Outline

1. Entity Relationship Model
2. Normal Forms
→ the Lecturer participation in Teaches is optional, so there is a 0 as minimum cardinality on the link between Lecturer and Teaches.

→ the Student participation is mandatory, so the minimum cardinality is 1.
Superkey = Set of attributes of an entity type so that for each entity e of that type, the set of values of the attributes uniquely identifies e.

e.g. a Person may be uniquely identified by \{ Name, NI# \}

Key = is a superkey which is minimal (aka “Candidate Key”)

e.g., a Person is uniquely identified by \{ NI# \}.

Prime Attribute = attribute that appears in a key
Non-Prime Attribute = attribute that appears in no key

Simple Key consists of one attribute
Composite Key consists of more than one attribute
Primary Keys

Primary Key = a key that has been chosen as such by the database designer

→ primary key guarantees logical access to every entity (attributes of a primary key are underlined)
Cyclic Relationship Type with Roles

![Diagram of Cyclic Relationship Type with Roles]
Relationship Type with Attributes

→ Each Supplier Supplies a Part at a certain Price
Weak Entity Types

Weak Entity Type = an entity type that does not have sufficient attributes to form a primary key (double rectangle)

→ depends on the existence of an identifying (or “owner”) entity type (they have an “identifying (ID) relationship – double diamond)
→ must have a discriminator (dashed underline) for distinguishing its entities

E.g. in an employee database, Child entities exist only if their corresponding Parent employee entity exists.

The primary key of a weak entity type is the combination of the primary key of its owner type and its discriminator.
Weak Entity Types

- Parent
  - Pname
  - Address
  - NI#
  - Age

- Child
  - Cname
  - Age

Has relationship between Parent and Child
ISA Relationship Types

→ If entities of a type have special properties not shared by all entities, then this suggests two entity types with an ISA relationship between them

→ AKA generalization / specialization (supertype / subtype rel.)

E.g. an Employee ISA Person and a Student ISA Person

→ If Employee ISA Person, then Employee inherits all attributes of Person.
ISA Relationships

Attributes of Employee: Name, Number, and Salary.
Informal Methods for ERD Construction

1. Identify the entity types (including weak entity types) of the application.
2. Identify the relationship (including ISA and ID) types.
3. Classify each relationship type identified in step 2 according to its multiplicity, i.e. if it is a one-to-one, many-to-one or many-to-many.
4. Determine the participation constraints for each entity type in each relationship type.
5. Draw an ERD with the entity types and the relationship types between them.
6. Identify the attributes of entity and relationship types and their underlying domains
7. Identify a primary key for each entity type.
8. Add the attributes and primary keys to the ERD drawn in step 5.
Example
2. Normal Forms
2. Normal Forms

→ Relational database design

→ What is a good database design?

→ How many tables?

→ What goes into which table?
2. Normal Forms

Bad database design causes problems, e.g.,

→ **redundancy** (facts are stored more than once)

→ **inconsistency** through update anomalies

→ **complexity** of queries and constraints
2. Normal Forms

Bad database design causes problems, e.g.,

→ **redundancy** (facts are stored more than once)

→ **inconsistency** through update anomalies

→ **complexity** of queries and constraints

We study several **rules of thumb** for good database design

**Normal Forms**

→ First Normal Form (1NF)

→ Boyce-Codd Normal Form (BCNF)

→ Fourth Normal Form (4NF)
Normal Forms

Relational Model
→ attributes & domains: attribute $A$ takes values from $\text{Dom}(A)$
→ relation schemas and database schemas

Schema (or “table header”) of a relation (name) $R$
→ set $\text{schema}(R)$ of attributes (or “column headers”)

