Databases – The Relational Model

Jörg Endrullis

VU University Amsterdam

Relational Model :: Example Database

Example Database (Students)

Students

sid	first	last	address
101	George	Orwell	London
102	Elvis	Presley	Memphis
103	Lisa	Simpson	Springfield
104	Bart	Simpson	Springfield
105	George	Washington	null

Meaning of the columns:

- `sid`: unique number that identifies the student
- `first`: first name
- `last`: last name
- `address`: city of residence (may be null)

Example Database (Exercises)

Exercises

category	number	topic	maxPoints
exam	1	SQL	14
homework	1	Logic	10
homework	2	SQL	10

Meaning of the columns:

- `category`: homework, midterm or exam
- `number`: exercise number within category
- `topic`: topic of exercise
- `maxPoints`: maximum number of points

Example Database (Results)

Students(sid, first, last, address)

Exercises(category, number, topic, maxPoints)

Results

sid	category	number	points
101	exam	1	12
101	homework	1	10
101	homework	2	8
102	exam	1	10
102	homework	1	9
102	homework	2	9
103	exam	1	7
103	homework	1	5

Meaning of the columns:

- `sid`: identifies the student (references `Students`)
- `category`, `number`: identifies exercise (references `Exercises`)
- `points`: graded points

Relational Model :: Database Schemas

Data Types and Domains

Table entries are **values** that conform to some **data type**.

Examples of SQL data types

- **strings**, e.g.
  - `varchar(n)` - strings of up to n characters ($n \leq 65535$)
  - `(long)text` - strings up to 4GB
- **numbers**, e.g.
  - `bit`, `int`, `float`, `...`
  - `numeric(p,s)` - decimal number $\underbrace{\quad}_{p-s \text{ digits}}.\underbrace{\quad}_{s \text{ digits}}$
- **binary data**, e.g. `blob`
- **date and time**, e.g. `date`, `time`, `datetime`, `timestamp` `...`

Available data types depend on the database management system, and the supported version of the SQL standard.

Data Types and Domains

The **domain** $dom(D)$ of a type D is the set of possible values.

$$dom(\texttt{int}) = \{\texttt{-2147483648}, \ldots, \texttt{2147483647}\}$$
$$dom(\texttt{numeric(2,0)}) = \{\texttt{-99}, \ldots, \texttt{99}\}$$

SQL allows to define **application-specific domains** as subsets of standard data types:

```
create domain ExampleDomain as numeric(2,0)
```

We may even add constraints:

```
create domain ExampleDomain as numeric(2,0) check(value > 0)
```

Domains are useful to document that two columns represent the same kind of objects and that comparisons are meaningful.

Relation Schema

Relation schema

A **relation schema** s (of a single relation) defines

- a (finite) sequence $A_1, \ldots, A_n$ of distinct **attribute names**,
- for each attribute A_i a **data type** (or **domain**) D_i.

A relation schema can be written as

$$s = (A_1 : D_1, \ldots, A_n : D_n)$$

```
Exercises(category, number, topic, maxPoints)
```

Creating a relation schema in SQL

```
create table Exercises (
  category  varchar(10),
  number    numeric(2,0),
  topic     varchar(40),
  maxPoints numeric(2,0)
)
```

Relation Schema: Notation

How to **communicate schemas from human to human**?
SQL `create table` statements are far from ideal.

Relation schema in SQL

```
create table Exercises (
  category  varchar(10),
  number    numeric(2,0),
  topic     varchar(40),
  maxPoints numeric(2,0)
)
```

If the column data types are not important, we can write

`Exercises(`<u>`category`</u>`,` <u>`number`</u>`, topic, maxPoints)`

Also widely in use: sketch of the table header

Exercises			
<u>category</u>	**<u>number</u>**	**topic**	**maxPoints**

Relational Database Schemas

Relational database schema

A **relational database schmema** S defines

- a finite set of **relation names** $\{ R_1, \ldots, R_m \}$,
- a **relation schema** $schema(R_i)$ for every relation R_i,
- a set of **integrity constraints** C (defined later).

In summary, $S = (\{ R_1, \ldots, R_m \}, schema, C)$.

Example: relational database schema

- relation names $\{$ `Students`, `Exercises`, `Results` $\}$
- relation schema for every relation name
  - `Students(`<u>`sid`</u>`, first, last, address)`
  - `Exercises(`<u>`category`</u>`, `<u>`number`</u>`, topic, maxPoints)`
  - `Results(`<u>`sid`</u>`, `<u>`category`</u>`, `<u>`number`</u>`, points)`

Examples of integrity constraints: keys and foreign keys.

