قواعد البيانات
المرحلة الثانية

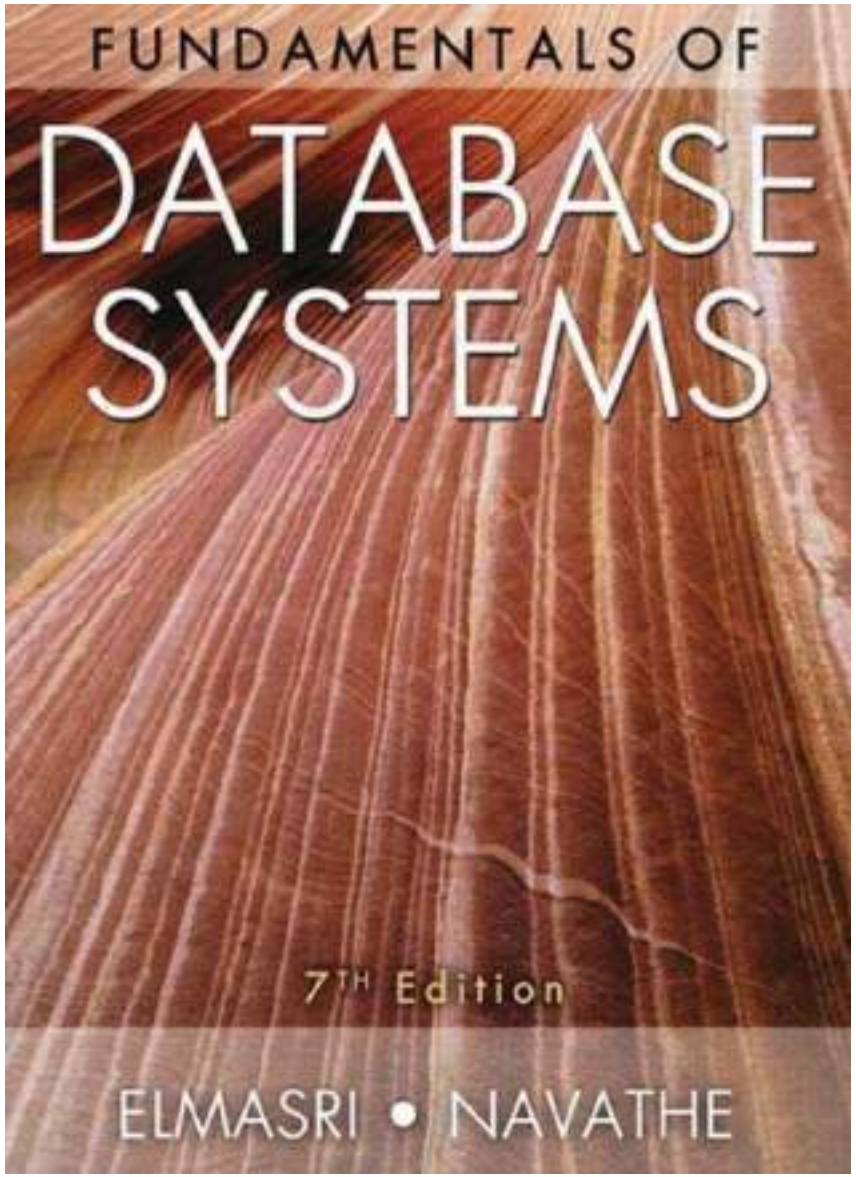
د. محمد خلدون الطالب (نظري + عملي)
د. شيماء أحمد رزوقي (عملي)

Fundamentals of Database

Dr. Mohammed Khaldoon Altalib

Chapter 1 Outline

- Introduction
- An Example
- Characteristics of the Database Approach
- Actors on the Scene
- Workers behind the Scene
- Advantages of Using the DBMS Approach



Databases and database systems are an essential component of life in modern society: most of us encounter several activities every day that involve some interaction with a database. For example, if we go to the bank to deposit or withdraw funds, if we make a hotel or airline reservation, if we access a computerized library catalog to search for a bibliographic item, or if we purchase something online—such as a book, toy, or computer—chances are that our activities will involve someone or some computer program accessing a database. Even purchasing items at a supermarket often automatically updates the database that holds the inventory of grocery items.

A **database** is a collection of related data.[1] By **data**, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. Nowadays, this data is typically stored in mobile phones, which have their own simple database software. This data can also be recorded in an indexed address book or stored on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database.

Database	A shared collection of logically related data and its description, designed to meet the information needs of an organization.

constitute a database. However, the common use of the term *database* is usually more restricted. A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the **miniworld** or the **universe of discourse (UoD)**. Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

TYPE OF DATABASES:

- **Traditional Database Applications**: is stored and accessed is either textual or numeric.

- **Big Data Storage systems, or NoSQL systems**: In the past few years, advances in technology have led to exciting new applications of database systems. The proliferation of social media Web sites, such as Facebook, Twitter, and Flickr, among many others, has required the creation of huge databases that store nontraditional data, such as posts, tweets, images, and video clips.

- **multimedia databases:** it is possible to store images, audio clips, and video streams digitally.

- **Geographic information systems (GISs):** can store and analyze maps, weather data, and satellite images.

- **Data warehouses and online analytical processing (OLAP) systems:** systems are used in many companies to extract and analyze useful business information from very large databases to support decision-making.

- **Real-time and active database technology:** is used to control industrial and manufacturing processes.

- **database search techniques** are being applied to the World Wide Web to improve the search for information that is needed by users browsing the Internet

DBMS | A software system that enables users to define, create, maintain, and control access to the database.

A **database management system (DBMS)** is a computerized system that enables users to create and maintain a database. The DBMS is a *general-purpose software system* that facilitates the processes of *defining*, *constructing*, *manipulating*, and *sharing* databases among various users and applications. **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**. **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS. **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data. **Sharing** a database allows multiple users and programs to access the database simultaneously.

- A meta data is the data about the data. It is the self-describing nature of databases. It holds the information about each data element in the database. Such as names, types, range of values, access authorization.

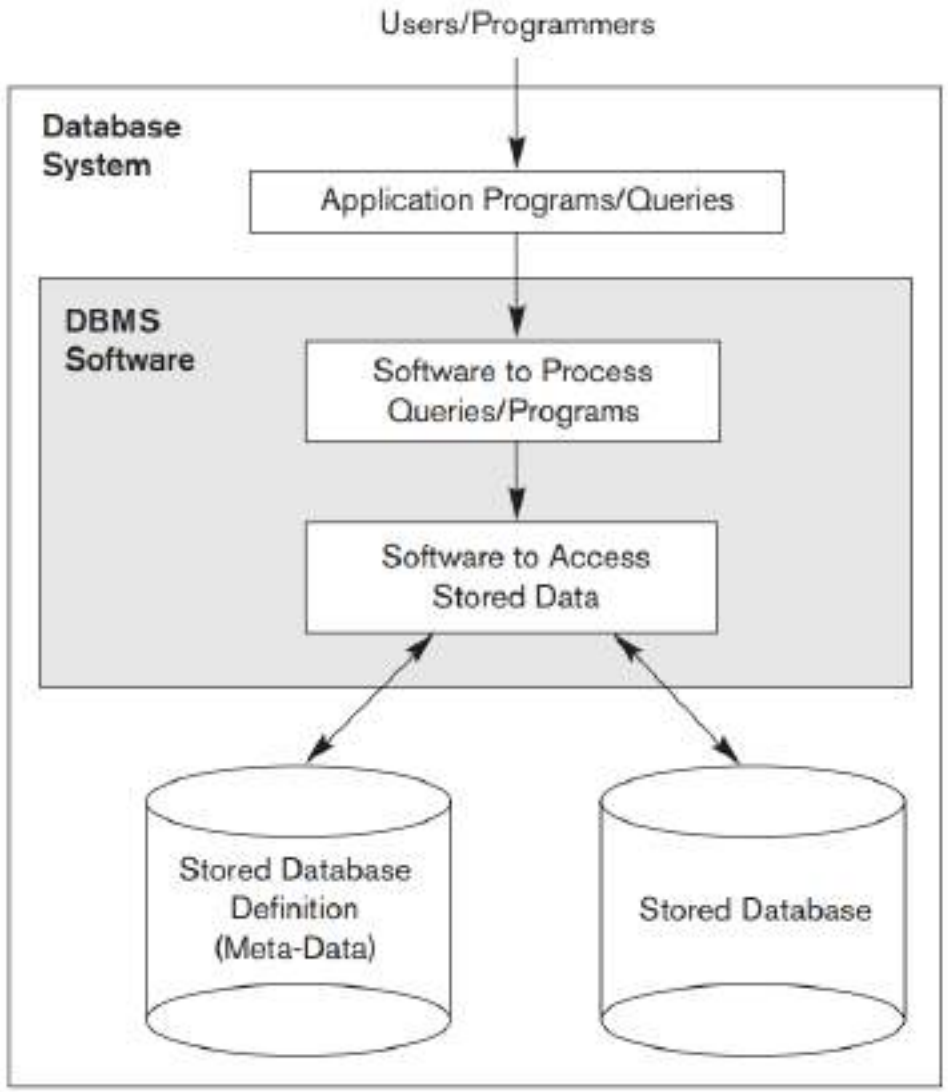


Figure 1.1
A simplified database system environment.

Data Abstraction

Data abstraction in a database refers to the process of hiding the complex details of how data is stored and maintained.

The characteristic that allows program-data independence and program-operation independence is called data abstraction. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented.

Transaction

a transaction refers to a unit of work that is performed against a database. It is a sequence of one or more SQL operations (like INSERT, UPDATE, DELETE, SELECT) that are executed as a single, logical unit. The goal of a transaction is to ensure data integrity and consistency.

1.4 Actors on the Scene

For a small personal database, such as the list of addresses discussed in Section 1.1, one person typically defines, constructs, and manipulates the database, and there is no sharing. However, in large organizations, many people are involved in the design, use, and maintenance of a large database with hundreds or thousands of users. In this section we identify the people whose jobs involve the day-to-day use of a large database; we call them the *actors on the scene*. In Section 1.5 we consider people who may be called *workers behind the scene*—those who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job.

1.4.1 Database Administrators

- A **Database Administrator (DBA)** is a specialized IT professional responsible for the management, maintenance, and security of databases within an organization. Their primary role is to ensure that databases are available, optimized for performance, secure, and functioning reliably to meet the needs of the organization or its users. DBAs typically work with database management systems (DBMS) like Oracle, MySQL, Microsoft SQL Server, PostgreSQL, or others to manage the data.

1.4.2 Database Designers

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements. In many cases, the designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop views of the database that meet the data and processing requirements of these groups

1.4.3 End Users

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.

1.4.4 System Analysts and Application Programmers (Software Engineers)

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements. **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers—commonly referred to as **software developers** or **software engineers**—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

1.5 Workers behind the Scene

These persons are typically not interested in the database content itself. Like

- DBMS system implementers who implement the DBMS modules and interfaces as a software package.

- Operators and maintenance personnel.

1- Controlling Redundancy

This redundancy in storing the same data multiple times leads to several problems. First, there is the need to perform a single logical update—such as entering data on a new student—multiple times: once for each file where student data is recorded. This leads to duplication of effort. Second, storage space is wasted when the same data is stored repeatedly, and this problem may be serious for large databases. Third, files that represent the same data may become inconsistent. This may happen because an update is applied to some of the files but not to others.

- **Summary of advantages of using the DBMS**

2- Restricting unauthorized access

- When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database.
- A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions.

3- Providing Persistent Storage for Program Objects

4- Providing Storage Structures and Search Techniques for Efficient Query Processing

- Indexes
- Buffering and caching
- query processing and optimization

- **5- Providing Backup and Recovery**
- The backup and recovery subsystem of the DBMS is responsible for recovery.

6- Providing Multiple User Interfaces

- By using graphical user interfaces (GUIs).

7- Representing Complex Relationships among Data

8- Enforcing Integrity Constraints

specifying a data type for each data item.

specifies uniqueness on data item values**,** known as a key or uniqueness constraint

Brief History Of Database Applications

1. Early File-Based Systems (1950s-1960s)

•**Description**: In the early days of computing, data was stored in **flat file systems**. These were simple collections of files, usually text or binary, with no relationships between them. Data access and management were handled by the application programs directly, which often resulted in data redundancy and inconsistency.

2. Hierarchical and Network Databases (1960s-1970s)

•**Hierarchical Databases**:

- The **hierarchical database model** was introduced in the 1960s with systems like **IBM's IMS (Information Management System)**. In this model, data is organized in a tree-like structure where each child record has a single parent.

•**Network Databases**:

•The **network database model**, represented by systems like **CODASYL DBTG** (Conference on Data Systems Languages Data Base Task Group), emerged in the late 1960s. It allowed more complex many-to-many relationships using a graph-like structure.

3. Relational Databases (1970s-Present)

•**Introduction of the Relational Model (1970s)**:

- The **relational model** was proposed by **Edgar F. Codd** in 1970, introducing the idea of organizing data into **tables (relations)**. This model allowed for greater flexibility, data independence, and the use of structured queries.
- The introduction of **Structured Query Language (SQL)** in the late 1970s became the standard for querying and managing relational databases.

4. Object-Oriented Databases (1980s-1990s)

•**Object-Oriented Databases (OODBMS)** emerged in the 1980s to address the limitations of the relational model when dealing with complex data, such as multimedia or engineering data.
•These databases were designed to store data as **objects** (similar to objects in programming languages), incorporating both data and the associated methods for manipulating that data.

5. Data Warehousing and OLAP (1990s)

- **Data Warehouses** became prominent in the 1990s as a way to store large amounts of historical data from multiple sources for analysis and reporting. Data warehouses are optimized for reading and analyzing large datasets.
- **Online Analytical Processing (OLAP)** tools allow users to perform complex queries and data analysis, often used for **business intelligence (BI)**.

6. NoSQL Databases (2000s-Present)

- The rise of the internet and **big data** in the 2000s led to the need for databases that could handle large-scale, unstructured, and semi-structured data. Traditional relational databases struggled to meet the demands of **web-scale** applications and **real-time data**.

- **NoSQL (Not Only SQL)** databases emerged as a solution. They are non-relational and often provide horizontal scalability, which makes them suitable for large, distributed systems.

chapter 2

Database System Concepts and Architecture

2.1 Data Models, Schemas, and Instances

A data model—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction.

structure of a database we mean the data types, relationships, and constraints that apply to the data.

Most data models also include a set of basic operations for specifying retrievals and updates on the database.

3. Physical Data Models

- **Purpose:** Define how data will be stored and accessed on physical storage media (e.g., disks, memory).
- **Usage:** Used by database administrators (DBAs) to optimize database performance and storage.

Key Characteristics:

- Deals with **data files, indexes, partitions, and memory allocations**.
- Includes physical details like **file paths, block sizes, compression,** and **replication** strategies.
- Maps logical structures (e.g., tables) to physical storage mechanisms.

Summary of Data Model Categories

Category	Description	Examples
Conceptual Models	Abstract models to represent entities and relationships.	Entity-Relationship Model, Object-Oriented Model
Logical Models	Defines the structure of data and relationships.	Relational, Hierarchical, Network, Document Models
Physical Models	Specifies how data is stored physically in the system.	Storage paths, Indexes, Data Compression, Partitioning

2.1.1 Categories of Data Models

Many data models have been proposed, which we can categorize according to the types of concepts they use to describe the database structure. **High-level** or **conceptual data models** provide concepts that are close to the way many users perceive data, whereas **low-level** or **physical data models** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. Concepts provided by physical data models are generally meant for computer specialists, not for end users. Between these two extremes is a class of **representational** (or **implementation**) **data models**,[4] which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

Conceptual data models use concepts such as entities, attributes, and relationships. An **entity** represents a real-world object or concept, such as an employee or a project from the miniworld that is described in the database. An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary. A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project. Chapter 3 presents the **entity–relationship model**—a popular high-level conceptual data model. Chapter 4 describes additional abstractions used for advanced modeling, such as generalization, specialization, and categories (union types).

