Database Modeling and Database Systems — Unit 1

Dr.-Ing Anna Androvitsanea

IU Internationale Hochschule GmbH

October 30, 2023

TOPIC OUTLINE

Relational Database Basics	1
Database Queries to Exactly One Table	2
Conception and Modeling of Relational Databases	3
Creating Relational Databases	4
Complex Database Queries on Multiple Tables	5

TOPIC OUTLINE

Manipulating Records in Databases	**6**
NoSQL Database System	**7**

Introductory Round

- Who are you?
- Anna
- iu Flightright GmbH
- Data Scientist and ML Engineer
- Have worked as a Civil Engineer and Archaeologist
- Data Scientist enthusiast

Overview

- Explain the most important relational database terms.
- Describe how data in relational databases is stored and read in a structured way.
- Explain the meaning and the use of SQL.
- Compare among different commercial DBMSs.

Basic Questions

- How many times have you used a database in the last week?
- What is a database, and why is it used?
- Why would we prefer to use a DBMS over a spreadsheet for data storage?

How Many Times Have You Used a Database in the Last Week?

While the answer may vary for each one of us, let's consider the following:

- Every time you accessed an online application (e.g., email, social media).
- Made an online purchase or reservation.
- Checked into a location using GPS.

For many, databases are accessed multiple times daily without even realizing it.

What is a Database and Why is it Used?

- A database is an organized collection of data, stored and accessed electronically.
- They ensure data integrity, consistency, and security.
- Databases can handle vast amounts of information and allow for quick data retrieval.
- Used in virtually every sector: from managing personal data to business operations and scientific research.

Why Use a DBMS Over a Spreadsheet for Data Storage?

- DBMSs are designed to handle large datasets, whereas spreadsheets have limits.
- Enhanced data integrity and security in DBMSs.
- Multiple users can access and modify data simultaneously.
- DBMSs support complex queries and data manipulation.
- Data in DBMSs can be related, allowing for relational database operations.

Advantages of Databases

- Storing data in databases is very advantageous.
- Databases are optimized for processing large amounts of data.
- Multiple users can access the data simultaneously.
- Databases provide consistency rules.

Components of a Database System

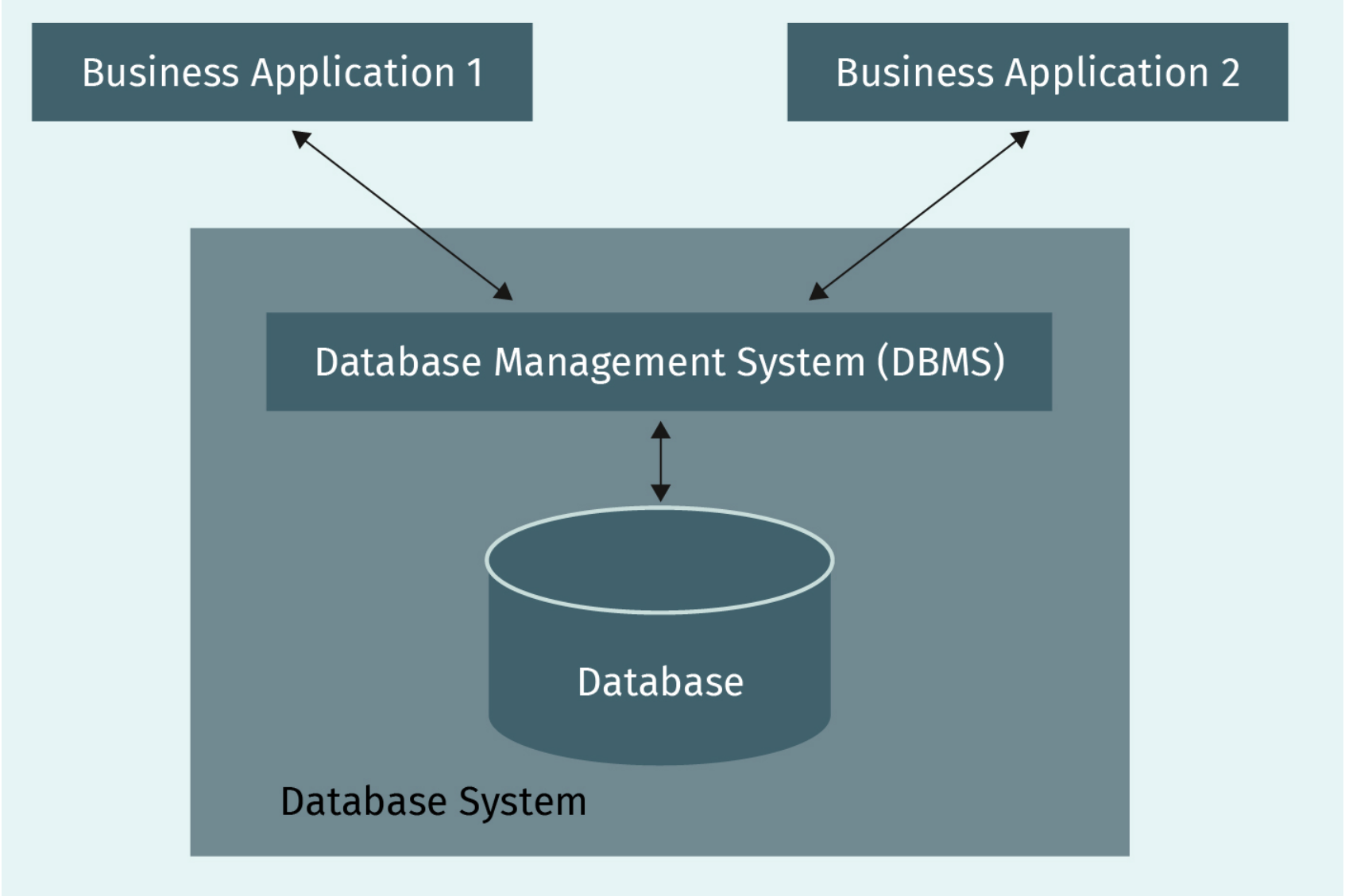


Figure: Components of a database system

Relational Databases

- Tables
  - Rows and Columns
    - Columns store a specific property of data.
    - Rows contain data about one entity.

