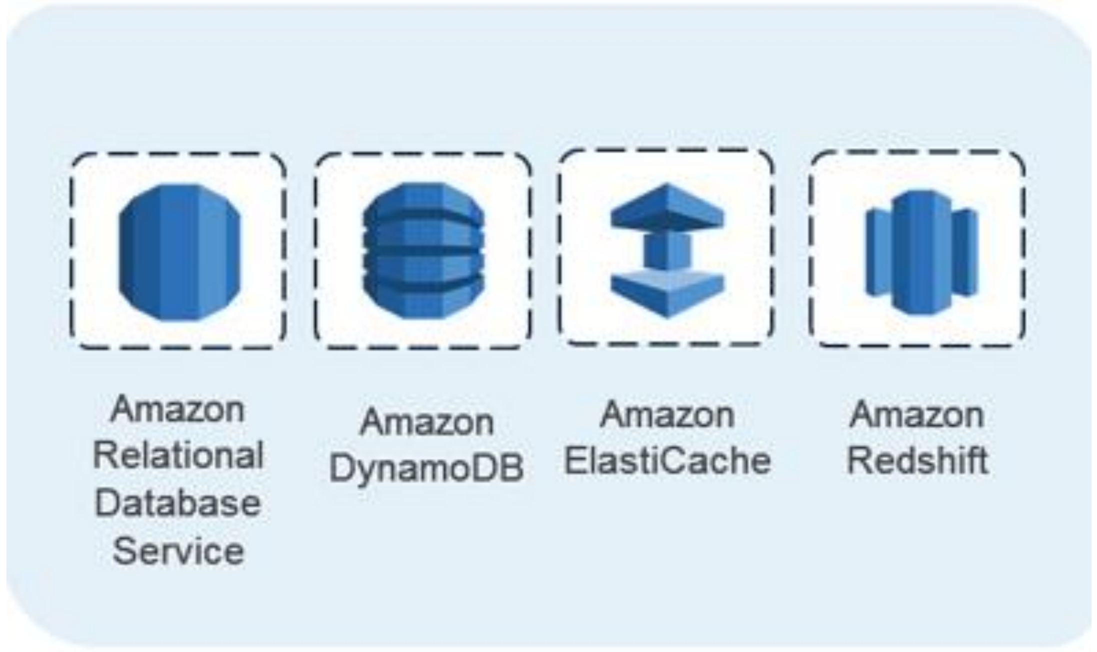
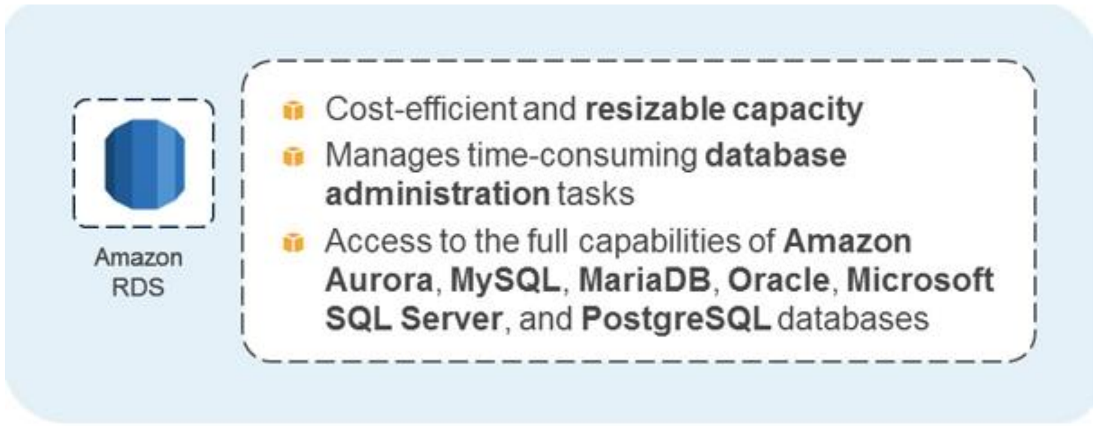
Cloud Databases

AWS Databases



AWS RDS

- Amazon RDS provide full capabilities of a familiar MySQL, MariaDB, Oracle, Microsoft SQL Server, or PostgreSQL database.
- Amazon RDS for MySQL provides two distinct but complementary replication features:
  - Multi-AZ deployments and
  - read replicas that can be used in conjunction to gain enhanced database availability, protect your latest database updates against unplanned outages, and scale beyond the capacity constraints of a single DB instance for read-heavy database workloads.
- Amazon Aurora is a MySQL compatible relational database engine that is part of Amazon RDS.



- Cost-efficient and **resizable capacity**
- Manages time-consuming **database administration** tasks
- Access to the full capabilities of **Amazon Aurora**, **MySQL**, **MariaDB**, **Oracle**, **Microsoft SQL Server**, and **PostgreSQL** databases

Amazon RDS Database Engines

Amazon Aurora | PostgreSQL | MySQL | MariaDB | ORACLE | Microsoft SQL Server

Introduction to Amazon Relational Database Service

Instance Type	vCPU	Memory (GiB)	PIOPS-Optimized	Network Performance
Standard				
db.m4.large	2	8	Yes	Moderate
db.m4.xlarge	4	16	Yes	High
db.m4.2xlarge	8	32	Yes	High
db.m4.4xlarge	16	64	Yes	High
db.m4.10xlarge	40	160	Yes	10 Gigabit
db.m3.medium	1	3.75	-	Moderate
db.m3.large	2	7.5	-	Moderate
db.m3.xlarge	4	15	Yes	High
db.m3.2xlarge	8	30	Yes	High
Memory Optimized				
db.r3.large	2	15	-	Moderate
db.r3.xlarge	4	30.5	Yes	Moderate
db.r3.2xlarge	8	61	Yes	High
db.r3.4xlarge	16	122	Yes	High
db.r3.8xlarge	32	244	-	10 Gigabit
Micro instances				
db.t2.micro	1	1	-	Low
db.t2.small	1	2	-	Low
db.t2.medium	2	4	-	Moderate
db.t2.large	2	8	-	Moderate

Key Features

- Lower Administrative Burden
  - Easy to Use
  - Monitoring and Metrics
  - Automatic Software Patching
- Performance
  - General Purpose (SSD) Storage
  - Provisioned IOPS (SSD) Storage
- Scalability
  - Push-button Compute Scaling
  - Easy Storage Scaling
- Availability and Durability
  - Automated Backups
  - Database Snapshots
  - Multi-AZ Deployments
  - Automatic Host Replacement
- Security
  - Encryption at Rest and in Transit
  - Network Isolation
  - Resource-level Permissions
- Manageability
  - Monitoring and Metrics
  - Event Notifications
  - Configuration Governance
- Cost-effectiveness
  - Pay Only for What You Use
  - Reserved Instances
  - Stop and Start

