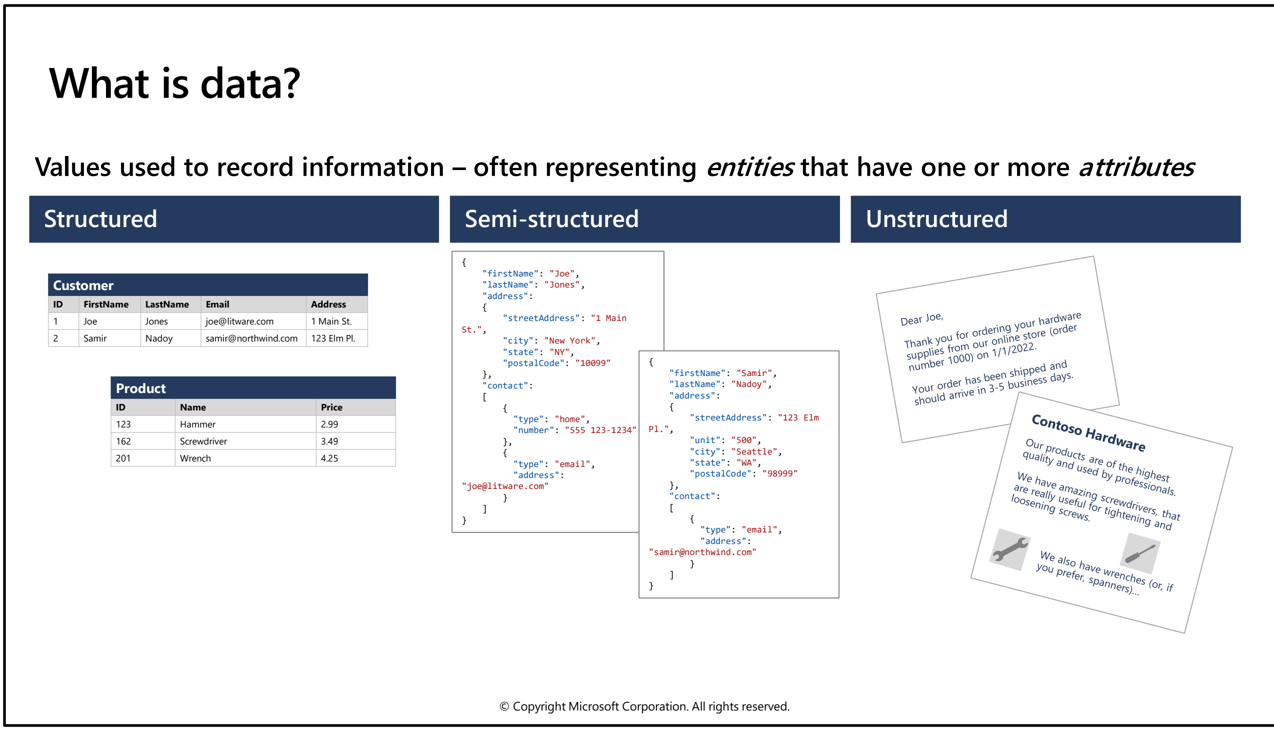
Microsoft Azure

Core Data Concepts & Azure Data Services

1: Core data concepts

The goal of this section is to introduce some fundamental, high-level concepts; which we'll explore in greater depth later in the course. Don't spend too much time getting into the details here.



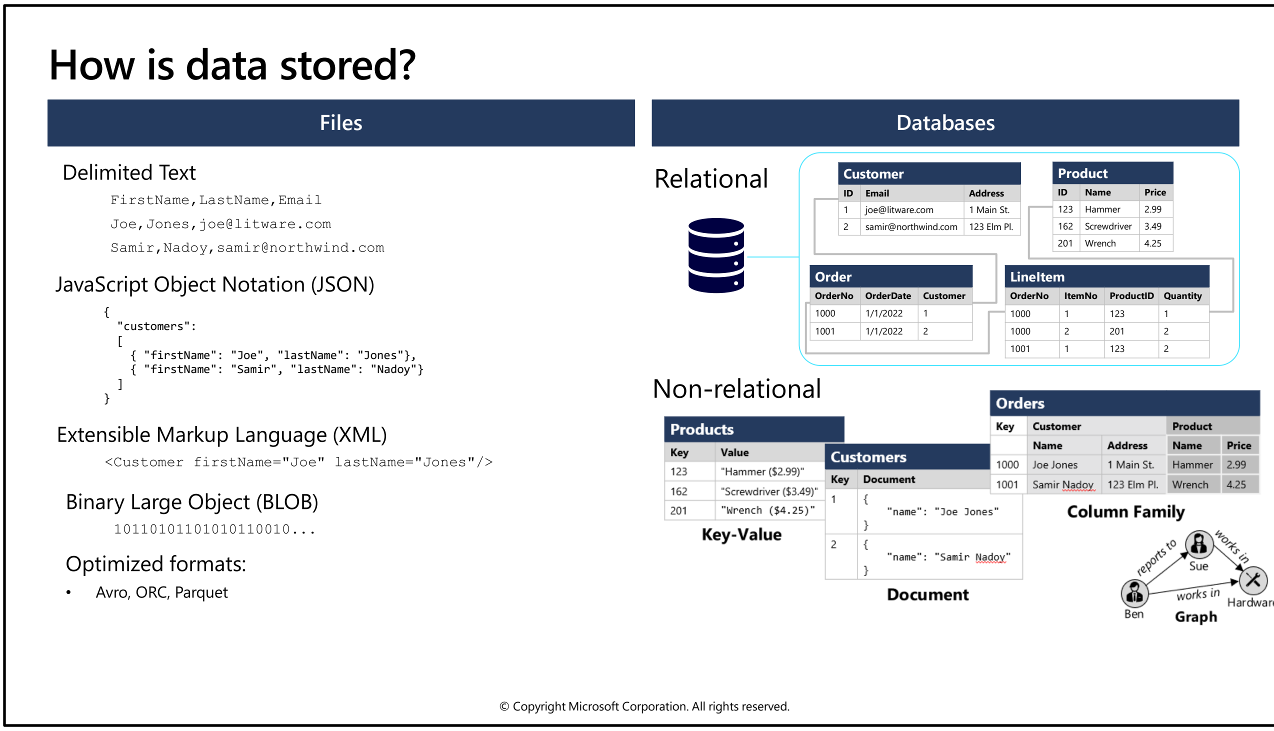
Data is a collection of facts such as numbers, descriptions, and observations used to record information. Data structures in which this data is organized often represents *entities* that are important to an organization (such as customers, products, sales orders, and so on). Each entity typically has one or more *attributes*, or characteristics (for example, a customer might have a name, an address, a phone number, and so on).

You can classify data as *structured*, *semi-structured*, or *unstructured*.

- Structured data is data that adheres to a fixed schema, so all of the data has the same fields or properties. Structured data is often stored in database tables with rows and columns, and multiple tables can reference one another by using key values in a *relational* model.
- Semi-structured data is information that has some structure, but which allows for some variation

```
between entity instances. For example, while
most customers may have an email address,
might have multiple email addresses, and some
might have none at all.
```

- **Unstructured** data is any data that is stored with no schema to organize discrete values. Examples include documents, free-form text, images, videos, audio streams, etc.



