

Databases

Relational and Non-Relational Databases and MySQL



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://about.softuni.bg>

Have a Question?



sli.do

#qa-fund

1. **Databases** – Introduction
2. **Relational** Databases: Tables and Relationships
3. **Non-Relational** (NoSQL) Databases
4. Database Management Systems (**DBMS**)
5. **SQL** Commands: SELECT, INSERT, UPDATE, DELETE
6. **JSON** Data Format
7. Working with **MySQL** + **Workbench**
8. Working with **MongoDB** + **Compass**





Databases

Introduction

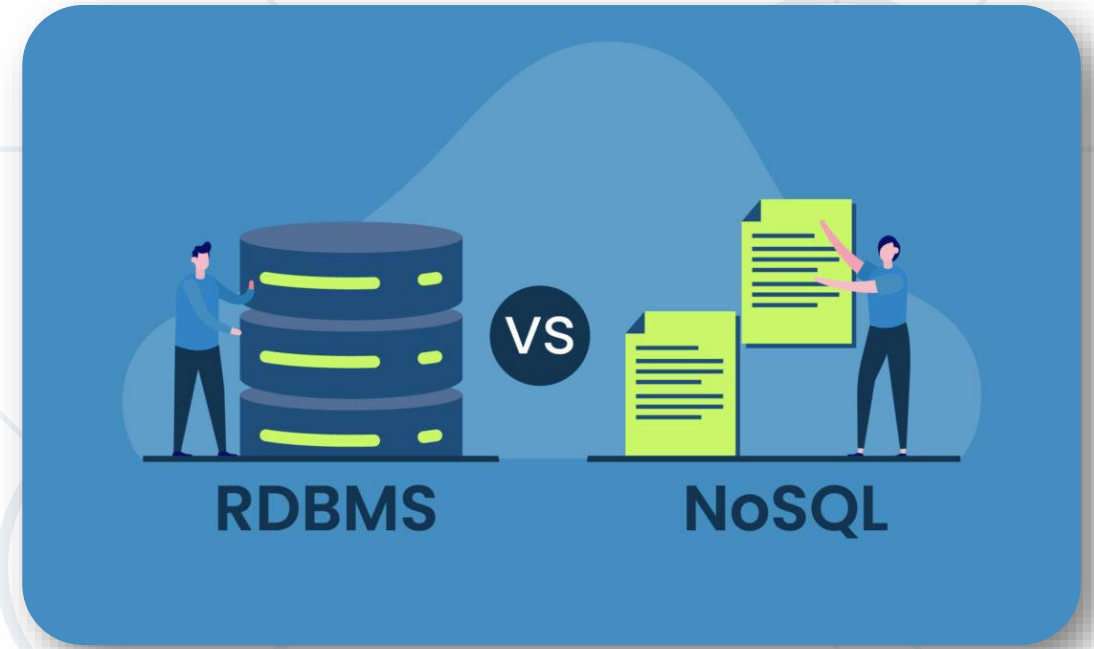
What is a Database?

- A **database** is a collection of data, organized to be easily accessed, managed and updated
- Modern databases are managed by **Database Management Systems** (DBMS)
 - Define database **structure**, e.g. tables, collections, columns, relations, indexes
 - Create / Read / Update / Delete data (CRUD operations)
 - Execute **queries** (filter / search data)

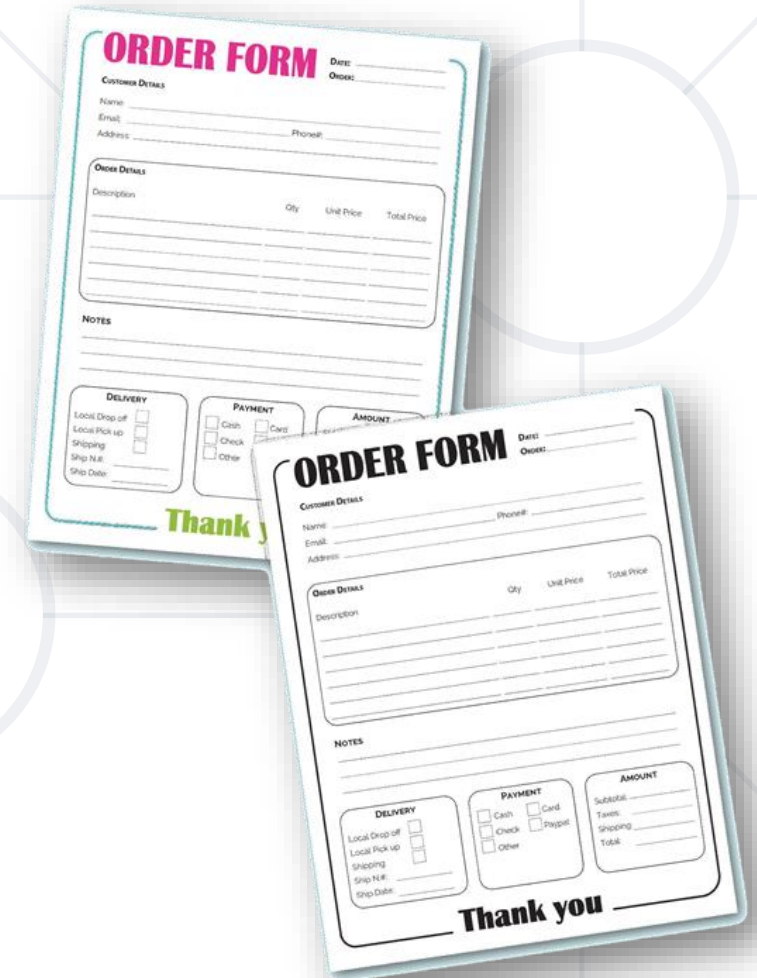


- Databases hold and manage data in the back-end systems
- **Relational databases** (SQL Databases)
 - Hold data in **tables** + **relationships**
 - Use the **SQL** language to query / modify data
 - Examples: MySQL, PostgreSQL, Web SQL in HTML5
- **Non-Relational databases** (No SQL Databases)
 - Hold **collections** of documents or key-value pairs
 - Examples: MongoDB, IndexedDB in HTML5

