Databases

Lecture 2

The Relational Model

The Relational Data Model

- proposed by Edgar Codd in 1970
    - 1981 – Turing Award
- the dominant data model today (represented by DBMSs like Oracle, SQL Server, etc)
- advantages
    - **simple data representation**
    - queries, even complex ones, can be easily expressed, using a **simple, high-level language**
- even non-technical users can understand the contents of a database

The Relational Data Model

1. relations
2. integrity constraints
3. relational databases
4. managing relational databases

1. Relations

- in an application, a **data collection** is used according to a **model**
- in the **relational model**, the data collection is organized as a set of **relations** (tables)
- a relation has a relation **schema** and a relation **instance**
- the relation instance is a table
- the relation schema describes the table's column heads

1. Relations

- the schema specifies the relation's name, and the name and domain for each field
- the domain of a field is specified through its name and has a set of associated values
- schema example for the Students relation:

  Students[*name*, *year_of_study*, *email*]
  or notation:

  Students(*name*: string, *year_of_study*: integer, *email*: string)

1. Relations

- instance example for the Students relation:

attributes (columns, fields)

name	year_of_study	email
BACOŢIU DENISA-CRISTINA	2	bde@cs.ubb.ro
MAXIM GEORGIANA-ELENA	2	mge@cs.ubb.ro
BATINCU DANIEL	2	bda@cs.ubb.ro

rows (records, tuples)

1. Relations

- **{A_1, A_2, ..., A_n}** - a set of attributes
- **D_i** = **Dom(A_i)** U **{?} -** the **domain of possible values** for **attribute A_i**
  - the **undefined** (**null**) value is denoted here by {?}; it is used to check if a value has been assigned to an attribute (the attribute could have the *undefined* value); this value doesn't have a certain data type; attribute values of different types can be compared against this value (numeric, string, etc)
- relation of **arity** (degree) n: $R \subseteq D_1 \times D_2 \times ... \times D_n$
- **R[A_1, A_2, ... , A_n]** - the **relation schema**

1. Relations

- the relation can be stored in a table of the form:

R	A_1	…	A_j	…	A_n
r_1	a_{11}	…	a_{1j}	…	a_{1n}
…	…	…	…	…	…
r_i	a_{i1}	…	a_{ij}	…	a_{in}
…	…	…	…	…	…
r_m	a_{m1}	…	a_{mj}	…	a_{mn}

where $a_{ij} \in D_j$, $j = 1,\ldots,n$, $i = 1,\ldots,m$.

1. Relations

- the values of an attribute are **atomic** and **scalar**
- the rows in a table **are not ordered**
    - e.g., 2 representations of the same *Students* relation instance

name	year_of_study	email
BACOŢIU DENISA-CRISTINA	2	bde@cs.ubb.ro
MAXIM GEORGIANA-ELENA	2	mge@cs.ubb.ro
BATINCU DANIEL	2	bda@cs.ubb.ro

name	year_of_study	email
MAXIM GEORGIANA-ELENA	2	mge@cs.ubb.ro
BACOŢIU DENISA-CRISTINA	2	bde@cs.ubb.ro
BATINCU DANIEL	2	bda@cs.ubb.ro

- the records in a table **are distinct** - a relation is defined as a set of distinct tuples (however, DBMSs do allow tables to contain duplicates)

1. Relations

- the **cardinality** of an instance is the number of tuples it contains

1. Relations

- a relation that is not well-designed:

Students[name, program, year_of_study, group]

name	program	year_of_study	group
Alexandra Mihai	CS ro	2	222
Vlad Dan	CS ro	2	222
Andreea Stancu	CS ro	2	226
Flaviu Pop	CS ro	3	226

  - the association between group=222 and (program='CS ro', year_of_study=2) is stored 2 times, for the first 2 students
  - the last 2 students are in the same group, but in different years - error
- to avoid such situations, relations must be **normalized**

2. Integrity Constraints

- **integrity constraints (restrictions)** are **conditions specified** on the **database schema**, restricting the data that can be stored in the database
- these constraints are **checked** when the data is **changed**; operations that violate the constraints are not allowed (e.g., introducing a student with a CNP identical to a different student's CNP, entering data of the wrong type in a column, etc)
- examples: **domain** constraints, **key** constraints, **foreign key** constraints

- **domain** constraints - conditions that must be satisfied by every relation instance: a column's values belong to its associated domain

