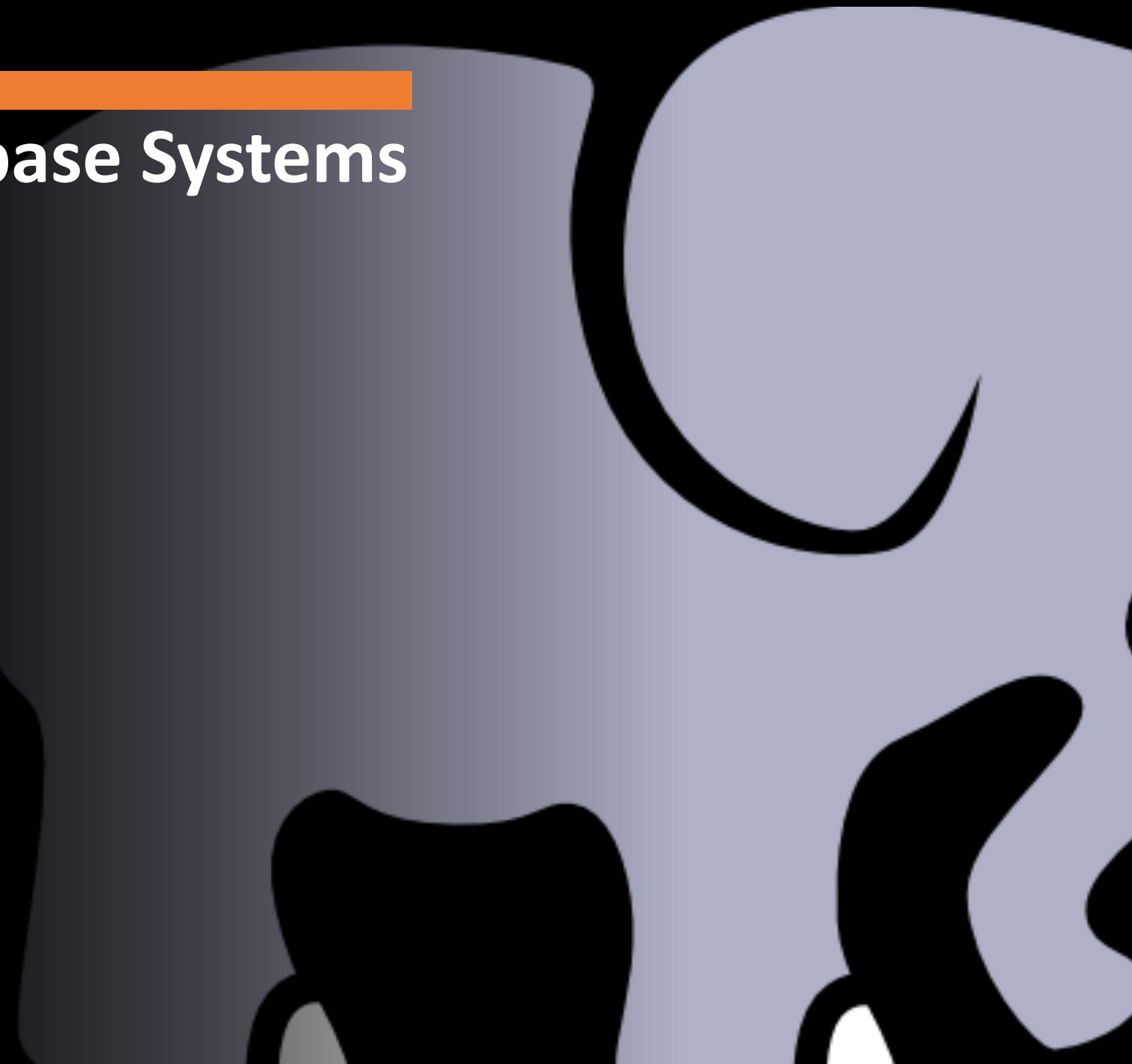
Introduction to Database Systems
I2DBS – Spring 2023

- Week 6:
- Normalization

Jorge-Arnulfo Quiané-Ruiz



Readings:

PDBM 6.2

Information

- **Homework 1:**
  - We provide feedback and solution this week
  - Also review document on LearnIT with common errors
- **Homework 2 due on March 24th at 23:59**
  - Use the exercise session to work on this!
  - Remember: 3 out of 4 mandatory; low bar for acceptance!
- **How will you submit ER diagrams in the exam?**
  - If you do it with software: Submit PNG/JPG/PDF file
  - If you do it by hand: Submit paper and proctors will scan
- **Exercise 7 (next week):**
  - Normalisation from this week (Lecture 6)
  - OS/Storage/Multicores from next week (Lecture 7)

