MODULE 6 · STUDY NOTES

Computing (Database Services)

The managed database portfolio — relational RDS and Aurora, NoSQL DynamoDB, purpose-built engines, and migrating databases into AWS with DMS and the Schema Conversion Tool

AWS databases are fully managed and purpose-built: rather than one engine for every job, AWS offers a family of services, each tuned to a category of workload. This module maps that family and drills into the three services a systems operator meets most often. It opens with the SQL-versus-NoSQL divide and how to match an application's data shape, size, access patterns, and performance needs to the right engine. It then covers **Amazon RDS** — the managed relational service with six engines, its instance-class and storage-type choices, automated backups and snapshots, Multi-AZ high availability with automatic failover, and read replicas for read scaling. **Amazon Aurora** follows as the cloud-optimized MySQL/PostgreSQL-compatible engine with a six-copy, three-AZ cluster architecture, and **Amazon DynamoDB** as the NoSQL document and key-value store with its partition-key data model and multi-Region global tables. The module closes with **AWS DMS** and the **AWS Schema Conversion Tool**, which move databases into AWS — homogeneous or heterogeneous — with minimal downtime, applied to a Mom & Pop Café scenario that lifts an at-risk single-instance database onto RDS.

LEARNING OBJECTIVES

- Differentiate the types of managed database services offered by AWS and identify their recommended use
- Identify factors to consider when selecting a database (engine and workload)
- Explain the purpose, function, and benefits of Amazon RDS, Amazon Aurora, and Amazon DynamoDB
- Describe the main features and benefits of Amazon RDS, Amazon Aurora, and Amazon DynamoDB
- Explain the benefits of AWS Database Migration Service (AWS DMS) and the capabilities of the AWS Schema Conversion Tool (AWS SCT)

6.1 AWS database services

AWS offers a variety of **fully managed** database services, each purpose-built for a category of application. *Managed* is the key word: AWS handles routine operational tasks — hardware provisioning, patching, backups, and scaling — that you would otherwise maintain yourself for an on-premises or standalone-server database. From a SysOps perspective that is the core benefit: the simplification and automation of management and operational work. The portfolio splits along the SQL/NoSQL line and by primary use case, so choosing well starts with knowing what each service is for.

THE AWS DATABASE FAMILIES

- **Amazon RDS** — managed relational (SQL) database for transactional applications; a choice of six engines. This module's main SQL focus, along with Aurora.
- **Amazon Redshift** — a fast, scalable, cost-effective SQL **data warehouse** for analytics workloads.
- **Amazon DynamoDB** — a NoSQL database supporting **document and key-value** models and transactional, internet-scale workloads. This module's main NoSQL focus.
- **Amazon Neptune** — a managed **graph** database for highly connected datasets; features continuous backup to Amazon S3 and point-in-time recovery with replication across AZs.
- **Amazon ElastiCache** — an **in-memory** data cache supporting the open-source **Redis** and **Memcached** engines; cache nodes are configured and launched from the console.

SQL and NoSQL differ in four fundamentals. A **SQL** database conforms to the relational model — data lives in **rows and columns**, where each row holds all the information about one entry and columns are the attributes that separate the data points. Its schema is **fixed**: columns must be locked before data entry (and can only be amended by altering the database entirely and taking it offline). Data is queried with **SQL**, which allows complex queries, and the database scales **vertically** by increasing hardware power. A **NoSQL** database is **non-relational** and stores data using one of many models — key-value pairs, documents, or graphs. The term once meant *non-SQL*, but today it means *not only SQL*, because a NoSQL server can also support SQL-based queries. Its schema is **dynamic**, each row need not contain data for every column, queries focus on **collections of documents**, and it scales **horizontally** by adding servers.

SQL (RELATIONAL)	NOSQL (NON-RELATIONAL)
e.g. a book catalog as rows of ISBN, Title, Author, Format ▪ Rows and columns; one row per entry ▪ Fixed schema — columns defined up front ▪ Queried with SQL; supports complex queries ▪ Scales vertically (more powerful hardware)	*e.g. the same book as one JSON key-value document* ▪ Key-value, document, or graph models ▪ Dynamic schema — add data rapidly ▪ Queried over collections of documents ▪ Scales horizontally (more servers)

Choosing a database service requires understanding both the capabilities of the service and the needs of the application, then matching the two. Some engines use the relational model and suit transactional applications; others follow the NoSQL model for internet-scale applications; still others are ideal for real-time workloads that need an in-memory cache. The workload characteristics that steer the choice are the **shape of the data**, its **size and computational requirements**, and its **access patterns and performance needs**.