2. Integrity Constraints

- **key** constraint - a constraint stating that a **subset of the attributes** in a relation is a **unique identifier** for every tuple in the relation; this subset of attributes is minimal
- $K \subseteq \{A_1, A_2, \ldots, A_n\}$ is a **key** for relation $R[A_1, A_2, \ldots, A_n]$ if:
  - the attributes in K can be used to identify every record in R
  - no subset of K has this property
- two different records are not allowed to have identical values in all the fields that constitute a key, hence specifying a key is a restriction for the database

2. Integrity Constraints

- e.g., in the Doctors[dep, did, lastname, firstname] relation, we can define the constraint: a doctor is uniquely identified by his / her department and did, i.e., 2 different doctors can either have the same department, or the same did, but not both; the key is the group of attributes **{dep, did}**
- e.g., Books[author, title, publisher, year]

author	title	publisher	year
C. J. Date	An Introduction to Database Systems	Addison-Wesley Publishing Comp.	2004
P. G. Wodehouse	Summer Lightning	Polirom	2003
Simon Singh	Fermat's Last Theorem	Humanitas	2012
Stephen Hawking	A Brief History of Time	Humanitas	2012

- a possible key is the group of attributes **{author, title, publisher, year}**; in such a case, the schema can be augmented to include another attribute (with distinct values; the latter can be automatically generated when records are added) ->

2. Integrity Constraints

bid	author	title	publisher	year
457	C. J. Date	An Introduction to Database Systems	Addison-Wesley Publishing Comp.	2004
4	P. G. Wodehouse	Summer Lightning	Polirom	2003
34	Simon Singh	Fermat's Last Theorem	Humanitas	2012
769	Stephen Hawking	A Brief History of Time	Humanitas	2012

- **{bid}** can now be chosen as the key; two different tuples will always have different values in the **bid** column
- the user must be careful not to define a key constraint that prevents the storage of correct tuple sets, e.g., if **{title}** is declared to be a key, the *Books* relation won't be allowed to contain tuples describing different books with the same title
- **superkey** - a set of fields that contains the key (e.g., the set **{bid, title}** is a superkey)

2. Integrity Constraints

bid	author	title	publisher	year
457	C. J. Date	An Introduction to Database Systems	Addison-Wesley Publishing Comp.	2004
4	P. G. Wodehouse	Summer Lightning	Polirom	2003
34	Simon Singh	Fermat's Last Theorem	Humanitas	2012
769	Stephen Hawking	A Brief History of Time	Humanitas	2012

- non-key fields can contain identical values in different tuples: the relation can contain 2 different books from the same year, i.e., 2 tuples with *year = 2004*; or 3 different books written by the same author, i.e., 3 tuples with *author = Stephen Hawking*, etc

2. Integrity Constraints

- a relation can have multiple keys: one key (an attribute or a group of attributes) is chosen as the **primary key**, while the others are considered **candidate keys**
- e.g., Schedule[day, hour, room, teacher, group, subject]

