Entity-Relationship Data Model

Proposed by P. Chen in 1976.

Used for the description of the conceptual schema of the database.

*Formal* notation but close to natural language.

Can be *mapped* to various data models:

- relational
- object-oriented
- XML
- ...

Textbook, chapters 6 and 7
### Basic ER model concepts

<table>
<thead>
<tr>
<th>Instance level</th>
<th>Schema level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>Entity type</td>
</tr>
<tr>
<td>Relationship (instance)</td>
<td>Relationship type</td>
</tr>
<tr>
<td>Attribute value</td>
<td>Cardinality constraints</td>
</tr>
<tr>
<td>Key value</td>
<td>Attribute</td>
</tr>
<tr>
<td>Key</td>
<td>Key</td>
</tr>
</tbody>
</table>
**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.
Examples: persons, employees, Citibank accounts, courses.
Entity types can overlap.

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

*Notation:*

entities: \( e_1, e_2, \ldots \)

“entity \( e \) is of type \( T \)” : \( T(e) \).
**Attributes**

**Domain:** a predefined set of primitive, atomic values.
Examples: integers, character strings, decimals.
Entity types *are not* domains.

**Attribute:** a (partial) function from an entity type to a domain.
Attributes represent *properties* of entities.
Examples:

\[
\begin{align*}
\text{Name} & : \text{Person} \rightarrow \text{String} \\
\text{Balance} & : \text{Account} \rightarrow \text{Decimal}
\end{align*}
\]

Notation:

\[A(e): \text{“the value of the attribute } A \text{ for the entity } e \text{”}\].
Example:

\[\text{Name}(e_1) = \text{‘Brown’}\]
Keys

**Key:** a (minimal) set of attributes that uniquely identifies every entity in an entity type.

*This is a schema-level notion.*

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>

There may be more than one key for an entity type. One is picked as the *primary* key.
Relationships

**Relationship type of arity** $k$: a subset of the Cartesian product of some entity types $E_1, \ldots, E_k$.

Examples:
- Teaches(Employee, Class)
- Supplies(Supplier, Customer, Product)
- Parent(Person, Person)

Relationship types represent associations between entity types. They can have attributes.

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

Examples:
- $\text{Teaches}(e_1, c_1)$ where $\text{Employee}(e_1)$ and $\text{Class}(c_1)$ and $\text{Name}(e_1) = 'Brown'$. 
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.
Diagrams I

- **Entity type**
- **Relationship type**
- **Attribute**

Key attributes are underlined.
Diagrams II

1:1

N:1

N:M
Advanced schema-level concepts

• **isa** relationships.
• weak entity types.
• complex attributes
• roles.
isa relationships

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

Example: Faculty isa Employee.

This is a schema-level notion.

If A isa B, then:

\[ \text{Attributes}(B) \subseteq \text{Attributes}(A) \] (inheritance of attributes),

\[ \text{Key}(A) = \text{Key}(B) \] (inheritance of key).

Example:

\[ \text{Rank} : \text{Faculty} \rightarrow \{ 'Assistant', 'Associate', \ldots \} \]

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

A is a *weak* entity type if:

- it 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: Employee, Dependent.

Identifying relationship type: $\text{DepOf}(\text{Dependent}, \text{Employee})$.

Borrowed key (of Employee): Name.

Partial key (of Dependent): FirstName.
Diagrams III

isa relationship

Identifying relationship

Weak entity type
Complex attributes

Attribute values can be:

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

Examples:

Multivalued attribute:

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

Composite attribute:

\[
\text{Address} : \text{Employee} \rightarrow \text{Street} \times \text{City} \times \text{Zipcode}
\]

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

- Multivalued attribute
- Composite attribute
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

\[
\text{ParentOf(} \text{Person, Person}\text{)}
\]

the introduction of role names gives

\[
\text{ParentOf(} \text{Parent : Person, Child : Person}\text{)}
\]
**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 more specific types.
- add attributes in more specific entity types.

**Example:**

Salary : Teacher → Decimal
Mapping ER diagrams 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>$Key(E_2)$</td>
</tr>
<tr>
<td></td>
<td>$\cup(Attrs(E_1) - Attrs(E_2))$</td>
</tr>
<tr>
<td>$E_1$ is a weak entity type identified by $R(E_1, E_2)$</td>
<td>$Key(E_2)$</td>
</tr>
<tr>
<td></td>
<td>$\cup(Attrs(E_1) - Attrs(E_2))$</td>
</tr>
<tr>
<td>$E_1$ is none of the above</td>
<td>$Attrs(E_1)$</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>$Key(E_1) \cup \cdots Key(E_n)$ $\cup Attrs(R)$</td>
</tr>
</tbody>
</table>

No relations are created from 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 $R(E_1, E_2)$.