Looking for an RA

APACHE wayang

Requirements:
- Good Java skills
- Linux, macOS, windows experience
- Highly motivated

Wish-List:
- Interested in open-source software
- Looking for system building experience
- Curious and creative developer

Position and Role
- Research Assistant for one year (half time)
- Playing and having fun with Apache Wayang
- Help us to develop and grow Apache Wayang

Interested?
- Send your CV and cover (motivation) letter to: joqu@itu.dk

Profile of the Week

IT University of Copenhagen

Raymond A. Lorie

Pioneer of Normalization Theory

- **1936:** Detroit, USA
- **1966:** PhD in Electrical Engineering from MIT
- **1966:** joined IBM Research in San Jose
- **1974:** co-invented System R
- **1974:** introduction of functional dependencies
- **1984:** ACM SIGMOD Contributions Award
- **1999:** ACM SIGMOD Innovations Award
- **2002:** IEEE John von Neumann Medal
- **2016:** R.I.P.

Design Outline

- **Conceptual Model**
  - ER notation
  - Transformation to DB schema
- **Improving the Design**
  - Normalization (this week)
  - Performance tuning (weeks 7-9)

Normalization

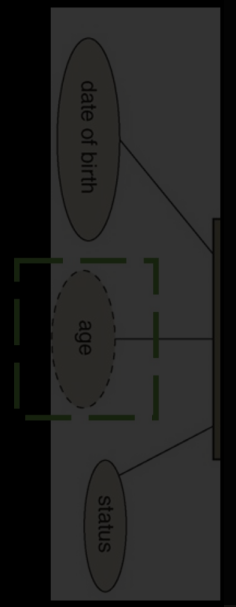
Readings:

PDBM 3.0-3.3, 6.3-6.4

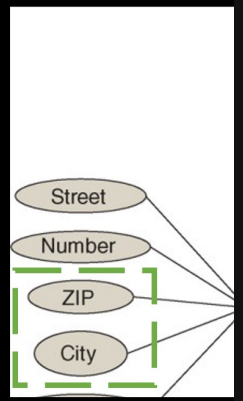
Motivation From Week 5 / Slide 15

Derived Attributes in SQL DDL

- Not discussed in the PDBM book!
- Option 1: Create an attribute and maintain it
  - E.g. with a trigger, or regular update processes
- Option 2: Create a view that computes it
- Neither is very good!



- **Sometimes there is a second kind of inter-attribute relationship**
  - Here: ZIP ⟶ City
  - Called functional dependencies (FDs)
  - ER diagrams may miss such relationships!
  - We fix this with normalization (week 6)



Redundancy Issues

- **Consider the table below Person(ID, Name, ZIP, City)**
- **What can go wrong during:**
  - Insert? – what if new person lives in København V?
  - Update? – what if a municipality is renamed?
  - Delete? – what if Johan is deleted?
  - Also: Extra storage

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

Solution: Decompose the Relation

Person

ID	Name	ZIP
1	Björn	2100
2	Johan	2300
3	Peter	2100

ZIP

ZIP	City
2100	København Ø
2300	København S

```css
SELECT P.ID, P.Name, P.ZIP, Z.City
FROM Person P
JOIN ZIP Z ON P.ZIP = Z.ZIP;
```

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

How to Decompose?

1. Add new relation
2. Fill new relation
3. Alter old relation

```sql
CREATE TABLE ZIP (
    ZIP INT PRIMARY KEY,
    City CHARACTER VARYING NOT NULL
);

INSERT INTO ZIP
SELECT DISTINCT ZIP, City
FROM Person;

ALTER TABLE Person ADD
    FOREIGN KEY (ZIP) REFERENCES ZIP(ZIP);

ALTER TABLE Person DROP COLUMN City;
```

Practice: How to Decompose?

1. **Add new relation(s)**
2. **Fill new relation(s)**
3. **Leave old relation unchanged!**
   - Join of new relations –
     should give same data as old relation!

sql

```sql
-- CREATE and INSERT INTO ZIP as before
-- No ALTER TABLE statements

CREATE TABLE NewPerson (
    ID INT PRIMARY KEY,
    Name CHARACTER VARYING NOT NULL,
    ZIP INT NOT NULL REFERENCES ZIP(ZIP)
);

INSERT INTO NewPerson
SELECT ID, Name, ZIP
FROM Person;
```

Redundancy Issues -- Revisited

- **Consider the table below Person(ID, Name, ZIP, City)**
- **What can go wrong during:**
  - Insert? – what if new person lives in København V?
  - Update? – what if a municipality is renamed?
  - Delete? – what if Johan is deleted?
  - Also: ~~Extra storage~~

Person

ID	Name	ZIP
1	Björn	2100
2	Johan	2300
3	Peter	2100

ZIP

ZIP	City
2100	København Ø
2300	København S

What Really Happened?

