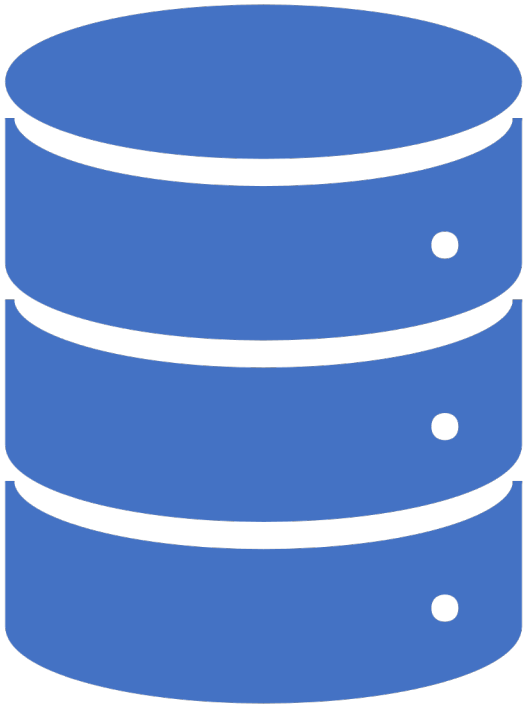


Lecture 2

The Relational Model

Data Models

- Entity-Relationship (1990s)
- Relational (1970s) (1NF)
- Network (1965)
- Hierarchical (1965)
- Object-Oriented
- noSQL
- Semistructured (XML) (1990s)

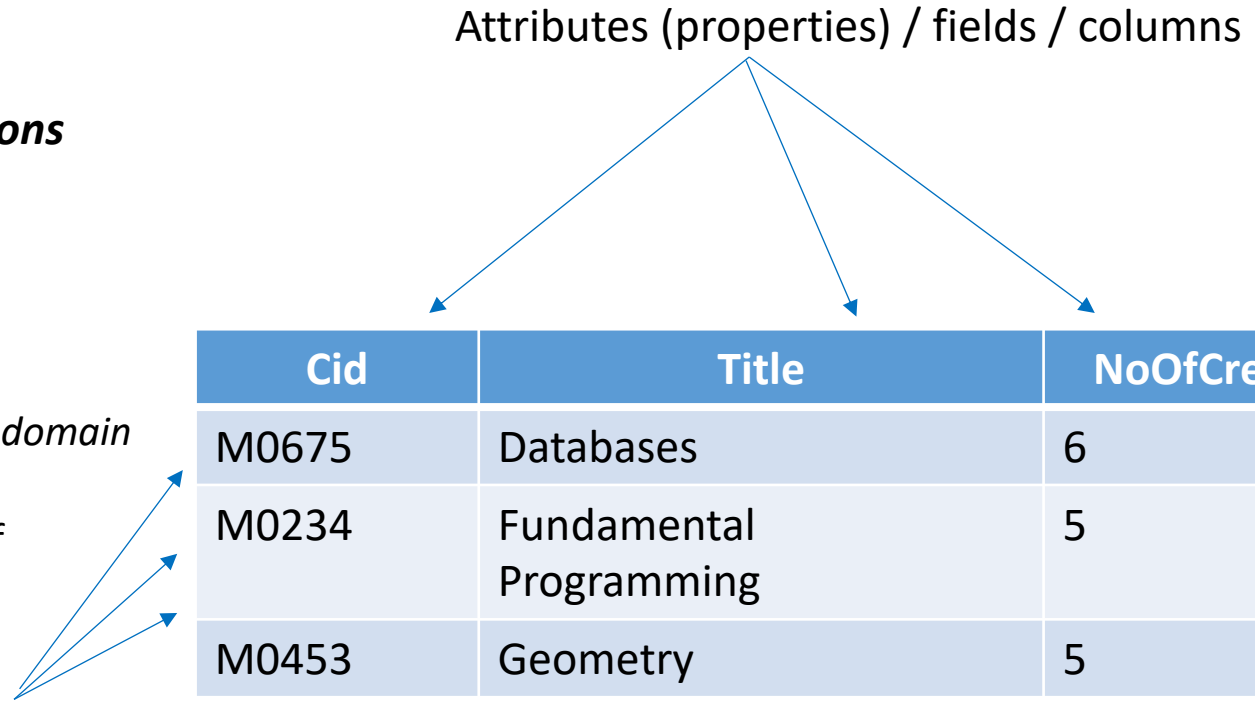
Relational Model

- Introduced by Edgar Codd in 1970
- The most used nowadays, can be used by anyone
- It is represented by DBMSs: SQL Server, Oracle, …
- Has ***a simple data representation***
- The queries can be expressed in a simple and high-level language
- A simple data structure: ***table***
  - Easy to understand
  - Usefull in the real word
  - Low level quering
- Use mathematics for describing and represent the records: ***relation***
  - Use a formal model and allows the formal quering
  - The properties can be proved mathematical

Relational Model

Relation

- A model is used for a data collection
- The data collection is organized in a set of ***relations*** (tables)
- A relation has a *relation* ***schema*** and a *relation* ***instance***
  - Relation schema: has the table's column head;
    - contain the *relation's name*, the *name and domain* for each field of the relation's
    - the domain of a field has name and a set of associated values
  - Relation instance: represented as a table
- Example: schema for the Courses relation
  - Courses[Cid, Title, NoOfCredits] or
  - Courses(Cid: char(5), Title: string, NoOfCredits: integer)
- Example: instance for the Courses relation



