Part 4: ER-Model II

Elmasri/Navathe:

- Ch. 3, “Data Modeling Using the Entity-Relationship Model”

I have also used the following sources:

- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Ed., Ch. 2, “Entity-Relationship Model”.
- Ramakrishnan: Database Management Systems, Mc-Graw Hill, 1998, Ch. 14, “Conceptual Design and the ER-Model”
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German), Teubner, 1997.
- Kemper/Eickler: Datenbanksysteme (in German), Ch. 2, Oldenbourg, 1997.
- Teorey: Database Modeling and Design, Third Edition, 1999.
- Barker: CASE*Method, Entity Relationship Modelling, Oracle/Addison-Wesley, 1990.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.

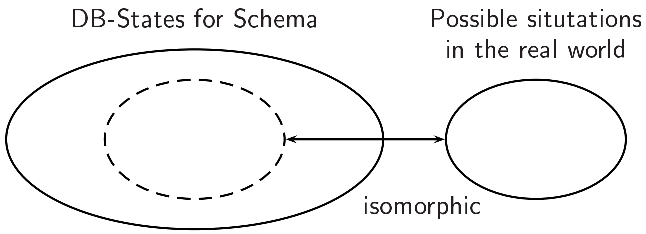
Objectives

- You should know what constraints are and why they are important.
- You should be able to explain key constraints and cardinalities.
- You should be able to choose the right keys and cardinalities for an application.
- You should be able to compare different keys and cardinalities (e.g. which is the stronger restriction?).
- You should be able to relate (min,max)-cardinalities to “one-to-many, mandatory-to-optional” etc. cardinalities.

Overview

Valid Database States (1)

- Goal of DB-Design: DB-States should be image of real world.
- But the definition of entity types, attributes, and relationships allows still many database states which do not correspond to states of the real world:



Valid Database States (2)

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

Valid Database States (3)

- You cannot expect a protection against entering wrong data.

  Unless you require two independent persons to enter the same data, the supervisor to sign it off, etc.
- I.e. it might be that the DB state corresponds a different situation of the real world than the actual one.

  In this case only the DB state is wrong (not the schema).
- You can expect a protection against entering data which does not make sense/is illegal (impossible in the real world).
- Such data would make DB state inconsistent with our general understanding of the real world, which may have all kinds of strange effects.

  If this is possible, the DB schema is wrong.

Constraints (1)

- Integrity Constraints (or short “Constraints”) are conditions which every database state must satisfy.
- This restricts the set of possible database states.

  Ideally to images of possible situations of the real world.
- The DBMS will refuse any change which would violate such a constraint. So invalid states are excluded.
- Constraints are also called “business rules”.

  Certain rules, e.g. that customers must be 18 years old or that advanced cryptology software is not sold to customers outside the US, can be enforced by not allowing to enter such an order into the database. A state violating the business rules is considered as invalid.

Constraints (2)

Language for specifying constraints:

- The ER-model has a special syntax for certain common types of constraints: keys and cardinalities.
- There are logical languages which allow users to define arbitrary constraints in the ER-model formally.
- However, for our purposes it is sufficient to define constraints (except keys and cardinalities) in natural language.

  Of course, the DBMS cannot understand and thus cannot enforce constraints in natural language. However, they can still be an important documentation for the development of application programs (should contain checks). We will treat constraints more formally in the relational model.

Reasons for Constraints (1)

- Constraints give some protection against data input errors.

  Obviously wrong/meaningless values are excluded.
- Constraints document knowledge about database states.
- Queries and programs become simpler if the programmer is not required to handle the most general / extreme cases (i.e., cases where the constraint is not satisfied).

  E.g. a program for printing course information becomes simpler if we know that each course has only one instructor.
- Constraints also help to enforce laws / company standards.
- If the schema should contain redundant information, we must use constraints to ensure that the copies agree.

Reasons for Constraints (2)

- If a programmer assumes that the data fulfills some condition, but it actually does not, strange effects can occur:
  - E.g. the programmer assumes that a certain column cannot contain null values.
  - So he/she uses no indicator variable when fetching data.
  - As long as there are no null values, this works.
  - But if no constraint prevents this, after some time somebody will enter a null value.
  - Then the program will break with a not user-friendly Oracle error message.
- Every assumption must be secured by a constraint.

Constraints and Exceptions (1)

- Constraints cannot have any exceptions.
- A good DBMS will reject any attempt to enter data which violates a specified constraint.
- It is a common experience that there will be later exceptional situations in which the database system seems unflexible because of the specified constraints.
- The introduction of some computerized system always changes the real world. So the decreased flexibility is not necessarily fatal, if the users can accept them.

