Conceptual design

ER Models: Entities

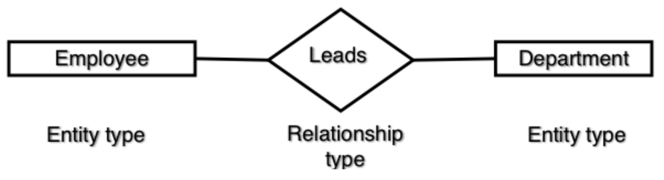
- **Entity:** A *thing* in the real world with an independent physical or conceptual existence that can be characterised through a set of attributes.
- Has an independent physical or conceptual existence:
  - `meerkat, Martin, Yarra, love, IBM, GEOM90018, river.`

ER Models: Entity types

- **Entity type: defines** a set of entities with the **same attributes** (as a template), but different attribute values (e.g.: STUDENT)
- **Entity set:** a **set of entities** of the **same entity type**.
  - Martin vs. Person
  - University vs. UoM

ER Models: Relationship Types

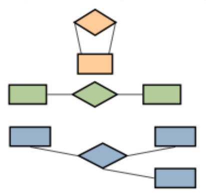
- **A Relationship Type *R*** between **entity types** E_1, E_2, …, E_n ***defines*** a **set of associations** amongst **entities** from these **entity types (relationship set)**
- Individual entities are said to **participate** in a relationship.
- A relationship type links one or more entity types together
- **In ER-Diagram relationsip types are represented as diamonds**



19

ER Models: Relationship Degree

- **Degree of relationship:** number of participating **entity types**
  - **Recursive: Degree 1**
  - **Binary: 2 entity types**
  - **Ternary (and above): 3 entity types**

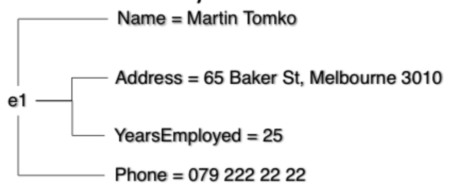


Relationship attributes may be:

- 1:1 relationships – moved to any of the participating entities
- 1:*n* relationships – moved to the entity on the *n* side
- m:n relationships – must be modeled and implemented as relationship attrtibutes

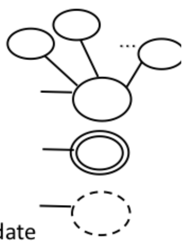
ER Models: Attributes

- **Attribute:** A particular property that describes an entity. Attributes consists of a name and data type.
- Each entity **is described** through the values of its attributes
- Each attribute of an entity has a **value**.



Entity of an employee

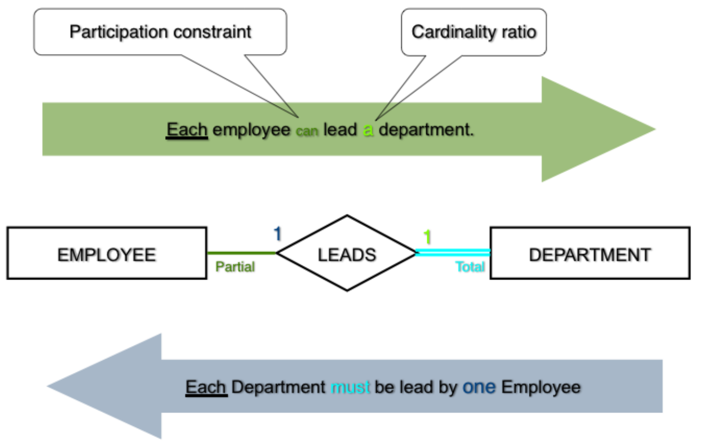
ER Models: Attribute types



- **Simple vs. Composite**
  - E.g.: Address (Street, City, State, PLZ)
- **Single-valued vs. multi-valued**
  - E.g.: Colours of a car (dark blue, black),
- **Stored vs. derived**
  - E.g.: Age computed based on DOB and current date
- **Complex & nested (composite + multi-valued)**
  - E.g.: A Person with multiple Employers has multiple work **address and in each of** them multiple phone numbers

ER Models: Attribute types - Geometries

- **A Geometry attribute is a special kind of attribute of type Geometry, that encodes the location and extent of an entity;**
- **Often, an entity with a geometry attribute is called a *feature;***
- **A feature has one or more geometries, and one or more non-geometry attributes.**


Participation constraint
Cardinality ratio
Each employee can lead a department.
EMPLOYEE
1
Partial
LEADS
1
Total
DEPARTMENT
Each Department must be lead by one Employee

