The Relational Model

- After completing this chapter, you sould be able to
  - ▷ explain the concepts of the **Relational Model**,
    - Schemas, state, domains
  - ▷ explain applications and problems of **null values**,
  - ▷ explain **integrity constraints** and their importance,
  - ▷ explain the meaning of **keys** and **foreign keys**,
  - ▷ read various notations for **relational schema**,
  - ▷ develop simple relational schemas.

The Relational Model

Overview

Example Database (1)

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

Example Database (2)

- Columns in table STUDENTS:
  - SID: "student ID" (unique number)
  - FIRST, LAST, EMAIL: first and last name, email address (may be null).
- Columns in table EXERCISES:
  - CAT: category (H: Homework, M/F: midterm/final exam)
  - ENO: exercise number within category
  - TOPIC, MAXPT: topic of exercise, maximum number of points.
- Columns in table RESULTS:
  - SID: student who handed in exercise (references STUDENTS)
  - CAT, ENO: identification of exercise (references EXERCISE)
  - POINTS: graded points

Data Values (1)

- All table entries are **data values** which conform to some given selection of **data types.**
- The set of available data types is determined by the RDBMS (and by the supported version of the SQL standard).
- Examples of types:
  strings, numbers (of different lengths and precision), data and time, money, binary data.
- The relational model itself is **independent** of any specific selection of data types.

Data Values (2)

- **Extensible DBMS** allow the user to define **new data types** (*e.g.*, multimedia or geometric data types).

  Two ways to offer such extensibility: (1) Link **external procedures** (*e.g.*, written in C) to the DBMS kernel. (2) DBMS comes with a built-in programming language (for server-side **stored procedures**), and types and operations defined in this language can be used in table declarations and queries. Real extensibility should also permit to define **new index structures** and to inform the **query optimizer** about properties of the new data types.

- This extensibility is an important feature of modern **Object-Relational DBMS** (*data blades*).

Data Values (3)

- The following definitions assume that
  - ▷ a set $\mathcal{D}$ of **data type names** (or **data types**) is given, and
  - ▷ for each $D \in \mathcal{D}$, a set $val(D)$ of **possible values** of that data type D. The set $val(D)$ is also known as the **domain** of D.
- For example:

$$val(\texttt{NUMERIC(2)}) = \{\texttt{-99}, \ldots, \texttt{99}\} \ .$$

  The distinction between name and interpretation (syntax and semantics) is typical for a formalization of knowledge (as done, *e.g.*, in mathematical logic).

Domains (1)

- The columns `ENO` in `RESULTS` and `ENO` in `EXERCISES` should have the same data type.

  The same holds for `EXERCISES.MAXPT` and `RESULTS.POINTS`.
- In SQL, the user may define **application-specific domains** as names for (subsets of) standard data types:

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

- We may even add the constraint that the exercise number has to be positive:

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

Domains (2)

- 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).

Mixing up real-world objects

"Which homework has a number that is the same as its number of points?"

- SQL's `CREATE DOMAIN` could be used to assign different domains to `EXERCISES.ENO` and `EXERCISES.MAXPT`.

  SQL does not forbid inter-domain comparisons (compare with type declarations in programming languages), but now the schema captures the real world a bit more precisely.

Atomic Attribute Values (1)

- The relational model treats the single column entries as **atomic.**
- Column entries may have no structure or contain "multiple values".
- In contrast, the NF^2 (**N**on-**F**irst **N**ormal **F**orm) data model recursivle allows table entries to be complete tables themselves.

Atomic Attribute Values (2)

- Example of an NF^2 table (not permitted in the classical relational model, not treated in this course [2]):

NF^2 table

HOMEWORKS

NO	TOPIC	MAXPOINTS	SOLVED_BY
1	Rel.Alg.	10	STUDENT: Ann Smith, Michael Jones; POINTS: 10, 9
2	SQL	10	STUDENT: Michael Jones, Richard Turner; POINTS: 9, 10

Mapping NF^2 to RDBMS?

Is this NF^2 table representable in a standard RDBMS?

Atomic Attribute Values (3)

- Of course, even in a RDBMS implementing the classical relational model, if, *e.g.*, `DATE` is a supported data type, the system will implement operations to extract day, month, year.
- However, this extraction happens on the **level of data types**, *not* on the level of the relational model itself.

