# Database Normalization

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

# Database Design Process

- **Requirement analysis**
  - Based on use cases, business process descriptions
- **Conceptual design**
  - Model what the DB is about, e.g. via ER diagrams
- **Schema normalization**
  - E.g., reduce data redundancy via transformation
- **Physical tuning**
  - E.g., decide which indices to create or sort order

# Database Design Process

- **Requirement analysis**
  - Based on use cases, business process descriptions
- **Conceptual design**
  - Model what the DB is about, e.g. via ER diagrams
- **Schema normalization**
  - E.g., reduce data redundancy via transformation
- **Physical tuning**
  - E.g., decide which indices to create or sort order

# Database Design Process

- **Requirement analysis**
  - Based on use cases, business process descriptions
- **Conceptual design**
  - Model what the DB is about, e.g. via ER diagrams
- **Schema normalization**
  - E.g., reduce data redundancy via transformation
- **Physical tuning**
  - E.g., decide which indices to create or sort order

**[RG Sec.15]**

# Schema Normalization

- Can prepare **first sketch** of database schema via ER
- Resulting schema will most likely be **sub-optimal**
  - I.e., the schema implies lots of **data redundancy**
  - Data redundancy leads to various **problems**
- Optimize initial schema via **schema normalization**

# Roadmap

- **Functional dependencies**: indicate redundancy
- **Normal forms**: desirable formal schema properties
- **Normalization algorithms**: transform to normal form

# Roadmap

- **Functional dependencies**: indicate redundancy
- **Normal forms**: desirable formal schema properties
- **Normalization algorithms**: transform to normal form

*Slides by Immanuel Trummer, Cornell University*

# Functional Dependency (FD)

- Used to detect data **redundancies** (want to remove)
- Values in some columns uniquely **decide** values in others
- Notation: **X → Y** means values in X decide values in Y

# Example Dependencies

| TA Name | Hours | Salary |
|---|---|---|
| John | Full Time | 1,000 |
| Mike | Part Time | 500 |
| Anna | Part Time | 500 |
| Lisa | Full Time | 1,000 |

**Hours → Salary**

*Slides by Immanuel Trummer, Cornell University*

# Example Dependencies

| TA Name | Hours | Salary |
|---|---|---|
| John | Full Time | 1,000 |
| Mike | Part Time | 500 |
| Anna | Part Time | 500 |
| Lisa | Full Time | 1,000 |

**What's the Problem?**

**Hours → Salary**

# Problems

- **Update anomaly**: could make TA salaries inconsistent
- **Insertion anomaly**: could lack salary info for new hours
- **Deletion anomaly**: could lose salary info after deletions

# Example Solution

| TA Name | Hours |
|---|---|
| John | Full Time |
| Mike | Part Time |
| Anna | Part Time |
| Lisa | Full Time |

| Hours | Salary |
|---|---|
| Full Time | 1,000 |
| Part Time | 500 |

**Hours → Salary**

# Solution Analysis

- We removed redundancy by **decomposing** table
  - FD **does not** connect columns in same table
  - Prior **anomalies** cannot happen anymore
- Must **avoid data loss** via decomposition
  - Can **reconstruct** based on FD (Hours → Salary)
  - Recompose by **looking up** Salary for Hours value
- Decomposing table may require additional **joins**!

# Functional Dependencies and Redundancy

- **FDs** state that values in X determine values in Y
- **Redundant** storage of Y if X stored multiple times
  - **Sufficient** to store Y once for each X value
- Want to design schema to **avoid** this in each case
  - Considering all possible **future** database states
- Want to avoid storing X and Y **in same table**, except ...