Database Schema
→ set of relation names together with their schemas

<table>
<thead>
<tr>
<th>ISBN</th>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-321-32132-1</td>
<td>Balloon</td>
<td>$34.00</td>
</tr>
<tr>
<td>0-55-123456-9</td>
<td>Main Street</td>
<td>$22.95</td>
</tr>
<tr>
<td>0-123-45678-0</td>
<td>Ulysses</td>
<td>$34.00</td>
</tr>
<tr>
<td>1-22-233700-0</td>
<td>Visual Basic</td>
<td>$25.00</td>
</tr>
</tbody>
</table>

$S = \{ \text{Book} \}$
$\text{schema(Book)} = \{ \text{ISBN}, \text{Price}, \text{Title} \}$

Database Instance $D$ of $S = \{ T_1, \ldots, T_n \}$ of tables $T_k$
(finite relations) of type $\text{schema}(R_k) = \{ A_1, \ldots, A_m \}$

$T_k$ fixes an order $A_1, \ldots, A_m$
If $(v_1, \ldots, v_m)$ is row of $T_k$, then $v_k$ in $\text{Dom}(A_k)$
Warning on NULL values

If \((v_1,..,v_m)\) is row of \(T_k\), then \(v_k\) in \(\text{Dom}(A_k)\)

With **SQL implementations**, this is not entirely correct. By ANSI specification of **SQL**: every column is **NULLable** by default.

→ we only know that \(v_k\) in \(\text{Dom}(A_k)\) union \{ NULL \}

**NULL** is a condition.
It means, a value is *unknown, missing, or irrelevant*.

→ Using **NULLs** can cause a lot of problems (from implementation to logic)
→ Check articles on the web!

Do not use NULLs!!!

None of the normal forms we discuss allows NULLs!
Warning on NULL values

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>HasDog</th>
<th>NI#</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steve</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OK.  (but could be inefficient)
**Warning on NULL values**

**Example**

<table>
<thead>
<tr>
<th>Name</th>
<th>HasDog</th>
<th>NI#</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steve</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OK. (but could be inefficient)

<table>
<thead>
<tr>
<th>Name</th>
<th>DogName</th>
<th>NI#</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>Einstein</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steve</td>
<td>Rex</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not OK.

→ create new table(s)
e.g. DogName
Warning on Duplicate Rows

In relational algebra, duplicate rows are not permitted.

In SQL, they are permitted.

→ be careful about this
→ do not design tables that contain duplicates
  (if you need them, administer them in a different way)

→ do not design queries that return duplicates
  (unless that is really what you want! – often it is better to return a histogram)
Duplicate Rows

In relational algebra, duplicate rows are not permitted.

In SQL, they are permitted.

→ be careful about this
→ do not design tables that contain duplicates
  (if you need them, administer them in a different way)

→ do not design queries that return duplicates
  (unless that is really what you want! – often it is better to return a histogram)

We would like that

→ the set of all attributes is a trivial superkey!

→ Then: every table has a PRIMARY KEY.
SQL

SQL queries have **Multiset Semantics**, i.e., answers contain **duplicates**.

```
SELECT Height FROM population;
1.83
1.83
1.75
1.83
1.83
```

Unless you use Set Operators (UNION, DIFFERENCE, INTERSECT, etc) (or the DISTINCT operator)

→ **duplicates** are removed!
Duplicate Rows

mysql> create table col (Number int, Color text);
mysql> insert into col values(1,"red");
mysql> insert into col values(1,"red");
mysql> select * from col;

+--------+-------+
| Number | Color |
|--------+-------|
| 1 | red |
| 1 | red |
+--------+-------+
Duplicate Rows

```sql
mysql> create table `col` (Number int, Color text,
primary key (Number, Color));
ERROR 1170 (42000): BLOB/TEXT column 'Color' used in key specification without a key length

mysql> create table `col` (Number int, Color varchar(1000),
primary key (Number, Color));
ERROR 1071 (42000): Specified key was too long; max key length is 767 bytes

mysql> create table `col` (Number int, Color varchar(100),
primary key (Number, Color));

mysql> insert into `col` values(1,"red");
mysql> insert into `col` values(1,"red");
ERROR 1062 (23000): Duplicate entry '1-red' for key 'PRIMARY'
```
Duplicate Rows

```sql
cREATE TABLE col 
  (Number INT, Color TEXT,
   primary key (Number, Color));
```