Relational Model :: Database States

Database States: Tuples

Tuples are used to formalise table rows.

A **tuple** t with respect to the relation schema

$$s = (A_1 : D_1, \ldots, A_n : D_n)$$

is a sequence $t = (d_1, \ldots, d_n)$ of values such that $d_i \in dom(D_i)$.

In other words: $t \in dom(D_1) \times \cdots \times dom(D_n)$.

For instance, `('exam', 1, 'SQL', 14)` is a tuple in the table

Exercises

category	number	topic	maxPoints
exam	1	SQL	14
homework	1	Logic	10
homework	2	SQL	10

Given a tuple t, we write $t.A_i$ for the value in column A_i.

For instance, `('exam', 1, 'SQL', 14).topic = 'SQL'`.

Database States

Let $S = (\{R_1, \ldots, R_m\}, \mathit{schema}, C)$ be a database schema.

A **database state** I for database schema S defines

- for every relation name R_i,
  a finite **set of tuples** $I(R_i)$ with respect to $\mathit{schema}(R_i)$

If $\mathit{schema}(R_i) = (A_1 : D_1, \ldots, A_n : D_n)$, then

$$I(R_i) \subseteq \mathit{dom}(D_1) \times \cdots \times \mathit{dom}(D_n)$$

Thus $I(R_i)$ is **a relation in the mathematical sense**.

Databases state = set of tables conforming to the schema:

Students

sid	first	last	address
101	George	Orwell	London
102	Elvis	Presley	Memphis
103	Lisa	Simpson	Springfield
104	Bart	Simpson	Springfield
105	George	Washington	null