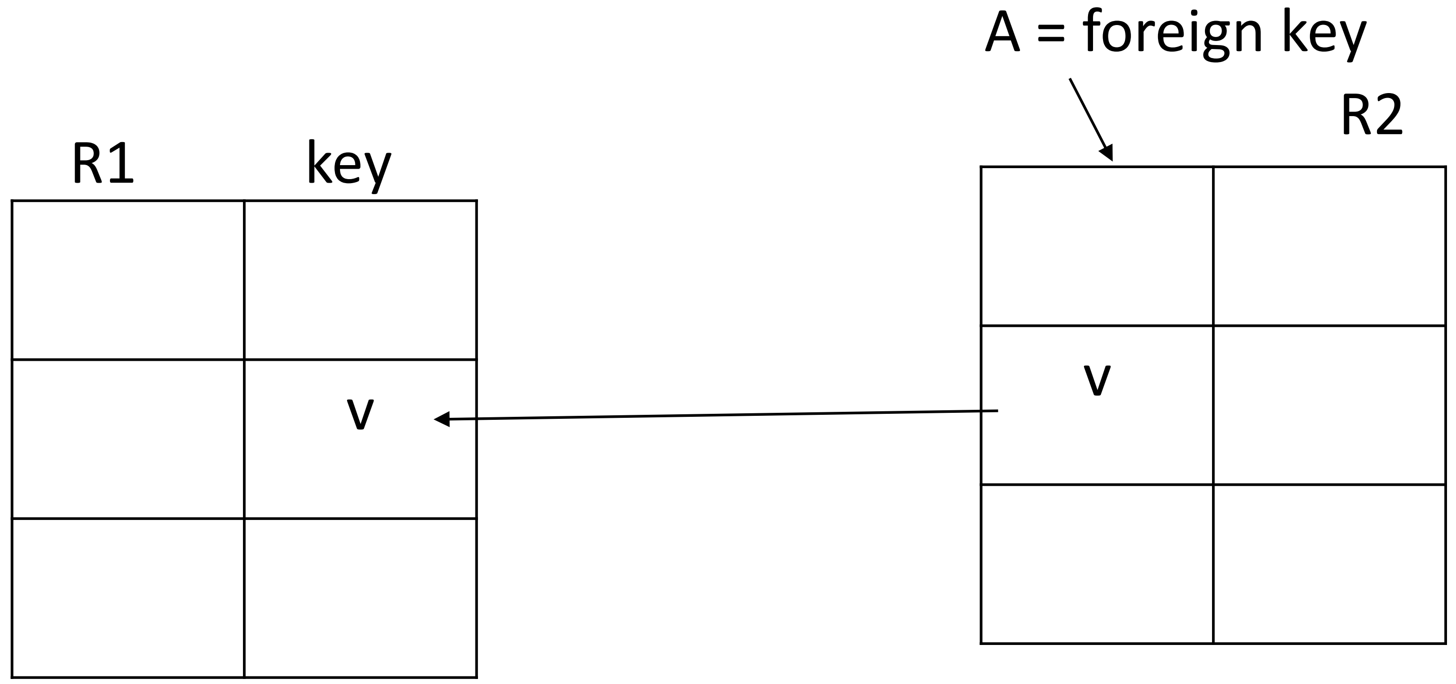
with the schedule for a week

- the following sets of attributes can be chosen as keys:

{day, hour, room}; {day, hour, teacher}; {day, hour, group}

2. Integrity Constraints

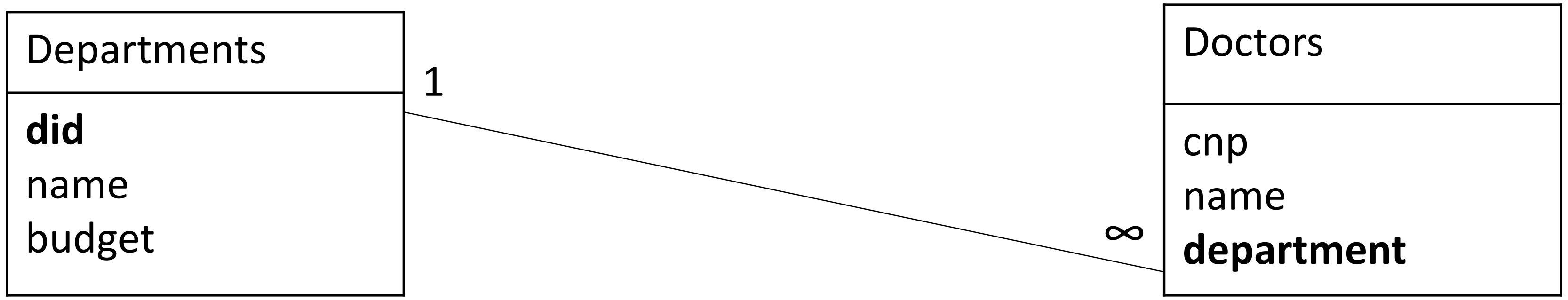
- **foreign key constraints** - the values of some attributes in a relation can also appear in another relation
- in the figure below, A is defined as a foreign key in R2; it refers to R1's key
- R1 and R2 need not be distinct



2. Integrity Constraints

- e.g., Departments[did, name, budget]

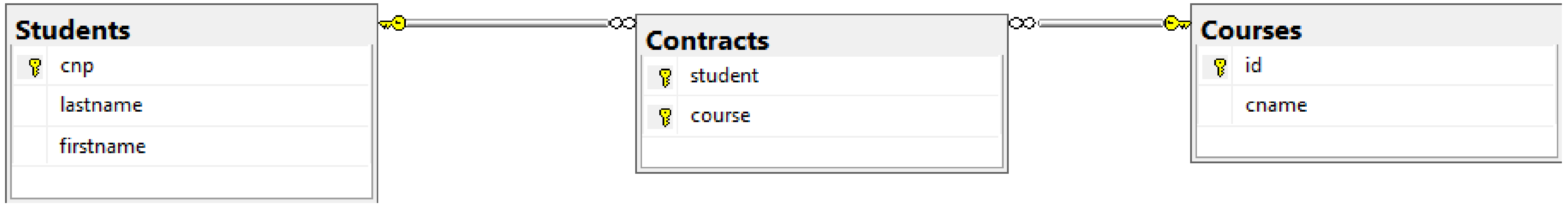
  Doctors[cnp, name, department]



