Principles of Data- and Knowledge-based Systems

Torsten Schaub
University of Potsdam
`torsten@cs.uni-potsdam.de`

Relational model: Overview

Outline

Example Database (1)

STUDENTS

<u>SID</u>	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

EXERCISES

<u>CAT</u>	<u>ENO</u>	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS

<u>SID</u>	<u>CAT</u>	<u>ENO</u>	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

Example Database (2)

- `STUDENTS`: one row for each student in the course.
  - `SID`: “Student ID” (unique number).
  - `FIRST`, `LAST`: First and last name.
  - `EMAIL`: Email address (can be null).
- `EXERCISES`: one row for each exercise.
  - `CAT`: Exercise category.
  - `ENO`: Exercise number (within category).
  - `TOPIC`: Topic of the exercise.
  - `MAXPT`: Max. no. of points (How many points is it worth?).
- `RESULTS`: one row for each submitted solution to an exercise.
  - `SID`: Student who wrote the solution.
  - `CAT`, `ENO`: Identification of the exercise.
  - `POINTS`: Number of points the student got for the solution.
  - A missing row means that the student did not yet hand in a solution to the exercise.

Data Values (1)

- Table entries are data values taken from some given selection of data types.
- The possible data types are given by the RDBMS (or the SQL standard).
- E.g. strings, numbers (of different lengths and precisions), date and time, money, binary data.
- The relational model (RM) itself is independent from any specific selection of data types.
- Extensible DBMS allow the user to define new data types (e.g. multimedia and geometric data types).
- This extensibility is one important feature of modern object-relational systems.

Data Values (2)

- As explained in Part 2 (Logic), the given data types are specified in form of a signature $\Sigma_{\mathcal{D}} = (\mathcal{S}_{\mathcal{D}}, \mathcal{P}_{\mathcal{D}}, \mathcal{F}_{\mathcal{D}})$ and a $\Sigma_{\mathcal{D}}$-interpretation $\mathcal{I}_{\mathcal{D}}$.
- In the following definitions we only need that
  - a set $\mathcal{S}_{\mathcal{D}}$ of data type names is given, and
  - for each $D \in \mathcal{S}_{\mathcal{D}}$ a set $\mathit{val}(D)$ of possible values of that type ($\mathit{val}(D) := \mathcal{I}_{\mathcal{D}}[D]$).
- E.g. the data type "`NUMERIC(2)`" has values `-99..+99`.

Domains (1)

- The columns `ENO` in `RESULTS` and `ENO` in `EXERCISES` should have the same data type (both are exercise numbers). The same holds for `EXERCISES.MAXPT` and `RESULTS.POINTS`.
- One can define application-specific “domains” as names (abbreviations) for the standard data types:

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

- One could even add the constraint that the number must be positive.

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

Domains (2)

- Then the column data type is defined indirectly via a domain:

$$\begin{tikzcd}
\begin{array}{c}\text{Column}\\ \text{“EXERCISES.ENO”}\end{array} \arrow[dr] & & \\
 & \begin{array}{c}\text{Domain}\\ \text{“EX\_NUMBER”}\end{array} \arrow[r] & \begin{array}{c}\text{Data Type}\\ \text{“NUMERIC(2)”}\end{array} \\
\begin{array}{c}\text{Column}\\ \text{“RESULTS.ENO”}\end{array} \arrow[ur] & &
\end{tikzcd}$$

- If it should ever be necessary to extend the set of possible homework numbers, e.g. to `NUMERIC(3)`, this structure ensures that no column is forgotten.

Domains (3)

- Domains are also useful in order to document that the two columns contain the same kind of thing, so comparisons between them are meaningful.
- E.g. even if the column "`POINTS`" has the same data type "`NUMERIC(2)`", this query makes little sense:

  "Which homework has a number that is
  the same as its number of points?"
- However, SQL does not forbid comparisons between values of different domains.