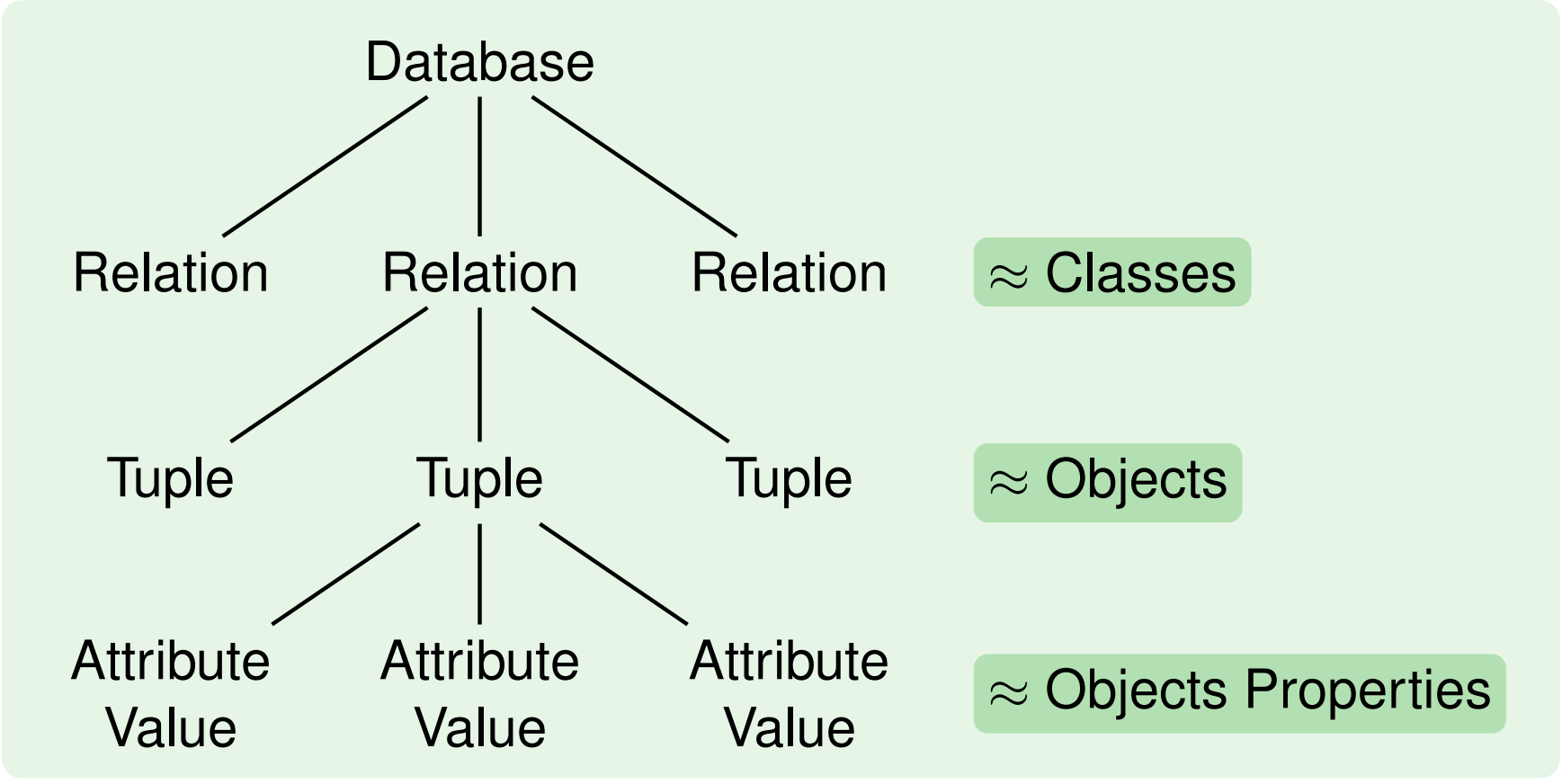
Exercises

category	number	topic	maxPoints
exam	1	SQL	14
homework	1	Logic	10
homework	2	SQL	10

Except:

- there is **no order** on the tuples (rows), and
- tables contain **no duplicate** tuples.

Summary Database States



Relational Model :: Null Values

Null Values

The relational model allows **missing attribute values**.
(Table entries may be empty.)

Students

sid	first	last	address
101	George	Orwell	London
102	Elvis	Presley	Memphis
103	Lisa	Simpson	Springfield
104	Bart	Simpson	Springfield
105	George	Washington	null

Formally, the set of possible values (the domain) for an attribute is extended by a new special value **“null”**.

This “null” is **not** the number 0 or the empty string.
A null value is different from all values of any data type.

Null Values

Null values are used to model a variety of scenarios:

- **No value exists.**
  A student might not have an e-mail address.
- **The attribute is not applicable for this tuple.**
  Some exercises are for training only: no points will be given.
- **A value exists (in the real world), but is not known.**
  In table `Students`, `address` *might be unknown for a student.*
- **Any value will do.**

Since the same null value is used for quite different purposes, there can be **no clear semantics.**

Null Values: Advantages

Without null values, it would be necessary to **split a relation** into many, more specific relations ("subclasses").

Example

`Students_with_address`, `Students_without_address`

Alternatively: introduce an additional relation with schema

`Students_Address(sid, address)`

But this complicates queries: *join operations are needed (upcoming).*

If null values are not allowed

- **users** might invent **fake values** to fill the missing columns

Fake values

Why are fake values a bad idea in database design?

Null Values: Problems

SQL uses a **three-valued logic: true, false, unknown**!
Any comparison with `null` yields the value **unknown**.

For users accustomed to two-valued logic, the outcome is often surprising.

Which of these queries return rows with `null` in column A?

1. `select * from R where A = 42`
2. `select * from R where not (A = 42)`
3. `select * from R where A = null`

None of these queries does!

To get the rows with `null` values, use `... where A is null`.

Some languages do not know about null values.

Explicit null value check and treatment required when reading attribute values into program variables. This complicates application programs.

Excluding Null Values

Since null values may lead to complications, SQL allows to control whether an attribute value may be null or not.

By default, null values are allowed.

```
create table Students (
  sid     numeric(3,0) not null,
  first   varchar(20)  not null,
  last    varchar(20)  not null,
  address varchar(80)
)
```

Declaring attributes as `not null` leads to

- simpler application programs, and
- fewer surprises during query evaluation.

Relational Model :: Integrity Constraints

Valid Database States

Primary goal of database design

Database should model **the relevant part of the real world**.

The plain definition of tables often allows **too many** (meaningless, illegal) database states.

A valid database state?

Customer

number	name	birth_year	city	...
1	Smith	1936	Pittsburgh	...
2	Jones	1965	Philadelphia	...
3	Brown	64	New York	...
3	Ford	2015	Washington	...

- customer numbers must be unique
- the year of birth must be greater than 1870
- customers must be at least 18 years old

Integrity Constraints

Integrity constraints (IC) are conditions which every database state has to satisfy.

- This restricts the set of possible database states.
  Ideally only admits images of possible real world scenarios.
- Integrity constraints are specified in the database schema.