Constraints and Exceptions (2)

- On the other hand, without constraints, the database contents will soon be chaotic.
- Thus, selecting constraints might be difficult.
- You should define as constraints only conditions which are unquestionable.
- “Mostly true” conditions may be mentioned as comments.
  - Some such conditions might be interesting in physical database design.
  - Also, application programs might ask the user for an additional confirmation when unlikely data is entered.

    E.g. a new customer older than 100 years.

Summary

- The real world is the measure of correctness for our schema: DB states should correspond to states of the real world.
- The three basic design errors are:
  - There are situations in the real world which do not correspond to a database state.

    I.e. you cannot enter data which actually occurs.
  - You cannot formulate a legal question about the real world as a query to the database.

    I.e. information is missing in the database.
  - Database states are possible which do not correspond to a legal state in the real world.

    I.e. you can enter invalid/illegal data.

Overview

Keys (1)

- A key of an entity-type E is an attribute which uniquely identifies the entities of this type.
- So there may never be two different entities which have the same value for the key attribute.
- For example, the social security number is a key of persons: No two different persons can have the same SSN.

  I have heard that there are very rare cases where two persons have the same SSN. If this should be true, the SSN is no key.
- The last name is not a key of our faculty members since we have Ida Flynn and Roger Flynn (two entities with the same value for the attribute “last name”).

Keys (2)

- It is also possible to declare the combination of two or more attributes as a key.
- In that case, it is only forbidden that two entities agree in all these attributes.

  I.e. we must be able to distinguish the entities by the value of at least one of the key attributes.
- E.g. if we declare first name and last name together as key of our faculty members, it is legal to have two professors with the same last name if they only differ in the first name.

  We can also have at the same time two other faculty members with the same first name when these two have different last names.

Keys (3)

- Keys are constraints: They must be satisfied for all DB states.

  Even if we had not two faculty members with the same last name now, declaring "Last Name" as a key means that we will never be able to enter two such faculty members.
- Of course, it is conceivable that we might get two faculty members with the same first and last name. Does this mean that first and last name together are also not acceptable?
- It may actually be a business rule that such a situation cannot happen: When we print lists of courses, we do not print the SSN, but the first and last name of the instructor.

  Students have a right to know who their instructor is.
- Exercise: What about students?

Keys (4)

Formal Semantics of the Key Constraint:

- So if an attribute A has been declared as a key of an entity-type E, then a database state $\mathcal{S}$ is only valid if

  For all entities $\mathsf{e}_1, \mathsf{e}_2 \in \mathcal{S}(\mathsf{E})$:

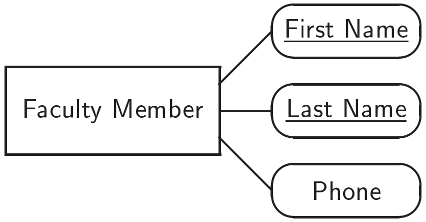
  $$\big(\mathcal{S}(\mathsf{A})\big)(\mathsf{e}_1) = \big(\mathcal{S}(\mathsf{A})\big)(\mathsf{e}_2) \implies \mathsf{e}_1 = \mathsf{e}_2.$$
- If the key is a combination of attributes $\mathsf{A}_1, \ldots, \mathsf{A}_k$, then different entities must differ in the value of at least one A_i:

  $$\mathsf{e}_1 \neq \mathsf{e}_2 \implies \big(\mathcal{S}(\mathsf{A}_1)\big)(\mathsf{e}_1) \neq \big(\mathcal{S}(\mathsf{A}_1)\big)(\mathsf{e}_2) \quad \text{or} \ldots \text{or} \quad \big(\mathcal{S}(\mathsf{A}_k)\big)(\mathsf{e}_1) \neq \big(\mathcal{S}(\mathsf{A}_k)\big)(\mathsf{e}_2).$$

Keys (5)

Graphical Syntax:

- Keys are marked in ER-diagrams by underlining the attributes which form a key:



- Only entity types can have key attributes. You cannot declare keys for relationships (but see cardinalities below).

Keys (6)

Multiple Keys:

- An entity type can have more than one key (e.g. if we store also the SSN of the faculty members).
- The graphical syntax allows only to specify a single key for each entity type.

  If we underlined also the SSN, then the three attributes {FirstName, LastName, SSN} would form together a key.
- You should select one key as the "primary key".

  Such a unique "primary key" is needed for the translation into the relational model. It is recommended to select a key which consists only of a single, short attribute (if available).