Cid	Title	NoOfCredits
M0675	Databases	6
M0234	Fundamental Programming	5
M0453	Geometry	5

Relational Model

Relation

- Relation = a set of attributes **$[A_1, A_2, A_3, \ldots, A_n]$**
- Domain of possible values for an attribute A_i = set of scalar values (integer, string, date, …)
  - **$D_i = Dom(A_i) \cup \{null\}$**, where i=1..n and {null} is the undefined value used to check if a value has been assigned to an attribute A_i or not
- Instance of a relation = [R] = a subset of $D_1 \times D_2 \times \ldots \times D_n$
- Degree (arity) = the number of all the attributes from a relation
  - Relation of degree / arity n: $R \subseteq D_1 \times D_2 \times \ldots \times D_n$
- Tuple = an element of the instance of a relation = a record
  - All the tuples of a relation must be distinct
- Cardinality = the number of tuples of a relation

Relational Model

Relation

- Let $R[A_1, A_2, A_3, \ldots, A_n]$ the relation schema – can be kept in a table with the following form

R	A_1	A_2	…	A_n
r_1	a_{11}	a_{12}	…	a_{1n}
r_2	a_{21}	a_{22}	…	a_{2n}
…	…	…	a_{ij}	…
r_m	a_{m1}	a_{m1}	…	a_{mn}

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

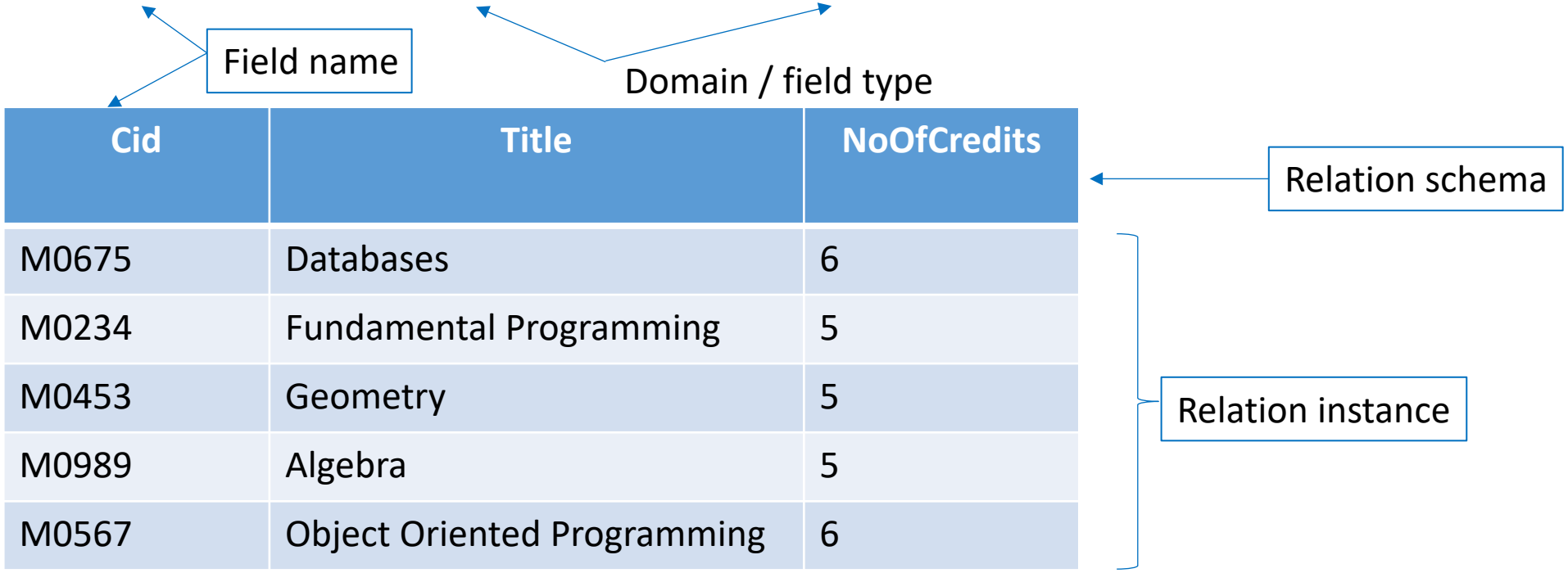
- the ***values*** of an ***attribute*** are ***scalar*** and ***atomic*** (An **atomic** value cannot be broken down into smaller pieces.)
- the ***rows*** from a table are ***not ordered***
- the tuples / records from a table are ***distinct***

Atomic e.g.: Dan
Atomic e.g.: Popescu
NOT atomic: Dan Popescu

Relational Model

Relation example:

Courses(Cid: char(5), Title: string, NoOfCredits: integer)



Cid	Title	NoOfCredits
M0675	Databases	6
M0234	Fundamental Programming	5
M0453	Geometry	5
M0989	Algebra	5
M0567	Object Oriented Programming	6

Cardinality = 5, Degree = 3

The tuples are distinct

Relational Model

Relation bad example:

Students(Sid: integer, Name: string, Surname: string, Group: integer, StudyYear:integer)

Sid	Name	Surname	Group	StudyYear
1	Avram	Paul	821	2
2	Cristea	Dan	821	3
3	Danila	Mihaela	822	2
4	Vasilescu	Dana	822	2
5	Hora	Lucian	822	2

- The pair (Group, StudyYear)= (822, 2) is stored 3 times, for the latest 3 students.