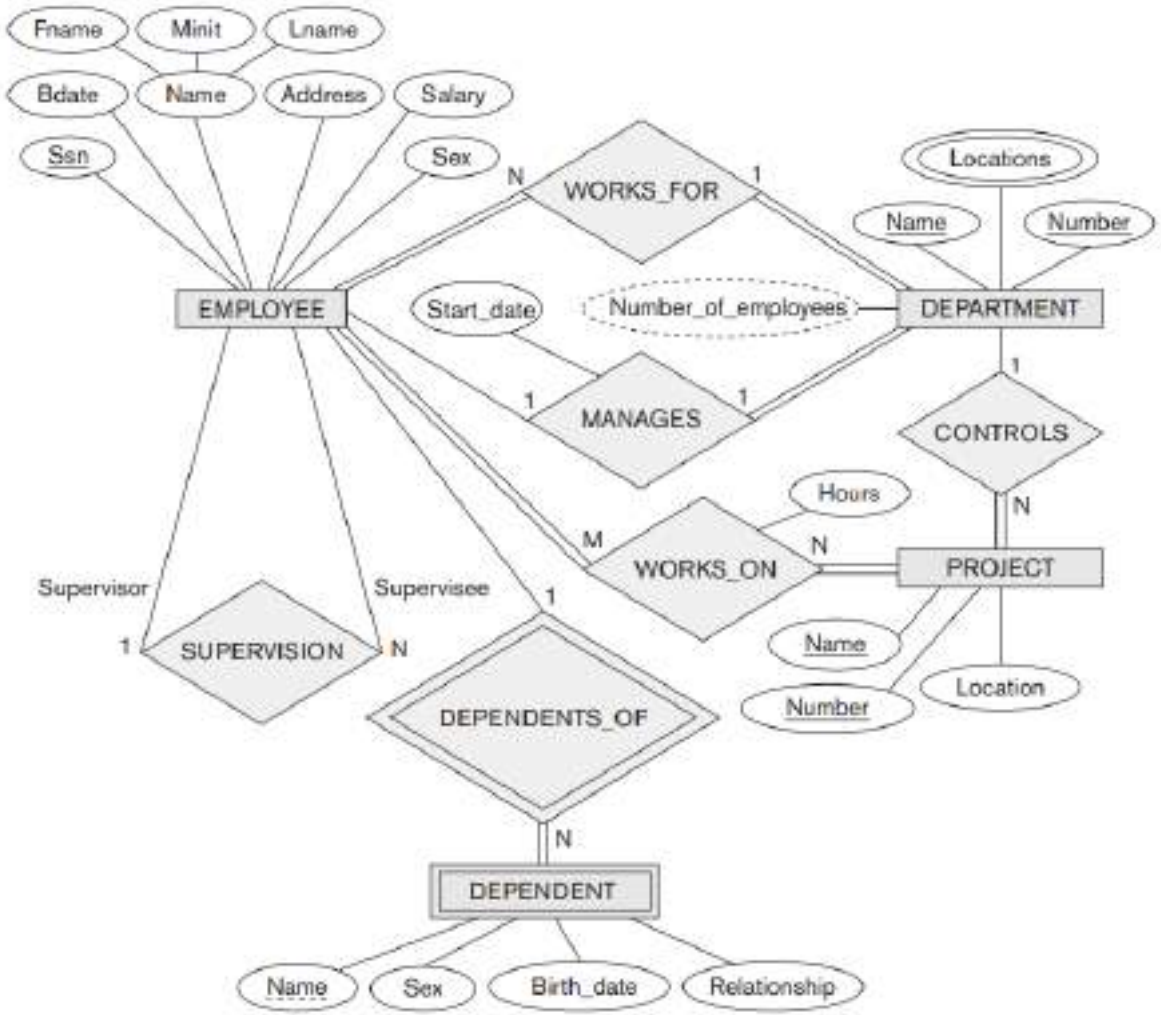
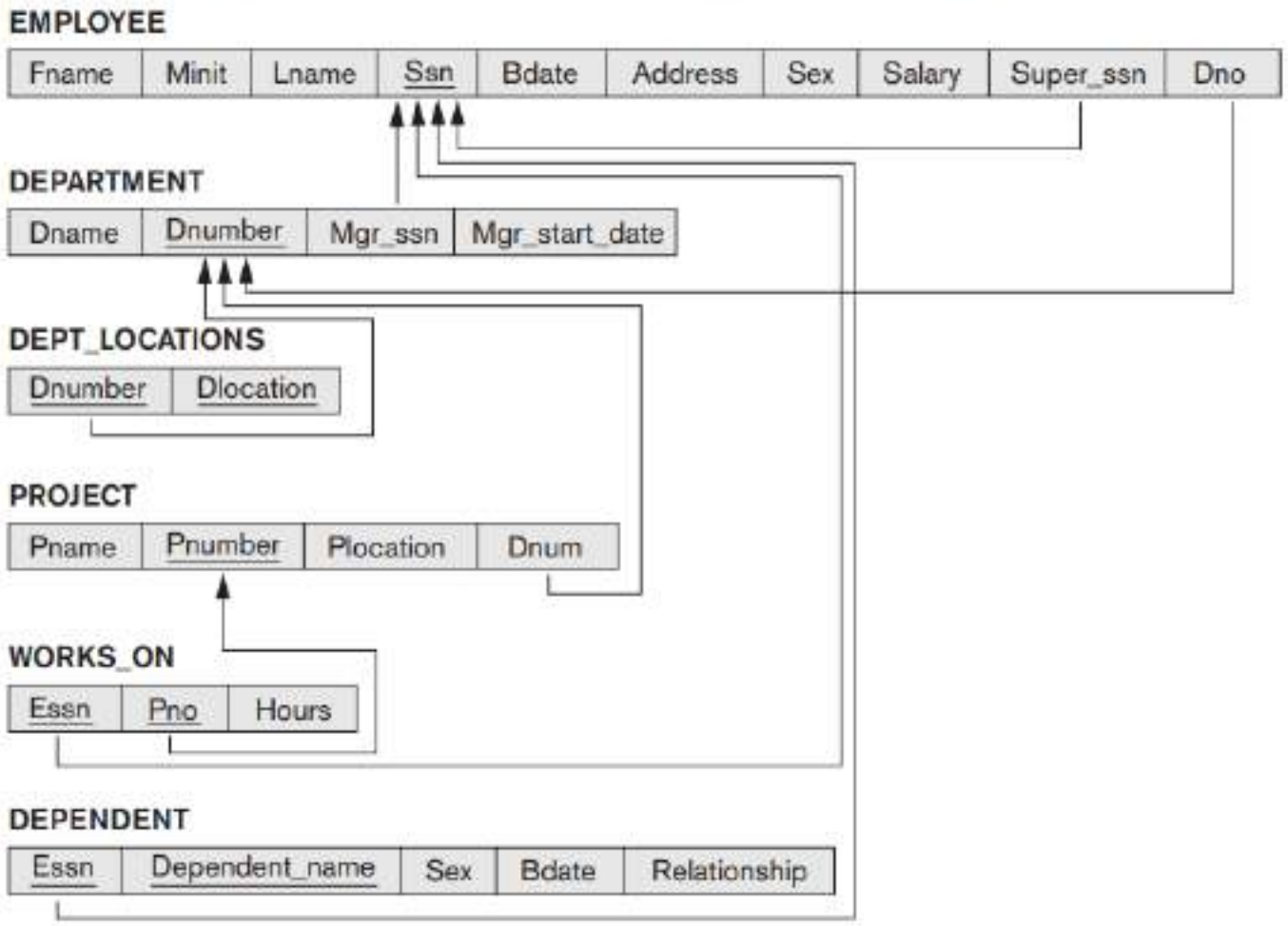

Fname
Minit
Lname
Name
Bdate
Address
Salary
Ssn
Sex
EMPLOYEE
WORKS_FOR
N
1
Locations
Name
Number
DEPARTMENT
Start_date
Number_of_employees
MANAGES
1
1
1
CONTROLS
1
N
Hours
M
WORKS_ON
N
PROJECT
Supervisor
Supervisee
1
1
SUPERVISION
N
Name
Location
Number
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
Birth_date
Relationship

Figure 3.2

Representational or implementation data models are the models used most frequently in traditional commercial DBMSs. These include the widely used **relational data model**, as well as the so-called legacy data models—the **network** and **hierarchical models**—that have been widely used in the past. Part 3 of the text is devoted to the relational data model, and its constraints, operations, and languages.[5] The SQL standard for relational databases is described in Chapters 6 and 7. Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.

Physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths. An

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

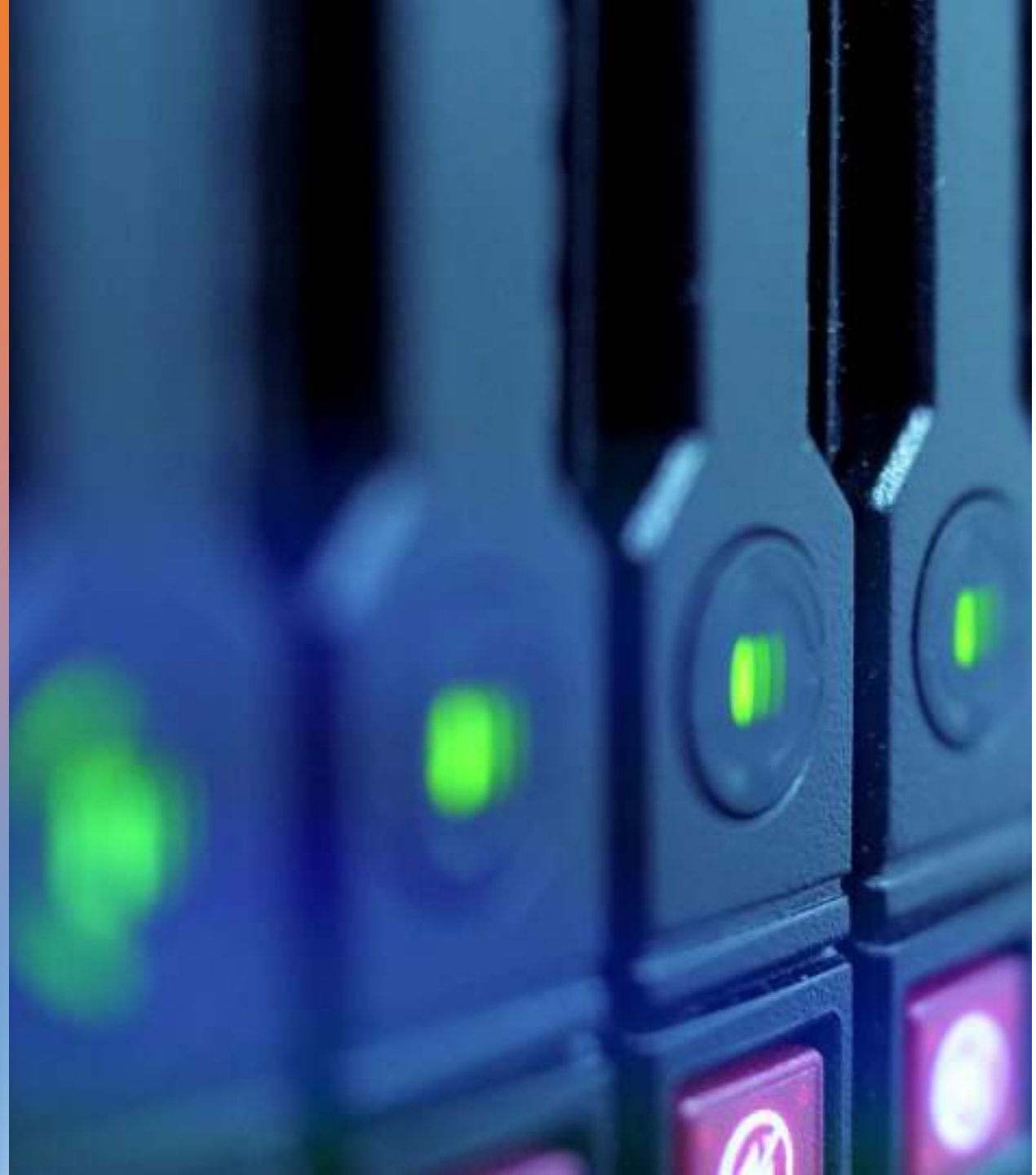
SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

- Database schema: The description of a database that is specified during database design and is not expected to change frequently. Figure 2.1 shows a schema diagram for the database. The diagram displays the structure of each record type.

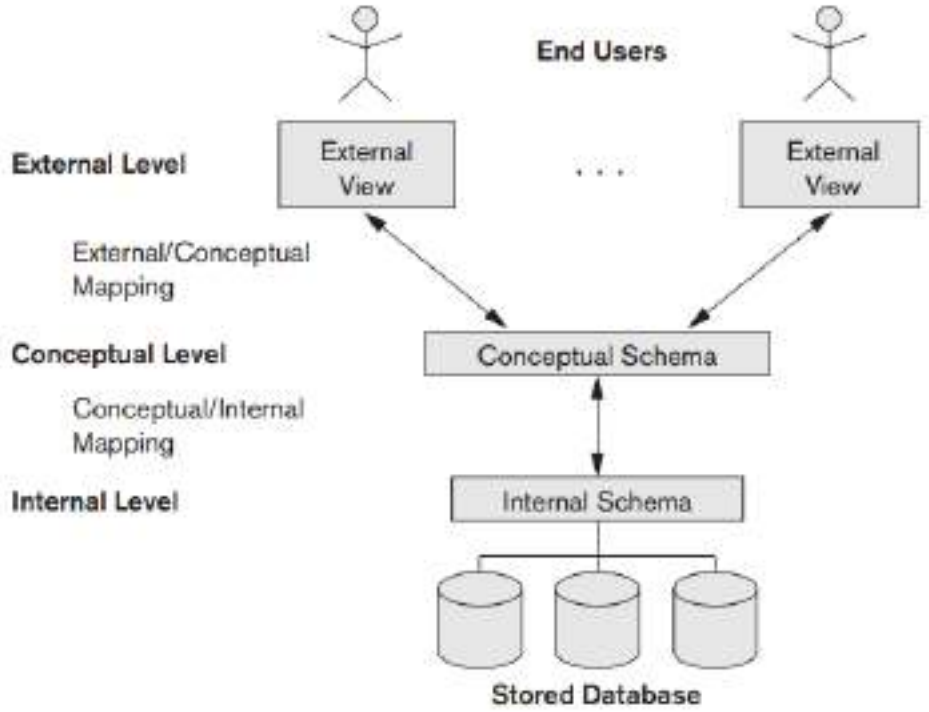


- **Database state or snapshot:** The data in the database at a particular moment in time. It is also called the current set of occurrences or instances in the database. Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state. empty state with no data When we define a new database.

2.2.1 The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.



2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.
3. The **external** or **view level** includes a number of **external schemas** or **user views**. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level conceptual data model.

2.2.2 Data Independence

The three-schema architecture can be used to further explain the concept of **data independence**, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external sche-

2.3.2 DBMS Interfaces

User-friendly interfaces provided by a DBMS may include the following:

Menu-based Interfaces for Web Clients or Browsing

Apps for Mobile Devices.

Forms-based Interfaces

Graphical User Interfaces.

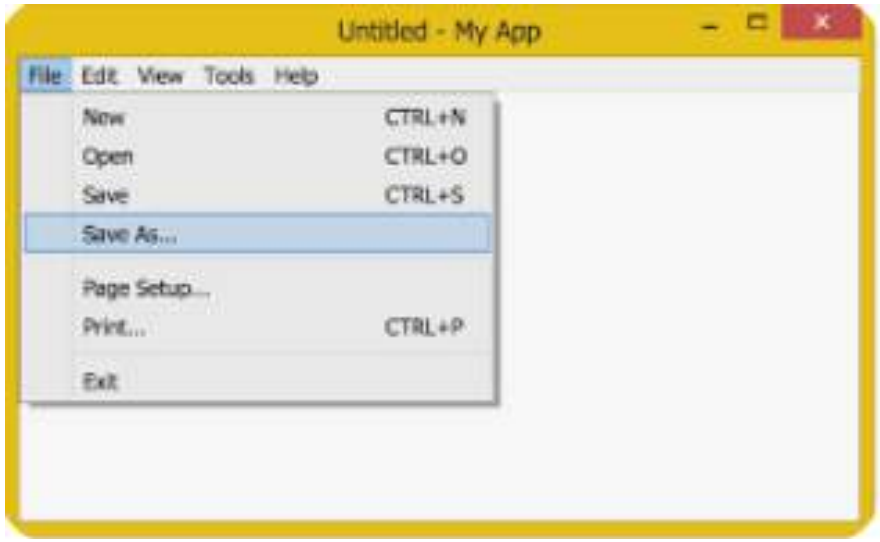
Natural Language Interfaces

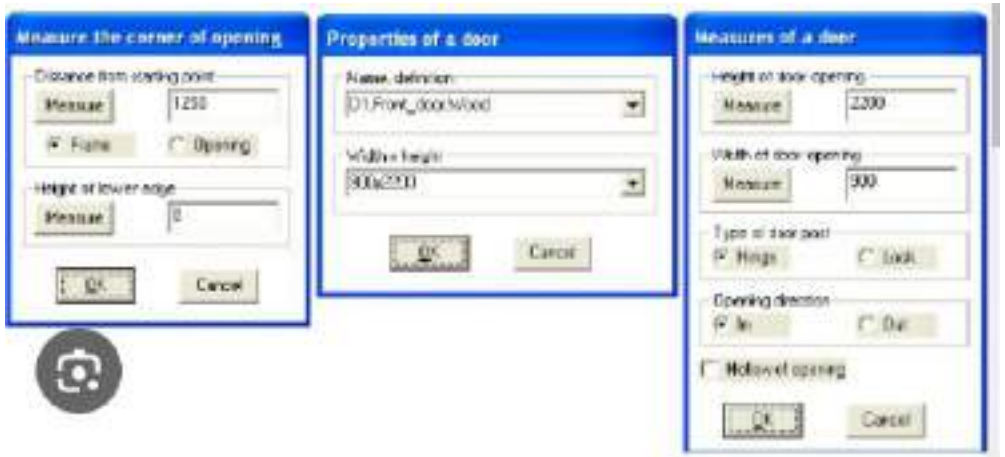
Keyword-based Database Search

Speech Input and Output.

Interfaces for Parametric Users

Interfaces for the DBA.