- RDBMS vs. NoSQL – Which is Better?
 - RDBMS focuses on relational databases
 - NoSQL focuses on Big Data and real-time web applications



- Conventional data storage
 - Orders
 - Receipts



From Data Storage to Databases

- We can **group related pieces of data** into separate columns:



ORDER FORM Date: _____
Order: _____

CUSTOMER DETAILS
Name: _____
Email: _____ Phone: _____
Address: _____

ORDER DETAILS
City: _____ Unit Price: _____ Total Price: _____
Description: _____

NOTES

DELIVERY
Local Drop off ☐
Local Pick up ☐
Shipping ☐
Ship N/A ☐
Ship Date: _____

PAYMENT
☐ Cash ☐ Card
☐ Check ☐ Paypal
☐ Other _____

AMOUNT
Subtotal: _____
Taxes: _____
Shipping: _____
Total: _____

Thank you _____



| ITEM NO. | DESCRIPTION | QUANTITY | UNIT PRICE | TOTAL |
|----------|------------------|----------|------------|----------|
| WM2536 | Women's Tall - M | 10 | \$10.00 | \$100.00 |
| GH2403 | Men's Tall - M | 5 | \$20.00 | \$100.00 |
| NC2267 | Children's - S | 10 | \$5.00 | \$50.00 |
| YX6278 | Men's - XL | 5 | \$10.00 | \$50.00 |

- Thus, we keep data in **tables** (like in Excel)
- Tables may be **related** (e. g. Products and Orders)

Why Do We Need Databases?

- **Data storage and processing** is a common need in the tech industry
- Data storage needs:
 - Ease of searching
 - Ease of updating
 - Performance
 - Accuracy and consistency
 - Security and access control
 - Redundancy





Relational Databases

Tables, Relationships and SQL

SQL Databases (Relational Databases)

- Relational (**SQL**) databases organize data in **tables**
 - Tables have strict structure (**columns** of certain **data types**)
 - Can have **relationships** to other tables
- Relational databases use the **structured query language (SQL)** for defining and manipulating data
 - Extremely powerful for complex queries
- **Relational databases** are the most widely used data management technology



The Relational DB Model

- Relational DB model organizes data into one or more **tables** of columns and rows with a **unique key** identifying each row and **foreign keys** defining **relationships**

Items

| ID | Order ID | Name | Quantity | Price |
|----|----------|-------|----------|--------|
| 5 | 1 | Table | 1 | 200.00 |
| 6 | 1 | Chair | 1 | 123.12 |

Customers

| ID | Name | Email |
|----|-------|-----------------|
| 5 | Peter | peter@gmail.com |
| 6 | Jayne | jayne@gmail.com |

Orders

| ID | Customer ID | Date | Total Price |
|----|-------------|----------|-------------|
| 1 | 5 | 11/1/17 | 323.12 |
| 2 | 1 | 11/15/17 | 13.99 |





Non-Relational Databases

NoSQL Databases and JSON Documents

NoSQL (Non-Relational) Databases

- A **NoSQL** databases have dynamic schema for **unstructured** data
- Data may be stored in several ways:
 - **Document-oriented** (JSON store)
 - **Column-oriented** (table store)
 - **Graph-based**
 - **Key-value store**



- **NoSQL databases** don't use tables
 - Instead, use **document collections** or **key-value pairs**
- More **scalable** and **high performance**
- Examples: **MongoDB**, **Cassandra**, **Redis**, etc.

Example of
JSON document
in MongoDB

```
{  
  "_id": ObjectId("59d3fe7ed81452db0933a871"),  
  "email": "peter@gmail.com",  
  "age": 22  
}
```

