Theoretical Basis of the Relational Model

Relational model, keys, integrity constraints, and transformation from ER

Michael Emmerich
JYU Faculty of Information Technology, Finland

January 27, 2026

Relational Model Basics

Basic Structure of the Relational Model

- In practice, **relations** are often compared to **tables** (e.g., in spreadsheets).
- The relational model is a **logical-level** data model: it describes *what* is stored and the allowed structure.
- We use a relation name and a set of **attributes** (columns); the table contains **tuples** (rows).

Example: a Relation as a Table (SUPPLIER)

- In a relational database all information of the logical database design is organized as a set of tables (relations).
- Here is an example table:

SupplierID	Name	City	FoundedYear
31	Worchware	Kaeli	2003
42	Fycbow	Shimo–carnia	1993
33	Myjice	Yarghe–pfuhar	2008
54	Bivoke	Lucan	1986
25	Vakoj	Idgipurn	2000

Relations can often be *thought of* as spreadsheet tables (but the relational model has strict rules and additional elements such as keys and schemata).

Core Definitions (high level)

- A **relation** R can be seen as a pair $\langle H, r \rangle$:
  - **Header** H: a set of attribute names (the "column names").
  - **Body** r: a set of tuples (the "rows").
- An **attribute** is an attribute name together with a domain of values.
- A **tuple** is an ordered collection of *attribute–value* pairs, one value per attribute in H.
- A **relational database** is a set of relations plus integrity constraints.

Auxiliary Terms

- **Relation schema** (or structure): the relation name plus its header.
- **Relation body**: the set of tuples in the relation.
- **Cardinality of a relation**: number of tuples (rows).
- **Cardinality of an attribute**: number of distinct values appearing in a column.

Worked Example: DELIVERY Relation

- Schema:

$$\texttt{DELIVERY}(\texttt{supplier\_no}, \texttt{part\_no}, \texttt{project\_no}, \texttt{quantity})$$

- Header is the attribute set $\{\texttt{supplier\_no}, \texttt{part\_no}, \texttt{project\_no}, \texttt{quantity}\}$.
- If the table has 5 rows, then the **cardinality of the relation** is 5.
- Example tuple (omitting attribute names):

$$(1,\ 2,\ \texttt{p123},\ 17)$$

- If the values in `quantity` are $\{17, 23, 9, 4, 12\}$, then the **attribute cardinality** of `quantity` is 5.

Special Value NULL vs. Quantity 0 in DELIVERY

- In a relation, an attribute value can be **known** (e.g., 17, 0) or **unknown / not provided** (NULL).
- NULL **in** quantity means: the quantity is *not defined / missing data*.
  - Example: the shipment record exists, but the delivered amount was not recorded.
  - NULL is *not* a number, and it is *not* equal to 0.
- **0 in** quantity means: the quantity is *exactly zero*.
  - Example: **"Loppu"** / *zero products available* is a precise statement: quantity $= 0$.
- Contrast with two tuples:

$$(1,\ 2,\ \texttt{p123},\ \texttt{NULL}) \quad \text{vs.} \quad (1,\ 2,\ \texttt{p123},\ 0)$$

- Consequence: treating NULL as 0 changes meaning and can corrupt results (e.g., sums, averages, and comparisons).

Revision (Quick Check)

Mark each statement as True/False.

1. The relation name is `DELIVERY`.
2. `supplier_no` and `part_no` are names of the relation's attributes.
3. The relation header is `supplier_no`, `part_no`, `project_no`, `quantity`.
4. The degree of the relation is five.
5. The cardinality of the relation is five.
6. In the relation, the cardinality of the attribute `part_no` is 4.
7. No null values appear in the relation.

Core Properties of Relations

According to the relational model, a relation should satisfy:

1. Every tuple has the same number of values (one per attribute; no "missing columns").
2. The order of tuples does **not** matter.
3. Tuples are **distinct** (no two identical rows).
4. The order of attributes in the header does **not** matter (names connect columns to meaning).
5. Attribute names should make the meaning clear (good naming matters).