Atomic Attribute Values

- The relational model treats the single table entries as atomic.
- I.e. the classical relational model does not permit to introduce structured and multi-valued column values.

- In contrast, the NF^2 ("Non First Normal Form") data model allows table entries to be complete tables in themselves.
- Support for "complex values" (sets, lists, records, nested tables) is another typical feature of object-relational systems.

Relational DB Schemas (1)

- A relation schema ρ (schema of a single relation) defines
  - a (finite) sequence $A_1 \ldots A_n$ of attribute names,
  - for each attribute A_i a data type (or domain) D_i.
- A relation schema can be written as

$$\rho = (A_1\colon D_1,\ \ldots,\ A_n\colon D_n).$$

Relational DB Schemas (2)

- A relational database schema $\mathcal{R}$ defines
  - a finite set of relation names $\{R_1, \ldots, R_m\}$, and
  - for every relation R_i, a relation schema $sch(R_i)$.
  - A set $\mathcal{C}$ of integrity constraints (defined below).
- That is, $\mathcal{R} = \big(\{R_1, \ldots, R_m\},\ sch,\ \mathcal{C}\big)$.

- Compared to the definitions in Part 2 (Logic), the attribute names are new.

Relational DB Schemas (3)

Consequences of the Definition

- Column names must be unique within a table:
  no table can have two columns with the same name.
- However, different tables can have columns with the same name (e.g. `ENO` in the example).
- For every column (identified by the combination of table name and column name) there is a unique data type.
- The columns within a table are ordered, i.e. there is a first, second, etc. column.
- Within a DB schema, table names must be unique:
  There cannot be two tables with the same name.
- A DBMS server can normally manage several database schemas.

Schemas: Notation (1)

- Consider the example table:

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

- One way to specify the schema precisely is via an SQL statement:

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

Schemas: Notation (2)

- A `CREATE TABLE` statement is needed for the DBMS, other notations can be used for communicating schemas between humans.
- When discussing the general database structure, the column data types are often not important.
- One concise notation is to write the table name followed by the list of attributes:

```
EXERCISES(CAT, ENO, TOPIC, MAXPT)
```

- If necessary, column datatypes can be added:

```
EXERCISES(CAT: CHAR(1), ...)
```

- One can also use the header (sketch) of the table:

EXERCISES			
CAT	ENO	TOPIC	MAXPT
⋮	⋮	⋮	⋮

Tuples (1)

- An *n*-tuple is a sequence of *n* values.
- E.g. XY-coordinates are pairs (X, Y) of real numbers.
  Pairs are tuples of length 2 ("2-tuples").
- The cartesian product $\times$ constructs sets of tuples, e.g.:

$$\mathbb{R} \times \mathbb{R} := \{(X, Y) \mid X \in \mathbb{R},\ Y \in \mathbb{R}\}.$$

Tuples (2)

- A tuple t with respect to the relation schema

  $$\rho = (A_1 \colon D_1, \ldots, A_n \colon D_n)$$

  is a sequence $(d_1, \ldots, d_n)$ of n values such that $d_i \in val(D_i)$.
- I.e. $t \in val(D_1) \times \cdots \times val(D_n)$.
- Given such a tuple, we write $t.A_i$ for the value d_i in the column A_i.
- E.g. one row in the example table "`EXERCISES`" is the tuple ('H', 1, 'Rel. Algeb.', 10).

Database States (1)

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

- A database state $\mathcal{I}$ for this database schema defines for every relation R_i a finite set of tuples with respect to the relation schema $sch(R_i)$.
- I.e. if $sch(R_i) = (A_{i,1} \colon D_{i,1},\ \ldots,\ A_{i,n_i} \colon D_{i,n_i})$, then