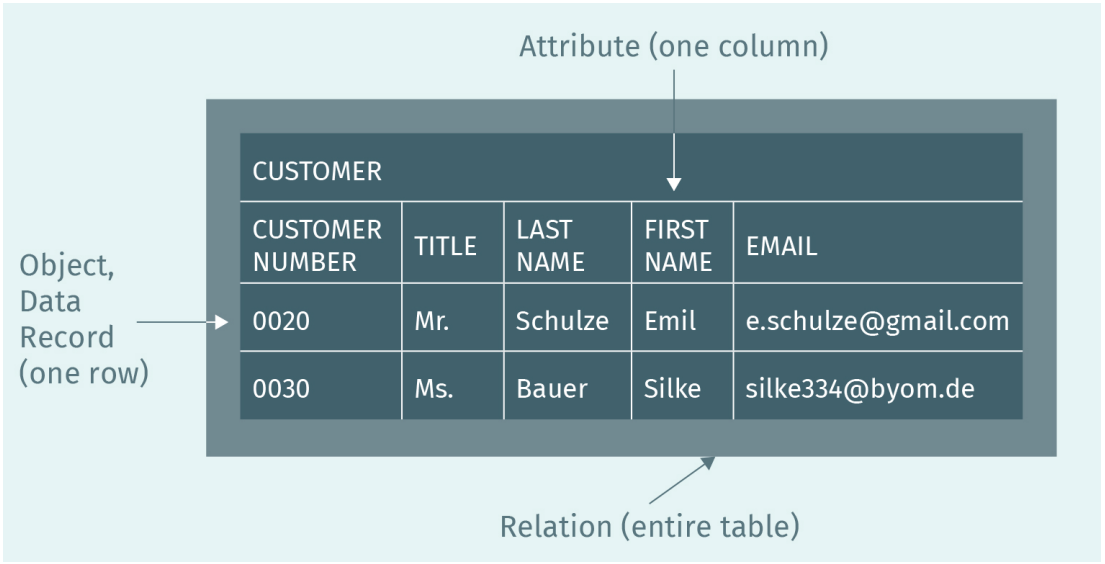


Figure: Components of a database system

Relation

- A relation, denoted as R, when applied to sets A1, A2, ..., An, represents a specific set of ordered tuples.
- The Cartesian product of these sets, representing all possible ordered combinations of elements from the sets.
- A subset of this Cartesian product, which represents the relation R.

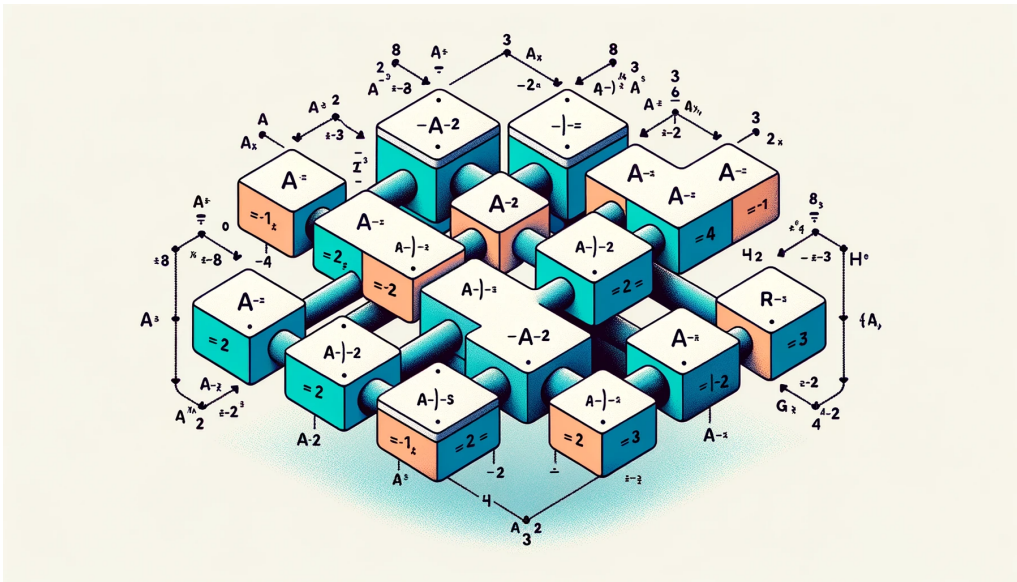


Figure: Mathematical representation of a relation

Table

Often referred to as a relation in relational database terminology.

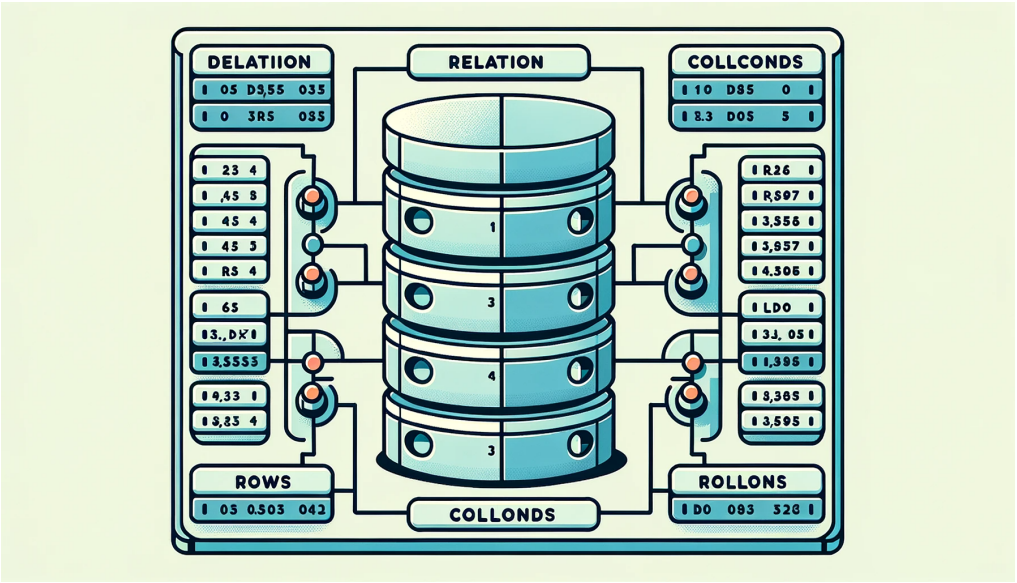


Figure: Database table illustration

Data Record / Tuple

A data record pertains to a complete set of information in a table's row.