- establishing a link betwen *Departments* (as the parent relation) and *Doctors* (child relation): **Departments.did = Doctors.department**
- a department stored in the *Departments* relation and identified through a code corresponds to all doctors with the department's code; the *department* attribute in the *Doctors* relation is a foreign key
- a foreign key can be used to store **1:n** associations among entities: a department can correspond to several doctors, and a doctor can be associated with at most one department

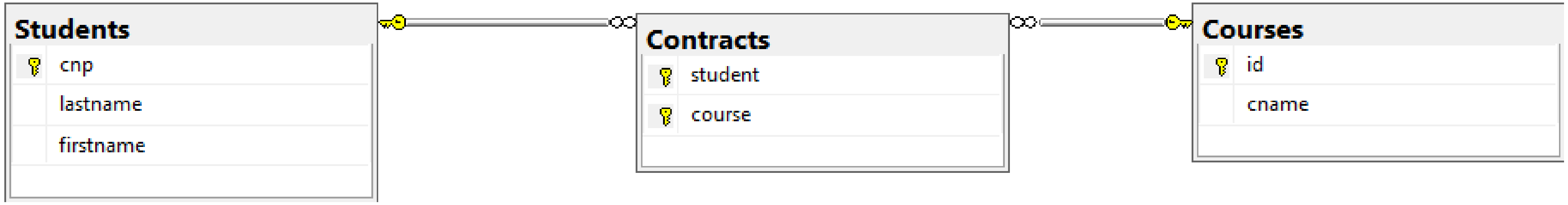
2. Integrity Constraints

- foreign keys can be used to store **m:n** associations among entities
- e.g., a student can take multiple courses, while a course can enroll multiple students; storing such associations requires a link table



- any value that appears in the *student* field of the *Contracts* relation must appear in the *cnp* field of the *Students* relation, but there may be students who haven't enrolled in any course yet (i.e., *cnp* values in *Students* that don't appear in the *student* column from *Contracts*)

2. Integrity Constraints



- the foreign key in the *Contracts* relation must have the same number of columns as the referenced key from *Students* (preferably, the primary key); the data types of the corresponding columns must be compatible, however, the column names can be different

2. Integrity Constraints

enforcing integrity constraints

- when the data is changed, the DBMS rejects operations that violate the specified integrity constraints
  - e.g., entering a row in *Contracts* that has no corresponding student row in *Students*
- in some cases, instead of rejecting the operation, the DBMS will make additional changes to the data, so in the end all the integrity constraints are satisfied
  - e.g., when removing a *Students* row that has referencing *Contracts* rows, the operation can be:
    - rejected – the row is not deleted
    - cascaded, i.e., the *Students* row and the referencing rows in *Contracts* are deleted