Encoding value lists using the `STRING` datatype

Encode list of integers $x_1, x_2, \ldots, x_n$ using a single string of the form "$x_1 ; x_2 ; \ldots ; x_n$".
The string is nevertheless atomic with respect to the relational model: SQL provides no means access x_1, for example. (A specific substring operation will support this.)

Relational DB Schemas (1)

Relation schema

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

- a (finite) sequence $A_1, \ldots, A_n$ of **attribute names,**

  The names must be distinct, *i.e.*, $A_i \neq A_j$ for $i \neq j$.
- for each attribute A_i a data type (or **domain**) D_i.

  Let $dom(A_i) := val(D_i)$ (set of possible values for A_i).
- A relation schema may be written as

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

Relational DB Schemas (2)

Relational database schema

A **relational database schmema** $\mathcal{S}$ 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).

  For example, keys and foreign keys.
- In summary, $\mathcal{S} = (\{R_1, \ldots, R_m\}, sch, \mathcal{C})$.

Relational DB Schemas (3)

- Consequences of these definitions:
  - ▷ Column names must be unique **within a table**.
  - ▷ Different tables may have columns sharing the same name (*e.g.*, ENO in the example).

    The columns may even have different data types (bad style).
  - ▷ 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 single DB schema, table names must be unique.

Schemas: Notation (1)

- Consider the table

EXERCISES

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

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

Relation schema in SQL

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

Schemas: Notation (2)

- SQL `CREATE TABLE` statements represent a rather poor way to communicate schemas (from human to human).
- Often it is useful to talk about abstractions of the schema (where, for example, the column data types are not important).
- One concise notation is the following:

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

- Also widely in use: **sketch of the table header**:

EXERCISES			
CAT	ENO	TOPIC	MAXPT

Tuples (1)

- Tuples are used to formalize table rows.
  - ▷ An n-tuple is a **sequence** of n values. Pairs are 2-tuples.

Tuples

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 $d_i \in val(D_i)$, *i.e.*, $t \in val(D_1) \times \cdots \times val(D_n)$.

- Given a tuple, we write $t.A_i$ (or $t[A_i]$) for d_i in column A_i.
- One tuple in table `EXERCISES` is `('H', 1, 'Rel.Alg.', 10)`.

Database States (1)

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

Database state

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

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

i.e., $I(R_i)$ is a relation in the mathematical sense.

- A database state interprets the symbols in the DB schema: it maps relation names to relations.

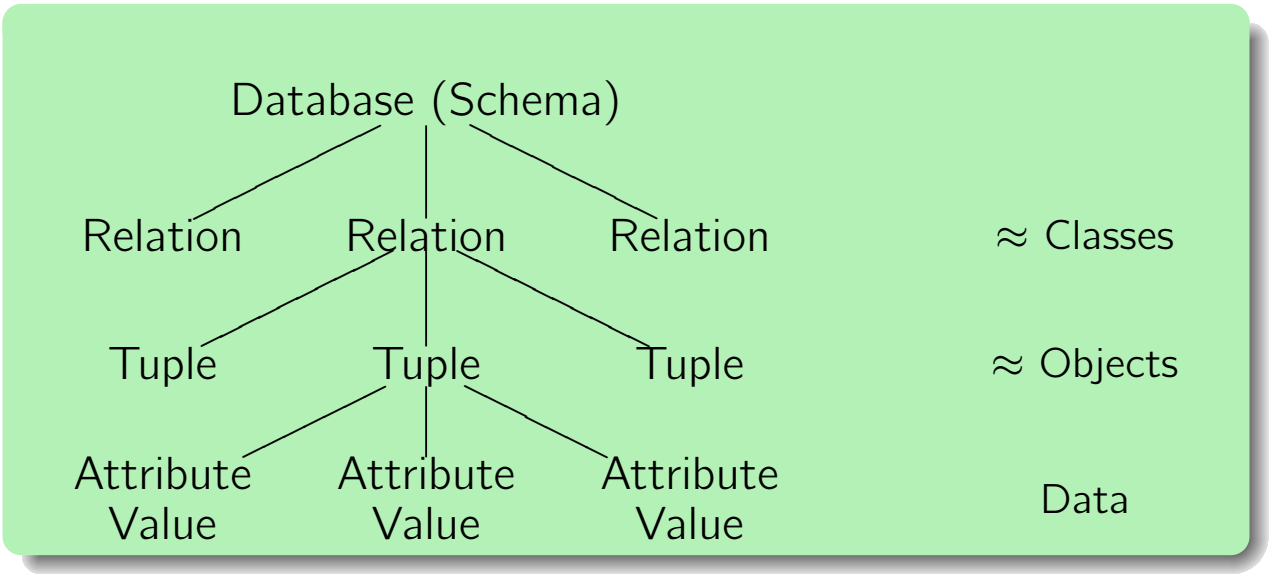
Database States (2)

