INTRODUCTION TO RELATIONAL DATABASES

Rodrigo García Carmona

Universidad San Pablo-CEU

Escuela Politécnica Superior

INTRODUCTION

WHAT IS A DBMS?

- DBMS (DataBase Management Systems) are:
  - **Persistent**, **efficient**, **reliable**, **convenient** and **secure** storage solutions for **huge data sets** that must be accessed by **multiple users** at the same time.

CHARACTERISTICS OF A DBMS

- **Persistent:** The data must remain after turning off the machines.
- **Efficient:** Thousands of queries or updates per second.
- **Reliable:** Available during 99.99999% of time.
- **Convenient:**
  - Decoupled from the physical data shape.
  - High-level query languages.
- **For huge data sets:** Terabytes or even more.
- **For multiple users:** Concurrent access control.

LAYERED STRUCTURE

- Applications that access the database are usually designed to use a *framework*.
- DBMS are usually bundled together with specific *middleware*.
- Applications that use a DBMS should not need to know about its characteristics…
  - …or even if the DBMS really exists!
- During this course we will use Java *middleware* and *frameworks*.

KEY IDEAS

- The data source might be:
  - A registry set or tabulated data
  - Hierarchical or network data models
  - Unorganized information
- In a DBMS we must define schemas and fill them with data.
- We can interact with a DBMS through 3 different languages:
- **DDL:** Data Definition Language
  - To define schemas
- **DML:** Data Modification Language
  - To make queries and modifications
- **DCL:** Data Control Language
  - To manage user access

KEY ROLES

- Several people interact with a DBMS through its lifecycle:
- **DBMS Implementer**
  - Builds the DBMS itself
- **Database designer**
  - Creates the schemas
- **Application Developer**
  - Programs applications that interact with the DBMS
- **Database administrator**
  - Manages the actual data
  - Maintains the DBMS

IMPORTANCE

- Knowingly or not, we are constantly using databases…
  - Internet services (email, social networks)
  - IT and computers
  - Logistics and trading
  - Government and administration
  - Banking and finance
  - **Healthcare**
  - …

RELATIONAL DATABASES

RELATIONAL MODEL

- The relational model is used in most real DBMS…
  - …although NoSQL is gaining traction.
- Very simple model.
- Two-dimensional tables.
- Built from mathematical **sets** and **relations**.
- Queries and updates are made using high-level languages…
  - …that are easy but expressive.
  - SQL or SQL-like.
- Very efficient, with advanced implementations.

BASIC TERMINOLOGY (I)

- **Database:** A set of **relations** (**tables**).
- Each relation has a set of **attributes** (**fields**, **columns**), identified by its name.
- Each **tuple** (**record**, **row**) has a **value** for each attribute.
- Each attribute has a **type** (**domain**).
- An attribute can be **unique** (no repetition in values among different tuples)

students

id	name	score	photo
123	Anna	6.5	^_^
234	Bob	3.3	NULL
345	Mike	NULL	-_-
…	…	…	…

universities

name	city	students
CEU	Madrid	11500
UPV	Valencia	40000
MIT	Cambridge	10000
…	…	…

BASIC TERMINOLOGY (II)

- **Schema:** Structural description of a database.
- **Instance:** Database contents at a given moment.
- **Primary Key:** Attribute or combination of attributes used to identify a row.
- **Foreign Key:** Attribute that references a Primary Key of other table.
- **NULL:** Special value that means “unknown” or “undefined”.
- An attribute can be **NOT NULL** (NULL is forbidden).