TABLE 6.1 AWS database recommendations

IF YOU NEED...	TYPE	CONSIDER
A managed relational database launched with a choice of six engines	Relational	Amazon RDS
A MySQL/PostgreSQL-compatible relational database built for the cloud	Relational	Amazon Aurora
A fully managed NoSQL database	NoSQL	Amazon DynamoDB
A managed graph database	NoSQL	Amazon Neptune
A managed Redis key-value store	NoSQL	Amazon ElastiCache
A columnar-storage SQL data warehouse	Relational	Amazon Redshift

TABLE 6.2 Usage scenarios – what each service is best at

SERVICE	BEST USED FOR
Amazon RDS / Aurora	Transactional apps — ERP, CRM, ecommerce — to log transactions and store structured data (Aurora adds open-source cost, high performance)
Amazon Redshift	Analytic apps for operational reporting and querying terabyte- to exabyte-scale data
Amazon ElastiCache	Real-time use cases needing submillisecond latency — gaming leaderboards, chat/messaging, streaming, IoT
Amazon Neptune	Navigating highly connected data — social news feeds, recommendations, fraud detection
Amazon DynamoDB	Internet-scale apps — hospitality, dating, ridesharing — serving content and storing structured and unstructured data

KEY TAKEAWAYS FOR THIS TOPIC

- Managed databases simplify and automate operational tasks — provisioning, patching, backups, and scaling.
- The choice of managed database is **application-driven**: pick SQL or NoSQL from the workload's storage, object size/type, and access patterns, then narrow by primary use case.
- Each offering has a **primary use case** — be familiar with it.
- Unsupported database engines or database versions can still be hosted on **Amazon EC2** (where you manage them yourself).

6.2 Amazon RDS

Amazon Relational Database Service (Amazon RDS) is a managed relational database service offering six familiar engines to choose from: **Amazon Aurora, MySQL, MariaDB, Oracle, Microsoft SQL Server, and PostgreSQL**. Because it is fully managed, RDS handles routine tasks — provisioning, patching, backup, recovery, and failure detection and repair. Its basic building block is the **DB instance**: an isolated database environment that can contain multiple user-created databases and is accessed with the same tools and applications you would use with a standalone database instance.

When you create a DB instance you specify its **instance class**, which sets the server's CPU, memory, and network capacity, and its **storage type**, which sets the type, performance, and size of the underlying Amazon EBS volume. Because classes and storage types differ in performance and price, you can tailor both to the needs of the database.

TABLE 6.3 Two choices when you create a DB instance

YOU SPECIFY	IT DETERMINES	OPTIONS
Instance class	CPU, memory, and network capacity (compute)	Standard · Memory Optimized · Burstable Performance
Storage type	Type, performance, and size of the EBS volume	General Purpose SSD (`gp2`) · Provisioned IOPS SSD (`io1`) · magnetic

Backups are automatic. RDS periodically backs up the **entire** DB instance to a storage-volume **snapshot** during a specified **backup window**, keeping it for a defined **backup retention period**. The first snapshot contains the full data; subsequent snapshots are **incremental**, holding only what changed since the most recent one. You can also create snapshots **manually** at any time. AWS CLI commands manage the snapshot lifecycle.

TABLE 6.4 RDS snapshot commands (AWS CLI)

TASK	COMMAND
Create snapshot	`aws rds create-db-snapshot --db-instance-identifier mytestdb --db-snapshot-identifier mydbsnapshot`
Restore from snapshot	`aws rds restore-db-instance-from-db-snapshot --db-instance-identifier mytestdb-new --db-snapshot-identifier mydbsnapshot`
Copy snapshot	`aws rds copy-db-snapshot --source-db-snapshot-identifier mydbsnapshot --target-db-snapshot-identifier mydbsnapshot-copy --copy-tags`
Delete snapshot	`aws rds delete-db-snapshot --db-snapshot-identifier mydbsnapshot`

One operational note on restores: after a DB instance is restored from a snapshot, you must add the new instance to the **security group** used by the original instance if you want the same connectivity as before.