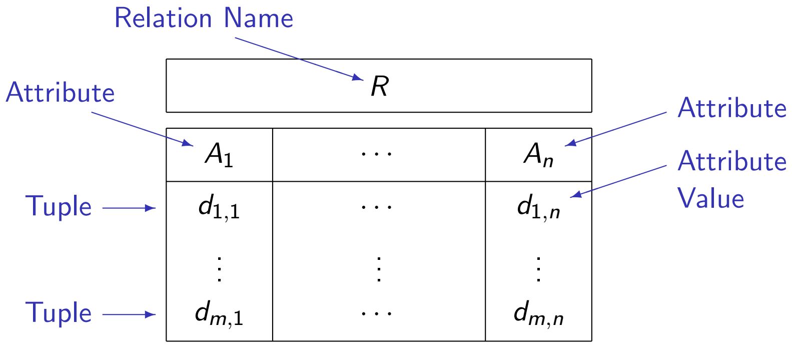
$$\mathcal{I}[R_i] \ \subseteq\ val(D_{i,1}) \ \times\ \cdots\ \times\ val(D_{i,n_i}).$$

- I.e. a DB state interprets the symbols in the DB schema: It maps relation names to relations.

Database States (2)

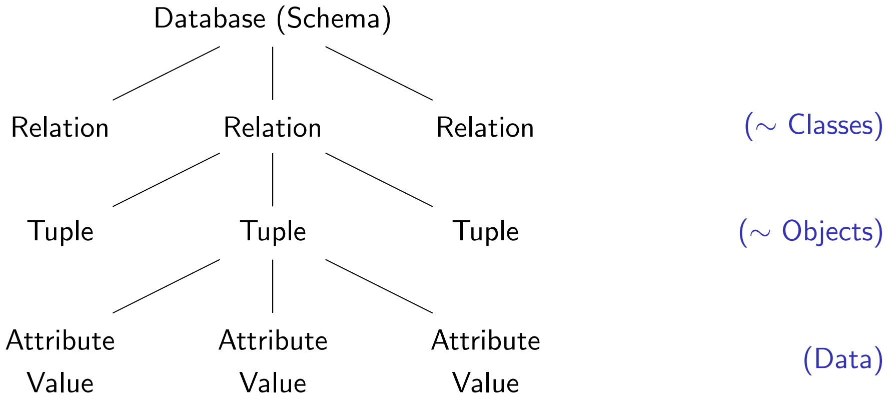
- In mathematics, the term “relation” is defined as “a subset of a cartesian product”.
- E.g. an order relation such as “$<$” on the natural numbers is formally: $\{(X, Y) \in \mathbb{N} \times \mathbb{N} \mid X < Y\}$.
- Relations are sets of tuples. Therefore:
  - The sequence of the tuples is undefined.
    - The tabular representation is a bit misleading, there is no first, second, etc. row.
    - Relations can be sorted on output.
  - There are no duplicate tuples.
    - Most current systems allow duplicate tuples as long as no key is defined (see below).

Summary (1)



Synonyms: Relation and Table.
Tuple, row, and record.
Attribute, column, field.
Attribute value, column value, table entry.

Summary (2)


Database (Schema)
Relation
Relation
Relation
(∼ Classes)
Tuple
Tuple
Tuple
(∼ Objects)
Attribute Value
Attribute Value
Attribute Value
(Data)

Update Operations (1)

- Updates transform a DB state $\mathcal{I}_{\text{old}}$ into a DB state $\mathcal{I}_{\text{new}}$. The basic update operations of the RM are:
  - Insertion (of a tuple into a relation):

$$\mathcal{I}_{\text{new}}[R] := \mathcal{I}_{\text{old}}[R] \cup \{(d_1, \ldots, d_n)\}$$

  - Deletion (of a tuple from a relation):

$$\mathcal{I}_{\text{new}}[R] := \mathcal{I}_{\text{old}}[R] \setminus \{(d_1, \ldots, d_n)\}$$

  - Modification / Update (of a tuple):

$$\begin{aligned}\mathcal{I}_{\text{new}}[R] := \big(\mathcal{I}_{\text{old}}[R] & \setminus \{(d_1, \ldots, d_i, \ldots, d_n)\}\big)\\ & \cup \{(d_1, \ldots, d_i', \ldots, d_n)\}\end{aligned}$$

Update Operations (2)

- Modification corresponds to a deletion followed by an insertion, but without interrupting the existence of the tuple.
- SQL has commands for inserting, deleting, and modifying an entire set of tuples (of the same relation).
- Updates can also be combined to a transaction.