Files

> animated slide, click to proceed

File formats for data include:

- Delimited text: Data is stored in plain text format with specific field delimiters and row terminators. The most common format for delimited data is *comma-separated values (CSV)* in which fields are separated by commas, and rows are terminated by a carriage return / new line. Optionally, the first line may include the field names. Other common formats include *tab-separated values (TSV)* and *space-delimited* (in which tabs or spaces are used to separate fields), and *fixed-width* data in which each field is allocated a fixed number of characters. Delimited text is a good choice for structured data.
- JavaScript Object Notation (JSON): A hierarchical document schema is used to define data entities (*objects*) that have multiple *attributes*. Each attribute might be an object (or a collection of objects); making JSON a very flexible format that's good for both structured and semi-structured data.
- Extensible Markup Language (XML): XML is a text-based format that defines data entities and their attributes using markup *tags*. XML was a commonly used format in the early 2000's, but the increasing popularity of JSON has reduced its prevalence. For example:

  <CustomerEmail FirstName="Joe" LastName="Jones">joe@litware.com</Customer>
- Binary Large Object (BLOB): BLOB is the term used to describe binary data. Technically, all files are BLOBs (as ultimately, all files are stored as bits); but plain text formats like CSV and JSON, store binary values that map to specific text characters based on a set of ASCII or UNICODE codes; and can be opened and read by humans. Other files such as Word documents, PDFs, images, audio or video streams, and so on use a binary format that can only be interpreted by compatible software applications. In Azure, unstructured data is usually stored as a *block blob* file – a format that supports basic read and write operations.
- Optimized formats: As the volume of data that organizations need to work with has grown, a number of data formats that include features to enable metadata, compression, indexing, and other optimization techniques for specific types of workload have been created and are in common use. These include:
  - **Avro**: *Avro* is a row-based format. It was created by Apache. Each record contains a header that describes the structure of the data in the record. This header is stored as JSON. The data is stored as binary information. An application uses the information in the header to parse the binary data and extract the fields it contains. Avro is a very good format for compressing data and minimizing

storage and network bandwidth requirements.

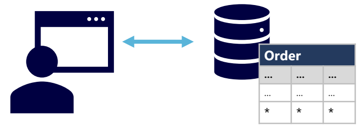
- **Optimized Row Columnar (ORC)**: ORC, organizes data into columns rather than rows. It was developed by HortonWorks for optimizing read and write operations in Apache Hive. Hive is a data warehouse system that supports fast data summarization and querying over very large datasets. Hive supports SQL-like queries over unstructured data. An ORC file contains *stripes* of data. Each stripe holds the data for a column or set of columns. A stripe contains an index into the rows in the stripe, the data for each row, and a footer that holds statistical information (count, sum, max, min, and so on) for each column.
- Parquet: Parquet is a columnar format created by Cloudera and Twitter. Data for each column is stored together in the same *row group*. Each row group contains one or more chunks of data. A Parquet file includes metadata that describes the set of rows found in each chunk. An application can use this metadata to quickly locate the correct chunk for a given set of rows and retrieve the data in the specified columns for these rows. Parquet specializes in storing and processing nested data types efficiently. It supports very efficient compression and encoding schemes.

Databases

A database is used to define a central system in which data can be stored and queried. In a simplistic sense, the file system on which files are stored is a kind of database; but when we use the term in a professional data context, we usually mean a dedicated system for managing data *records* rather than files.

- Relational Databases are commonly used to store and query structured data. The data is stored in tables that represent *entities*, such as customers, products, or sales orders. Each instance of an entity is assigned a *primary key* that uniquely identifies it; and these keys are used to reference the entity instance in other tables. For example, a customer's primary key can be referenced in a sales order record to indicate which customer placed the order. This use of keys to reference data entities enables a relational database to be *normalized*; in other words, it eliminates duplication of data values – the details of an individual customer are stored only once; not for each sales order the customer places. The tables are managed and queried using Structured Query Language (SQL) – which is based on an ANSII standard, so it's similar across multiple database systems
- Non-Relational databases are data management systems that do not apply a relational schema to the data. Common types of non-relational database include *key-value* stores, in which each record consists of a unique key and an associated value, which can be in any format; *document* databases, which are a specific form of key-value database in which the value is a JSON document (which the system is optimized to query), ***Column family databases***, which store tabular data comprising rows and columns but you can divide the columns into groups known as column-families. Each column family holds a set of columns that are logically related together, and *graph* databases, which store entities as *nodes* with links to define relationships between them. Non-relational databases are often referred to as *NoSQL* database, even though some support a variant of the SQL language.

Transactional data workloads



Data is stored in a database that is optimized for *online transactional processing* (OLTP) operations that support applications

A mix of *read* and *write* activity

For example:

- Read the *Product* table to display a catalog
- Write to the *Order* table to record a purchase

Data is stored using *transactions*

Transactions are "ACID" based:

- **Atomicity –** each transaction is treated as a single unit of work, which succeeds completely or fails completely
- **Consistency** – transactions can only take the data in the database from one valid state to another
- **Isolation** – concurrent transactions cannot interfere with one another
- **Durability –** when a transaction has succeeded, the data changes are persisted in the database

OLTP is typically a *live* system in which data storage is optimized for both *read* and *write* operations in order to support *transactional* workloads

Updates are made transactionally, for example, an order is placed. Each individual order is stored in the OLTP system.

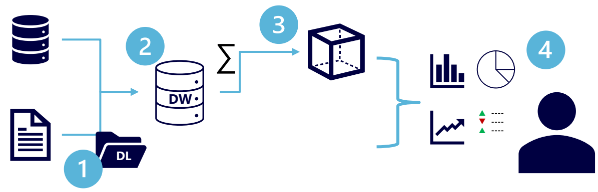
Transactions support ACID semantics:

- **Atomicity** – each transaction is treated as a single unit, which success completely or fails completely. For example, a transaction that involved debiting funds from one account and crediting the same amount to another account must complete both actions. If either action can't be completed, then the other action must fail.
- **Consistency** – transactions can only take the data in the database from one valid state to another. To continue the debit and credit

example above, the completed state of the transaction must reflect the transfer of funds from one account to the other.

- **Isolation** – concurrent transactions cannot interfere with one another and must result in a consistent database state. For example, while the transaction to transfer funds from one account to another is in-process, another transaction that checks the balance of these accounts must return consistent results - the balance-checking transaction can't retrieve a value for one account that reflects the balance *before* the transfer, and a value for the other account that reflects the balance *after* the transfer.
- **Durability** – when a transaction has been committed, it will remain committed. After the account transfer transaction has completed, the revised account balances are persisted so that even if the database system were to be switched off, the committed transaction would be reflected when it is switched on again.

Analytical data workloads



1. Data files may be stored in a central *data lake* for analysis
2. An extract, transform, and load (ETL) process copies data from files and OLTP databases into a *data warehouse* that is optimized for *read* activity
3. Data in the data warehouse may be aggregated and loaded into an online analytical processing (OLAP) model, or *cube*
4. The data in the data lake, data warehouse, and analytical model can be queried to produce reports and dashboards

- Analytical workloads are typically read-only systems that store vast volumes of historical data or business metrics
- Analytics can be based on a snapshot of the data at a given point in time, or a series of snapshots.
- An example of analytical information is a report on monthly sales.

Data lakes are common in *large-scale data warehousing* scenarios, where a large volume of non-relational data must be collected and analyzed.

Data warehouses are an established way to store data in a relational schema that is optimized for read operations – primarily queries to support reporting and data visualization. The data warehouse schema may require some *denormalization* of data in an OLTP data source (introducing some duplication to make queries perform faster)

Online Analytical Processing (OLAP) is an aggregated type of data storage that is optimized for analytical workloads. Data is imported aggregated so that aggregations of numeric *facts* (for example, sales revenue, or number of items sold) can be pre-calculated across *dimensions* (for example, product, date, or geographic location). The aggregation will typically occur at different levels allowing you to drill down or up, for example to find total sales by region, by city, or by an individual address. Because OLAP data is aggregated periodically, once the aggregation has been performed, queries which need the summaries that it contains are very fast.

Note: Different types of user might perform data analytical work at different stages of the overall architecture. For example:

- Data scientists might work directly with data files in a data lake to explore and model data.
- Data Analysts might query tables directly in the data warehouse to produce complex reports and visualizations.
- Business users might consume pre-aggregated data in an analytical model (cube) in the form of reports or dashboards.

1: Knowledge check

How is data in a relational table organized?