- **Consider the table below Person(ID, Name, ZIP, City)**

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

- **Problem: Functional Dependency ZIP ⟶ City**
  - If two records have the same ZIP value, they are guaranteed to have the same City value
  - ER does not capture this relationship easily
- **Solution: Decomposition!**

Towards Normal Forms

- **This table is in 2NF**
  - Person: ID ⟶ Name
  - Person: ID ⟶ ZIP
  - Person: ID ⟶ City
  - Person: ZIP ⟶ City (transitive FD)

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

- **These tables are in BCNF**
  - <key> ⟶ <attribute>
  - Person: ID ⟶ Name
  - Person: ID ⟶ ZIP
  - ZIP: ZIP ⟶ City

Person

ID	Name	ZIP
1	Björn	2100
2	Johan	2300
3	Peter	2100

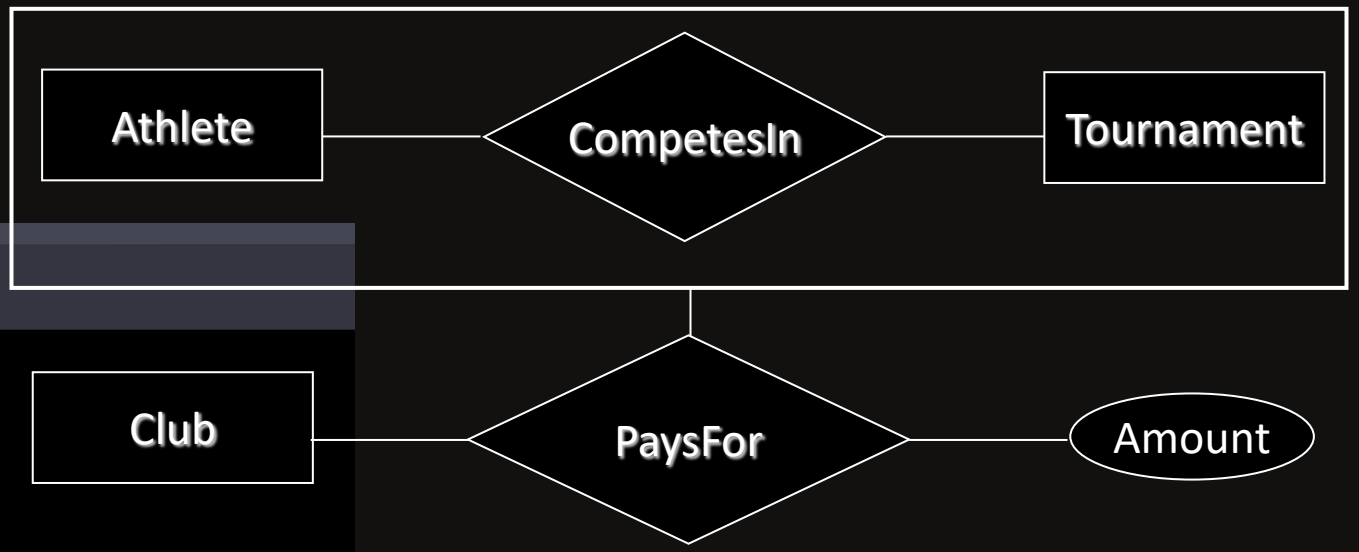
ZIP

ZIP	City
2100	København Ø
2300	København S

- **We like BCNF = no redundancy!**
- **Every attribute is dependent on the key, the whole key and nothing but the key!**

Another Example: Mistake in Aggregation?

