3 – Databases and Logical data modelling

Data Management

Michele Mastroianni
Michele.mastroianni@unicampania.it
mmastroianni@unisa.it

Università degli Studi della Campania *Luigi Vanvitelli*

Dipartimento di Matematica e Fisica

Database

What is a database ?

- A collection of files storing related data

Examples of databases

- Accounts database; payroll database; University students database; Amazon's products database; airline reservation database

An Example: Online Bookseller

What data do we need?

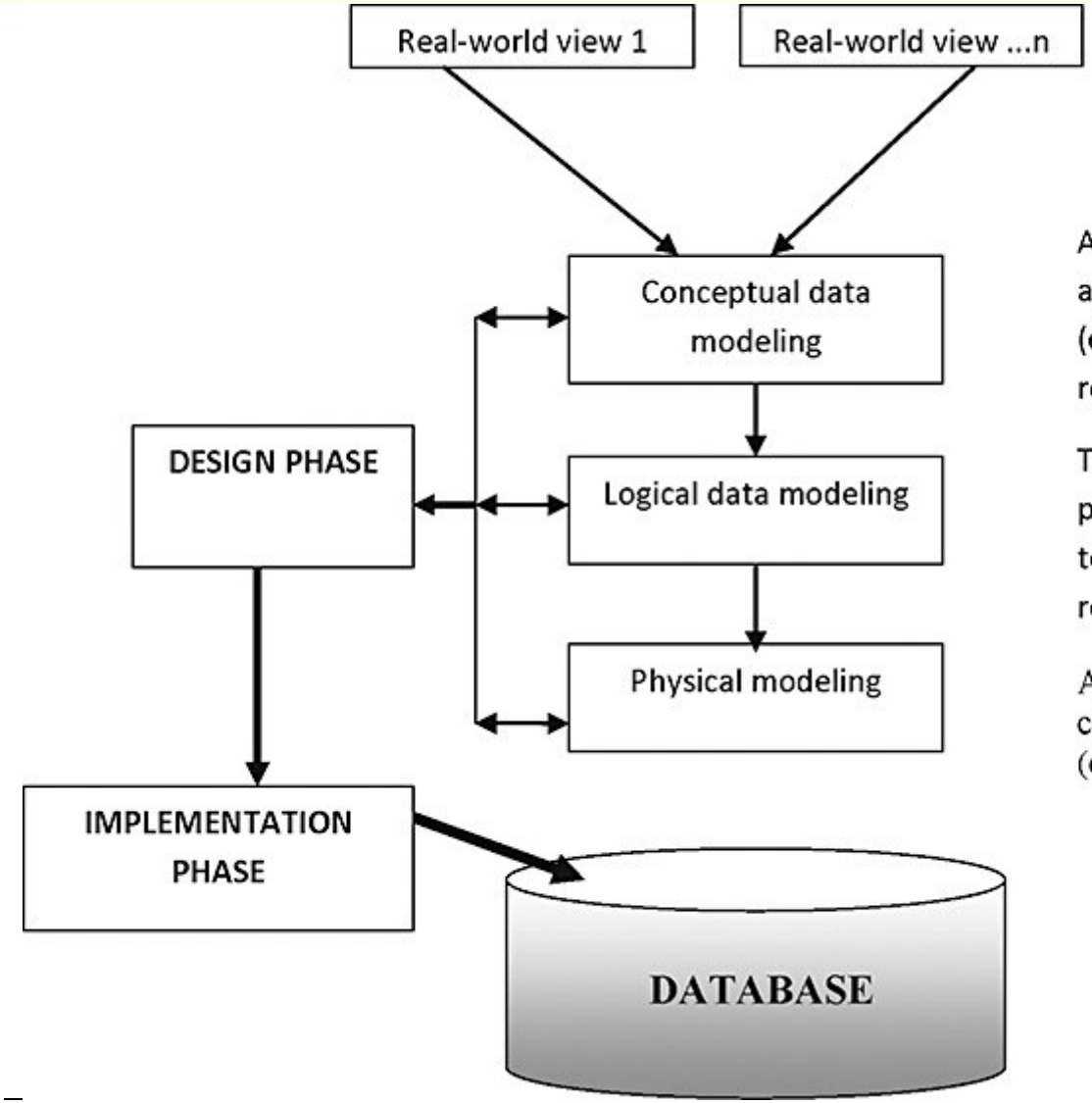
- Data about books, customers, pending orders, order histories, trends, preferences, etc.
- Data about sessions (clicks, pages, searches)

What capabilities on the data do we need?