Example of a Bad "Relation" (why rules matter)

- A "Top 5 suppliers" table can break relational rules if:
  1. some rows have missing values (null where not intended),
  2. row position is used as a rank (order would then matter),
  3. identical rows exist (duplicates),
  4. two columns share the same name (ambiguous),
  5. a column name is unclear (meaning not defined).
- Fixes: add an explicit `rank` attribute, enforce keys, rename columns, and define domains.

Keys and Integrity Constraints

Integrity Constraints

- Integrity constraints describe **allowed states** of a database.
- Key ideas introduced here:
  - **Primary keys** (uniquely identify tuples)
  - **Foreign keys** (references between relations)
  - Additional constraints (e.g., range constraints, business rules)

Primary Key and Candidate Key

- Every tuple should be **identifiable** (distinguishable) from all others.
- A **candidate key** is an attribute set that:
  - is **unique** (no two tuples share the same key value),
  - is **minimal** (removing any attribute destroys uniqueness).
- A **primary key** (PK) is one chosen candidate key (often underlined in schemas).
- Not always possible to choose a "perfect" key: stability and practicality matter.

Example: Candidate Keys in STUDENT

Consider a student registry relation:

$$\texttt{STUDENT}(\underline{\texttt{std\_no}},\ \texttt{ssn},\ \texttt{first\_name},\ \texttt{last\_name},\ \texttt{postal\_code},\ \texttt{city})$$

- Candidate keys in the domain:
  - $\{\texttt{std\_no}\}$ (university internal student number)
  - $\{\texttt{ssn}\}$ (social security number) *if stable and unique in your context*
- Not candidate keys (examples):
  - $\{\texttt{std\_no}, \texttt{ssn}\}$ is not minimal.
  - $\{\texttt{first\_name}, \texttt{last\_name}, \texttt{postal\_code}\}$ is not guaranteed unique.

Choosing Good Primary Keys (rules of thumb)

- Prefer keys that are:
  - stable (do not change),
  - easy to manage and index,
  - minimal (few attributes),
  - meaningful *enough* or safely artificial. *always defined*: should never be NULL
- Often, a synthetic identifier is used (e.g., sequential id, UUID, NanoID, Snowflake ID).

Foreign Key (FK)

- A **foreign key** (FK) in relation R is an attribute set whose values must come from a key of some relation S.
- Purpose: represent references and enforce **referential integrity**.
- Notation example:

$$R(\ldots, a, \ldots), \quad S(\underline{a}, \ldots), \quad R.a \to S.a$$

- A FK does *not* need to be unique; many tuples may reference the same target tuple.

Example Schema with PKs and FKs

- Primary keys are underlined, foreign keys indicated by arrows.
- Notes:
  - Attributes of a PK may also act as FKs.
  - FKs may be null (depending on constraints), unlike PKs.
  - If the referenced tuple is deleted/updated, the DBMS must handle the impact (restrict, set null, cascade).

Example (illustrative style):

SUPPLIER(supplier_no, name, city)
PART(part_no, name)
PROJECT(project_no, name, budget)
DELIVERY(supplier_no,part_no,project_no, quantity)

DELIVERY.supplier_no → SUPPLIER.supplier_no
DELIVERY.part_no → PART.part_no
DELIVERY.project_no → PROJECT.project_no

Revision + Other Integrity Constraints

Quick check (True/False):

1. A relation has one or more primary keys.
2. A relation can have more than one candidate key.
3. A tuple consists of attribute names and values.
4. A relation's foreign key refers to an attribute set (1..n attributes) of some other relation.

Examples of other constraints:

- Salary must not exceed supervisor's salary.
- Only one person who is also employed by the department can be a department manager.
- A phone number must start with +358.
- An attribute value cannot be null (NOT NULL).

Allowed Data Types and Normalization (preview)

- Attribute values are typically atomic (numbers, strings, dates, ...).
- If a relation violates modeling requirements, it may be **unnormalized**.
- It is often possible to fix unnormalized relations by **normalizing** them (later topic).
- Different data models and DBMSs support different data types (covered in the SQL part).