Keys (7)

Implication between Key Constraints:

- Key constraints become stronger (i.e. eliminate more database states) when less attributes participate in them.

  “Less” means here $\subseteq$, not a smaller number of attributes.
- E.g. if {LastName} is a key, then {FirstName, LastName} automatically satisfies the unique identification property.
- A constraint $\mathcal{C}_1$ can logically imply another constraint $\mathcal{C}_2$: Whenever a state satisfies $\mathcal{C}_1$, it also satisfies $\mathcal{C}_2$.

  Then the stronger constraint $\mathcal{C}_1$ is sufficient.
- However, here {LastName} is too strong: It eliminates also states which correspond to possible real world states.

Keys (8)

Minimality of Keys:

- Many authors require that a key must not only uniquely identify entities, but also must be a minimal set of attributes with this property (with respect to $\subseteq$).

  I.e. you cannot take one attribute away from the key without destroying the unique identification property.
- However, this minimality requirement is
  - not an integrity constraint, which restricts the possible database states,
  - but an assertion on the real world, that there are situations where a smaller attribute set would not suffice.

Keys (9)

Importance of Keys:

- The ER-model does not require that a key is declared for each entity-type (since entities have an “object identity”).
- However, when we want to translate an ER-schema into the relational model, we need a key for every entity type.

  But see below for “weak entity types”.
- If there is no natural key, often identifying numbers are added (e.g. “order number”, “course number”).
- Often, thinking about a key helps to understand the real meaning of an entity type better.

  E.g. course offering vs. abstract concept of a course (topic).

Keys (10)

- The selection of a natural key (not an artificial identification number) is quite difficult.

  There are always more exceptions than you think.
- Even if an identification number is used, you should think about the identification mechanisms which are used in the application programs.

  It is dangerous if different application programs use different identification mechanisms for the same entity type.
- The situation is especially difficult if the non-presence in the database is also used to make statements about other entities in the real world.
- Good keys should not change over time (no updates).

Avoidance of Duplicates

- The purpose of keys is not only the identification of entities (then artificial numbers would always be a good choice).
- Keys should also help to avoid duplicates in the database.
- Often, duplicates are not exact copies: For instance, other shorthands or capitalization is used.

  Of course, the pure concept of a key then does not help. Furthermore, you do not really need a constraint — some warning would suffice.
- Think about possibilities for detecting duplicates before you start writing application programs.

Exercises (1)

- Keys in the relational model are very similar (no two rows with the same key values). Select a key:

HOMEWORK_RESULTS

Student	Exercise	Points
John Smith	1	10
John Smith	2	12
Maria Brown	1	9

- Give an example for a violation of this key (add a row):

- Could "Points" also be declared as key?

Exercises (2)

- Consider an appointment calendar:

APPOINTMENTS

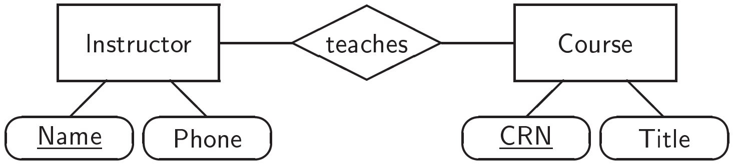
Date	From	Until	Room	What
Jan. 19	10:00	11:00	IS 726	Michael
Jan. 19	14:00	15:00	IS 726	Siripun
Jan. 19	18:00	20:50	CRAWF 169	INFSCI 2710

- What would be a correct key? Are there alternatives?
- Give an example for a non-minimal key (superkey).
- Are additional constraints needed? I.e. can there be invalid database states, even if the key is satisfied?

Overview

1. Constraints I: General Remarks
2. Constraints II: Keys
3. First Look at the Translation into Tables
4. Constraints III: Cardinalities

Example (1)



- Basically, every entity type is translated into a table with a column for every attribute of the entity type.
- Relationships can sometimes be translated into extra columns (referencing another table), and sometimes into tables of their own.

Example (2)

- If every Course is taught by only one Instructor, we simply put the name of the instructor (the key of the “Instructor” entity type) as an additional column into the course table:

Instructor

<u>Name</u>	Phone
Brass	9404
Spring	9429

Course

<u>CRN</u>	Title	Name
20727	DB	Brass
42232	DS	Brass
31822	IS	Spring

- Then the relationship “teaches” between instructor and course is represented, and no extra table is needed.

