700 XP

**Describe concepts of relational data**

# Introduction

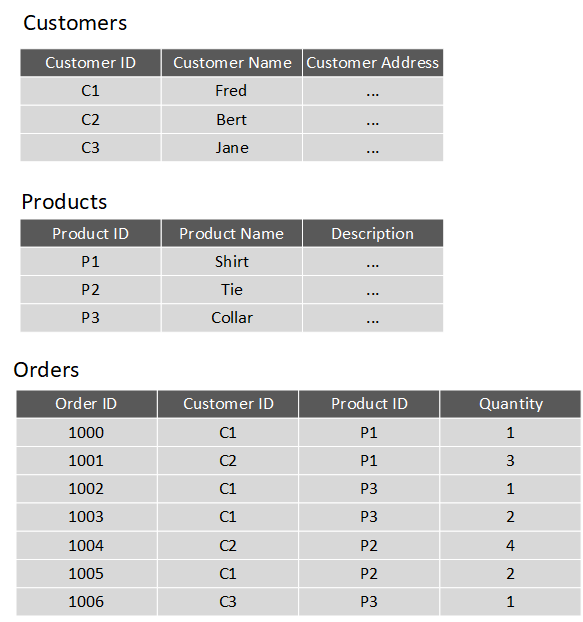
* In the early years of databases, every application stored data in its own unique structure.
* When developers wanted to build applications to use that data, they had to know a lot about the particular data structure to find the data they needed.
* These data structures were **inefficient, hard to maintain, and hard to optimize for delivering good application performance.**
* The relational database model was designed to solve the problem of multiple arbitrary data structures.
* The relational model provided a standard way of representing and querying data that could be used by any application.
* From the beginning, developers recognized that the chief strength of the relational database model was in its use of tables, which were an intuitive, efficient, and flexible way to store and access structured information.
* The simple yet powerful relational model is used by organizations of all types and sizes for a broad variety of information management needs.
* Relational databases are used **to track inventories, process ecommerce transactions, manage huge amounts of mission-critical customer information, and much more**.
* A relational database is useful for storing any information containing related data elements that must be organized in a rules-based, consistent way

# Explore the characteristics of relational data

1. **One of the main benefits of computer databases is that they make it easy to store information so it's quick and easy to find.**
2. For example, an ecommerce system might use a database to record information about the products an organization sells, and the details of customers and the orders they've placed.
3. **A relational database provides a model for storing the data, and a query capability that enables you to retrieve data quickly**.