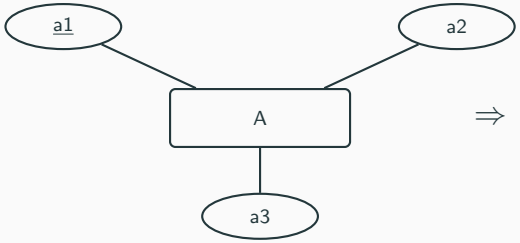
Transformation: From ER to Relations

Transformation (Conceptual → Relational)

- A conceptual model (e.g., ER diagram) captures domain concepts and relationships.
- The relational model has only **relations**, **primary keys**, **foreign keys**, and constraints.
- There is no direct 1:1 correspondence between ER constructs and relations.
- **Transformation** is the rule-based conversion of an ER diagram into a relational schema.
- Rules here are adapted from standard database textbooks (e.g., Elmasri & Navathe).

Rule: Strong Entity Set

Create one relation for each strong entity set.
Include all ordinary (simple) attributes.
The primary key is formed from the key attribute(s).



$\Rightarrow$ $A(\underline{a1}, a2, a3)$

Rules: Relationships with different Cardinality Constraints

- Relationship (with cardinality constraints). How many times min..max can/must an entity participate in the relationship set?
- Use **min** (0 or 1) and **max** (1 or N) near each side.

STUDENT(StudID, ...),COURSE(CourseID, ...), EnrolledIn(StudID, CourseID),
FK: EnrolledIn.StudId → STUDENT.StudId, EnrolledIn.CourseId → STUDENT.CourseId

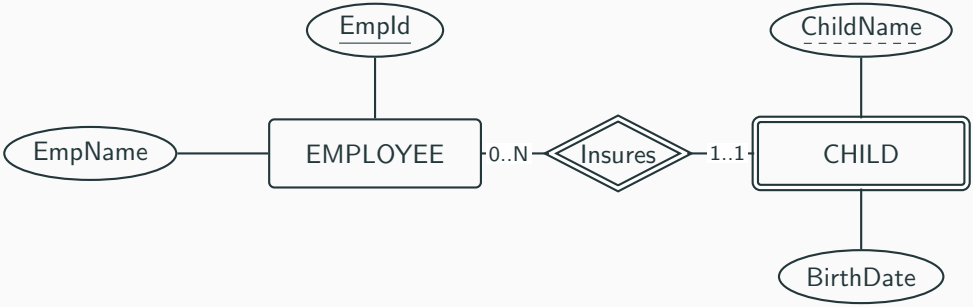
PERSON(PersonId, ...), CAR(CarID, PersonId, ...), FK: CAR.PersonId → PERSON.PersonId

Notably only two relations are used.

PLAYER(PlayerId,...), TEAM(TeamId,...), playsIn(PlayerId,..., TeamId),
FK: PlaysIn.PlayerId → PLAYER.PlayerId

Rule: Weak Entity Set

- Recap: A **weak entity** cannot be uniquely identified by its own attributes alone.
- It is identified through an **identifying relationship** with a strong entity.
- Rule of thumb: max cardinality of weak entity in identifying relationship is often 1.



Relational schema

`EMPLOYEE(EmpId, EmpName)`

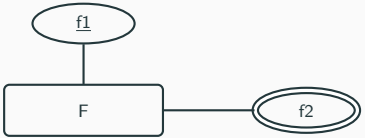
`CHILD(EmpId, ChildName, BirthDate)`

`CHILD.EmpId` $\rightarrow$ `EMPLOYEE.EmpId`

Rule: Transforming Attributes

- **Rule (Derived attributes)**: discard derived attributes (compute when needed).
- **Rule (Composite attributes)**: replace a composite attribute by its components; discard the composite node.
- **Rule (Multivalued attributes)**: create a separate "attribute relation" containing:
  - the key of the owner entity set, and
  - the multivalued attribute

  and make the owner key a foreign key.



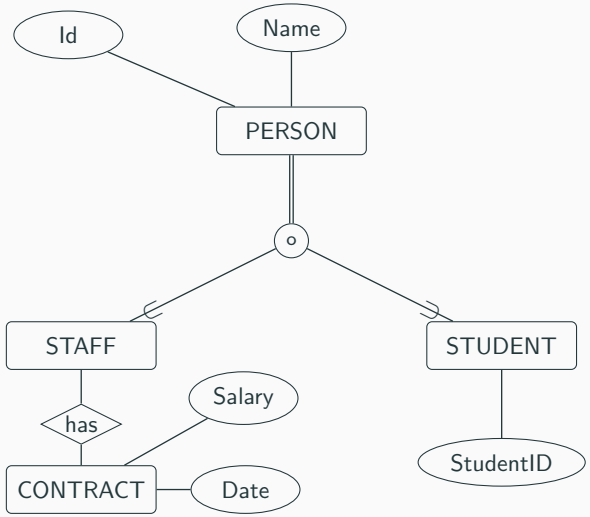
$$\Rightarrow \begin{array}{l} F(\underline{f1}) \\ F2(\underline{f1}, f2) \\ F2.f1 \rightarrow F.f1 \end{array}$$