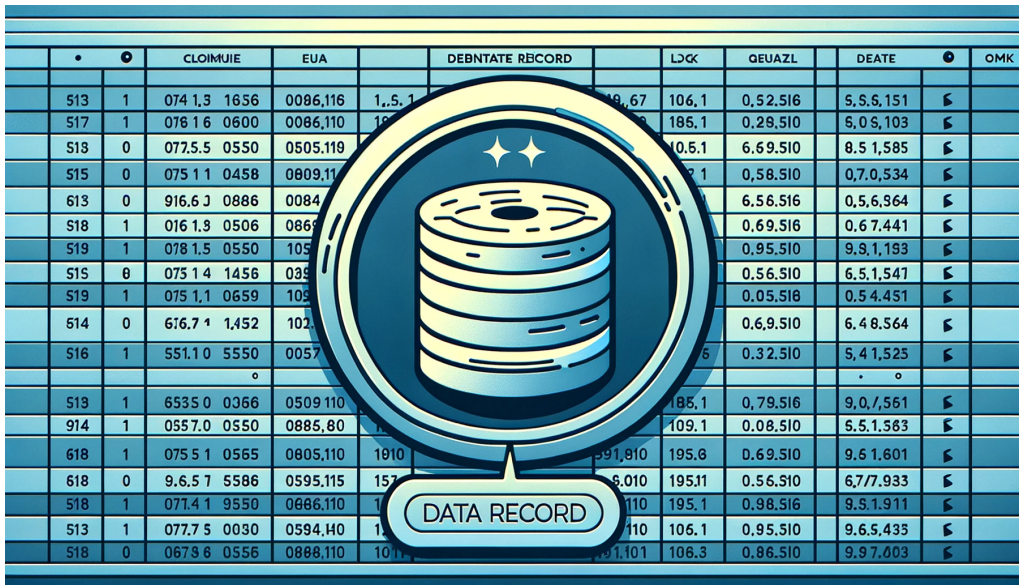


Figure: Highlighted data record in a table

Entity

An entity is an object or concept distinct in the real world.

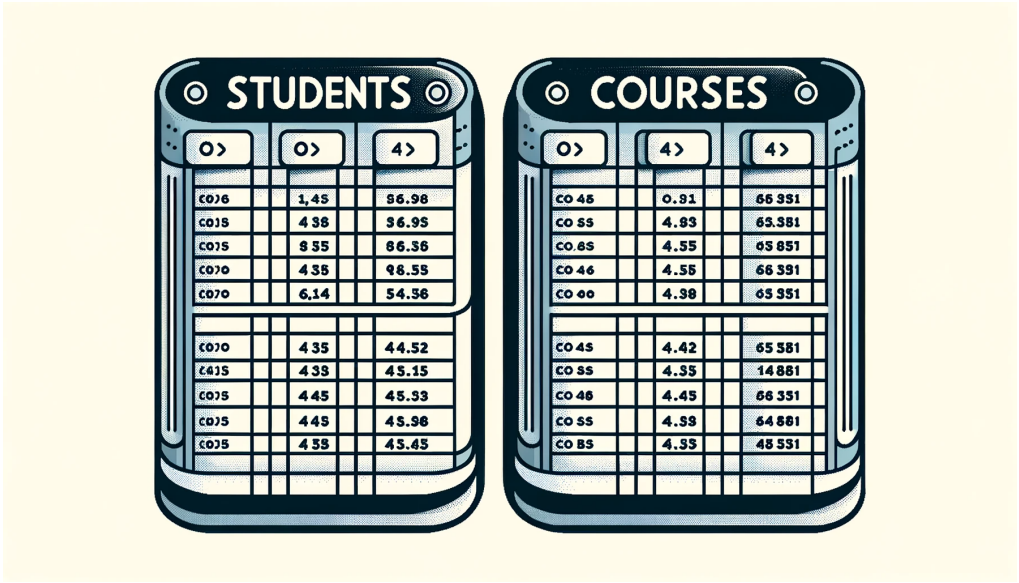


Figure: Database entities illustration

Attribute (or Property)

Attributes describe properties of an entity.

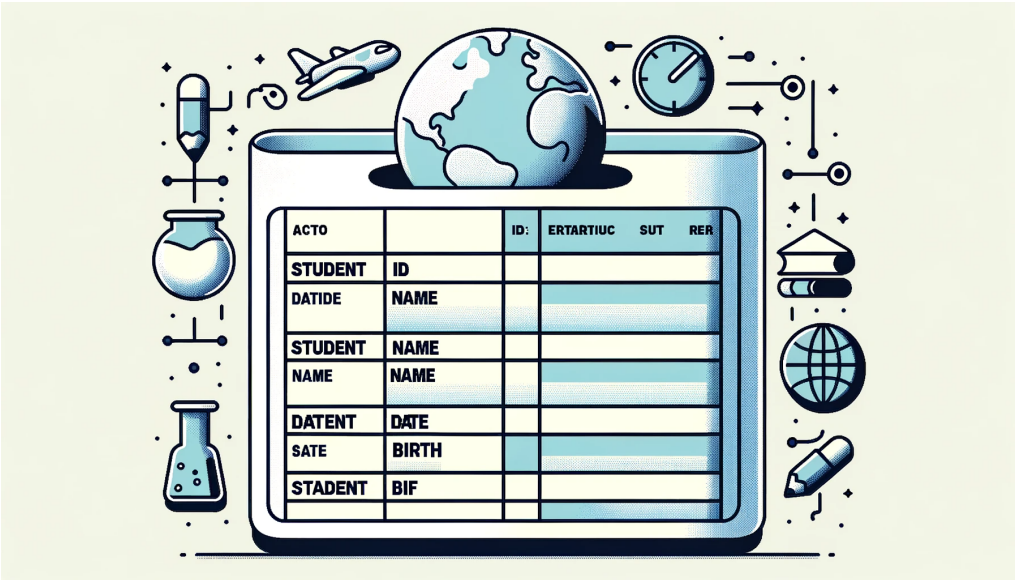


Figure: Attributes in a database table

Keys in Relational Databases

- Primary key: A unique attribute that identifies a record.
- Artificial key (Surrogate key): A meaningless attribute used as a primary key.
- Foreign key: References a primary key of another table.