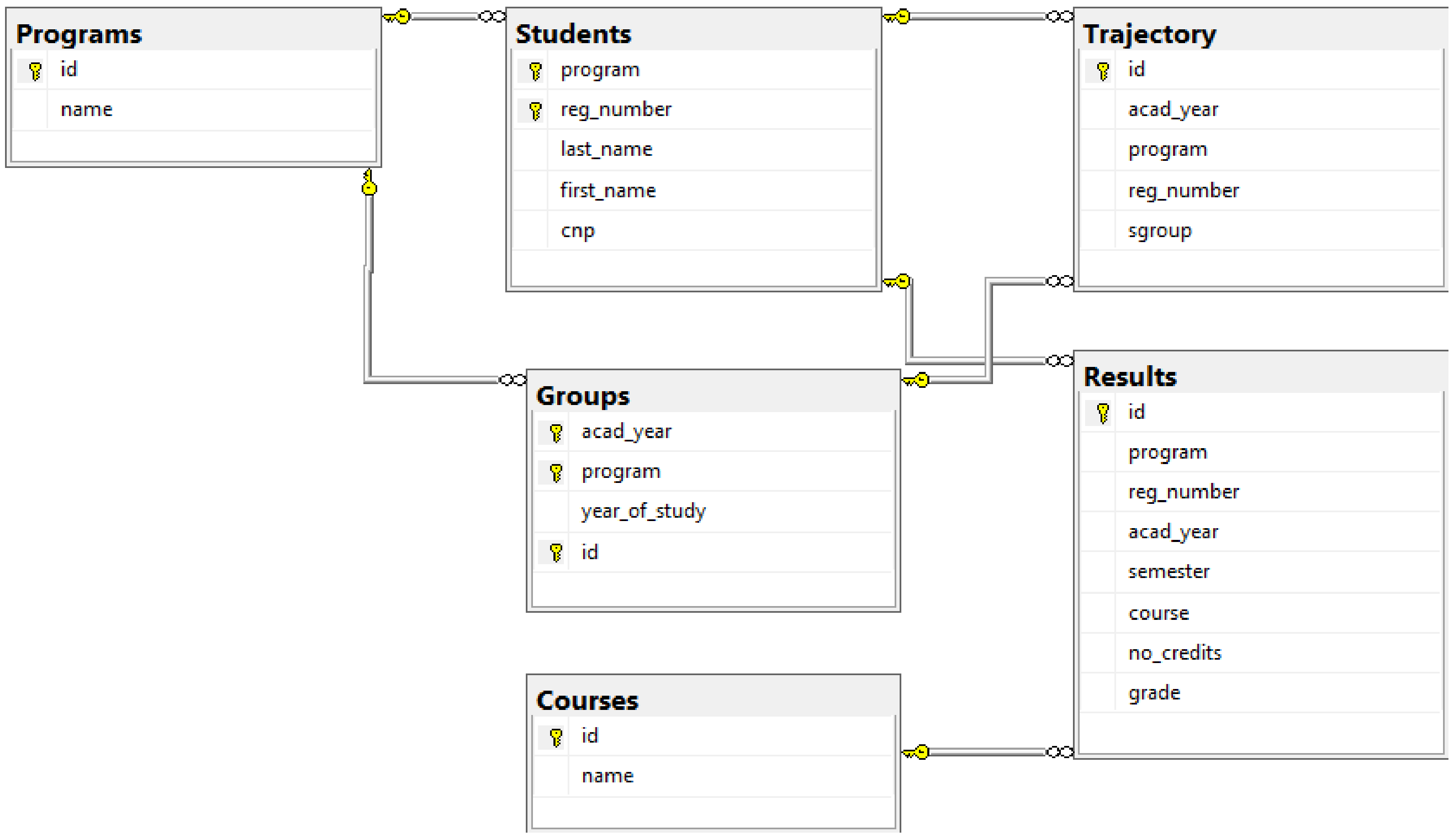
    * other options - see the CREATE TABLE statement in this lecture

3. Relational Databases

- **relational database**
  - collection of relations with distinct names
- **relational database schema**
  - collection of schemas for the relations in the database
- **database instance**
  - collection of relation instances, one / relational schema in the database schema
  - must be **legal**, i.e., it must satisfy all the specified integrity constraints