The database management system will **refuse any update** leading to a database state that violates any of the constraints.

Integrity Constraints in SQL

The SQL `create table` allows the following **constraints**:

- **Not Null**:

  No value in this column can be the null value.

- **Key constraints:**

  Each key value can appear once only.

- **Foreign keys constraints:**

  Values in a column must also appear as key values in another table.

- **Check constraints**:

  Column values must satisfy a given predicate.
  SQL allows for inter-column `CHECK` *constraints.*

```
create table Products (
  id    int           primary key,
  name  varchar(255)  not null,
  price numeric(10,2) check(price > 0)
)
```

Summary: Integrity Constraints

Why specify integrity constraints?

- Constraints **document** knowledge about valid DB states.
- Some protection against **data input errors.**
- **Enforcement of law / company standards.**
- **Protection against inconsistency** if data is stored redundantly.
- **Queries/application programs become simpler** if the programmer may assume that the data fulfils certain properties.

Relational Model :: Keys

Keys

A **key** of a relation R is a set of attributes $\{A_1, \ldots, A_n\}$ that **uniquely identify** the tuples in R.

The **key constraint** is satisfied in the DB state I if and only if

$$t.A_1 = u.A_1 \;\&\; \ldots \;\&\; t.A_n = u.A_n \quad \Longrightarrow \quad t = u$$

for all tuples $t, u \in I(R)$.

So, different tuples differ in at least one of the values $A_1, \ldots, A_n$.

If `sid` is declared a key for `Students`, this is illegal:

Students

sid	first	last	address
101	George	Orwell	...
102	Elvis	Presley	...
101	Lisa	Simpson	...

Once a key has been declared the DBMS will refuse any insertion of tuples with duplicate key values.

Composite Keys

If $\{A, B\}$ is a key, rows may agree in A **or** B, but not **both**.

Students

sid	**first**	**last**	**address**
103	Lisa	Simpson	...
104	Bart	Simpson	...
106	Bart	Smit	...

This relation

- **violates** the key constraint `first`,
- **violates** the key constraint `last`,
- but **satisfies** the key constraint $\{$`first`, `last`$\}$.

Quiz

Do all relations have a key?

Minimality of Keys

Students		
sid	**first**	**last**
103	Lisa	Simpson
104	Bart	Simpson
106	Bart	Smit

What keys satisfy the key constraints?

- {sid} minimal
- {first, last} minimal
- {sid, first}
- {sid, last}
- {sid, first, last}

Implication between key constraints

If A is a key and $A \subsetneq B$, then B is also a key.
The key B is **weaker** (more database states are valid) than A.

Any superset of a key is itself a key.

A key $\{A_1, \ldots, A_k\}$ is **minimal** if no proper subset is a key.

In the literature, often keys are required to be minimal.

Multiple Keys

A relation may have **more than one minimal key**.

In the relational model, one key is designated as **primary key.** A primary key **cannot be null.**

All other keys are called **alternate** or **secondary keys.**

The primary key attributes are often marked by **underlining**:

$$R(\underline{A_1}, \ldots, \underline{A_k}, A_{k+1}, \ldots, A_n)$$

Here $\{ A_1, \ldots, A_k \}$ is the primary key of R.

It is good design practice to define a primary key that

- consists of a **single (simple) attribute** only,
- is **never updated**.

This is good for

- consistency (applications might store the key), and
- indexing and retrieving items.

Keys are Contraints

Keys are constraints: they refer to all possible DB states, not only the current one.

```
create table Exercises (
  ...
  primary key (category, number)
)
```

Students

sid	first	last	address
101	George	Orwell	...
103	Lisa	Simpson	...

In this database state `first` is a key for `Students`. However, this is too restrictive for all intended database states. A future insertion of `George Washington` would be impossible.

When declaring keys, think about all intended database states!

Key Quiz

Keys for an appointment calendar

Appointments

date	start	end	room	event
Jan. 19	10:00	11:00	WN 726	Seminar
Jan. 19	14:00	16:00	WN 726	Lecture
May 24	14:00	18:00	Amsterdam	Meeting

- What would be correct minimal keys for this database?
- What would be an example for a superkey?
- Are additional constraints useful to exclude database states that a key would still permit?

Relational Model :: Foreign Keys

Foreign Keys

The relational model does **not provide explicit relationships, links, or pointers**.

Idea: use the key attributes to reference a tuple.

The values for the key attributes uniquely identify a tuple.

