Conceptual Database Design

Jan Chomicki
University at Buffalo

University at Buffalo
Outline

1. Entity-Relationship Data Model

2. Mapping E-R schemas to relations
Proposed by Peter Chen in 1976.
Entity-Relationship (E-R) Data Model

Proposed by Peter Chen in 1976.

Features

- used for the description of the conceptual schema of the database
- not used for database implementation
- formal notation
- close to natural language
Entity-Relationship (E-R) Data Model

Proposed by Peter Chen in 1976.

**Features**

- used for the description of the **conceptual schema** of the database
- not used for database implementation
- formal notation
- close to natural language

**Can be mapped to various data models**

- relational
- object-oriented, object-relational
- XML
**Basic ER model concepts**

<table>
<thead>
<tr>
<th><strong>Schema level</strong></th>
<th><strong>Instance level</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Domain element (value)</td>
</tr>
<tr>
<td>Entity type</td>
<td>Entity</td>
</tr>
<tr>
<td>Relationship type</td>
<td>Relationship (instance)</td>
</tr>
<tr>
<td>Cardinality constraints</td>
<td>Valid relationships</td>
</tr>
<tr>
<td>Attribute</td>
<td>Attribute value</td>
</tr>
<tr>
<td>Key</td>
<td>Unique key value</td>
</tr>
</tbody>
</table>
Entities

[Definition]
Entity:
Something that exists and can be distinguished from other entities.

Examples:
A person, an account, a course.

Entity type:
A set of entities with similar properties. Entity types can overlap.

Examples:
Persons, employees, Citibank accounts, UB courses.

Entity type extension:
The set of entities of a given type in a given database instance.

Notation:

e_1, e_2, ...

"entity e is of type T":
T(e).

Jan Chomicki (University at Buffalo)
Conceptual database design
Entities

Entity

Something that exists and can be distinguished from other entities.

Examples

A person, an account, a course.
Entities

**Entity**

Something that **exists** and can be **distinguished** from other entities.

**Examples**

A person, an account, a course.

---

**Entity type**

A set of entities with similar **properties**. Entity types can overlap.

**Examples**

Persons, employees, Citibank accounts, UB courses.
Entities

**Entity**
Something that **exists** and can be **distinguished** from other entities.

**Examples**
A person, an account, a course.

**Entity type**
A set of entities with similar **properties**. Entity types can overlap.

**Examples**
Persons, employees, Citibank accounts, UB courses.

**Entity type extension**
The set of entities of a given type in a given database instance.
## Entities

### Entity

Something that **exists** and can be **distinguished** from other entities.

### Entity type

A set of entities with similar properties. Entity types can overlap.

### Entity type extension

The set of entities of a given type in a given database instance.

### Examples

A person, an account, a course.

Persons, employees, Citibank accounts, UB courses.

### Notation

- entities: $e_1, e_2, \ldots$
- “entity $e$ is of type $T$”: $T(e)$. 

Jan Chomicki (University at Buffalo)  Conceptual database design  5 / 22
Attributes

Domain
A predefined set of primitive, atomic values (entity types are not domains!).

Examples
Integers, character strings, decimals.

Attribute
A (partial) function from an entity type to a domain, representing a property of the entities of that type.

Examples
Name: Person → String
Balance: Account → Decimal

Notation
A(e): "the value of the attribute A for the entity e".

Example
Name(e₁) = 'Brown'
Attributes

**Domain**
A predefined set of primitive, atomic values (entity types are not domains!).

**Examples**
Integers, character strings, decimals.
### Attributes

<table>
<thead>
<tr>
<th>Domain</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>A predefined set of primitive, atomic values (entity types are not domains!).</td>
<td>Integers, character strings, decimals.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Examples</th>
</tr>
</thead>
</table>
| A (partial) function from an entity type to a domain, representing a property of the entities of that type. | Name: Person → String  
Balance: Account → Decimal |
### Attributes

<table>
<thead>
<tr>
<th><strong>Domain</strong></th>
<th><strong>Examples</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A predefined set of primitive, atomic values (entity types <em>are not</em> domains!).</td>
<td>Integers, character strings, decimals.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Examples</strong></th>
</tr>
</thead>
</table>
| A (partial) function from an entity type to a domain, representing a property of the entities of that type. | Name: Person → String  
Balance: Account → Decimal |

<table>
<thead>
<tr>
<th><strong>Notation</strong></th>
<th><strong>Example</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A(e): “the value of the attribute A for the entity e”.</td>
<td>Name(e₁) = ’Brown’</td>
</tr>
</tbody>
</table>
Keys

A (minimal) set of attributes that uniquely identifies every entity in an entity type.