Primary Key

A primary key is a unique attribute used to identify each record in a table. For example, in a university database, the table for students might have a column called "StudentID". Each student is given a unique ID number when they enroll, ensuring that no two students have the same ID. This "StudentID" would serve as the primary key for the students table.

Artificial Key (Surrogate Key)

An artificial or surrogate key is a system-generated unique identifier, often an auto-incremented integer, used to uniquely identify each record in a relational database table. Unlike natural keys, which are derived from the data being stored and may change over time, surrogate keys are stable and not tied to any business logic.

For example, consider a bookstore's database with a table for books. While attributes like title or ISBN are unique for books, they can be long, cumbersome, and might change (e.g., if a book is re-titled). Instead, the database might assign each book a unique, simple "BookID". This "BookID" ensures efficient database operations and provides flexibility in design, among other benefits. Surrogate keys are essential when there's no clear natural key, ensuring data integrity and simplifying database management.

Foreign Key

A foreign key is an attribute in a table that references the primary key of another table, establishing a link between two tables' records. In a university database, there could be a table for course enrollments. This table might have a column "StudentID" that indicates which student is enrolled in a specific course. This "StudentID" column in the enrollments table would be a foreign key referencing the "StudentID" primary key in the students table.

Relationships in Relational Databases

- 1:1 Relationships: Insert the primary key of any of the tables into the other table as a foreign key.
- 1:N Relationships: Insert the primary key of the parent table into the child table as a foreign key.
- N:M Relationships: Break the relationship into two 1:N relationships using a relationship table.

1:1 Relationship

A 1:1 relationship implies each entity from one table corresponds to only one entity in another table. For instance, in an organization, each employee is assigned a unique locker for personal storage. This establishes a 1:1 relationship between employees and lockers.

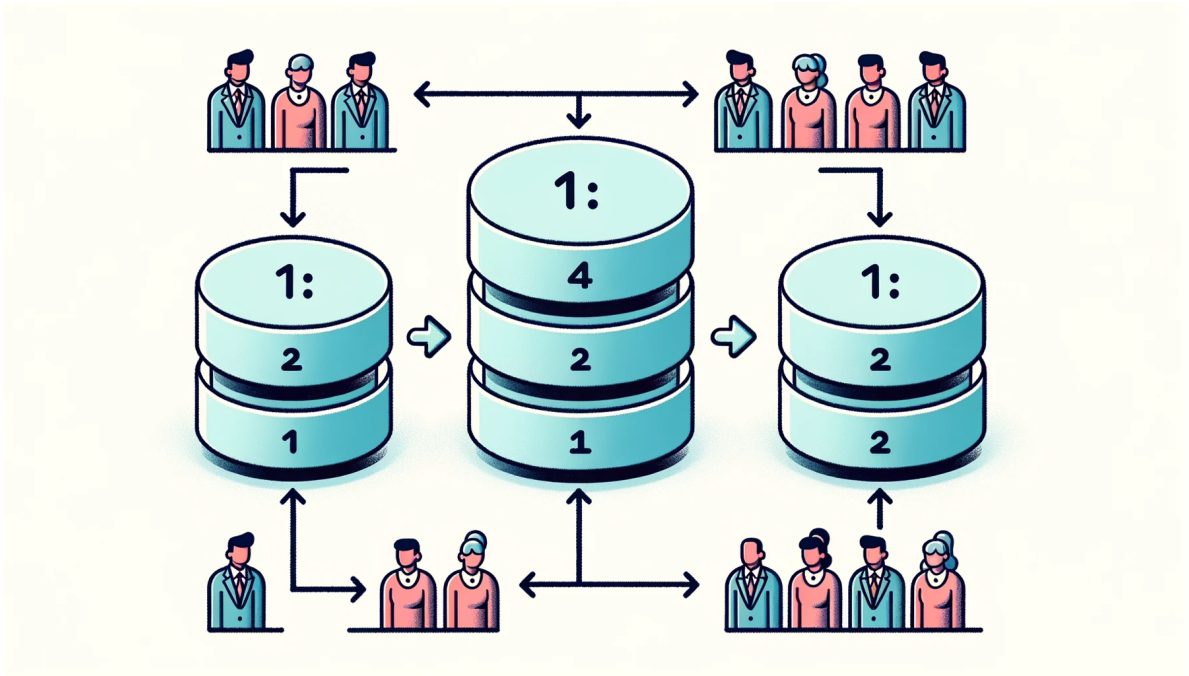


Figure: Illustration of a 1:1 Relationship

1:N Relationship

A 1:N (or "one-to-many") relationship indicates each entity from one table can be related to multiple entities in another table. For example, in a library, one author can write multiple books, but each book has only one author, leading to a 1:N relationship between authors and books.