When to Use

- Yes
  - Complex transactions
  - Complex queries
  - Medium to high query/write rates:
    - Up to 30K IOPS (15K reads + 15K writes)
    - Workload can fit in a single node
    - High durability
- No
  - Massive read/write rates (e.g., 150K write/second)
  - Data size or throughput demands shading (e.g., 10s or 100s TBs)
  - Simple Get/Put and queries that a NoSQL database can handle
  - RDMS customization

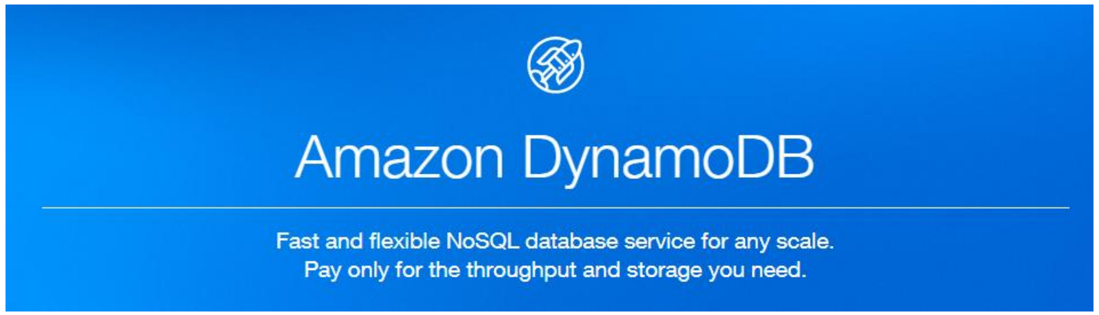
Security

- Control of Amazon RDS DB instance access via security groups
- AWS RDS encrypted instances are available
- Use IAM to control which RDS operations each individual user has permission to call
- Encrypt connections between application and DB instances using SSL
- Transparent Data Encryption (TDE) is supported when SQL Server and Oracle are used
- Transmit notifications about relevant events on RDS instances

Tutorials

- Creating an Aurora DB Instance on an Aurora Cluster and Connecting to a Database
- Creating a MariaDB DB Instance and Connecting to a Database
- Creating a Microsoft SQL Server DB Instance and Connecting to a Database
- Creating a MySQL DB Instance and Connecting to a Database
- Creating an Oracle DB Instance and Connecting to a Database
- Creating a PostgreSQL DB Instance and Connecting to a Database

AWS DynamoDB

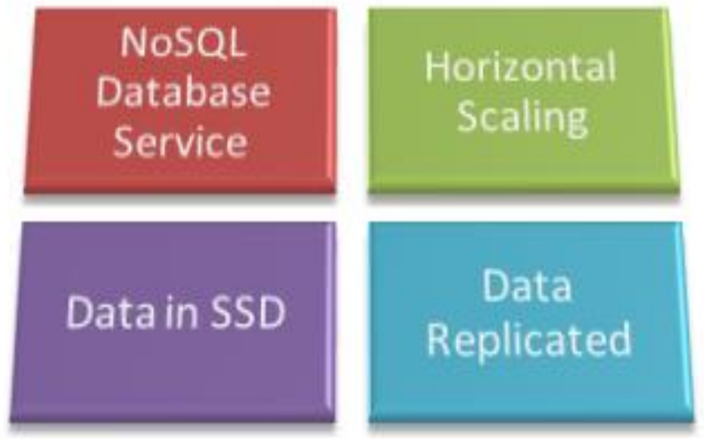


Amazon DynamoDB

Fast and flexible NoSQL database service for any scale.
Pay only for the throughput and storage you need.

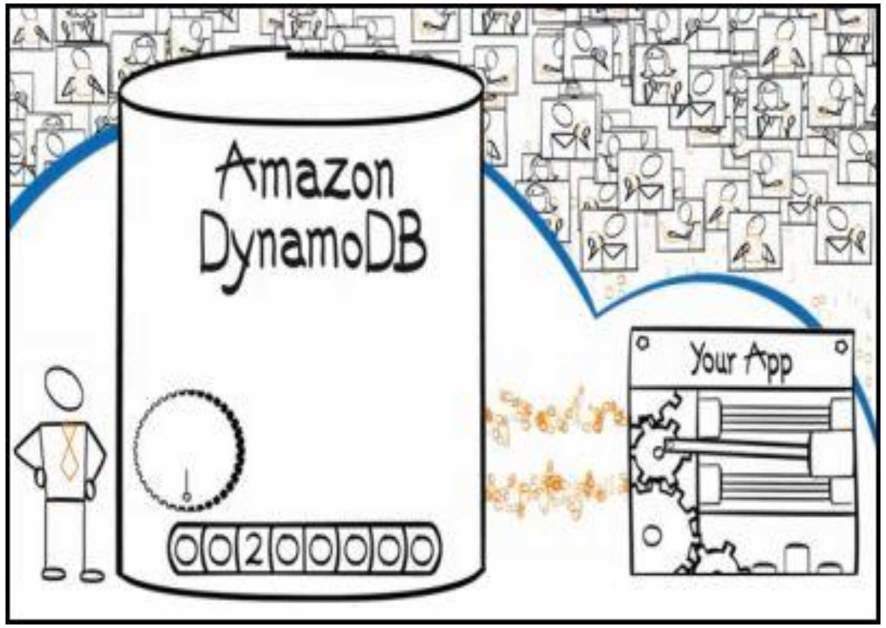
Introduction to DynamoDB

- Fully managed NoSQL database service by Amazon
- Database type: Key-value stores
- Designed to address the core problems of database management, performance, scalability, and reliability



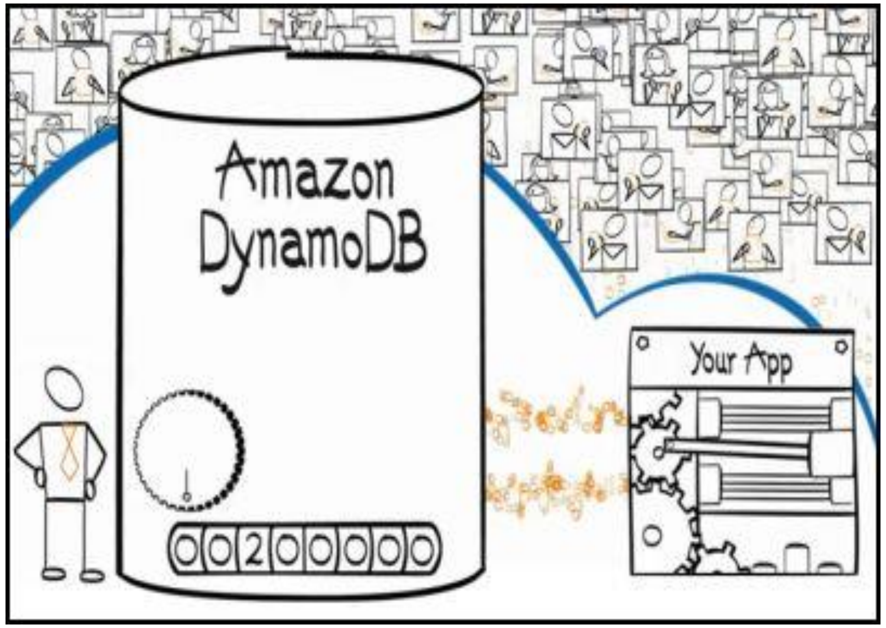
Features

- Scalable
  - Provisioned Throughput
  - Fully Distributed, Shared Nothing Architecture
- Fast Performance
  - Average service-side latencies < 10 ms
  - The service runs on Solid State Disks - consistent, fast latencies at any scale
- Easy Administration and Cost Effective
  - a fully managed service by Amazon
- Fault-tolerant
  - Synchronous replication across multiple zones in a region