Outline

Null Values (1)

- The relational model allows missing attribute values, i.e. table entries can be empty.
- Formally, the set of possible values for an attribute is extended by a new value “null”.
- If R has the schema $(A_1 \colon D_1, \ldots, A_n \colon D_n)$, then

$$\mathcal{I}[R] \subseteq \big(val(D_1) \cup \{null\}\big) \times \cdots \times \big(val(D_n) \cup \{null\}\big).$$

- “Null” is not the number 0 or the empty string! It is different from all values of the data type.

Null Values (2)

- Null values are used in a variety of different situations, e.g.:
  - A value exists, but is not known.
  - No value exists.
  - The attribute is not applicable to this tuple.
  - A value will be assigned later ("to be announced").
  - Any value will do.
- A comittee once found 13 different meanings for a null value.

Null Values (3)
Advantages

- Without null values, it would be necessary to split most relations in many relations (“subclasses”):
  - E.g. `STUDENT_WITH_EMAIL`, `STUDENT_WITHOUT_EMAIL`.
  - Or extra relation: `STUD_EMAIL(SID, EMAIL)`.
  - This complicates queries.
- If null values are not allowed, users will invent fake values to fill the missing columns.

Null Values (4)

Problems

- Since the same null value is used for very different purposes, there can be no clear semantics.
- SQL uses a three-valued logic (true, false, unknown) for evaluating conditions with null values.
- Most programming languages do not have null values. This complicates application programs.

Excluding Null Values (1)

- Since null values lead to complications, it can be specified for each attribute whether or not a null value is allowed.
- It is important to invest careful thought as to where null values are needed.
- Declaring many attributes “not null” will result in simpler programs and fewer surprises with queries.
- However, flexibility is lost: Users are forced to enter values for all “not null” attributes.

Excluding Null Values (2)

- In SQL, one writes `NOT NULL` after the data type for an attribute which cannot be null.
- E.g. `EMAIL` in `STUDENTS` can be null:

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

Excluding Null Values (3)

- In SQL, null values are allowed by default, and one must explicitly request "`NOT NULL`".
- Often only a few columns can contain null values.
- Therefore, when using simplified schema notations, it might be better to use the opposite default:

  $$\texttt{STUDENTS(SID, FIRST, LAST, EMAIL}^{o}\texttt{)}$$

- In this notation, attributes which can take a null value must be explicitly marked with a small "o" (optional) in the exponent.

Outline

Constraints

Overview

- (Integrity) Constraints are conditions that every (valid) database state must satisfy
- E.g. in the SQL `CREATE TABLE` statement, the following types of constraints can be specified:
  - `NOT NULL`: A column cannot be null.
  - Keys: Each key value can appear only once.
  - Foreign keys: Values in a column must also appear as key values in another table.
  - `CHECK`: Column values must satisfy a condition.

Keys: Unique Identification (1)

- A key of a relation R is an attribute/column A that uniquely identifies the tuples/rows in R.
- E.g. if `SID` has been declared as key of `STUDENTS`, this database state is illegal:

STUDENTS

<u>SID</u>	FIRST	LAST	EMAIL
101	Ann	Smith	...
101	Michael	Jones	...
103	Richard	Turner	(null)
104	Maria	Brown	...

Keys: Unique Identification (2)

- If `SID` has been declared as key of `STUDENTS`, the DBMS will refuse the insertion of a second row with the same value for `SID` as an existing row.
- Note that keys are constraints: They refer to all possible DB states, not only the current one.
- Even though in the above database state (with only four students) the last name (`LAST`) could serve as a key, this would be too restrictive: E.g. the future insertion of "`John Smith`" would be impossible.