Foreign keys

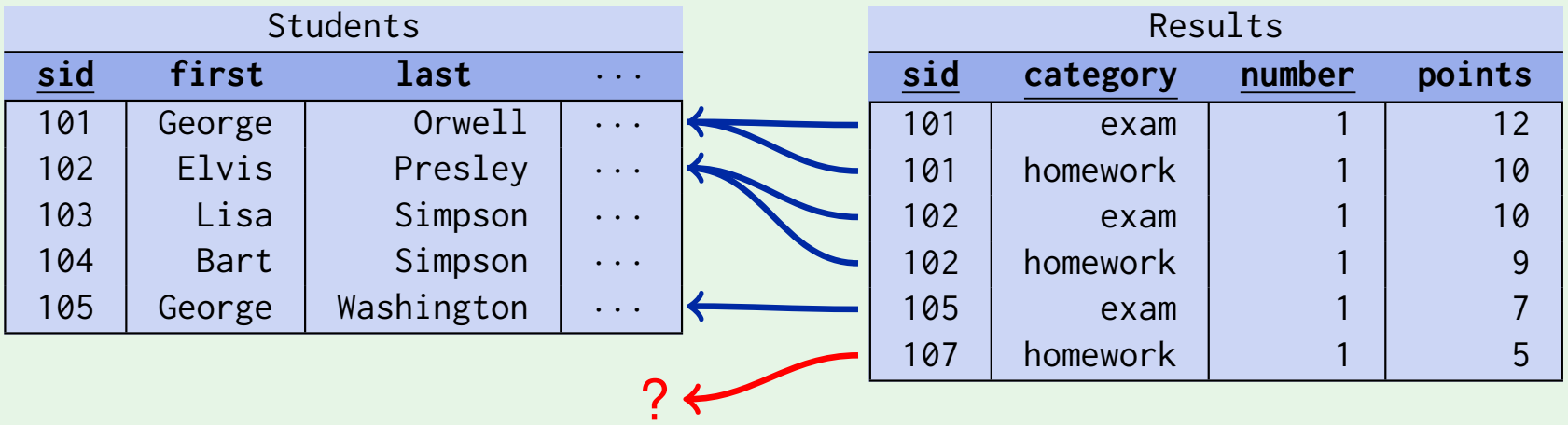
To refer from a relation R to tuples of S:
add the **primary key attributes** of S to the attributes of R

Such a reference is only "**stable**" if the "address" of a tuple does not change, that is, **if the key attributes are not updated**.

Foreign Keys

A foreign key implements a **one-to-many relationship**.

In table `Results`, `sid` is a foreign key referencing `Students`



Foreign keys are not themselves keys.
Here `sid` is not a key of `Results`.

We need an **existence guarantee** for key values in `Students`.

The set of `sid` values appearing in `Students` forms a **dynamic domain** for the attribute `Results.sid`.

Foreign Key Constraints

Foreign key constraints in SQL

```
create table Results (
  ...
  foreign key (sid) references Students(sid)
)
```

The **foreign key constraint** ensures that

for every tuple in $t \in$ `Results` where t.`sid` is not `null`,
there exist a tuple $u \in$ `Students`
such that t.`sid` $=$ u.`sid`

Foreign keys **may be null**, unless with `not null` constraint.

This corresponds to a "null" pointer in programming languages.

The **referential integrity** of the database is ensured by enforcing the foreign key constraints.

Foreign Key Constraints

Once the foreign key is declared, the **the following update update operations violate the foreign key constraint**:

Insertion into table `Results`
without matching tuple in `Students`

- DBMS **rejects** the update

Deletion from table `Students`
if the deleted tuple is referenced in `Results`

- DBMS **rejects** the update, or
- deletion **cascades**, that is, tuples in `Results` referencing the deleted tuple will also be deleted, or
- the foreign key is **set to null** in `Results`.

Configure using: `on delete cascade | set null | ...`

Foreign Keys and Notation

Only keys may be referenced (primary or secondary).

References to non-key attributes are not permitted.

A table with a composite key must be referenced with a **composite foreign key that has the same number of attributes and domains**.

It is not required that the corresponding attributes have identical names.

Foreign keys are denoted with arrows ($\rightarrow$) in the schema. Composite keys appear in parentheses:

```
Students(sid, first, last, address)
Exercises(category, number, topic, maxPoints)
Results(sid → Students, (category, number) → Exercises, points)
```

Typically the primary key is referenced, so it suffices to list the target relation.