## **Understand the characteristics of relational data**

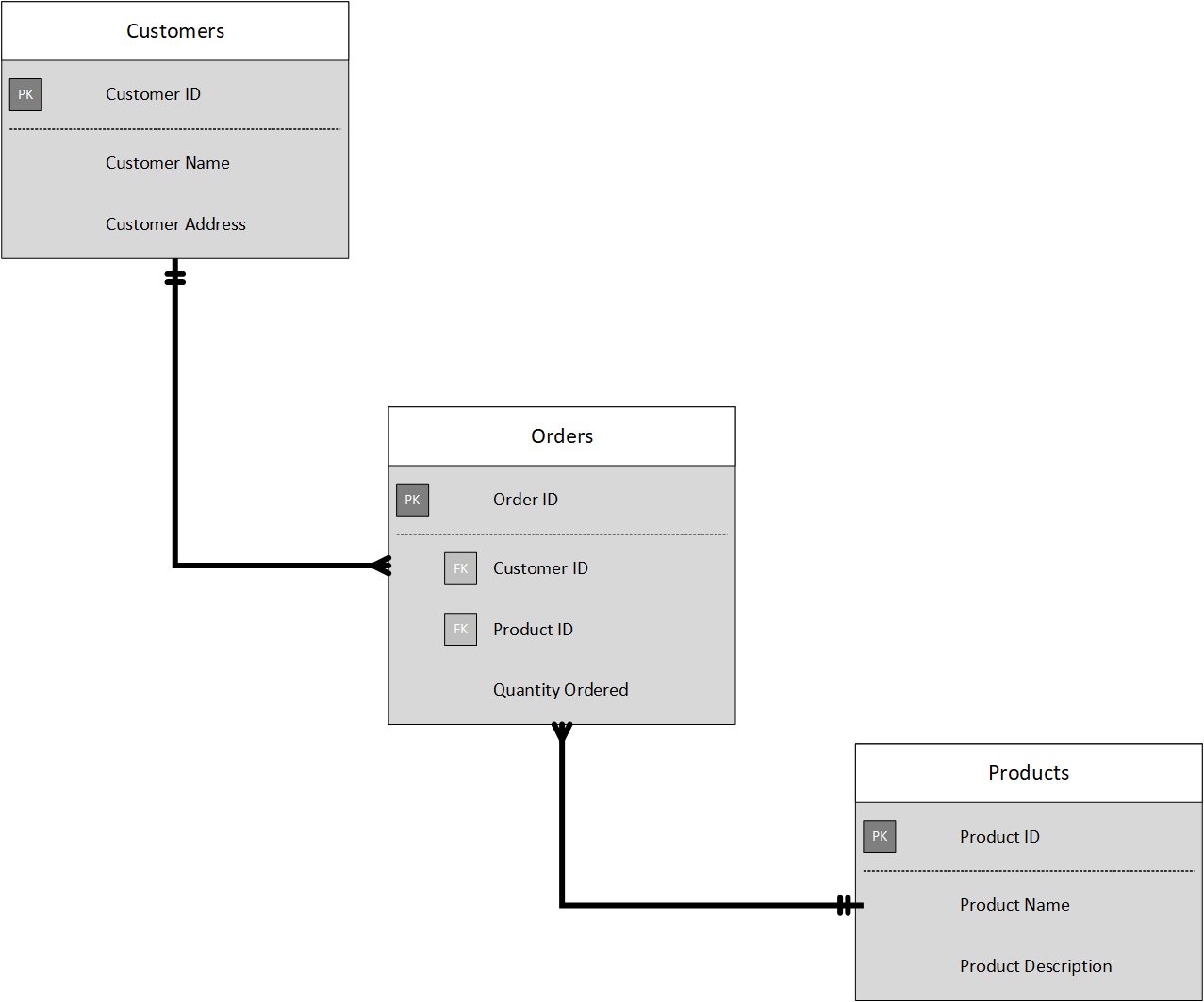
1. In a relational database, you model collections of **entities** from the real world as tables.
2. **An entity is described as a thing about which information needs to be known or held.**
3. In the ecommerce example, you might create tables for customers, products, and orders. A table contains rows, and each row represents a single instance of an entity. In the ecommerce scenario, each row in the customers table contains the data for a single customer, each row in the products table defines a single product, and each row in the orders table represents an order made by a customer.
4. The rows in a table have one or more columns that define the properties of the entity, such as the customer name, or product ID. All rows in the same table have the same columns.
5. **Some columns are used to maintain relationships between tables. This is where the relational model gets its name from.**
6. In the image below, the Orders table contains both a Customer ID and a Product ID. The Customer ID relates to the Customers table to identify the customer that placed the order, and the Product ID relates to the Products table to indicate what product was purchased.



You design a relational database by creating a data model. The model below shows the structure of the entities from the previous example. In this diagram, the columns marked PK are the Primary Key for the table. The primary key indicates the column (or combination of columns) that uniquely identify each row. **Every table should have a primary key.**

The diagram also shows the relationships between the tables. The lines connecting the tables indicate the type of relationship. In this case, the relationship from customers to orders is 1-to-many (one customer can place many orders, but each order is for a single customer). Similarly, the relationship between orders and products is many-to-1 (several orders might be for the same product).

The columns marked FK are **Foreign Key** columns. **They reference, or link to, the primary key of another table, and are used to maintain the relationships between tables**. A foreign key also helps to identify and **prevent anomalies, such as orders for customers that don't exist in the Customers table**. In the model below, the Customer ID and Product ID columns in the Orders table link to the customer that placed the order and the product that was ordered:



**The main characteristics of a relational database are:**

* All data is tabular. Entities are modeled as tables, each instance of an entity is a row in the table, and each property is defined as a column.
* All rows in the same table have the same set of columns.
* A table can contain any number of rows.
* A primary key uniquely identifies each row in a table. No two rows can share the same primary key.
* A foreign key references rows in another, related table. For each value in the foreign key column, there should be a row with the same value in the corresponding primary key column in the other table.