ERROR 1170 (42000): BLOB/TEXT column 'Color' used in key specification without a key length

```sql
cREATE TABLE col 
  (Number INT, Color VARCHAR(1000),
   primary key (Number, Color));
```

ERROR 1071 (42000): Specified key was too long; max key length is 767 bytes

```sql
cREATE TABLE col 
  (Number INT, Color VARCHAR(100),
   primary key (Number, Color));
```

```sql
INSERT INTO col VALUES(1, "red");
INSERT INTO col VALUES(1, "red");
```

ERROR 1062 (23000): Duplicate entry '1-red' for key 'PRIMARY'

→ when your MySQL script loads csv-files via `LOAD DATA`, then you do not see these error messages!!!
Duplicate Rows

→ depending on your PRIMARY KEY, the LOADer of MySQL will silently eliminate duplicates for you

→ this is bad practise

→ if you do it, you loose points on Assignment 1!

→ when your MySQL script loads csv-files via `LOAD DATA`, then you do not see these error messages!!!
Normal Forms

- prevent modification anomalies
- prevent data inconsistency
- make tables less redundant

while preserving information (and dependencies)

Modification anomalies:

- same information present in multiple rows. Partial updates may result in inconsistent table (i.e., providing conflicting answers) “update anomaly”

- certain facts cannot be recorded at all “insertion anomaly”

- deletion of data representing a fact may necessitate deletion of other completely different facts “deletion anomaly”
Normal Forms

1NF
2NF
3NF
BCNF
4NF
5NF
DKNF
Normal Forms

1NF = Choose Appropriate Data Types

2NF

3NF

BCNF = Do Not Represent the Same Fact Twice

4NF = Do not Store Unrelated Information in the Same Relation
Normalization

Normalization = process of bringing a given database into a given normal form

Typically, normalization is achieved by decomposing tables into smaller tables.

Decomposition in turn is realized via projections.
First Normal Form (1NF)

→ for every attribute \( A \), \( \text{Dom}(A) \) contains only atomic (indivisible) values
→ value of each attribute contains only a single value from the domain

[Codd, 1971]

→ most database systems do not allow to define tables inside tables

→ you could insert long strings (comma separated)

Never do that!

<table>
<thead>
<tr>
<th>ISBN</th>
<th>Title</th>
<th>AuName</th>
<th>AuPhone</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-55-123456-9</td>
<td>Main Street</td>
<td>Jones, Smith</td>
<td>123-333-3333, 654-223-3455</td>
</tr>
<tr>
<td>0-123-45678-0</td>
<td>Ulysses</td>
<td>Joyce</td>
<td>666-666-6666</td>
</tr>
<tr>
<td>1-22-233700-0</td>
<td>Visual Basic</td>
<td>Roman</td>
<td>444-444-4444</td>
</tr>
</tbody>
</table>
First Normal Form (1NF)

Bring a table into **1NF** through decomposition:
1) place all items of a repeating group into new table
2) duplicate in new table the primary key of the original table