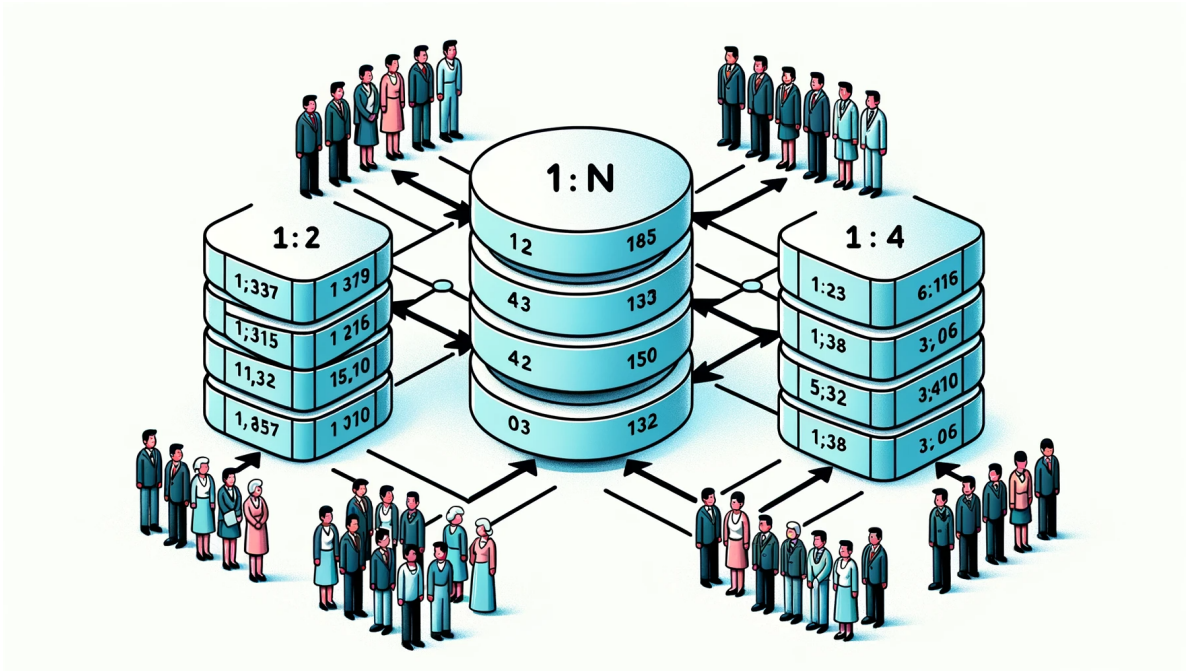


Figure: Illustration of a 1:N Relationship

N:M Relationship

An N:M (or "many-to-many") relationship denotes that multiple entities from one table can relate to multiple entities in another table. For instance, in a school, students can enroll in multiple courses, and each course can host multiple students. This forms an N:M relationship between students and courses, typically resolved using a bridge table.

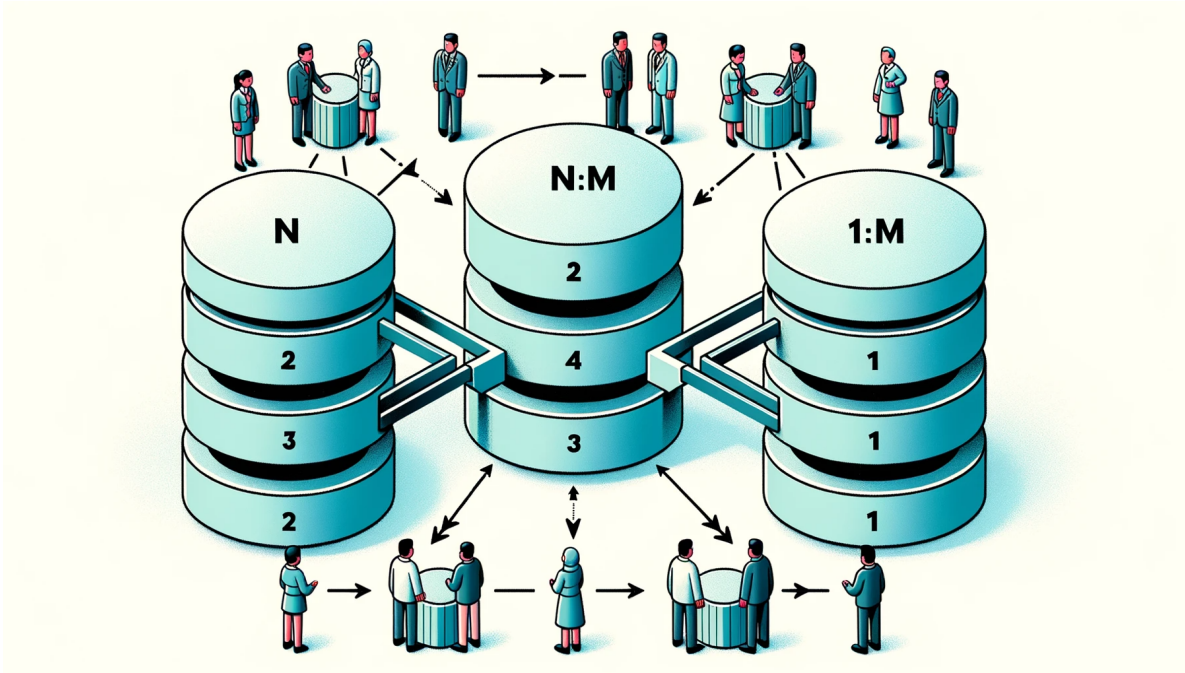


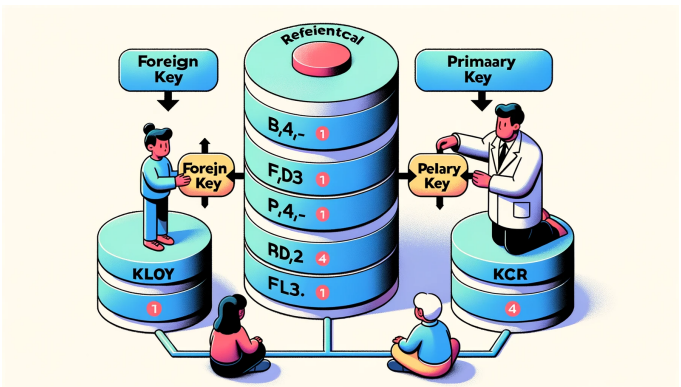
Figure: Illustration of an N:M Relationship

Referential Integrity

Referential integrity is a critical concept in relational database systems that ensures consistency and reliability of data. At its core, referential integrity mandates that:

- Any value of a foreign key in a child table (or referencing table) must have a corresponding value in the primary key of the parent table (or referenced table).
- If a primary key value changes (or is deleted) in the parent table, all corresponding foreign key values in the child table must be modified accordingly or restricted from deletion.

This mechanism ensures data consistency across tables and prevents orphan records, ensuring the database remains reliable and accurate.