Untitled - My App
File Edit View Tools Help
New CTRL+N
Open CTRL+O
Save CTRL+S
Save As...
Page Setup...
Print... CTRL+P
Exit


Measure the corner of opening
Properties of a door
Measures of a door
Measure
Cancel

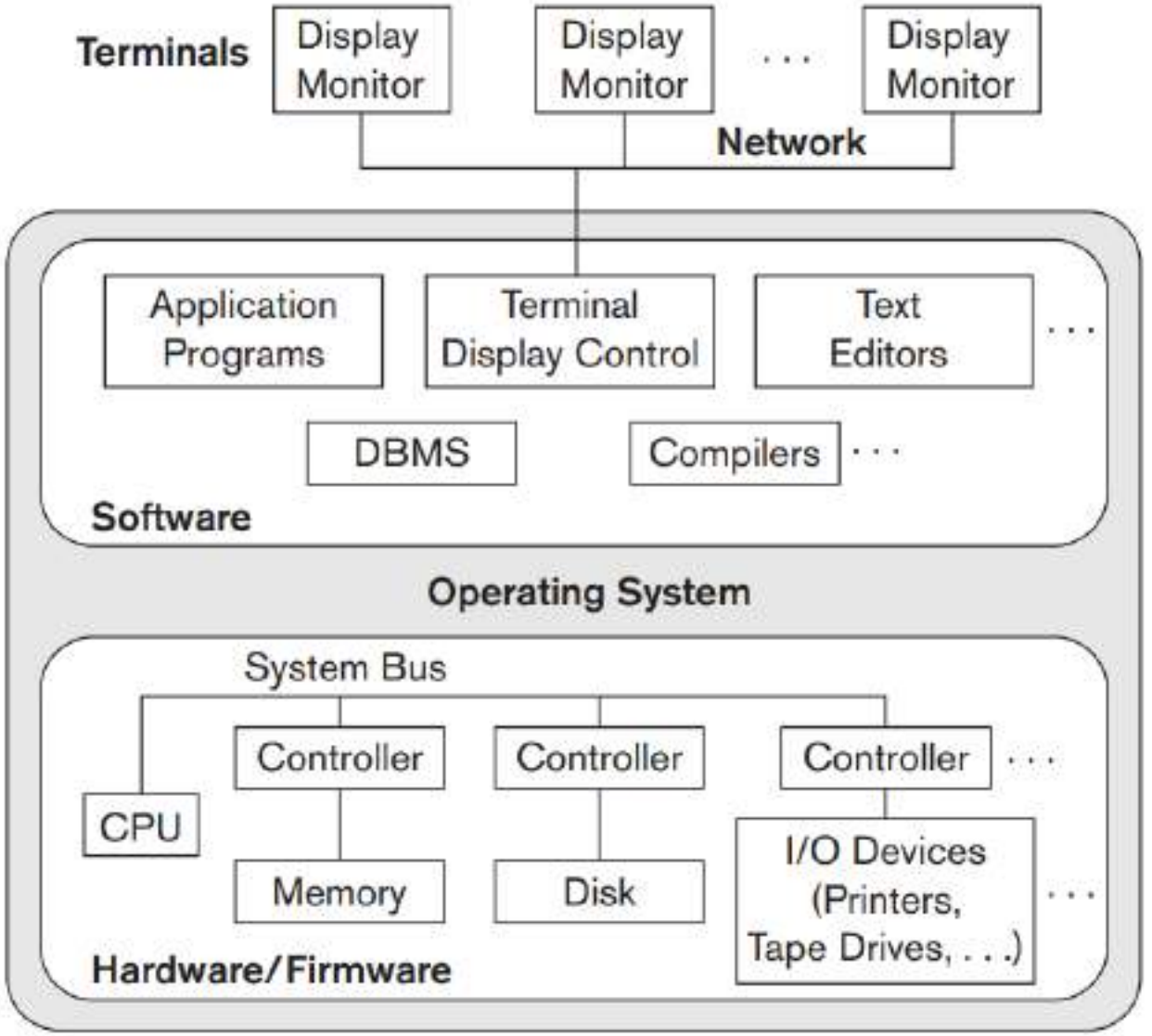

DIEBOLD
Choose a transaction
Press Cancel on the digital board to exit
Quick withdrawal
Cash transactions
Transfer «Khutka na kartku»
Balance or account statement
Other services
Payments
Online-Banking


Find a nearby cafe with WiFi and vegetarian options that's open after 10 pm

2.5 Centralized and Client/Server Architectures for DBMSs

2.5.1 Centralized DBMSs Architecture

Architectures for DBMSs have followed trends similar to those for general computer system architectures. Older architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality. The reason was that in older systems, most users accessed the DBMS via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system housing the DBMS, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.

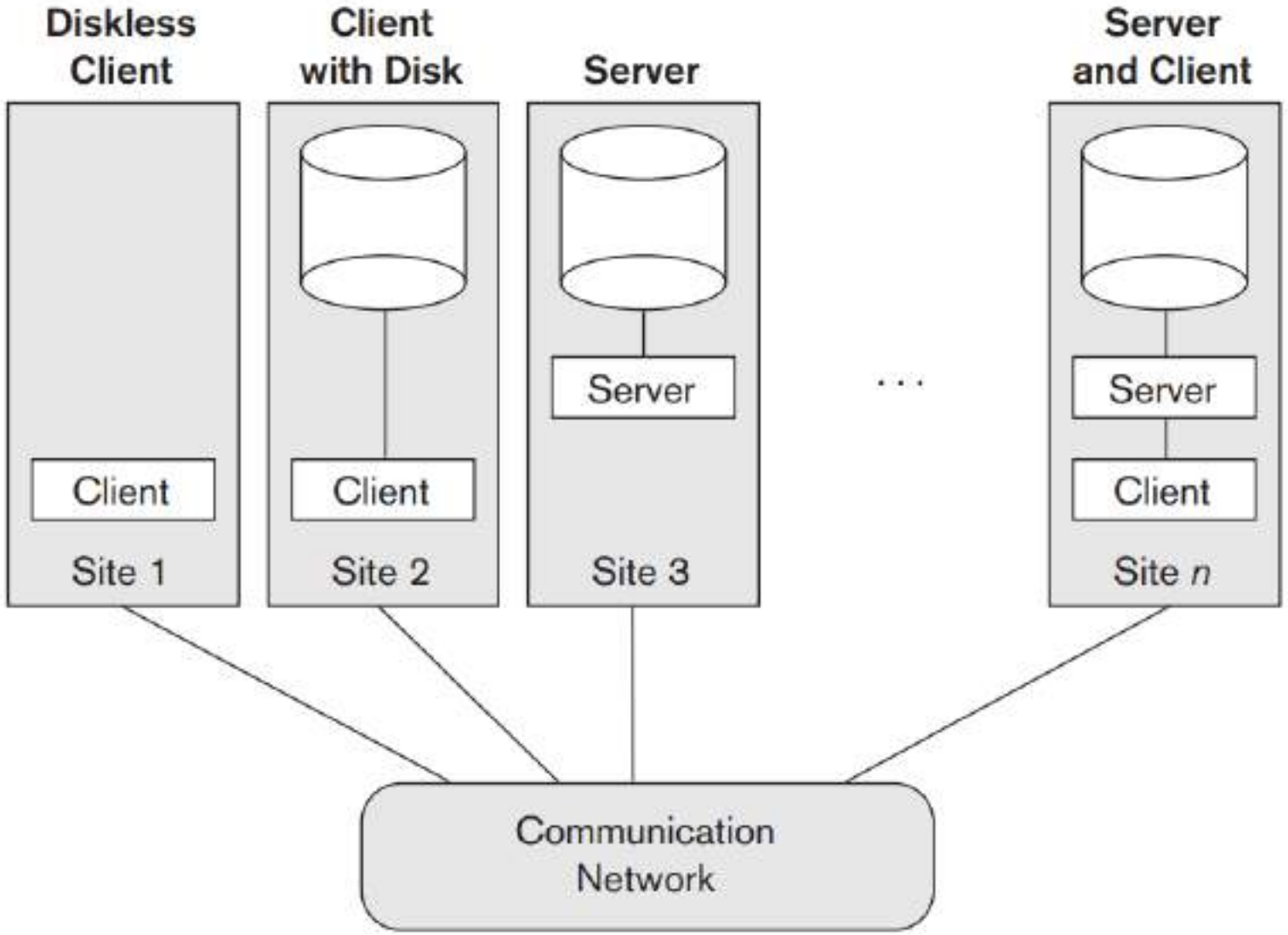

Terminals
Display Monitor
Display Monitor
Display Monitor
Network
Application Programs
Terminal Display Control
Text Editors
DBMS
Compilers
Software
Operating System
System Bus
Controller
Controller
Controller
CPU
Memory
Disk
I/O Devices (Printers, Tape Drives, . . .)
Hardware/Firmware

2.5.2 Basic Client/Server Architectures

First, we discuss client/server architecture in general; then we discuss how it is applied to DBMSs. The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network. The idea is to define **specialized servers** with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a **file server** that maintains the files of the client machines. Another machine can be designated as a **printer server** by being connected to various printers; all print requests by the clients are forwarded to this machine. **Web servers** or **e-mail servers** also fall into the specialized server category. The resources provided by specialized servers can be accessed by many client machines. The **client machines** provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications. This concept can be carried over to other software packages, with specialized programs—such as a CAD (computer-aided design) package—being stored on specific server machines and being made accessible to multiple clients. Figure 2.5 illustrates

The concept of client/server architecture assumes an underlying framework that consists of many PCs/workstations and mobile devices as well as a smaller number of server machines, connected via wireless networks or LANs and other types of computer networks. A **client** in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality—such as database access—that does not exist at the client, it connects to a server that provides the needed functionality. A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access. In general, some machines install only client software, others only server software, and still others may include both client and server software, as illustrated in Figure 2.6. However, it is more common that client and server software usually run on separate

Figure 2.6
Physical two-tier client/server architecture.



2.5.3 Two-Tier Client/Server Architectures for DBMSs

The architectures described here are called **two-tier architectures** because the software components are distributed over two systems: client and server. The advantages of this architecture are its simplicity and seamless compatibility with existing systems. The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.

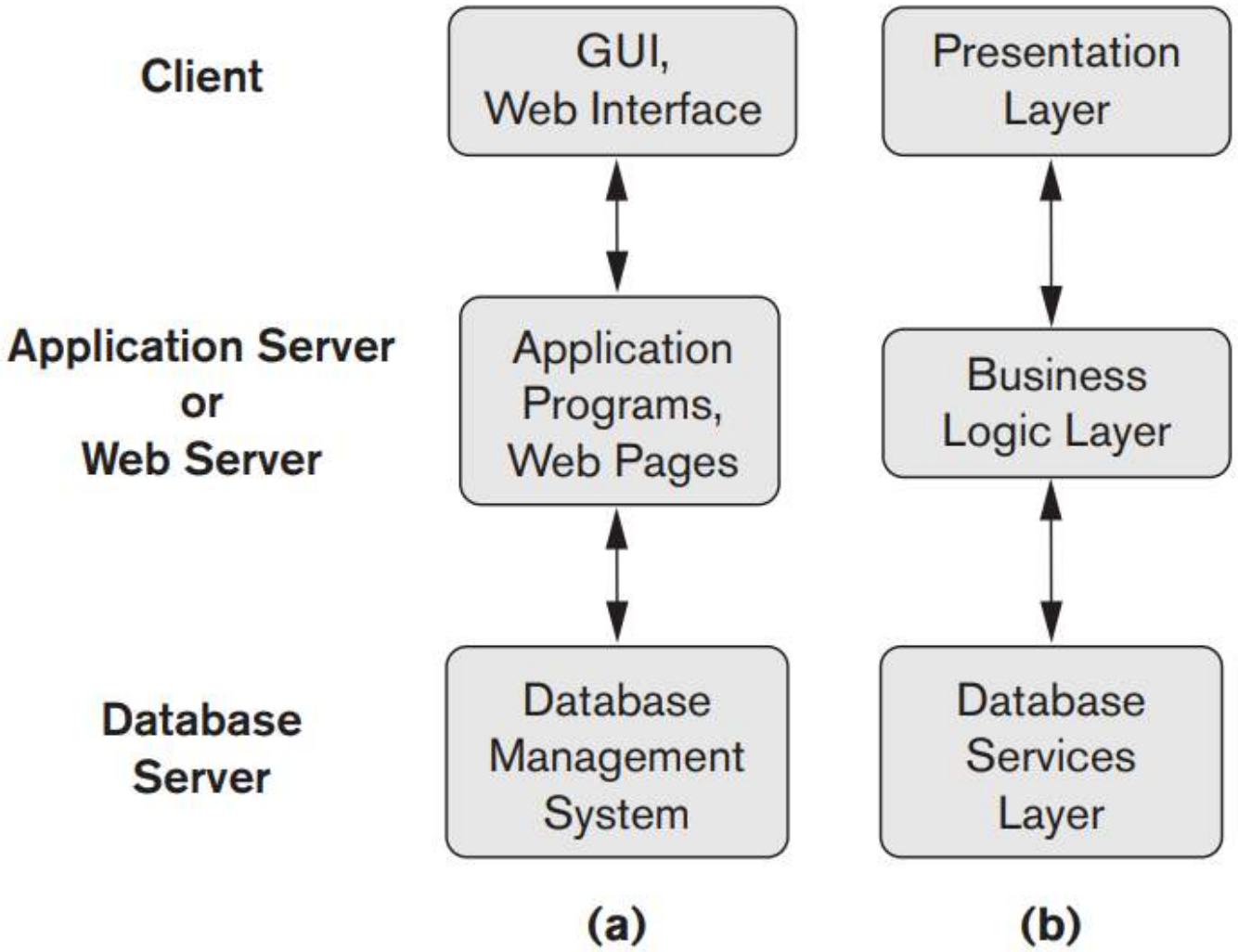


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

2.6 Classification of Database Management Systems

Database Management Systems (DBMS) can be classified based on several criteria such as data model, structure, and architecture. Here's a detailed classification of DBMS:

- **1. Based on Data Model**
- This classification is done based on how the data is modeled and structured.
- **a. Hierarchical DBMS**
- **Structure**: Data is organized in a tree-like structure.
- **Example**: IBM Information Management System (IMS).
- **Use Cases**: Banking and telecommunication industries.

- **b. Network DBMS**
- **Structure**: Data is represented as a graph, It's more flexible than the hierarchical model.
- **Example**: Integrated Data Store (IDS), CA-IDMS.
- **Use Cases**: Real-time systems, telecom networks.

c. Relational DBMS (RDBMS)

- **Structure**: Data is stored in tables (relations), SQL is used for querying.
- **Example**: MySQL, PostgreSQL, Oracle, SQL Server.
- **Use Cases**: Business data processing, web applications, ERP systems.

d. Object-Oriented DBMS (OODBMS)

- **Structure**: Data is stored in objects, similar to the object-oriented programming paradigm.
- **Example**: ObjectDB.
- **Use Cases**: Complex data models, multimedia databases, CAD applications.

e. Object-Relational DBMS (ORDBMS)

- **Structure**: Combines features of both relational and object-oriented databases. It supports objects, classes, and inheritance but uses relational concepts like tables.
- **Example**: PostgreSQL, Oracle.
- **Use Cases**: Applications needing complex data types, such as GIS and multimedia systems.

f. NoSQL DBMS

- **Structure**: Non-relational database models that provide flexibility in schema design and scalability. Common subtypes include document stores, key-value stores, column-family stores, and graph databases.
- **Example**: MongoDB (document store), Cassandra (column store), Redis (key-value store), Neo4j (graph database).
- **Use Cases**: Big data applications, real-time analytics, social media platforms.

2. Based on Number of Users

a. Single-User DBMS

•Only one user at a time can access the database. Typically used in personal or desktop systems.

•**Example**: Microsoft Access.

b. Multi-User DBMS

•Multiple users can access and manipulate the database simultaneously, ensuring concurrency and consistency.

•**Example**: MySQL, Oracle, PostgreSQL.

3. Based on Architecture

a. Centralized DBMS

•The database is stored and managed on a single server, with users accessing it over a network. All operations are handled centrally.

•**Example**: IBM Mainframe-based systems.

b. Distributed DBMS

•Data is distributed across multiple sites, either within the same network or across different locations. Each site has its own local DBMS, and the system appears as a single database to users.

•**Example**: Google Spanner, Apache Cassandra.

•**Types**:

- **Homogeneous**: All sites use the same DBMS.
- **Heterogeneous**: Different DBMS at each site.

- **c. Client-Server DBMS**
- The DBMS is split into two parts: the client (front-end) and the server (back-end). Clients send requests, and the server processes them.
- **Example**: MySQL, Microsoft SQL Server.

- **d. Cloud DBMS**
- The DBMS is hosted on a cloud infrastructure, allowing for scalability, flexibility, and reduced management overhead. Data is accessed through cloud service providers.
- **Example**: Amazon RDS, Google Cloud SQL, Microsoft Azure SQL Database.

4. Based on Data Storage and Access Method

a. In-Memory DBMS

- Stores data in main memory (RAM) rather than on disk to provide faster read and write operations.
- **Example**: Redis, SAP HANA.
- **Use Cases**: Real-time analytics, financial applications, gaming.

b. Disk-Based DBMS

- Data is stored on disks, and operations are slower compared to in-memory systems due to disk I/O operations.
- **Example**: Traditional RDBMS like Oracle, MySQL.

5. Based on Scale and Use

a. Desktop DBMS

- Meant for single-user applications, typically installed on personal computers. The database size and complexity are generally small.
- **Example**: Microsoft Access, SQLite.

b. Enterprise DBMS

- Designed for large-scale, mission-critical applications with high availability, scalability, and security features. Supports complex queries and multiple users.
- **Example**: Oracle, IBM DB2.

- **6. Based on License and Cost**
- **a. Open-Source DBMS**
- Free and open to modifications, allowing users to inspect, modify, and enhance the source code.
- **Example**: MySQL, PostgreSQL, MongoDB.
- **b. Proprietary DBMS**
- Licensed and controlled by a vendor. Users typically have to pay for licenses and support.
- **Example**: Oracle, Microsoft SQL Server.