- The first 2 students are in the same group, but in different study years.

To avoid these situations, the relation must be ***normalized***.

Relational Databases

Database = set of relations (tables) with relationships between them used to retrieve useful information to a potential user

Structure of a database = set of the structures of the database relations

Instance of a database = set of the instances of the database relations

Graphical representation of the relations:

- Students(Sid:integer, Name:string, Surname:string, Group: integer, StudyYear:integer)
- Courses(Cid: char(5), Title: string, NoOfCredits: integer)
- Participations(Sid:integer, Cid:char(5), Pdate:date)

Students
Sid
Name
Surname
Group
StudyYear

Participations
Sid
Cid
Pdate

Courses
Cid
Title
NoOfCredits

Integrity constraints

- **Integrity constraints** = restrictions = specified conditions on the database schema that must be fulfilled by each instance of the database
  - are specified when the relation is defined
  - are checked when the relation is modified
  - operations that violate the integrity constraints are not allowed
    - e.g. introducing 2 identical values for the CourseId, or for the StudentId (identification number)
    - e.g. entering wrong type values in a column (on an integer position is introduced a string)
- The integrity constraints are based on the facts from the real world / models.
- An instance of a relation is correct if satisfies all the integrity constraints specified. DBMS does not allow the integrity constraints that are not satisfied.
- Examples: **domain** constraints, **key** constraints, **foreign key** constraints

Integrity constraints

Domain constraints - some conditions that must be satisfied by every relation instance: the values of each column should belong to the associated domain

Examples:

- Students(**Sid**:integer, **Name**:string, **Surname**:string, **Group**: integer, **StudyYear**:integer)
  - Domain constraint: **Group**: integer, **StudyYear**:integer, …
  - Range constraint: 1≤**StudyYea**r≤3

- Grades(**Gid**:integer, **FinalGrade**:integer, **LabGrade**:integer, **PracticalTestGrade**:integer, **WrittenTestGrade**:integer)
  - FinalGrade = LabGrade + PracticalTestGrade + WrittenTestGrade – **NOT** an integrity constraint
    - can be managed through code (stored procedures, …)

Integrity constraints

Key constraints - to ensure that a *subset of attributes* in a relation is *unique idientified* for every tuple in the relation; this subset should be minimal

- Each tuple must have different values in all the fields that are part of the key (no identical values are allowed for the fields that are part of the key)
- $C \subseteq [A_1, A_2, A_3, \ldots, A_n]$ is a **key** for relation $R[A_1, A_2, A_3, \ldots, A_n]$ if
  a. the attributes in C are used to identify each record in R (or, there are no 2 tuples that have the same value for all the attributes)
  b. no subset of C has this property
- **Super key** - a set of fields that contain the key (or, the upper point b. is not fulfilled)
- **Candidate key** – if there are multiple keys for a relation
- **Primary key** - one of the candidate key is selected

- Example: Let Professor[Pid, Did, Name, Surname, Salary] be a relation. The key is the groups of attributes **(Pid, Did)** due to the constraint: a professor is uniquely identified by his / her Pid and Did (department id), or, 2 different professors can have the same department or Did, but different Pid.

Integrity constraints

Key constraints

- *Example:* PhotoCamera[Brand, Model, Year]

Brand	Model	Year
Nikon	D3500	2019
Nikon	Digital Coolpix B500	2018
Sony	DSC-H300	2016
Sony	Alpha A7	2020

PCid	Brand	Model	Year
12	Nikon	D3500	2019
23	Nikon	Digital Coolpix B500	2018
45	Sony	DSC-H300	2016
36	Sony	Alpha A7	2020

- A possible key can be the group of attributes **(Brand, Model, Year)**, case in which can be included another attribute, with distinct values →

- The key can be **PCid**, because 2 different tuples will always have different values in the PCid column

Try not to define a key constraint that prevents the storage of correct tuple sets

- e.g. if **(Model)** is the key, then the *PhotoCamera* relation cannot contain tuples that describe different PhotoCameras with the same model

Superkey: **(Pcid, Model)**

Non-key fields: contain identical values in different tuples

- e.g. 2 different PhotoCameras from the same brand or from the same year

Integrity constraints

Key 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 ***candidates keys***
- *Example:* Program[Day, OpenHour, CloseHour, Room, Employee, EmployeeGroup, PrincipalTask]
  - the following sets of attributes can be chosen as keys:

    (Day, OpenHour, CloseHour, Room)