High availability comes from a Multi-AZ deployment. This configuration automatically creates a **standby copy** of the DB instance in another Availability Zone within the same VPC. After an initial full copy, transactions are **synchronously replicated** to the standby, so it stays perfectly current. Running across multiple AZs enhances availability during planned maintenance and protects the database against instance failure and AZ disruptions. In a typical layout the primary runs in AZ 1 and serves applications in both AZ 1 and AZ 2, while the standby in AZ 2 provides data redundancy.

On failover, RDS promotes the standby automatically. If the primary fails, RDS brings the standby online as the new primary and directs requests to it. Because applications reference the database by name through the **RDS DNS endpoint**, the failover happens **without any change to application code** — and because replication is synchronous, there is **no data loss**.

Scaling works along two axes. You can increase a server's capacity by **changing its instance class** (more CPU and memory) or its **storage capacity** — and modifying allocated storage happens **without downtime**. For read-heavy workloads you add **read replicas**: read-only copies kept current by **asynchronous** replication that serve read traffic and let you scale out beyond the capacity of a single instance while the primary continues to handle reads and writes.

MULTI-AZ STANDBY	READ REPLICA
for availability and durability	*for read scaling and performance*
▪ Standby in a second AZ, synchronously replicated	▪ Read-only copy, asynchronously replicated
▪ Serves no read traffic on its own	▪ Absorbs read-heavy query load
▪ Automatic failover — standby becomes primary	▪ Scales reads beyond one instance
▪ No data loss; no application code change	▪ Primary keeps handling read and write

COMMON PITFALL

Watch the downtime rule and don't confuse the two replication features. **Changing the instance class requires downtime**, but **increasing allocated storage does not**. And the **Multi-AZ standby is for availability** (synchronous, auto-failover, no reads), while **read replicas are for read scaling** (asynchronous, read-only) — they are not interchangeable.

RDS KEY TAKEAWAYS AND RECOMMENDED PRACTICES

- RDS supports multiple **open-source and commercial** database engines.
- The **instance class** defines compute; the **storage type** defines the storage type, size, and IOPS.
- **Standby** (Multi-AZ) and **read replicas** provide enhanced availability, durability, and read performance.
- **Automated backups** and **manual snapshots** are the methods for backing up and restoring a database.
- Enable automatic backups and set the **backup window** for a time of minimal write activity; if using Multi-AZ, **test the failover** to learn how long the switch-over takes and confirm the application reconnects.

6.3 Amazon Aurora

Amazon Aurora is a fully managed relational database engine — and one of the six RDS engines — that is **compatible with MySQL and PostgreSQL** and optimized for the cloud. Its MySQL/PostgreSQL compatibility lets you run existing applications and tools **without modification**, using familiar SQL. Aurora is built to combine the performance and availability of high-end commercial databases with the simplicity and cost of an open-source database.

BENEFITS OF AMAZON AURORA

- **Fast** — up to **5x** the throughput of standard MySQL and **3x** that of standard PostgreSQL; scale compute up and down and grow storage as needed.
- **Compatible** — works with MySQL and PostgreSQL, so existing applications and tools run unchanged.
- **Cost-effective** — pay-as-you-go: you pay only for the storage and I/O operations your database consumes.
- **Serverless option** — an on-demand configuration where the database automatically starts up, shuts down, and scales capacity to match the application, with no instances or clusters to manage.

Aurora's high-availability architecture stores **six copies of your data across three Availability Zones** and allows up to **15 read replicas**. Creating an Aurora instance creates a **DB cluster** — one or more DB instances plus a **cluster volume** that manages their data. The cluster volume is a virtual storage volume that **spans multiple AZs**, with each AZ holding a copy of the cluster data (Aurora's storage protection groups are **10 GB**). A cluster has exactly one **primary instance** (it supports reads and writes and performs all data modifications) and up to **15 Aurora replicas** (read-only). Multiple replicas distribute the read workload, and placing them in separate AZs increases availability.