- [x] Rows and Columns
- [ ] Header and Footer
- [ ] Pages and Paragraphs

Which of the following is an example of unstructured data?

- [ ] A comma-delimited text file with *EmployeeID*, *EmployeeName*, and *EmployeeDesignation* fields
- [x] Audio and Video files
- [ ] A table within relational database

What is a data warehouse?

- [ ] A non-relational database optimized for read and write operations
- [x] A relational database optimized for read operations
- [ ] A storage location for unstructured data files

Allow students a few minutes to think about the questions, and then use the animated slide to reveal the correct answers.

2: Data roles and services

Data professional roles

Database Administrator

Database provisioning, configuration and management

Database security and user access

Database backups and resiliency

Database performance monitoring and optimization

Data Engineer

Data integration pipelines and ETL processes

Data cleansing and transformation

Analytical data store schemas and data loads

Data Analyst

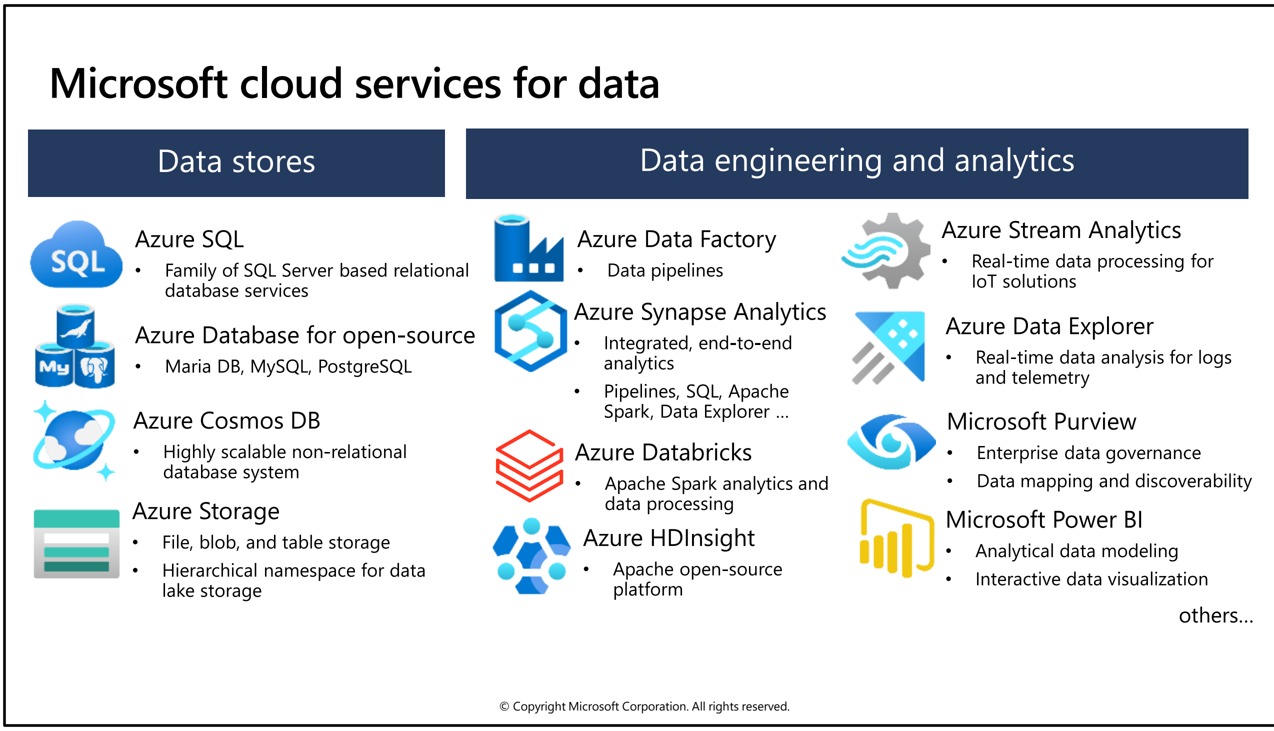
Analytical modeling

Data reporting and summarization

Data visualization

Discuss the three roles that are listed, but consider that in real-world scenarios, actual job roles might be a combination of these roles, or a subset of a single role, based on the size of the organization.

Note also that there are additional data-related roles not mentioned here, such as data scientist and data architect; and that there are other technical professionals that work with data, including application developers and software engineers. We're focusing on these three roles because they represent the core data-related operations in most organizations and reflect common job titles for data professionals.



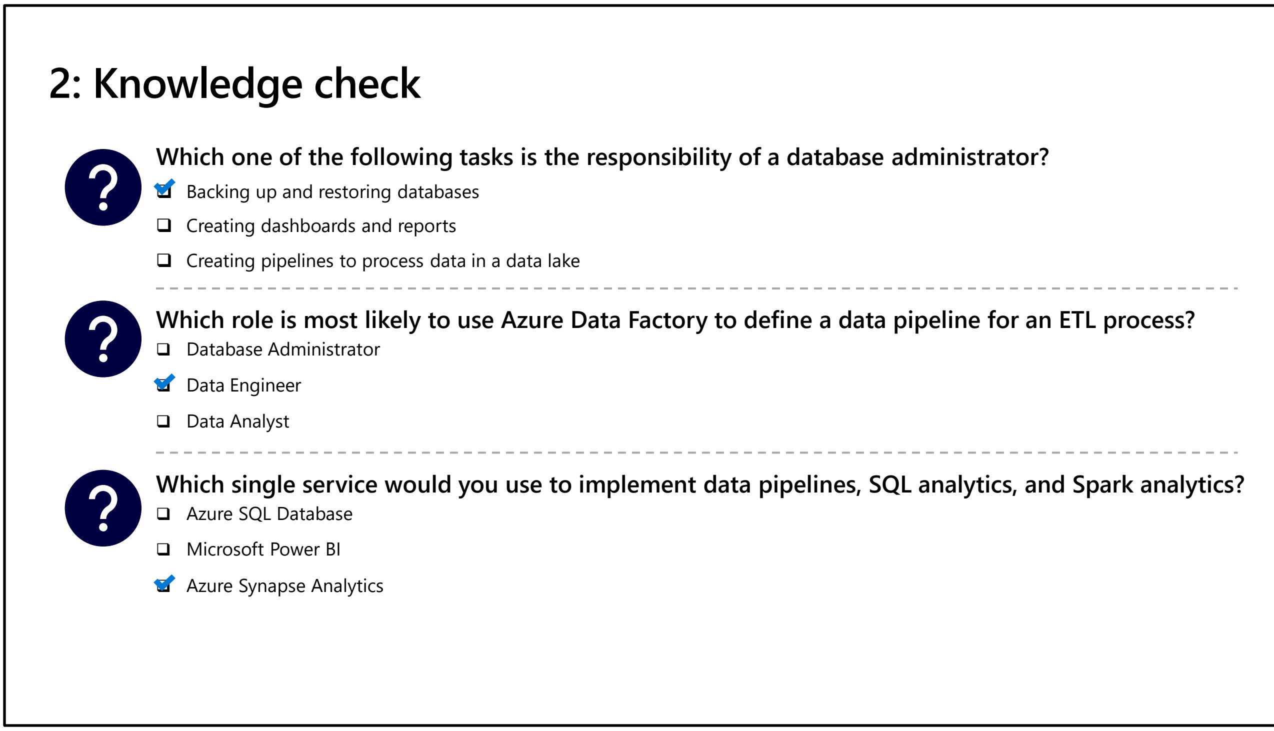
The slide shows some of the most commonly used service for working with data. The list is not exhaustive.

*We'll explore many of these later in the course, so **don't spend a lot of time explaining the features of each individual service in detail**.*

Azure SQL covers a family of relational database solutions based on the Microsoft SQL Server database engine. Options include:

- Azure SQL Database – a fully managed platform-as-a-service (PaaS) database hosted in Azure
- Azure SQL Managed Instance – a hosted instance of SQL Server, which allows more flexible configuration than Azure SQL DB but with more administrative responsibility for the owner.
- Azure SQL VM – a virtual machine with an installation of SQL Server, allowing maximum configurability with full management responsibility.