students

id	name	score	photo
123	Anna	6.5	^_^
234	Bob	3.3	NULL
345	Mike	NULL	-_-
…	…	…	…

universities

name	city	students
CEU	Madrid	11500
UPM	Madrid	40000
MIT	Cambridge	10000
…	…	…

HOW TO CREATE AND USE A RELATIONAL DATABASE

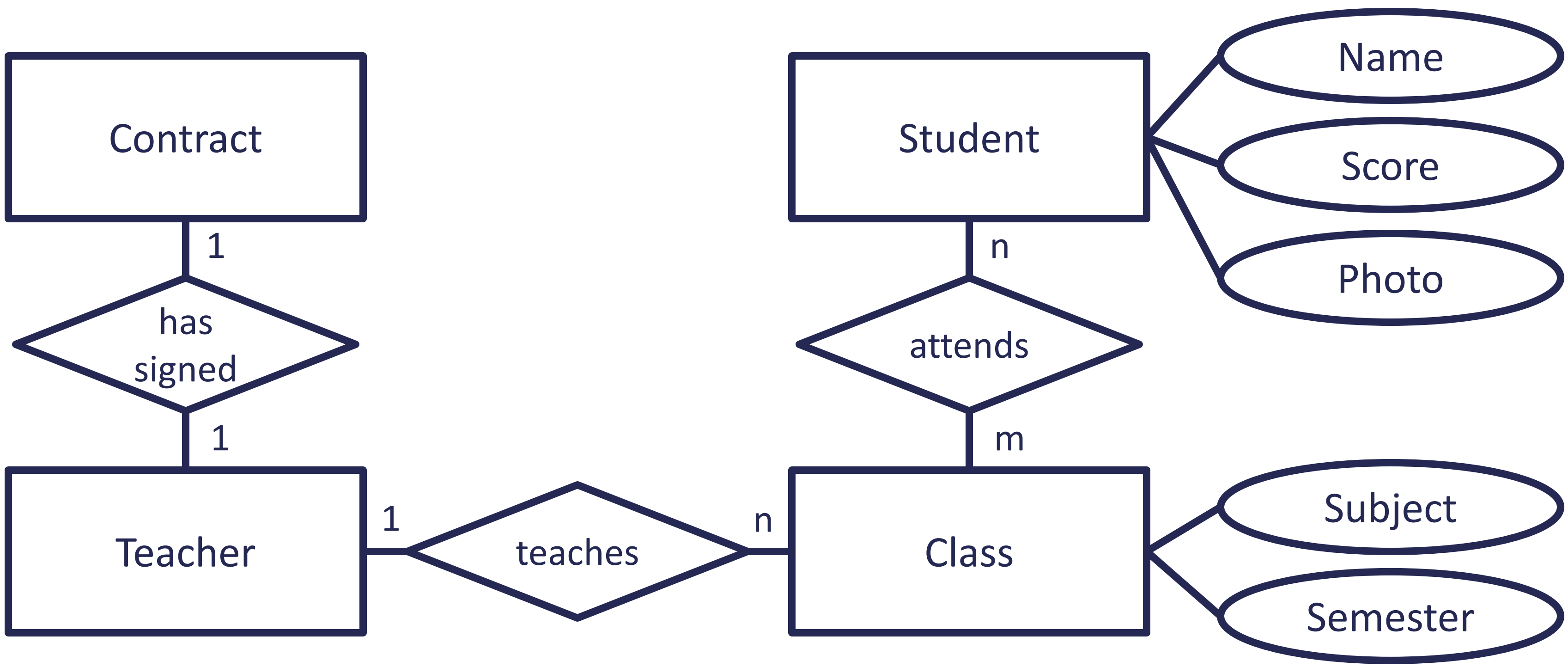
- **To create:**
  1. Design the schema (on paper!)
  2. Write down the schema using a **DDL**
  3. Load the initial dataset using **DML**.
- **To use:**
  1. Think about the query (on paper!)
  2. Write down the query using a **DML**

SCHEMA DEFINITION

ENTITY-RELATIONSHIP MODEL

- The **E-R (Entity-Relationship) model** is an analysis tool used to build databases:
- **Entity:** a type of object in the real world.
    - Represented with a square box.
    - Each entity might have properties.
        - Represented with ovals.
- **Relationship:** how two entities are related.
    - Represented with a diamond with two lines
    - **Relationship types:**
        - One-to-One
        - One-to-Many
        - Many-to-Many

SAMPLE E-R MODEL



DATABASE NORMALIZATION

- **Database normalization** is the process of organizing the attributes and tables of a database to minimize redundancy.
- We transform one table into several smaller tables.
- With normalization we aim to:
    - **Minimize redundancy:** Data should not be duplicated.
    - **Isolate data:** Insertions, updates and deletes should affect only one table.
    - **Avoid losing information:** Relationships between tables are expressed with foreign keys.
- **Normal forms:**
    - **1st Normal Form (1NF):** Only one value per field.
      "No duplicate rows"
    - **2nd Normal Form (2NF):** Transitive functional dependency is allowed.
      "Values are determined from the key or from a value determined from the key"
    - **3rd Normal Form (3NF):** Only non-transitive functional dependency is allowed.
      "Values con only be determined from the key"

UNNORMALIZED FORM

invoice id	date	client id	client name	product id	product name	product price	VAT	amount
001	2014-09-17	C01	Dracotienda	ISH01	La Puerta de Ishtar	38.00€	4%	10
				ISH02	Ishtar – Pantalla	19.50€	4%	5
002	2014-09-17	C02	Tesoros de la Marca	ISH01	La Puerta de Ishtar	38.00€	4%	7
				ABL01	Ablaneda	14.00€	4%	3
003	2015-01-10	C01	Dracotienda	ABL01	Ablaneda	14.00€	4%	10
				RYU01	Ryuutama	24.00€	4%	5