- Insert/remove books, find books by author/title/etc., analyze past order history, recommend books, …
- Data must be accessed efficiently, by many users
- Data must be safe from failures and malicious users

What a DBMS Does

- Describe real-world entities in terms of stored data
- Persistently store large datasets
- Efficiently query & update
- Change structure (e.g., add attributes)
- Concurrency control: enable simultaneous updates
- Security and integrity



The real - world modeled relative to user - oriented perspective

All datasets required for the application are identified (entities, properties and relationships, constraints)

Translation of entities, properties and relationships to suit data structure (e.g. relational, object-oriented)

Apportioning of datasets to computer storage units (e.g. bytes)

Relational Model

columns / attributes / fields

- Data is a collection of relations / tables:

rows / tuples / records

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

- mathematically, relation is a set of tuples
  - each tuple appears 0 or 1 times in the table
  - order of the rows is unspecified

The Relational Data Model

- Degree of a relation = #attributes
- Each attribute has a type.
  - Examples types:
    - Strings: CHAR(20), VARCHAR(50), TEXT
    - Numbers: INT, SMALLINT, FLOAT
    - MONEY, DATETIME, …
    - Few more that are vendor specific

Keys

- Key = one (or multiple) attributes that uniquely identify a record

Key

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Keys

- Key = one (or multiple) attributes that uniquely identify a record

Key

Not a key

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Keys

- Key = one (or multiple) attributes that uniquely identify a record

Key

Not a key

Is this a key?

No: future updates to the database may create duplicate no_employees

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Multi-attribute Key

Key = fName,lName
(what does this mean?)

fName	lName	Income	Department
Alice	Smith	20000	Testing
Alice	Thompson	50000	Testing
Bob	Thompson	30000	SW
Carol	Smith	50000	Testing

Multiple Keys

Key

Another key

SSN	fName	lName	Income	Department
111-22-3333	Alice	Smith	20000	Testing
222-33-4444	Alice	Thompson	50000	Testing
333-44-5555	Bob	Thompson	30000	SW
444-55-6666	Carol	Smith	50000	Testing

We can choose one key and designate it as *primary key*

E.g.: primary key = SSN

Foreign Key

```
Company(cname, country, no_employees, for_profit)
Country(name, population)
```

Foreign key to Country.name

Company

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

Country

name	population
USA	320M
Japan	127M

Keys: Summary

- Key = columns that uniquely identify tuple
  - Usually we underline
  - A relation can have many keys, but only one can be chosen as *primary key*
- Foreign key:
  - Attribute(s) whose value is a key of a record in some other relation
  - Foreign keys are sometimes called *semantic pointer*

SQL

- Originally 'Sequel' - Structured English query Language, part of an IBM project in the 70's
- Sequel was already taken, so it became SQL - Structured Query Language
- ANSI Standards
  - SQL-86, 89, 92, 99, 2003
  - Current SQL:2008
- Most modern DBMS use a variety of SQL
  - Few (if any) are true to the standard
  - Oracle 10g SQL which we will be using is mostly compliant to SQL:2003

SQL

- SQL provides
  - A data definition language (DDL)
  - A data manipulation language (DML)
  - A data control language (DCL)
- In addition SQL
  - Can be used from other languages
  - Is often extended to provide common programming constructs (such as if-then tests, loops, variables, etc.)

Notes

- SQL is (usually) not case-sensitive, but we'll write SQL keywords in upper case for emphasis
- SQL statements will be written in **`BOLD COURIER FONT`**
- Strings in SQL are surrounded by single quotes:
  `'I AM A STRING'`
- Single quotes within a string are doubled:
  `'I''M A STRING'`
- The empty string:`''`

Non-Procedural Programming

- SQL is a declarative (non-procedural) language
  - Procedural - say exactly what the computer has to do
  - Non-procedural – describe the required result (not the way to compute it)
- Example: Given a database with tables
  - Student with attributes ID, Name, Address
  - Module with attributes Code, Title
  - Enrolment with attributes ID, Code
- Get a list of students who take the module 'Database Systems'

Non-Procedural (SQL)

```
SELECT Name FROM Student, Enrolment
WHERE (Student.ID = Enrolment.ID)
  AND (Enrolment.Code =
     (SELECT Code FROM Module WHERE
        Title = 'Database Systems'))
```

