**Relational data**

In a relational database, you model collections of entities from the real world as *tables*. An entity can be anything for which you want to record information; typically important objects and events. For example, in a retail system example, you might create tables for customers, products, orders, and line items within an order. A table contains rows, and each row represents a single instance of an entity.

Relational databases are a common way for transactional applications to store and manage data. They consist of a schema of *tables*, which are linked through common key values. You use SQL to query and manipulate the data in the tables, and can enrich the database by creating objects like views, stored procedures, and indexes.

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

While there are many complex rules that define the process of refactoring data into various levels (or *forms*) of normalization, a simple definition for practical purposes is:

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

Instances of each entity are uniquely identified by an ID or other key value, known as a primary key; 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. You 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.

In some cases, a key (primary or foreign) can be defined as a composite key based on a unique combination of multiple columns. For example, the LineItem table in the example above uses a unique combination of OrderNo and ItemNo to identify a line item from an individual order.

**Explore SQL:**

SQL stands for *Structured Query Language*, and is used to communicate with a relational database. It's the standard language for relational database management systems. SQL statements are used to perform tasks such as update data in a database, or retrieve data from a database. Some common relational database management systems that use SQL include Microsoft SQL Server, MySQL, PostgreSQL, MariaDB, and Oracle.

You can use SQL statements such as **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE**, and **DROP** to accomplish almost everything that you need to do with a database.

**SQL statement types**

SQL statements are grouped into three main logical groups:

* Data Definition Language (DDL)
* Data Control Language (DCL)
* Data Manipulation Language (DML)

**DDL statements**

You use DDL statements to create, modify, and remove tables and other objects in a database (table, stored procedures, views, and so on).

The most common DDL statements are:

| **Statement** | **Description** |
| --- | --- |
| CREATE | Create a new object in the database, such as a table or a view. |
| ALTER | Modify the structure of an object. For instance, altering a table to add a new column. |
| DROP | Remove an object from the database. |
| RENAME | Rename an existing object. |

The datatypes available for columns in a table will vary between database management systems. However, most database management systems support numeric types such as INT (an integer, or whole number), DECIMAL (a decimal number), and string types such as VARCHAR (VARCHAR stands for variable length character data).

**DCL statements**

Database administrators generally use DCL statements to manage access to objects in a database by granting, denying, or revoking permissions to specific users or groups.

The three main DCL statements are:

| **Statement** | **Description** |
| --- | --- |
| GRANT | Grant permission to perform specific actions |
| DENY | Deny permission to perform specific actions |
| REVOKE | Remove a previously granted permission |

**DML statements**

You use DML statements to manipulate the rows in tables. These statements enable you to retrieve (query) data, insert new rows, or modify existing rows. You can also delete rows if you don't need them anymore.

The four main DML statements are:

| **Statement** | **Description** |
| --- | --- |
| SELECT | Read rows from a table |
| INSERT | Insert new rows into a table |
| UPDATE | Modify data in existing rows |
| DELETE | Delete existing rows |

The basic form of an **INSERT** statement will insert one row at a time. By default, the **SELECT**, **UPDATE**, and **DELETE** statements are applied to every row in a table. You usually apply a **WHERE** clause with these statements to specify criteria; only rows that match these criteria will be selected, updated, or deleted.

# Describe database objects

A **view** is a virtual table based on the results of a **SELECT** query. You can think of a view as a window on specified rows in one or more underlying tables.

A **stored procedure** defines SQL statements that can be run on command. Stored procedures are used to encapsulate programmatic logic in a database for actions that applications need to perform when working with data. You can define a stored procedure with parameters to create a flexible solution for common actions that might need to be applied to data based on a specific key or criteria.

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.

For a table containing few rows, using the index is probably not any more efficient than simply reading the entire table and finding the rows requested by the query (in which case the query optimizer will ignore the index). However, when a table has many rows, indexes can dramatically improve the performance of queries.

You can create many indexes on a table. So, if you also wanted to find products based on price, creating another index on the **Price** column in the **Product** table might be useful. However, indexes aren't free. An index consumes 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. You must strike a balance between having indexes that speed up your queries versus the cost of performing other operations.