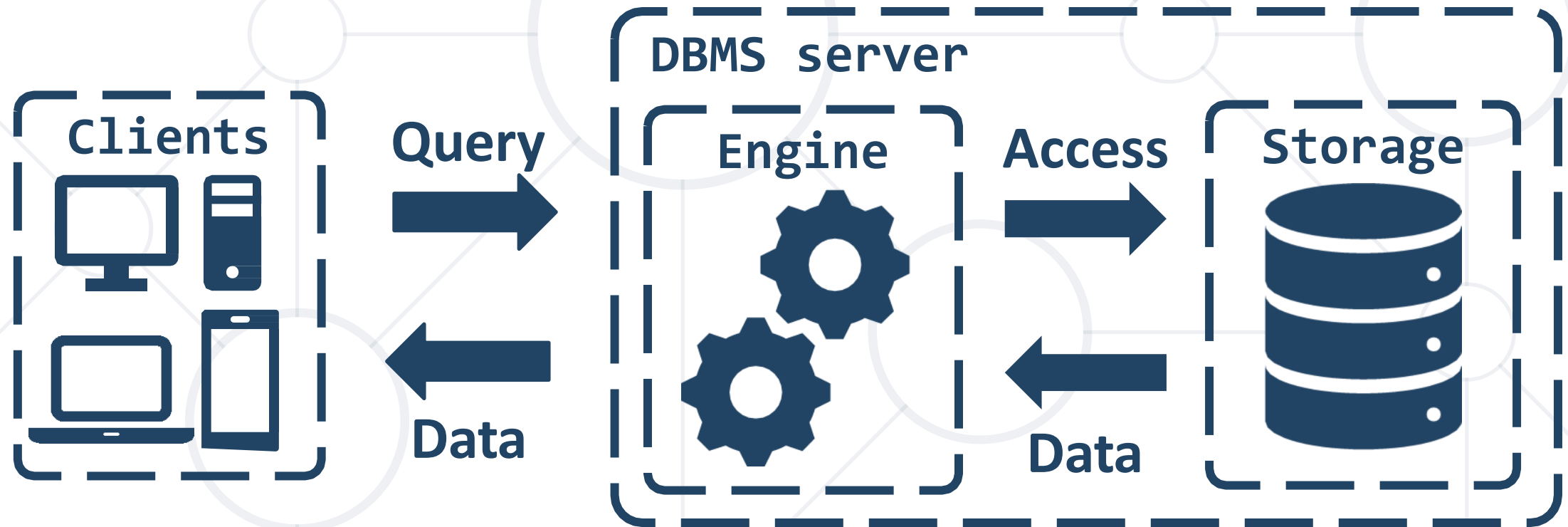


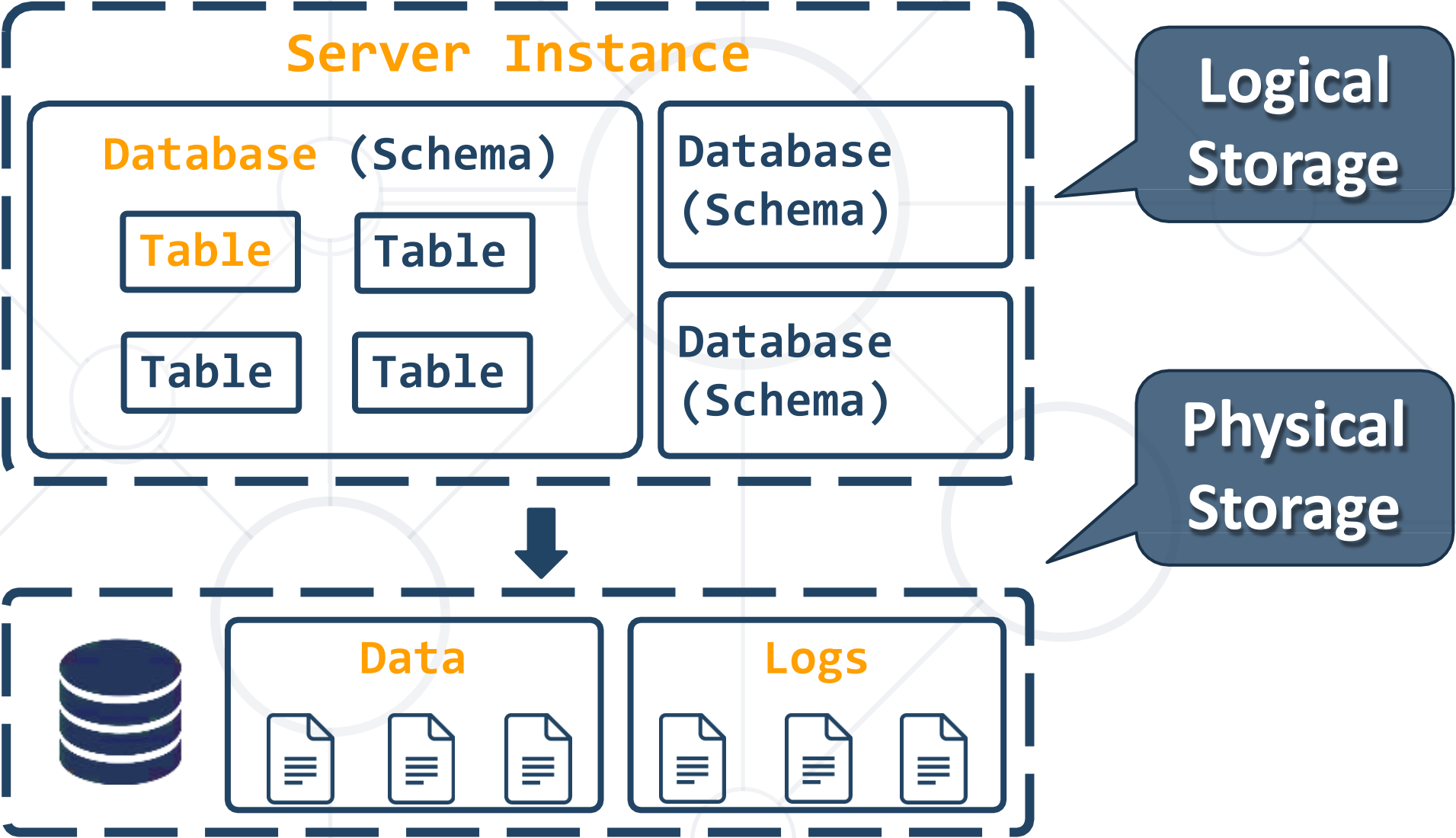

Database Management Systems

(DBMS)

- A **D**ata**b**ase **M**anagement **S**ystem (**DBMS**) is a software, used to **define, manipulate, retrieve** and **manage** data in a database
 - DBMS **stores and manages** the data itself, the data format, field names and data types, record structure and file structure
- DBMS **examples** (database servers):
 - MySQL, MS SQL Server, Oracle, PostgreSQL
 - MongoDB, Cassandra, Redis, HBase
 - Amazon DynamoDB, Azure Cosmos DB

- DBMS servers use the **client-server model**:







Structured Query Language

SQL Language

Structured Query Language

- Query language designed for managing data in a relational database
- Developed at **IBM** in the early 1970s
- To communicate with the DB engine we use **SQL**