# When To Allow All FD Columns in Same Table?

# Finding FDs

- Common mistake: try finding FDs by **looking at data**
- Data only captures **current state** of the database
  - **Not all** functional dependencies may appear
  - Data may suggest misleading "**pseudo FDs**"
- Two valid **sources** for mining FDs:
  - **Domain knowledge**
  - **Inferring** new FDs from given FDs

# Inferring FDs

- Notation **F1 |= F2** means FDs F1 **imply** FDs F2
  - No relation can satisfy F1 without satisfying F2
- Can infer all FDs by applying **Armstrong's Axioms**:
  - **Reflexivity**: if $Y \subseteq X$ then $X \rightarrow Y$ is implied
  - **Augmentation**: if $X \rightarrow Y$ then $XZ \rightarrow YZ$ for any Z
  - **Transitivity**: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

# FD Closure

- **Closure** of a set of FDs are all implied FDs
- F+ = {f|F |= f}
- Can be calculated using **Armstrong's axioms**
- F is a **cover** for G if F+ = G+
- The closure can be **extremely large**

# Example: Inferring FDs

- F={
  {Course} → {Lecturer},
  {Course} → {Department},
  {Lecturer, Department} → {Office}
  }

- FDs **inferred** from F:
  - {Course, Department} → {Department}
  - {Course, Lecturer} → {Department, Lecturer}
  - {Course} → {Office}

# Example: Inferring FDs

- F={
  {Course} → {Lecturer},
  {Course} → {Department},
  {Lecturer, Department} → {Office}
  }

- FDs **inferred** from F:
  - {Course, Department} → {Department}
  - {Course, Lecturer} → {Department, Lecturer}
  - {Course} → {Office}

# Details on Inference

1. {Course} → {Lecturer}
2. {Course} → {Department}
3. {Lecturer, Department} → {Office}

***Given***

---

4. {Course} → {Course, Lecturer}
5. {Course, Lecturer} → {Lecturer, Department}
6. {Course} → {Lecturer, Department}
7. {Course} → {Office}

***Inferred***

# Attribute Closure

- Entire closure is typically **too large** to be useful
- Attribute closure gets all FDs for **fixed left attributes**
  - **X+** for attributes X is attribute closure
- Useful for checking if one **specific** FD is implied

# Finding Attribute Closures

- **Goal**: get attribute closure of X given FDs F
- **Repeat** until no changes
  - Start with **closure X**
  - **Iterate** over all FDs A → B in F
    - **If** closure ⊆ A **then** add B to closure

# Example: Attribute Closures

- **F = {A → D, AB → E, BI → E, CD → I, E → C}**
  - Want to find attribute closure **(AE)+**
  - Before Iteration 1: closure is **(AE)**
  - Before Iteration 2: closure is **(ACDE)**
  - Before Iteration 3: closure is **(ACDEI)**
  - (No change in Iteration 3)

# Attribute Closure Complexity

- **Goal**: get attribute closure of X given FDs F
- **Repeat** until no changes
  - Start with **closure X**
  - **Iterate** over all FDs A → B in F
    - **If** closure ⊆ A **then** add B to closure

# Attribute Closure Complexity

- **Goal**: get attribute closure of X given FDs F
- **Repeat** until no changes
  - Start with **closure X**
  - **Iterate** over all FDs A → B in F
    - **If** closure ⊆ A **then** add B to closure **O(Nr. Attributes)**

*Slides by Immanuel Trummer, Cornell University*

# Attribute Closure Complexity

- **Goal**: get attribute closure of X given FDs F
- **Repeat** until no changes
  - Start with **closure X**
  - **Iterate** over all FDs A → B in F — **O(Nr. FDs) Iterations**
    - **If** closure ⊆ A **then** add B to closure — **O(Nr. Attributes)**

*Slides by Immanuel Trummer, Cornell University*

# Attribute Closure Complexity

- **Goal**: get attribute closure of X given FDs F
- **Repeat** until no changes — **O(Nr. FDs) Iterations**
  - Start with **closure X**
  - **Iterate** over all FDs A → B in F — **O(Nr. FDs) Iterations**
    - **If** closure ⊆ A **then** add B to closure — **O(Nr. Attributes)**

*Slides by Immanuel Trummer, Cornell University*

# Attribute Closure Complexity

- **Goal**: get attribute closure of X given FDs F
- **Repeat** until no changes — **O(Nr. FDs) Iterations**
  - Start with **closure X**
  - **Iterate** over all FDs A → B in F — **O(Nr. FDs) Iterations**
    - **If** closure ⊆ A **then** add B to closure — **O(Nr. Attributes)**