- Relations are **sets of tuples.**
  - ▷ The sequence of tuples in a relation is **undefined.**

    In this sense, the tabular representation is misleading. There is no first row, second row, **etc.** The storage manager of the RDBMS can exploit this property of the relational model to optimize tuple placement within a relation (file).
  - ▷ (Relations may be sorted on output.)
  - ▷ There are **no duplicate tuples.**

    All RDBMS allow the presence of duplicate tuples as long as no key is defined on a relation (see below). An alternative formalization of relations as **bags of tuples** could capture this.

Summary



Update Operations (1)

- **Updates** transform a DB state I_{old} into a DB state I_{new}.

  The basic **update operations** of the relational model are:

  ▷ **Insertion** (of a tuple into a relation):
  $$I_{\text{new}}(R) := I_{\text{old}}(R) \cup \{(d_1, \ldots, d_n)\}$$

  ▷ **Deletion** (of a tuple from a relation):
  $$I_{\text{new}}(R) := I_{\text{old}}(R) - \{(d_1, \ldots, d_n)\}$$

  ▷ **Modification** (of a tuple):
  $$I_{\text{new}}(R) := I_{\text{old}}(R) - \{(d_1, \ldots, d_n)\} \cup \{(d'_1, \ldots, d'_n)\}$$

Update Operations (2)

- Modification corresponds to a deletion followed by an insertion, but without interrupting the existence of the updated tuple.

  Database constraints might require that a tuple with certain attribute values for the key attributes exists.

- SQL provides commands for inserting, deleting, and modifying an entire **set of tuples** (of the same relation).
- (Typically, updates are combined into a single transaction.)

The Relational Model

Overview

Null Values (1)

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

$$I(R) \subseteq (val(D_1) \cup \{\text{null}\}) \times \cdots \times (val(D_n) \cup \{\text{null}\}) \ .$$

- "Null" is *not* the number 0 or the empty string.
  A null value is different from all values of any data type.

Null Values (2)

- Null values are used to model a variety of real-world scenarios:[3]
  - ▷ **A value exists (in the real world), but is not known.**
    In table `STUDENTS`, `EMAIL` might be missing for a student.
  - ▷ **No value exists.**
    A student might not have an e-mail address; not everyone has a first and last name.
  - ▷ **The attribute is not applicable for this tuple.**
    Some exercises are for training only: no points will be given.
  - ▷ **Any value will do.**

[3] A committee once found 13 different meaning for null values.

Null Values: Advantages (3)

- Without null values, it would be necessary to **split a relation** into many, more specific relations ("subclasses"):
  - ▷ Examples: `STUDENT_WITH_EMAIL`, `STUDENT_WITHOUT_EMAIL`.
  - ▷ Alternatively: introduce an additional relation with schema `STUD_EMAIL (SID, EMAIL)`.
  - ▷ This complicates queries.

    Joins and union operations are needed (upcoming).
- If null values are not allowed, users will invent **fake values** to fill the missing columns.

Fake values

Why are fake values a bad idea in database design?

Null Values: Problems (5)

- 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 majority), the outcome is often surprising—common equivalences do not hold.

- Most programming languages do not know about null values. This complicates application programs.

  When an attribute value is read into a program variable, an explicit null value check and treatment is required (SQL: indicator variables).

Excluding Null Values

- Since null values may lead to complications, SQL provides control whether an attribute value may be null or not. By default, null values are allowed.
- Declaring many attributes as `NOT NULL` leads to simpler application programs and fewer surprises during query evaluation.