Note that some services are not easily categorized – for example, Azure Synapse Analytics includes some of the data pipeline processing capabilities of Azure Data Factory, a SQL Server based relational database engine that is optimized for data warehousing, and a Spark processing engine that offers similar functionality to Azure Databricks (Spark is an Apache open source technology for processing large volumes of data in parallel using programming languages like Scala and Python).



Allow students a few minutes to think about the questions, and then use the animated slide to reveal the correct answers.

3: Explore relational data concepts

Relational tables

Data is stored in tables

Tables consists of rows and columns

All rows have the same columns

Each column is assigned a datatype

Customer

ID	FirstName	MiddleName	LastName	Email	Address	City
1	Joe	David	Jones	joe@litware.com	1 Main St.	Seattle
2	Samir		Nadoy	samir@northwind.com	123 Elm Pl.	New York

Product

ID	Name	Price
123	Hammer	2.99
162	Screwdriver	3.49
201	Wrench	4.25

Order

OrderNo	OrderDate	Customer
1000	1/1/2022	1
1001	1/1/2022	2

LineItem

OrderNo	ItemNo	ProductID	Quantity
1000	1	123	1
1000	2	201	2
1001	1	123	2

In a relational database schema, data is stored in tables; which consist of rows and columns. Relational tables are a format for *structured* data, and each row in a table has the same columns; though in some cases, not all columns need to have a value – for example, a customer table might include a **MiddleName** column; which can be empty (or *NULL*) for rows that represent customers with no middle name or whose middle name is unknown).

Each column stores data of a specific *datatype*. For example, An **Email** column in a **Customer** table would likely be defined to store character-based (text) data (which might be fixed or variable in length), a **Price** column in a **Product** table might be defined to store decimal numeric data, while a **Quantity** column in an **Order** table might be constrained to integer numeric values; and an **OrderDate** column in the same **Order** table would be defined to store date/time values. The available datatypes that you can use when defining a table depend on the database system you are using; though there are standard datatypes defined by the American National Standards Institute (ANSI) that are supported by most database systems.

Normalization

Sales Data

OrderNo	OrderDate	Customer	Product	Quantity
1000	1/1/2022	Joe Jones, 1 Main St, Seattle	Hammer ($2.99)	1
1000	1/1/2022	Joe Jones- 1 Main St, Seattle	Screwdriver ($3.49)	2
1001	1/1/2022	Samir Nadoy, 123 Elm Pl, New York	Hammer ($2.99)	2
…	…	…	…	…

- Separate each *entity* into its own table
- Separate each discrete *attribute* into its own column
- Uniquely identify each entity instance (row) using a *primary key*
- Use *foreign key* columns to link related entities

Customer

ID	FirstName	LastName	Address	City
1	Joe	Jones	1 Main St.	Seattle
2	Samir	Nadoy	123 Elm Pl.	New York

Product

ID	Name	Price
123	Hammer	2.99
162	Screwdriver	3.49
201	Wrench	4.25

Order

OrderNo	OrderDate	Customer
1000	1/1/2022	1
1001	1/1/2022	2

LineItem

OrderNo	ItemNo	ProductID	Quantity
1000	1	123	1
1000	2	201	2
1001	1	123	2

Don't get bogged down in details of 1st, 2nd, 3rd, 4th, etc, normal form for this audience. The essential learning point is that normalization is commonly used in relational databases to separate data for each entity into multiple related tables, minimizing duplication of data values and enforcing data integrity through specific data types for each piece of data and referential integrity (for example to ensure that orders only reference valid customers).

Normalization is a term used by database professionals for a schema design process that minimizes data duplication and enforces data integrity.

To understand the core principles of normalization, suppose the table on the left of the slide represents a spreadsheet that a company uses to track its sales. Notice that the customer and product details are duplicated for each individual item sold; and that the customer name and postal address, and the product name and price are combined in the same spreadsheet cells.

Now look at how normalization has changed the way the data is stored. Each *entity* that is represented in the data (customer, product, sales order, and line item) is stored in its own table, and each discrete attribute of those entities is in its own column. Instances of each entity are uniquely identified by an ID or other key value, and when one entity references another (for example, an order has an associated customer), the primary key of the related entity is stored as a foreign key – so we can look up the address of the customer (which is stored only once) for each record in the Order table by referencing the corresponding record in the Customer table. Typically, a relational database management system (RDBMS) can enforce *referential integrity* to ensure that a value entered into a foreign key field has an existing corresponding primary key in the related table – for example, preventing orders for non-existent customers.

Structured Query Language (SQL)

SQL is a standard language for use with relational databases

Standards are maintained by ANSI and ISO

Most RDBMS systems support proprietary extensions of standard SQL

Data Definition Language (DDL)	Data Control Language (DCL)	Data Manipulation Language (DML)
CREATE, ALTER, DROP, RENAME	*GRANT, DENY, REVOKE*	*INSERT, UPDATE, DELETE, SELECT*

Data Definition Language (DDL)

```
CREATE TABLE Product
(
  ProductID INT PRIMARY KEY,
  Name VARCHAR(20) NOT NULL,
  Price DECIMAL NULL
);
```

Product

ID	Name	Price

Data Control Language (DCL)

```
GRANT SELECT, INSERT, UPDATE
ON Product
TO user1;
```

Product

ID	Name	Price
123	Hammer	2.99
162	Screwdriv	3.49
201	Wrench	4.25

Data Manipulation Language (DML)

```
SELECT Name, Price
FROM Product
WHERE Price > 2.50
ORDER BY Price;
```

Results

Name	Price
Hammer	2.99
Screwdriver	3.49
Wrench	4.25

The goal of this topic is *<u>not</u>* *to teach students how to write SQL queries; but rather to help them understand that SQL is a standard language used to define and work with relational data structures in a database, and to differentiate between the three common kinds of SQL statements to manage database object definitions, control access, and manipulate data.*

SQL is a standard language for working with relational databases, with syntax standards that are maintained by the American National Standards Institute (ANSI) and International Standards Organization (ISO). Most relational database systems (RDBMS) vendors extend the standard language with some proprietary syntax – for example Transact-SQL / T-SQL (Microsoft SQL Server based systems), PL/SQL (Oracle), and pgSQL (PostgreSQL).