Note on Cardinalities (in these rules)

- In the transformation rules below, cardinalities refer to **maximum** cardinalities.
- A 0:1 relationship means max participation of an entity in a relationship is 1 (e.g., "a person can be in at most one active marriage").
- A 1:1 relationship means exact cardinality is 1 (e.g., "a car has exactly one registration").
- In the relational model, **minimum** cardinality is generally not expressible in the standard ER model; additional integrity constraints are needed (we can encode them later, e.g., in SQL).

Recap: Abstraction Structures (EER)

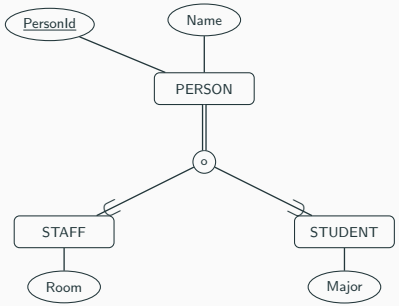
- Use **E**xtended ER (EER) to model inheritance-like structures.
- A **super-entity** has shared attributes.
- **sub-entities** specialize it (e.g., STUDENT, STAFF).
- Subtypes can have specific relationships (e.g. STAFF has CONTRACT)
- Disjoint vs overlapping; total vs partial.
  - Supertype: **PERSON**; subtypes: **STUDENT**, **STAFF** (U-marks = children; 1/2 lines = partial/total coverage).
  - Circle label: **d** = disjoint, **o** = overlapping.



Transforming Abstraction Structures (EER)

There are (at least) four common approaches:

1. **Way 1**: one relation for each super- and sub-entity set; sub-relation key includes super key.
2. **Way 2**: one relation for each sub-entity set; include both sub and super attributes.
3. **Way 3**: one relation for the whole hierarchy; add one `type` attribute (may lead to many NULLs).
4. **Way 4**: like Way 3, but use boolean "flags" (e.g., `isStaff`, `isStudent`).



Way 1:
PERSON(PersonId,Name)
STAFF(PersonId, room)
STUDENT(PersonId,Major)
FK: STAFF.PersonId→PERSON.PersonId,
STUDENT.PersonId→PERSON.PersonId
Way 3:
PERSON(PersonId,Name, Room, Major,
IsStaff:bool, IsStudent:bool)

Heuristics for Abstraction Structures (translated labels)

- **Disjoint + Total**: often choose Way 2 (or Way 3 if acceptable NULLs).
- **Disjoint + Partial**: often choose Way 1 (avoid repetition).
- **Overlapping + Total**: often choose Way 4 (flags), otherwise Way 1.
- **Overlapping + Partial**: often choose Way 4; otherwise, Way 1 (be careful with repetition).

Key Decision Principle: Use keys to enforce constraints and avoid redundancy.

Applying Rules (practical advice)

- Real ER diagrams may trigger multiple rules; there are exceptions.
- A general approach:
  1. transform entity sets,
  2. transform attributes,
  3. transform relationship sets,
  4. transform abstraction structures.
- If several rules could apply:
  - try suitable rules sequentially,
  - compare results from the database viewpoint (fewer relations, clearer structure, fewer NULLs).
- All relationships (except identifying ones) can always be transformed via Rule 8 by creating a relationship relation (but you may need to normalize later).
- Way 1 for abstraction structures always works, but can cause repetition.

Summary

- The relational model represents data as relations (tables) with strict structural properties.
- Keys: candidate keys and primary keys ensure identifiability; foreign keys represent references and enforce integrity.
- Transformation rules provide a systematic mapping from ER/EER designs to relational schemas.
- When multiple transformations are possible, choose the one that fits the domain and leads to a clean schema.