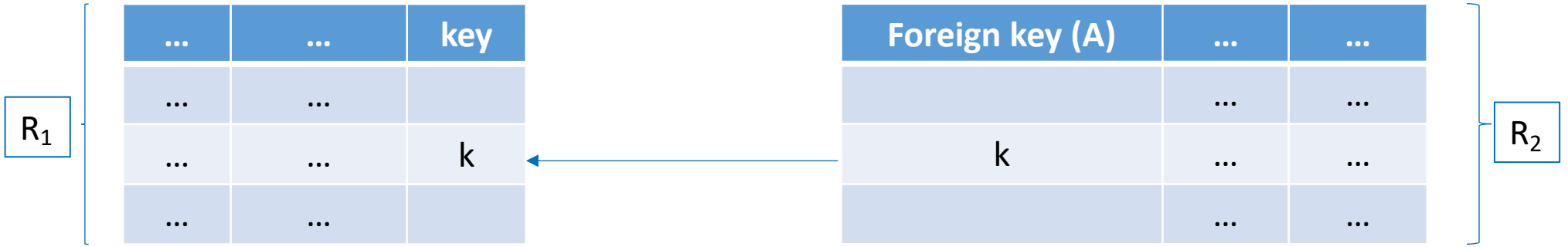
    (Day, OpenHour, CloseHour, Employee)

    (Day, OpenHour, EmployeeGroup)

Integrity constraints

Foreign key constraints

- **Foreign key** = a field of a relation used to reference a tuple from another relation
- The foreign key corresponds to the primary key from another relation
- A is the foreign key from relation R_2, A refers to the key of relation R_1, where R_1 and R_2 need not to be distinct
- The key and the foreign key will have the same data type and the same values and will not be NULL



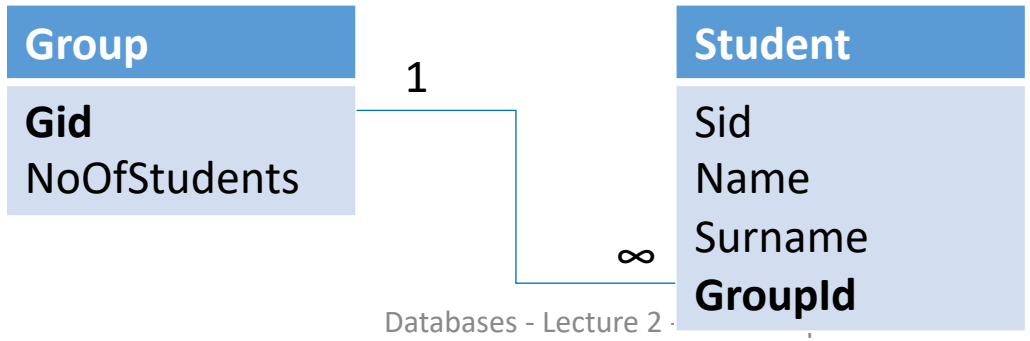
Integrity constraints

Foreign key constraints

- Example: Group[Gid, NoOfStudents] – *parent table* – Gid is the primary key

  Student[Sid, Name, Surname, GroupId] – *child table* – GroupId is the foreign key

- The relation (link) between the *Group* table (parent table) and the *Student* table (child table) involves the ***primary key from the parent table*** and the ***foreign key from the child table***: Group.Gid=Student.GroupId
- Relation ***1:n*** reflect the relation ***parent table:child table***; parent table has the primary key and child table has the foreign key; parent table has to be created first and child table after.
  - Primary key = the attribute that has distinct values for each record / tuple and it is not null
  - Foreign key = the attribute that points to the primary key from another table and has the same data type and the same values as the primary key and it is not null; the column names for the primary key and for the foreign key can be different.
  - Relation 1:n is established between the primary key and the corresponding foreign key from the 2 tables considered in the relation



A foreign key can store ***1:n*** associations between the tables: in a group there are several students and a student is from just one group.
A group, that is identified by the Gid, from the Group table, corresponds to all the students with the same group Gid; the *GroupId* attribute from the Student table is a foreign key.

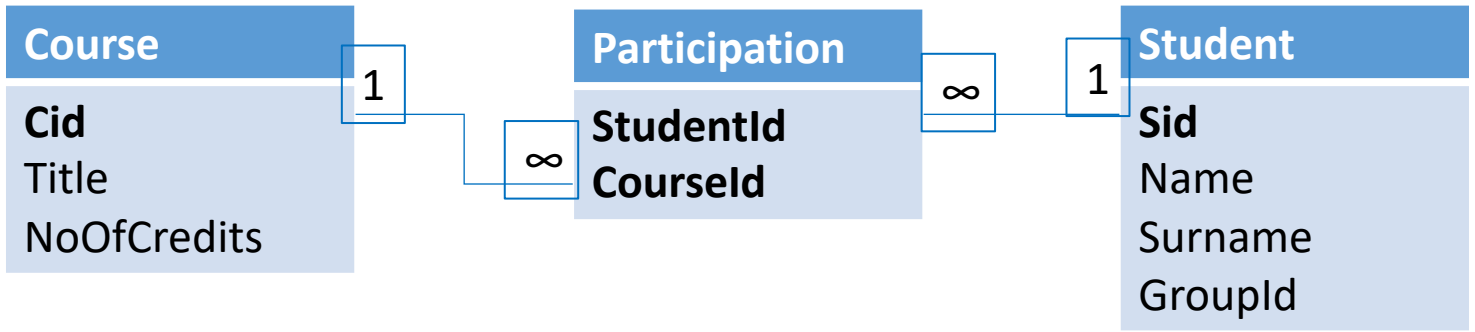
Integrity constraints

Foreign key constraints

- Example: relation ***m:n*** that use the foreign keys to obtain the associations between the entities

Course[Cid, Title, NoOfCredits]– Cid is the primary key
Student[Sid, Name, Surname, GroupId]– Sid is the primary key
Participation[CourseId, StudentId]