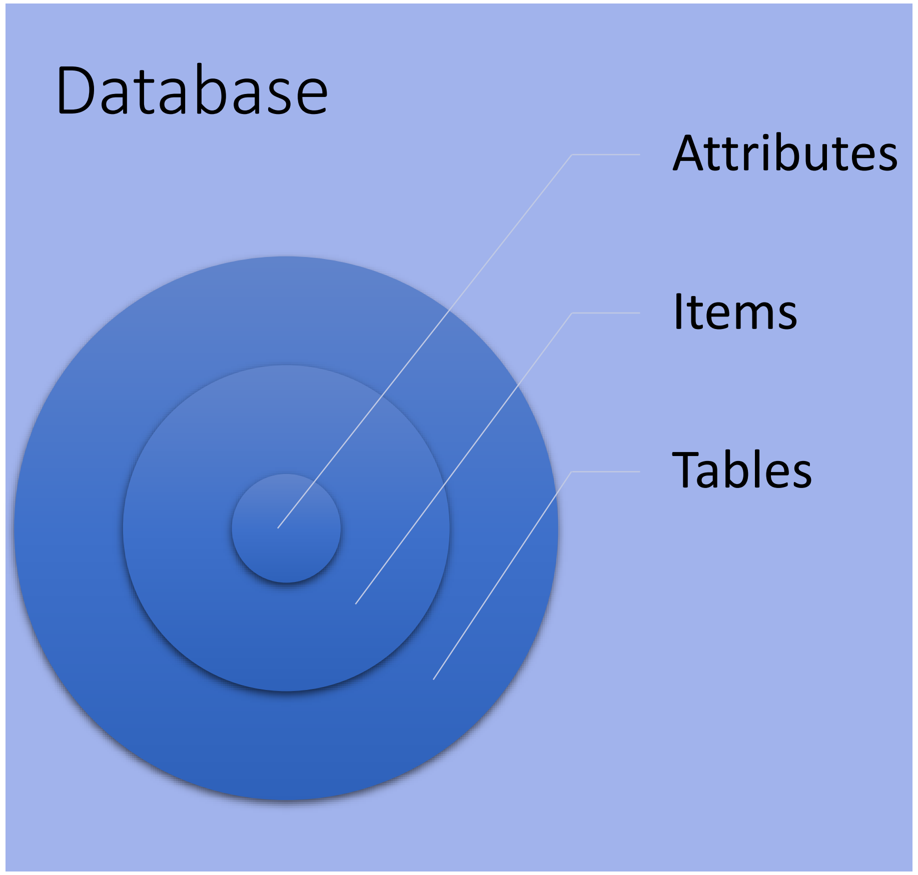


Features

- Flexible
  - Does not have a fixed schema
- Efficient Indexing
  - Every item identified by a primary key
- Strong consistency
  - Implemented with Atomic Counters
  - Disk-only writes
- Secure with Monitoring
  - AWS Identity and Access Management
  - CloudWatch for monitoring request throughput, latency and resource consumption
- Amazon Elastic MapReduce Integration
- Amazon Redshift Integration



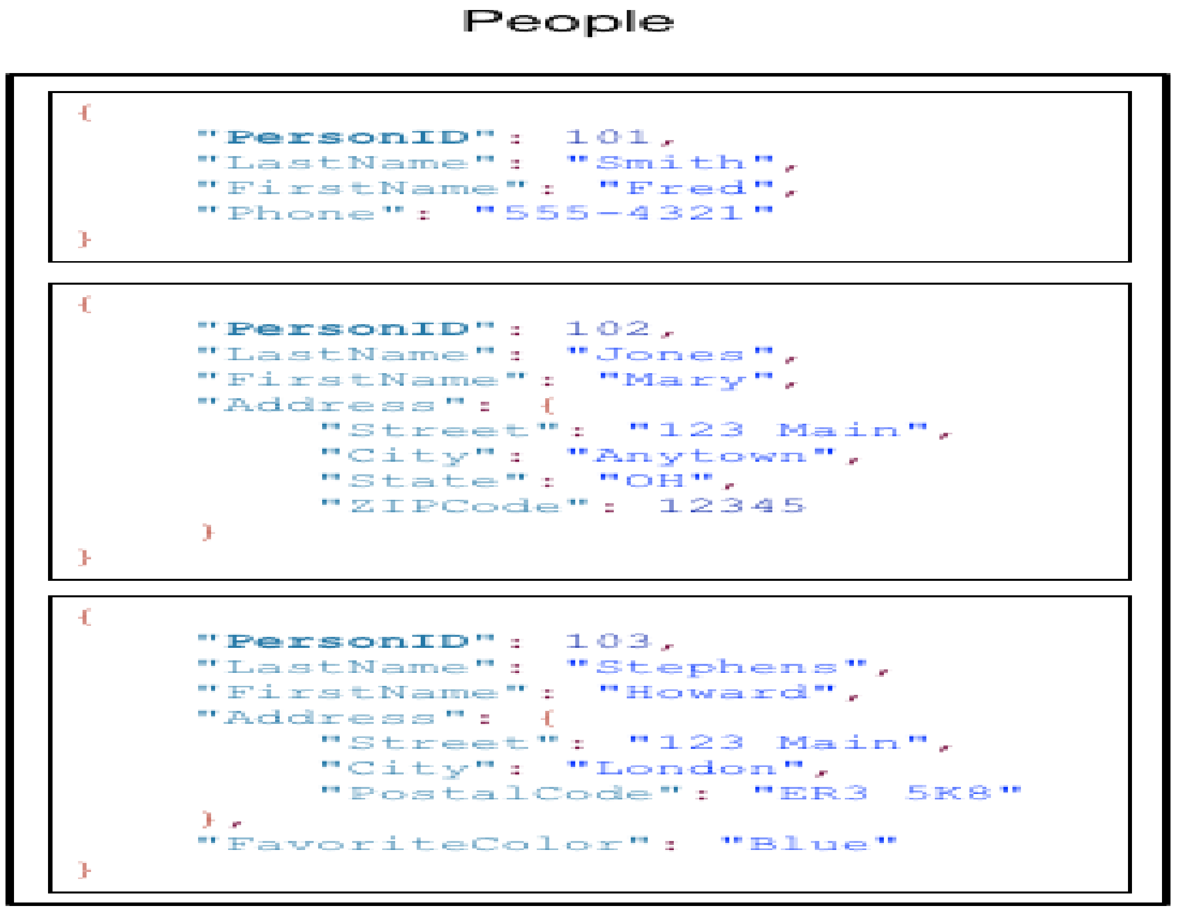
Data model concepts



- Except for the primary key, DynamoDB is schema-less
- Each item can have any number of attributes
- An attribute is a name-value pair
  - can be single valued or multi-valued set