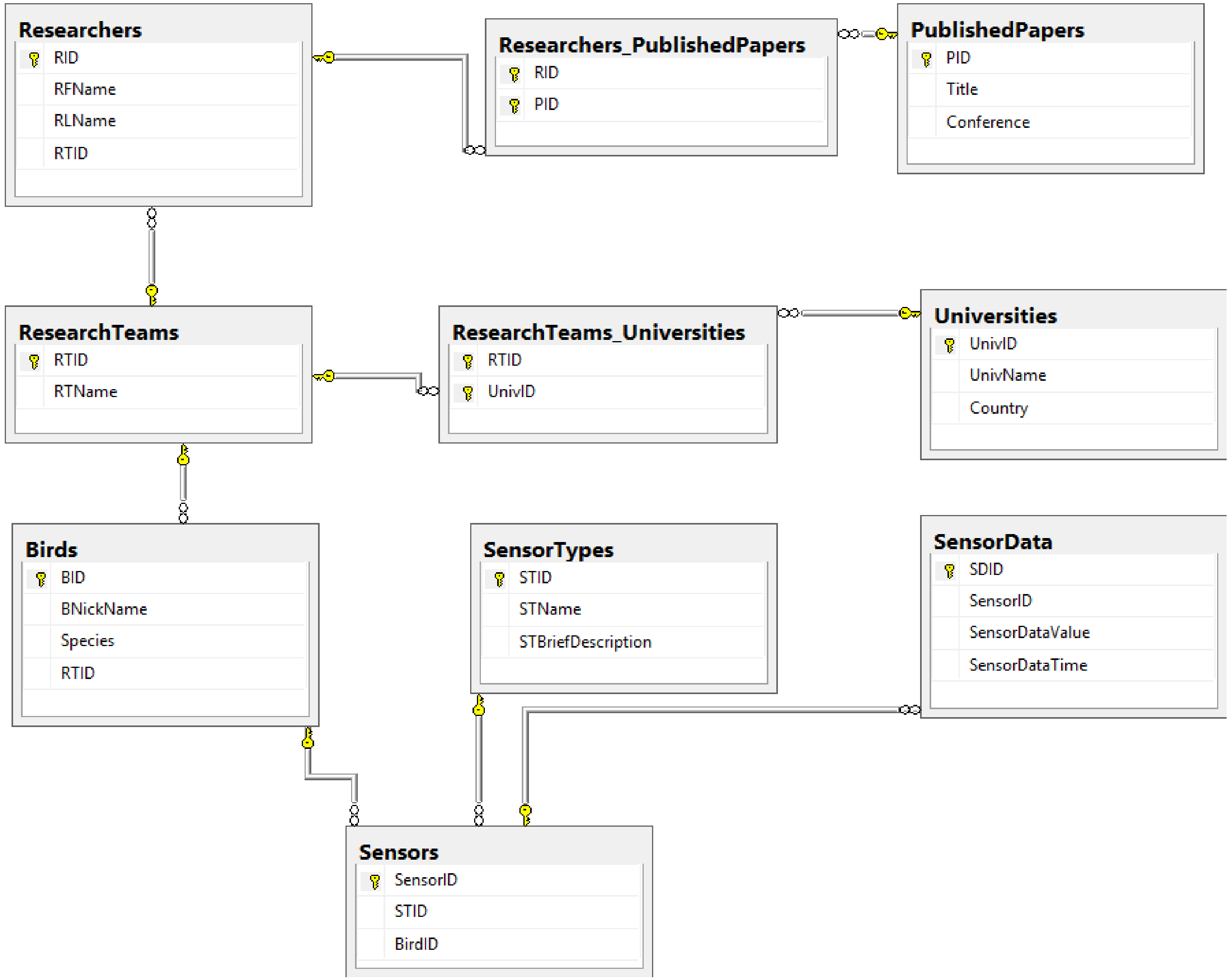
3. Relational Databases

- ex1:



3. Relational Databases

- ex2:



4. Managing Relational Databases with SQL

- **SQL** – Structured Query Language
- 1970 - E.F. Codd formalizes the relational model
- 1974 - the SEQUEL language (Structured English Query Language) is defined at IBM (in San Jose)
- 1976 - SEQUEL/2, a changed version of SEQUEL, is defined at IBM; following a revision, it becomes the SQL language
- 1986 - SQL adopted by ANSI (American National Standards Institute)
- 1987 - SQL adopted by ISO (International Standards Organization)
- versions (extensions): SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016

4. Managing Relational Databases with SQL

- defining components
  - CREATE, ALTER, DROP (SQL DDL)
- managing and retrieving data
  - SELECT, INSERT, UPDATE, DELETE (SQL DML)
- managing transactions
  - START TRANSACTION, COMMIT, ROLLBACK

4. Managing Relational Databases with SQL

- CREATE TABLE – defines a new table

CREATE TABLE table_name

(column_definition [, column_definition] ... [, table_restrictions])

column_definition: column_name data_type[(length)] [**DEFAULT value**]
[column_restriction]

```
CREATE TABLE Students
     (sid INT PRIMARY KEY,
      cnp CHAR(13),
      lastname VARCHAR(50),
      firstname VARCHAR(50) DEFAULT 'TBA',
      age INT CHECK (age >= 18))
```

- data type categories: numeric, string, date-time, etc

* we use the *age* attribute for simplicity; it is preferable to store the date of birth, as it doesn't change every year

4. Managing Relational Databases with SQL

- **restrictions associated with a column**:
  - NOT NULL - can't have undefined values
  - PRIMARY KEY - the column is a primary key
  - UNIQUE - the values in the column are unique
  - CHECK(condition) - the condition that must be satisfied by the column's values (simple conditions that evaluate to *true* or *false*)
  - FOREIGN KEY REFERENCES parent_table[(column_name)] [ON UPDATE action] [ON DELETE action]