Example (3)

- Putting instead the key of the “Course” entity type into the table for the “Instructor” entity type would be wrong:

Instructor

<u>Name</u>	Phone	CRN
Brass	9404	20727, 42232
Spring	9429	31822

Course

<u>CRN</u>	Title
20727	DB
42232	DS
31822	IS

- The relational model allows only atomic attribute values.
  In our example, every course is only taught by one instructor, but one instructor can teach multiple courses.

Example (4)

- If there can be team-taught courses, i.e. one course can have more than one instructor (and one instructor can tach more than one course), we need an extra table for the relationship:

Instructor

<u>Name</u>	Phone
Brass	9404
Spring	9429

teaches

<u>Name</u>	<u>CRN</u>
Brass	20727
Brass	42232
Spring	42232
Spring	31822

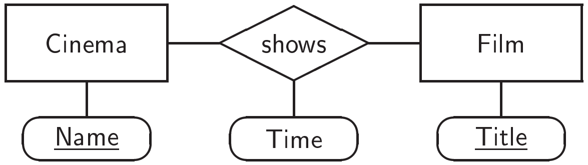
Course

<u>CRN</u>	Title
20727	DB
42232	DS
31822	IS
12998	IR

- The key of the relationship table consists of the keys of the two entity types (combined).

Example (5)

- Relationship attributes are translated into additional columns of the relationship table (or the entity table):



shows		
Name	Title	Time
CinemaX	Star Wars	3pm
CinemaX	Star Wars	6pm

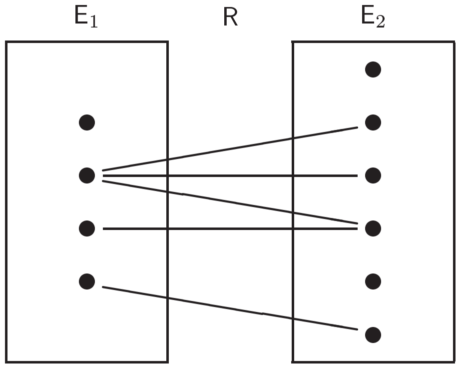
Key Constraint Violated!

Overview

1. Constraints I: General Remarks
2. Constraints II: Keys
3. First Look at the Translation into Tables
4. Constraints III: Cardinalities

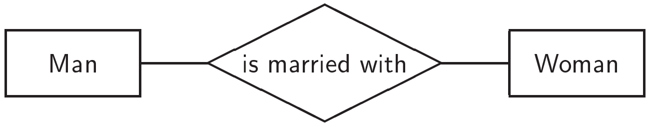
Cardinalities (1)

General Relationship:



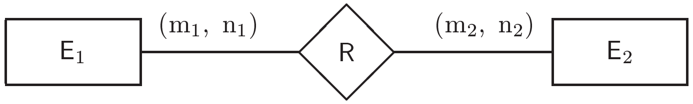
Cardinalities (2)

- In general, there is no restriction on how often an entity participates in a relationship.
- An entity can be connected to one entity of the other type, to more than one, or it can have no partner at all.
- However, often we have some knowledge of how many E_2-entities an E_1-entity can be related.



Cardinalities (3)

The (min,max)-Notation:



- In this notation, we specify an interval for the number of outgoing edges of every entity.
- I.e. an entity of type E$_1$ may be related to between m_1 and n_1 entities of type E$_2$.
- Correspondingly, m_2 is the minimal number of E$_1$ entities, to which an E$_2$ entity must be related, and n_2 is the maximal number to which it may be related.

Cardinalities (4)

Formal Semantics:

- I.e. for every DB state $\mathcal{S}$ and every entity $e_1 \in \mathcal{S}(E_1)$:

$$m_1 \;\leq\; |\{e_2 \in \mathcal{S}(E_2) \mid (e_1, e_2) \in \mathcal{S}(R)\}| \;\leq\; n_1.$$

- I.e. for every DB state $\mathcal{S}$ and every entity $e_2 \in \mathcal{S}(E_2)$:

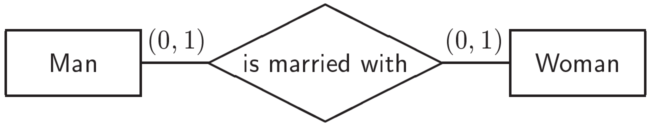
$$m_2 \;\leq\; |\{e_1 \in \mathcal{S}(E_1) \mid (e_1, e_2) \in \mathcal{S}(R)\}| \;\leq\; n_2.$$

Extension:

- We allow to use $*$ as the maximum, which means "no limit".
- So $(0, *)$ means no restriction at all (min: 0, max: ∞). Therefore this cardinality is often not shown in ER-diagrams.