COMMON PITFALL

Aurora's replicas are not the same as an RDS Multi-AZ standby. Aurora achieves availability through its own **cluster** design — six data copies across three AZs and up to 15 **read-only** replicas over a shared cluster volume — rather than a single synchronous standby. Both raise availability, but the mechanisms differ.

6.4 Amazon DynamoDB

Amazon DynamoDB is a fast, flexible, and highly scalable **non-relational (NoSQL)** database service that supports both the **document** and **key-value store** models. It is the main choice among the AWS NoSQL offerings, well suited to transactional, internet-scale workloads.

BENEFITS OF AMAZON DYNAMODB

- **Fully managed** — create a table and set a target utilization for automatic scaling; DynamoDB handles provisioning, setup, patching, running the distributed cluster, and partitioning data as you scale, plus point-in-time recovery, backup, and restore for all tables.
- **Low-latency queries** — average server-side latency is typically **single-digit milliseconds**, using automatic partitioning and SSD technology to deliver low latency at any scale.
- **Fine-grained access control** — integrates with **AWS IAM**, so each user gets unique credentials and scoped access to services and resources.
- **Flexibility** — stores, queries, and updates data as **JSON** documents, ideal for semi-structured data; integrates with **Amazon CloudWatch** for throughput and latency statistics.

The DynamoDB data model stores data in **tables**. A table is a collection of **items**, and each item is a collection of **attributes** — you can think of items as rows and attributes as columns. DynamoDB divides a table's items into **partitions** based on the **partition key** value; a partition is an allocation of SSD-backed storage that DynamoDB automatically replicates across multiple AZs within a Region and manages for you. The partition key is also called the **hash attribute**. An optional **sort key** stores all items with the same partition key value physically close together and orders them, representing a one-to-many relationship and enabling querying on the sort key.

TABLE 6.5 DynamoDB primary key types

PRIMARY KEY TYPE	MADE OF	INDEXING AND ITEM IDENTITY
Partition (simple)	One attribute — the partition key	DynamoDB builds an unordered index on the partition key; each item is identified by its partition key value
Partition and sort (composite)	Partition key + sort key	Unordered index on the partition key, sorted index on the sort key; each item is identified by the combination of both

Global tables provide high availability and scalability across Regions. A global table is a collection of one or more DynamoDB tables — called **replica tables** — owned by a single AWS account, with **one replica per Region**, each carrying the same table name, primary key, and set of data items. You choose the Regions when you create the global table; DynamoDB creates identical tables in those Regions and propagates all ongoing data changes to every replica. This suits massively scaled applications with globally dispersed users, who get fast performance by reaching the nearest replica and continued access to the data if one Region becomes temporarily unavailable.

COMMON PITFALL

Moving from a relational database to DynamoDB means rethinking data modeling, not just swapping services — understand the SQL-versus-NoSQL differences in keys, indexing, and schema. DynamoDB's scaling and single-digit-millisecond performance make it ideal for **high-throughput, latency-sensitive** applications such as web, mobile, IoT, and gaming.

6.5 AWS DMS and the Schema Conversion Tool

The **AWS Database Migration Service (AWS DMS)**, together with the **AWS Schema Conversion Tool (AWS SCT)**, migrates databases into AWS quickly, securely, and with **minimal downtime** — keeping your applications running throughout. DMS supports most widely used commercial and open-source databases (Oracle, PostgreSQL, MySQL, Amazon Aurora) and handles both **homogeneous** migrations (such as Oracle to Oracle) and **heterogeneous** migrations between different platforms (such as Oracle to MySQL, or MySQL to Aurora). It can migrate an on-premises database to a database running on Amazon EC2, supports **one-time** migration as well as **continuous replication** (capturing source changes and applying them to the target in a transactionally consistent way), and can migrate across data models — SQL↔SQL, SQL↔NoSQL, or NoSQL↔NoSQL. Its one firm rule: **at least one endpoint must be on an AWS service** — on-premises to on-premises is not supported.

