Databases

Jörg Endrullis

VU University Amsterdam

The Relational Model

Overview

Example Database (Students)

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
102	Michael	Jones	null
103	Richard	Turner	...
104	Maria	Brown	...

- Columns in table STUDENTS:
    - SID: "student ID" (unique number)
    - FIRST, LAST: first and last name
    - EMAIL: email address (may be null)

Example Database (Exercises)

EXERCISES			
CAT	**ENO**	**TOPIC**	**MAXPT**
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

- Columns in table EXERCISES:
    - CAT: category
        - H: Homework
        - M: midterm exam
        - F: final exam
    - ENO: exercise number within category
    - TOPIC: topic of exercise
    - MAXPT: maximum number of points

Example Database

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
102	Michael	Jones	null
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES			
CAT	**ENO**	**TOPIC**	**MAXPT**
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

RESULTS			
SID	**CAT**	**ENO**	**POINTS**
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

- Columns in table RESULTS:
    - SID: student who did the exercise (references STUDENTS)
    - CAT, ENO: identification of exercise (references EXERCISE)
    - POINTS: graded points

Data Values, Types and Domains

All table entries are **data values** which conform to some given selection of **data types.**

Examples of data types

- strings, e.g.
    - VARCHAR(n) - strings of up to *n* characters
    - TEXT - variable length storage of strings up to 2GB
- numbers (of different lengths and precision), e.g.
    - INT - an integer
    - NUMERIC(n) - decimal number with *n* digits
- date and time
- binary data, e.g.
    - BLOB - large binary object

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

Data Values, Types and Domains

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

$$val(\text{INT}) = \{-2147483648, \ldots, 2147483647\}$$
$$val(\text{NUMERIC(2)}) = \{-99, \ldots, 99\}$$

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

```
CREATE DOMAIN EXNUM AS NUMERIC(2)
```

We may even add constraints:

```
CREATE DOMAIN EXNUM AS NUMERIC(2) CHECK(VALUE > 0)
```

Domains are useful to document that two columns represent the same kind of real-world object (such that, for example, comparisons between values in the columns are meaningful).

Relation Schema

Relation schema

A **relation schema** s (schema 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 $s = (A_1 : D_1, \ldots, A_n : D_n)$.

Let $dom(A_i) = val(D_i)$ be the set of possible values for A_i.

Creating a relation schema in SQL

EXERCISES			
CAT	**ENO**	**TOPIC**	**MAXPT**
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

Relation schema in SQL

```
CREATE TABLE EXERCISES
  (CAT   CHAR(1),
   ENO   NUMERIC(2),
   TOPIC VARCHAR(40),
   MAXPT NUMERIC(2))
```

Relation Schema: Notation

SQL `CREATE TABLE` statements represent a rather poor way to communicate schemas (from human to human):

> **Relation schema in SQL**
>
> ```
> CREATE TABLE EXERCISES
>   (CAT   CHAR(1),
>    ENO   NUMERIC(2),
>    TOPIC VARCHAR(40),
>    MAXPT NUMERIC(2))
> ```

Often it is useful to talk about abstractions of the schema:

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

  EXERCISES (CAT, ENO, TOPIC, MAXPT)

- Also widely in use: sketch of the table header

EXERCISES			
CAT	**ENO**	**TOPIC**	**MAXPT**

Relational Database Schemas

Relational database schema

A **relational database schmema** $\mathcal{S}$ defines

- A finite set of **relation names** $\{R_1, \dots, R_m\}$.
- For every relation R_i, a **relation schema** $schema(R_i)$.
- A set $\mathcal{C}$ of **integrity constraints** (defined below).

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

Example: relational database schema

- relation names

  { STUDENTS, EXERCISES, RESULTS }

- relation schema for every relation name
  - STUDENTS (SID, FIRST, LAST, EMAIL)
  - EXERCISES (CAT, ENO, TOPIC, MAXPT)
  - RESULTS (SID, CAT, ENO, POINTS)

Tuples

Tuples are used to formalize table rows.

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

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

is a sequence $(d_1, \ldots, d_n)$ of n values such that $d_i \in val(D_i)$.

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

EXERCISES			
CAT	**ENO**	**TOPIC**	**MAXPT**
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

$('H', 1, 'Rel.Alg.', 10)$ is a tuple in the table EXERCISES.

Given a tuple, we write $t.A_i$ (or $t[A_i]$) for the value in column A_i.

For instance, $('H', 1, 'Rel.Alg.', 10).\text{MAXPT} = 10$.