***Complexity in O(Nr. FDs² * Nr. Attributes)***

*Slides by Immanuel Trummer, Cornell University*

# Finding All Relation Keys

- **Iterate** over all attribute sets A
- **Check** if A is a key:
  - Calculate **attribute closure** (A)+
  - It's a **key** if (A)+ includes all attributes

# Roadmap

- **Functional dependencies**: indicate redundancy
- **Normal forms**: desirable formal schema properties
- **Normalization algorithms**: transform to normal form

# Boyce-Codd Normal Form (BCNF)

- A schema if in **BCNF** if the following conditions holds
  - **For all FDs** A → b whose attributes are in same table
    - Either b is element in A ("**trivial**" FD)
    - Or A **contains a key** of its associated table
- This must apply to given and inferred FDs!

***Does not permit any redundancy!***

# BCNF Example

- **For all FDs** A → b with attributes in same table
  - Either b is element in A ("**trivial**" FD)
  - Or A **contains a key** of its associated table
- ***Is this schema in BCNF?***
  - **Schema**: $table_1$(a,b), $table_2$(a,d,e), $table_3$(c,d)
  - **(Initial) FDs**: {a → b, bc → d, a → c, d → ae}

# Third Normal Form (3NF)

- A schema if in **3NF** if the following conditions holds
  - **For all FDs** A → b whose attributes are in same table
    - Either b is element in A ("**trivial**" FD)
    - Or A **contains a key** of its associated table
    - Or b **is part** of some minimal key
- This must apply to given and inferred FDs!

# Third Normal Form (3NF)

- A schema if in **3NF** if the following conditions holds
  - **For all FDs** A → b whose attributes are in same table
    - Either b is element in A ("**trivial**" FD)
    - Or A **contains a key** of its associated table
    - Or b **is part** of some minimal key

**Allows Some Redundancy**

- This must apply to given and inferred FDs!

# 3NF Example

- **For all FDs** A → b with attributes in same table
  - Either b is element in A ("**trivial**" FD)
  - Or A **contains a key** of its associated table
  - Or b **is part** of some minimal key
- ***Is this schema in 3NF?***
  - **Schema**: $table_1$(a,b), $table_2$(a,d,e), $table_3$(c,d)
  - **(Initial) FDs**: {a → b, bc → d, a → c, d → ae}

# Comparison of Normal Forms

- **BCNF** disallows any redundancy
  - **Pro**: avoids all negative effects of redundancy
  - **Con**: may require breaking up dependencies
- **3NF** allows redundancy in some cases
  - **Pro**: can always preserve dependencies
  - **Con**: may still have some negative effects

# Comparison of Normal Forms

- **BCNF** disallows any redundancy
  - **Pro**: avoids all negative effects of redundancy
  - **Con**: may require breaking up dependencies
- **3NF** allows redundancy in some cases
  - **Pro**: can always preserve dependencies
  - **Con**: may still have some negative effects

***I.e., verifying FDs may require joins***

# Roadmap

- **Functional dependencies**: indicate redundancy
- **Normal forms**: desirable formal schema properties
- **Normalization algorithms**: transform to normal form

# Decomposition

- Normal forms impose conditions on FDs in **single table**
- **Decompose** tables into smaller tables to satisfy them
- Decomposition must allow to **reconstruct** original data
  - Assume we decomposed **R into X and Y**
  - We can do so if **X∩Y→X** or **X∩Y→Y** is an FD
    - Can then **match** each row from Y to one row from X/
      Can then **match** each row from X to one row from Y

# Lossless Decomposition

| TA Name | Hours | Office |
|---|---|---|
| John | Full Time | 401b |
| Mike | Part Time | 205 |
| Anna | Part Time | 310 |
| Lisa | Full Time | 112 |

| Hours | Salary |
|---|---|
| Full Time | 1,000 |
| Part Time | 500 |

**Hours → Salary**

*Slides by Immanuel Trummer, Cornell University*

# Lossless Recomposition

| TA Name | Hours | Salary | Office |
|---|---|---|---|
| John | Full Time | 1,000 | 401b |
| Mike | Part Time | 500 | 205 |
| Anna | Part Time | 500 | 310 |
| Lisa | Full Time | 1,000 | 112 |