TABLE 6.6 DMS source/target combinations

SOURCE ↓ / TARGET →	ON PREMISES	AWS: RDS, REDSHIFT, S3, DYNAMODB	AMAZON EC2
On premises	Not supported	Supported	Supported
Amazon RDS / Amazon S3	Supported	Supported	Supported
Amazon EC2	Supported	Supported	Supported
Azure SQL	Not supported	Supported	Supported

THE FOUR DMS COMPONENTS AND THE WORKFLOW

- **Replication instance** — an Amazon EC2 instance in your VPC that runs the migration tasks; made highly available with a **Multi-AZ** synchronous standby, and its storage can be encrypted with the default KMS key or a customer master key (CMK).
- **Task** — the process that performs the migration on the replication instance; it can migrate existing data only, migrate existing data **and** replicate ongoing changes, or replicate changes only. Multiple tasks can run concurrently.
- **Source endpoint** — the source database, running on premises, on Amazon RDS, or on Amazon EC2.
- **Target endpoint** — the target database on AWS.
- Workflow: **create a replication server → specify source and target endpoints → create one or more tasks** to migrate the data.

The Schema Conversion Tool handles what DMS does not. During a migration, DMS creates only the **minimal target objects** required to move the data efficiently — it does **not** perform schema or code conversion. To convert a schema or code, you use **AWS SCT**, which automates conversion of a database or data-warehouse schema from one engine to another and supports application-SQL conversion. It converts source **views, stored procedures, and functions**, and clearly marks any object that must be converted manually. AWS SCT runs on **Windows, macOS, Fedora, and Ubuntu** and uses **SSL** connections to both source and target engines.

TABLE 6.7 Conversions supported by AWS SCT

SOURCE DATABASE	TARGET DATABASE ON AMAZON RDS
Oracle Database	Amazon Aurora, MySQL, PostgreSQL, MariaDB
Oracle Data Warehouse	Amazon Redshift
Microsoft SQL Server	Amazon Aurora, Amazon Redshift, MySQL, PostgreSQL, MariaDB
Teradata	Amazon Redshift
IBM Netezza	Amazon Redshift
Greenplum	Amazon Redshift
Vertica	Amazon Redshift
MySQL and MariaDB	PostgreSQL
PostgreSQL	Amazon Aurora, MySQL, MariaDB
Amazon Aurora	PostgreSQL
IBM Db2 LUW	Amazon Aurora, MySQL, PostgreSQL
Apache Cassandra	Amazon DynamoDB

The migration is a two-step process. First, use **AWS SCT** to convert the source schema to the target schema. Then, once the destination schema is built, use **AWS DMS** to load and replicate the data. After the migration finishes, switch the applications over to the target database. Plan the migration steps around your **availability requirements**.

> COMMON PITFALL
>
> Keep the division of labor straight: **AWS SCT converts** (schema, code, views, procedures, functions, application SQL); **AWS DMS moves and replicates the data** and creates only the objects it needs to do so. On a migration question, if the scenario is about translating a schema between engines, the answer is SCT; if it is about moving or continuously replicating data, the answer is DMS.

6.6 Choosing the right service in practice

The module closes with applied practice. A useful **test-taking strategy** is to isolate the **key words** in a scenario, because they rule out distractors and point to the answer. In the module's FinTech scenario, a bank wants to **graph connected data** about its customers' financial behaviors — the word *graph* (connected data) points past Aurora, DynamoDB, ElastiCache, and Redshift to the purpose-built graph database.

> WORKED SCENARIO – THE GRAPH KEYWORD
>
> A FinTech bank, entirely in the cloud, wants to link customer profiles and offer visualizations of financial habits, and asks which service to prioritize to **graph connected data on customers' financial behaviors**. The answer is **Amazon Neptune** — a managed graph database built to navigate highly connected data, exactly the use case behind recommendations, fraud detection, and social feeds. Aurora and DynamoDB store the transactional data, ElastiCache caches, and Redshift analyzes — but only Neptune graphs the relationships.

PROJECT SCENARIO – MOM & POP CAFÉ TO AMAZON RDS

The Mom & Pop Café keeps its order-history database on a **single Amazon EC2 instance** — a single point of failure that could shut the business down. The recommended remediation is to migrate that database to **Amazon RDS**, gaining a resilient Multi-AZ architecture, easier backups, and cost savings from eliminating the manual labor of upgrades. In Activity 6 you migrate the café's **MariaDB (MySQL)** database off a T2-small **LAMP** (Linux, Apache, MariaDB, PHP) instance and onto an RDS instance placed in **two private subnets across two AZs**, fronted by a **DB subnet group** and a security group whose inbound rule allows **MySQL/Aurora on TCP 3306** from the application's security group.