- A student participate in several courses; in a course participate several students; such associations requires an aditional table (Participation)
- Any value from the *StudentId* from the *Participation* table must exist in the *Sid* field from the *Student* table; but there may be students who haven't participated in any course yet (*Sid* from the *Student* table that don't appear in the *StudentId* column from the *Participation* table)
- Any value from the *CourseId* from the *Participation* table must exist in the *Cid* field from the *Course* table; but there may be courses without participations (*Cid* from the *Course* table that don't appear in the *CourseId* column from the *Participation* table)



Integrity constraints

Enforcing integrity constraints

- DBMS rejects the operations that violate the existing integrity constraints, when data is changed
  - e.g. entering a tuple in *Participation* table that doesn't has a corresponding student row in *Student* table
- Sometimes, DBMS can perform additional changes to the data, such that the integrity constraints be satisfied.
  - e.g. removing a student row, from the *Student* table, that has a reference in *Participation* table, may cause the action:
    - rejected: the row is not deleted
    - cascaded: the row from the *Student* table is deleted and also the corresponding rows from the *Participation* table
- Example: CourseId M05287 from Participation table doesn't exist as a primary key in Course table, column Cid

Course

Cid	Title	NoOfCredits
M03483	Databases	6
M05286	Geometry	5

Participation

CourseId	StudentId
M03483	1
M05286	2
M05287	1
M05286	4

Integrity constraints

Enforcing integrity constraints

- Example: StudentId 4 from Participation table doesn't exist as a primary key in Student table, column Sid

Participation

CourseId	StudentId
M03483	1
M05286	2
M05286	4

Student

Sid	Name	Surname	GroupId
1	Popescu	Alin	821
2	Vestea	Daniela	822

- Example: m:n

Course

Cid	Title	NoOfCredits
M03483	Databases	6
M05286	Geometry	5

Participation

CourseId	StudentId
M03483	1
M05286	2
M05287	1
M05286	4

Student

Sid	Name	Surname	GroupId
1	Popescu	Alin	821
2	Vestea	Daniela	822

Integrity constraints

Enforcing integrity constraints

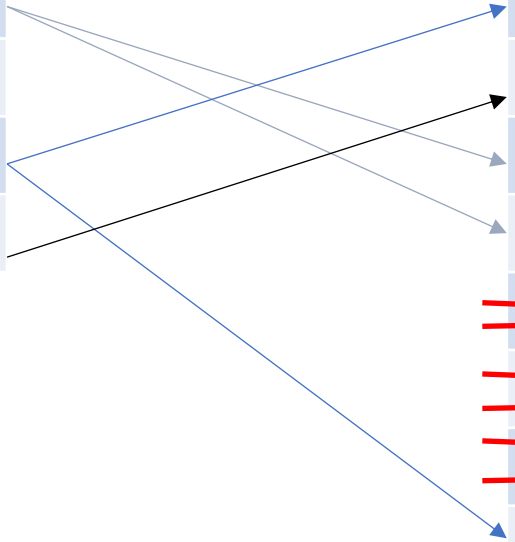
- Example: GroupId 227, null (foreign key) from Student table doesn't exist as a primary key in Group table, column Gid (primary key)
- No student associated in group 822; multiple students associated to group 821 and 923; one student associated to group 924