4. Managing Relational Databases with SQL

- **restrictions associated with a table**:
  - PRIMARY KEY(column_list) - defining the primary key for the table
  - UNIQUE(column_list) - the values in the column list are unique
  - CHECK(condition) - the condition that must be satisfied by a row
  - FOREIGN KEY(column_list) REFERENCES parent_table[(column_list)][ON UPDATE action][ON DELETE action]

4. Managing Relational Databases with SQL

- **possible actions for a foreign key:**
  - NO ACTION
    - the operation is not allowed if it violates integrity constraints
  - SET NULL
    - the foreign key value is set to *null*
  - SET DEFAULT
    - the foreign key value is set to the default value
  - CASCADE
    - the delete / update is performed on the parent table, but it generates corresponding deletes / updates in the child table

```sql
CREATE TABLE Programs
  (id SMALLINT PRIMARY KEY,
  name VARCHAR(70))

CREATE TABLE Students
  (program SMALLINT REFERENCES Programs(id),
  reg_number CHAR(10),
  last_name CHAR(30),
  first_name CHAR(30),
  cnp CHAR(13) UNIQUE,
  PRIMARY KEY(program, reg_number))

CREATE TABLE Groups
  (acad_year SMALLINT,
  program SMALLINT REFERENCES Programs(id),
  year_of_study SMALLINT,
  id CHAR(10),
  PRIMARY KEY (acad_year, program, id))

CREATE TABLE Courses
  (id CHAR(10) PRIMARY KEY,
  name VARCHAR(70))
```

```sql
CREATE TABLE Trajectory
  (id INT PRIMARY KEY IDENTITY(1,1),
  acad_year SMALLINT,
  program SMALLINT,
  reg_number CHAR(10),
  sgroup CHAR(10),
  FOREIGN KEY(program, reg_number) REFERENCES
Students(program, reg_number),
  FOREIGN KEY(acad_year, program, sgroup)
REFERENCES Groups(acad_year, program, id))

CREATE TABLE Results
  (id INT PRIMARY KEY IDENTITY(1,1),
  program SMALLINT,
  reg_number CHAR(10),
  acad_year SMALLINT,
  semester SMALLINT,
  course CHAR(10) REFERENCES Courses(id),
  no_credits DECIMAL(3,1),
  grade SMALLINT,
  FOREIGN KEY(program, reg_number) REFERENCES
Students(program, reg_number))
```

```sql
CREATE TABLE Universities
  (UnivID SMALLINT PRIMARY KEY IDENTITY(1,1),
  UnivName NVARCHAR(100) NOT NULL,
  Country NVARCHAR(40))

CREATE TABLE ResearchTeams
  (RTID SMALLINT IDENTITY(1,1),
  RTName NVARCHAR(100) UNIQUE,
  CONSTRAINT PK_ResearchTeams PRIMARY KEY(RTID))

CREATE TABLE ResearchTeams_Universities
  (RTID SMALLINT FOREIGN KEY REFERENCES
ResearchTeams(RTID) ,
  UnivID SMALLINT REFERENCES Universities(UnivID),
  PRIMARY KEY(RTID, UnivID))

CREATE TABLE SensorTypes
  (STID TINYINT PRIMARY KEY IDENTITY(1,1),
  STName NVARCHAR(50) UNIQUE,
  STBriefDescription NVARCHAR(300) DEFAULT 'TBW')

CREATE TABLE Birds
  (BID INT PRIMARY KEY IDENTITY(1,1),
  BNickName NVARCHAR(50),
  Species VARCHAR(100),
  RTID SMALLINT,
  CONSTRAINT FK_Birds_ResearchTeams FOREIGN KEY(RTID)
REFERENCES ResearchTeams(RTID))

CREATE TABLE Sensors
  (SensorID INT PRIMARY KEY IDENTITY(1,1),
  STID TINYINT REFERENCES SensorTypes ON DELETE NO ACTION
ON UPDATE CASCADE,
  BirdID INT REFERENCES Birds(BID))

CREATE TABLE SensorData
  (SDID INT PRIMARY KEY IDENTITY(1,1),
  SensorID INT REFERENCES Sensors(SensorID),
  SensorDataValue REAL,
  SensorDataTime DATETIME2)

CREATE TABLE Researchers
  (RID SMALLINT PRIMARY KEY IDENTITY(1,1),
  RFName NVARCHAR(90) NOT NULL,
  RLName NVARCHAR(90) NOT NULL,
  RTID SMALLINT REFERENCES ResearchTeams(RTID))

CREATE TABLE PublishedPapers
  (PID INT PRIMARY KEY IDENTITY(1,1),
  Title NVARCHAR(200),
  Conference NVARCHAR(200))

CREATE TABLE Researchers_PublishedPapers
  (RID SMALLINT REFERENCES Researchers(RID) ,
  PID INT REFERENCES PublishedPapers(PID),
  PRIMARY KEY(RID, PID))
```