- Consider the PaysFor relationship from ER slides




```sql
CREATE TABLE PaysFor (
  AID INTEGER,
  TID INTEGER,
  CID INTEGER REFERENCES Clubs,
  amount INTEGER NOT NULL,
  FOREIGN KEY (AID, TID) REFERENCES CompetesIn (AID, TID),
  PRIMARY KEY (AID, TID, CID)
);
```

Another Example: Add Data

- **Perhaps AID and CID are related?**
  - What if AID ⟶ CID – athlete belongs to one club!
  - Means that the key is in fact only AID, TID

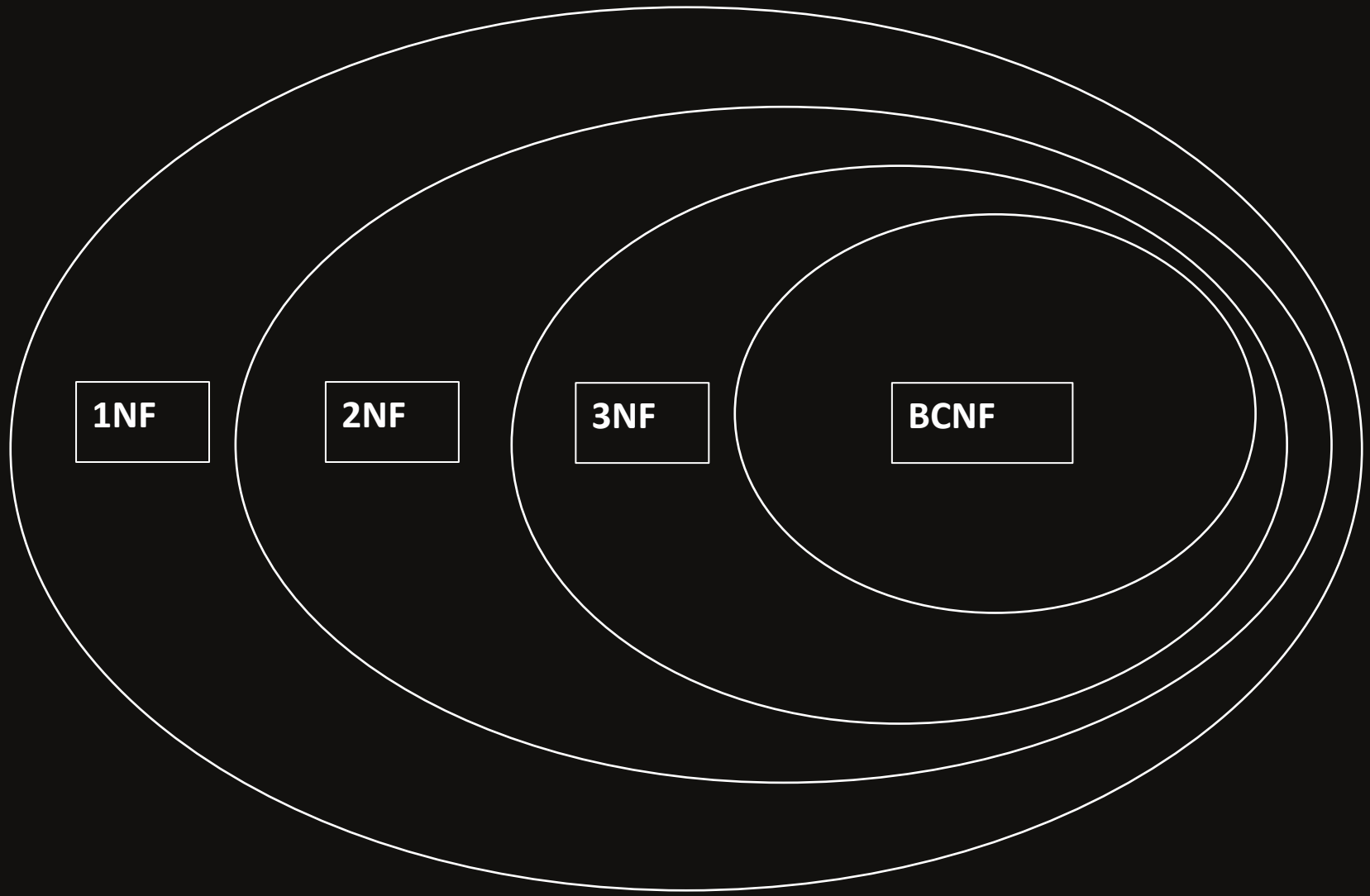
```sql
CREATE TABLE PaysFor (
  AID INTEGER,
  TID INTEGER,
  CID INTEGER REFERENCES Clubs,
  amount INTEGER NOT NULL,
  FOREIGN KEY (AID, TID) REFERENCES CompetesIn (AID, TID),
  PRIMARY KEY (AID, TID, CID)
);
```