CREATE TABLE

```
CREATE TABLE
<name> (
  <col-def-1>,
  <col-def-2>,
        :
  <col-def-n>,
  <constraint-1>,
        :
  <constraint-k>)
```

- You supply
  - A name for the table
  - A list of column definitions
  - A list of constraints (such as keys)

Column Definitions

```
<col-name> <type>
 [NULL|NOT NULL]
 [DEFAULT <val>]
 [constraint-1 [,
  constraint-2[,
   ...]]]
```

- Each column has a name and a type
- Common types
  - `INT`
  - `REAL`
  - `CHAR(n)`
  - `VARCHAR(n)`
  - `DATE`

Column Definitions

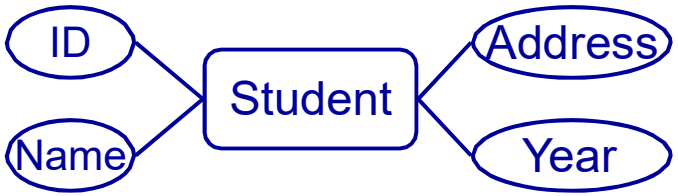
- Columns can be specified as **`NULL`** or **`NOT NULL`**
- **`NOT NULL`** columns cannot have missing values
- If neither is given then columns are assumed **`NULL`**

- Columns can be given a default value
- You just use the keyword DEFAULT followed by the value, e.g.:

```
num INT DEFAULT 0
```

Example

```
CREATE TABLE Student (
  stuID INT NOT NULL,
  stuName VARCHAR(50) NOT NULL,
  stuAddress VARCHAR(50),
  stuYear INT DEFAULT 1)
```



Constraints

CONSTRAINT

<name>

<type>

<details>

- Common **<type>**s
  - **PRIMARY KEY**
  - **UNIQUE**
  - **FOREIGN KEY**
  - **INDEX**

- Each constraint is given a name – Access SQL requires a name, but some others don't
- Constraints which refer to single columns can be included in their definition

Primary Keys

- Primary Keys are defined through constraints
- A **PRIMARY KEY** constraint also includes a **UNIQUE** constraint and makes the columns involved **NOT NULL**

- The **<details>** for a primary key is a list of columns which make up the key

```
CONSTRAINT <name>
  PRIMARY KEY
  (col1, col2, …)
```

Unique Constraints

- As well as a single primary key, any set of columns can be specified as **UNIQUE**
- This has the effect of making candidate keys in the table
- The **<details>** for a unique constraint are a list of columns which make up the candidate key

```
CONSTRAINT <name>
  UNIQUE
  (col1, col2, …)
```

Example

```
CREATE TABLE Student (
  stuID INT NOT NULL,
  stuName VARCHAR(50) NOT NULL,
  stuAddress VARCHAR(50),
  stuYear INT DEFAULT 1,
  CONSTRAINT pkStudent
    PRIMARY KEY (stuID))
```

Deleting Tables

- To delete a table use
- **DROP TABLE**
- **[IF EXISTS]**
- **<name>**
- Example:
- **DROP TABLE Module**

- BE CAREFUL with any SQL statement with DROP in it
  - You will delete any information in the table as well
  - You won't normally be asked to confirm
  - There is no easy way to undo the changes

Changing Tables

- Sometimes you want to change the structure of an existing table
  - One way is to DROP it then rebuild it
  - This is dangerous, so there is the ALTER TABLE command instead
- ALTER TABLE can
  - Add a new column
  - Remove an existing column
  - Add a new constraint
  - Remove an existing constraint

ALTERing Columns

To add or remove columns use

```
ALTER TABLE <table>
 ADD COLUMN <col>

ALTER TABLE <table>
 DROP COLUMN <name>
```

Examples

```
ALTER TABLE Student
  ADD COLUMN
  Degree VARCHAR(50)

ALTER TABLE Student
  DROP COLUMN Degree
```

ALTERing Constraints

To add or remove columns use

```
ALTER TABLE <table>
  ADD CONSTRAINT
      <definition>

ALTER TABLE <table>
  DROP CONSTRAINT
      <name>
```

Examples

```
ALTER TABLE Module
  ADD CONSTRAINT
  ck UNIQUE (title)

ALTER TABLE Module
  DROP CONSTRAINT ck
```