Group

Gid	NoOfStudents
821	30
822	31
923	27
924	28

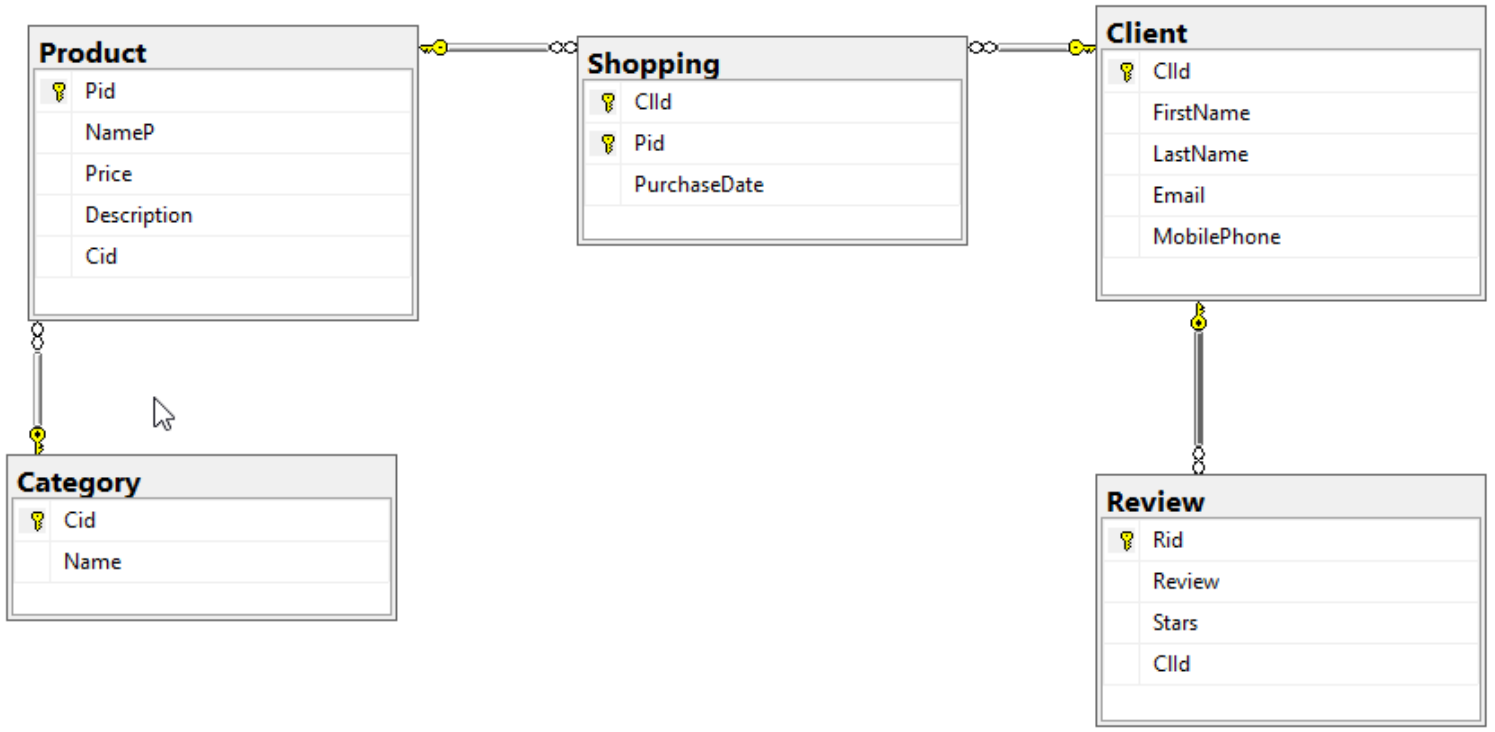
Student

Sid	Name	Surname	GroupId
1	Almasan	Alina	923
2	Florea	Cristina	924
3	Cisteian	Paula	821
4	Hora	Mihai	821
5	Mihali	Dan	227
6	Lupea	Ionut	227
7	Crisan	Olivia	NULL
8	Oprescu	Paul	923



Relational Databases

- ***Relational database*** – a set of relations that have distinct names
- ***Relational database schema*** – a set of schemas of the relations from the database
- ***Database instance*** – a set of relation instances that have to be correct (e.g. it must satisfy all the specified integrity constraints)
- Example: database PhotoCamera



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 introduced at IBM, in San Jose
- 1976 – SEQUEL receives changes and becomes SEQUEL/2, also defined at IBM; after another revision, SEQUEL/2 becomes the SQL language
- 1986 – SQL is adopted by the ANSI (American National Standards Institute)
- 1987 – SQL is adopted by ISO (International Standards Organisation)
- Extensions / versions of SQL: SQL-86, SQL-89 (with minor revision), SQL-92 (with major revision – 1120 pages), SQL-1999 (major extensions – 2084 pages), SQL-2003 (SQL / XML sections – 3606 pages), SQL-2006, SQL-2008, SQL-2011, SQL-2016

Relational Databases with SQL

- **DBMS languages:**
- ***Data Definition Language (DDL)***
  - Refers to defining components
  - Define the conceptual structure
  - Define and describe the integrity constraints
  - Can inluence the physical structure in some DBMSs
  - CREATE, ALTER, DROP (Create, modify and delete databases, tables and views) – reflect the structure
- ***Data Manipulation Language (DML)***
  - Refers to managing and retriving the data
  - Contain operations aplied to the database instances
  - INSERT, UPDATE, DELETE – insert, modify and delete records (from the tables)
  - SELECT – query the database
- Manage the transactions: BEGIN TRANSACTION, COMMIT, ROLLBACK
- Host languages: Programming languages that allows the users to include DML commands in their source code
- Control the access to the database, tables and view – by inserting, modifying and deleting access rights

Relational Databases with SQL

- ***Data Definition Language (DDL)***
- **CREATE DATABASE** – allows to create a database
  - ***CREATE DATABASE database_name***
    - e.g. CREATE DATABASE Faculty;

- **ALTER DATABASE** – allows to modify a database
  - ***ALTER DATABASE database_name***
    MODIFY NAME = new_database_name |
    COLLATE collate_type |
    SET option_specification [WITH termination]
    - e.g. ALTER DATABASE Faculty
      MODIFY Name = CSFaculty;

- **DROP DATABASE** – allows to drop a database
  - ***DROP DATABASE database_name***
    - e.g. DROP DATABASE CSFaculty;

Relational Databases with SQL

- ***Data Definition Language (DDL)***

CREATE TABLE – allows to create a table

CREATE TABLE table_name(

column_name_1 data_type[(length)] [DEFAULT value] [column_restriction],

[column_name_2 data_type[(length)] [DEFAULT value] [column_restriction],

…

column_name_n data_type[(length)] [DEFAULT value] [column_restriction]]

[,table_restriction])

Where, ***column_restriction*** can specify a primary key constraint, a unique constraint, a check constraint, a not null value, …

e.g. CREATE TABLE Student (
Sid INT PRIMARY KEY,
Name VARCHAR(50) NOT NULL,
Surname VARCHAR(50),
Dob DATE DEFAULT '01/01/2000',
Login CHAR(10),
CityFaculty VARCHAR CHECK (CityFaculty='Cluj Napoca' OR CityFaculty='Brasov'))