**Hours → Salary**

*Slides by Immanuel Trummer, Cornell University*

# Lossy Decomposition

| TA Name | Hours |
|---|---|
| John | Full Time |
| Mike | Part Time |
| Anna | Part Time |
| Lisa | Full Time |

| Hours | Salary | Office |
|---|---|---|
| Full Time | 1,000 | 401b |
| Part Time | 500 | 205 |
| Part Time | 500 | 310 |
| Full Time | 1,000 | 112 |

**Hours → Salary**

# Lossy Recomposition

| TA Name | Hours | Salary | Office |
|---|---|---|---|
| John | Full Time | 1,000 | 401b |
| Mike | Part Time | 500 | 205 |
| Anna | Part Time | 500 | 310 |
| Lisa | Full Time | 1,000 | 112 |
| John | Full Time | 1,000 | 112 |
| Mike | Part Time | 500 | 310 |
| Anna | Part Time | 500 | 205 |
| Lisa | Full Time | 1,000 | 401b |

**Hours → Salary**

# Towards BCNF

- **Repeat** while some FD A→b on R violates BCNF rules
  - **Decompose** R into R-b and Ab
- All decompositions are **lossless** as (R-b)∩Ab=A→b
- Will **terminate** as tables get smaller and smaller
- End result may **depend** on decomposition order

# BCNF Example

- CSJDPQV, key C, JP→C, SD→P, J→S

- ***Bring this into BCNF!***

# BCNF (One) Solution

- CSJDPQV, key C, JP→C, SD→P, J→S
- For SD→P, **decompose** into SDP, CSJDQV
- For J→S, **decomposes** CSJDQV into JS, CJDQV
- **Final** database schema: SDP, JS, CJDQV

# Dependency Preservation

- Assume we **decompose** R into X and Y
- Assume we **enforce** FDs on X and Y separately
  - I.e., we enforce all FDs that only use attributes on X
  - Then we enforce all FDs that only use attributes on Y
- This enforces all FDs on R if **dependency preserving**

# Decomposition Properties

- Reminder: **lossless** vs. **dependency preserving**
- **None** of the two properties implies the other!
- **BCNF** may lose dependency preservation
- **3NF** fixes that

# Towards 3NF

- Same procedure as for BCNF with one **extension**
  - If dependency A→b **broken** then add relation Ab
- Want to use **minimal cover** FDs for this!
  - Right hand side of each FD is a **single attribute**
  - Closure changes when **deleting any FD**
  - Closure changes when **deleting any attribute**