1ST NORMAL FORM

invoice id	date	client id	client name
001	2014-09-17	C01	Dracotienda
002	2014-09-17	C02	Tesoros de la Marca
003	2015-01-10	C01	Dracotienda

invoice id	product id	product name	product price	VAT	amount
001	ISH01	La Puerta de Ishtar	38.00€	4%	10
001	ISH02	Ishtar - Pantalla	19.50€	4%	5
002	ISH01	La Puerta de Ishtar	38.00€	4%	7
002	ABL01	Ablaneda	14.00€	4%	3
003	ABL01	Ablaneda	14.00€	4%	10
003	RYU01	Ryuutama	24.00€	4%	5

2^{ND} NORMAL FORM

invoice id	product id	amount
001	ISH01	10
001	ISH02	5
002	ISH01	7
002	ABL01	3
003	ABL01	10
003	RYU01	5

product id	product name	product price	VAT
ISH01	La Puerta de Ishtar	38.00€	4%
ISH02	Ishtar - Pantalla	19.50€	4%
ABL01	Ablaneda	14.00€	4%
RYU01	Ryuutama	9.50€	4%

invoice id	date	client id	client name
001	2014-09-17	C01	Dracotienda
002	2014-09-17	C02	Tesoros de la Marca
003	2015-01-10	C01	Dracotienda

3RD NORMAL FORM

product id	product name	product price	VAT
ISH01	La Puerta de Ishtar	38.00€	4%
ISH02	Ishtar - Pantalla	19.50€	4%
ABL01	Ablaneda	14.00€	4%
RYU01	Ryuutama	24.00€	4%

invoice id	product id	amount
001	ISH01	10
001	ISH02	5
002	ISH01	7
002	ABL01	3
003	ABL01	10
003	RYU01	5

client id	client name
C01	Dracotienda
C02	Tesoros de la Marca

invoice id	client id	date
001	C01	2014-09-17
002	C02	2014-09-17
003	C01	2015-01-10

HOW TO CREATE TABLES USING SQL

```
CREATE TABLE students (
id INTEGER NOT NULL,
name VARCHAR(255) UNIQUE NOT NULL,
score FLOAT,
photo BLOB,
PRIMARY KEY(id)
);

CREATE TABLE universities (
name VARCHAR(255) NOT NULL,
city VARCHAR(255) NOT NULL,
students INTEGER,
PRIMARY KEY(name)
);
```

HOW TO LOAD DATA USING SQL

```
INSERT INTO students (id, name, score, photo)
VALUES (123, ‘Anna’, 6.5, ^_^);
INSERT INTO students (id, name, score)
VALUES (234, ‘Bob’, 3.3);
INSERT INTO students (id, name, photo)
VALUES (345, ‘Mike’, -_-);

INSERT INTO universities (name, city, students)
VALUES (‘CEU’, ‘Madrid’, 11500);
INSERT INTO universities (name, city, students)
VALUES (‘UPV’, ‘Valencia’, 40000);
INSERT INTO universities (name, city, students)
VALUES (‘MIT’, ‘Cambdrige’, 10000);
```

FROM E-R DIAGRAM TO TABLES

- We can turn an E-R diagram directly into a set of normalized tables by applying the following rules:
  - Each **entity** becomes a **table**.
  - Each **property** becomes a **column** in a table.
  - Each table must have a **primary key**. If no property can fulfill this role, or we don’t want any of them to do it, we must add a primary column to each entity table.
  - Each **1-to-1 relationship** becomes a **foreign key** column in the table of one of its sides, you can chose which. This foreign key points to the primary key of the other side.
  - Each **1-to-n relationship** becomes a **foreign key** column in the table of the ‘n’ side. This foreign key points to the primary key of the ‘1’ side.
  - Each **n-to-n relationship** becomes a **table** with **two foreign key** columns, each pointing to the primary key of one of the sides. The primary key of this new table is the combination of the two foreign key columns.