SIX BENEFITS OF AMAZON RDS (WHY THE CAFÉ MIGRATES)

- Easy to administer
- Highly scalable
- Available and durable
- Fast
- Secure
- Inexpensive

Module summary

- AWS databases are fully managed and purpose-built; match the workload — data shape, size and compute, access patterns, and performance — to the service, and host unsupported engines or versions on EC2.
- SQL is relational: rows and columns, fixed schema, SQL queries, vertical scaling. NoSQL is non-relational: key-value/document/graph, dynamic schema, document-collection queries, horizontal scaling — and *not only SQL*.
- The service map: RDS and Aurora (relational, transactional), Redshift (data warehouse/analytics), DynamoDB (NoSQL, internet-scale), Neptune (graph), ElastiCache (in-memory cache).
- Amazon RDS is managed relational with six engines; the instance class sets compute and the storage type sets storage, size, and IOPS; automated backups plus incremental and manual snapshots protect the data.
- RDS availability comes from a Multi-AZ synchronous standby with automatic, no-data-loss failover via the DNS endpoint; read scaling comes from asynchronous read replicas. Changing the instance class needs downtime; scaling storage does not.
- Amazon Aurora is MySQL/PostgreSQL-compatible (5x/3x throughput), stores six data copies across three AZs, runs a DB cluster of one primary plus up to 15 read replicas over a shared cluster volume, and offers a Serverless option.
- Amazon DynamoDB is a NoSQL document and key-value store: tables hold items hold attributes; a partition key (with an optional sort key) forms the primary key; global tables replicate across Regions; single-digit-millisecond latency and IAM access control.
- AWS DMS migrates databases into AWS — homogeneous or heterogeneous, one-time or continuous — with minimal/zero downtime, requiring at least one AWS endpoint; AWS SCT converts schema and code, and the two-step process runs SCT then DMS.

Review questions

1. **Name the AWS service best fit for each: a data warehouse for analytics, an in-memory cache, and a graph database.**

   Answer: Amazon Redshift for the data warehouse, Amazon ElastiCache for the in-memory cache, and Amazon Neptune for the graph database.

2. **Contrast the Multi-AZ standby and read replicas in Amazon RDS.**

   Answer: The Multi-AZ standby is a synchronously replicated copy in a second AZ for high availability — it serves no reads, loses no data, and is automatically promoted to primary on failover. Read replicas are asynchronously replicated read-only copies that scale read-heavy workloads beyond a single instance while the primary keeps handling reads and writes.

3. **Which RDS modifications require downtime, and which do not?**

   Answer: Changing the instance class requires downtime; increasing the allocated storage capacity happens without downtime.

4. **Describe Amazon Aurora's storage-level high availability.**

   Answer: Aurora uses a DB cluster with a shared cluster volume that spans AZs and stores six copies of the data across three Availability Zones. The cluster has one primary instance (read/write) and up to 15 Aurora replicas (read-only) that distribute reads and, across AZs, raise availability.

5. **Explain DynamoDB's table/item/attribute model and its two primary-key types.**

   Answer: A table is a collection of items (like rows); an item is a collection of attributes (like columns). The primary key is either a partition (simple) key of one attribute, or a partition-and-sort (composite) key of two — the partition key value selects the partition, and the sort key orders items within it.

6. **What must be true of every AWS DMS migration, and what conversion can DMS not do?**

   Answer: At least one endpoint must be on an AWS service — on-premises to on-premises is unsupported. DMS creates only the minimal objects needed to move the data and does not convert schema or code; that is the job of AWS SCT, which converts schema, code, and application SQL.

7. **A FinTech application must graph connected data about customers' financial behavior. Which service, and why?**

   Answer: Amazon Neptune — a managed graph database purpose-built to navigate highly connected datasets, the same pattern behind recommendations, fraud detection, and social feeds. The keyword graph rules out the relational, NoSQL key-value, cache, and warehouse options.