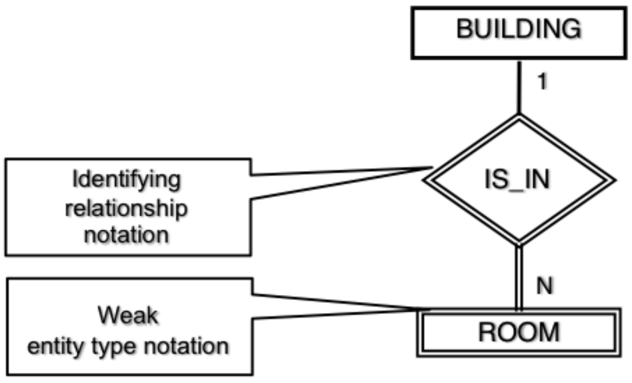
Some things can not have an autonomous existence

They are modeled as weak entity types

Weak entity types - Notation

- **Weak entity: Entity type symbol with double lines**
- **Identifying relationship: relationship symbol with double lines**

Weak entity types, Space and Time



Logical DB design

Relational model: Relations

- **A relational database is a collection of <u>relations</u> (often *visualized* as tables);**
- **Each relation has a set of <u>attributes</u> with values drawn from a particular <u>domain;</u>**
- **Relation scheme: the set of attribute names and their domains (data type + additional constraints)**
- **Database scheme is a set of relation schemes**

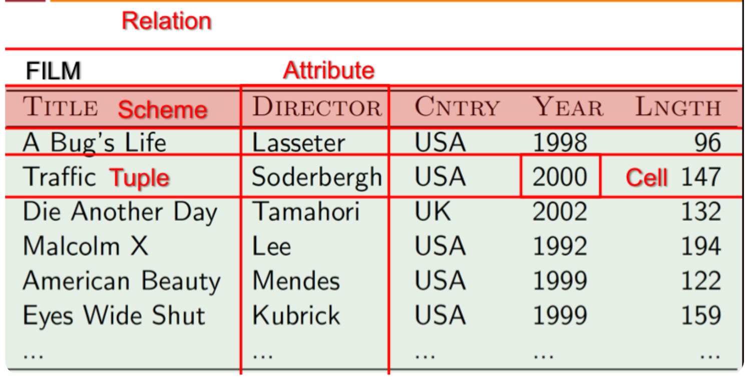
Relational model: Tuples

- **The data in a relation is structured as a set of <u>tuples;</u>**
- **Each tuple consists of data items (cells), with a single value for each attribute.**
- **Unknown or missing values represented as NULL**
- **Each tuple contains as many values as there are attributes in a relation;**
- **Each data item is drawn from the domain of values for its attribute**
- **In the <u>relational model</u>, the order of tuples is insignificant;**
- **In a relation, all tuples are <u>distinct</u> from each other.**

Relational model and databases

- **Relational database management systems** are an **implementation** of the relational model.
- **In the implementation, physical writing of data in sequence imposes ordering on row**

Example: FILM relation (as table)


Relation
Attribute
Scheme
Tuple
Cell

FILM

TITLE	DIRECTOR	CNTRY	YEAR	LNGTH
A Bug's Life	Lasseter	USA	1998	96
Traffic	Soderbergh	USA	2000	147
Die Another Day	Tamahori	UK	2002	132
Malcolm X	Lee	USA	1992	194
American Beauty	Mendes	USA	1999	122
Eyes Wide Shut	Kubrick	USA	1999	159
…	…	…	…	…

Properties of relations

- **Relations are *<u>set compatible</u>* if they have the same set of attributes;**
- **Degree of a relation: number of attributes (columns);**
- **Cardinality of a relation: number of tuples (rows);**

Deg(FILM) = 5

|FILM| = 6

TITLE	DIRECTOR	CNTRY	YEAR	LNGTH
A Bug's Life	Lasseter	USA	1998	96
Traffic	Soderbergh	USA	2000	147
Die Another Day	Tamahori	UK	2002	132
Malcolm X	Lee	USA	1992	194
American Beauty	Mendes	USA	1999	122
Eyes Wide Shut	Kubrick	USA	1999	159
…	…	…	…	…

Example: EMPLOYEE relation (as tuple)

Tuple Attribute Data Item

t = < (Name, Dick Davidson), (SSN, 422-11-2320), (HomePhone, null), (Address, 3452 Elgin Road), (OfficePhone, 749-1253), (Age, 25), (GPA, 3.53)>

Relation keys

- **Candidate key: a minimal set of attributes that uniquely identify each tuple in a relation;**
- **One candidate key is usually chosen as *Primary Key;***
- ***Will always be underlined***
- **Foreign key*:* in a relation, a FK is an attribute which is also a PK in another relation.**

STAR(NAME: *string*, BIRTH_YEAR: *integer*, GENDER: *char*, NTY: *string*)

STAR(NAME, BIRTH_YEAR, GENDER, NTY)