PaysFor

AID	TID	CID	amount
1	1	1	1000
1	2	1	5000
1	3	1	600
2	1	1	1000
3	2	2	1000

- **This table is in 1NF**
  - AID, TID ⟶ Amount
  - AID ⟶ CID

Normal Forms



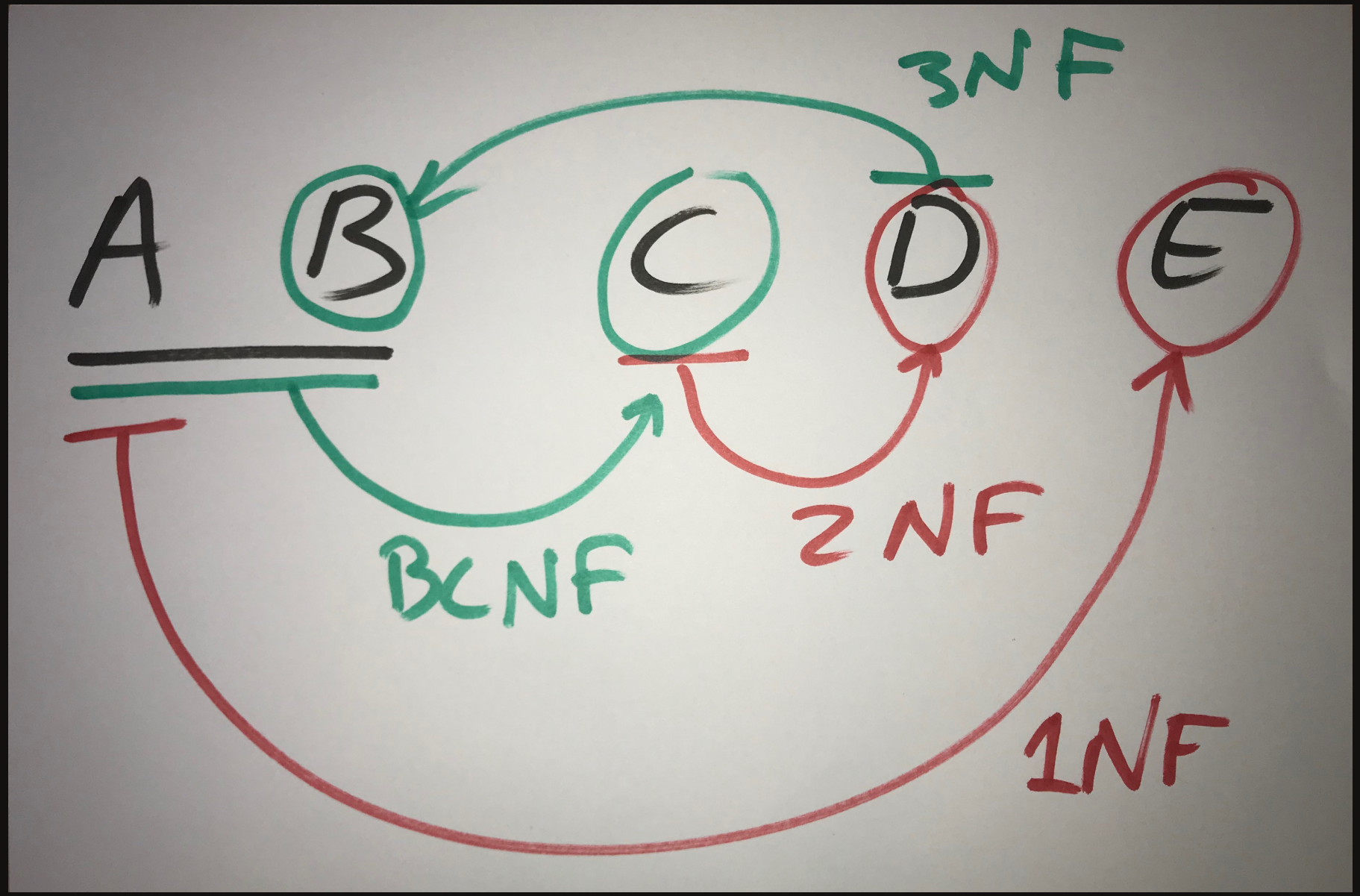
Third Normal Form = 3NF

- **We saw 1NF, 2NF and BCNF (= πNF, 3.5NF)**
- **What is 3NF?**

- **Prototype: R(A, B, C), C ⟶ A**
- **Example from Iceland:**
  - Boats(Area, Number, ZIP, …)
  - Area = TH, ÞH, AK, …
  - ZIP ⟶ Area
  - This is 3NF

- **We do NOT normalise this!**
  - We DO note that (Number, ZIP) is also a key (UNIQUE)

Visualizing FDs and Normal Forms



Can We Get a Simple Decomposition Algorithm?