Examples

Entity type | Key
--- | ---
Americans | SSN
ATT accounts | Phone number
NY vehicles | License plate number
US vehicles | (License plate number, State)

An entity type can have multiple keys, one key is selected as the primary key.
Keys

Key

A (minimal) set of attributes that uniquely identifies every entity in an entity type.
Keys

**Key**

A (minimal) set of attributes that uniquely identifies every entity in an entity type.

**Examples**

<table>
<thead>
<tr>
<th>Entity type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Americans</td>
<td>SSN</td>
</tr>
<tr>
<td>ATT accounts</td>
<td>Phone number</td>
</tr>
<tr>
<td>NY vehicles</td>
<td>License plate number</td>
</tr>
<tr>
<td>US vehicles</td>
<td>(License plate number, State)</td>
</tr>
</tbody>
</table>
Keys

Key

A (minimal) set of attributes that uniquely identifies every entity in an entity type.

Examples

<table>
<thead>
<tr>
<th>Entity type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Americans</td>
<td>SSN</td>
</tr>
<tr>
<td>ATT accounts</td>
<td>Phone number</td>
</tr>
<tr>
<td>NY vehicles</td>
<td>License plate number</td>
</tr>
<tr>
<td>US vehicles</td>
<td>(License plate number, State)</td>
</tr>
</tbody>
</table>

- an entity type can have **multiple** keys
- one key is selected as the **primary** key.
Relationships

A subset of the Cartesian product of some entity types $E_1, \ldots, E_k$, representing an association between the entity types. Relationship types can have attributes. Examples:

- Teaches(Employee, Class)
- Sells(Vendor, Customer, Product)
- Parent(Person, Person)

Relationship instance of arity $k$ is a $k$-tuple of entities of the appropriate types. Example:

Teaches($e_1$, $c_1$) where Employee($e_1$) and Class($c_1$) and Name($e_1$) = 'Brown'.
Relationships

Relationship type of arity $k$
A subset of the Cartesian product of some entity types $E_1, \ldots, E_k$, representing an association between the entity types. Relationship types can have attributes.

Examples
- Teaches(Employee,Class)
- Sells(Vendor,Customer,Product)
- Parent(Person,Person)
Relationships

Relationship type of arity $k$
A subset of the Cartesian product of some entity types $E_1, \ldots, E_k$, representing an association between the entity types. Relationship types can have attributes.

Examples
Teaches(Employee,Class)
Sells(Vendor, Customer, Product)
Parent(Person, Person)

Relationship instance of arity $k$
A $k$-tuple of entities of the appropriate types.

Example
Teaches($e_1, c_1$) where Employee($e_1$) and Class($c_1$) and Name($e_1$)=’Brown’.
Cardinality constraints

Binary relationship type $R(A, B)$ is:

- **1 : 1** if for every entity $e_1$ in $A$ there is at most one entity $e_2$ in $B$ such that $R(e_1, e_2)$ and *vice versa*.

- **N : 1** if for every entity $e_1$ in $A$ there is at most one entity $e_2$ in $B$ such that $R(e_1, e_2)$.

- **N : M** otherwise.
Advanced schema-level concepts

- isa relationships
- weak entity types
- complex attributes
- roles.
isa relationships

Definition
A isa B if every entity in the entity type A is also in the entity type B.

Example
Faculty isa Employee.

If A isa B, then:
- Attrs(B) ⊆ Attrs(A) (inheritance of attributes),
- Key(A) = Key(B) (inheritance of key).

Example
Rank: Faculty → {'Assistant', 'Associate', . . .}
Rank is not defined for non-faculty employees (or defined differently).
### Definition

A *isa* $B$ if every entity in the entity type $A$ is also in the entity type $B$.

### Example

Faculty *isa* Employee.
**Definition**

A isa B if every entity in the entity type A is also in the entity type B.

**Example**

Faculty isa Employee.