Keys: Unique Identification (3)

- A key can also consist of several attributes.
  Such a key is called a "composite key".
- E.g. this relation satisfies the key `FIRST`, `LAST`:

STUDENTS			
SID	<u>FIRST</u>	<u>LAST</u>	EMAIL
101	Ann	Smith	...
102	John	Smith	...
103	John	Miller	...

Keys: Minimality (1)

- Let F be a formula specifying that `LAST` is a key.
- Let G be a formula that corresponds to the composed key consisting of `FIRST` and `LAST`.
- Then $F \vdash G$, i.e. every DB state that satisfies the key `LAST` also satisfies the key `FIRST, LAST`.
- Therefore, if `LAST` were declared as a key, it would be not interesting that `FIRST, LAST` also has the unique identification property.

Keys: Minimality (2)

- One will never specify two keys such that one is a subset of the other.
- However, the key "`LAST`" is not satisfied by the example state on Slide 207.
- If the database designer wants to permit this state, the key constraint "`LAST`" is too strong.

- Once the key "`LAST`" is excluded, the composed key "`FIRST, LAST`" becomes again interesting.
- The database designer must now find out whether there could ever be two students in the course with the same first and last name.

Keys: Minimality (3)

- Natural keys nearly always could possibly have exceptions, yet, if exceptions are extremely rare, one could still consider declaring the key:
  - Disadvantage If the situation occurs, one will have to modify the name of one of the students in the database and to manually edit all official documents printed from the database.
  - Advantage Students can be identified in application programs by first name and last name.

Keys: Minimality (4)

- If the designer decides that the disadvantage of the key "`FIRST, LAST`" is greater than the advantage, i.e. that the key is still too strong, he/she could try to add further attributes.
- But the combination "`SID, FIRST, LAST`" is not interesting, because "`SID`" alone is already a key.
- If, however, the designer should decide that "`FIRST, LAST`" is "sufficiently unique", it would be minimal, even if "`SID`" is another key.

Multiple Keys

- A relation may have more than one key.
- E.g. `SID` is a key of `STUDENTS`, and `FIRST, LAST` might be another key of `STUDENTS`.
- One of the keys is designated as the "primary key".
- Other keys are called "alternate/secondary keys".

Keys: Notation (1)

- The primary key attributes are often marked by underlining them in relational schema specifications:

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

STUDENTS			
<u>SID</u>	FIRST	LAST	EMAIL
101	Ann	Smith	$\cdots$
102	Michael	Jones	(null)
103	Richard	Turner	$\cdots$
104	Maria	Brown	$\cdots$

- Usually, the attributes of a relation are ordered such that the primary key consists of the first attributes.

Keys: Notation (2)

- In SQL, keys can be defined as follows:

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

Keys and Null Values

- The primary key cannot be null, other keys should not be null.
- It is as not acceptable if already the "object identity" of the tuple is not known.

Keys and Updates

- It is considered poor style if key attribute values are modified (updated).
- But SQL does not enforce this constraint.

The Weakest Possible Key

- A key consisting of all attributes of a relation requires only that there can never exist two different tuples which agree in all attributes.
- Style Recommendation: Define at least one key for every relation in order to exclude duplicate tuples.

Keys

Summary

- Declaring a set of attributes as a key is a bit more restrictive than the unique identification property:
  - Null values are excluded at least in the primary key.
  - One should avoid updates, at least of the primary key.
- However, the uniqueness is the main requirement for a key.
  Everything else is secondary.

Outline

Foreign Keys (1)

- The relational model has no explicit relationships, links or pointers.
- Values for the key attributes identify a tuple.
- To refer to tuples of R in a relation S, include the primary key of R among the attributes of S.
- E.g. the table `RESULTS` has the attribute `SID`, which contains primary key values of `STUDENTS`.