Relational Databases with SQL

- ***Data Definition Language (DDL)***

CREATE TABLE – allows to create a table

Data types:

- Exact numerics: bigint, bit, decimal, int, smallint, tinyint, …
- Approximate numerics: real, float
- Date and time: date, datetime, time, …
- Character strings: char, varchar, text
- Unicode character strings: nchar, nvarchar, ntext
- Binary strings: binary, varbinary, image
- Others: cursor, rowversion, xml, table, …

* For all the details, please check SQL data types on Microsoft website (and not only)

Relational Databases with SQL

- ***Data Definition Language (DDL)***

CREATE TABLE – allows to create a table

- ***Restrictions associated with a column***
  - ***NOT NULL*** – must exact a value (no undefined values are allowed)
  - ***PRIMARY KEY*** – the column is a primary key
  - ***UNIQUE*** – the values in the column are unique
  - ***CHECK(condition)*** – the condition has to be satisfied by the column's values (it is evaluated with *true or false*)
  - ***FOREIGN KEY REFERENCES parent_table(primary_key_column_name) [ON UPDATE action] [ON DELETE action]*** – the column is a foreign key

- ***Restrictions associated with a table***
  - ***PRIMARY KEY(column_list)*** – the primary key for the table
  - ***UNIQUE(column_list)*** – the values in the column_list are unique
  - ***CHECK(condition)*** – the condition has to be satisfied by a row
  - ***FOREIGN KEY(column_list) REFERENCES parent_table(primary_key_column_list) [ON UPDATE action] [ON DELETE action]*** – the foreign key for the table

Relational Databases with SQL

- ***Data Definition Language (DDL)***

CREATE TABLE – allows to create a table

- ***Actions for foreign key***
  - ***NO ACTION*** – the operation cannot be performed if the integrity constraints are violated
  - ***SET NULL*** – the foreign key value is set to be *null*
  - ***SET DEFAULT*** – the foreign key value is set to be the default value
  - ***CASCADE*** – the update / delete operation is performed with success in the parent table and also generate corresponding updates / deletes in the child table

Relational Databases with SQL

```
CREATE TABLE Category(
Cid INT PRIMARY KEY IDENTITY(1,1),
Name VARCHAR(30))

CREATE TABLE Product(
Pid INT PRIMARY KEY IDENTITY(1,1),
NameP VARCHAR(100),
Price INT CHECK(Price>0) NOT NULL,
Description VARCHAR(1000),
Cid INT FOREIGN KEY REFERENCES Category(Cid))

CREATE TABLE Client(
Clid INT PRIMARY KEY IDENTITY(1,1),
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50),
EMail VARCHAR(50),
MobilePhone CHAR(11) DEFAULT '40700000000'
)
```

```
CREATE TABLE Shopping(
Clid INT FOREIGN KEY REFERENCES Client(Clid),
Pid INT FOREIGN KEY REFERENCES Product(Pid),
PurchaseDate DATE
CONSTRAINT pk_Shopping PRIMARY KEY(Clid,Pid)
)

CREATE TABLE Review1(
Rid INT PRIMARY KEY IDENTITY,
Stars INT CHECK(Stars>0 AND Stars<=5),
Review VARCHAR(1000) DEFAULT 'Please add the review',
Clid INT REFERENCES Client
ON DELETE CASCADE
ON UPDATE NO ACTION
)
```

Relational Databases with SQL

- ***Data Definition Language (DDL)***

ALTER TABLE – allow the modifications on a structure of an already created table

ALTER TABLE table_name operation

where ***operation*** can be

- ***ADD column_definition*** – add a new column to the table
- ***ALTER COLUMN column_definition*** – modify / change the structure of a column of the table
- ***DROP COLUMN column_name*** – remove a column from a table

- Example:

ALTER TABLE Student ADD Dob DATE	ALTER TABLE Student ALTER COLUMN Dob DATETIME	ALTER TABLE Student DROP COLUMN Dob

Relational Databases with SQL

- ***Data Definition Language (DDL)***

Add / remove a CONSTRAINT

- ***ADD [CONSTRAINT constraint_name] PRIMARY KEY(column_list)*** – add a primary key constraint
- ***ADD [CONSTRAINT constraint_name] UNIQUE(column_list)*** – add a unique cosntraint
- ***ADD [CONSTRAINT constraint_name] FOREIGN KEY (column_list) REFERENCES table_name[(column_list)] [ON UPDATE action] [ON DELETE action]*** – add a foreign key constraint
- ***DROP [CONSTRAINT] constraint_name*** – remove a constraint

Example:

ALTER TABLE Student ADD CONSTRAINT PK_Student PRIMARY KEY(Sid) -- or ALTER TABLE Student ADD PRIMARY KEY(Sid)	ALTER TABLE Student ADD CONSTRAINT UK_Student UNIQUE(Name) -- or ALTER TABLE Student ADD UNIQUE(Name)	ALTER TABLE Student ADD CONSTRAINT FK_Student FOREIGN KEY (GroupId) REFERENCES Group(Gid) -- or ALTER TABLE Student ADD FOREIGN KEY (GroupId) REFERENCES Group(Gid)