**Note**

**Creating a relational database model for a large organization is a not a trivial task. It can take several iterations to define tables to match the characteristics described above. Sometimes you have to split an entity into more than one table. This process is called**[**normalization**](https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description)**.**

1. Most relational databases support **Structured Query Language (SQL)**.
2. You use SQL to **create tables, insert, update, and delete rows in tables, and to query data**.
3. You use the CREATE TABLE command to create a table, the INSERT statement to store data in a table, the UPDATE statement to modify data in a table, and the DELETE statement to remove rows from a table. The SELECT statement retrieves data from a table. The example query below finds the details of every customer from the sample database shown above.

SQLCopy

|  |
| --- |
| SELECT CustomerID, CustomerName, CustomerAddress  FROM Customers |

Rather than retrieve every row, you can filter data by using a **WHERE** clause. The next query fetches the order ID and product ID for all orders placed by customer 1.

SQLCopy

|  |
| --- |
| SELECT OrderID, ProductID  FROM Orders  WHERE CustomerID = 'C1' |

You can combine the data from multiple tables in a query using a **join** operation. A join operation spans the relationships between tables, **enabling you to retrieve the data from more than one table at a time**. The following query retrieves the name of every customer, together with the product name and quantity for every order they've placed. Notice that each column is qualified with the table it belongs to:

SQLCopy

|  |
| --- |
| SELECT Customers.CustomerName, Orders.QuantityOrdered, Products.ProductName  FROM Customers JOIN Orders  ON Customers.CustomerID = Orders.CustomerID  JOIN Products  ON Orders.ProductID = Products.ProductID |

You can find full details about SQL on the Microsoft website, on the [Structured Query Language (SQL)](https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql) page.

## **Explore relational database use cases**

You can use a relational database any time you can easily model your data as a collection of tables with a fixed set of columns.

In theory, you could model almost any dataset in this way, but some scenarios lend themselves to the relational model better than others.

For example, if you have a collection of music, video, or other media files, attempting to force this data into the relational model could be difficult. You may be better off using unstructured storage, such as that available in Azure Blob storage.

Similarly, social networking sites use databases to store data about millions of users, each of whom can be linked to any number of other users in a highly complex web of relationships. This type of data lends itself more to a graph database structure rather than a collection of relational tables.

**Relational databases are commonly used in ecommerce systems,**

**but one of the major use cases for using relational databases is Online Transaction Processing (OLTP). OLTP applications are focused on transaction-oriented tasks that process a very large number of transactions per minute. Relational databases are well suited for OLTP applications because they naturally support insert, update, and delete operations. A relational database can often be tuned to make these operations fast. Also, the nature of SQL makes it easy for users to perform ad-hoc queries over data.**

**Examples of OLTP applications that use relational databases are:**

* **Banking solutions**
* **Online retail applications**
* **Flight reservation systems**
* **Many online purchasing applications.**

# Explore relational data structures

1. **A relational database comprises a set of tables**.
2. **A table can have zero (if the table is empty) or more rows**.
3. **Each table has a fixed set of columns.**
4. **You can define relationships between tables using primary and foreign keys, and you can access the data in tables using SQL.**

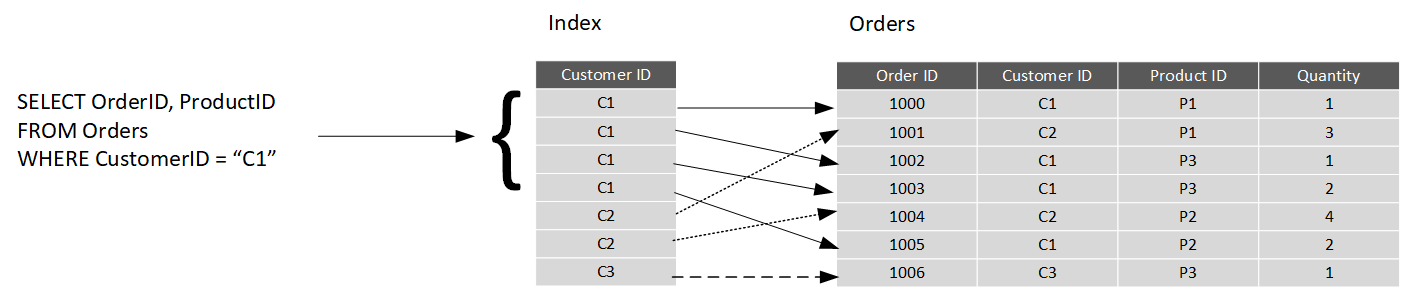
Apart from tables, a typical relational database contains other structures that help to optimize data organization, and improve the speed of access.