Foreign Keys (2)

SID in RESULTS is a foreign key referencing STUDENTS:

STUDENTS			
<u>SID</u>	FIRST	LAST	···
101	Ann	Smith	···
102	Michael	Jones	···
103	Richard	Turner	···
104	Maria	Brown	···

RESULTS			
<u>SID</u>	<u>CAT</u>	<u>ENO</u>	POINTS
101	H	1	10
101	H	2	8
102	H	1	9
102	H	2	9
103	H	1	5
105	H	1	7

The constraint that is needed here is that every SID value in RESULTS also appears in STUDENTS.

Foreign Keys (3)

- When `SID` in `RESULTS` is a foreign key that references `STUDENTS`, the DBMS will reject any attempt to insert a solution for a non-existing student.
- Thus, the set of `SID`-values that appear in `STUDENTS` are a kind of "dynamic domain" for the attribute `SID` in `RESULTS`.
- In relational algebra (see Part 4 Relational Algebra), the projection $\pi_{\mathtt{SID}}$ returns the values of the column `SID`. Then the foreign key condition is:

$$\pi_{\mathsf{SID}}(\mathsf{RESULTS}) \subseteq \pi_{\mathsf{SID}}(\mathsf{STUDENTS}).$$

Foreign Keys (4)

- The foreign key constraint ensures that for every tuple t in RESULTS there is a tuple u in STUDENTS such that $t.\mathtt{SID} = u.\mathtt{SID}$.
- The key constraint for STUDENTS ensures that there is at most one such tuple u.

Foreign Keys (5)

- Enforcing foreign key constraints ensures the “referential integrity” of the database.
- A foreign key implements a “one-to-many” relationship: One student has solved many exercises.
- The table `RESULTS` which contains the foreign key is called the “child table” of the referential integrity constraint, and the referenced table `STUDENTS` is the “parent table”.

Foreign Keys (6)

- The table RESULTS contains another foreign key that references the solved exercise.
- Exercises are identified by category (e.g. homework, midterm, final) and number (CAT and ENO):

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
⋮	⋮	⋮	⋮

EXERCISES

CAT	ENO	⋯	MAXPT
H	1	⋯	10
H	2	⋯	10
M	1	⋯	14

Foreign Keys (7)

- A table with a composed key (like `EXERCISES`) must be referenced with a composed foreign key that has the same number of columns.
- Corresponding columns must have the same data type.
- It is not required that corresponding columns have the same name.
- In the example, the composed foreign key requires that every combination of `CAT` and `ENO` which appears in `RESULTS`, must also appear in `EXERCISES`.
- Columns are matched by their position in the declaration:
  E.g. if the key is `(FIRST, LAST)` and the foreign key is `(LAST, FIRST)` insertions will very probably give an error.
- Only keys can be referenced: One cannot reference only part of a composite key or a non-key attribute.

Foreign Keys: Notation (1)

- In the attribute list notation, foreign keys can be marked by an arrow and the referenced relation. Composed foreign keys need parentheses:

$$
\begin{array}{l}
\texttt{RESULTS(}\underline{\texttt{SID}} \rightarrow \texttt{STUDENTS,}\\
\qquad\qquad (\underline{\texttt{CAT}}\texttt{, }\underline{\texttt{ENO}}) \rightarrow \texttt{EXERCISES, POINTS)}\\
\texttt{STUDENTS(}\underline{\texttt{SID}}\texttt{, FIRST, LAST, EMAIL)}\\
\texttt{EXERCISES(}\underline{\texttt{CAT}}\texttt{, }\underline{\texttt{ENO}}\texttt{, TOPIC, MAXPT)}
\end{array}
$$

- Since normally only the primary key is referenced, it is not necessary to specify the corresponding attribute in the referenced relation.