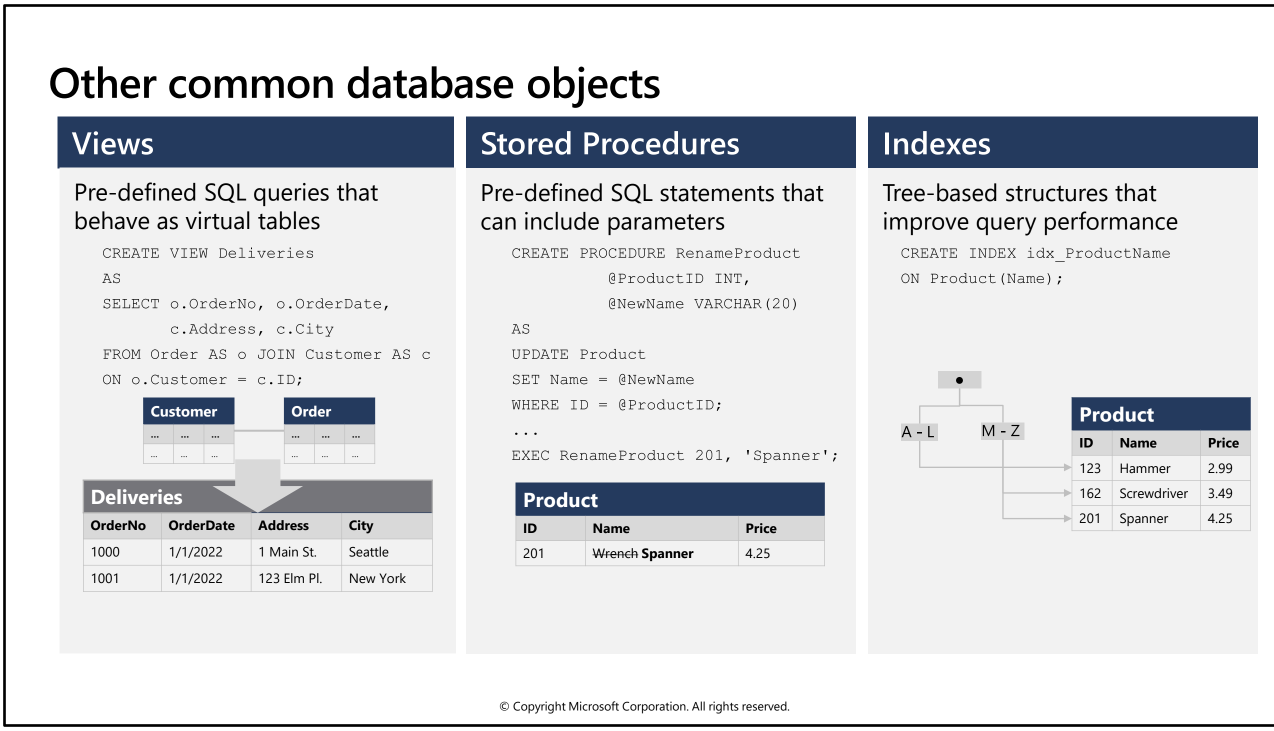
There are three broad types of SQL statements that can be used in a database system:

- Data Definition Language (DDL) is used to manage objects such as tables in the database. For example, you can CREATE new objects, and ALTER or DROP existing objects. The example on the slide shows a CREATE TABLE statement used to create a new, empty table named Product.
- Data Control Language (DCL) is used to manage access to objects in a database. You can GRANT, DENY, or REVOKE specific permissions for specific users (and groups of users). The example on the slide grants user1 permission to use SELECT, INSERT, and UPDATE statements on the Product table.
- Data Manipulation Language (DML) is the most commonly used type of SQL, and is generally used to INSERT, UPDATE, DELETE, or SELECT data in tables. The example on the slide assumes that some data has previously been inserted into the Product table, and shows the results returned by a SELECT query that retrieves the name and price of all products with a price greater than 2.50, sorted in order of price.

This slide shows a core set of SQL statements and examples. The SQL language is extensive, and there are other statements not shown here. Additionally, the syntax for the statements that are shown here can be much more complex than these simple examples.

If students are interested in exploring SQL beyond this data fundamentals course, recommend they attend course DP-080: Querying Data with Microsoft Transact-SQL (details at https://docs.microsoft.com/learn/certifications/courses/dp-080t00) or review the ***Get Started Querying with***

Transact-SQL *learning path on Microsoft Learn at* ***https://docs.microsoft.com/learn/paths/get-started-querying-with-transact-sql/****.*



>Animated slide, click to proceed

Don't go into great detail about the implementation of these objects. The key learning point is to be aware at a high-level of some of the common types of object found in a database other than tables.

In addition to tables, databases can contain other kinds of object that enable you to work with data.

- Views are pre-defined SQL SELECT queries that return a tabular dataset. Views behave as virtual tables, and can themselves be queried using SELECT statements, just like tables. They're often used to abstract the normalized schema of the database to encapsulate data from one or more tables.
- Stored Procedures are pre-defined SQL statements that can be run on-demand. They can be parameterized, and are often used to encapsulate data operations to insert, delete, or update records for data entitites.
- Indexes are tree-based structures that enable the database query engine to find individual records based on specific column values more quickly than if they just read the entire table.

These types of database object, and others, enable you to build a comprehensive relational database that applications can use to store, manage, and retrieve details of entities efficiently and securely.

3: Knowledge check

Which one of the following statements is a characteristic of a relational database?

- [ ] All columns in a table must be of the same data type
- [x] A row in a table represents a single instance of an entity
- [ ] Rows in the same table can contain different columns

Which SQL statement is used to query tables and return data?

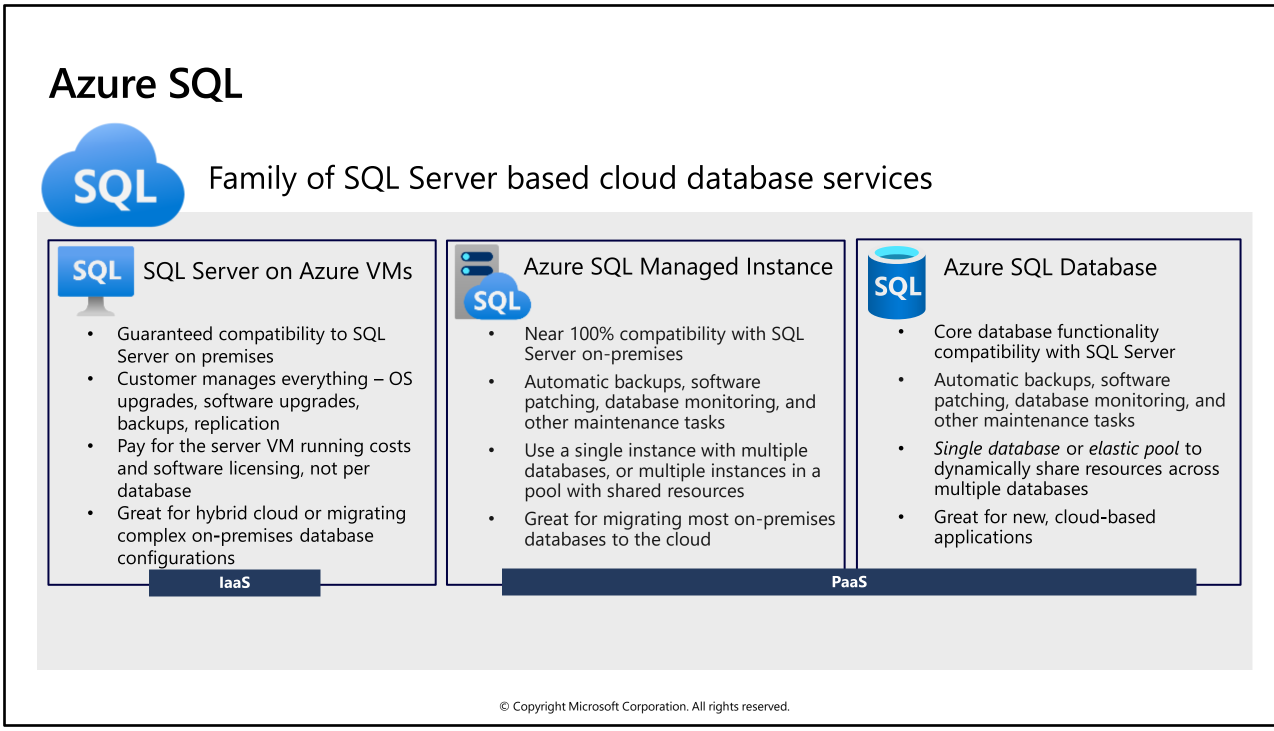
- [ ] QUERY
- [ ] READ
- [x] SELECT

What is an index?

- [x] A structure that enables queries to locate rows in a table quickly
- [ ] A virtual table based on the results of a query
- [ ] A pre-defined SQL statement that modifies data

Allow students a few minutes to think about the questions, and then use the animated slide to reveal the correct answers.

4: Explore Azure services for relational data



Note that this slide does not cover Azure SQL Edge, which is a SQL Server-based service for edge computing – predominantly for Internet-of-things (IoT) scenarios.