Referential Integrity: Examples

- **Foreign Key Reference:**
  - Imagine a university database with two tables: *Students* and *Courses*.
  - The *Courses* table has "InstructorID" as a foreign key, referring to the "TeacherID" primary key in the *Students* table.
  - This ensures every course's instructor exists in the *Students* table.
  - Thus, for an "InstructorID" in *Courses*, there's a matching "TeacherID" in *Students*.
- **Changing or Deleting Primary Key Values:**
  - Continuing with the university example: If a teacher leaves and their record in *Students* is deleted or modified...
  - The corresponding "InstructorID" values in the *Courses* table must either be updated or restricted from deletion.
  - This ensures no courses are linked to non-existent or incorrect instructors.

Database Management System (DBMS)

A Database Management System (DBMS) is a specialized software system designed to efficiently store, retrieve, and manage data in databases. A DBMS acts as an intermediary between users or application programs and the databases. It provides a systematic and organized way to handle large volumes of data, ensuring data security, consistency, and integrity.

Figure: Illustration of a Database Management System (DBMS)

Database Management System (DBMS): Data Storage

Data Storage: Organized and structured storage of data.

- *Example*: Saving customer details in a retail database, where each customer has a unique ID, name, address, and purchase history.

Database Management System (DBMS): Data Retrieval

Data Retrieval: Efficient querying and retrieval of data.

- *Example*: Fetching the purchase history of a specific customer using their unique ID in a retail database.

Database Management System (DBMS): Data Security

Data Security: Protection against unauthorized access.

- *Example*: Restricting access to salary details in a company's HR database to only HR personnel.

Database Management System (DBMS): Data Integrity

Data Integrity: Ensuring accuracy and consistency of data.

- *Example*: Implementing validation rules to ensure email addresses entered into a contacts database follow the correct format.

Database Management System (DBMS): Concurrency Control

Concurrency Control: Managing simultaneous data access by multiple users.

- *Example*: In an airline reservation system, ensuring two agents don't book the same seat for different passengers at the same time.

Operations on Records

Finding and Deleting Records in the Database

Finding Records in Relational Databases

Finding and returning data from a database is the most common operation performed on a dataset.

Determining Relations and Attributes

- Determine relevant relations (tables).
- Identify attributes to be returned as the result.
- Restrict and sort the output.

Example: Query Execution

Using data from various tables, determine locations and postal codes for specific customers.

Step 1: Determining Relevant Relations

Table 3: ADDRESS Relation Table

CUSTOMER						
CUS-TOMER ID	CUS-TOMER NUMBER	TITLE	LAST NAME	FIRST NAME	EMAIL	ADDRESS ID
094904	00100	Mr.	Schwarz	David	david.schwarz@hfk-online.de	2
112536	00300	Ms.	Bauer	Silke	silke334@byom.de	3
348902	**00700**	**Mrs.**	**Wagner**	**Lisa**	**lisa-wag-ner@posteo.de**	**7**

Table 5: ADDRESS Table Including Sample Data

BILLINGADDRESS	
ADDRESSID	CUSTOMERID
2	094904
3	098483
2	112536
4	094904
7	**348902**
8	**348902**

Table 4: ADDRESS Table Including Sample Data

ADDRESS						
ADDRESS ID	STREET	HOUSE NUMBER	POSTAL CODE	CITY	COUNTRY	CUS-TOMER ID
2	Berliner Str.	55b	60311	Frankfurt	Germany	094904
3	Marienstr.	25	10117	Berlin	Germany	094904
4	Alte Str.	49	53123	Bonn	Germany	112536
5	Salzburger Str.	27	93047	Regens-burg	Germany	112536
6	Bahn-hofstr.	5	69115	Heidelberg	Germany	112536
7	**Domstr.**	**18**	**20095**	**Hamburg**	**Germany**	**348902**
8	**Stuttgar-ter Str.**	**73**	**01189**	**Dresden**	**Germany**	**348902**

Step 1: Determining Relevant Relations

Using the data from Table 3, Table 4, and Table 5, we aim to determine all locations and postal codes to which David Schwarz had shipments delivered. CUSTOMER and ADDRESS are the key relations. The DELIVERYADDRESS relationship is stored in the attribute CUSTOMERID in the ADDRESS table.

Locating Mr. Schwarz's CUSTOMERID and matching addresses

We need to determine David Schwarz's CUSTOMERID. Once identified, we can locate all addresses with a matching CUSTOMERID in the ADDRESS table.

Table 3: ADDRESS Relation Table

CUSTOMER						
CUS-TOMER ID	CUS-TOMER NUMBER	TITLE	LAST NAME	FIRST NAME	EMAIL	ADDRESS ID
094904	00100	Mr.	Schwarz	David	david.schwarz@hfk-online.de	2
112536	00300	Ms.	Bauer	Silke	silke334@byom.de	3
348902	**00700**	**Mrs.**	**Wagner**	**Lisa**	**lisa-wag-ner@posteo.de**	**7**

Table 5: ADDRESS Table Including Sample Data

BILLINGADDRESS	
ADDRESSID	CUSTOMERID
2	094904
3	098483
2	112536
4	094904
7	**348902**
8	**348902**

Step 3: Filtering Addresses

After locating the relevant addresses, we restrict them based on the POSTAL CODE and LOCATION attributes. The result displays the specific locations.

Table 4: ADDRESS Table Including Sample Data

ADDRESS						
ADDRESS ID	**STREET**	**HOUSE NUMBER**	**POSTAL CODE**	**CITY**	**COUNTRY**	**CUS-TOMER ID**
2	Berliner Str.	55b	60311	Frankfurt	Germany	094904
3	Marienstr.	25	10117	Berlin	Germany	094904
4	Alte Str.	49	53123	Bonn	Germany	112536
5	Salzburger Str.	27	93047	Regens-burg	Germany	112536
6	Bahn-hofstr.	5	69115	Heidelberg	Germany	112536
7	**Domstr.**	**18**	**20095**	**Hamburg**	**Germany**	**348902**
8	**Stuttgar-ter Str.**	**73**	**01189**	**Dresden**	**Germany**	**348902**

Figure: Resulting addresses: (60311 Frankfurt) and (10117 Berlin)

Step 1: Identify Relations and Relationships

Refer to Table 9 to identify the relations and relationships affected by the deletion of Silke Bauer's records. The CUSTOMER and ADDRESS tables, as well as all relationship tables, play a crucial role.