Preview: Simple Decomposition Algorithm!

1. **Find all important functional dependencies (FDs)**
2. **While FD < 3NF exists:**
   Decompose!

- **Will result in BCNF or 3NF**
  - No way from 3NF to BCNF …
  - … except by losing a dependency

Flash-Back to Keys I

- **A candidate key for a relation is a set K of its attributes that satisfy:**
  - Uniqueness: The values of the attribute(s) in K uniquely identify a tuple
  - Minimality: The uniqueness property goes away if we remove any attribute from K
- **If only uniqueness is satisfied the attributes are said to form a superkey**
- **Example:**
  Person(ID, CPR, Name, ZIP)
  (ID) is a candidate key
  (CPR) is a candidate key
  (CPR, Name) is a superkey
  (ZIP, Name) is not a key

Flash-Back to Keys II

- **Important: Candidate key is defined with respect to what data can legally occur, not with respect to any particular instance of the relation**
  - Non-unique attributes are not keys
  - Unique attributes MAY be keys

- **One candidate key is selected as the primary key**
  - There could be several candidate keys to choose from
  - Example: Student(id, cpr, exam_no, email, name)
  - *For normalization, it is irrelevant which key is chosen as primary key*

Keys and Functional Dependencies

- **Key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD**
  - ID → ID, Name, ZIP

  OR
  - ID → ID, ID → Name, ID □ ZIP

  OR
  - ID → Name, ID → ZIP (since ID → ID is clearly true)

- **Keys are unique:**
  - The left side cannot have the same values
  - … but other attributes can

Functional Dependencies (FDs)

- **Definition:**
  - A functional dependency (FD) on a relation schema R is a constraint X → Y, where X and Y are subsets of attributes of R
  - A FD X → Y is satisfied in an instance r of R if for every pair of tuples, t and s:
    if t and s agree on all attributes in X then they must agree on all attributes in Y
  - Confused? Think about X = ZIP → Y = City

Note

Intuitively: If X and Y were the only attributes in a relation, then X would be a key!

Two (Un-)Important Types of FDs

- **FDs with a superkey on the left are unavoidable**
  - If A is a candidate key for a relation then clearly A→B for any attribute B
  - Similarly if {A1,A2} forms a superkey we have A1A2→B for any B, etc
- **FDs like X → A are trivial if A ∈ X**
  - Therefore X → Y is trivial if Y ⊆ X
- **They are (un-)important because they do not require decomposition!**

Redundant FDs

- **If we already know that ID → ZIP is ID, Name → ZIP useful?**
- **If we already know that ID → ZIP and ZIP → City is ID → City useful?**
- **These are examples of redundant FDs**
  - Can be computed from others
  - Do not require decomposition!
- **Can be formally defined and computed**
  - ... but you'll know them when you see them!

What is Decomposition?

- **Consider relation R and (important) functional dependency X → Y that violates 3NF/BCNF**
  - Decompose R into R1 and R2 where
    - R1 = R – Y (everything but Y = the right side)
    - R2 = XY (the whole FD = both left and right side)
- **This has the following nice properties**
  - R2 is (normally) in BCNF
  - Joining R1 and R2 (with = on all X attributes) yields R
- **Example: Person(ID, Name, ZIP, City), ZIP ⟶ City**
  - X = ZIP, Y = City
  - R-Y = Person(ID, Name, ZIP)
  - XY = ZIP(ZIP, City)

Our Simple Decomposition Algorithm!

1. **Find all the FDs**
   Remove trivial, unavoidable, redundant FDs
2. **While FD < 3NF exists:**
   Decompose!

- **Will result in BCNF or 3NF**
  - No way from 3NF to BCNF …
  - … except by losing a dependency
  - … so we don't decompose 3NF!!!

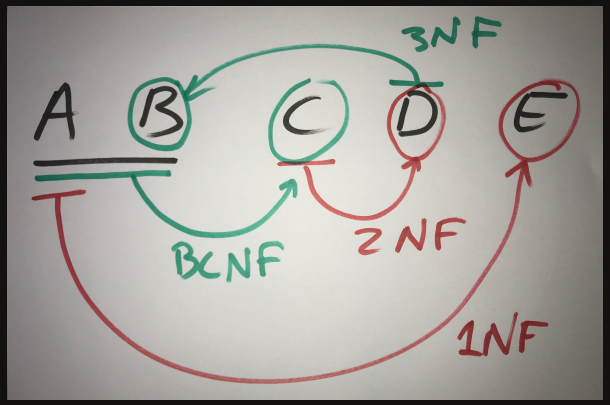
Example: Decomposition

- R = ABCD
- AB → CD
- C → D

- New Table: CD
- Old Table: ABC

- Both in BCNF
- What happened to AB → D?