Database States

Let a database schema $(\{R_1, \ldots, R_m\}, schema, \mathcal{C})$ be given.

A **database state I** for this database schema defines for every relation name R_i to a finite **set of tuples** $I(R_i)$ with respect to the relation schema $schema(R_i)$.

So, if $schema(R_i) = (A_1 : D_1, \ldots, A_n : D_n)$, then

$$I(R_i) \subseteq val(D_1) \times \cdots \times val(D_n)$$

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

You can think of the state as tables conforming to the schema

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
102	Michael	Jones	null
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES			
CAT	**ENO**	**TOPIC**	**MAXPT**
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

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

Summary Relational Database Schemas
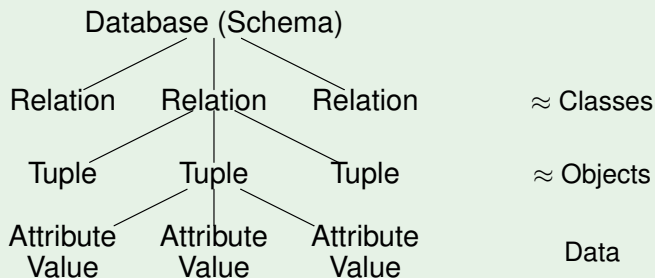
The Relational Model

Overview

Null Values

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

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
102	Michael	Jones	null
103	Richard	Turner	...
104	Maria	Brown	...

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

"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, EMAIL *might be missing for a student.*

- **Any value will do.**

Null Values: Advantages

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

Example

STUDENT_WITH_EMAIL,   STUDENT_WITHOUT_EMAIL

Alternatively: introduce an additional relation with schema

STUD_EMAIL (SID, EMAIL)

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

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

SQL uses a **three-valued logic** (true, false, unknown) for the evaluation of comparisons that involve null values.
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! Use `... WHERE A IS NULL`.

Some programming 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.

Students may not have an e-mail address

```
CREATE TABLE STUDENTS (
            SID      NUMERIC(3)  NOT NULL,
            FIRST    VARCHAR(20) NOT NULL,
            LAST     VARCHAR(20) NOT NULL,
            EMAIL    VARCHAR(80)          )
```

Declaring many attributes as NOT NULL

- leads to simpler application programs
- fewer surprises during query evaluation

The Relational Model

Overview

Valid Database States

Primary goal of DB design

The database should be an **image of the relevant subset of the real world**.

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

A valid database state?

CUSTOMER				
CUST_NO	**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.

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.

Constraints in SQL

In the SQL `CREATE TABLE` statement, the following **types of constraints** may be specified:

- **`NOT NULL`**:
  No value in this column can be the null value.

- **Keys:**
  Each key value can appear once only.

- **Foreign keys:**
  Values in a column must also appear as key values in another table.

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

Which of the following are constraints?

1. It is possible that a student gets 0 points for a solution.

2. A student can get at most 3 points more than the maximum number of points stored in EXERCISES (extra credit).

3. The attribute CAT in can only have the values H, M, F.

4. The CAT means: H for homework, M for mid-term exam, ...

5. Student IDs should be unique.

Summary: Constraints

Why specify constraints?

- (Some) protection against **data input errors.**
- Constraints **document** knowledge about DB states.
- **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.

The Relational Model

Overview

Keys

A **key** of a relation R is an attribute A that **uniquely identifies** the tuples in R.

The **key constraint** is satisfied in the DB state I if and only if, for all tuples $t, u \in I(R)$ the following holds:

$$t.A = u.A \quad \implies \quad t = u$$

In other words: different tuples have different values for A.

Example

If attribute SID has been declared as key for STUDENTS, this database state is illegal:

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
101	Michael	Jones	(null)
103	Michael	Turner	...

Keys

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

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

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Michael	Turner	...

Even though the above DB state would allow the attribute LAST to serve as a key for STUDENTS, this would be **too restrictive.**

The future insertion of "John Smith" would be impossible.

Composite Keys

In general, a key can consist of **several attributes**. Such keys are also called **composite keys.**

If columns *A*, *B* together form a composite key, it is forbidden that there are two tuples $t \neq u$ which agree in **both** attributes.

$$t.A = u.A \ \wedge \ t.B = u.B \quad \Longrightarrow \quad t = u$$

Columns may in agree *A* **or** *B*, though, e.g.:

This relation satisfies the composite key FIRST, LAST:

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
102	John	Smith	...
103	John	Miller	...