Foreign Keys: Notation (2)

- The above example is untypical because all foreign keys are part of keys. This is not required, e.g.

$$\texttt{COURSE\_CATALOG(}\underline{\texttt{NO}}\texttt{, TITLE, DESCRIPTION)}$$

$$\begin{aligned}&\texttt{COURSE\_OFFER(}\underline{\texttt{CRN}}\texttt{, CRSNO} \rightarrow \texttt{COURSE\_CATALOG, TERM,}\\ &\qquad\qquad \texttt{(INST\_FIRST, INST\_LAST)} \rightarrow \texttt{FACULTY)}\end{aligned}$$

$$\texttt{FACULTY(}\underline{\texttt{FIRST}}\texttt{, }\underline{\texttt{LAST}}\texttt{, OFFICE, PHONE)}$$

Keys: Notation (3)

- In SQL, foreign keys can be defined as follows:

```
CREATE TABLE RESULTS(
            SID      NUMERIC(3)   NOT NULL,
            CAT      CHAR(1)      NOT NULL,
            ENO      NUMERIC(2)   NOT NULL,
            POINTS   NUMERIC(4,1) NOT NULL,
            PRIMARY KEY(SID, CAT, ENO),
            FOREIGN KEY(SID)
                     REFERENCES STUDENTS,
            FOREIGN KEY(CAT,ENO)
                     REFERENCES EXERCISES)
```

More about Foreign Keys (1)

Foreign Keys and Null Values

- Unless a “not null” constraint is explicitly specified, foreign keys can be null.
- The foreign key constraint is satisfied even if the referencing attributes are “null”. This corresponds to “nil” pointer.
- If a foreign key consists of more than one attribute, they should either all be null, or none should be null.

More about Foreign Keys (2)

Mutual References

- It is possible that parent and child are the same table, e.g.

  $\texttt{EMP(}\underline{\texttt{EMPNO}}\texttt{, ENAME, JOB, MGR}^{o}\rightarrow\texttt{EMP, DEPTNO}\rightarrow\texttt{DEPT)}$

  $\texttt{PERSON(}\underline{\texttt{NAME}}\texttt{, MOTHER}^{o}\rightarrow\texttt{PERSON, FATHER}^{o}\rightarrow\texttt{PERSON)}$

- Two relations can reference each other, e.g.

  $\texttt{EMPLOYEES(}\underline{\texttt{EMPNO}}\texttt{, ..., DEPT}\rightarrow\texttt{DEPARTMENTS)}$

  $\texttt{DEPARTMENTS(}\underline{\texttt{DNO}}\texttt{, ..., LEADER}^{o}\rightarrow\texttt{EMPLOYEES).}$

Please Remember

- Foreign keys are not themselves keys!
- Only a key of a relation can be referenced, not arbitrary attributes.
- If the key of the referenced relation consists of two attributes, the foreign key must also consist of two attributes of the same data types in the same order.

Foreign Keys and Updates (1)

The following operations can violate a foreign key

- Insertion into the child table `RESULTS` without a matching tuple in the parent table `STUDENTS`.
- Deletion from the parent table `STUDENTS` when the deleted tuple is still referenced.
- Update of the foreign key `SID` in the child table `RESULTS` to a value not in `STUDENTS`.
- Update of the key `SID` of the parent table `STUDENTS` when the old value is still referenced.

Foreign Keys and Updates (2)

- Deletions from RESULTS (child table) and insertions into STUDENTS (parent table) can never lead to a violation of the foreign key constraint.
- The insertion of dangling references is rejected.
  The DB state remains unchanged.

Foreign Keys and Updates (3)

Reactions on Deletions of Referenced Key Values

- The deletion is rejected.
- The deletion cascades: All tuples from `RESULTS` that reference the deleted `STUDENTS` tuple are deleted, too.
- The foreign key is set to null.
- The foreign key is set to a declared default value.

Foreign Keys and Updates (4)

Reactions on Updates of Referenced Key Values

- The update is rejected. The DB state remains unchanged.
- The update cascades.
- The foreign key is set to null.
- The foreign key is set to a declared default value.