Azure SQL is the generic term used to describe a family of Azure relational database services that are based on Microsoft SQL Server. SQL Server is an industry-leading relational database management system (RDBMS) that is used in on-premises solutions by some of the biggest organizations in the world. The Azure SQL services are based on the same database engine, making them a great solution for organizations that want to migrate existing on-premises databases to the cloud; as well as new applications that are designed as cloud-based from conception.

- **SQL Server on Azure Virtual Machines** is an infrastructure-as-a-service (IaaS) solution in which a full instance of SQL Server is installed in a virtual machine that is hosted in Azure. This makes it a good candidate for migration projects, where 1:1 compatibility with an existing on-premises SQL Server instance is required or for hybrid scenarios with a mix of cloud-based and on-premises databases that must maintain compatibility. Because it's an IaaS solution, you have full control of the configuration of the database; which also means you have responsibility to manage administrative tasks – just as you would for a SQL Server instance in your own data center. Costs for the service are based on SQL Server licensing and the cost of running the VM in Azure.
- **Azure SQL Managed Instance** is a platform-as-a-service (PaaS) service that enables you to pre-provision compute resources and deploy several individual SQL Server managed instances up to your pre-provisioned compute level. Core administrative tasks are automated while providing a high-degree of compatibility with on-premises SQL Server. You can choose to deploy a single managed instance that supports multiple databases, or you can create a pool of instances that share underlying infrastructure resources for cost-efficiency. SQL Managed Instance is a great choice for most migration scenarios, where you need to move an on-premises SQL Server database to the cloud with minimal changes.
- **Azure SQL Database** is another platform-as-a-service (PaaS) solution that offers the lowest-cost Azure SQL option. You have minimal administrative control over the service beyond creating the database schema, importing and exporting data, and configuring access controls. Azure SQL Database enables you to deploy a single database or an *elastic pool* that shares resources across multiple databases. Azure SQL Database is a great choice for new applications that require a low-cost relational data store

with minimal administrative overhead.

The list is in decreasing order of administrative control/responsibility and cost. SQL Server on a VM is the most expensive option; but allows you greater control over server and database configuration. However, you also have full responsible for server maintenance and management. Azure SQL Database is the lowest cost option, but supports fewer configuration options, Most database maintenance other than access controls is automated for you. SQL Managed Instance offers a balance of cost, administrative control, and maintenance automation.

Managed solutions that do more for you

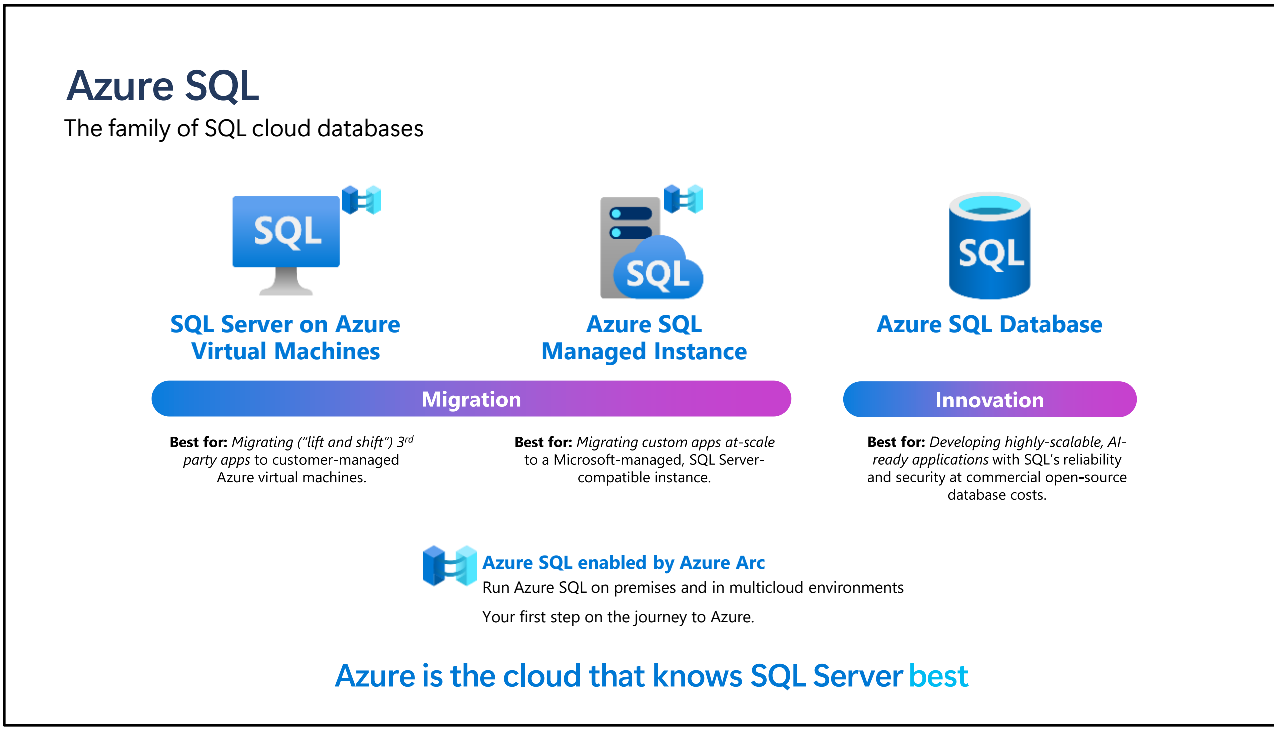
SQL Server on Azure VMs	Azure SQL Managed Instance	Azure SQL Database	Azure SQL Edge*
	Intelligent performance/security	Intelligent performance/security	Intelligent performance/security
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Database	Database	Database	Database
SQL instance-level features	SQL instance-level features		
High Availability /DR/Backups	High Availability/ DR/Backups	High Availability/ DR/Backups	High Availability/ DR/Backups
Database provision/ Patch/Scaling	Database provision/ Patch/Scaling	Database provision/ Patch/Scaling	Database provision/ Patch/Scaling
Operating system	Operating system	Operating system	Operating system (container)
Virtualization	Virtualization	Virtualization	Container Platform
Hardware	Hardware	Hardware	Hardware & Operating System
Datacenter management	Datacenter management	Datacenter management	Device management (IoT Hub)

Managed by customer

Managed by Microsoft

Machine learning capability

*in connected scenario



Azure SQL is a family of fully-managed, secure and intelligent SQL database services. Azure offers the widest range of deployment options for SQL from edge to cloud. With Azure SQL, you can rehost SQL workloads on SQL Server on Azure Virtual Machines, modernize existing applications with Azure SQL Managed Instance and support modern cloud applications with Azure SQL Database and Azure SQL Edge.

Azure SQL is built upon the same SQL Server engine, so you can migrate applications with ease and continue to use the tools, languages and resources you're familiar with. Your skills and experience transfer to the cloud and edge, so you can do even more with what you already have.

The services within Azure SQL support a variety of scenarios:

SQL Server on Azure Virtual Machines

Lift-and-shift your SQL workloads with ease and maintain with 100% SQL Server compatibility and operating system-level access

Azure SQL Managed Instance

Modernize your existing SQL Server applications at scale with an intelligent fully managed service

Azure SQL Database

Support modern cloud applications on an intelligent, managed service that includes serverless compute

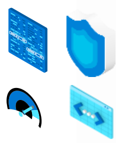
SQL Server on Azure VMs and SQL Managed Instance are also now Azure Arc enabled, allowing you to run these services on the infrastructure of your choice, when a hybrid approach is required.