## **What is an index?**

**An index helps you search for data in a table.**

Think of an index over a table like an index at the back of a book. A book index contains a sorted set of references, with the pages on which each reference occurs. When you want to find a reference to an item in the book, you look it up through the index. You can use the page numbers in the index to go directly to the correct pages in the book. Without an index, you might have to read through the entire book to find the references you're looking for.

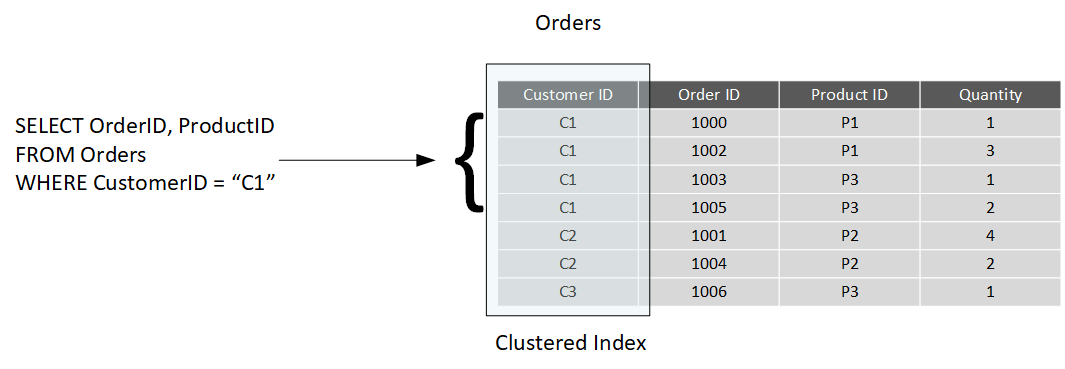
When you create an index in a database, you specify a column from the table, and the index contains a copy of this data in a sorted order, with pointers to the corresponding rows in the table. When the user runs a query that specifies this column in the WHERE clause, the database management system can use this index to fetch the data more quickly than if it had to scan through the entire table row by row. In the example below, the query retrieves all orders for customer C1. The Orders table has an index on the Customer ID column. The database management system can consult the index to quickly find all matching rows in the Orders table.



You can create many indexes on a table. So, if you also wanted to find all orders for a specific product, then creating another index on the Product ID column in the Orders table, would be useful. However, indexes aren't free. An index might consume additional storage space, and each time you insert, update, or delete data in a table, the indexes for that table must be maintained. This additional work can slow down insert, update, and delete operations, and incur additional processing charges. **Therefore, when deciding which indexes to create, you must strike a balance between having indexes that speed up your queries versus the cost of performing other operations**. In a table that is read only, or that contains data that is modified infrequently, more indexes will improve query performance.

If a table is queried infrequently, but subject to a large number of inserts, updates, and deletes (such as a table involved in OLTP), then creating indexes on that table can slow your system down.

Some relational database management systems also support **clustered indexes.** A clustered index physically reorganizes a table by the index key. This arrangement can improve the performance of queries still further, because the relational database management system doesn't have to follow references from the index to find the corresponding data in the underlying table. The image below shows the Orders table with a clustered index on the Customer ID column.