Integrity constraints

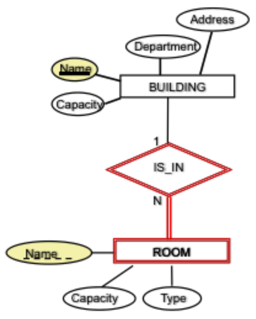
- **Domain constraint:** within each tuple in a relation, the value of an attribute A is always from the domain dom(A) of the attribute value domain;
- **Key constraint:** key attributes must uniquely identify tuples in a relation (assures that no two tuples are identical);
- **Entity integrity constraint:** no primary key can be NULL;
- **Referential integrity constraint:** relates to the linkage of tables through primary and foreign keys. Only foreign key values from the set of existing primary key values of the referred relation are allowed. The value domains of the two keys must be equal.

Normalization

- **Normalization of relations: process of organizing columns in relations in a database scheme optimizing data storage and minimizing data redundancy;**
- **Aim:** be able to edit data in a single relation – to **assure database consistency.**
- **Normalization reduces the occurrence of insertion, update and deletion anomalies:**
  - Insertion anomaly: inability to add data to the database due to absence of other data.
  - Update anomaly: data inconsistency that results from data redundancy and a partial update.
  - Deletion anomaly: unintended loss of data due to deletion of other data.

Weak entities & partial keys

- **A weak entity has a partial key**
  - A weak entity depends on its parent entity
  - It is identified by a composite key, where one part is the primary key of the parent entity, and the second part is a partial key
  - My office: Eng-B304 – there may be many blocks B, and many rooms 304. Only together I uniquely identify the office.
  - Dashed line underlying partial key attribute



Maintaining weakly spatial relationship

- The referential integrity of weakly spatial relationships can not be assured by foreign keys
- We maintain this by checking on the relationship through database triggers – little program procedures that automatically verify (*compute*) whether the relationship is satisfied (upon insert/update in DB)

Database normalisation II

- **1NF: each data item (cell) is atomic;**
  - {Address='1 Barry St'} => {Number=1, StrN='Barry', StrType='Street'}
- **2NF: 1NF + every non-key attribute is irreducibely dependent on the PK**

ENROLMENT(STUDENT_ID, COURSE_ID, GIVEN_NAME, FAMILY_NAME)

ENROLMENT(STUDENT_ID, COURSE_ID)

STUDENT(STUDENT_ID, GIVEN_NAME, FAMILY_NAME)

- **3NF: in 2NF and every non-key attribute is transitively dependent of the key attribute.**
  - Also known as: every non-key attribute must provide a fact about the key, the whole key and nothing but the key

Relational algebra

Fundamental relational operators in relational algebra:

- **Union:** binary, set-based
- **Difference:** binary, set-based
- **Product:** binary, set-based
- **Project:** unary, purely relational, requires a list of attributes
- **Restrict:** unary, purely relational, requires a restrict condition

Relational algebra is set based, closed, and generally non-associative and non-commutative

Relational operators: Union

- $A \cup B$ has all the tuples from A and B
- Duplicate tuples are coalesced (valid relations have unique tuples)
- To be well formed, A and B must be *union compatible* (have the same number of attributes from the same domains)
- Union is associative and commutative

Relational operators: Difference

- $A - B$ has all the tuples from A that are not present in B
- A and B must again be *union compatible*
- Difference is neither associative nor commutative

Relational operators: Project

- Project operator is unary $\pi_{<attribute\ list>}(A)$
- Project outputs a new relation that has a subset of attributes specified by the attribute list
- Identical tuples in the output relation are coalesced
- Reordering of project does not lead to equivalent expressions

Relational operators: Product

- $A \times B$ (Cartesian product) has all the attributes of A and B, and every tuple of A concatenated with every tuple in B
- The degree of $A \times B$ is the degree of A plus the degree of B
- The cardinality of $A \times B$ is the cardinality of A multiplied by the cardinality of B.
- A and B need not be be union compatible
- Product is associative and commutative (cf. Cartesian product)

Relational operators: Restrict

- Restrict operator is unary $\sigma_{<condition>}(A)$
- Restrict outputs a new relation that has a subset of tuples specified by the condition
- Reordering of restrict does lead to equivalent expressions

Derived operators

- **Three main *derived* operators**
- **Achieved by combination of the 5 basic operators:**
  - Intersection: binary, set based
  - Divide: binary, set based (*note*: rarely used)
  - Join: binary, relational

Derived operators: Intersection

- $A \cap B$ has all the tuples from A that are also present in B
- A and B must be union compatible
- Intersection is associative and commutative

Derived operators: Join

- $\bowtie_{att_1, att_2}(A, B)$ outputs the combined relation where tuples agree on a specified attribute (natural join)
- Join is the most computationally intensive of all relational operators to compute
- Join is associative and commutative

JOINS (Product + Restrict)