface {400} constraint (also known as “Uniform Interface”) states that all services and service consumers within a REST-compliant architecture must share a single, overarching technical interface. As the primary constraint that distinguishes REST from other architecture types, Interface {400} is generally applied using the methods and media types provided by HTTP and other Internet standards ⬩**Code-On-Demand** {407} This optional constraint is primarily intended to allow logic within clients (such as Web browsers) to be updated independently from server-side logic. Code-On-Demand {407} typically relies on the use of Web-based technologies, such as Web browser plug-ins, applets, or client-side scripting languages (i.e. JavaScript). Code-On-Demand {407} can further be applied to services and service consumers. For example, a service can be designed to dynamically defer portions of logic to service consumer programs. For example, this type of functionality can be used in support of Stateless {395}, which dictates when session state should be deferred back to the service consumer. Code-On-Demand {407} can also build upon this by further deferring the processing effort. This approach may be justifiable when service logic can be executed by the consumer more efficiently or effectively

#### DOCKER

Docker = platform for packaging, distributing, and running applications. Allows you to package your application together with its whole environment (libraries that the app requires or files usually available on the filesystem of an installed operating system). Docker makes it possible to transfer this package to a central repository from which it can then be transferred to any computer running Docker and executed there. 3 concepts: ➊**Images:** Docker-based container image is something you package your application and its environment into. It contains the filesystem that will be available to the application and other metadata, such as the path to the executable that should be executed when the image is run. ➋**Registries**: A Docker Registry is a repository that stores your Docker images and facilitates easy sharing of those images between different people and computers. When you build your image, you can either run it on the computer you’ve built it on, or you can push (upload) the image to a registry and then pull (download) it on another computer and run it there. Certain registries are public, allowing anyone to pull images from it, while others are private, only accessible

to certain people or machines. ➌**Containers**: Docker-based container is a regular Linux container created from a Docker-based container image. A running container is a process running on the host running Docker, but it’s completely isolated from both the host and all

other processes running on it. The process is also resource-constrained, meaning it can only access and use the amount of resources (CPU, RAM, and so on) that are allocated to it.

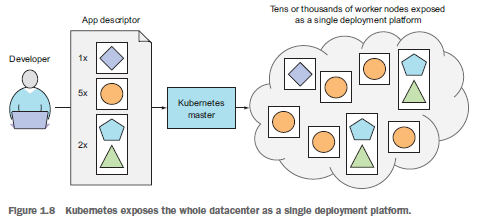


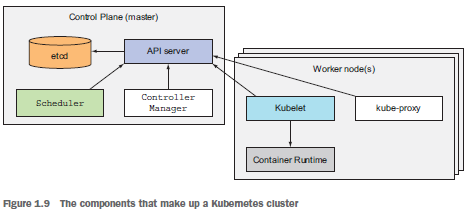
#### KUBERNETES

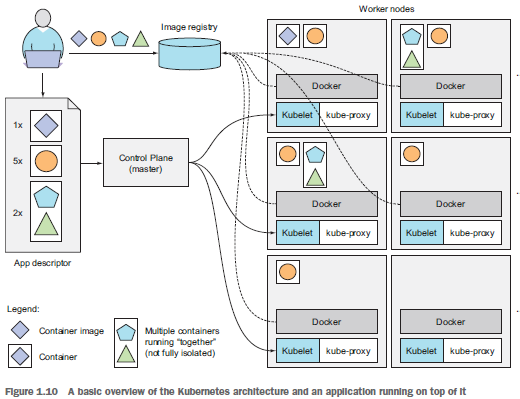
Kubernetes is a software system that allows you to easily deploy and manage containerized

applications on top of it. It relies on the features of Linux containers to run heterogeneous

applications without having to know any internal details of these applications and without having to manually deploy these applications on each host. Because these apps run in containers, they don’t affect other apps running on the same server, which is critical when you run applications for completely different organizations on the same hardware. Kubernetes enables you to run your software applications on thousands of computer nodes as if all those nodes were a single, enormous computer. It abstracts away the underlying infrastructure and, by doing so, simplifies development, deployment, and management for both development and the operations teams. Deploying applications through Kubernetes is always the same, whether your cluster contains only a couple of nodes or thousands of them. The size of the cluster makes no difference at all. Additional cluster nodes simply represent an additional amount of resources available to deployed apps.







## PYTHON

#### LUIGI

#### DASK

#### PANDAS

#### SQL ALCHEMY

## R

## JAVASCRIPT

## SCIKIT-LEARN

## TENSORFLOW

## PYTORCH

## MACHINE LEARNING

**⬩Regression questions**: ‘How much’ and ‘how many’. For example, how much will my car be worth in two years? ⬩**Classification questions**: such as ‘Type of object’. For example, what to class does this object belong? ⬩**Clustering or grouping questions**. For example, what are the different clusters for this particular set of objects? ⬩**Abnormality detection questions**. For example, is this object abnormal based on what is defined as normal?