**In database management systems that support them, a table can only have a single clustered index.**

## **What is a view?**

**A view is a virtual table based on the result set of a query**.

In the simplest case, you can think of a view as a window on specified rows in an underlying table. For example, you could create a view on the Orders table that lists the orders for a specific product (in this case, product P1) like this:

SQLCopy

|  |  |
| --- | --- |
| |  | | --- | | CREATE VIEW P1Orders AS  SELECT CustomerID, OrderID, Quantity  FROM Orders  WHERE ProductID = "P1" | |

You can query the view and filter the data in much the same way as a table. The following query finds the orders for customer C1 using the view. This query will only return orders for product P1 made by the customer:

SQLCopy

|  |
| --- |
| SELECT CustomerID, OrderID, Quantity  FROM P1Orders  WHERE CustomerID = "C1" |

A view can also join tables together. If you regularly needed to find the details of customers and the products that they've ordered, you could create a view based on the join query shown in the previous unit:

SQLCopy

|  |
| --- |
| CREATE VIEW CustomersProducts AS  SELECT Customers.CustomerName, Orders.QuantityOrdered, Products.ProductName  FROM Customers JOIN Orders  ON Customers.CustomerID = Orders.CustomerID  JOIN Products  ON Orders.ProductID = Products.ProductID |

The following query finds the customer name and product names of all orders placed by customer C2, using this view:

SQLCopy

|  |
| --- |
| SELECT CustomerName, ProductName  FROM CustomersProducts  WHERE CustomerID = "C2" |

# Choose the right platform for a relational workload

Cloud computing has grown in popularity, promising flexibility for enterprises, opportunities for saving time and money, and improving agility and scalability. On the other hand, on-premises software, installed on a company’s own servers and behind its firewall, still has its appeal. On-premises applications are reliable, secure, and allow enterprises to maintain close control.

Relational database management systems are one example of where the cloud has enabled organizations to take advantage of improved scalability. However, this scalability has to be balanced against the need for close control over the data. Data is arguably one of the most valuable assets that an organization has, and some companies aren't willing or able to hand over responsibility for protecting this data to a third party.

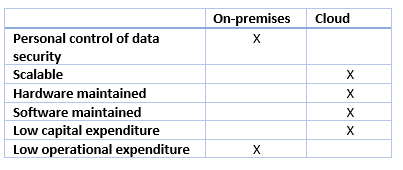
In this unit, you'll look at some of the advantages and disadvantages of running a database management system in the cloud.

## **Compare on-premises hosting to the cloud**

Whether a company places its relational workload in the cloud or whether it decides to keep it on premises, data security will always be paramount. But for those businesses in highly regulated industries, the decision might already be made for them as to whether to host their applications on-premises. Knowing that your data is located within your in-house servers and IT infrastructure might also provide more peace of mind.

Hosting a relational database on-premises requires that an enterprise not only purchases the database software, but also maintains the necessary hardware on which to run the database. The organization is responsible for maintaining the hardware and software, applying patches, backing up databases, restoring them when necessary, and generally performing all the day-to-day management required to keep the platform operational. Scalability is also a concern. If you need to scale your system, you will need to upgrade or add more servers. You then need to expand your database onto these servers. This can be a formidable task that requires you to take a database offline while the operation is performed. In the cloud, many of these operations can be handled for you by the data center staff, in many cases with no (or minimal) downtime. You're free to focus on the data itself and leave the management concerns to others (this is what you pay your Azure fees for, after all).

A cloud-based approach uses virtual technology to host a company’s applications offsite. There are no capital expenses, data can be backed up regularly, and companies only have to pay for the resources they use. For those organizations that plan aggressive expansion on a global basis, the cloud has even greater appeal because it allows you to connect with customers, partners, and other businesses anywhere with minimal effort. Additionally, cloud computing gives you nearly instant provisioning because everything is already configured. Thus, any new software that is integrated into your environment is ready to use immediately once a company has subscribed. With instant provisioning, any time spent on installation and configuration is eliminated and users can access the application right away.



## **Understand IaaS and PaaS**

You generally have two options when moving your operations and databases to the cloud. You can select an IaaS approach, or PaaS.

**IaaS** is an acronym for Infrastructure-as-a-Service. Azure enables you to create a virtual infrastructure in the cloud that mirrors the way an on-premises data center might work. You can create a set of virtual machines, connect them together using a virtual network, and add a range of virtual devices. In many ways, this approach is similar to the way in which you run your systems inside an organization, except that you don't have to concern yourself with buying or maintaining the hardware. However, you're still responsible for many of the day-to-day operations, such as installing and configuring the software, patching, taking backups, and restoring data when needed. You can think of IaaS as a half-way-house to fully managed operations in the cloud; you don't have to worry about the hardware, but running and managing the software is still very much your responsibility.