Composite Keys

Implication between key constraints

A key constraint becomes **weaker** (i.e., less restrictive, more DB states are valid) if attributes are added to the key.

STUDENTS			
SID	**FIRST**	**LAST**	**EMAIL**
101	Ann	Smith	...
102	John	Smith	...
103	John	Miller	...

This relation

- **violates** the key constraint FIRST,
- **violates** the key constraint LAST,
- but **satisfies** the key constraint FIRST,LAST.

Weak keys

Do all relations have a key (what is the weakest possible key)?

Minimality of Keys

STUDENTS		
SID	**FIRST**	**LAST**
101	Ann	Smith
102	John	Smith
103	John	Miller

What keys satisfy the key constraints?
- {SID} minimal
- {FIRST, LAST} minimal
- {SID, FIRST}
- {SID, LAST}
- {SID, FIRST, LAST}

If a set of attributes A satisfies the key constraint, then **any superset** K that includes A will automatically also have the unique identification property.

A key with attribute set $\{A_1, \ldots, A_k\}$ is **minimal** if no A_i can be removed from the set without destroying the unique identification property.

The usual definition of keys requires that the set of key attributes $\{A_1, \ldots, A_k\}$ is minimal.

Multiple Keys

A relation may have **more than one (minimal) key** (last slide).

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**:

$$R(\underline{A_1 : D_1}, \ldots, \underline{A_k : D_k}, A_{k+1} : D_{k+1}, \ldots, A_n : D_n)$$

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 for an appointment calendar

APPOINTMENTS				
DATE	**START**	**END**	**ROOM**	**EVENT**
Jan. 19	10:00	11:00	IS 726	Seminar
Jan. 19	14:00	16:00	IS 726	Lecture
May 24	14:00	18:00	Amsterdam	Meeting

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

The Relational Model

Overview

1. Relational Model Concepts: Schema, State

2. Null Values

3. Constraints: General Remarks

4. Key Constraints

5. Foreign Key Constraints

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.
*The key attributes values may serve as **logical tuple addresses**.*

Foreign keys

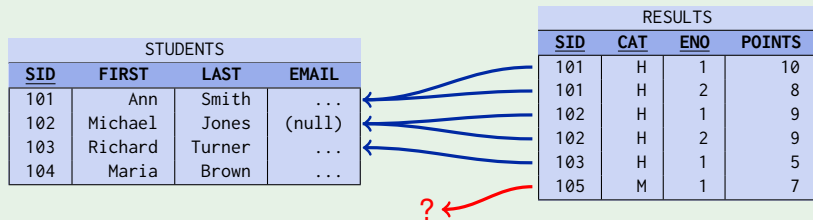To refer from a relation R to tuples of S

- add the **primary key attributes** of S to the attributes of R

Such a reference is only "stable" if the (logical) address of a tuple does not change (if the key attributes are not updated).

Foreign Keys

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

SID in RESULTS is a foreign key referencing STUDENTS

We need an **existence guarantee** for key values in STUDENTS.

The set of SID values appearing in STUDENTS should be a kind of **dynamic domain** for the attribute REULTS.SID.

Foreign Key Constraints

Foreign Key Constraints

```
CREATE TABLE RESULTS (
   ...
   FOREIGN KEY (SID) REFERENCES STUDENTS(SID)
)
```

The foreign key constraint ensures that

for every tuple in t in RESULTS,
there exist a tuple u in STUDENTS
such that $t.SID = u.SID$

Enforcing foreign key constraints ensures the **referential integrity** of the database.

Foreign Key Constraints

Once a 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.

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 relation
schema, composite keys appear in parentheses:

            RESULTS (SID $\rightarrow$ STUDENTS,
                     (CAT, ENO) $\rightarrow$ EXERCISES,
                     POINTS)
           STUDENTS (SID,FIRST,LAST,EMAIL)
          EXERCISES (CAT,ENO,TOPIC,MAXPT)

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

Foreign Keys are not Keys

Foreign keys **may be null**, unless with `NOT NULL` constraint.
This corresponds to a "null" pointer in programming languages.

Foreign keys are not themselves keys.

The Relational Model: Objectives

After completing this chapter, you should be able to

- explain the concepts of the **Relational Model**,
    - Schemas, state, domains
    - read various notations for relational schema

- explain applications and problems of **null values**,

- explain **integrity constraints** and their importance,

- explain the meaning of **keys** and **foreign keys**,

- develop simple relational schemas.