Foreign Keys and Updates (5)

- When specifying a foreign key, decide which reaction is best.
- The default is the first alternative ("No Action").
- At least for deletions from the parent table, all systems should support also the cascading deletion.
- Other alternatives exist only in few systems at the moment.

Outline

Summary

- Relational database schemas
- Relational database states
- Update operations
- Null values
- Keys and foreign keys

Bibliography

- The following list of references is compiled from the open source bibliography available at

  `https://github.com/krr-up/bibliography`

- Feel free to submit corrections via pull requests !

[1] S. Abiteboul, R. Hull, and V. Vianu.
Foundations of Databases.
Addison-Wesley, 1995.

[2] C. Aggarwal, editor.
Data Streams — Models and Algorithms, volume 31 of *Advances in Database Systems*.
Springer-Verlag, 2007.

[3] K. Apt, H. Blair, and A. Walker.
Towards a theory of declarative knowledge.
In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1987.

[4] M. Arenas, L. Bertossi, and J. Chomicki.
Consistent query answers in inconsistent databases.

In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'99)*, pages 68–79. ACM Press, 1999.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation, and Applications.
Cambridge University Press, 2003.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom.
Models and issues in data stream systems.
In L. Popa, editor, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*, pages 1–16. ACM Press, 2002.

[7] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

[8] S. Ceri, G. Gottlob, and L. Tanca.
Logic Programming and Databases.
Springer-Verlag, 1990.

[9] R. Elmasri and S. Navathe.
Fundamentals of database systems.
Addison-Wesley, 1994.

[10] R. Fagin, J. Ullman, and M. Vardi.
On the semantics of updates in databases. preliminary report.
In *Proceedings of the Second ACM Conference SIGACT-SIGMOD*, pages 352–365, 1983.

[11] H. Gallaire, J. Minker, and J. Nicolas.
Logic and databases: A deductive approach.
Computing Surveys, 16(2):153–185, 1984.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
Potassco User Guide.

University of Potsdam, 2 edition, 2015.

[13] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning.
Morgan and Claypool Publishers, 2012.

[14] M. Gelfond and Y. Kahl.
Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.
Cambridge University Press, 2014.

[15] M. Gelfond and V. Lifschitz.
Classical negation in logic programs and disjunctive databases.
New Generation Computing, 9:365–385, 1991.

[16] H. Katsuno and A. Mendelzon.
On the difference between updating a knowledge database and revising it.

In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 387–394. Morgan Kaufmann Publishers, 1991.

[17] V. Lifschitz.
Closed-world databases and circumscription.
Artificial Intelligence, 27:229–235, 1985.

[18] V. Lifschitz.
Nonmonotonic databases and epistemic queries.
In J. Myopoulos and R. Reiter, editors, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 381–386. Morgan Kaufmann Publishers, 1991.

[19] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[20] V. Lifschitz, F. van Harmelen, and B. Porter, editors.

Handbook of Knowledge Representation.
Elsevier Science, 2008.

[21] L. Liu and M. Özsu, editors.
Encyclopedia of Database Systems.
Springer-Verlag, 2009.

[22] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[23] J. Minker, editor.
Foundations of Deductive Databases and Logic Programming.
Morgan Kaufmann Publishers, 1988.

[24] R. Reiter.
On closed world data bases.
In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.

[25] R. Reiter.

Towards a logical reconstruction of relational database theory.
In M. Brodie, J. Myopoulos, and J. Schmidt, editors, *On conceptual modeling: Perspectives from Artificial Intelligence, Datbases and Programming Languages*, pages 191–233. Springer-Verlag, 1984.

[26] R. Reiter.
On asking what a database knows.
In J. Lloyd, editor, *Computational Logic*, pages 96–113.
Springer-Verlag, 1990.

[27] J. Ullman.
Principles of Database Systems.
Computer Science Press, Rockville MD, 1982.

[28] J. Ullman.
Principles of Database and Knowledge-Base Systems.
Computer Science Press, 1988.