#### Regression

#### Boost

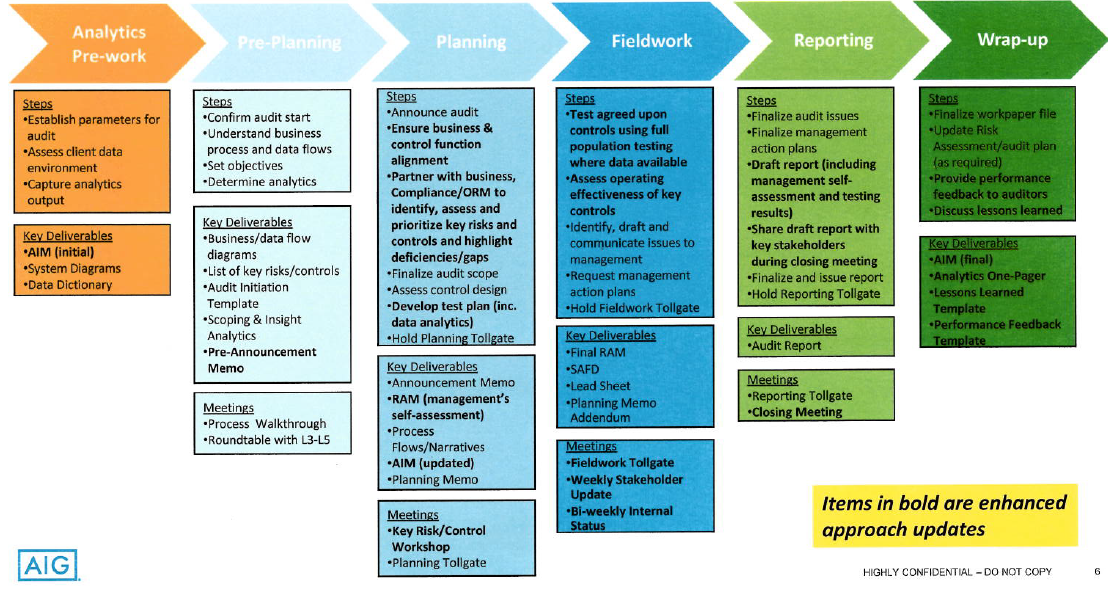
#### Deep Learning

## Feature Engineering

➊**Resampling Imbalanced Data**: Balanced Accuracy, Precision-Recall Curves, F1-score, SMOTE (Synthetic Minority Oversampling Technique) ➋**Creating New Features**: DFS (Deep Feature Synthesis) ➌**Handling Missing Values**: Iterative Imputer (R imputation packages missForest, mi, mice, etc.) ➍**Outlier Detection**: Isolation Forest