<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>

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 $R(E_1, E_2)$.

<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 $E_1$ such that $E_1\text{ isa } E_2$</td>
<td>$Key(E_1)$</td>
</tr>
<tr>
<td>$R(E_1, E_2)$</td>
<td>$Key(E_1)$, $Key(E_2)$</td>
</tr>
</tbody>
</table>
Schema optimization

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

Example.

**COURSE**

| CNAME | CNUMBER |

and

**MEETS_IN**

| CNUMBER | BUILDING | RNUMBER |

can be combined to yield:

**COURSE**

| CNAME | CNUMBER | BUILDING | RNUMBER |
Designing relational databases

Method 1:
1. design an E-R schema.
2. map it to a relational schema.

Method 2:
1. design an E-R schema.
2. map it to a relational schema.
3. modify the relational schema.

Method 3:
1. start with a single relational schema containing all attributes.
2. decompose it.

The schema resulting from Method 1 is guaranteed to be “good”, while those resulting from Methods 2 and 3 have to be analyzed for “goodness”.
“Good” and “bad” database schemas

“Bad” schema:

- **Repetition** of information. Leads to redundancy and update anomalies.
- **Inability to represent** information. Leads to anomalies in insertion and deletion.

“Good” schema:

- relation schemas in **normal form** (anomaly-free): 3NF, BCNF.
- decompositions have the lossless join property and preserve dependencies.
Functional dependencies (FDs)

Relation schema $R(A_1, \ldots, A_n)$.

Sets of attributes: $X, Y, Z, \ldots \subseteq \{A_1, \ldots, A_n\}$.

**Functional dependency**: a pair $X \rightarrow Y$.

Notation:

- $A_1 \cdots A_n$ instead of $\{A_1, \ldots, A_n\}$.
- $XY$ instead of $X \cup Y$. 
Theory of functional dependencies

**FD satisfaction:** $r$ satisfies $X \rightarrow Y$ if for all tuples $t_1, t_2 \in r$:

if $t_1[X] = t_2[X]$, then also $t_1[Y] = t_2[Y]$.

**Entailment:** A set of FDs $F$ entails $X \rightarrow Y$, if every relation that satisfies all the dependencies in $F$, also satisfies $X \rightarrow Y$.

Notation: $F \models X \rightarrow Y$ ($F$ entails $X \rightarrow Y$).

**Closure of a dependency set** $F$: the set of dependencies entailed by $F$.

Notation: $F^+ = \{ X \rightarrow Y : F \models X \rightarrow Y \}$.
Keys

Given $R(A_1, \ldots, A_n)$ and a set of dependencies $F$ over $R$.

$X \subseteq \{A_1, \ldots, A_n\}$ is a key of $R$ if:

1. the dependency $X \rightarrow A_1 \cdots A_n$ is in $F^+$.
2. for all proper subsets $Y$ of $X$, the dependency $Y \rightarrow A_1 \cdots A_n$ is not in $F^+$.

Related notions:

- primary key: one designated key.
- candidate key: one of the keys.
- superkey: superset of a key.
Inference of functional dependencies

**Problem:** how to tell whether \( X \rightarrow Y \in F^+ \).

Notation: \( U \) - the set of all the attributes of \( R \).

**Inference rules (Armstrong axioms):**

1. *reflexivity*: if \( Y \subseteq X \subseteq U \), then infer \( X \rightarrow Y \) (*trivial* dependency).
2. *augmentation*: if \( Z \subseteq U \), then from \( X \rightarrow Y \) infer \( XZ \rightarrow YZ \).
3. *transitivity*: from \( X \rightarrow Y \) and \( Y \rightarrow Z \), infer \( X \rightarrow Z \).
Properties of axioms

Armstrong axioms are:

- **sound**: if $X \rightarrow Y$ is derived from $F$, then $X \rightarrow Y \in F^+$.
- **complete**: if $X \rightarrow Y \in F^+$, then $X \rightarrow Y$ is derived from $F$.

Additional (implied) inference rules:

4. **union**: from $X \rightarrow Y$ and $X \rightarrow Z$, infer $X \rightarrow YZ$.

5. **decomposition**: if $Z \subseteq Y$, then from $X \rightarrow Y$ infer $X \rightarrow Z$. 
Boyce-Codd Normal Form (BCNF)

Notation:
- $R$ is a relation schema.
- $F$ is the set of FDs associated with $R$.
- $F^+$ is the dependency closure of $F$.
- $A$ is an attribute of $R$.

$R$ is in BCNF if for every functional dependency $X \rightarrow A \in F^+$:
- $A \in X$ (trivial FD), or
- $X$ contains a key of $R$.