2.6 Classification of Database Management Systems

Several criteria can be used to classify DBMSs. The first is the **data model** on which the DBMS is based. The main data model used in many current commercial DBMSs is the **relational data model**, and the systems based on this model are known as **SQL systems.** The **object data model** has been implemented in some commercial systems but has not had widespread use. Recently, so-called **big data systems,** also known as **key-value storage systems** and **NOSQL systems,** use various data models: **document-based, graph-based, column-based,** and **key-value data models.** Many legacy applications still run on database systems based on the **hierarchical** and **network data models.**

The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called **object-relational DBMS**s. We can categorize DBMSs based on the data model: relational, object, object-relational, NOSQL, key-value, hierarchical, network, and other.

Some experimental DBMSs are based on the XML (eXtended Markup Language) model, which is a **tree-structured data model.** These have been called **native XML DBMSs.** Several commercial relational DBMSs have added XML interfaces and storage to their products.

The second criterion used to classify DBMSs is the **number of users** supported by the system. **Single-user systems** support only one user at a time and are mostly used with PCs. **Multiuser systems**, which include the majority of DBMSs, support concurrent multiple users.

The third criterion is the **number of sites** over which the database is distributed. A DBMS is **centralized** if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site. A **distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites connected by a computer network. Big data systems are often massively distributed, with hundreds of sites. The data is often replicated on multiple sites so that failure of a site will not make some data unavailable.

Homogeneous DDBMSs use the same DBMS software at all the sites, whereas **heterogeneous** DDBMSs can use different DBMS software at each site. It is also possible to develop **middleware software** to access several autonomous preexisting databases stored under heterogeneous DBMSs. This leads to a **federated** DBMS (or **multidatabase system**), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DDBMSs use client-server architecture, as we described in Section 2.5.

The fourth criterion is cost. It is difficult to propose a classification of DBMSs based on cost. Today we have open source (free) DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services. The main RDBMS products are available as free examination 30-day copy versions as well as personal versions, which may cost under $100 and allow a fair amount of functionality. The giant systems are being sold in modular form with components to handle distribution, replication, parallel processing, mobile capability, and so on, and with a large number of parameters that must be defined for the configuration. Furthermore, they are sold in the form of licenses—site licenses allow unlimited use of the database system with any number of copies running at the customer site. Another type of license limits the number of concurrent users or the number of user seats at a location. Standalone single-user versions of some systems like Microsoft Access are sold per copy or included in the overall configuration of a desktop or laptop. In addition, data warehousing and mining features, as well as support for additional data types, are made available at extra cost. It is possible to pay millions of dollars for the installation and maintenance of large database systems annually.

We can also classify a DBMS on the basis of the **types of access path** options for storing files. One well-known family of DBMSs is based on inverted file structures. Finally, a DBMS can be **general purpose** or **special purpose**. When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and telephone directory systems developed in the past are special-purpose DBMSs. These fall into the category of **online transaction processing** (**OLTP**) systems, which must support a large number of concurrent transactions without imposing excessive delays.

العملي

د.محمد خلدون الطالب

د. شيماء أحمد رزوقي


Microsoft®
SQL Server®
TM

What is SQL Server?

- SQL Server is a relational database management system (RDBMS) developed by Microsoft. It is designed to store, retrieve, and manage data in a structured format using SQL (Structured Query Language). Here are some key features and components of SQL Server:

- **Key Features**
- **Relational Database Management**:
    - SQL Server organizes data into tables, which can be related to each other through foreign keys.
- **SQL Language Support**:
    - It uses T-SQL (Transact-SQL), an extension of SQL, for querying and managing the database.
- **Data Integrity and Security**:
    - It provides robust security features, including user authentication, role-based access control, and encryption.
- **High Availability and Disaster Recovery**:
    - SQL Server includes features like Always On Availability Groups, replication, and backup/restore capabilities to ensure data availability.
- **Performance Optimization**:
    - It offers tools for query optimization, indexing, and in-memory processing to improve performance.
- **Business Intelligence**:
    - SQL Server integrates with tools like SQL Server Reporting Services (SSRS) and SQL Server Analysis Services (SSAS) for reporting and data analysis.
- **Integration Services**:
    - SQL Server Integration Services (SSIS) allows for data extraction, transformation, and loading (ETL) processes.
- **Scalability**:
    - It can handle large volumes of data and is suitable for small applications as well as enterprise-level solutions.

- **Editions**
- SQL Server is available in several editions, including:
- **SQL Server Express**: A free, lightweight edition suitable for small applications.
- **SQL Server Standard**: Offers core database capabilities for small to medium-sized applications.
- **SQL Server Enterprise**: Provides advanced features for large-scale applications and data warehouses.

Install SQL Server 2022 and SSMS 19


SQL

- **SQL (Structured Query Language)** is a standardized programming language used to manage and manipulate relational databases. There are several types and categories of SQL, each serving different purposes. Here are the main types:

- **1. Data Query Language (DQL)**
- **Purpose**: Used to query the database and retrieve data.
- **Key Commands**:
  - SELECT: Retrieves data from one or more tables.

- **2. Data Definition Language (DDL)**
- **Purpose**: Used to define and manage all structures in a database.
- **Key Commands**:
  - CREATE: Creates a new database object (table, index, etc.).
  - ALTER: Modifies an existing database object.
  - DROP: Deletes an existing database object.
  - TRUNCATE: Removes all records from a table but retains the structure.

- **3. Data Manipulation Language (DML)**
- **Purpose**: Used to manipulate data stored in the database.
- **Key Commands**:
  - INSERT: Adds new records to a table.
  - UPDATE: Modifies existing records in a table.
  - DELETE: Removes records from a table.

- **4. Data Control Language (DCL)**
- **Purpose**: Used to control access to data in the database.
- **Key Commands**:
  - GRANT: Gives users access privileges to the database.
  - REVOKE: Removes access privileges from users.

- **5. Transaction Control Language (TCL)**
- **Purpose**: Used to manage transactions in the database.
- **Key Commands**:
  - COMMIT: Saves all changes made during the current transaction.
  - ROLLBACK: Undoes changes made during the current transaction.
  - SAVEPOINT: Sets a point to which you can later roll back.

- **6. Embedded SQL**
- **Purpose**: Allows SQL statements to be embedded within other programming languages (e.g., C, Java).
- **Key Characteristics**: Provides a way to execute SQL commands from within applications.

- **7. Procedural SQL**
- **Purpose**: Extends SQL with procedural programming capabilities.
- **Key Characteristics**: Includes control structures like loops and conditionals, often used in stored procedures and functions (e.g., PL/SQL for Oracle, T-SQL for SQL Server).

Summary

Each type of SQL serves a specific role in database management, from querying and defining data structures to controlling access and managing transactions. Understanding these types is essential for effective database manipulation and management.

Version of SQL?

- SQL itself is a standard language for managing relational databases, but various database management systems (DBMS) implement their own versions of SQL, often with additional features and extensions. Here are some of the most notable versions and implementations of SQL:

1. ANSI SQL

•**Description**: The American National Standards Institute (ANSI) has established standards for SQL, with the first standard published in 1986.

•**Versions**:

- SQL-89: The first version standardized by ANSI.
- SQL-92: Introduced significant enhancements, including new data types and integrity constraints.
- SQL:1999 (SQL3): Added support for object-oriented features, triggers, and recursive queries.
- SQL:2003: Introduced XML support and window functions.
- SQL:2008: Added features like additional XML enhancements and new standard functions.
- SQL:2011: Introduced temporal data support.
- SQL:2016: Added JSON support and more.

2. T-SQL (Transact-SQL)
•**Description**: An extension of SQL used primarily with Microsoft SQL Server.
•**Features**: Includes procedural programming features, variables, error handling, and built-in functions.

3. PL/SQL (Procedural Language/SQL)
•**Description**: A procedural extension of SQL used in Oracle databases.
•**Features**: Supports procedural constructs like loops, conditionals, and exception handling.

4. MySQL
•**Description**: An open-source relational database management system that uses its own version of SQL.
•**Features**: Includes various extensions for performance optimization and replication.

5. PostgreSQL
Description: An advanced open-source relational database that implements a rich version of SQL.
Features: Supports advanced data types, full-text search, and custom functions.

6. SQLite
Description: A lightweight, file-based database system that uses a simplified version of SQL.
Features: Supports most SQL standards but lacks some advanced features found in larger DBMSs.

7. IBM Db2 SQL
Description: The SQL dialect used in IBM's Db2 database system.
Features: Includes support for advanced data types and optimization features.

01 – Installing MS SQL Server & Management Studio

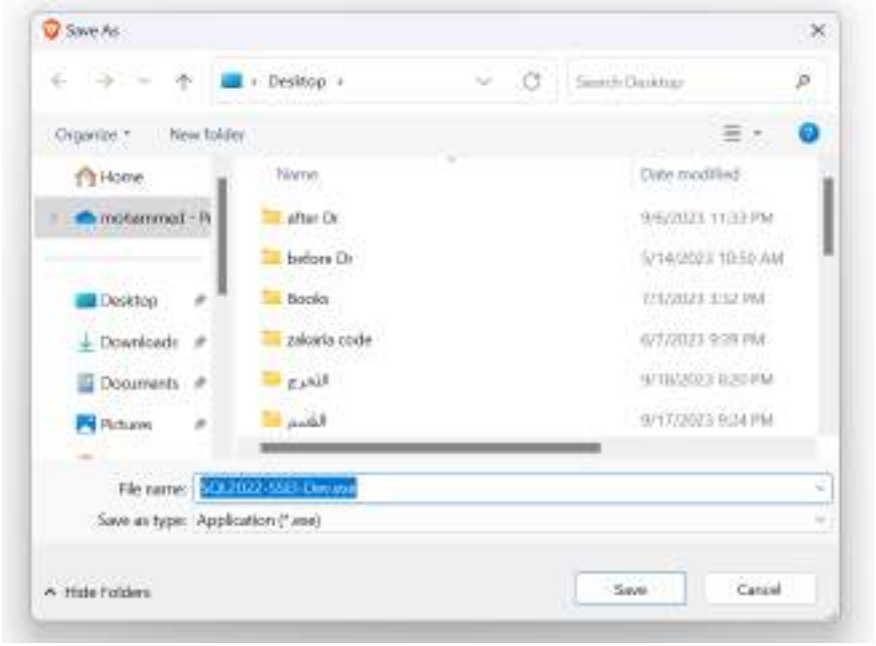
1

Developer

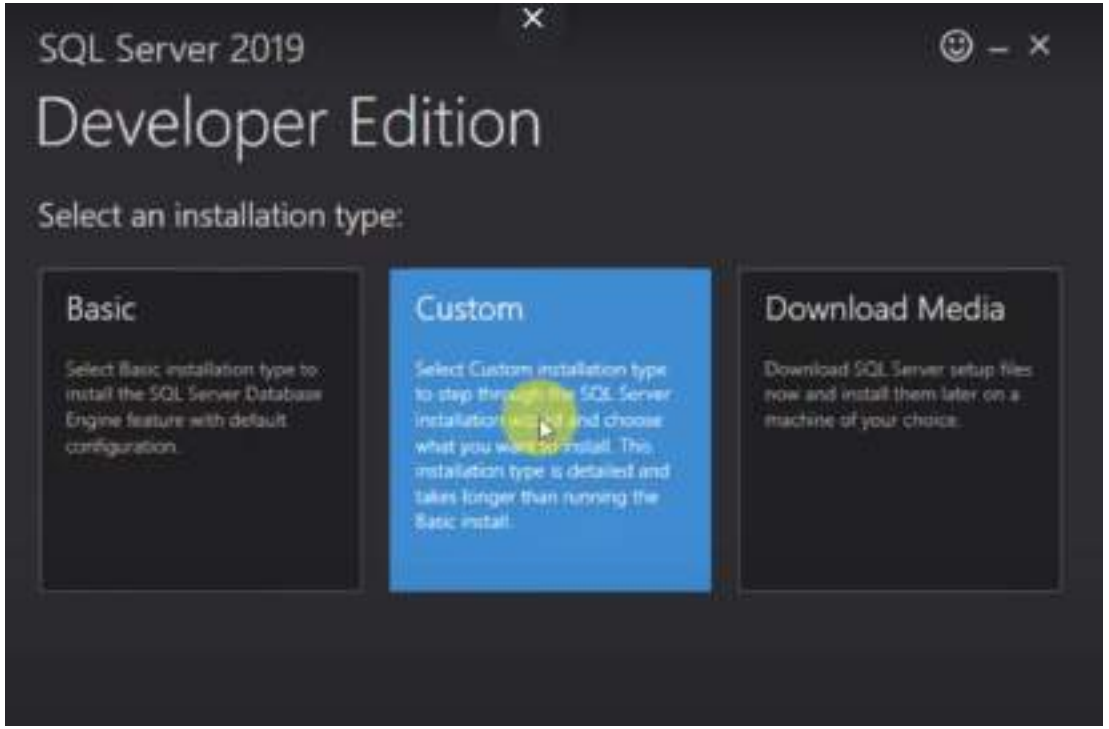
SQL Server 2022 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment.

Download now

2

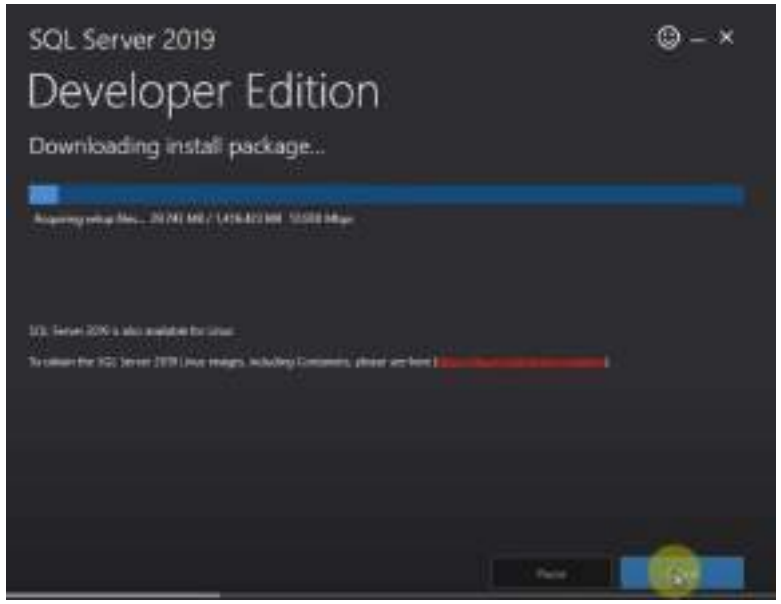


3

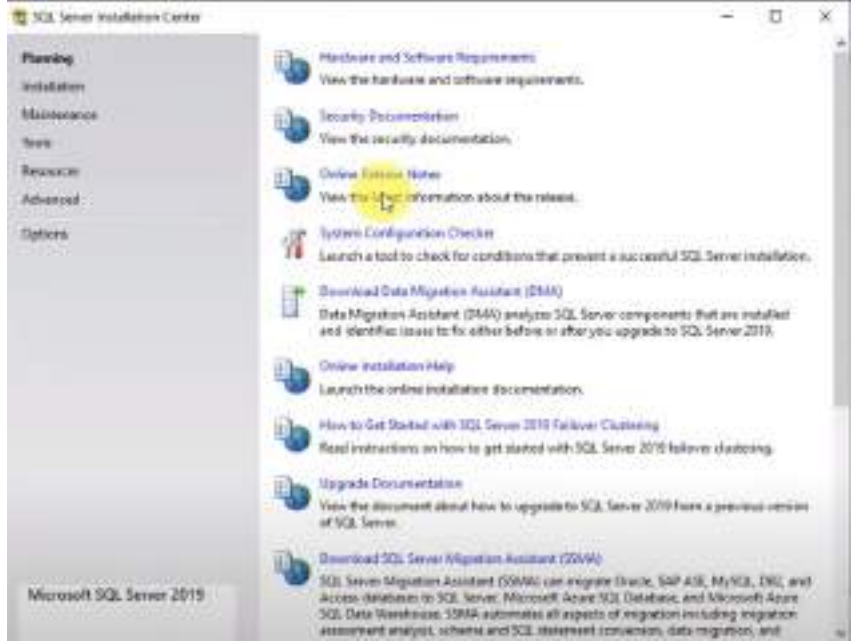


4

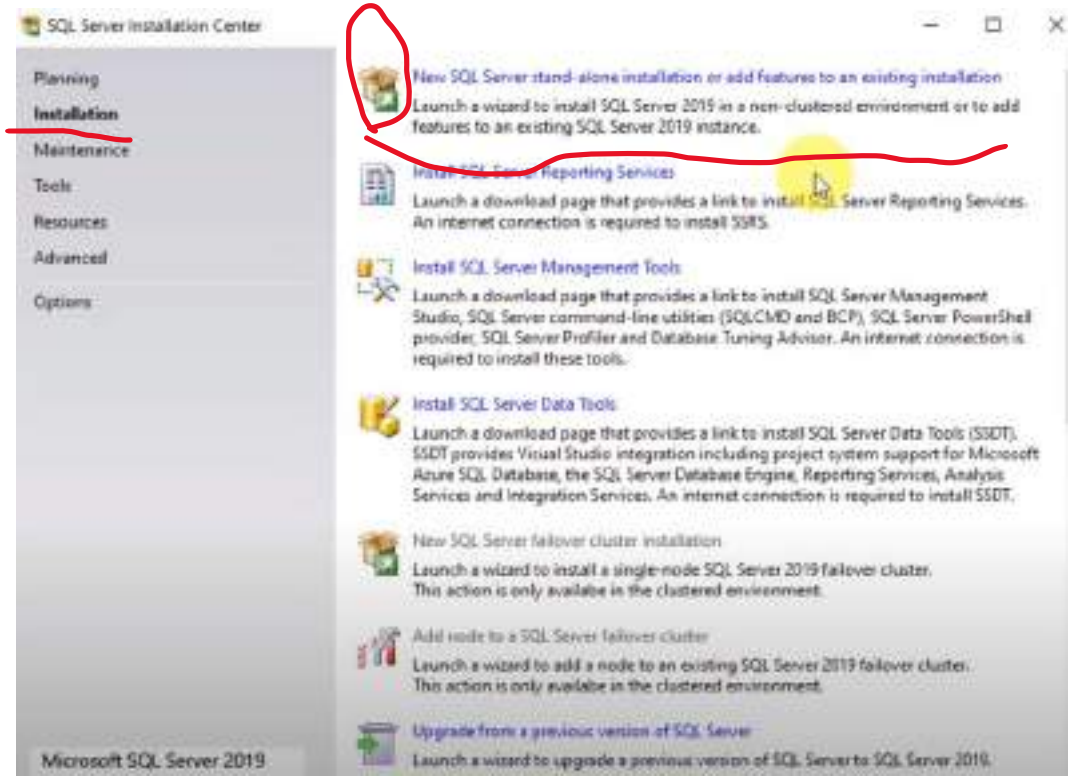
1


SQL Server 2019
Developer Edition
Downloading install package...