4. Managing Relational Databases with SQL

- ALTER TABLE – changes the structure of a defined table

  ALTER TABLE table_name operation

```
ALTER TABLE Students
ADD FavSymphony VARCHAR(50)
```

- possible operations (differences among DBMSs)
  - add / change / remove a column
    - ADD column_definition
    - {ALTER COLUMN | MODIFY} column_definition
    - DROP COLUMN column_name

4. Managing Relational Databases with SQL

- add / remove a constraint
  - ADD [CONSTRAINT constraint_name] PRIMARY KEY(column_list)
  - ADD [CONSTRAINT constraint_name] UNIQUE(column_list)
  - ADD [CONSTRAINT constraint_name] FOREIGN KEY (column_list) REFERENCES table_name[(column_list)] [ON UPDATE action] [ON DELETE action]
  - DROP [CONSTRAINT] constraint_name

4. Managing Relational Databases with SQL

- DROP TABLE – removes a table

  DROP TABLE table_name

```
DROP TABLE Students
```

- the Data Definition Language (DDL) is the subset of SQL used to create / remove / change various components (e.g., tables)

4. Managing Relational Databases with SQL

- changing data in a table
- the INSERT command – adding records

```
INSERT INTO table_name[(column_list)] VALUES (value_list)
INSERT INTO table_name[(column_list)] subquery,
```

where *subquery* refers to a set of records (generated with the SELECT statement)

```
INSERT INTO Students (sid, cnp, lastname, firstname, age)
 VALUES (1, '123456789012', 'Popescu', 'Maria', 20)
```

4. Managing Relational Databases with SQL

- changing data in a table
- the UPDATE command – changing records

```
UPDATE table_name
SET column_name=expression [, column_name=expression] ...
[WHERE condition]
```

- the command changes the records in the table that satisfy the condition in the WHERE clause; if the WHERE clause is omitted, all the records in the table are changed; the values of the columns specified in SET are changed to the associated expressions' values

```
UPDATE Students
SET age = age + 1
WHERE cnp = '123456789012'
```

4. Managing Relational Databases with SQL

- changing data in a table
- the DELETE command – removing records

```
DELETE FROM table_name
[WHERE condition]
```

- the command deletes the records in the table that satisfy the condition in the WHERE clause; if the WHERE clause is omitted, all the table's records are removed

```
DELETE
FROM Students
WHERE lastname = 'Popescu'
```

4. Managing Relational Databases with SQL

- **filter conditions**
  - expression relational_operator expression
  - expression [NOT] BETWEEN valmin AND valmax
  - expression [NOT] LIKE pattern ("%" - any substring, "_" - one character)
  - expression IS [NOT] NULL
  - expression [NOT] IN (value [, value] ...)
  - expression [NOT] IN (subquery)
  - expression relational_operator {ALL | ANY} (subquery)
  - [NOT] EXISTS (subquery)

4. Managing Relational Databases with SQL

- **filter conditions**
  - elementary condition (previously described)
  - (condition)
  - NOT condition
  - condition_1 AND condition_2
  - condition_1 OR condition_2

4. Managing Relational Databases with SQL

- 3-valued logic (truth values: *true*, *false*, *unknown*)

	TRUE	FALSE	NULL
NOT	FALSE	TRUE	NULL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

References

- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (2nd Edition), McGraw-Hill, 2000
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008
- [Ha96] HANSEN, G., HANSEN, J., Database Management And Design (2nd Edition), Prentice Hall, 1996
- [Ra07] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, McGraw-Hill, 2007, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Si10] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, McGraw-Hill, 2010, http://codex.cs.yale.edu/avi/db-book/
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems, http://infolab.stanford.edu/~ullman/fcdb.html