If A isa B, then:

- $\text{Attrs}(B) \subseteq \text{Attrs}(A)$ (inheritance of attributes),
- $\text{Key}(A) = \text{Key}(B)$ (inheritance of key).
**isa relationships**

**Definition**

A isa B if every entity in the entity type A is also in the entity type B.

**Example**

Faculty isa Employee.

If A isa B, then:

- $\text{Attrs}(B) \subseteq \text{Attrs}(A)$ (inheritance of attributes),
- $\text{Key}(A) = \text{Key}(B)$ (inheritance of key).

**Example**

Rank : Faculty $\rightarrow \{’\text{Assistant’},’\text{Associate’},\ldots\}$

Rank is not defined for non-faculty employees (or defined differently).
Weak entity types

**Definition**

A weak entity type is an entity type if:

- A does not have a key.
- The entities in A can be identified through an identifying relationship type R (A, B) with another entity type B.

The entities in A can be identified by the combination of:
- The borrowed key of B.
- Some partial key of A.

**Example**

**Entity types:** Account, Check.

**Identifying relationship type:** Issued.

**Borrowed key (of Account):** AccNo.

**Partial key (of Check):** CheckNo.
Weak entity types

Definition

A is a *weak* entity type if:

- A does not have a key.
- the entities in A can be identified through an identifying relationship type \( R(A, B) \) with another entity type \( B \).

Example

Entity types: Account, Check.
Identifying relationship type: Issued.
Borrowed key (of Account): AccNo.
Partial key (of Check): CheckNo.
Weak entity types

**Definition**

A is a *weak* entity type if:
- A does not have a key.
- the entities in A can be identified through an identifying relationship type $R(A, B)$ with another entity type $B$.

The entities in A can be identified by the combination of:
- the *borrowed* key of $B$.
- some *partial* key of $A$. 

Example

Entity types: Account, Check.
Identifying relationship type: Issued.
Borrowed key (of Account): AccNo.
Partial key (of Check): CheckNo.
Weak entity types

Definition

A is a *weak* entity type if:

- A does not have a key.
- the entities in $A$ can be identified through an identifying relationship type $R(A, B)$ with another entity type $B$.

The entities in $A$ can be identified by the combination of:

- the *borrowed* key of $B$.
- some *partial* key of $A$.

Example

Entity types: Account, Check.
Identifying relationship type: Issued.
Borrowed key (of Account): AccNo.
Partial key (of Check): CheckNo.
Complex attributes

Multivalued attributes:
- Faculty → \{ ′B.A., ′B.S., ′Ph.D., ...} 

Composite attributes:
- Address: Employee → Street × City × Zipcode

Multivalued and composite attributes can be expressed using other constructs of the E-R model.
Complex attributes

Attribute values

- sets (multivalued attributes).
- tuples (composite attributes).
Complex attributes

Attribute values
- sets (multivalued attributes).
- tuples (composite attributes).

Multivalued attribute

Degrees: Faculty → 2\{'B.A.','B.S.','...','Ph.D.','...\}
Complex attributes

Attribute values
- sets (multivalued attributes).
- tuples (composite attributes).

Multivalued attribute
Degrees: Faculty $\rightarrow$ 2{"B.A.','B.S.',...,'Ph.D.',...}

Composite attribute
Address: Employee $\rightarrow$ Street $\times$ City $\times$ Zipcode
Complex attributes

Attribute values

- sets (multivalued attributes).
- tuples (composite attributes).

Multivalued attribute

Degrees : Faculty $\rightarrow$ $2\{'B.A.', 'B.S.',..., 'Ph.D.',...\}$

Composite attribute

Address : Employee $\rightarrow$ Street $\times$ City $\times$ Zipcode

Multivalued and composite attributes can be expressed using other constructs of the E-R model.
Roles

Roles are necessary in a relationship type that relates an entity type to itself. Different occurrences of the same entity type are distinguished by different role names.

Example

In the relationship type \( \text{ParentOf}(\text{Person}, \text{Person}) \) the introduction of role names gives \( \text{ParentOf}(\text{Parent:Person}, \text{Child:Person}) \).

Jan Chomicki (University at Buffalo)
Conceptual database design
Roles are necessary in a relationship type that relates an entity type to itself. Different occurrences of the same entity type are distinguished by different role names.
Roles are necessary in a relationship type that relates an entity type to itself. Different occurrences of the same entity type are distinguished by different role names.