You can run any software for which you have the appropriate licenses using this approach. You're not restricted to any specific database management system.

The IaaS approach is best for migrations and applications requiring operating system-level access. SQL virtual machines are **lift-and-shift. That is, you can copy your on-premises solution directly to a virtual machine in the cloud.** The system should work more or less exactly as before in its new location, except for some small configuration changes (changes in network addresses, for example) to take account of the change in environment.

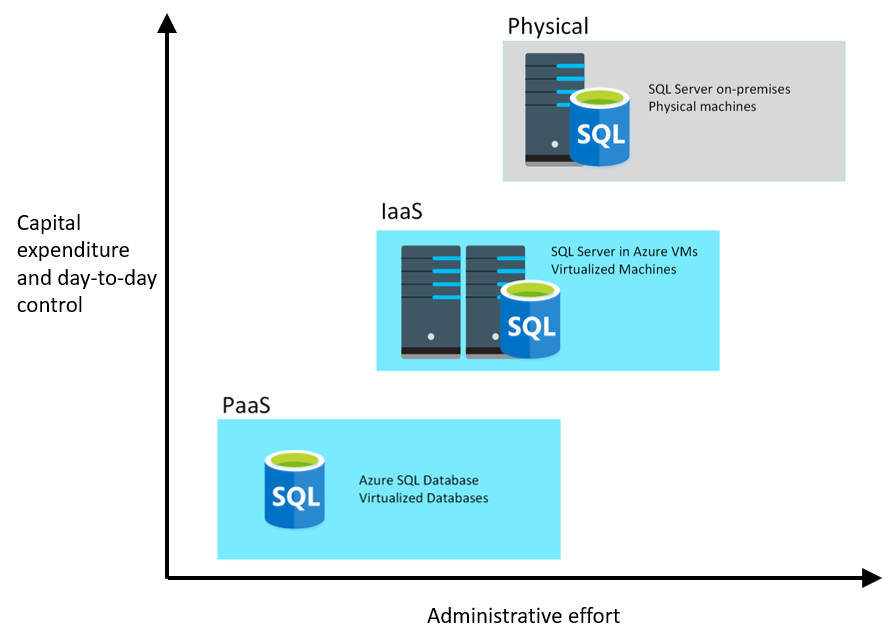
**PaaS** stands for Platform-as-a-service. Rather than creating a virtual infrastructure, and installing and managing the database software yourself, a PaaS solution does this for you. You specify the resources that you require (based on how large you think your databases will be, the number of users, and the performance you require), and Azure automatically creates the necessary virtual machines, networks, and other devices for you. You can usually scale up or down (increase or decrease the size and number of resources) quickly, as the volume of data and the amount of work being done varies; Azure handles this scaling for you, and you don't have to manually add or remove virtual machines, or perform any other form of configuration.

Azure offers several PaaS solutions for relational databases, include

* Azure SQL Database,
* Azure Database for PostgreSQL,
* Azure Database for MySQL,
* Azure Database for MariaDB.

These services run managed versions of the database management systems on your behalf. You just connect to them, create your databases, and upload your data. However, you may find that there are some functional restrictions in place, and not every feature of your selected database management system may be available. These restrictions are often due to security issues. For example, they might expose the underlying operating system and hardware to your applications. In these cases, you may need to rework your applications to remove any dependencies on these features.

The image below illustrates the benefits and tradeoffs when running a database management system (in this case, SQL Server) on-premises, using virtual machines in Azure (IaaS), or using Azure SQL Database (PaaS). The same generalized considerations are true for other database management systems.



Relational databases are widely used for building real world applications. Understanding the characteristics of relational data is important. A relational database is based on tables. You can run many database management systems on-premises and in the cloud.

* [Description of the database normalization basics](https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description)
* [Structured Query Language (SQL)](https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql)
* [Technical overview of SQL Database](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview)
* [SQL Server Technical Documentation](https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15)
* [SQL Database PaaS vs IaaS Offerings](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-paas-vs-sql-server-iaas)