Table 9: CUSTOMER Table before Deletion

CUSTOMER					
CUSTOMERID	**CUSTOMER**	**TITLE**	**LASTNAME**	**FIRST NAME**	**EMAIL**
094904	00100	Mr.	Schwarz	David	david.schwarz@hkf-online.de
112536	**00300**	**Ms.**	**Bauer**	**Silke**	**silke334@byom.de**
348902	00700	Mrs.	Wagner	Lisa	lisa-wagner@posteo.de

Source: Jörg Dreikauss, 2022.

Figure: Table 9: Identifying Relevant Relations and Relationships

Step 2: Determine and Remove Records

Using Table 10, we determine the CUSTOMERID of Ms. Bauer. We then locate and delete all corresponding records. Additionally, addresses 5 and 6 are highlighted for deletion.

Table 10: ADDRESS Table before Deletion

ADDRESS					
ADDRESS	**STREET**	**HOUSE NUMBER**	**POSTAL CODE**	**LOCATION**	**COUNTRY**
2	Berliner Str.	55b	60311	Frankfurt	Germany
3	Marienstr.	25	10117	Berlin	Germany
4	Alte Str.	49	53123	Bonn	Germany
5	**Salzburger Str.**	**27**	**93047**	**Regensburg**	**Germany**
6	**Bahnhofstr.**	**5**	**69115**	**Heidelberg**	**Germany**
7	Domstr.	18	20095	Hamburg	Germany
8	Stuttgarter Str.	73	01189	Dresden	Germany

Source: Jörg Dreikauss, 2022.

Figure: Table 10: Deletion Process Highlighted

Step 3: Verification of Foreign Keys

With Table 11, we ensure all foreign keys remain valid post-deletion. Special attention is given to the entry (3; 098483) which also needs removal.

Table 11: Relationship Tables MAINRESIDENCE, DELIVERYADDRESS, and INVOICEADDRESS before Deletion

MAINRESIDENCE		DELIVERYADDRESS		BILLINGADDRESS	
ADDRESS	**CUSTOMERID**	**ADDRESS**	**CUSTOMERID**	**ADDRESS**	**CUSTOM-ERID**
2	094904	2	094904	2	094904
3	**112536**	3	094904	**3**	**098483**
7	348902	**4**	**112536**	**2**	**112536**
		5	**112536**	4	094904
		6	**112536**	7	348902
		7	348902		
		8	348902		

Source: Jörg Dreikauss, 2022.

Figure: Table 11: Verifying Foreign Keys and Additional Deletions

SQL and Relational Database Systems

Relational databases require a database management system (DBMS) to efficiently operate. Additionally, a standardized interface is essential for querying and manipulating the stored data.

Structured Query Language (SQL)

SQL is an internationally standardized programming language for data queries (ISO/IEC 9075). It offers a universal interface for relational database systems, enabling database creation, controlled data access, and management of access rights.

Sublanguages of SQL

SQL commands can be categorized into:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)
- Transaction Control Language (TCL)
- Data Control Language (DCL)

Structured Query Language (SQL)

SQL is the standard language for relational database management systems. It's used for querying and manipulating data.
Example:

```
SELECT first_name, last_name FROM employees WHERE

department = 'HR';
```

This SQL query retrieves the first and last names of employees in the HR department.

Data Definition Language (DDL)

DDL deals with database schemas and structures.
Example:

```
CREATE TABLE employees (
employee_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
department VARCHAR(50)
);
```

This DDL statement creates a new table called employees.

Transaction Control Language (TCL)

TCL deals with the transactional control of data.
Example:

```
START TRANSACTION;
UPDATE employees SET department = 'Finance' WHERE

last_name = 'Doe';
COMMIT;
```

This TCL example starts a transaction, updates an employee's department, and then commits the changes.

Data Control Language (DCL)

DCL deals with permissions and data access.
Example:

```
GRANT SELECT, UPDATE ON employees TO user123;
```

This DCL statement gives user123 the permission to select and update records in the employees table.

DBMS Implementations

While SQL offers a standardized syntax across different DBMS implementations, there are still variations. These differences can be found in data types for attributes, supported functions, and the execution speeds of individual DBMSs. Hence, changing the database layer in an application is a challenging task.

Typical Relational Database Management Systems

Various DBMSs are available in the market. Some examples are:

- Oracle: One of the oldest and most trusted DBMSs, popular in large enterprises.
- Microsoft SQL Server: A relational database system developed by Microsoft.
- MariaDB: An open-source relational database system, known as a MySQL fork, but has since evolved with its own unique features.
- IBM Db2: A collection of data management products by IBM.
- PostgreSQL: An advanced, open-source relational database system known for its extensibility and strict SQL compliance.
- ... and many more.

Note: This is a brief overview; there are many other commercial DBMSs available in the market.

Comparison of DBMSs

DBMS	**Typ.**	**Use**	**Lic.**	**Feat.**	**Perf.**
MariaDB	O-src	Web	GPL v2	MySQl Rep.	High
Oracle DB	Prop.	Ent.	Comm.	Sec.	V. High
IBM DB2	Prop.	Ent.	Comm.	AI-opt	V. High
MS SQL	Prop.	Bus.	Comm.	Win Int.	High
PostgreSQL	O-src	W/E	PG	Ext.	High

Primary Use Cases of Typical Relational DBMSs

- **MariaDB**
  - Use Case: Primarily used for **Web Applications**.
  - Note: Common choice for web-based platforms and applications.
- **Oracle Database**
  - Use Case: Targets large-scale **Enterprise Applications**.
  - Note: Known for robustness and advanced security features.
- **IBM DB2**
  - Use Case: Designed for **Enterprise Solutions**.
  - Note: Offers AI-powered optimization for better performance.
- **Microsoft SQL Server**
  - Use Case: Commonly used for **Business Solutions**.
  - Note: Tightly integrated with Windows Server and other Microsoft products.
- **PostgreSQL**
  - Use Case: Versatile; used for both **Web** and **Enterprise Applications**.
  - Note: Known for its extensibility and compliance with SQL standards.