Key components

- **Tables** – Similar to other database systems, DynamoDB stores data in tables. A *table* is a collection of data. For example, table called *People* can be use to store personal contact information.
- **Items** – Each table contains multiple items. An *item* is a group of attributes that is uniquely identifiable among all of the other items. Items in DynamoDB are similar in many ways to rows, records, or tuples in other database systems. E.g., each item in table People represents a person.
- **Attributes** – Each item is composed of one or more attributes. An *attribute* is a fundamental data element, something that does not need to be broken down any further. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems. For example, an item in the example *People* table contains attributes called *PersonID*, *LastName*, *FirstName*, and so on.



People

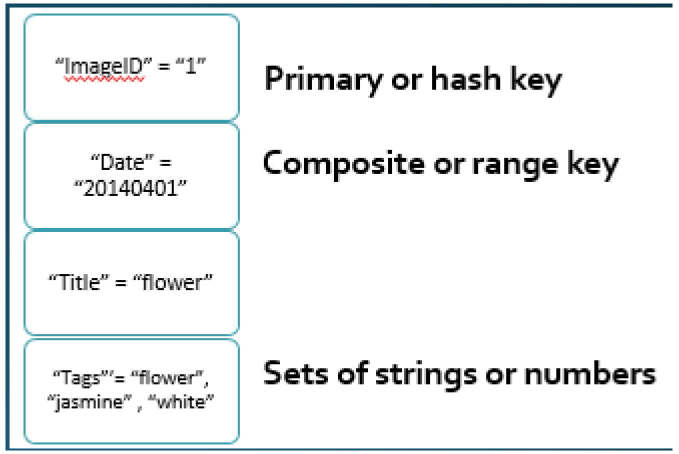
```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

Data model concepts

- Primary keys
  - Hash type primary key
  - Hash and range type primary key
- Secondary Indexes
  - Local secondary index
  - Global secondary index
- DynamoDB data types
  - Scalar data types
  - Multivalued data types



Supported Operations

- Table operations
  - create, update and delete tables
- Item operations
  - add, update and delete items from a table
  - retrieve a single item (GetItem) or multiple items (BatchGetItem)
- Query and Scan
  - query a table using the hash attribute and an optional range filter.
  - If the table has a secondary index, you can also Query the index using its key
  - Scan operation reads every item in the table or secondary index

Supported Operations

- Data Read and Consistency considerations
  - Multiple copies of each item to ensure durability
  - Eventually Consistent Reads
  - Strongly Consistent Reads
- Conditional updates and concurrency control
  - updates made by one client don't overwrite updates made by another client
  - “conditional write” and “atomic counter”

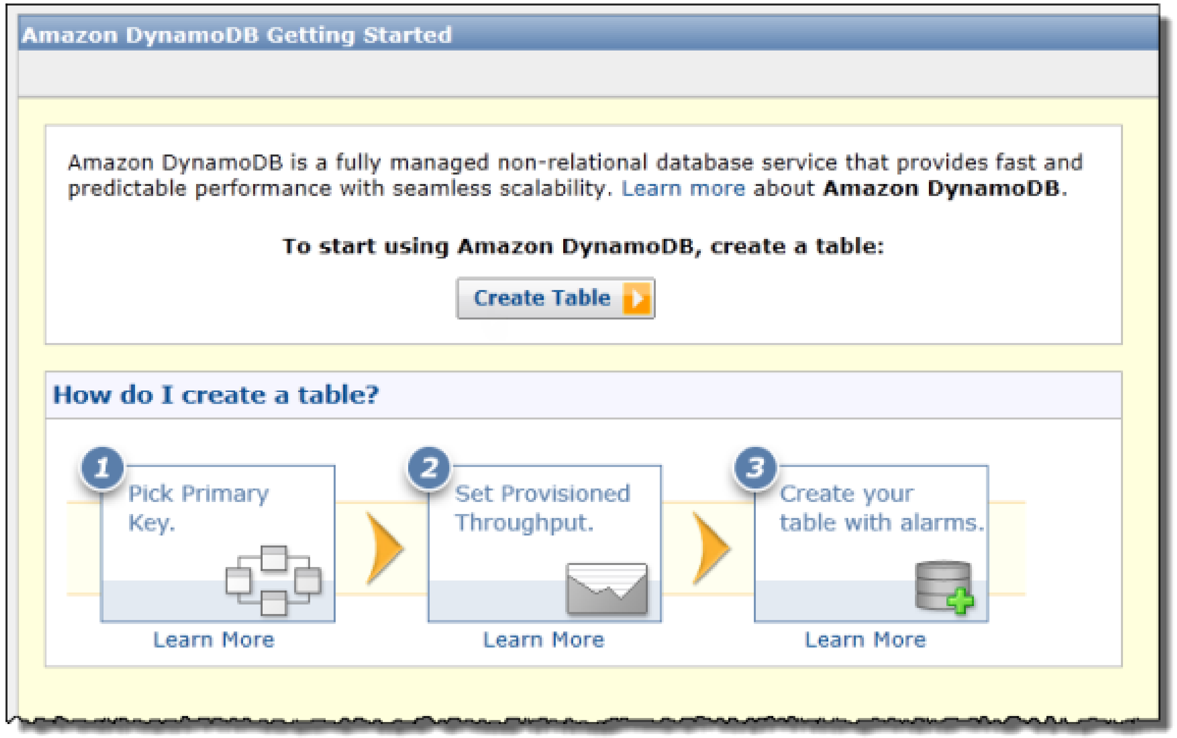
Considerations

- Item size is limited to 64KB
- Attribute values can not be null or empty
- Hash primary key attribute value is limited to 1024 bytes
- Range primary key attribute value is limited to 2048 bytes
- Up to 5 local as well as global secondary indexes per table

Start using DynamoDB! ☺

- The AWS Management Console for Amazon DynamoDB is available at

https://console.aws.amazon.com/dynamodb/home



Amazon DynamoDB Getting Started

Amazon DynamoDB is a fully managed non-relational database service that provides fast and predictable performance with seamless scalability. Learn more about **Amazon DynamoDB**.

To start using Amazon DynamoDB, create a table:

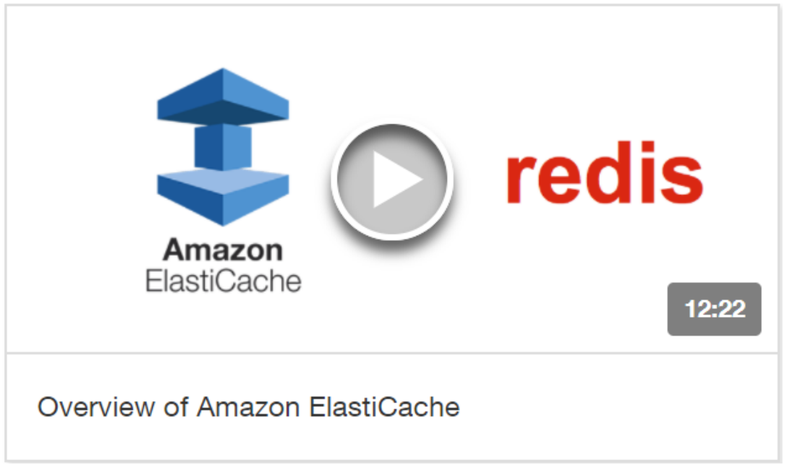
Homework

- Java and DynamoDB
- JavaScript and DynamoDB
- Node.js and DynamoDB
- .NET and DynamoDB
- PHP and DynamoDB
- **Python and DynamoDB**
- Ruby and DynamoDB

ElasticCache

ElaticCache

- What is it?
  - Fully managed, in-memory data store and cache in the cloud. Compatible with Redis and Memcached.
  - Provides sub-millisecond latency to power real-time applications.
- Benefits
  - Automatically detects and replaces failed nodes
  - Provides enhanced visibility into key performance metrics via Amazon CloudWatch



ElasticCache Scenario

- ElastiCache is a managed service that lets you create in-memory key-value stores.
- First of all, you will create a cluster and then update your application to use it.
- Imagine a phonebook application.
  - Whenever we want to retrieve the address of a business, we will need to access the database to retrieve that information.
  - For popular businesses such as banks or post offices, for example, it is likely that the exact same query will be run very frequently.
  - Instead of always reaching to the database, you can update your application logic to check the cache first and, if the information isn't cached, retrieve it from the database and then cache the result:

```
Sub get_address(name)
  address = cache.get(name, "address")
  If address Is Empty Then // cache miss
    address = db.query("SELECT address from businesses
WHERE name = ?", name)
    cache.set(name, "address", address)
  End If
  Return address
End Sub
```

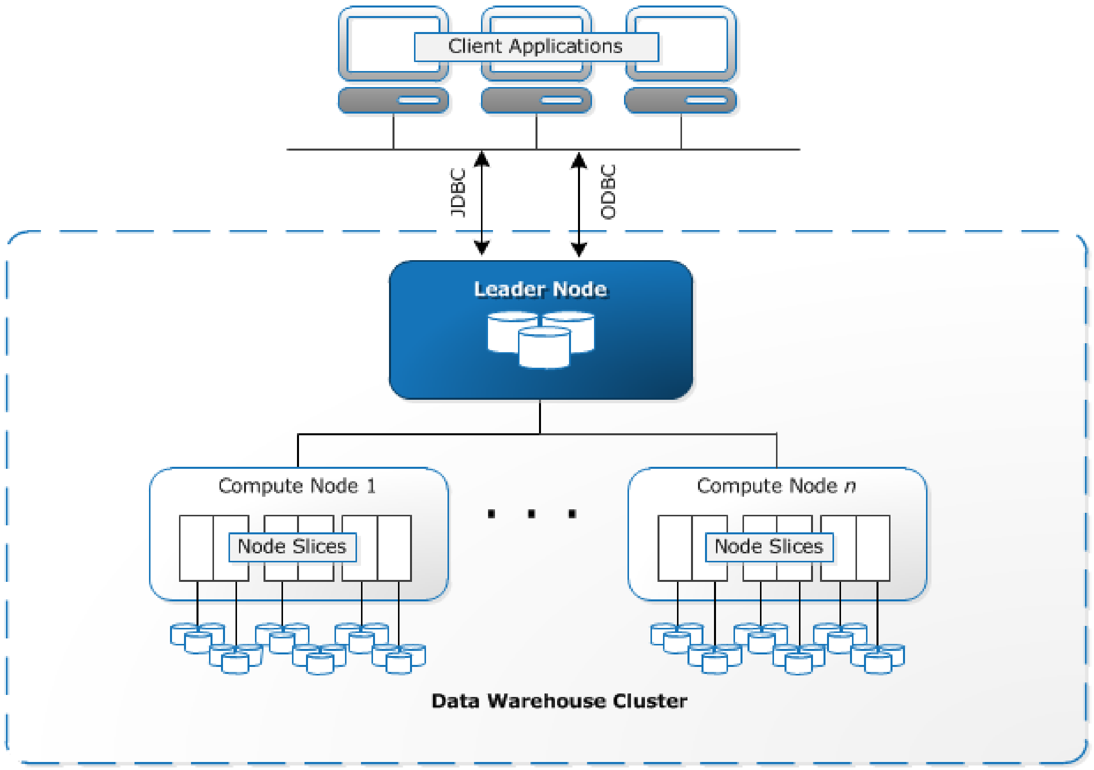
AWS RedShift

What is AWS RedShift

- Amazon Redshift is a fully managed data warehouse
- It allows you to run complex analytic queries against petabytes of structured data, using query optimization, columnar storage on high-performance local disks, and massively parallel query execution.
- Amazon Redshift also includes Redshift Spectrum, allowing you to directly run SQL queries against exabytes of unstructured data in Amazon S3.
- You can use open data formats, including CSV, TSV, Parquet, Sequence, and RCFile.
- Redshift Spectrum automatically scales query compute capacity based on the data being retrieved, so queries against Amazon S3 run fast, regardless of data set size.

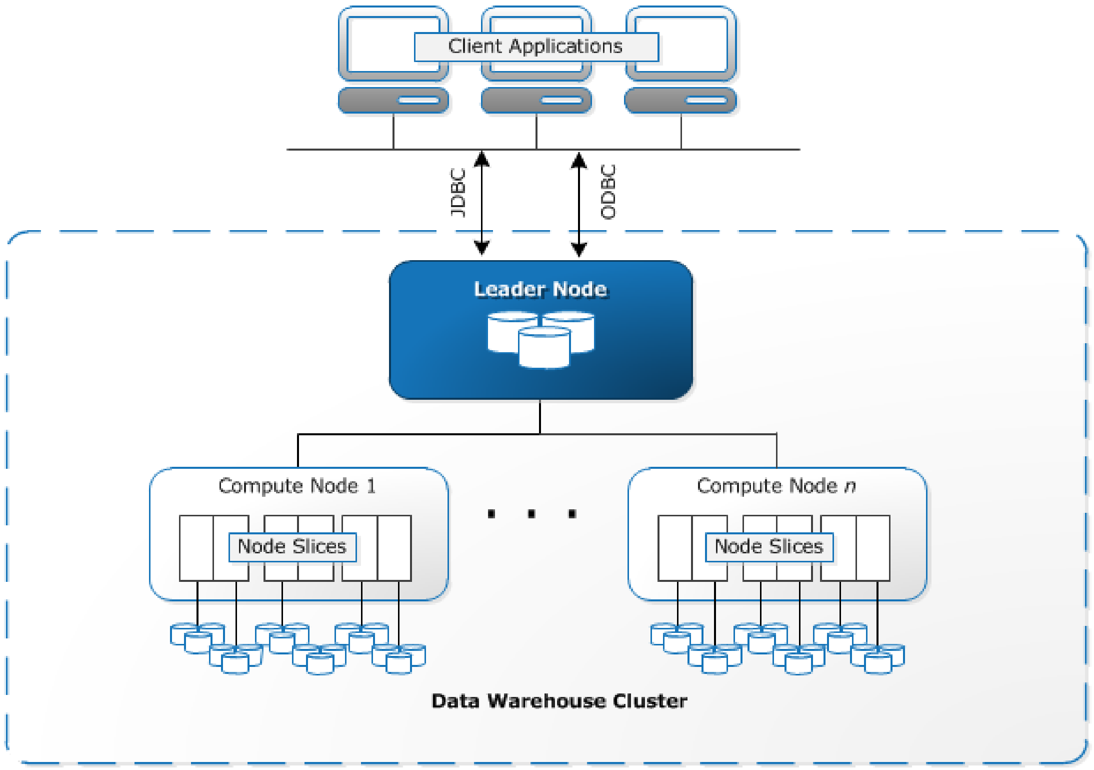
Architecture

- **Client applications**
  - integrates with various data loading and ETL BI reporting, data mining, and analytics tools.
  - Is based on industry-standard PostgreSQL



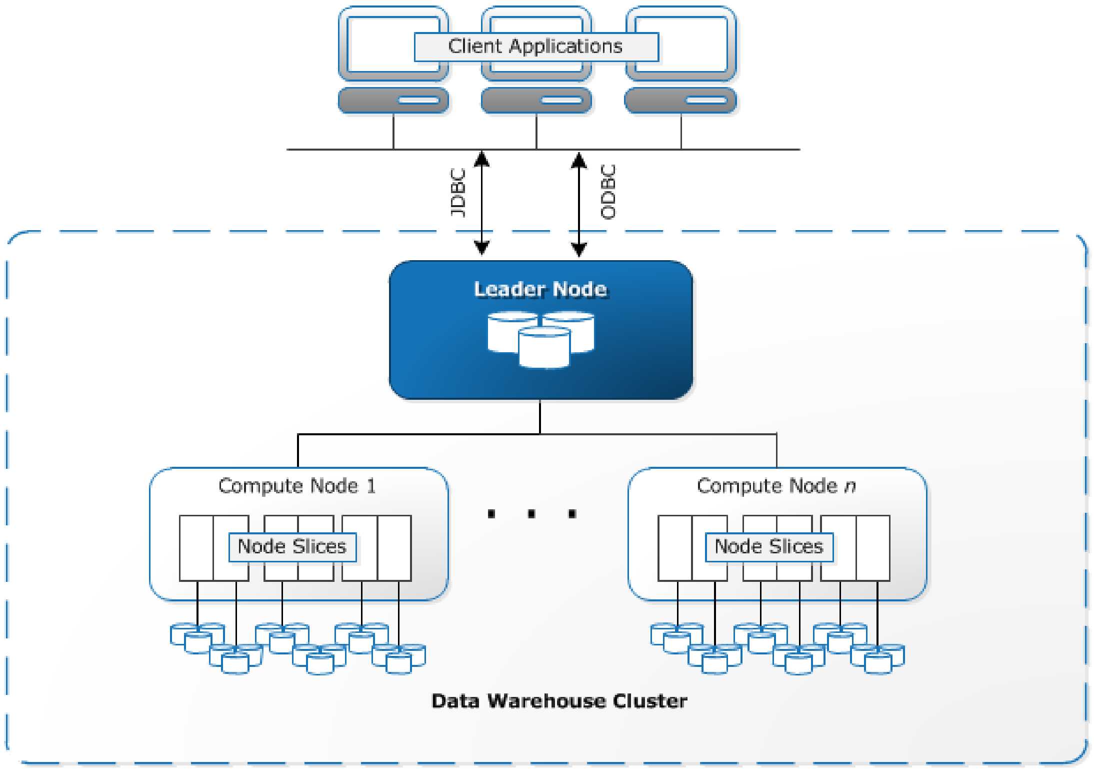
Architecture

- **Connections**
  - Amazon Redshift communicates with client applications by using industry-standard PostgreSQL JDBC and ODBC drivers.



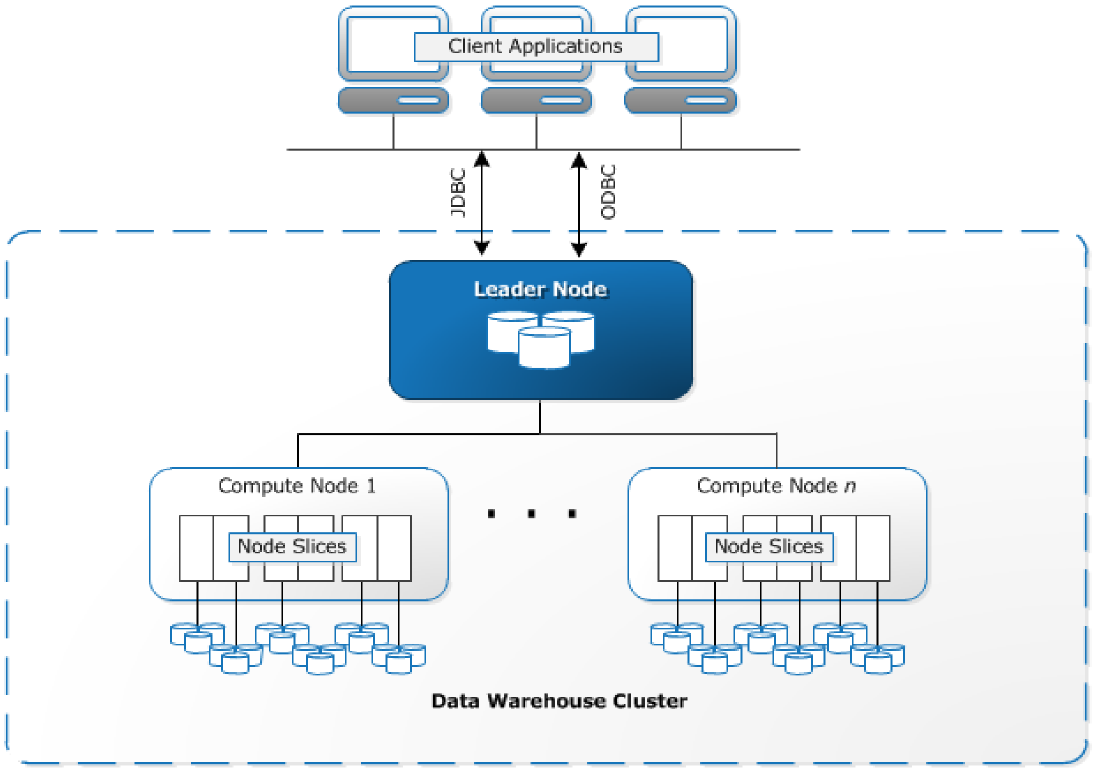
Architecture

- **Clusters**
  - A cluster is composed of one or more *compute nodes*.
  - If a cluster is provisioned with two or more compute nodes, an additional *leader node* coordinates the compute nodes and handles external communication.
  - Client application interacts directly only with the leader node. The compute nodes are transparent to external applications.