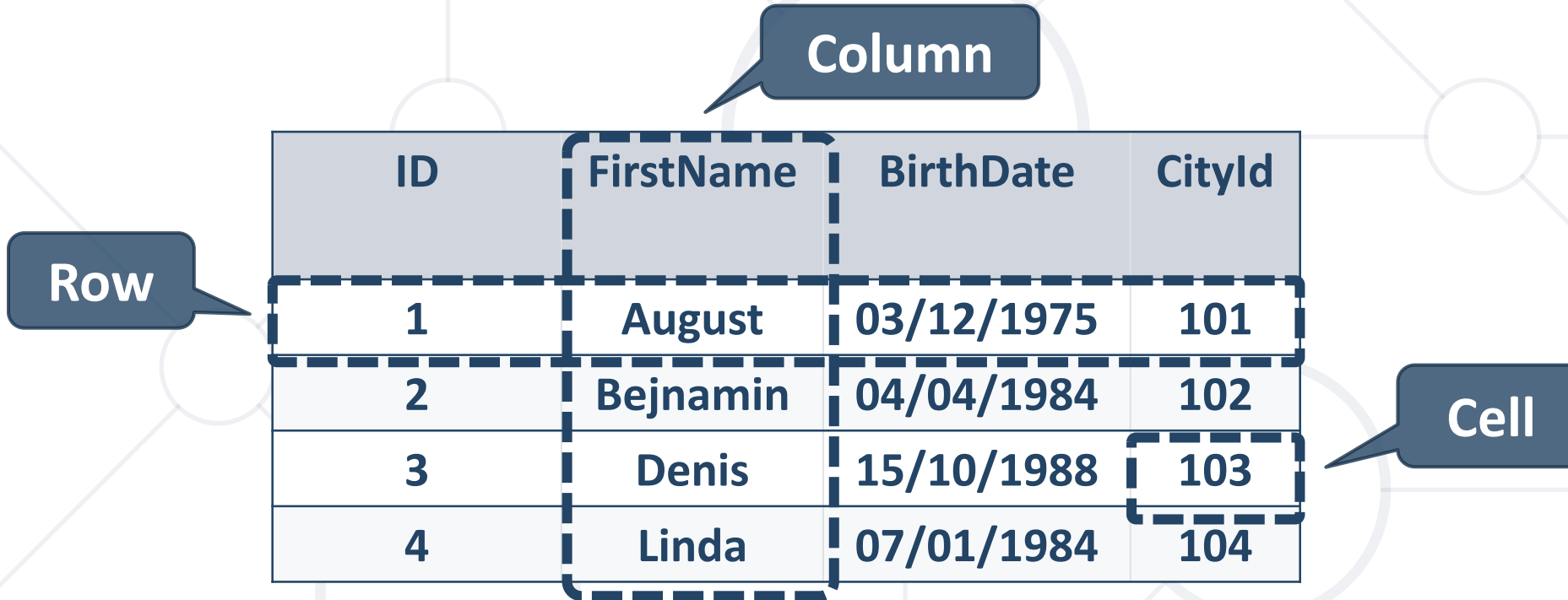


Structured Query Language

- **SQL** == query language designed for managing data in **relational** databases (RDBMS)
 - Used to communicate with the database engine
- Logically, SQL is divided into four sections:
 - **Data definition**: describe the **structure** of data
 - **Data manipulation**: **store** and **retrieve data**
 - **Data control**: define who can **access the data**
 - **Transaction control**: bundle **operations** together and perform **commit** / **rollback**



- The **table** is the main **building block** in the relational databases



| ID | FirstName | BirthDate | CityId |
|----|-----------|------------|--------|
| 1 | August | 03/12/1975 | 101 |
| 2 | Bejnamin | 04/04/1984 | 102 |
| 3 | Denis | 15/10/1988 | 103 |
| 4 | Linda | 07/01/1984 | 104 |

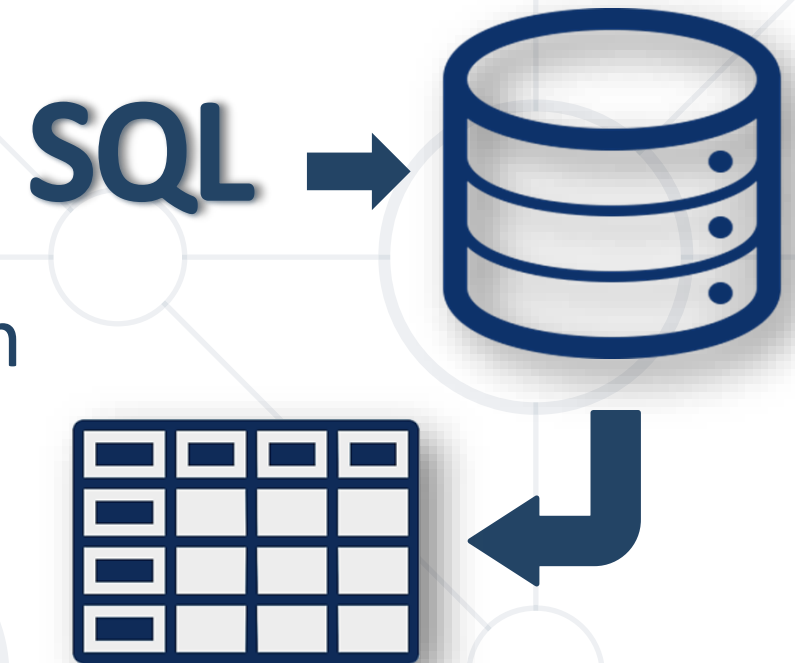
- Each **row** is called a **record** or **entity**
- Columns (**fields**) define the **type** of data they contain