Understanding Migration and Modernization

Applications | Data | Infrastructure

Migration (IaaS)

Drivers: Timelines, threats & business model



Expiring contracts,
security/resilience/scalability &
CAPEX to OPEX

Move workloads "as-is"

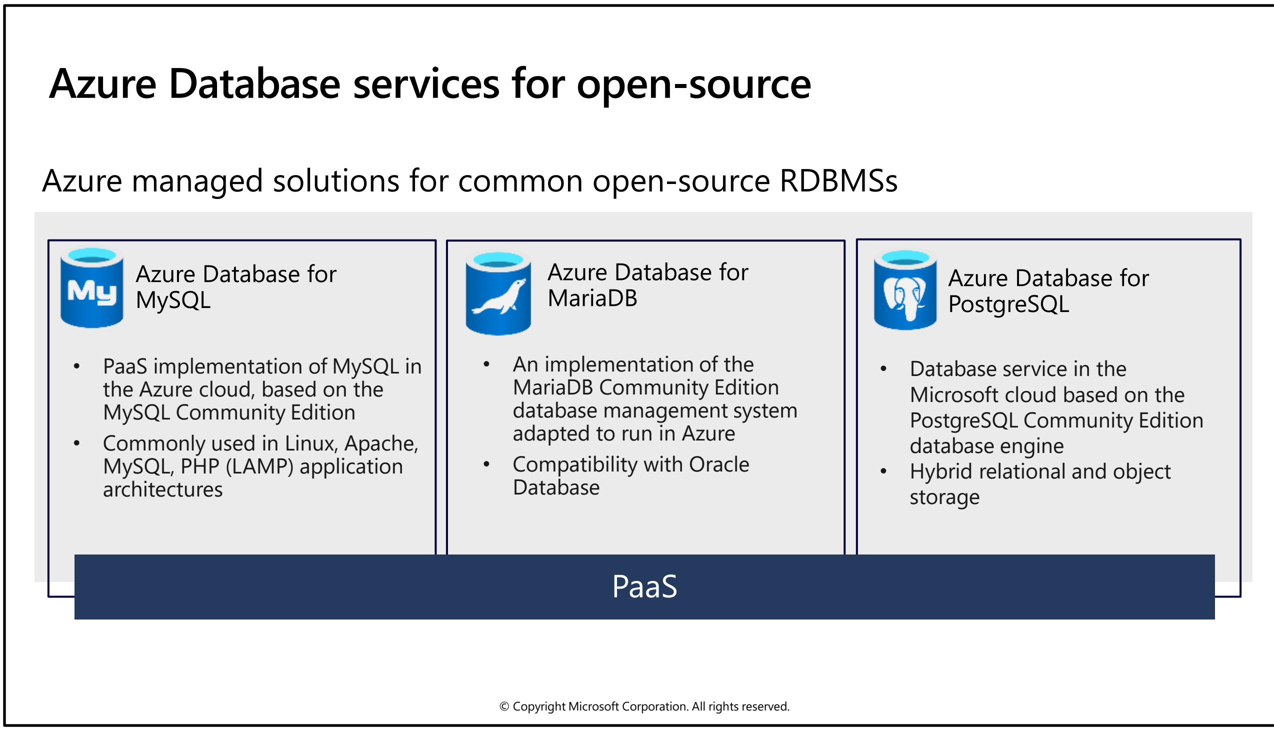
Modernization (PaaS/SaaS)

Drivers: Productivity, cost optimization &
business insights

Application Innovations(TTM),
reduced resources and lock-in,
data analytics (AI)

Update the application stack

Azure on your terms



MySQL started life as a simple-to-use open-source database management system. It is commonly used in *Linux, Apache, MySQL, and PHP* (LAMP) stack apps.

MariaDB is a newer database management system, created by the original developers of MySQL. The database e ngine has since been rewritten and optimized to improve performance. MariaDB offers compatibility with Oracle Database (another popular commercial database management system).

PostgreSQL is a hybrid relational-object database. You can store data in relational tables, but a PostgreSQL database also enables you to store cus tom data types, with their own non-relational properties.

4: Knowledge check

Which deployment option offers the best compatibility when migrating an existing SQL Server on-premises solution?

- [ ] Azure SQL Database (single database)
- [ ] Azure SQL Database (elastic pool)
- [x] Azure SQL Managed Instance

Which of the following statements is true about Azure SQL Database?

- [x] Most database maintenance tasks are automated
- [ ] You must purchase a SQL Server license
- [ ] It can only support one database

Which database service is the simplest option for migrating a LAMP application to Azure?

- [ ] Azure SQL Managed Instance
- [x] Azure Database for MySQL
- [ ] Azure Database for PostgreSQL

Allow students a few minutes to think about the questions, and then use the animated slide to reveal the correct answers.

5: Fundamentals of Azure Cosmos DB

Modern apps face new challenges

Managing and syncing data distributed around the globe

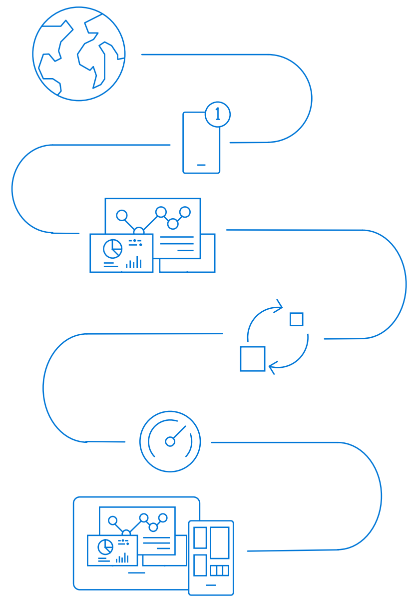
Delivering highly-responsive, real-time personalization

Processing and analyzing large, complex data

Scaling both throughput and storage based on global demand

Offering low-latency to global users

Modernizing existing apps and data



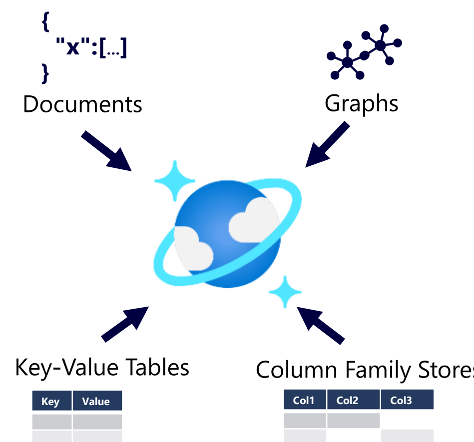
Azure Cosmos DB

A FULLY-MANAGED GLOBALLY DISTRIBUTED DATABASE SERVICE BUILT TO GUARANTEE EXTREMELY LOW LATENCY AND MASSIVE SCALE FOR MODERN APPS

What is Azure Cosmos DB?

A multi-model, global-scale *NoSQL* database management system

- Support for multiple storage APIs
- Real time access with fast read and write performance
- Enable *multi-region writes* to replicate data globally; enabling users in specified regions to work with a local replica



Relational databases store data in relational tables, but sometimes the structure imposed by this model can be too rigid, and often leads to poor performance unless you spend time implementing detailed tuning. Other models, collectively known as *NoSQL* databases exist. These models store data in other structures, such as documents, graphs, key-value stores, and column family stores.

Azure Cosmos DB supports multiple application programming interfaces (APIs) that enable developers to use the programming semantics of many common kinds of data store to work with data in a Cosmos DB database. The internal, document-based storage structure is abstracted, enabling developers to use Cosmos DB to store and query data using APIs with which they are already familiar.

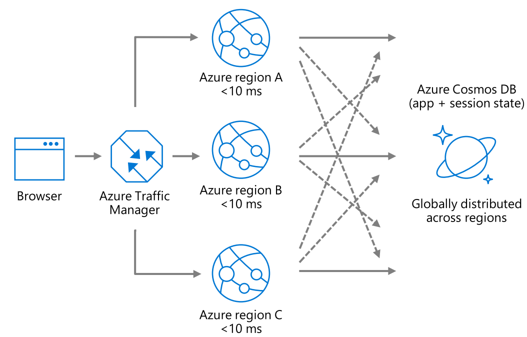
Cosmos DB uses indexes and partitioning to provide fast read and write performance and can scale to massive volumes of data.