Example

In the relationship type \texttt{ParentOf(Person, Person)} the introduction of role names gives \texttt{ParentOf(Parent:Person, Child:Person)}
ER design

General guidelines
- Schema: stable information, instance: changing information.
- Avoid redundancy (each fact should be represented once).
- No need to store information that can be computed.
- Keys should be as small as possible.
- Introduce artificial keys only if no simple, natural keys available.

How to choose entity types
- Things that have properties of their own, or
- Things that are used in navigating through the database.
- Avoid null attribute values if possible by introducing extra entity types.
ER design

General guidelines

- schema: stable information, instance: changing information.
- avoid redundancy (each fact should be represented once).
- no need to store information that can be computed.
- keys should be as small as possible.
- introduce artificial keys only if no simple, natural keys available.
ER design

General guidelines

- schema: stable information, instance: changing information.
- avoid redundancy (each fact should be represented once).
- no need to store information that can be computed.
- keys should be as small as possible.
- introduce artificial keys only if no simple, natural keys available.

How to choose entity types

- things that have properties of their own, or
- things that are used in navigating through the database.
- avoid null attribute values if possible by introducing extra entity types.
isa relationship design

Generalization (bottom-up)
- generalize a number of different entity types (with the same key) to a single type.
- factor out common attributes.

Example:
- Student isa Person
- Teacher isa Person
- Name: Person → String

Specialization (top-down)
- specialize an entity type to one or more specific types.
- add attributes in more specific entity types.

Example:
- Salary: Teacher → Decimal
isa relationship design

Generalization (bottom-up)

- generalize a number of different entity types (with the same key) to a single type.
- factor out common attributes.

Example

Student isa Person
Teacher isa Person
Name : Person → String
isa relationship design

Generalization (bottom-up)
- generalize a number of different entity types (with the same key) to a single type.
- factor out common attributes.

Example
Student isa Person
Teacher isa Person
Name : Person → String

Specialization (top-down)
- specialize an entity type to one or more specific types.
- add attributes in more specific entity types.

Example
Salary : Teacher → Decimal
Mapping E-R schemas to relations

Assumption
No complex attributes.

Multiple stages
1. creating relation schemas from entity types.
2. creating relation schemas from relationship types.
3. identifying keys.
4. identifying foreign keys.
5. schema optimization.
Mapping E-R schemas to relations

Assumption

No complex attributes.
Mapping E-R schemas to relations

**Assumption**
No complex attributes.

**Multiple stages**
1. creating relation schemas from **entity types**.
2. creating relation schemas from **relationship types**.
3. identifying **keys**.
4. identifying **foreign keys**.
5. schema **optimization**.
Mapping entity types to relations

<table>
<thead>
<tr>
<th>Entity type</th>
<th>Relation schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$ such that $E_1$ isa $E_2$</td>
<td>$\text{Key}(E_2)$ \cup (\text{Attrs}(E_1) - \text{Attrs}(E_2))</td>
</tr>
<tr>
<td>$E_1$ is a weak entity type identified by $R(E_1, E_2)$</td>
<td>$\text{Key}(E_2)$ \cup (\text{Attrs}(E_1) - \text{Attrs}(E_2))</td>
</tr>
<tr>
<td>$E_1$ is none of the above</td>
<td>$\text{Attrs}(E_1)$</td>
</tr>
</tbody>
</table>
Mapping relationship types to relations

<table>
<thead>
<tr>
<th>Relationship type</th>
<th>Relation schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R(E_1, \ldots, E_n)$</td>
<td>$\text{Key}(E_1) \cup \cdots \cup \text{Key}(E_n)$ $\cup \text{Attrs}(R)$</td>
</tr>
</tbody>
</table>
## Mapping relationship types to relations

<table>
<thead>
<tr>
<th>Relationship type</th>
<th>Relation schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R(E_1, \ldots, E_n)$</td>
<td>$\text{Key}(E_1) \cup \cdots \cup \text{Key}(E_n)$ $\cup \text{Attrs}(R)$</td>
</tr>
</tbody>
</table>

No relations are created from *isa* or identifying relationships.
Mapping relationship types to relations

<table>
<thead>
<tr>
<th>Relationship type</th>
<th>Relation schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R(E_1, \ldots, E_n)$</td>
<td>$\text{Key}(E_1) \cup \cdots \cup \text{Key}(E_n)$ $\cup \text{Attrs}(R)$</td>
</tr>
</tbody>
</table>

No relations are created from \textit{isa} or identifying relationships.

Different occurrences of the same attribute name should be named differently.
Identifying keys

Relation schema $W$ is the result of mapping an entity type $E_1$ or a relationship type $R(E_1, E_2)$. 
Identifying keys

Relation schema $W$ is the result of mapping an entity type $E_1$ or a relationship type $R(E_1, E_2)$.

<table>
<thead>
<tr>
<th>Source of $W$</th>
<th>Key of $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity type $E_1$</td>
<td>$Key(E_1)$</td>
</tr>
<tr>
<td>Weak entity type $E_1$</td>
<td>Union of borrowed and partial keys of $E_1$</td>
</tr>
<tr>
<td>$R(E_1, E_2)$ is 1 : 1</td>
<td>$Key(E_1)$ or $Key(E_2)$</td>
</tr>
<tr>
<td>$R(E_1, E_2)$ is $N : 1$</td>
<td>$Key(E_1)$</td>
</tr>
<tr>
<td>$R(E_1, E_2)$ is $N : M$</td>
<td>$Key(E_1) \cup Key(E_2)$</td>
</tr>
</tbody>
</table>
Identifying keys

Relation schema $W$ is the result of mapping an entity type $E_1$ or a relationship type $R(E_1, E_2)$.

<table>
<thead>
<tr>
<th>Source of $W$</th>
<th>Key of $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity type $E_1$</td>
<td>$\text{Key}(E_1)$</td>
</tr>
<tr>
<td>Weak entity type $E_1$</td>
<td>Union of borrowed and partial keys of $E_1$</td>
</tr>
<tr>
<td>$R(E_1, E_2)$ is 1:1</td>
<td>$\text{Key}(E_1)$ or $\text{Key}(E_2)$</td>
</tr>
<tr>
<td>$R(E_1, E_2)$ is N:1</td>
<td>$\text{Key}(E_1)$</td>
</tr>
<tr>
<td>$R(E_1, E_2)$ is N:M</td>
<td>$\text{Key}(E_1) \cup \text{Key}(E_2)$</td>
</tr>
</tbody>
</table>

These rules can be generalized to arbitrary relationship types $R(E_1, \ldots, E_n)$. 
Identifying foreign keys

Relation schema $W$ is the result of mapping an entity type $E_1$ or a relationship type $R(E_1, E_2)$. 
Identifying foreign keys

Relation schema $W$ is the result of mapping an entity type $E_1$ or a relationship type $R(E_1, E_2)$.

<table>
<thead>
<tr>
<th>Source of $W$</th>
<th>Foreign keys of $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity type $E_1$</td>
<td>No foreign keys</td>
</tr>
<tr>
<td>Weak entity type $E_1$</td>
<td>Borrowed key of $E_1$</td>
</tr>
<tr>
<td>Entity type $E_1$</td>
<td>$\text{Key}(E_1)$</td>
</tr>
<tr>
<td>such that $E_1$ isa $E_2$</td>
<td></td>
</tr>
<tr>
<td>$R(E_1, E_2)$</td>
<td>$\text{Key}(E_1)$, $\text{Key}(E_2)$</td>
</tr>
</tbody>
</table>
Schema optimization

Combine relation schemas with *identical* keys coming from *the same* entity type.
Schema optimization

Combine relation schemas with *identical* keys coming from *the same* entity type.

Student(SName,Address) can be combined with Advising(SName,Faculty) to yield Student(SName,Address,Faculty).
Schema optimization

Combine relation schemas with *identical* keys coming from *the same* entity type.

**Student(SName,Address)** can be combined with **Advising(SName,Faculty)** to yield **Student(SName,Address,Faculty)**.

**Different keys**

**Student(SName,Address)** *should not* be combined with **Grades(SName,Course,Grade)**.
Schema optimization

Combine relation schemas with identical keys coming from the same entity type.

Student(SName, Address) can be combined with Advising(SName, Faculty) to yield Student(SName, Address, Faculty).

Different keys

Student(SName, Address) should not be combined with Grades(SName, Course, Grade).

Different entity types

Student(SName, Address) should not be combined with Graduate(SName).