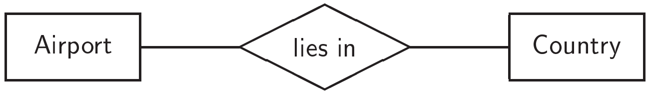
Cardinalities (5)

Example:



Exercise:

- What would be the right cardinalities in this example?



- Suppose we want to store which faculty member can teach which courses (i.e. knows the subject). Define cardinalities.

How to Select Cardinalities

- Draw an example of a valid database state: Some entities of type E$_1$, some of type E$_2$, and edges between related entities.
- Now count the outgoing edges for all entities of type E$_1$.
- E.g. in the relationship depicted on slide 4-35, there are E$_1$-entities with 0, 1, and 3 outgoing edges.
- This means that the strongest cardinality restriction you can define is $(0, 3)$. This is satisfied by the example state.
- But your knowledge of the application might lead to weaker restrictions (i.e. larger intervals), e.g. $(0, 5)$ or even $(0, *)$.
  (a, b) is weaker than (c, d) if $a \leq c$ and $d \leq b$.
- Do the same with E$_2$.

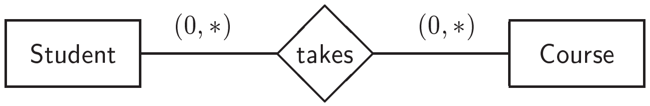
Common Cases (1)

- Normally, the minimum cardinality will be 0 or 1, and the maximum cardinality will be 1 or $*$.
  - So only $(0, 1)$, $(1, 1)$, $(0, *)$, $(1, *)$ are common in practice.
- However, in order to understand a relationship, one must know the cardinality specifications on both sides.
- Often, one uses the maximum cardinalities to distinguish between “many to many”, “one to many”/“many to one”, and “one to one” relationships.
- The minimum cardinalities define whether we have total/mandatory or partial/optional participation.
  - Total: Every entity must participate in the relationship.

Common Cases (2)

“Many to Many” Relationships:

- Both maximum cardinalities “$*$”
  - The minimum cardinalities can be 0 or 1 ($\rightarrow$ participation).

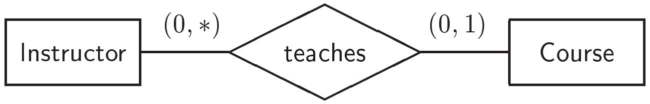


- One student can be enrolled in many courses, and one course can have many students.
- This is the most general case.
- When translated in the relational model, “many to many” relationships will need an extra table.

Common Cases (3)

“One to Many” Relationships:

- Maximum cardinality 1 on the “many” side and $*$ on the “one” side. (Note that 1/one and $*$/many are inversed!)



- E.g. one instructor teaches many courses, but each course has at most one instructor (“one instructor, many courses”).
  - So this is “one to many” from instructor to course.
- “One to many” relationships do not need an extra table.
- “Many to one”: symmetric (e.g. “taught by”).

Common Cases (4)

“One to One” Relationships:

- Maximum cardinality 1 on both sides:

- A man can be married with at most one woman, a woman can be married with at most one man.
  - Or: “Is head of” between employee and department: Every department has exactly one head (total participation), every employee can lead at most one department (normal employees lead no department: partial participation).

Choosing Cardinalities (1)

(0,*) or (1,*)?

- All states of the real world must be stored in the database, not only normal states.
  E.g. normally an instructor teaches at least one course, but there might be a new instructor who does not yet teach a course.
- This means that “(0,*)” is often preferable to “(1,*)”.
- This especially simplifies the insertion of new entities.
  It is a good exercise to define “elementary transactions”, which transform one valid database state into another valid DB state.
- But if an entity with 0 relationships is really invalid, you must choose “(1,*)”.

Choosing Cardinalities (2)

(0,1) or (1,1)?

- This depends on the application: Can there be situations where an entity has no partner via this relationship?
- “(1,1)” simplifies the translation into the relational model.
  In contrast, the translation of “(1,*)” is more complicated than the translation of “(0,*)”.

(1,1) on both sides?

- Consider merging the two entities.

(1,1) on one side, (0,1) on the other?

- Consider using specialization (see below).

Choosing Cardinalities (3)

(0,2) or (0,*)?

- A small maximal cardinality gives additional options for the translation into the relational model.
- So it might be worth specifying it — if it is guaranteed to hold in all situations.

(0,30) or (0,*)?

- A large maximal cardinality only complicates things.
- It should only be specified as a constraint if there is some concrete reason for this specific number.