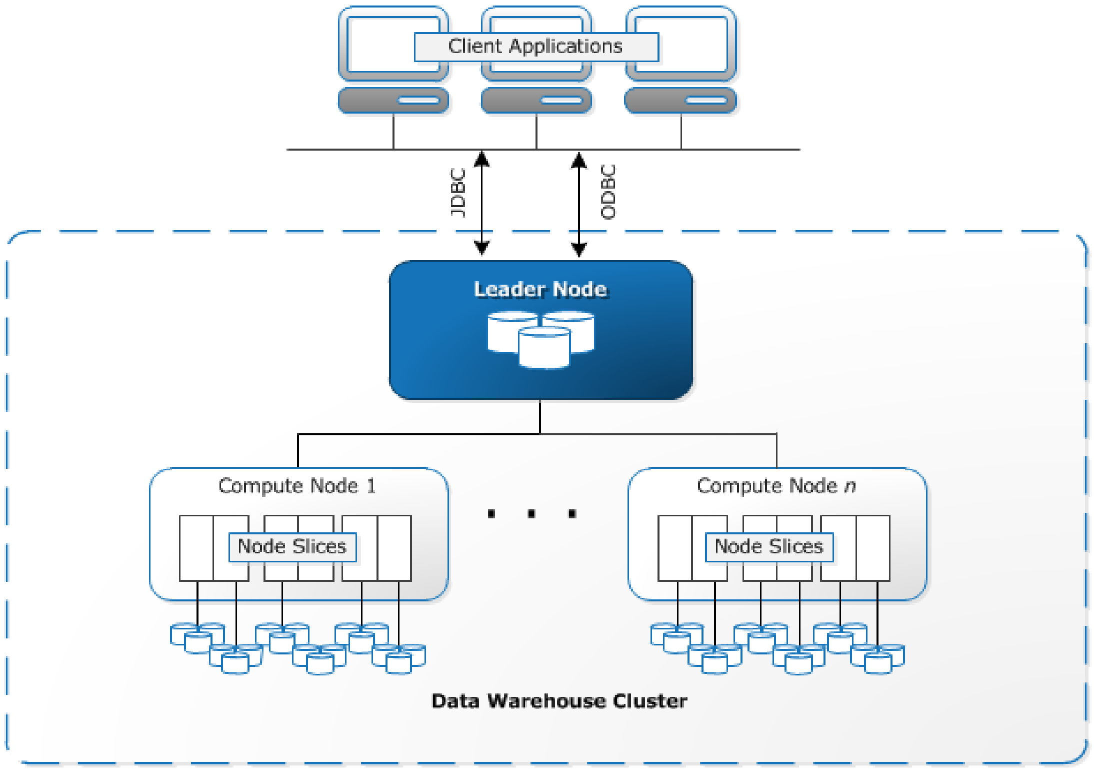
Architecture

- Compute Node
  - Each compute node has its own dedicated CPU, memory, and attached disk storage, which are determined by the node type.
  - two node types;
    - dense storage nodes and
    - dense compute nodes.
  - Each node provides two storage choices: from a single 160 GB node to multiple 16 TB nodes to support a petabyte of data or more.



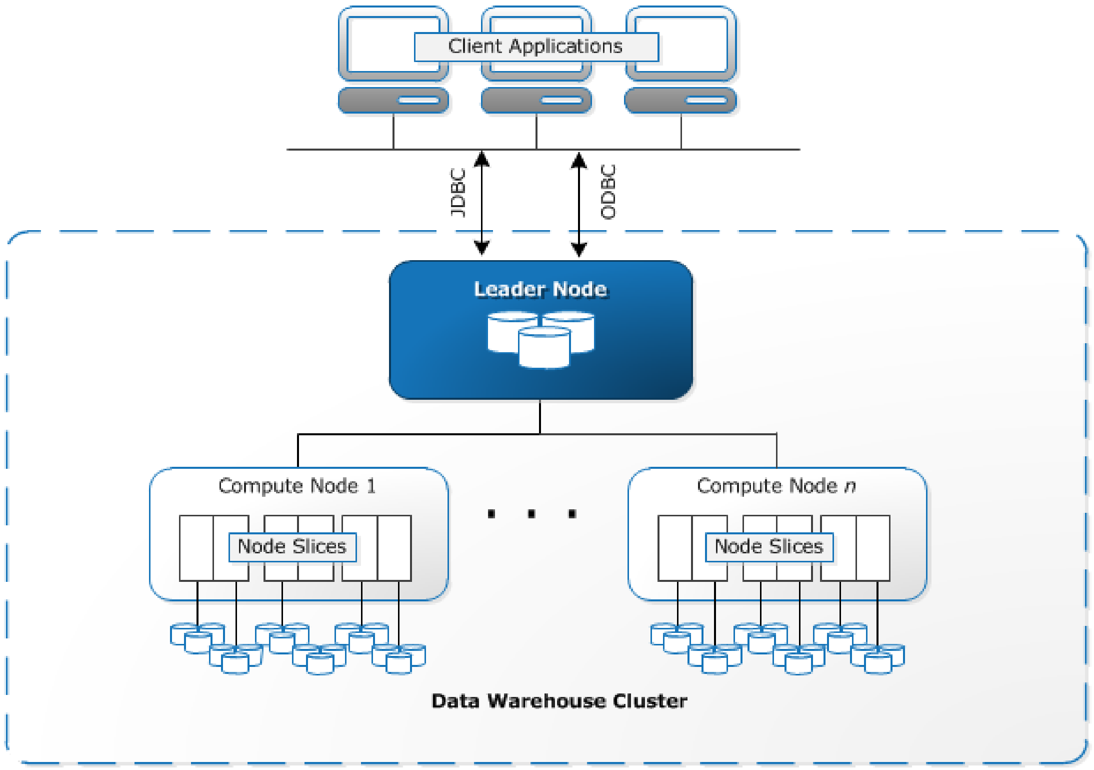
Architecture

- **Node Slides**
  - A compute node is partitioned into slices.
  - Each slice is allocated a portion of the node's memory and disk space, where it processes a portion of the workload assigned to the node.
  - The leader node manages distributing data to the slices and apportions the workload for any queries or other database operations to the slices.
  - The slices then work in parallel to complete the operation.
  - When you create a table, you can optionally specify one column as the distribution key. When the table is loaded with data, the rows are distributed to the node slices according to the distribution key that is defined for a table. Choosing a good distribution key enables Amazon Redshift to use parallel processing to load data and execute queries efficiently.



Architecture

- **Databases**
  - A cluster contains one or more databases.
  - User data is stored on the compute nodes. Your SQL client communicates with the leader node, which in turn coordinates query execution with the compute nodes.
  - Amazon Redshift is a relational database management system (RDBMS) optimized for high-performance analysis and reporting of very large datasets.
  - Amazon Redshift is based on PostgreSQL 8.0.2.



Key Features

- Massively parallel processing
- Columnar data storage