QUERIES

NATURAL LANGUAGE VS. QUERY LANGUAGE

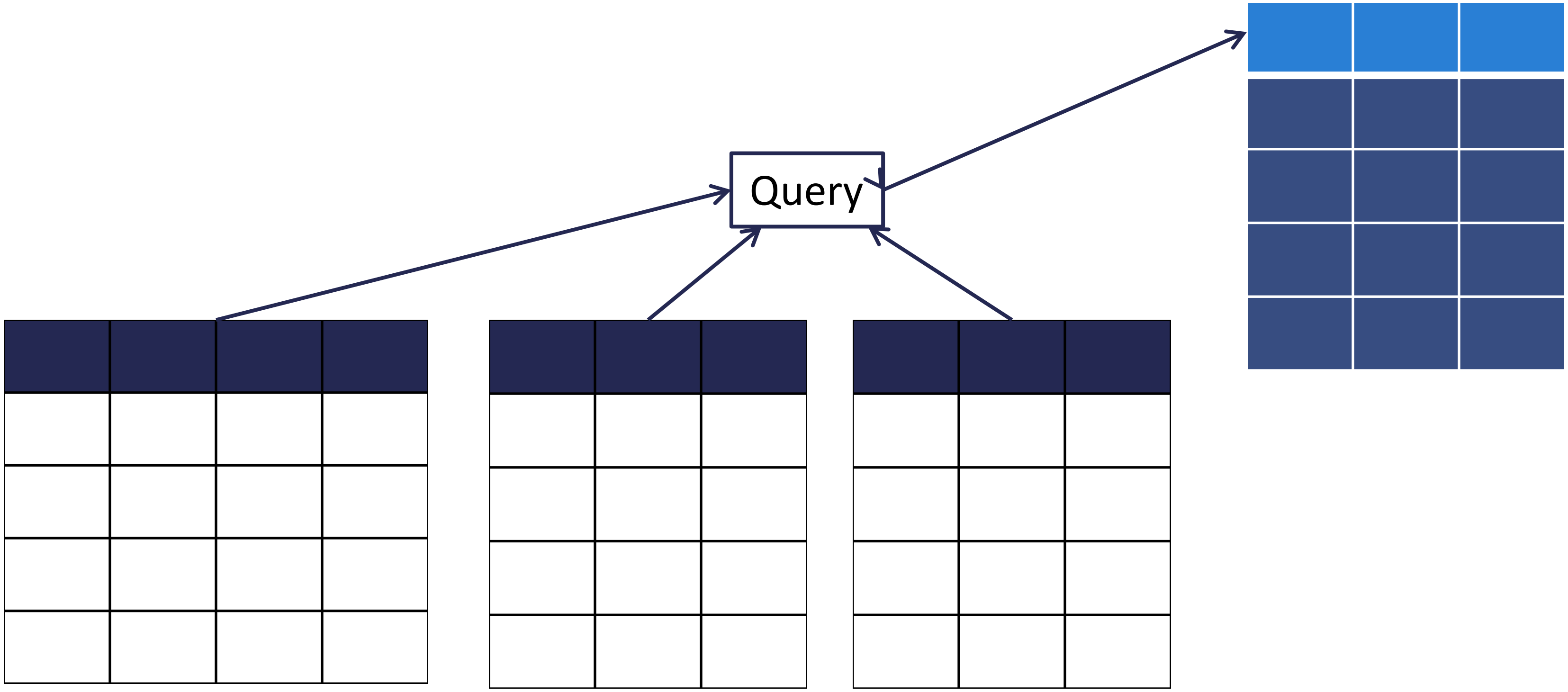
- Sample queries using natural language:
  - *The names of all universities in Madrid with more than 20000 students.*
  - *All students with a score of less than 8.5 than want to apply to MIT.*
  - *The university with the highest admitted students' average score.*
- These sample queries using query language:
  - ```
    SELECT name FROM universities
    WHERE city IS Madrid AND students > 20000
    ```
  - ```
    SELECT * FROM students, applications
    WHERE students.id=applications.id
    AND students.score < 8.5
    AND applications.university IS 'MIT'
    ```
  - Out of the scope of this lesson…

QUERY LANGUAGES

- Queries could be relatively easy or difficult…
  - For the user: to build.
  - For the database: to execute efficiently.
  - **There is no correlation** between these two aspects.
- The query language is the DML, and it is also used to modify the data in a database, not only to access it.
- There are several query languages:
  - **SQL:**
    - The most used. Shown in previous slide.
  - **Relational Algebra:**
    - Very formal. Not used in practice.
    - $\pi_{id}\sigma_{score<8.5 \wedge university='MIT'}(students*applications)$

DATA RETURNED

- After a query, the DBMS gives its response in the form of **tables**.



DATA EXTRACTION OPERATIONS

- Relational databases follow rules from discrete mathematics.
- Data extraction operations (queries!) are built upon set and relations theory.
- **Set operations:**
  - **Union:** Tables with the same schema.
  - **Difference:** Tables with the same schema.
  - **Intersection:** Tables with the same schema.
  - **Cartesian product:** Tables with the same or different schemas.
- **Relational operations:**
  - **Projection:** Extracts a column.
  - **Selection:** Extracts a row.
  - **Join:** Builds a new table from other two, following a join condition.
  - **Division:** From two tables, extracts the rows from the first table that are also in the second, but only the columns that aren't in the second.

DML SENTENCES

- We have called all sentences “queries”, but there are several types:
  - **Selection:** To extract data.
    - SELECT in SQL.
  - **Insertion:** To add new data.
    - INSERT in SQL.
  - **Modification:** To change already existing data.
    - UPDATE in SQL.
  - **Deletion:** To remove already existing data.
    - DELETE in SQL.
- These queries realize the CRUD (Create, Read, Update, Delete) functions of persistent storage.