- Example of SQL query:

```
SELECT * FROM people
```

- The query is executed by the DBMS system
 - It returns a sequence of data rows, e.g.

| id | email | first_name | last_name |
|----|--------------------|------------|-----------|
| 1 | smith@yahoo.co.uk | John | Smith |
| 2 | pwh@gmail.com | Peter | White |
| 3 | anne@anne.com | Anne | Green |
| 4 | jason.jj@gmail.com | Jason | Anderson |



- Subdivided into several language elements

- Queries
- Clauses
- Expressions
- Predicates
- Statements





MySQL

Working with Relational Database

- **Open-source** relational database management system
- Used in many **large-scale websites** including Google, Facebook, YouTube etc.
- Works on **many** system platforms – macOS, Windows, Linux
- Download **MySQL Community Server**
 - **Windows:** <https://dev.mysql.com/downloads/mysql/>
 - **Ubuntu/Debian:** <https://dev.mysql.com/downloads/repo/apt/>



- We split the data and introduce **relationships** between the tables to **avoid** repeating information

| user_id | first | last | registered |
|---------|---------|---------|------------|
| 203 | David | Rivers | 05/02/2016 |
| 204 | Sarah | Thorne | 07/17/2016 |
| 205 | Michael | Walters | 11/23/2015 |

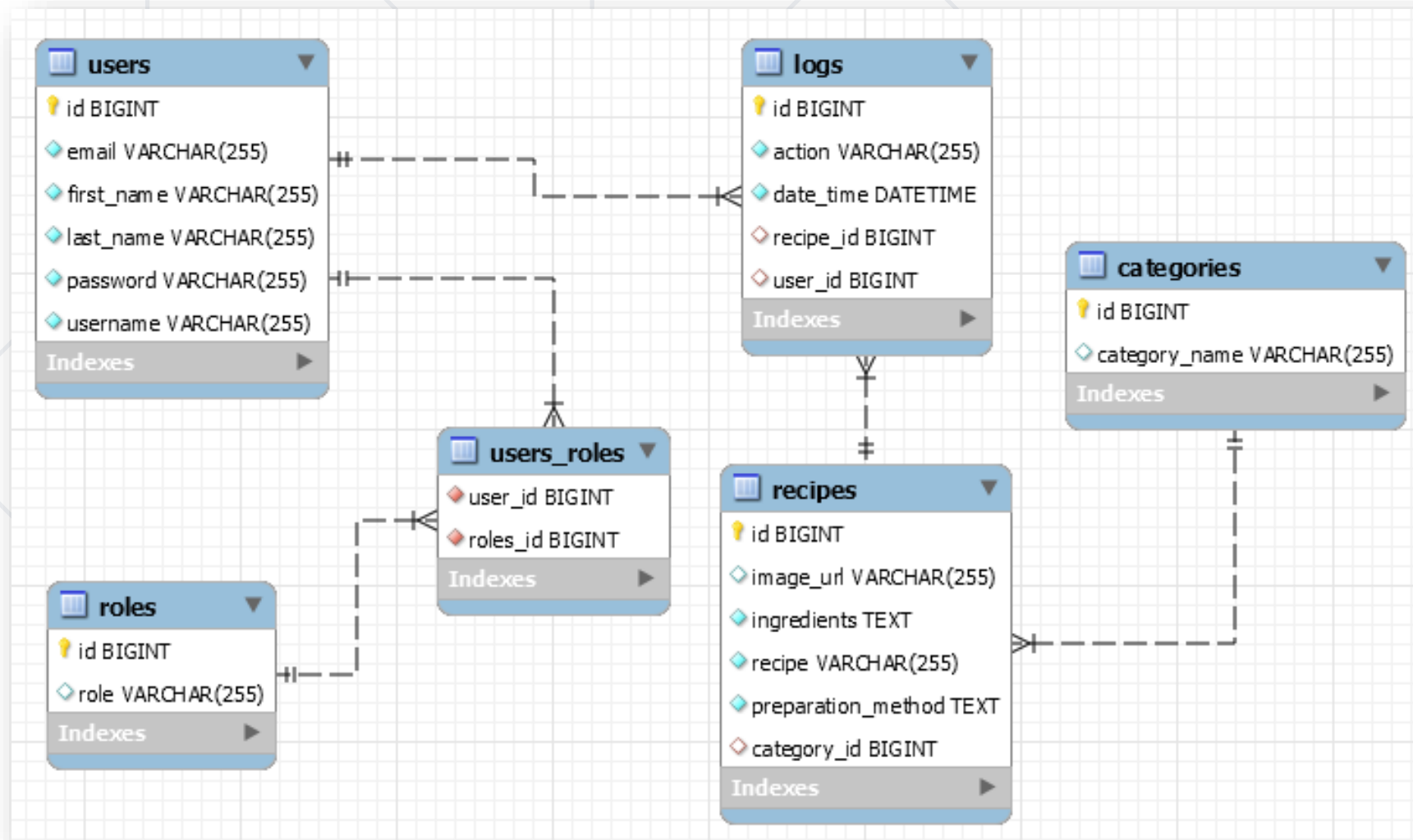
Primary Key

Foreign Key

| user_id | email |
|---------|-------------------------|
| 203 | drivers@mail.cx |
| 204 | sarah@mail.cx |
| 205 | walters_michael@mail.cx |
| 203 | david@homedomain.cx |

- Connection via **Foreign Key** in one table pointing to the **Primary Key** in another

E/R Diagrams



- We can **communicate** with the database engine via **SQL**
- SQL commands provide greater **control** and **flexibility**
- To **create** a database in MySQL:

```
CREATE DATABASE employees;
```