<table>
<thead>
<tr>
<th>ISBN</th>
<th>Title</th>
<th>AuName</th>
<th>AuPhone</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-55-123456-9</td>
<td>Main Street</td>
<td>Jones, Smith</td>
<td>123-333-3333, 654-223-3455</td>
</tr>
<tr>
<td>0-123-45678-0</td>
<td>Ulysses</td>
<td>Joyce</td>
<td>666-666-6666</td>
</tr>
<tr>
<td>1-22-233700-0</td>
<td>Visual Basic</td>
<td>Roman</td>
<td>444-444-4444</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ISBN</th>
<th>AuName</th>
<th>AuPhone</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-321-32132-1</td>
<td>Sleepy</td>
<td>321-321-1111</td>
</tr>
<tr>
<td>0-321-32132-1</td>
<td>Snoopy</td>
<td>232-234-1234</td>
</tr>
<tr>
<td>0-321-32132-1</td>
<td>Grumpy</td>
<td>665-235-6532</td>
</tr>
<tr>
<td>0-55-123456-9</td>
<td>Jones</td>
<td>123-333-3333</td>
</tr>
<tr>
<td>0-55-123456-9</td>
<td>Smith</td>
<td>654-223-3455</td>
</tr>
<tr>
<td>0-123-45678-0</td>
<td>Joyce</td>
<td>666-666-6666</td>
</tr>
<tr>
<td>1-22-233700-0</td>
<td>Roman</td>
<td>444-444-4444</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ISBN</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-321-32132-1</td>
<td>Balloon</td>
</tr>
<tr>
<td>0-55-123456-9</td>
<td>Main Street</td>
</tr>
<tr>
<td>0-123-45678-0</td>
<td>Ulysses</td>
</tr>
<tr>
<td>1-22-233700-0</td>
<td>Visual Basic</td>
</tr>
</tbody>
</table>
First Normal Form (1NF)

→ for every attribute $A$, $\text{Dom}(A)$ contains only atomic (indivisible) values
→ value of each attribute contains only a single value from the domain

[Codd, 1971]

“atomic value” = “value that cannot be decomposed”

Problematic

→ Character string?
→ Fixed-point number?
→ ISBN (includes language and publisher identifiers)?

C. J. Date: “The notion of atomicity has no absolute meaning”
Table versus Relation

Chris Date, “What First Normal Form Really Means” (2000)

A table is in 1NF if and only if it is “isomorphic to some relation”, specifically:

1. There is no top-to-bottom ordering on the rows.
2. There is no left-to-right ordering on the columns.
3. There are no duplicate rows.
4. Every row-and-column intersection contains exactly one value from the applicable domain (and nothing else).
5. All columns are regular [i.e. rows have no hidden components such as row IDs, object IDs, or hidden timestamps].
Second Normal Form (2NF)

A table is in 2NF, if

→ it is in 1NF
→ every non-prime attribute depends on the whole of every candidate key

Example (Not 2NF)

Schema(R) = \{City, Street, HouseNumber, HouseColor, CityPopulation\}

1. \{City, Street, HouseNumber\} \rightarrow \{HouseColor\}
2. \{City\} \rightarrow \{CityPopulation\}
3. CityPopulation is non prime
4. CityPopulation depends on \{City\} which is NOT the whole of the (unique) candidate key \{City, Street, HouseNumber\}

Functional dependency  \[D \rightarrow E: \text{for every } D\text{-tuple, there is at most one } E\text{-tuple}
\]

“E (functionally) depends on D”
Second Normal Form (2NF)

→ do you see the potential Redundancy in this table?

Example (Not 2NF)
Schema(R) = \{City, Street, HouseNumber, HouseColor, CityPopulation\}

1. \{City, Street, HouseNumber\} → \{HouseColor\}
2. \{City\} → \{CityPopulation\}
3. CityPopulation is non prime
4. CityPopulation depends on \{City\} which is NOT the whole of the (unique) candidate key \{City, Street, HouseNumber\}

Functional dependency \(D \rightarrow E\): for every D-tuple, there is at most one E-tuple “E (functionally) depends on D”
Second Normal Form (2NF)

Bring a 1NF table into 2NF
→ move an attribute depending on a strict subset of a candidate key into a new table, together with this strict subset
→ the strict subset becomes the key of the new table

Example (Convert to 2NF)
Old Schema → {City, Street, HouseNumber, HouseColor, CityPopulation}
New Schema → {City, Street, HouseNumber, HouseColor}
New Schema → {City, CityPopulation}
END

Lecture 5