X → Y
R1 = R – Y
R2 = XY



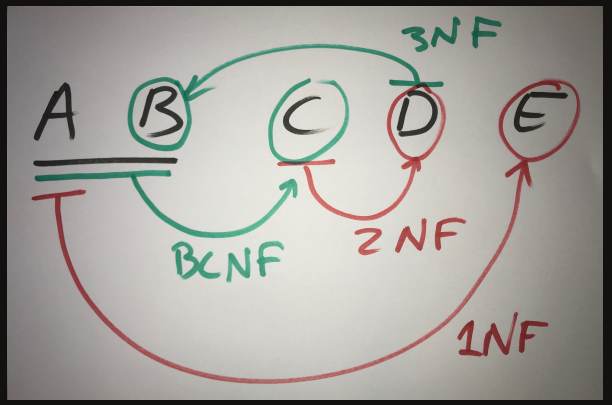
Example: Decomposition

- **R = ABCD (AB → CD)**
- **A → D**
  - A = personID, B = projectID, C = start date, D = name of person

$X \rightarrow Y$
R1 = R – Y
R2 = XY

- **New Table: AD**
- **Old Table: ABC**

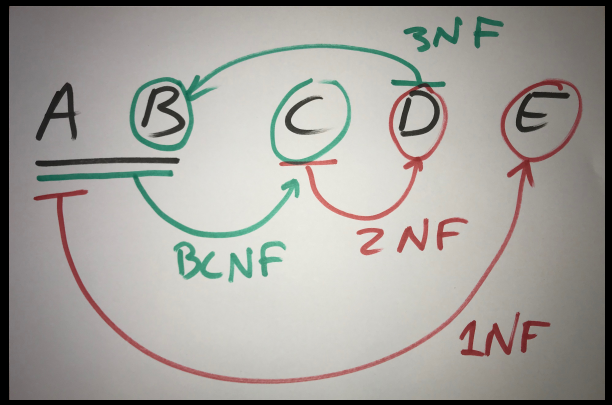
- **Both in BCNF**
- **What happened to AB → D?**



Example: Decomposition

- **R = ABCD**
- **A → C**
- **A → D**
  - A → CD
- **New Table: ACD** **// Merge tables with identical keys**
- **Old Table: AB**
- **Both in BCNF**

X → Y
R1 = R – Y
R2 = XY



Example: Decomposition

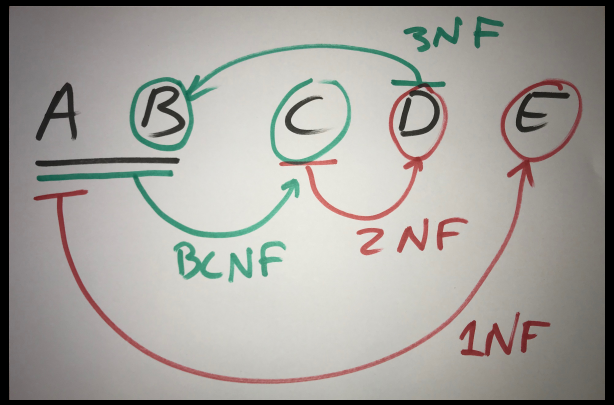
- **R = ABCD**
- **D → A**

$X \rightarrow Y$

R1 = R – Y

R2 = XY

- **R is in 3NF**
- **No decomposition into BCNF**

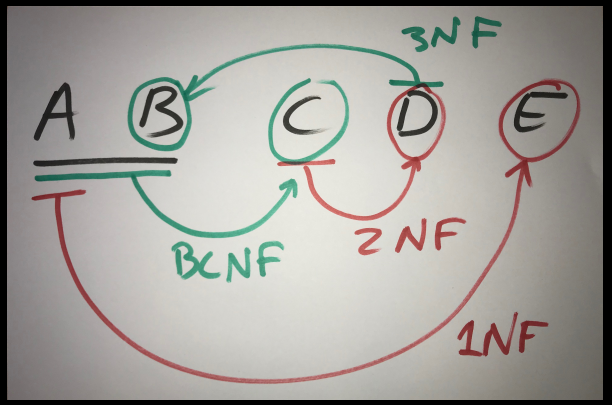


Example: Decomposition

- R = ABCD
- A → C, C → D
- Which one do we use first? … and why?