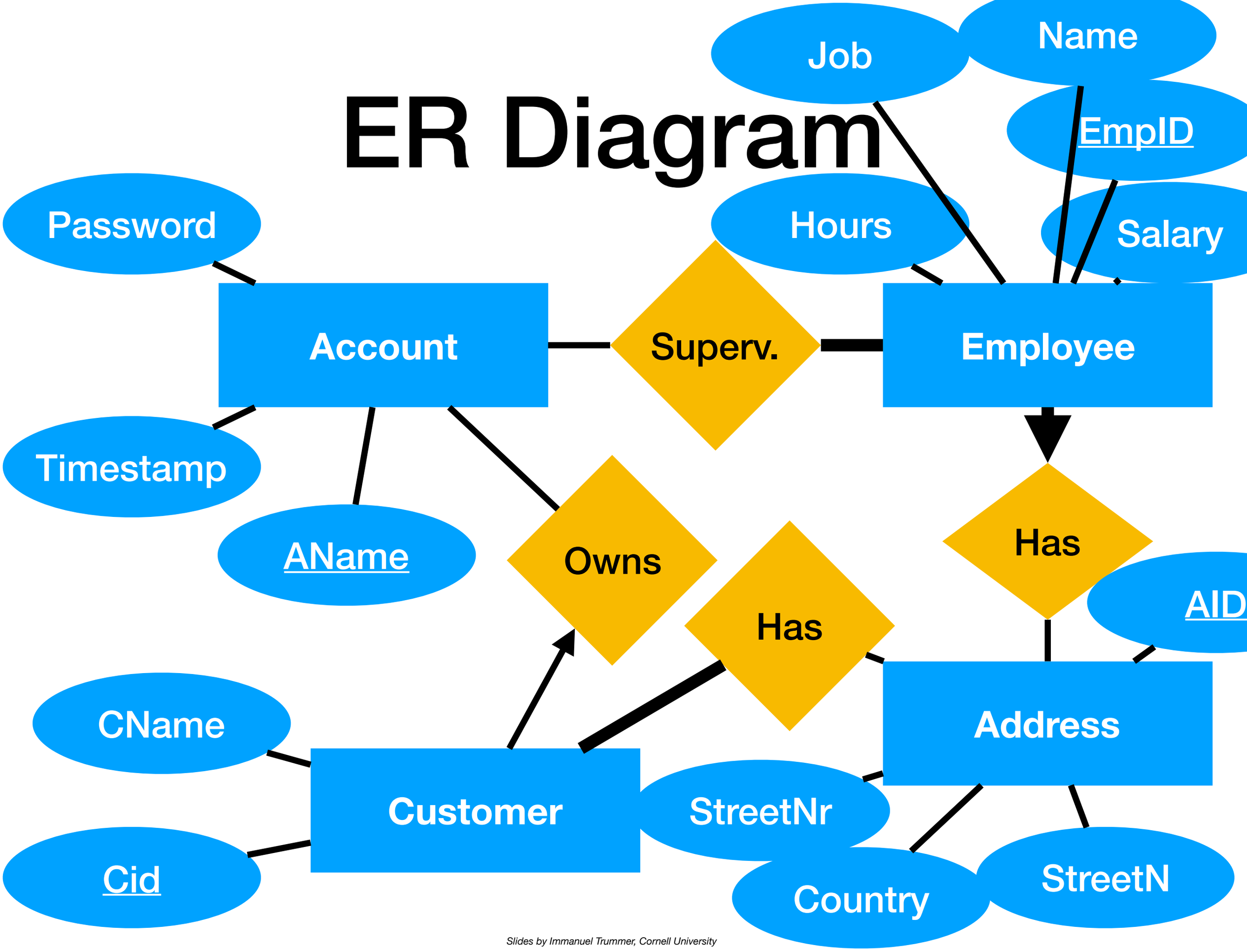
# DB Design Example

*Slides by Immanuel Trummer, Cornell University*

# *Draw ER diagram*

- We design the database for a **Web shop**, described below
- **Accounts** have a (unique) name, a password, and a timestamp of the last login.
- **Customers** own at most one account, they have a customer ID (unique), a name, and one or multiple addresses.
- Each **address** has an address ID, a street name and number, and an associated country.
- **Employees** have an employee ID (unique), a name, one primary address, and at least one assigned account to supervise. They are associated with a job title, the number of hours worked weekly, and salary.

# ER Diagram




*Slides by Immanuel Trummer, Cornell University*

# *Translate into Schema!*

# DB Schema - First Draft

- **CustomerWaccount**(Cid,CName,Aname, Timestamp,Password)
- **Employee**(EmpID,Name,Job,Hours,Salary,AID)
- **Supervises**(EmpID,Cid)
- **Address**(AID,StreetN,StreetNr,Country)
- **CustomerHasAddress**(Cid,AID)

# Bring to BCNF

# Normalized Schema

- Functional **dependencies**:
  - Key and trivial constraints (allowed by BCNF)
  - Job, Hours → Salary (needs to be resolved!)
- **Decompose** Employee table:
  - From Employee(EmpID,Name,Job,Hours,Salary,AID)
  - Into Employee(EmpID,Name,Job,Hours,AID), JobHoursToSalary(Job,Hours,Salary)