- Data compression
- Query optimization
- Compiled code

Key Features

- Massively parallel processing
- Columnar data storage
- Data compression
- Query optimization
- Compiled code

- MPP enables fast execution of the most complex queries operating on large amounts of data. Multiple compute nodes handle all query processing leading up to final result aggregation, with each core of each node executing the same compiled query segments on portions of the entire data.
- rows of a table are distributed to the compute nodes so that the data can be processed in parallel.
- Loading data from flat files takes advantage of parallel processing by spreading the workload across multiple nodes while simultaneously reading from multiple files.

Key Features

- Massively parallel processing
- Columnar data storage
- Data compression
- Query optimization
- Compiled code

- Storing database table information in a columnar fashion reduces the number of disk I/O requests and reduces the amount of data you need to load from disk.
- Loading less data into memory enables Amazon Redshift to perform more in-memory processing when executing queries.
- When columns are sorted appropriately, the query processor is able to rapidly filter out a large subset of data blocks.

Key Features

- Massively parallel processing
- Columnar data storage
- Data compression
- Query optimization
- Compiled code

- Data compression reduces storage requirements, thereby reducing disk I/O, which improves query performance.
- When you execute a query, the compressed data is read into memory, then uncompressed during query execution.
- Loading less data into memory enables Amazon Redshift to allocate more memory to analyzing the data.

Key Features

- Massively parallel processing
- Columnar data storage