INSERT, UPDATE, DELETE

- **`INSERT`** - add a row to a table
- **`UPDATE`** - change row(s) in a table
- **`DELETE`** - remove row(s) from a table
- **`UPDATE`** and **`DELETE`** use '**`WHERE`** clauses' to specify which rows to change or remove
- BE CAREFUL with these - an incorrect **`WHERE`** clause can destroy lots of data

INSERT

```
INSERT INTO
  <table>
  (col1, col2, …)
  VALUES
  (val1, val2, …)
```

- The number of columns and values must be the same
- If you are adding a value to every column, you don't have to list them
- SQL doesn't require that all rows are different (unless a constraint says so)

INSERT

Student

ID	Name	Year
1	John	1

```
INSERT INTO Student
 (ID, Name, Year)
 VALUES (2, 'Mary', 3)
```

Student

ID	Name	Year
1	John	1
2	Mary	3

```
INSERT INTO Student
 (Name, ID)
 VALUES ('Mary', 2)
```

Student

ID	Name	Year
1	John	1
2	Mary	

```
INSERT INTO Student
 VALUES (2, 'Mary', 3)
```

Student

ID	Name	Year
1	John	1
2	Mary	3

UPDATE

```
UPDATE <table>
SET col1 = val1
  [,col2 = val2…]
[WHERE
  <condition>]
```

- All rows where the condition is true have the columns set to the given values
- If no condition is given all rows are changed so BE CAREFUL
- Values are constants or can be computed from columns

UPDATE

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Mary	2

```
UPDATE Student
SET Year = 1,
    Name = 'Jane'
WHERE ID = 4
```

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Jane	1

```
UPDATE Student
SET Year = Year + 1
```

Student

ID	Name	Year
1	John	2
2	Mark	4
3	Anne	3
4	Mary	3

DELETE

- Removes all rows which satisfy the condition

```
DELETE FROM
  <table>
   [WHERE
   <condition>]
```

- If no condition is given then ALL rows are deleted - BE CAREFUL
- Some versions of SQL also have `TRUNCATE TABLE <T>` which is like `DELETE FROM <T>` but it is quicker as it doesn't record its actions

DELETE

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Mary	2

```
DELETE FROM
  Student
  WHERE Year = 2
```

Student

ID	Name	Year
1	John	1
2	Mark	3

```
DELETE FROM Student
```

or

```
TRUNCATE TABLE Student
```

Student

ID	Name	Year

SELECT

- The SQL command you will use most often
  - Queries a set of tables and returns results as a table
  - Lots of options, we will look at many of them
  - Usually more than one way to do any given query

- SQL's SELECT is different from the relational algebra's selection σ
- We'll see translation of SQL queries into relational algebra later

SQL SELECT Overview

```
SELECT
  [DISTINCT | ALL] <column-list>
  FROM <table-names>
  [WHERE <condition>]
  [ORDER BY <column-list>]
  [GROUP BY <column-list>]
  [HAVING <condition>]
```

- *([]- optional, | - or)*

Simple SELECT

```
SELECT <columns>
  FROM <table>
```

`<columns>` can be

- A single column
- A comma-separated list of columns
- `*` for 'all columns'

- Given a table Student with columns
  - stuID
  - stuName
  - stuAddress
  - stuYear

Sample SELECTs

```
SELECT * FROM Student
```

stuID	*stuName*	*stuAddress*	*stuYear*
1	Anderson	15 High St	1
2	Brooks	27 Queen's Rd	3
3	Chen	Lenton Hall	1
4	D'Angelo	Derby Hall	1
5	Evans	Lenton Hall	2
6	Franklin	13 Elm St	3
7	Gandhi	Lenton Hall	1
8	Harrison	Derby Hall	1

Sample SELECTs

```
SELECT stuName FROM Student
```

stuName
Anderson
Brooks
Chen
D'Angelo
Evans
Franklin
Gandhi
Harrison

Sample SELECTs

```
SELECT stuName, stuAddress
  FROM Student
```

stuName	*stuAddress*
Anderson	15 High St
Brooks	27 Queen's Rd
Chen	Lenton Hall
D'Angelo	Derby Hall
Evans	Lenton Hall
Franklin	13 Elm St
Gandhi	Lenton Hall
Harrison	Derby Hall

References

1. Elmasri , R., Navathe, S.B., **Fundamentals of Database Systems**, 7th Edition, Pearson Ed., 2016, ISBN: 978-0133970777

Italian readers could prefer

1. Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (2006). **Basi di dati: modelli e linguaggi di interrogazione (seconda edizione)**. McGraw-Hill.