Relational Databases with SQL

- ***Data Definition Language (DDL)***

Add / remove a CONSTRAINT

- ***ADD [CONSTRAINT constraint_name] PRIMARY KEY(column_list)*** – add a primary key constraint
- ***ADD [CONSTRAINT constraint_name] UNIQUE(column_list)*** – add a unique cosntraint
- ***ADD [CONSTRAINT constraint_name] FOREIGN KEY (column_list) REFERENCES table_name[(column_list)] [ON UPDATE action] [ON DELETE action]*** – add a foreign key constraint
- ***DROP [CONSTRAINT] constraint_name*** – remove a constraint

Example:

ALTER TABLE Student ADD CONSTRAINT FK_Student FOREIGN KEY (GroupId) REFERENCES Group(Gid) ON UPDATE NO ACTION ON DELETE CASCADE	ALTER TABLE Student ADD CONSTRAINT FK_Student FOREIGN KEY (GroupId) REFERENCES Group(Gid) ON UPDATE SET NULL ON DELETE SET DEFAULT	ALTER TABLE Student DROP CONSTRAINT FK_Student -- or ALTER TABLE Student DROP FK_Student

Relational Databases with SQL

- ***Data Definition Language (DDL)***

DROP TABLE – removes a table

DROP TABLE table_name

Example: DROP TABLE Student

Relational Databases with SQL

- ***Data Manipulation Language (DML)***
- Is a subset of SQL used to add / modify / remove components (e.g. records)
- INSERT command – used to add records in the tables

INSERT INTO table_name[(column_list)] VALUES (value_list)

INSERT INTO table_name[(column_list)] subquery

where, **value_list** must have the same order as the ***column_list*** and **subquery** is a set of records obtained from another ***SELECT statement***

Example:

- INSERT INTO Student(Sid, Name, Surname, Dob, Login, CityFaculty) VALUES (1, ‘Popescu’, ‘Dan’, ‘12/10/2000’, ‘poda456123’, ‘Cluj Napoca’)

OR

- INSERT INTO Student VALUES (1, ‘Popescu’, ‘Dan’, ‘12/10/2000’, ‘poda456123’, ‘Cluj Napoca’)

Relational Databases with SQL

- ***Data Manipulation Language (DML)***
- UPDATE command – used to change the values of the records in the tables

  UPDATE table_name

  SET column_name_1=expression_1 [, column_name_2=expression_2] …

  [WHERE condition]

- The command changes the values of the records from the table if the condition from the ***WHERE*** clause is fulfilled
- If the ***WHERE*** clause is omitted then all the records from the table are changed in the set columns
- The values of the columns specified in the ***SET*** clause are modified to the associated expressions' values

Example: Student[Sid, Name, Surname, Dob, Login, CityFaculty, Age], where Age is new added column

UPDATE Student SET Login='student123'	UPDATE Student SET CityFaculty='Brasov' WHERE Sid>10 AND Sid<100	UPDATE Student SET Login='student123', Age=Age+1 WHERE CityFaculty='Cluj Napoca'

Relational Databases with SQL

- ***Data Manipulation Language (DML)***
- DELETE command – used to remove the records from the tables

  DELETE FROM table_name

  [WHERE condition]

- The command removes / deletes the records from the table if the condition from the ***WHERE*** clause is fulfilled
- If the ***WHERE*** clause is omitted then all the records from the table are removed / deleted

Example: Student[Sid, Name, Surname, Dob, Login, CityFaculty, Age], where Age is new added column

DELETE FROM Student	DELETE FROM Student WHERE Name='Popescu' OR Age>25	DELETE FROM Student WHERE CityFaculty LIKE 'C%'

Relational Databases with SQL

- ***Data Manipulation Language (DML)***

Filter conditions:

- expression compare_operator expression1
- expression IS [NOT] NULL
- [NOT] EXISTS (subquery)
- expression [NOT] IN (subquery)
- expression [NOT] IN (value1, [value2] …)
- expression [NOT] BETWEEN min_value AND max_value
- expression [NOT] LIKE pattern – ‘%’ - replace 0 or more characters, ‘_’ – replace one character
- expression compare_operator {ANY | ALL} (subquery)
- (condition)
- [NOT] condition
- condition1 AND condition2
- condition1 OR condition2

Relational Databases with SQL

- ***Data Manipulation Language (DML)***

Logic values: TRUE, FALSE, UNKNOWN / NULL

	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:

- C.J. Date, *An Introduction to Database Systems (8th Edition)*, Addison-Wesley, 2003.
- H. Garcia-Molina, J. Ullman, J. Widom, *Database Systems: The Complete Book, Prentice Hall Press*, 2008.
- G. Hansen, J. Hansen, *Database Management And Design (2nd Edition),* Prentice Hall, 1996.
- R. Ramakrishnan, J. Gehrke, *Database Management Systems*, McGraw- Hill, 2007. http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- R. Ramakrishnan, J. Gehrke, *Database Management Systems (2nd Edition),* McGraw-Hill, 2000.
- A. Silberschatz, H. Korth, S. Sudarshan, *Database System Concepts*, McGraw-Hill, 2010. http://codex.cs.yale.edu/avi/db-book/
- L. Țâmbulea, *Curs Baze de date*, Facultatea de Matematică și Informatică, UBB, 2013-2014.
- J. Ullman, J. Widom, *A First Course in Database Systems*, http://infolab.stanford.edu/~ullman/fcdb.html