- Data compression
- Query optimization
- Compiled code

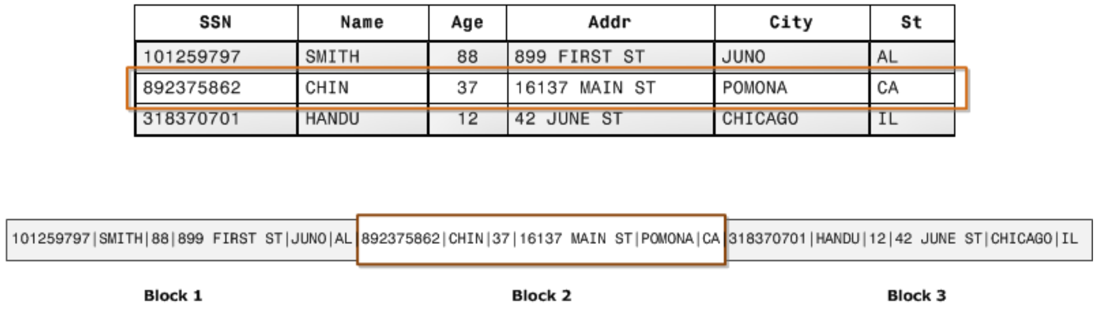
- Incorporates a query optimizer that is MPP-aware and also takes advantage of the columnar-oriented data storage.
  - multi-table joins
  - subqueries, and
  - aggregation.

Key Features

- Massively parallel processing
- Columnar data storage
- Data compression
- Query optimization
- Compiled code

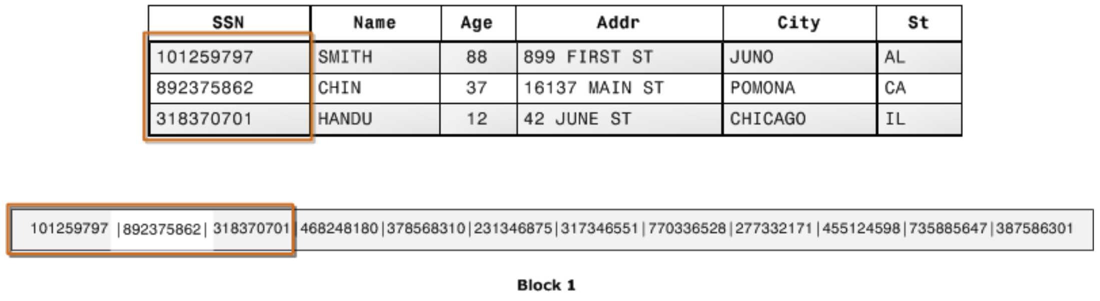
- The leader node distributes fully optimized compiled code across all of the nodes of a cluster.
- Compiling the query eliminates the overhead associated with an interpreter and therefore increases the execution speed, especially for complex queries.
- The compiled code is cached and shared across sessions on the same cluster, so subsequent executions of the same query will be faster, often even with different parameters.

Row Storage vs Columnar Storage



SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

```
101259797|SMITH|88|899 FIRST ST|JUNO|AL|892375862|CHIN|37|16137 MAIN ST|POMONA|CA|318370701|HANDU|12|42 JUNE ST|CHICAGO|IL
```



SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

```
101259797 |892375862| 318370701|468248180|378568310|231346875|317346551|770336528|277332171|455124598|735885647|387586301
```

a major bottleneck in handling big data is disk access. Columnar databases boost performance by reducing the amount of data that needs to be read from disk, both by efficiently compressing the similar columnar data and by reading only the data necessary to answer the query.

Tutorial

- Step 1: Set Up Prerequisites
- Step 2: Create an IAM Role
- Step 3: Launch a Sample Amazon Redshift Cluster
- Step 4: Authorize Access to the Cluster
- Step 5: Connect to the Sample Cluster
- Step 6: Load Sample Data from Amazon S3