You can enable multi-region writes, adding the Azure regions of your choice to your Cosmos DB account so that globally distributed users can each work with data in their local replica.

Data distributed and available globally

Put your data where your users are to give real-time access and uninterrupted service to customers anywhere in the world.

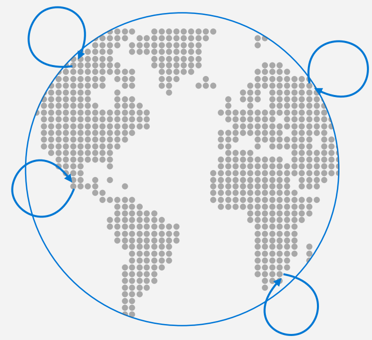
- Turnkey global data replication across all Azure regions
- Guaranteed low-latency experience for global users
- Resiliency for high availability and disaster recovery



GUARANTEED LOW LATENCY

PROVIDE USERS AROUND THE WORLD WITH FAST ACCESS TO DATA

Serve <10 ms read and <15 ms write requests at the 99th percentile from the region nearest to users, while delivering data globally.



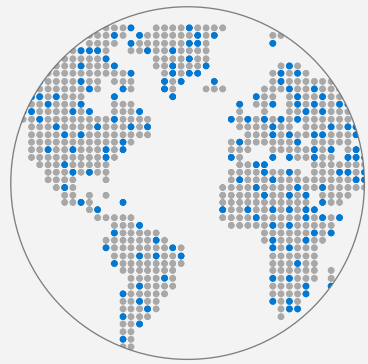
Single digit latency -> SLA

Turnkey global distribution

PUT YOUR DATA WHERE YOUR USERS ARE

Automatically replicate all your data around the world, and across more regions than Amazon and Google combined.

- Available in [all Azure regions]
- Manual and automatic failover
- Automatic & synchronous multi-region replication



Elastic Scale out -> Tunable Consistency

Small storage – large throughput (e.g. notification broadcast/poll)
Large storage – small throughput (e.g. classic data/log store)

Azure Cosmos DB APIs

Azure Cosmos DB for NoSQL

- Native API for Cosmos DB

```
SELECT *
FROM customers c
WHERE c.id = "joe@litware.com"
```

```
{
  "id": "joe@litware.com",
  "name": "Joe Jones",
  "address": {
    "street": "1 Main St.",
    "city": "Seattle"
  }
}
```

Azure Cosmos DB for MongoDB

- Compatibility with MongoDB

```
db.products.find({ id: 123})
```

```
{
  "id": 123,
  "name": "Hammer",
  "price": 2.99}
}
```

Azure Cosmos DB for PostgreSQL

- Compatibility with PostgreSQL

id	name	dept	manager
1	Sue Smith	Hardware	Joe Jones
2	Ben Chan	Hardware	Sue Smith

Azure Cosmos DB for Table

- Key-value storage API
- Compatible with Azure Table Storage

PartitionKey	RowKey	Name
1	123	Joe Jones
1	124	Samir Nadoy

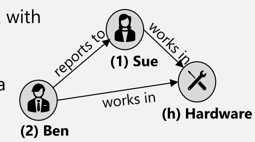
Azure Cosmos DB for Apache Cassandra

- Compatibility with Apache Cassandra

id	name	dept	manager
1	Sue Smith	Hardware	
2	Ben Chan	Hardware	Sue Smith

Azure Cosmos DB for Apache Gremlin

- Used to work with *graph* data
- *vertices* are connected via relationships (*edges*)



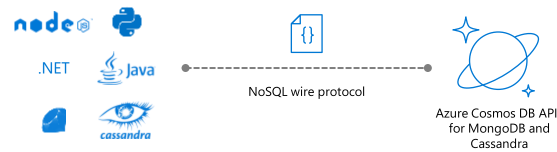
APIs supported in Azure Cosmos DB include:

- **Azure Cosmos DB for NoSQL**: The native API in Cosmos DB manages data in JSON document format, and uses SQL syntax to work with the data.
- **Azure Cosmos DB for MongoDB**: MongoDB is a popular open source database in which data is stored in Binary JSON (BSON) format. The Azure Cosmos DB MongoDB API enables developers to use MongoDB client libraries to and code to work with data in Azure Cosmos DB.
- **Azure Cosmos DB for PostgreSQL:** Azure Cosmos DB for PostgreSQL is a native PostgreSQL, globally distributed relational database that automatically shards data to help you build highly scalable apps.
- **Azure Cosmos DB for Table**: The Table API is used to work with data in key-value tables, similar to Azure Table Storage. The Azure Cosmos DB Table API offers greater scalability and performance than Azure Table Storage.
- Azure Cosmos DB for Apache Cassandra: The Cassandra API is compatible with Apache Cassandra, which is a popular open source database that uses a column-family storage structure. Column families are tables, similar to those in a relational database, with the exception that it's not mandatory for every row to have the same columns.
- Azure Cosmos DB for Apache Gremlin: The Gremlin API is used to with with data in a *graph* structure; in which entities are defined as *vertices* that form nodes in connected graph. Nodes are connected by *edges* that represent relationships. The example on the slide shows two kinds of vertex (employee and department) and edges that connect them (employee "Ben" *reports to* employee "Sue", and both employees *work in* the "Hardware" department).

Migrate nosql apps

Make data modernization easy with seamless lift and shift migration of NoSQL workloads to the cloud.

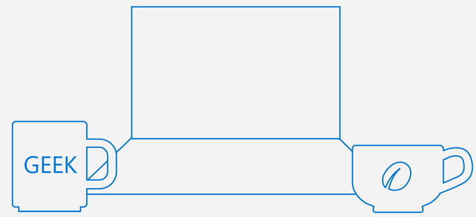
- Azure Cosmos DB APIs for MongoDB and Cassandra bring app data from anywhere to Azure Cosmos DB
- Leverage existing tools, drivers, and libraries, and continue using existing apps' current SDKs
- Turnkey geo-replication
- No infrastructure or VM management required



Handle any data with no schema or indexing required

Azure Cosmos DB's schema-less service automatically indexes all your data, regardless of the data model, to delivery blazing fast queries.

- Automatic index management
- Synchronous auto-indexing
- No schemas or secondary indices needed
- Works across every data model



Item	Color	Microwave safe	Liquid capacity	CPU	Memory	Storage
Geek mug	Graphite	Yes	16ox	???	???	???
Coffee Bean mug	Tan	No	12oz	???	???	???
Surface book	Gray	???	???	3.4 GHz Intel Skylake Core i7-6600U	16GB	1 TB SSD

Look at Andrew's "variety" slide

Automatic and synchronous indexing of all ingested content - hash, range, geo-spatial, and columnar

- No schemas or secondary indices ever needed

Resource governed, write optimized database engine with latch free and log structured techniques

Online and in-situ index transformations

While the database is fully schema-agnostic, schema-extraction is built in

- Customers can get Avro schemas from the database

5: Knowledge check

Which Cosmos DB API should you use to store and query JSON documents in Azure Cosmos DB?

- [x] Azure Cosmos DB for NoSQL
- [ ] Azure Cosmos DB for Apache Cassandra
- [ ] Azure Cosmos DB for Table

Which Azure Cosmos DB API should you use to work with data in which entities and their relationships to one another are represented in a graph using vertices and edges?

- [ ] Azure Cosmos DB for MongoDB
- [ ] Azure Cosmos DB for NoSQL
- [x] Azure Cosmos DB for Apache Gremlin

How can you enable globally distributed users to work with their own local replica of a Cosmos DB database?

- [ ] Create an Azure Cosmos DB account in each region where you have users
- [ ] Use the Table API to copy data to Azure Table Storage in each region where you have users
- [x] Enable multi-region writes and add the regions where you have users

Allow students a few minutes to think about the questions, and then use the animated slide to reveal the correct answers.