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)          )
```

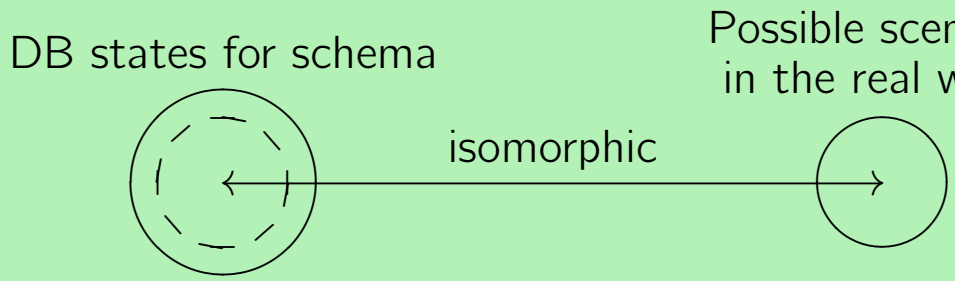

The Relational Model

Overview

1. Relational Model Concepts: Schema, State
2. Null Values
3. Constraints: General Remarks
4. Key Constraints
5. Foreign Key Constraints

Valid Database States (1)

- Primary goal of DB design: the database should be an **image of the relevant subset of the real world.**
- The plain definition of tables and columns, however, often allows **too many** (meaningless, illegal) database states:



Valid Database States (2)

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	1990	Washington	...

- Customer numbers must be unique.
- The year of birth must be greater than 1870.
- Customers must be at least 18 years old.

Constraints (1)

- **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 as part of the database schema (component $\mathcal{C}$).
- The DBMS will **refuse any update** which would lead to a database state I_{new} that violates any of the constraints.

Constraints (2)

- 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 are required to satisfy a given predicate.

    SQL allows for inter-column `CHECK` constraints.

Constraints (3)

Which of the following are constraints?

- It is possible that a student gets 0 points for a solution.
- A student can get at most 3 points more than the maximum number of points stored in `EXERCISES` (extra credit).
- The attribute `CAT` in `EXERCISES` can only have the values `H`, `M`, `F`.
- The `CAT` value in `EXERCISES` means: `H` for homework, `M` for mid-term exam, `F` for final exam.
- Student IDs should be unique.

Trivial/Implied Constraints

- A **trivial constraint** is a condition that is always satisfied.
- A constraint A logically **implies** a constraint B if whenever A is true, B is also true ($A \Rightarrow B$).

  In other words, the database states satisfying A are a subset of the states satisfying B.

- Example:

  A : "Every instructor teaches one or two courses."

  B : "Instructors can teach at most four courses."

- Trivial/implied constraints should *not* be specified.

  Adds complication and checking overhead, but does not restrict the set of valid database states.

Summary

- **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 fulfills certain properties.

The Relational Model

Overview

1. Relational Model Concepts: Schema, State
2. Null Values
3. Constraints: General Remarks
4. Key Constraints
5. Foreign Key Constraints

Keys (1)

Key

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: if $t.A = u.A$ then $t = u$.

- 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 (2)

- 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.
  - ▷ 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.

Keys (3)

- 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, u which agree in *both* attributes (*i.e.*, $t.A = u.A \wedge t.B = u.B$). Columns may in agree A *or* B, though.

- Example: this relation satisfies the composite key `FIRST,LAST`:

STUDENTS

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

Keys (4)

- **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.
- Example: the relation on the previous slide,
  - ▷ **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)?

Keys (5)

- **Minimality of keys:**
  - ▷ If attribute A has been declared as key for relation R, then **any superset** K of attributes that includes A will automatically also have the unique identification property. K is also known as **superkey.**
  - ▷ The usual definition of keys requires that the set of key attribute $\{A_1, \ldots, A_k\}$ is **minimal.**

    No A_i may be removed from the set without destroying the unique identification property.

Keys (6)

- **Multiple keys:**
  - ▷ A relation may have more than one key. A relation may also have **more than one minimal key.**
- In the relational model, one key is designated as the **primary key.** A primary key **cannot be null.**
- All other keys are referred to as **alternate** or **secondary keys.**
- It is good design practice to define a primary key that consists of a **single (simple) attribute** only and **will never be updated.**

Primary keys

What might be the rationale behind this design advice?

Keys (7)

- The primary key attributes are often marked by **underlining** them in the relation schema specifications:

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

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

- It is common practice to reorder the attributes of a relation such that key attributes come first in attribute order.

Key Quiz

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
Jan. 19	20:00	23:00	ANNO	Jever

- What would be correct 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

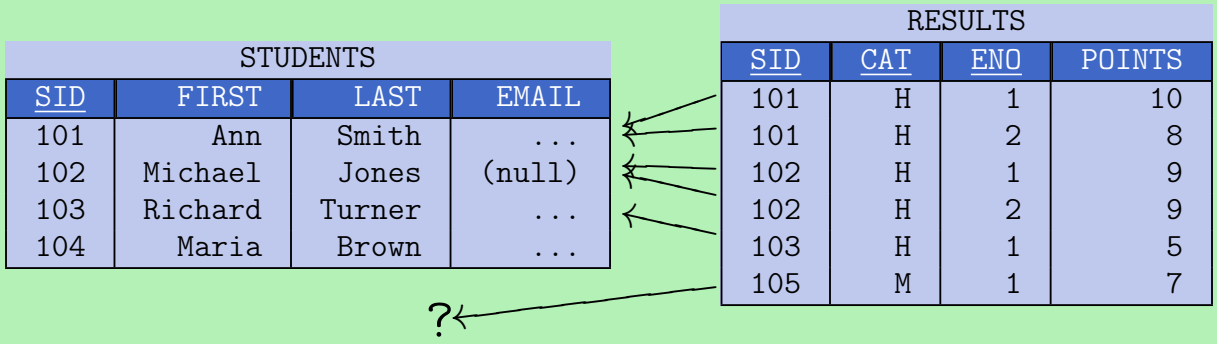
Foreign Keys (1)

- The relational model does **not provide explicit relationships, links, or pointers.**
- The values for the key attributes uniquely identify a tuple.

  The key attributes values may serve as **logical tuple addresses.**
- To refer to tuples of R in a relation S, add the **primary key attributes of** R to the attributes of S.
- Such a reference is only "stable" if the (logical) address of a tuple does not change (*i.e.*, if the key attributes are not updated).

Foreign Keys (2)

SID in RESULTS is a foreign key referencing STUDENTS



- A foreign key implements a **one-to-many relationship**.
- We need some kind of **existence guarantee** for key values in STUDENTS.

Foreign Keys (3)

- When RESULTS.SID is a **foreign key** that references table STUDENTS, the DBMS will reject any update to insert a solution (tuple in RESULTS) for a non-exisstent student.
- The set of SID value actually appearing in STUDENTS are a kind of **dynamic domain** for the attribute REULTS.SID.
- In relational algebra (next chapter), the projection $\pi_A(R)$ returns the set of values in column A in relation R. The **foreign key constraint** thus is:

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

Key constraints and π

Can π also express the key constraint "A is key in R"?

Foreign Keys (4)

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

  Pairs of such tuples t, u can be brought together by a relational algebra operation called "Join" (next chapter). This corresponds to the **dereferencing** of pointers in other models.

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

- The key constraint in STUDENTS ensures that there is at most one such tuple u.

Foreign Keys (5)

- A table with a composite key (like EXERCISES) 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.

- **Only keys may be referenced** (primary or secondary).

  References to partial composite keys or to non-key attributes are not permitted.

-  **Foreign keys are not themselves keys.**

Foreign Keys: Notation (6)

- Typically, foreign keys are denotes with arrows ($\rightarrow$) in the relation schema, composite keys appear in parentheses:

```
  RESULTS (SID → STUDENTS,(CAT, ENO) → EXERCISES,
           POINTS)
 STUDENTS (SID,FIRST,LAST,EMAIL)
EXERCISES (CAT,ENO,TOPIC,MAXPT)
```

- Since typically only the primary key is referenced, it is sufficient to simply list the target relation.

Foreign Keys (7)

- Unless a `NOT NULL` constraint is present, **foreign keys may be null.**

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

- Foreign key references may be **mutual**, *i.e.*, intra-table references are allowed:

Mutual references

```
EMP(EMPNO,ENAME,JOB,MGR → EMP,DEPTNO → DEPT)
PERSON(NAME,MOTHER → PERSON,FATHER → PERSON)
```

Mutual references and updates

How can tuples be inserted into tables `EMP` or `PERSON`?

Foreign Keys and Updates

- Once the foreign key has been declared in the DB schema, the following **update operations violate the foreign key constraint:**

  ① **Insertion** into table `RESULTS` without a matching tuple in table `STUDENTS`.
  - ▷ The DBMS **rejects** such an insertion.

  ② **Deletion** from table `STUDENTS` when the deleted tuple is still referenced in `RESULTS`.
  - ▷ The DBMS **rejects** the deletion, or
  - ▷ the deletion **cascades:** tuples in `RESULTS` referencing the deleted tuple will be also be deleted, or
  - ▷ the foreign key is **set to null** in `RESULTS`.