Database
name

- **Display** all databases in MySQL:

```
SHOW DATABASES;
```

- **Creating** tables

Table name

Column name

```
CREATE TABLE people (  
  id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
  email VARCHAR(40) NOT NULL,  
  first_name VARCHAR(40) NOT NULL,  
  last_name VARCHAR(40) NOT NULL  
);
```

Primary key
definition

Data type

- **Inserting** values

Table name

Column name

```
INSERT INTO people (email, first_name, last_name)
VALUES
```

```
('john@gmail.com', 'John', 'Smith'),
('smith@yahoo.co.uk', 'John', 'Smith'),
('peter@gmail.com', 'Peter', 'White'),
('anne@anne.com', 'Anne', 'Green'),
('jason.jj@gmail.com', 'Jason', 'Anderson');
```

Values for each
column

- **Retrieve** all records from a table

```
SELECT * FROM people;
```

* retrieves all columns

- You can **pick** (select) **the columns** to retrieve

```
SELECT first_name, last_name FROM people;
```

List of columns

- You can **limit the number of rows**

```
SELECT first_name, last_name FROM people  
LIMIT 3;
```

Number of
rows to return

- Retrieve all records, matching a **filter**

```
SELECT * FROM people  
WHERE email = 'peter@gmail.com';
```

Filter the returned rows by a condition

- Filter** and **sort** data

```
SELECT * FROM people  
WHERE id > 2 AND id < 5  
ORDER BY id;
```

Filter by multiple conditions

Sort by given column / expression

Filtering the Selected Rows

- Use **DISTINCT** to eliminate **duplicate** results

```
SELECT DISTINCT last_name  
FROM people;
```

- Filter rows by specific **conditions** using the **WHERE** clause

```
SELECT first_name, email  
FROM people  
WHERE last_name = 'Smith';
```

- Other **logical operators** can be used for greater control

```
SELECT first_name, last_name  
FROM people WHERE id <= 3;
```

- Sort rows with the **ORDER BY** clause

- ASC**: ascending order, default

```
SELECT first_name, last_name  
FROM people  
ORDER BY last_name;
```

| | first_name | last_name |
|---|------------|-----------|
| ▶ | Jason | Anderson |
| | Anne | Green |
| | John | Smith |
| | John | Smith |
| | Peter | White |

- DESC**: descending order

```
SELECT first_name, last_name  
FROM people  
ORDER BY last_name DESC;
```

| | first_name | last_name |
|---|------------|-----------|
| ▶ | Peter | White |
| | John | Smith |
| | John | Smith |
| | Anne | Green |
| | Jason | Anderson |

- **Updating** rows

```
UPDATE people
SET last_name = 'Adams'
WHERE first_name = 'Anne';
```

Updates the last
name of person

```
UPDATE people
SET first_name = 'Peter',
    last_name = 'Black',
    email = 'pw@email.com'
WHERE id = 3;
```

Updates multiple
fields

Deleting Data and Objects

- **Deleting** table rows

```
DELETE FROM people WHERE id = 4;
```

- Deleting (**dropping**) database objects

- Table

Delete all records in a table

```
TRUNCATE TABLE people;
```

Delete the table itself

```
DROP TABLE people;
```

- Entire database

```
DROP DATABASE employees;
```

- These actions **cannot be undone**



JSON Data Format

Definition and Syntax

- **JSON** (JavaScript Object Notation) is a lightweight data format
 - Human and machine-readable plain text
 - Based on **JavaScript** objects
 - Independent of development platforms and languages
 - JSON data consists of:
 - Key-value pairs: **{ key : value }**
 - Values (**strings**, **numbers**, etc.)
 - Arrays: **[value1, value2, ...]**

```
{  
  "firstName": "Peter",  
  "courses": ["C#", "JS", "ASP.NET"]  
  "age": 23,  
  "hasDriverLicense": true,  
  "date": "2012-04-23T18:25:43.511Z",  
  // ...  
}
```

- The **JSON** data format follows the rules of object creation in JS

- **Strings, numbers** and **Booleans**:

```
"this is a string and is valid JSON"
```

```
3.14
```

```
true
```

- **Arrays**:

```
[5, "text", true]
```

- **Objects** (key-value pairs):

```
{  
  "firstName": "Svetlin", "lastName": "Nakov",  
  "jobTitle": "Technical Trainer", "age": 30  
}
```



Mongo DB

Working with Non-Relational Database

- **MongoDB** == free **open-source** cross-platform **document-oriented database**
 - Keeps collections of **JSON** documents (with or without schema)
- Sample usages: **mobile app** backend, product **catalog**, **poll** system, **blog** system, Web content management system (**CMS**)
- Supports evolving data requirements
 - The DB structure **may change** over the time
- Supports **indexing** for increased performance

Install MongoDB

- Download from: mongodb.com/try/download/community


MongoDB Community Server

The Community version of our distributed database offers a flexible document data model along with support for ad-hoc queries, secondary indexing, and real-time aggregations to provide powerful ways to access and analyze your data.

The database is also offered as a fully-managed service with [MongoDB Atlas](#). Get access to advanced functionality such as auto-scaling, serverless instances (in preview), full-text search, and data distribution across regions and clouds. Deploy in minutes on AWS, Google Cloud, and/or Azure, with no downloads necessary.