Each instance of a relation schema which is in BCNF does not contain a redundancy (that can be detected using FDs alone).
Third Normal Form (3NF)

$R$ is in 3NF if for every functional dependency $X \rightarrow A \in F^+$:

- $A \in X$ (trivial FD), or
- $X$ contains a key of $R$, or
- $A$ is part of some key of $R$.

If $R$ is in BCNF, it is also in 3NF.

There are relations that are in 3NF but not in BCNF:

<table>
<thead>
<tr>
<th>CSZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITY</td>
</tr>
<tr>
<td>STREET</td>
</tr>
<tr>
<td>ZIP</td>
</tr>
</tbody>
</table>

with dependencies:

- CITY STREET $\rightarrow$ ZIP
- ZIP $\rightarrow$ CITY
Redundancies in 3NF

A relation schema in 3NF may still have an instance with redundancies.

Example.

**CSZ**

<table>
<thead>
<tr>
<th>CITY</th>
<th>STREET</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>First</td>
<td>77077</td>
</tr>
<tr>
<td>SF</td>
<td>Third</td>
<td>77077</td>
</tr>
</tbody>
</table>

The dependency ZIP $\rightarrow$ CITY violates BCNF and identifies a redundancy.
Decompositions

**Decomposition:** replacement of a relation schema $R$ by two relation schema $R_1$ and $R_2$ such that:

- both $R_1$ and $R_2$ are subsets of $R$,
- $R_1 \cup R_2 = R$.

**Lossless decomposition:** $(R_1, R_2)$ is a lossless decomposition of $R$ with respect to a set of FDs $F$ if for every instance $r$ of $R$ that satisfies $F$:

$$\pi_{R_1}(r) \Join \pi_{R_2}(r) = r.$$  

A simple criterion for checking whether a decomposition $(R_1, R_2)$ is lossless:

- $F \models R_1 \cap R_2 \rightarrow R_1$, or
- $F \models R_1 \cap R_2 \rightarrow R_2$.

A sequence of decompositions of $R$ into $R_1$ and $R_2$, $R_1$ into $R'_1$ and $R''_1$ etc. may be viewed as a decomposition of $R$ into more than two relation schemas.
Dependency preservation

Dependencies associated with relation schema $R_1$ and $R_2$ in a decomposition $(R_1, R_2)$:

$$F_{R_1} = \{ X \rightarrow Y | X \rightarrow Y \in F^+ \land XY \subseteq R_1 \}$$

$$F_{R_2} = \{ X \rightarrow Y | X \rightarrow Y \in F^+ \land XY \subseteq R_2 \}.$$ 

$(R_1, R_2)$ preserves a dependency $f$ iff $f \in (F_{R_1} \cup F_{R_2})^+$. 
Decomposition into BCNF

Notation:
- $F$ - set of FDs associated with $R$.

ALGORITHM:
For some nontrivial nonkey dependency $X \rightarrow A$ in $F^+$:

1. create a relation with the schema $XA$.
2. remove $A$ from $R$.

If the resulting schemas are not in BCNF, decompose them further.

This algorithm produces a lossless decomposition into BCNF which does not have to preserve dependencies.
Decomposition (synthesis) into 3NF

Minimal cover $F'$ for $F$:

- set of FDs equivalent to $F$ ($F^+ = (F')^+$),
- all FDs in $F'$ are of the form $X \rightarrow A$,
- further simplification by removing dependencies or attributes from dependencies in $F'$ yields a set of FDs inequivalent to $F$.

ALGORITHM:

Create $F'$.

Create a relation $XA$ for every dependency $X \rightarrow A \in F'$.

Create a relation $X$ for some key $X$ of $R$.

Remove redundancies.

This algorithm produces a lossless decomposition into 3NF which preserves dependencies.
Multivalued dependencies (MVDs)

Relation schema $R(A_1, \ldots, A_n)$.

Sets of attributes: $X, Y, Z, \ldots \subseteq \{A_1, \ldots, A_n\}$.

**Multivalued dependency**: a pair $X \rightarrow Y$.

**MVD satisfaction**: $r$ satisfies $X \rightarrow Y$ if for all tuples $t_1, t_2 \in r$:

if $t_1[X] = t_2[X]$, then there is a tuple $t_3 \in r$ such that $t_3[XY] = t_1[XY]$ and $t_3[Z] = t_2[Z]$,

where $Z = \{A_1, \ldots, A_n\} - XY$.

**Entailment**: defined in the same way as for FDs.
Fourth Normal Form (4NF)

$F$ is the set of FDs and MVDs associated with a relation schema $R = \{A_1, \ldots, A_n\}$.

$R$ is in 4NF if for every multivalued dependency $X \rightarrow Y$ entailed by $F$:

- $Y \subseteq X$ or $XY = \{A_1, \ldots, A_n\}$ (trivial MVD), or
- $X$ contains a key of $R$. 