Primary Use Cases and Features of Typical Relational DBMSs

- **MariaDB**
  - Use: **Web Apps**
  - Feature: MySQL **Rep**lacement
- **Oracle Database**
  - Use: **Enterprise**
  - Feature: Advanced **Sec**urity
- **IBM DB2**
  - Use: **Enterprise**
  - Feature: **AI Opt**imization
- **Microsoft SQL Server**
  - Use: **Business**
  - Feature: Windows **Integ**ration

What is a MySQL Replacement?

MariaDB serves as a **drop-in replacement** for MySQL:

- **MariaDB**
  - Use: **Web Apps**
  - Feature: MySQL **Rep**lacement
- **Oracle Database**
  - Use: **Enterprise**
  - Feature: Advanced **Sec**urity
- **IBM DB2**
  - Use: **Enterprise**
  - Feature: **AI Opt**imization
- **Microsoft SQL Server**
  - Use: **Business**
  - Feature: Windows **Integ**ration
- **PostgreSQL**
  - Use: **Web/Enterprise**
  - Feature: High **Extensibility**

In essence, MariaDB retains compatibility with MySQL but also offers its own enhancements.

Complex Infrastructure Behind a DBMS

The complexity and infrastructure of a DBMS go beyond just the software. It involves server rooms, networking, and ensuring high availability and reliability.

Overview

- Explain the most important relational database terms.
- Describe how data in relational databases is stored and read in a structured way.
- Explain the meaning and the use of SQL.
- Compare among different commercial DBMSs.

Important Relational Database Terms

Relational database terms form the foundation for understanding database design and operations. Some essential terms include:

- **Relation**: A table consisting of rows and columns.
- **Tuple**: A single row in a table representing a specific item.
- **Attribute**: A column in a table representing a specific property or characteristic.
- **Primary Key**: A unique identifier for a tuple.
- **Foreign Key**: An attribute that creates a link between two tables.

Storing and Reading Data in Relational Databases

Data in relational databases is organized into tables, ensuring efficient storage and retrieval.

- Tables are collections of related data items.
- Each row (or tuple) in a table represents a unique data item or record.
- Columns (or attributes) define the properties of these records.
- Data retrieval is facilitated through structured queries, often using SQL.

Meaning and Use of SQL

SQL (Structured Query Language) is the standard language for interacting with relational databases.

- It allows for data retrieval, insertion, update, and deletion.
- Enables the creation and modification of database structures.
- Facilitates data integrity and security through constraints and permissions.
- Its standard syntax ensures compatibility across different database systems, though there might be system-specific variations.

Comparison Among Commercial DBMSs

Different commercial Database Management Systems (DBMSs) offer varying features, performance metrics, and costs.

- **Oracle Database**: Enterprise-focused, highly scalable, and feature-rich.
- **IBM DB2**: Known for high performance and robust data integration.
- **Microsoft SQL Server**: Integrated with other Microsoft products, suitable for businesses of all sizes.
- **MariaDB**: Open-source and a popular alternative to MySQL.

Considerations for choosing a DBMS include scalability needs, budget constraints, and specific feature requirements.

Transfer Task: Converting Text to Relations

Convert the following part of a job offer letter into a record in a relation by extracting the entities and attributes:

- "Dear Mr. Johns, We are delighted to offer you the sales manager position at ABC bank, Branch 4 in Berlin, with an annual salary of €122,000 in addition to the health benefits and vacation time. The start date for this position will be January 23. Your supervisor will be Mme. Franklin and her office number is 215."

Transfer Task: Converting Text to Relations

Entities and Attributes from a Job Offer Letter:

- Name: Mr. Johns
- Position: Sales Manager
- Company: ABC Bank
- Branch: 4 in Berlin
- Salary: €122,000
- Benefits: Health benefits, vacation time
- Start Date: January 23
- Supervisor: Mme. Franklin
- Office Number: 215

Installing MariaDB on Windows

1. Visit the official MariaDB website and download the Windows MSI package.
2. Run the MSI installer.
3. Follow the installation wizard:
   - Choose components to install (Server, Client, etc.).
   - Set the root password.
   - Configure the service settings.
4. Complete the installation.
5. Verify the installation by opening the MariaDB command prompt.

Installing MariaDB on macOS

1. Use Homebrew: `brew install mariadb`
2. After installation, start MariaDB: `brew services start mariadb`
3. To ensure MariaDB starts on login: `ln -sfv /usr/local/opt/mariadb/*.plist /Library/LaunchAgents`
4. Verify the installation: `mysql -u root`

Installing MariaDB on Linux

1. For Debian/Ubuntu:
   - `sudo apt-get update`
   - `sudo apt-get install mariadb-server`
2. For Red Hat/CentOS:
   - `sudo yum install mariadb-server`
3. Start the service: `sudo systemctl start mariadb`
4. Enable at boot: `sudo systemctl enable mariadb`
5. Secure the installation: `sudo mysql_secure_installation`
6. Verify the installation: `mysql -u root -p`

Installing PostgreSQL on Windows

1. Visit the official PostgreSQL download page: `https://www.postgresql.org/download/windows/`
2. Click on the "Download the installer" link.
3. Choose the desired version and download the installer.
4. Run the installer and follow the on-screen instructions.
5. During installation, you will be prompted to select components, leave the default selection.
6. Set the password for the 'postgres' user.
7. Choose a port (default is 5432).
8. Complete the installation.

Installing PostgreSQL on Mac

1. Visit the official PostgreSQL download page: `https://www.postgresql.org/download/macosx/`
2. You can use the interactive installer by EDB or Homebrew.
3. For Homebrew:
   - Open Terminal.
   - Run `brew install postgresql`
   - Start the service using `brew services start postgresql`
4. For EDB installer, download and run the installer. Follow on-screen instructions.

Installing PostgreSQL on Linux

1. Installation methods vary across different Linux distributions.
2. For Debian/Ubuntu:
   - `sudo apt-get update`
   - `sudo apt-get install postgresql postgresql-contrib`
3. For Fedora:
   - `sudo dnf install postgresql-server`
   - `sudo systemctl enable postgresql`
   - `sudo systemctl start postgresql`
4. For other distributions, refer to the official documentation: `https://www.postgresql.org/download/linux/`