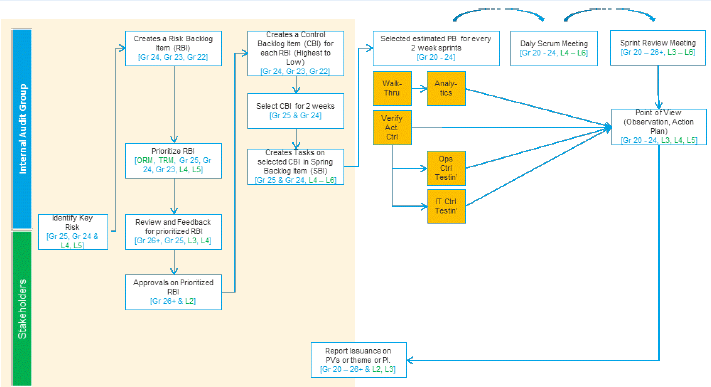
# EXHIBIT

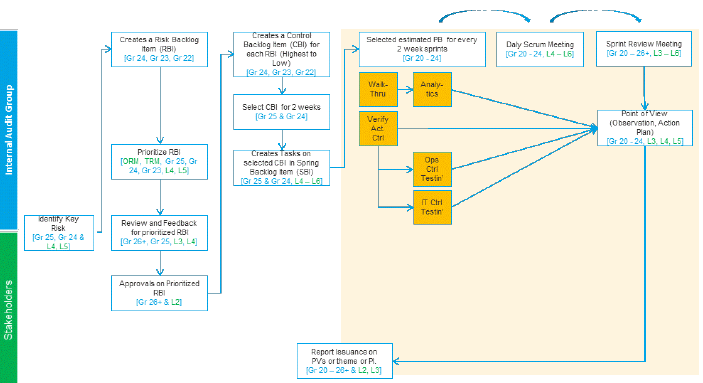
## Audit Lifecycle



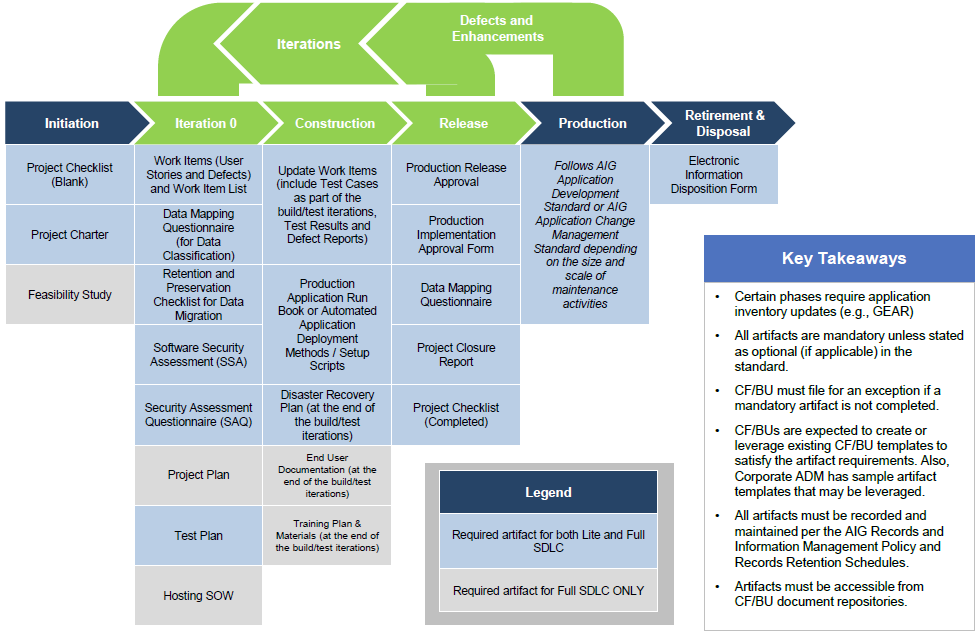
## AGILE AUDIT

Agile concepts: •Audit Increment planning” build a backlog of key risks and controls •Execute each sprint (2 week intervals) •After each sprint have a sprint review meeting with L4 to discuss results and initiate. After each sprint have tollgate to discuss stopping or continuing with audit •After each sprint and before next Sprint have Lessons learned session to discuss went well in sprint and what needs enhancements from next sprint •Holding daily scrum meetings (10 minutes) to discuss progress from yesterday, plan for current day and if any escalation is required

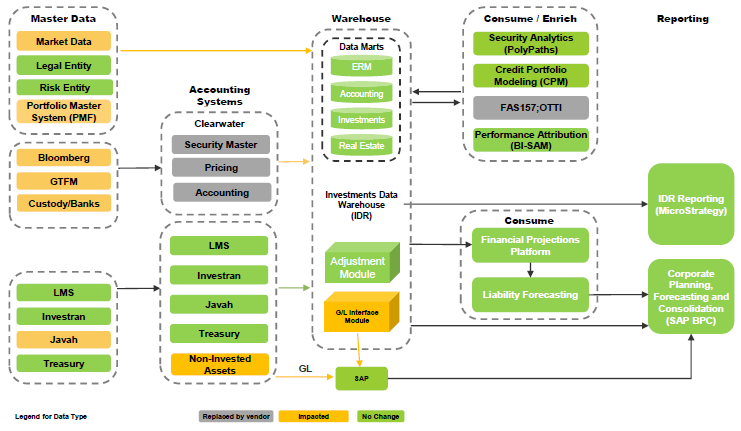




## APPLICATION DEVELOPMENT AGILE



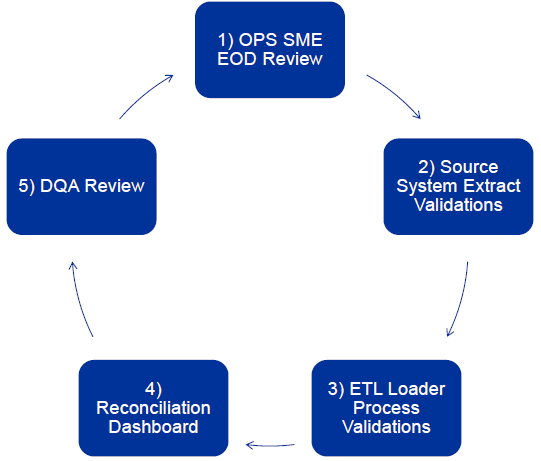
## NOVA Target State

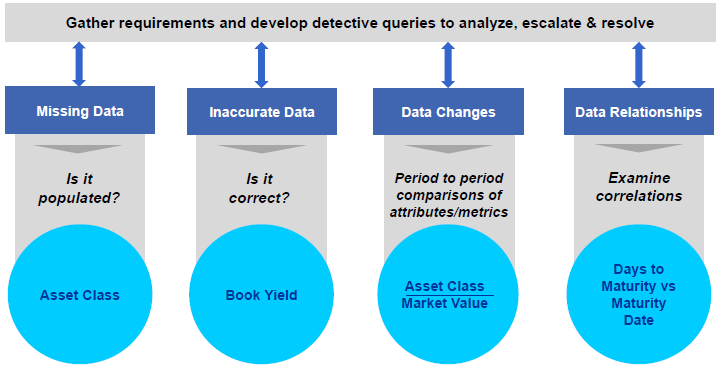


## IDR Data Governance



## IDR Data Quality Lifecycle





## IDR State Architecture