Available Downloads

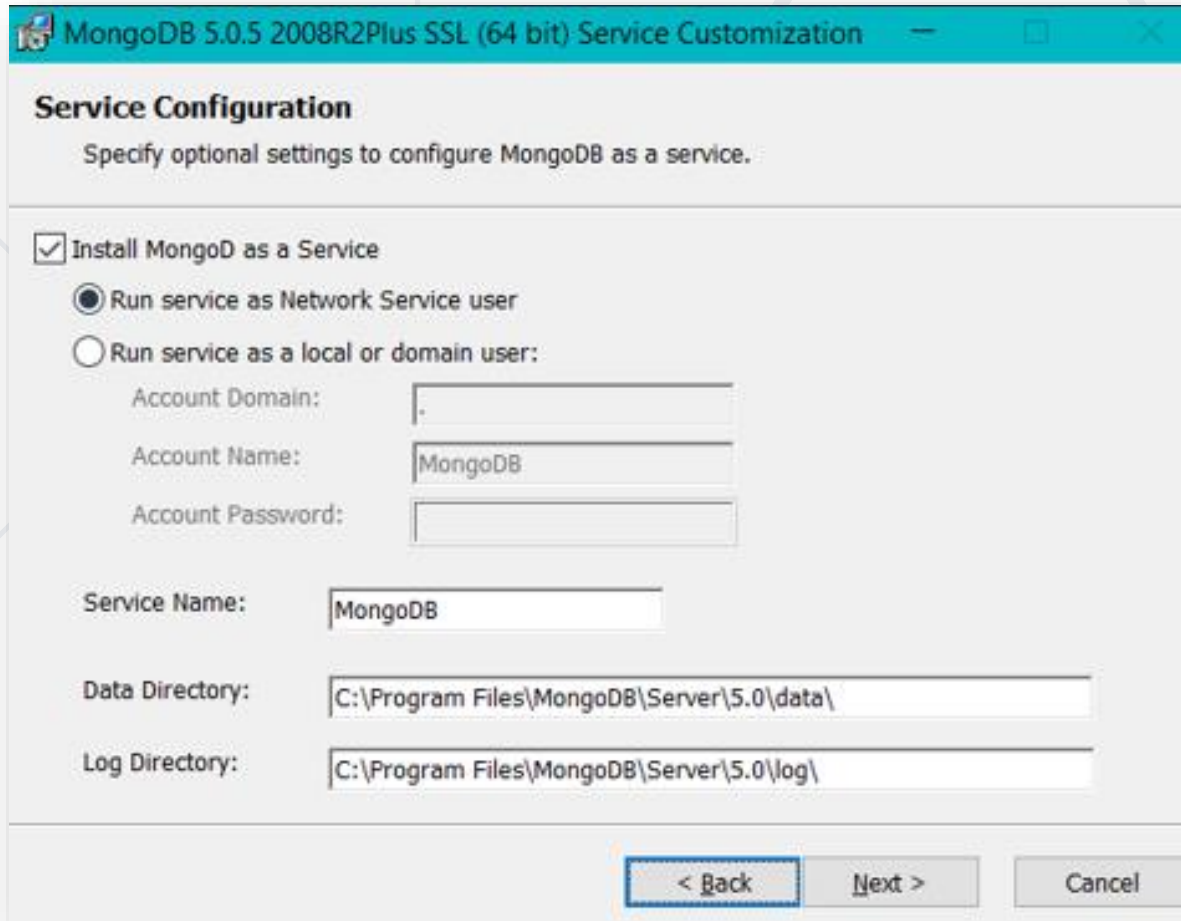
| | | |
|----------|-----------------|---|
| Version | 5.0.5 (current) | ✓ |
| Platform | Windows | ✓ |
| Package | msi | ✓ |

 **Download**

Copy Link

- The package includes **MongoDB Compass**

- During installation, configure the **MongoDB service**:



MongoDB 5.0.5 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

☒ Install MongoDB as a Service

☒ Run service as Network Service user

☐ Run service as a local or domain user:

Account Domain:

Account Name:

Account Password:

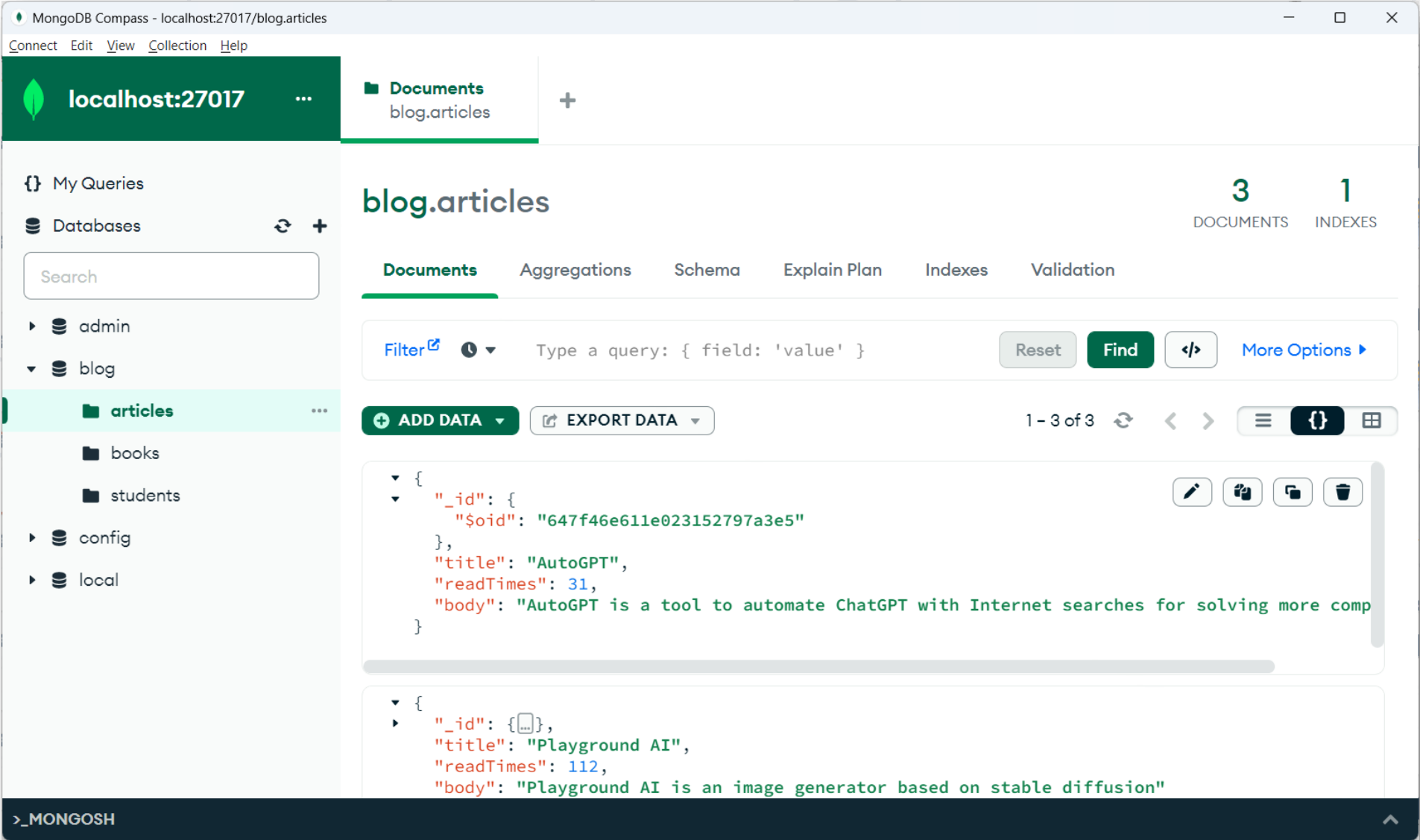
Service Name:

Data Directory:

Log Directory:

< Back Next > Cancel

- Choose one of the many
 - **MongoDB Compass** is included in the installer
- For example
 - Compass – <https://www.mongodb.com/products/compass>
 - Robo 3T – <https://robomongo.org/download>
 - NoSQLBooster – <https://nosqlbooster.com>



- Mongoose supports **many** queries

- For equality/non-equality

```
Student.findOne({'lastName': 'Petrov'})
```

```
Student.find({}).where('age').gt(7).lt(14)
```

```
Student.find({}).where('facultyNumber').equals('12399')
```

- Selection of some properties

```
Student.findOne({'lastName': 'Kirilov'}).select('name age')
```

- Sorting

```
Student.find({}).sort({age:-1})
```

- Limit & skip

```
Student.find({}).sort({age:-1}).skip(30).limit(10)
```

- Different methods could be **stacked** one upon the other

```
Student.find({})  
  .where('firstName').equals('gosho')  
  .where('age').gt(18).lt(65)  
  .sort({age:-1})  
  .skip(10)  
  .limit(10)
```

- Install "**MongoDb Shell**" and run it from the command line:

- Type the command "**mongo**"

```
show dbs
```

```
use mytestdb
```

```
db.mycollection.insertOne({"name": "George"})
```

```
db.mycollection.find({"name": " George"})
```

```
db.mycollection.find({})
```

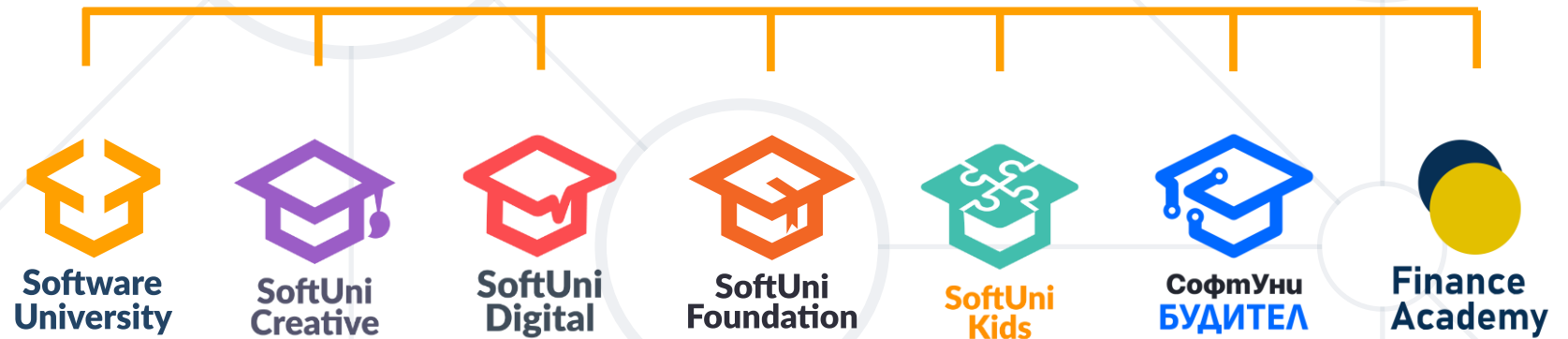
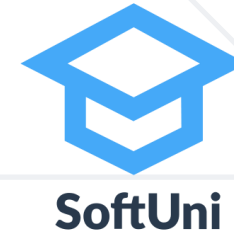
- Additional information at:

<https://www.mongodb.com/try/download/shell>

- **Databases**: store data tables and collections
- **Relational databases**: tables and relationships
- **Non-Relational**: document collections
- **DBMS** (database servers), e.g. MySQL, MongoDB
- **SQL** commands: SELECT, INSERT, UPDATE, DELETE, ...
- **JSON** document: {"name":"Joe", "age":25}
- Working with **MySQL + Workbench**
- Working with **Mongo DB + Compass**



Questions?



SoftUni Diamond Partners



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, about.softuni.bg
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity