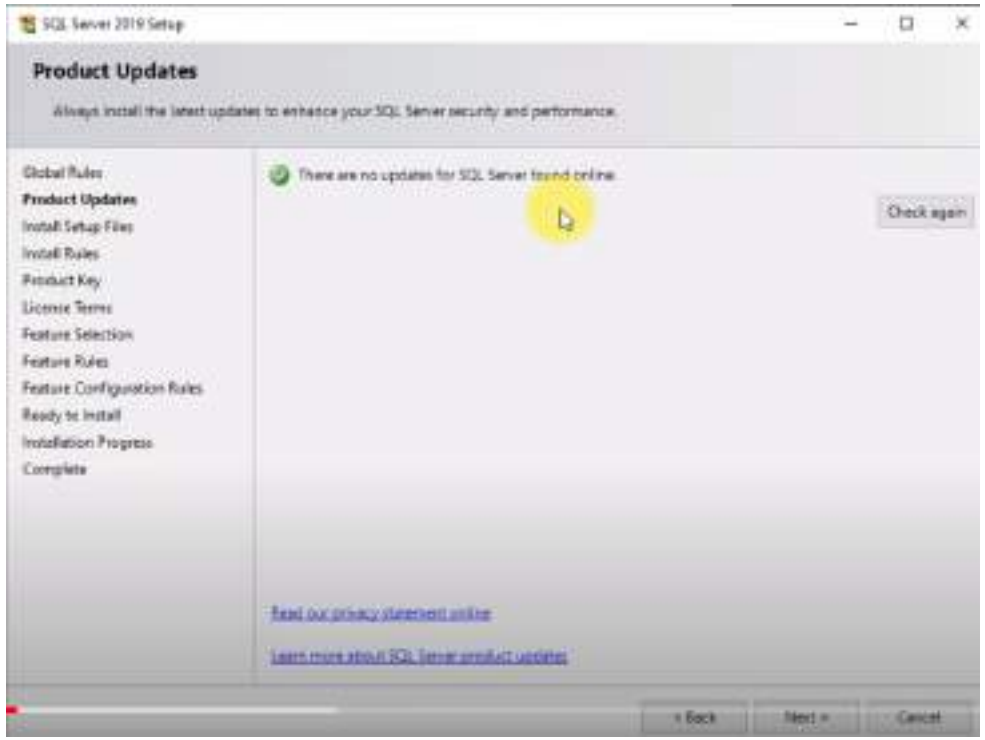
2


SQL Server Installation Center
Planning
Installation
Maintenance
Tools
Resources
Advanced
Options
Microsoft SQL Server 2019

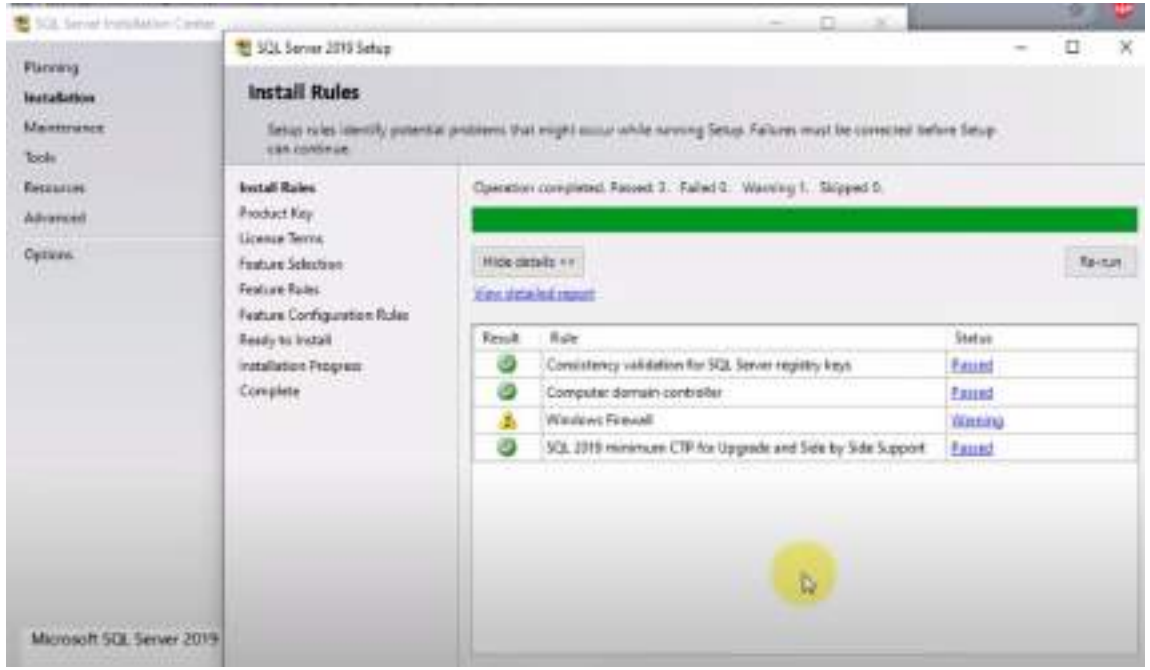
3


SQL Server Installation Center
Planning
Installation
Maintenance
Tools
Resources
Advanced
Options
New SQL Server stand-alone installation or add features to an existing installation
Launch a wizard to install SQL Server 2019 in a non-clustered environment or to add features to an existing SQL Server 2019 instance.
Launch a download page that provides a link to install SQL Server Reporting Services. An internet connection is required to install SSRS.
Install SQL Server Management Tools
Launch a download page that provides a link to install SQL Server Management Studio, SQL Server command-line utilities (SQLCMD and BCP), SQL Server PowerShell provider, SQL Server Profiler and Database Tuning Advisor. An internet connection is required to install these tools.
Install SQL Server Data Tools
Launch a download page that provides a link to install SQL Server Data Tools (SSDT). SSDT provides Visual Studio integration including project system support for Microsoft Azure SQL Database, the SQL Server Database Engine, Reporting Services, Analysis Services and Integration Services. An internet connection is required to install SSDT.
New SQL Server failover cluster installation
Launch a wizard to install a single-node SQL Server 2019 failover cluster. This action is only available in the clustered environment.
Add node to a SQL Server failover cluster
Launch a wizard to add a node to an existing SQL Server 2019 failover cluster. This action is only available in the clustered environment.
Upgrade from a previous version of SQL Server
Launch a wizard to upgrade a previous version of SQL Server to SQL Server 2019.
Microsoft SQL Server 2019

4


SQL Server 2019 Setup
Product Updates
Always install the latest updates to enhance your SQL Server security and performance.
Global Rules
Product Updates
Install Setup Files
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
There are no updates for SQL Server found online.
Check again
Read our privacy statement online
Learn more about SQL Server product updates
< Back
Next >
Cancel

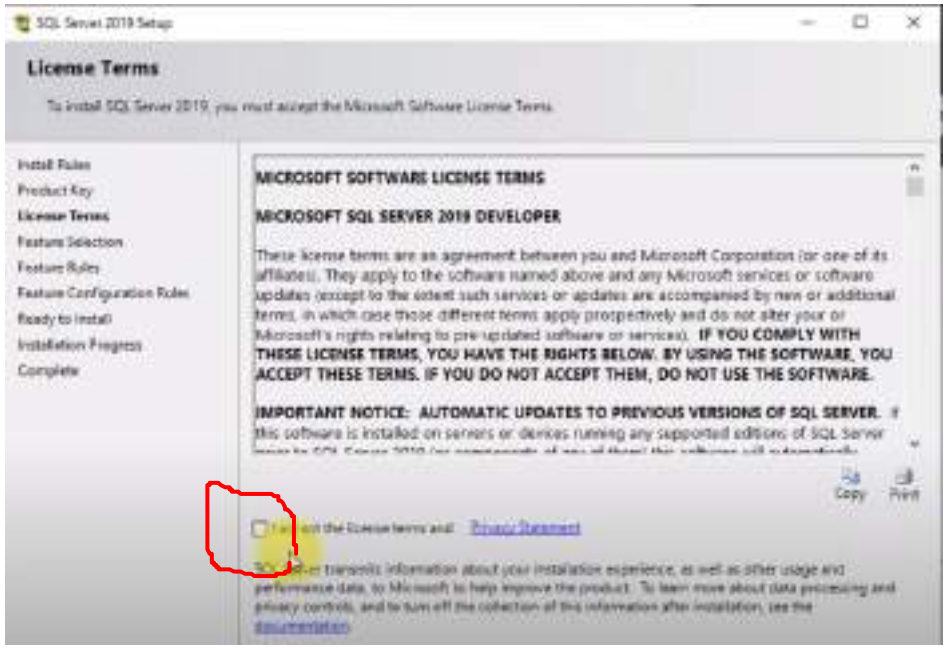
1


SQL Server 2019 Setup
Install Rules
Setup rules identify potential problems that might occur while running Setup. Failures must be corrected before Setup can continue.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Operation completed. Passed: 3. Failed: 0. Warning: 1. Skipped: 0.
Hide details <<
Re-run
View detailed report
Result
Rule
Status
Consistency validation for SQL Server registry keys
Passed
Computer domain controller
Passed
Windows Firewall
Warning
SQL 2019 minimum CTP for Upgrade and Side by Side Support
Passed
Microsoft SQL Server 2019

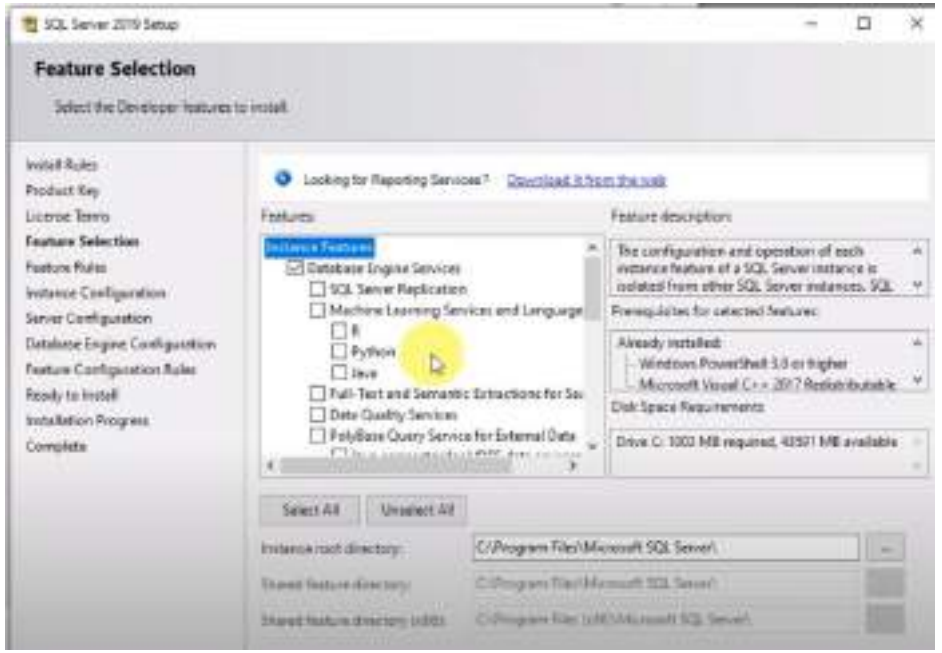
2

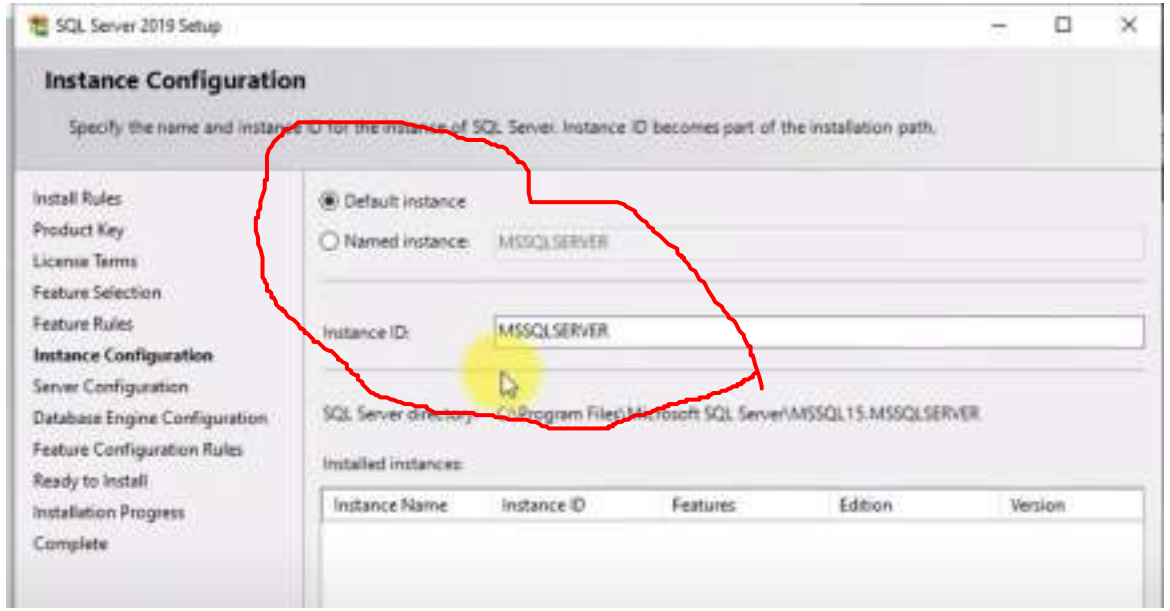

SQL Server 2019 Setup
Product Key
Specify the edition of SQL Server 2019 to install.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Validate this instance of SQL Server 2019 by entering the 25-character key from the Microsoft certificate of authenticity or product packaging. You can also specify a free edition of SQL Server: Developer, Evaluation, or Express. Evaluation has the largest set of SQL Server features, as documented in SQL Server Books Online, and is activated with a 180-day expiration. Developer edition does not have an expiration, has the same set of features found in Evaluation, but is licensed for non-production database application development only. To upgrade from one installed edition to another, run the Edition Upgrade Wizard.
Specify a free edition:
Developer
Enter the product key:

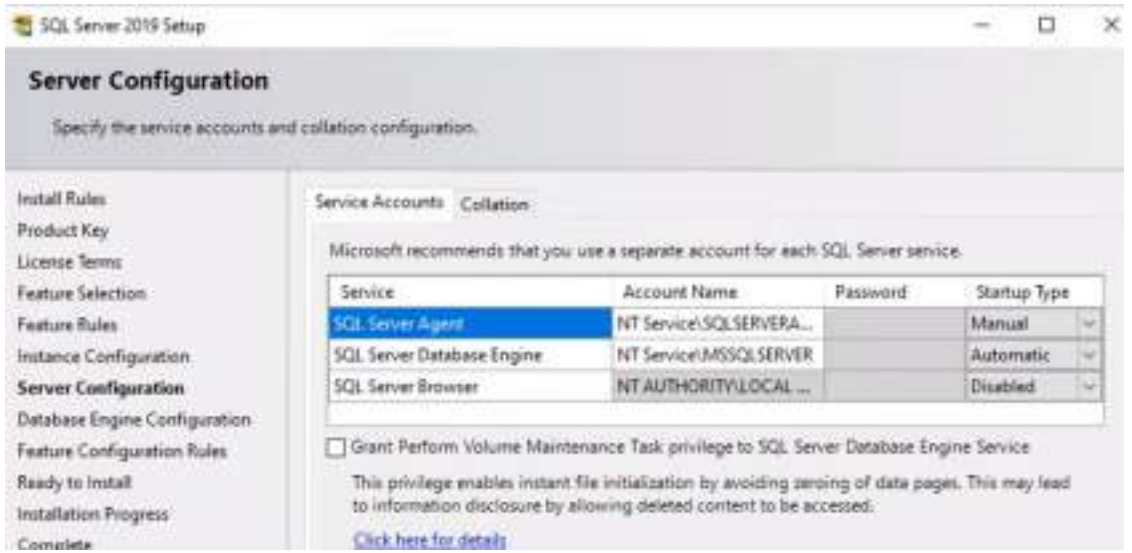
3

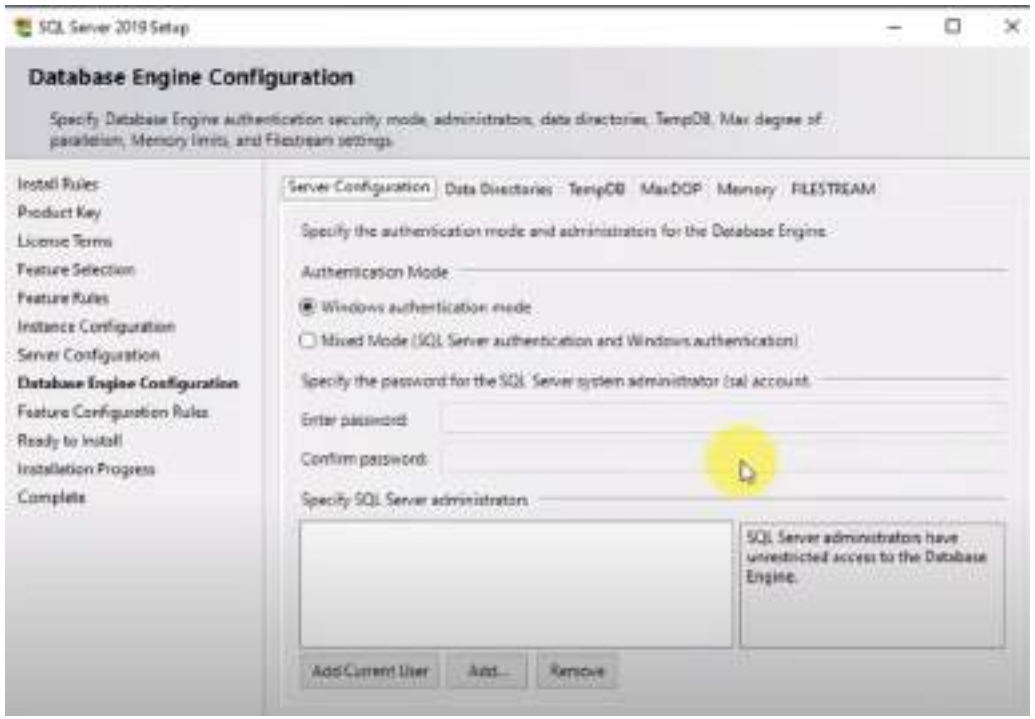

SQL Server 2019 Setup
License Terms
To install SQL Server 2019, you must accept the Microsoft Software License Terms.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT SQL SERVER 2019 DEVELOPER
These license terms are an agreement between you and Microsoft Corporation (or one of its affiliates). They apply to the software named above and any Microsoft services or software updates (except to the extent such services or updates are accompanied by new or additional terms, in which case those different terms apply prospectively and do not alter your or Microsoft's rights relating to pre-updated software or services). IF YOU COMPLY WITH THESE LICENSE TERMS, YOU HAVE THE RIGHTS BELOW. BY USING THE SOFTWARE, YOU ACCEPT THESE TERMS. IF YOU DO NOT ACCEPT THEM, DO NOT USE THE SOFTWARE.
IMPORTANT NOTICE: AUTOMATIC UPDATES TO PREVIOUS VERSIONS OF SQL SERVER. If this software is installed on servers or devices running any supported editions of SQL Server
Copy
Print
I accept the license terms and
Privacy Statement
documentation

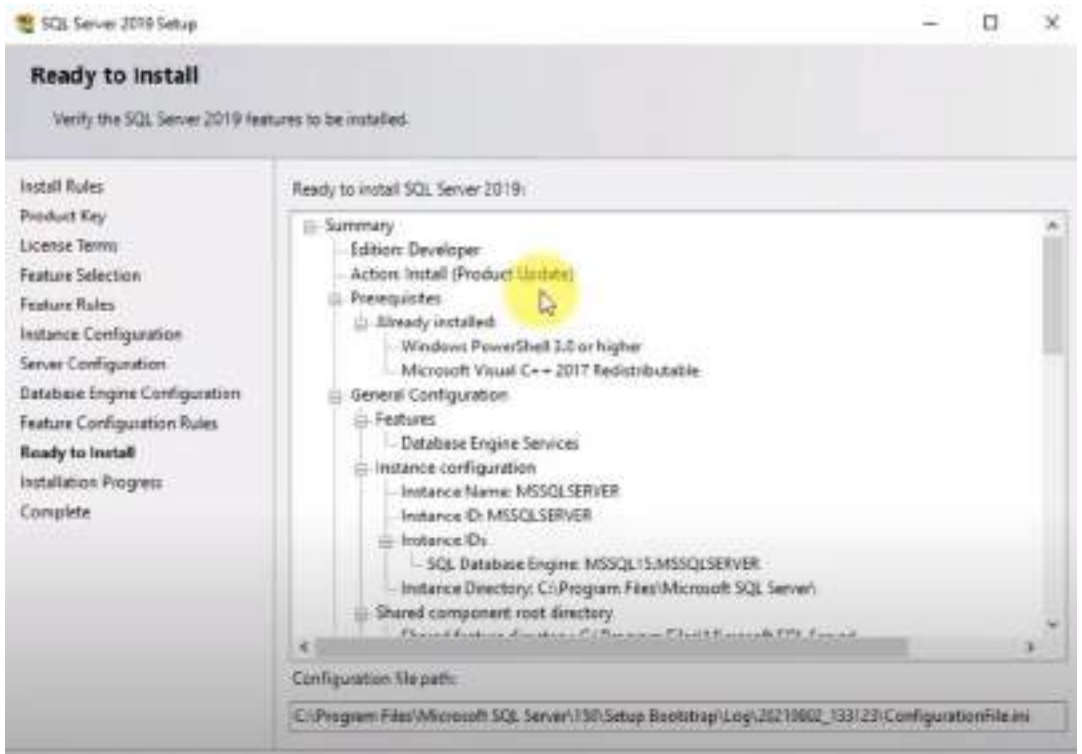
4


SQL Server 2019 Setup
Feature Selection
Select the Developer features to install.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Looking for Reporting Services?
Download it from the web
Features:
Feature description:
Instance Features
Database Engine Services
SQL Server Replication
Machine Learning Services and Language
R
Python
Java
Full-Text and Semantic Extractions for Search
Data Quality Services
PolyBase Query Service for External Data
The configuration and operation of each instance feature of a SQL Server instance is isolated from other SQL Server instances. SQL
Prerequisites for selected features:
Already installed:
Windows PowerShell 3.0 or higher
Microsoft Visual C++ 2017 Redistributable
Disk Space Requirements
Drive C: 1003 MB required, 43571 MB available
Select All
Unselect All
Instance root directory:
C:\Program Files\Microsoft SQL Server\
Shared feature directory:
Shared feature directory (x86):

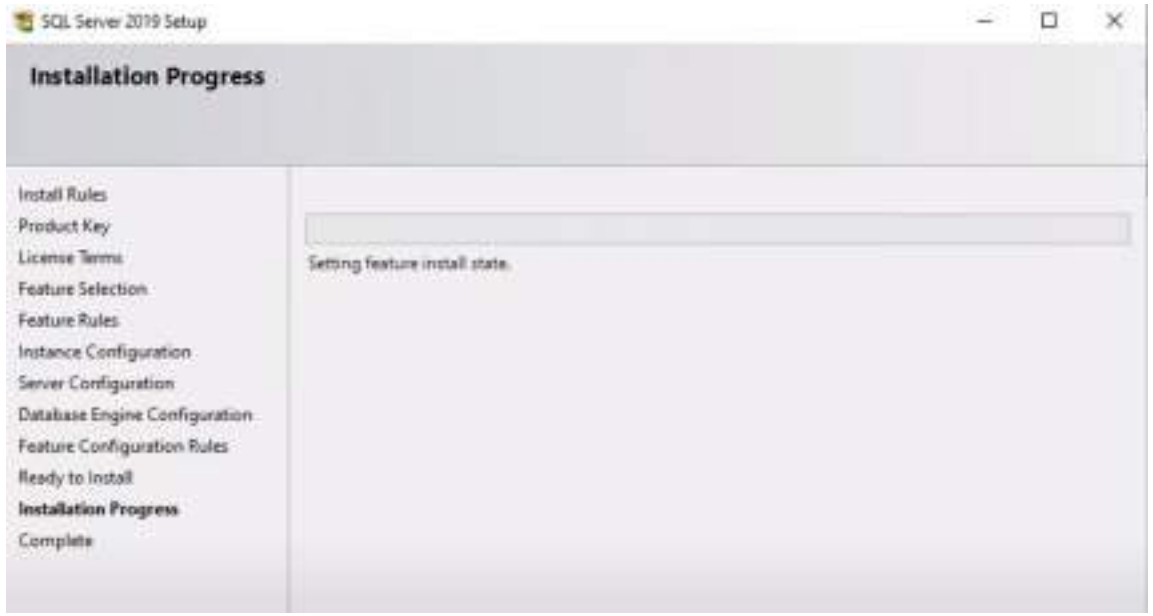

SQL Server 2019 Setup
Instance Configuration
Specify the name and instance ID for the instance of SQL Server. Instance ID becomes part of the installation path.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Default instance
Named instance:
MSSQLSERVER
Instance ID:
MSSQLSERVER
SQL Server directory:
C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER
Installed instances:
Instance Name
Instance ID
Features
Edition
Version


SQL Server 2019 Setup
Server Configuration
Specify the service accounts and collation configuration.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Service Accounts
Collation
Microsoft recommends that you use a separate account for each SQL Server service.
Service
Account Name
Password
Startup Type
SQL Server Agent
NT Service\SQLSERVERA...
Manual
SQL Server Database Engine
NT Service\MSSQLSERVER
Automatic
SQL Server Browser
NT AUTHORITY\LOCAL ...
Disabled
Grant Perform Volume Maintenance Task privilege to SQL Server Database Engine Service
This privilege enables instant file initialization by avoiding zeroing of data pages. This may lead to information disclosure by allowing deleted content to be accessed.
Click here for details

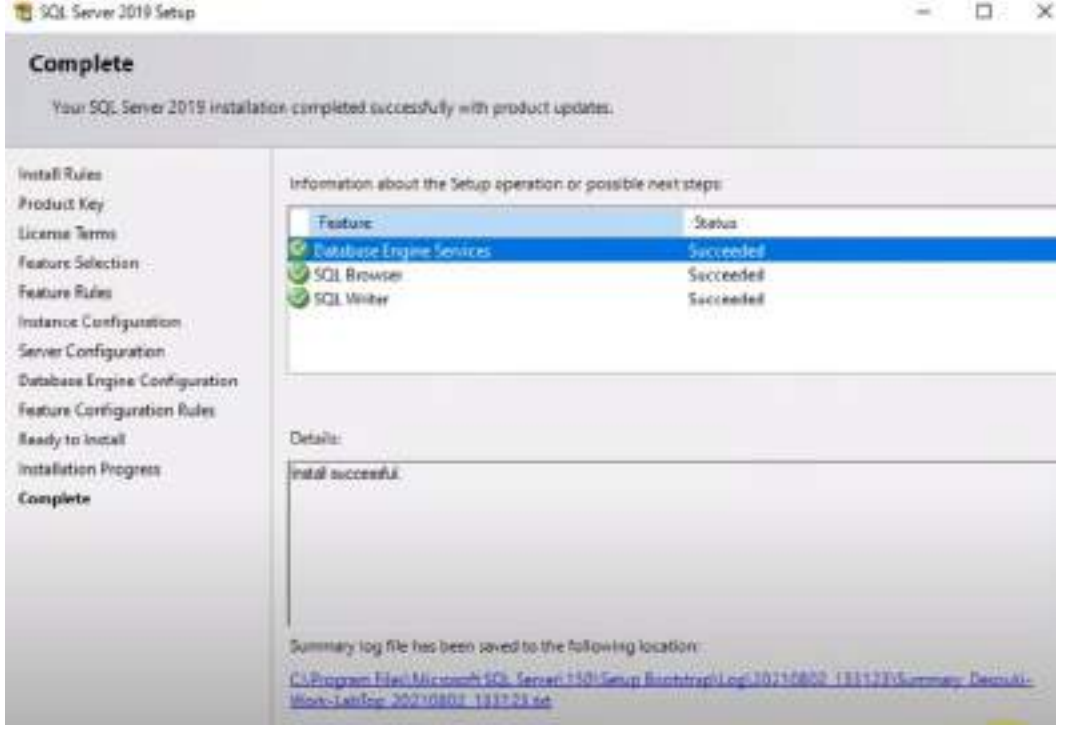

SQL Server 2019 Setup
Database Engine Configuration
Specify Database Engine authentication security mode, administrators, data directories, TempDB, Max degree of parallelism, Memory limits, and Filestream settings.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Server Configuration
Data Directories
TempDB
MaxDOP
Memory
FILESTREAM
Specify the authentication mode and administrators for the Database Engine.
Authentication Mode
Windows authentication mode
Mixed Mode (SQL Server authentication and Windows authentication)
Specify the password for the SQL Server system administrator (sa) account.
Enter password:
Confirm password:
Specify SQL Server administrators
SQL Server administrators have unrestricted access to the Database Engine.
Add Current User
Add...
Remove


SQL Server 2019 Setup
Ready to Install
Verify the SQL Server 2019 features to be installed.
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Ready to install SQL Server 2019:
Summary
Edition: Developer
Action: Install (Product Update)
Prerequisites
Already installed
Windows PowerShell 3.0 or higher
Microsoft Visual C++ 2017 Redistributable
General Configuration
Features
Database Engine Services
Instance configuration
Instance Name: MSSQLSERVER
Instance ID: MSSQLSERVER
Instance IDs
SQL Database Engine: MSSQL15.MSSQLSERVER
Instance Directory: C:\Program Files\Microsoft SQL Server\
Shared component root directory
Configuration file path:
C:\Program Files\Microsoft SQL Server\150\Setup Bootstrap\Log\20210802_133123\ConfigurationFile.ini

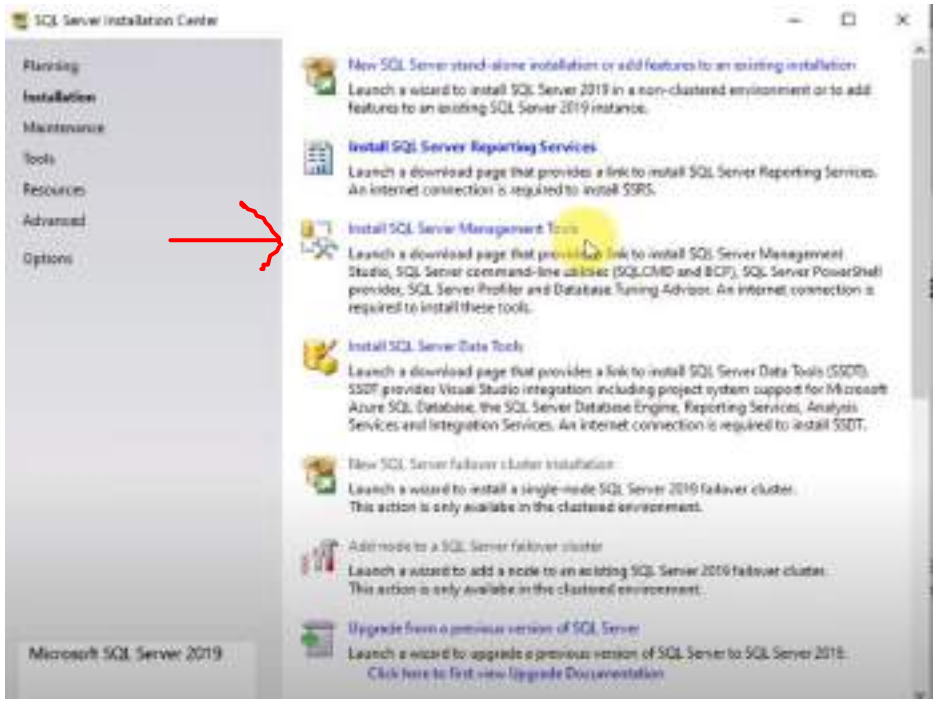
1


SQL Server 2019 Setup
Installation Progress
Install Rules
Product Key
License Terms
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Setting feature install state.

2


SQL Server 2019 Setup
Complete
Your SQL Server 2019 installation completed successfully with product updates.
Information about the Setup operation or possible next steps:
Feature
Status
Database Engine Services
Succeeded
SQL Browser
Succeeded
SQL Writer
Succeeded
Details:
Install succeeded.
Summary log file has been saved to the following location:

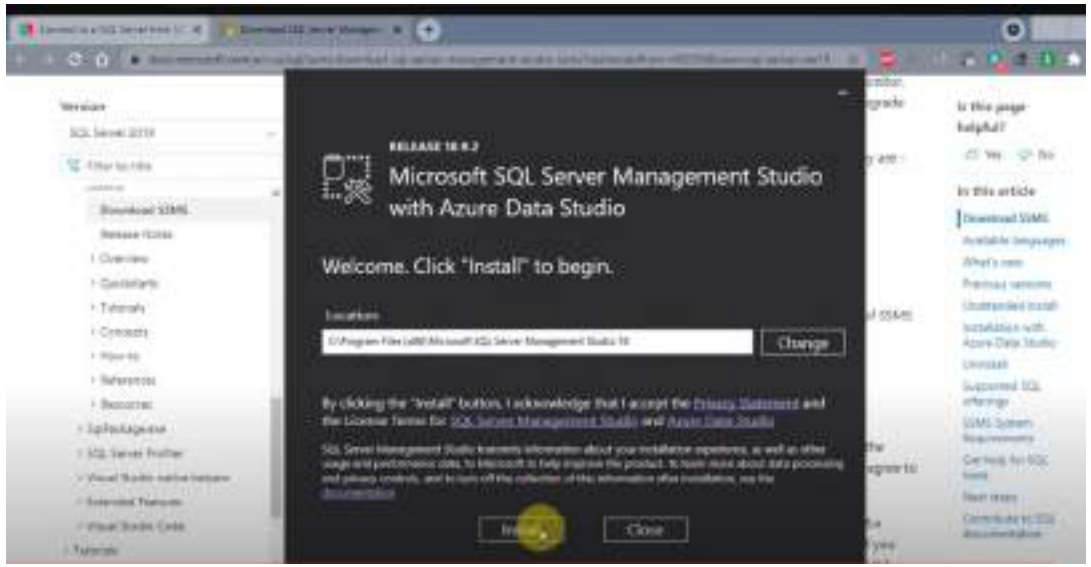
3


SQL Server Installation Center
Planning
Installation
Maintenance
Tools
Resources
Advanced
Options
Install SQL Server Management Tools
Install SQL Server Data Tools
Upgrade from a previous version of SQL Server
Microsoft SQL Server 2019

4

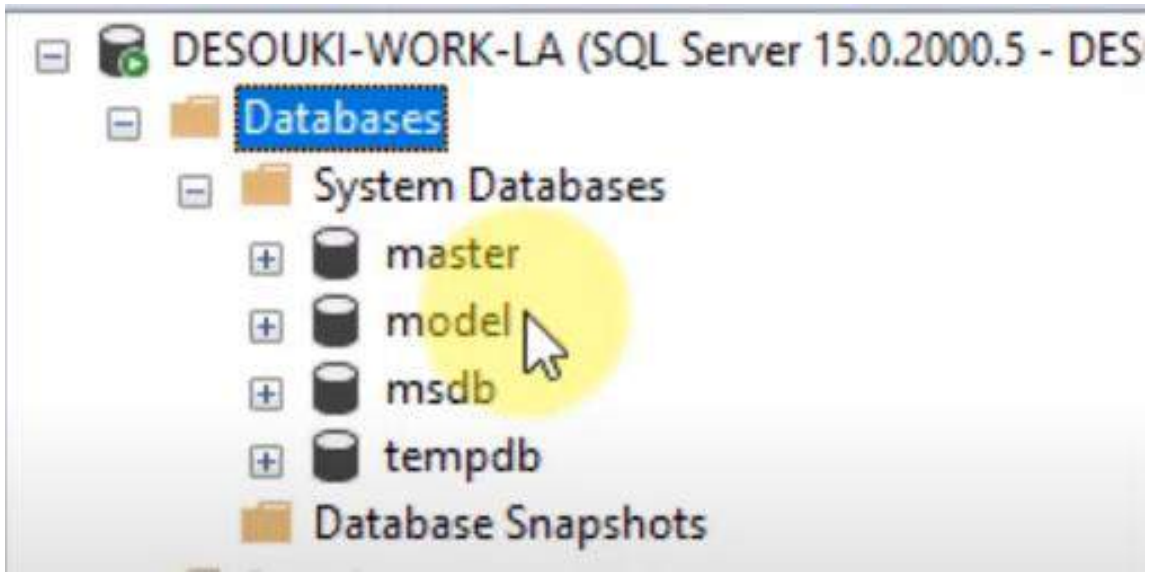

Version
SQL Server 2019
Download SSMS
Release Notes
Overview
Quickstarts
Tutorials
Concepts
How-to
References
Resources
SqlPackage.exe
SQL Server Profiler
Visual Studio native helpers
Extended Features
Visual Studio Code
Tutorials
and administer instances of SQL Server and databases. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications, and build queries and scripts.
Use SSMS to query, design, and manage your databases and data warehouses, wherever they are - on your local computer, or in the cloud.
Download SSMS
Download SQL Server Management Studio (SSMS) 18.9.2
SSMS 18.9.2 is the latest general availability (GA) version. If you have a previous GA version of SSMS 18 installed, installing SSMS 18.9.2 upgrades it to 18.9.2.
Release number: 18.9.2
Build number: 15.0.18386.0
Release date: July 15, 2021
If you have comments or suggestions, or you want to report issues, the best way to contact the SSMS team is at SQL Server user feedback. By using SQL Server Management Studio, you agree to its license terms and privacy statement.
The SSMS 18.x installation doesn't upgrade or replace SSMS versions 17.x or earlier. SSMS 18.x installs side by side with previous versions, so both versions are available for use. However, if you
Is this page helpful?
Yes
No
In this article
Download SSMS
Available languages
What's new
Previous versions
Unattended install
Installation with Azure Data Studio
Uninstall
Supported SQL offerings
SSMS System Requirements
Get help for SQL tools
Next steps
Contribute to SQL documentation

1


RELEASE 18.9.2
Microsoft SQL Server Management Studio
with Azure Data Studio
Welcome. Click "Install" to begin.
Location:
Change
Install
Close

2


RELEASE 18.9.2
Microsoft SQL Server Management Studio
with Azure Data Studio
Setup Completed
All specified components have been installed successfully.
Close



System database	Description
master Database	Records all the system-level information for an instance of SQL Server.
msdb Database	Is used by SQL Server Agent for scheduling alerts and jobs.
model Database	Is used as the template for all databases created on the instance of SQL Server. Modifications made to the **model** database, such as database size, collation, recovery model, and other database options, are applied to any databases created afterward.
Resource Database	Is a read-only database that contains system objects that are included with SQL Server. System objects are physically persisted in the **Resource** database, but they logically appear in the **sys** schema of every database.
tempdb Database	Is a workspace for holding temporary objects or intermediate result sets.

Create-SQL

Lab_2

https://www.sqlservertutorial.net/

- Section 12. Data definition
- This section shows you how to manage the most important database objects including databases and tables.
- CREATE DATABASE – show you how to create a new database in an SQL Server instance using the CREATE DATABASE statement and SQL Server Management Studio.
- DROP DATABASE – learn how to delete existing databases.
- CREATE SCHEMA – describe how to create a new schema in a database.
- ALTER SCHEMA – show how to transfer a securable from one schema to another within the same database.
- DROP SCHEMA – learn how to delete a schema from a database.

Creating a new database using the CREATE DATABASE statement

The `CREATE DATABASE` statement creates a new database. The following shows the minimal syntax of the `CREATE DATABASE` statement:

```
CREATE DATABASE database_name;
```

In this syntax, you specify the name of the database after the `CREATE DATABASE` keyword.

The database name must be unique within an instance of SQL Server. It must also comply with the SQL Server identifier's rules. Typically, the database name has a maximum of 128 characters.
The following statement creates a new database named `TestDb` :

```
CREATE DATABASE TestDb;
```

Using the SQL Server DROP DATABASE statement to delete a database

To remove an existing database from a SQL Server instance, you use the `DROP DATABASE` statement.

The `DROP DATABASE` statement allows you to delete one or more databases with the following syntax:

```
DROP DATABASE  [ IF EXISTS ]
    database_name
    [,database_name2,...];
```

Before deleting a database, you must ensure the following important points:

- First, the `DROP DATABASE` statement deletes the database and also the physical disk files used by the database. Therefore, you should have a backup of the database in case you want to restore it in the future.
- Second, you cannot drop the database that is currently being used.

The following example uses the `DROP DATABASE` statement to delete the `TestDb` database:

```
DROP DATABASE IF EXISTS TestDb;
```

SQL Server CREATE SCHEMA

Summary: in this tutorial, you will learn how to use the SQL Server `CREATE SCHEMA` to create a new schema in the current database.

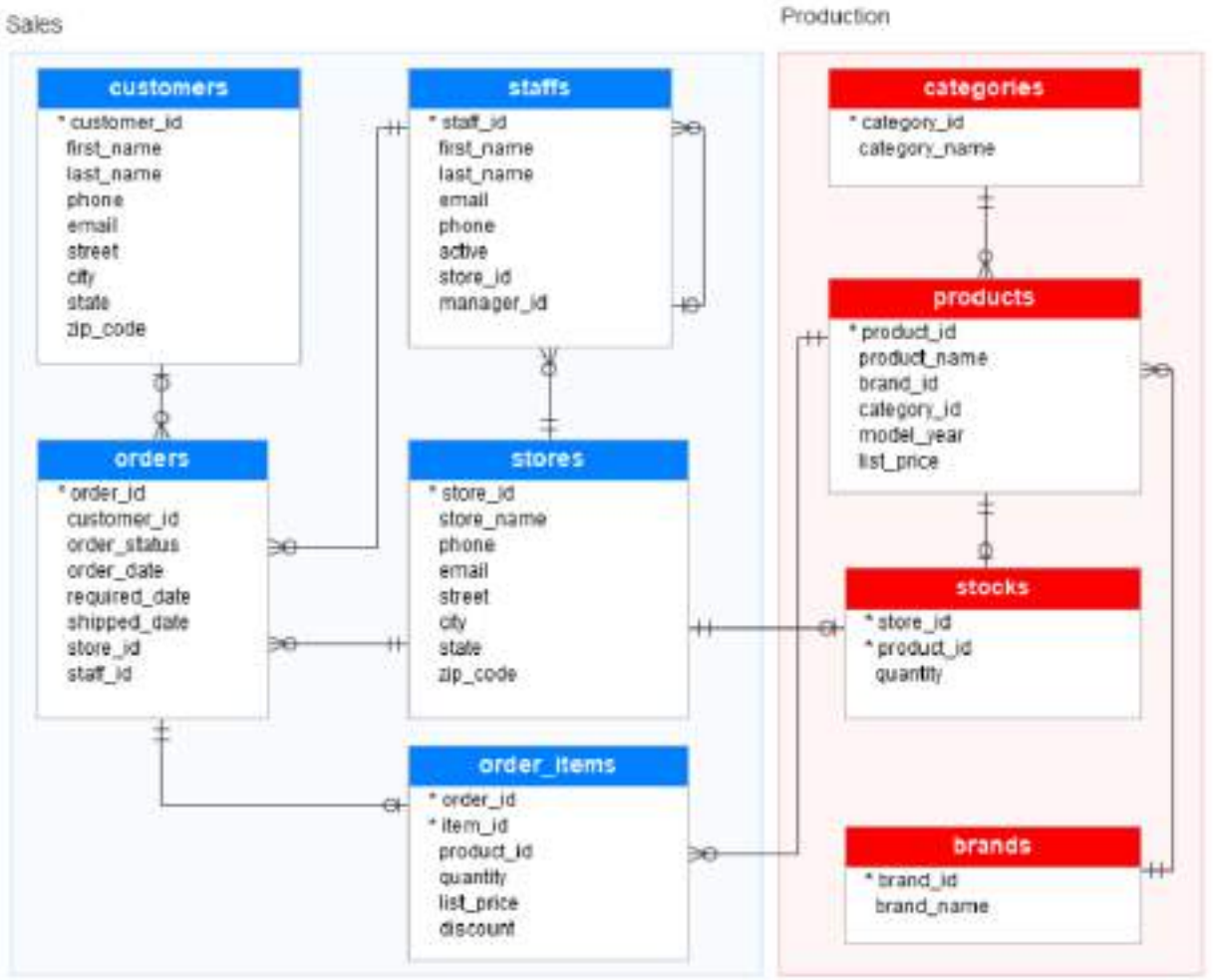
What is a schema in SQL Server

A schema is a collection of database objects including tables, views, triggers, stored procedures, indexes, etc. A schema is associated with a username which is known as the schema owner, who is the owner of the logically related database objects.

A schema always belongs to one database. On the other hand, a database may have one or multiple schemas. For example, in our `BikeStores` sample database, we have two schemas: `sales` and `production`. An object within a schema is qualified using the `schema_name.object_name` format like `sales.orders`. Two tables in two schemas can share the same name so you may have `hr.employees` and `sales.employees`.

Summary: in this tutorial, you'll learn about the SQL Server sample database called BikeStores.

The following illustrates the BikeStores database diagram:



SQL Server CREATE SCHEMA statement example

The following example shows how to use the `CREATE SCHEMA` statement to create the `customer_services` schema:

```
CREATE SCHEMA customer_services;
GO
```

Note that `GO` command instructs the SQL Server Management Studio to send the SQL statements up to the `GO` statement to the server to be executed.

SQL Server DROP SCHEMA statement overview

The `DROP SCHEMA` statement allows you to delete a schema from a database.

The following shows the syntax of the `DROP SCHEMA` statement:

```
DROP SCHEMA [IF EXISTS] schema_name;
```

In this syntax:

- First, specify the name of the schema that you want to drop. If the schema contains any objects, the statement will fail. Therefore, you must delete all objects in the schema before removing the schema.
- Second, use the `IF EXISTS` option to conditionally remove the schema only if the schema exists. Attempting to drop a nonexisting schema without the `IF EXISTS` option will result in an error.

```
DROP SCHEMA logistics;
```

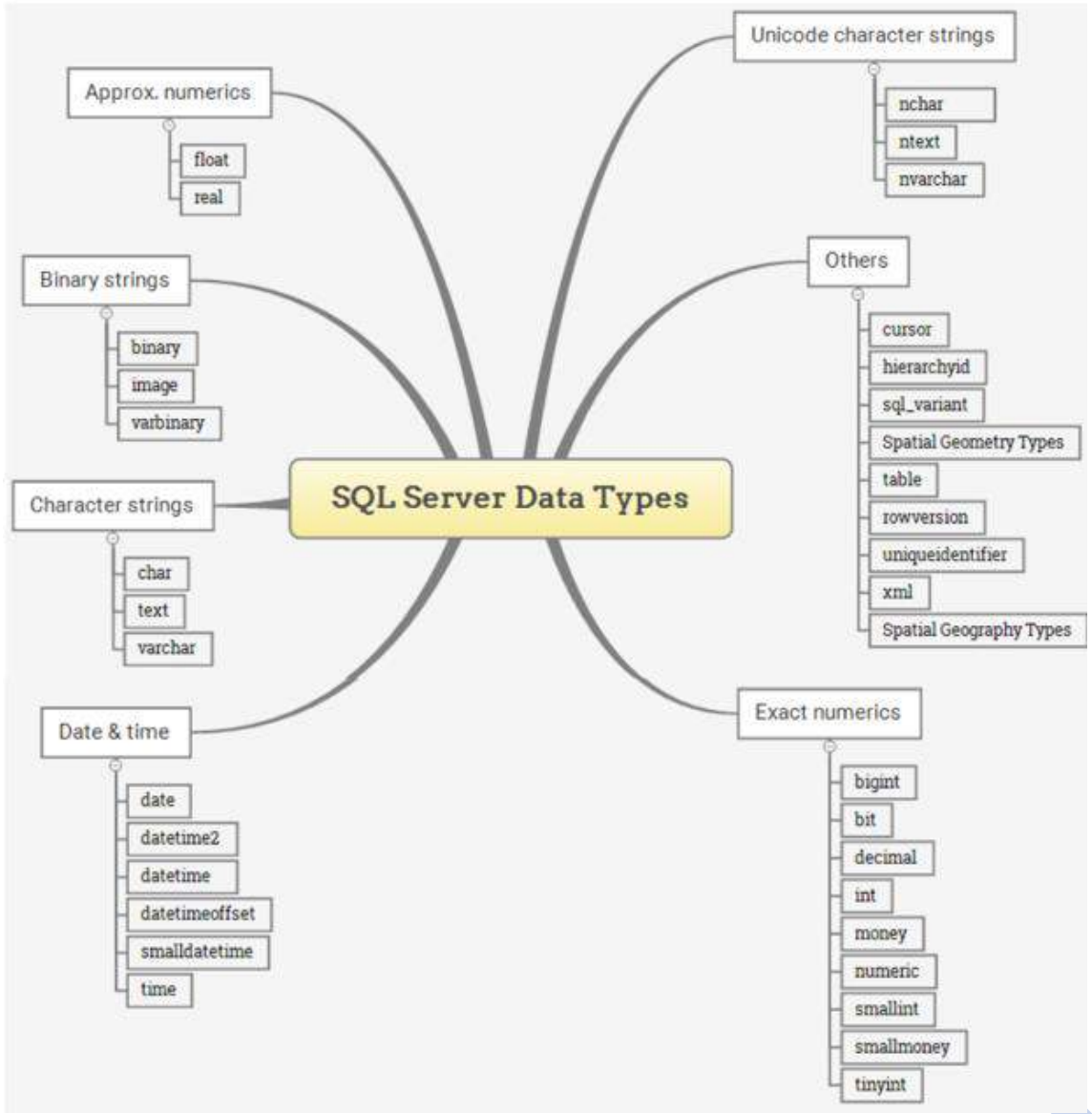
Introduction to the SQL Server CREATE TABLE statement

Tables are used to store data in the database. Tables are uniquely named within a database and schema. Each table contains one or more columns. And each column has an associated data type that defines the kind of data it can store e.g., numbers, strings, or temporal data.

To create a new table, you use the `CREATE TABLE` statement as follows:

```
CREATE TABLE [database_name.][schema_name.]table_name (
    pk_column data_type PRIMARY KEY,
    column_1 data_type NOT NULL,
    column_2 data_type,
    ...,
    table_constraints
);
```

- First, specify the name of the database in which the table is created. The `database_name` must be the name of an existing database. If you don't specify it, the `database_name` defaults to the current database.
- Second, specify the schema to which the new table belongs.
- Third, specify the name of the new table.
- Fourth, each table should have a primary key which consists of one or more columns. Typically, you list the primary key columns first and then other columns. If the primary key contains only one column, you can use the `PRIMARY KEY` keywords after the column name. If the primary key consists of two or more columns, you need to specify the `PRIMARY KEY` constraint as a table constraint. Each column has an associated data type specified after its name in the statement. A column may have one or more column constraints such as `NOT NULL` and `UNIQUE`.
- Fifth, a table may have some constraints specified in the table constraints section such as `FOREIGN KEY`, `PRIMARY KEY`, `UNIQUE` and `CHECK`.


SQL Server Data Types
Approx. numerics
float
real
Binary strings
binary
image
varbinary
Character strings
char
text
varchar
Date & time
date
datetime2
datetime
datetimeoffset
smalldatetime
time
Unicode character strings
nchar
ntext
nvarchar
Others
cursor
hierarchyid
sql_variant
Spatial Geometry Types
table
rowversion
uniqueidentifier
xml
Spatial Geography Types
Exact numerics
bigint
bit
decimal
int
money
numeric
smallint
smallmoney
tinyint

Data Types

1. Numeric Data Types

Numeric data types are used to store numbers, including integers, decimals, and floating-point numbers.

- **INT**: A whole number (integer) between -2,147,483,648 and 2,147,483,647.
- **TINYINT**: An integer between 0 and 255.
- **SMALLINT**: An integer between -32,768 and 32,767.
- **BIGINT**: An integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.
- **DECIMAL(p, s)** or **NUMERIC(p, s)**: Fixed precision and scale numbers. p specifies precision (total number of digits), and s specifies scale (number of digits after the decimal point).
- **FLOAT**: A floating-point number. It allows for approximate numeric values with a variable number of digits.

•**REAL**: A smaller range floating-point number compared to FLOAT.
•**MONEY**: A currency value with four decimal places, ranges from -922,337,203,685,477.5808 to 922,337,203,685,477.5807.
•**SMALLMONEY**: A currency value, ranges from -214,748.3648 to 214,748.3647.

2. String (Character) Data Types

String data types are used to store text data.
•**CHAR(n)**: Fixed-length string of length n (from 1 to 8,000). If the string is shorter than n, SQL Server pads it with spaces.
•**VARCHAR(n)**: Variable-length string with a maximum length of n (up to 8,000). It doesn't pad spaces.
•**TEXT**: Variable-length string data with a maximum length of 2GB of text data (this is deprecated and should be avoided).

•**NCHAR(n)**: Fixed-length Unicode string of length n (from 1 to 4,000). Used for storing international characters.
•**NVARCHAR(n)**: Variable-length Unicode string with a maximum length of n (up to 4,000).
•**NTEXT**: Variable-length Unicode string data with a maximum length of 2GB of text data (also deprecated).

1. **Numeric Data Types**

Used to store numbers (both integer and floating-point).

Data Type	Description	Storage
`INT`	Integer (whole number)	4 bytes
`BIGINT`	Large integer	8 bytes
`SMALLINT`	Small integer	2 bytes
`TINYINT`	Very small integer (0-255)	1 byte
`DECIMAL(p, s)` / `NUMERIC(p, s)`	Fixed precision and scale numbers	Varies
`FLOAT`	Approximate number (floating-point)	4 or 8 bytes
`REAL`	Smaller floating-point number	4 bytes
`BIT`	Boolean-like values (0, 1, or NULL)	1 bit per row

2. **Character/String Data Types**

Used to store text.

Data Type	Description	Max Size
`CHAR(n)`	Fixed-length string	n bytes
`VARCHAR(n)`	Variable-length string	Up to 2 GB
`TEXT`	Large text data	Up to 2 GB (deprecated)
`NCHAR(n)`	Fixed-length Unicode string	2 × n bytes
`NVARCHAR(n)`	Variable-length Unicode string	Up to 2 GB
`NTEXT`	Large Unicode text data	Up to 2 GB (deprecated)

3. **Date and Time Data Types**

Used to store date and time values.

Data Type	Description	Storage
`DATE`	Stores only date (YYYY-MM-DD)	3 bytes
`TIME`	Stores only time (HH:MM)	3-5 bytes
`DATETIME`	Date and time (YYYY-MM-DD HH:MM)	8 bytes
`SMALLDATETIME`	Date and time (accurate to minute)	4 bytes
`DATETIME2`	Extended precision date and time	6-8 bytes
`DATETIMEOFFSET`	Date, time, and time zone offset	8-10 bytes

4. **Binary Data Types**

Used to store binary data like files, images, etc.

Data Type	Description	Max Size
`BINARY(n)`	Fixed-length binary data	n bytes
`VARBINARY(n)`	Variable-length binary data	Up to 2 GB
`IMAGE`	Large binary data (deprecated)	Up to 2 GB

5. **Money Data Types**

Used to store currency or monetary values.

Data Type	Description	Storage
`MONEY`	Large monetary values	8 bytes
`SMALLMONEY`	Smaller monetary values	4 bytes

6. **Unique Identifier Data Types**

Used to store globally unique identifiers (GUIDs).

Data Type	Description	Storage
`UNIQUEIDENTIFIER`	Stores a GUID	16 bytes

7. **XML and JSON Data Types**

Used to store semi-structured data.

Data Type	Description	Max Size
`XML`	Stores XML-formatted data	Up to 2 GB
`JSON` (via `NVARCHAR`)	JSON-formatted data	Up to 2 GB

8. **Spatial Data Types**

Used to store spatial/geographical data.

Data Type	Description
`GEOMETRY`	Stores planar geometric data (2D)
`GEOGRAPHY`	Stores geospatial data (latitude/longitude)

9. **Other Data Types**

Various other special-purpose data types.

Data Type	Description	Storage
`SQL_VARIANT`	Stores values of various data types	Varies
`TIMESTAMP` / `ROWVERSION`	Auto-generated unique binary numbers	8 bytes
`CURSOR`	Stores a reference to a database cursor	N/A
`TABLE`	Stores a result set for temporary use	N/A

Deprecated Data Types

- `TEXT`, `NTEXT`, `IMAGE`: These are deprecated and should be replaced with `VARCHAR(MAX)`, `NVARCHAR(MAX)`, and `VARBINARY(MAX)` respectively.

Summary of Use Cases

- `INT`: Use for whole numbers like ID values.
- `VARCHAR`: Store variable-length text (non-Unicode).
- `NVARCHAR`: Store Unicode text (e.g., international characters).
- `DATETIME2`: Use for precise date and time values.
- `VARBINARY(MAX)`: Store files or images.
- `XML` / JSON: Store semi-structured data.
- `GEOGRAPHY`: Store location-related data.

Type of Constriants

PRIMARY KEY

FOREIGN KEY

CHECK Constraint

UNIQUE Constraint

NOT NULL Constraint

Introduction to SQL Server `PRIMARY KEY` constraint

A primary key is a column or a group of columns that uniquely identifies each row in a table. You create a primary key for a table by using the `PRIMARY KEY` constraint.

If the primary key consists of only one column, you can define use `PRIMARY KEY` constraint as a column constraint:

```
CREATE TABLE table_name (
    pk_column data_type PRIMARY KEY,
    ...
);
```

```sql
create table sales_schema.customers
(
customer_id int primary key,
first_name varchar(15) not null,
last_name varchar(15) not null,
email varchar(50) not null,
phone varchar(15) ,
state varchar(15) ,
city varchar(15) not null,
street varchar(30) not null,
zip_code varchar(5)
);
```

SQL Server FOREIGN KEY

Summary: In this tutorial, you will learn how to use the SQL Server foreign key constraint to enforce a link between the data in two tables.

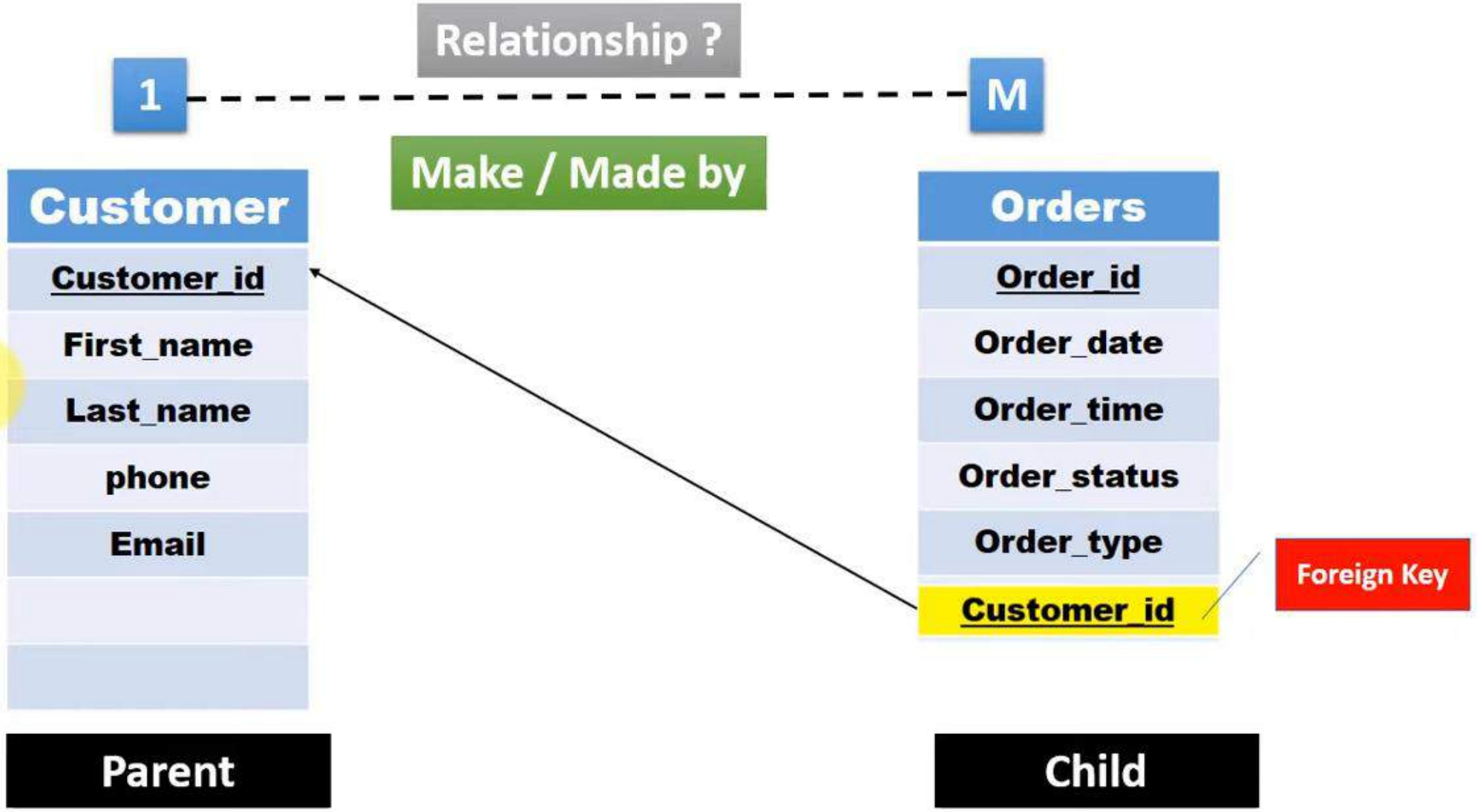
Introduction to the SQL Server foreign key constraint

Consider the following `vendor_groups` and `vendors` tables:

```
CREATE TABLE procurement.vendor_groups (
    group_id INT IDENTITY PRIMARY KEY,
    group_name VARCHAR (100) NOT NULL
);

CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
);
```

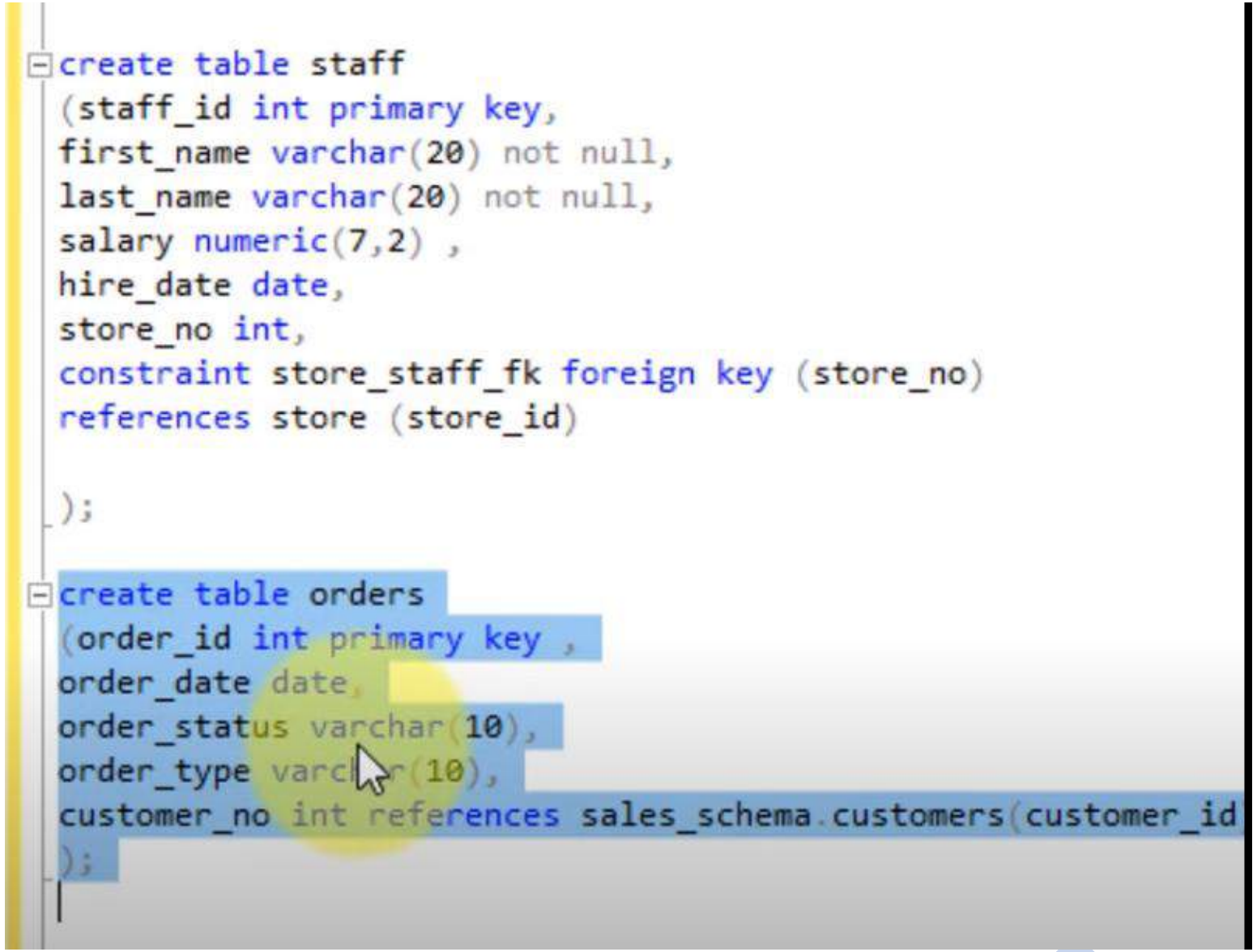
```
CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
        CONSTRAINT fk_group FOREIGN KEY (group_id)
        REFERENCES procurement.vendor_groups(group_id)
);
```


Relationship ?
1
M
Make / Made by
Customer
Customer_id
First_name
Last_name
phone
Email
Orders
Order_id
Order_date
Order_time
Order_status
Order_type
Customer_id
Foreign Key
Parent
Child

```
create table store
(store_id int primary key ,
store_name varchar(30),
city varchar(20) not null,
phone varchar(10)
)

create table staff
(staff_id int primary key,
first_name varchar(20) not null,
last_name varchar(20) not null,
salary numeric(7,2) ,
hire_date date,
store_no int,
constraint store_staff_fk foreign key (store_no)
references store (store_id)

);
```


create table staff
(staff_id int primary key,
first_name varchar(20) not null,
last_name varchar(20) not null,
salary numeric(7,2) ,
hire_date date,
store_no int,
constraint store_staff_fk foreign key (store_no)
references store (store_id)
);
create table orders
(order_id int primary key ,
order_date date,
order_status varchar(10),
order_type varchar(10),
customer_no int references sales_schema.customers(customer_id
);

SQL Server UNIQUE Constraint

Summary: in this tutorial, you will learn how to use the SQL Server `UNIQUE` constraint to ensure the uniqueness of data contained in a column or a group of columns.

Introduction to SQL Server UNIQUE constraint

SQL Server `UNIQUE` constraints allow you to ensure that the data stored in a column, or a group of columns, is unique among the rows in a table.

The following statement creates a table whose data in the `email` column is unique among the rows in the `hr.persons` table:

```
CREATE SCHEMA hr;
GO

CREATE TABLE hr.persons(
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE
);
```

Introduction to SQL Server CHECK constraint

The `CHECK` constraint allows you to specify the values in a column that must satisfy a Boolean expression.

For example, to require positive unit prices, you can use:

```
CREATE SCHEMA test;
GO

CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2) CHECK(unit_price > 0)
);
```

The End