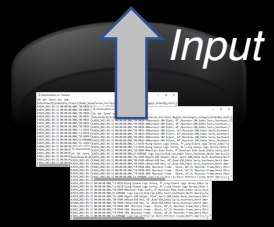
X → Y
R1 = R – Y
R2 = XY

- New Table: CD
- New Table: AC
- Old Table: AB



- All three in BCNF
- What happened to AB → D

RDBMS



Raw Data

How Do We Find Keys?

Finding Keys

- **Is attribute A a key to R?**
  - What does it determine (directly and indirectly)
  - Recursively add “right sides” of FDs to A until done…
  - If A determines R: Yes
- **Are attribute set X a key to R?**
  - What do they together determine (directly and indirectly)
  - Recursively add “right sides” of FDs to X until done…
  - If X determines R: Maybe
  - If no subset of X is a key: Yes
- **Finding all keys of R:**
  In theory: Examine all attribute subsets of R
  In practice: Focus on left sides of FDs, avoid supersets

Finding Keys: Examples

- **Consider the table Person(ID, Name, ZIP, City)**

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

- **FDs: ID → name, ID → ZIP, ZIP → City**
  - Is City key? No → not on left side of any FD
  - Is ID key? Yes → determines Name, ZIP, City
  - Is ZIP key? No → only determines ZIP, City
  - Is (ID, name) key? No → superset of key

Two (or more) Keys and BCNF

- **Consider this Person relation**
  - With two keys!

Person

ID	CPR	Name	ZIP
1	0123456789	Björn	2100
2	9876543210	Johan	2300
3	1122334455	Peter	2100

```sql
CREATE TABLE Person (
    ID INT PRIMARY KEY,
    CPR CHAR(10) NOT NULL UNIQUE,
    Name CHARACTER VARYING NOT NULL,
    ZIP INT NOT NULL REFERENCES ZIP(ZIP)
);
```

Two (or more) Keys and BCNF

- **Consider this Person relation**

Person

ID	CPR	Name	ZIP
1	0123456789	Björn	2100
2	9876543210	Johan	2300
3	1122334455	Peter	2100

- **What are the FDs?**
  - ID ⟶ CPR, ID ⟶ Name, ID ⟶ ZIP
  - CPR ⟶ ID, CPR ⟶ Name, CPR ⟶ ZIP
  - All are: <key> ⟶ <something>
- **The table is in BCNF – there is no redundancy!**

3NF and Keys

- **Prototype: R(<u>A</u>, <u>B</u>, C), C → A**
  - AB is a key
  - CB is also a key!
  - 3NF always has (at least) two keys!

- **Example from Iceland:**
  - Boats(Area, Number, ZIP, …)
  - ZIP → Area
- **We do NOT normalise this!**
  - We DO note that Number, ZIP is also a key (UNIQUE)

Example: Normalization

- **Advices(Teacher, Student, Dept, Program, Course)**
  - Teacher → Dept
  - Course → Program

- **Outcome:**
  - Faculty(Teacher, Dept)
  - Catalog(Course, Program)
  - Advices(Teacher, Student, ~~Dept, Program~~, Course)

How do We Find Functional Dependencies?

Discovering FDs: Inspection Method

- **How can we find FDs?**

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

- **Run a "thought experiments":**
  - Assume that an attribute is a key of a sub-relation
  - Consider which attributes could be in that relation?
- **Study application design requirements:**
  - Ex: Each vendor can only sell one part to each project
    - vendor project → part

Discovering FDs: SQL Method

- **Study relation instances (when available)**
  + Check application design requirements
  + Check reality

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

What is the SQL method?

- **Assume that ZIP → City holds:**
  - For each ZIP value, how many different City values?

```sql
SELECT 'Person: ZIP --> City' AS FD,
    CASE WHEN COUNT(*)=0 THEN 'MAY HOLD'
    ELSE 'does not hold' END AS VALIDITY
FROM (
    SELECT P.ZIP
    FROM Person P
    GROUP BY P.ZIP
    HAVING COUNT(DISTINCT P.City) > 1
) X;
```

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

Discovering FDs: SQL Method

- **Study relation instances (when available)**
  - Can make a script to test all combinations, e.g.:
    - Use Java to write SQL queries into text file
    - Run the text file using psql
- **Remember: Script can only say MAY HOLD!**
  - Consider City → ZIP

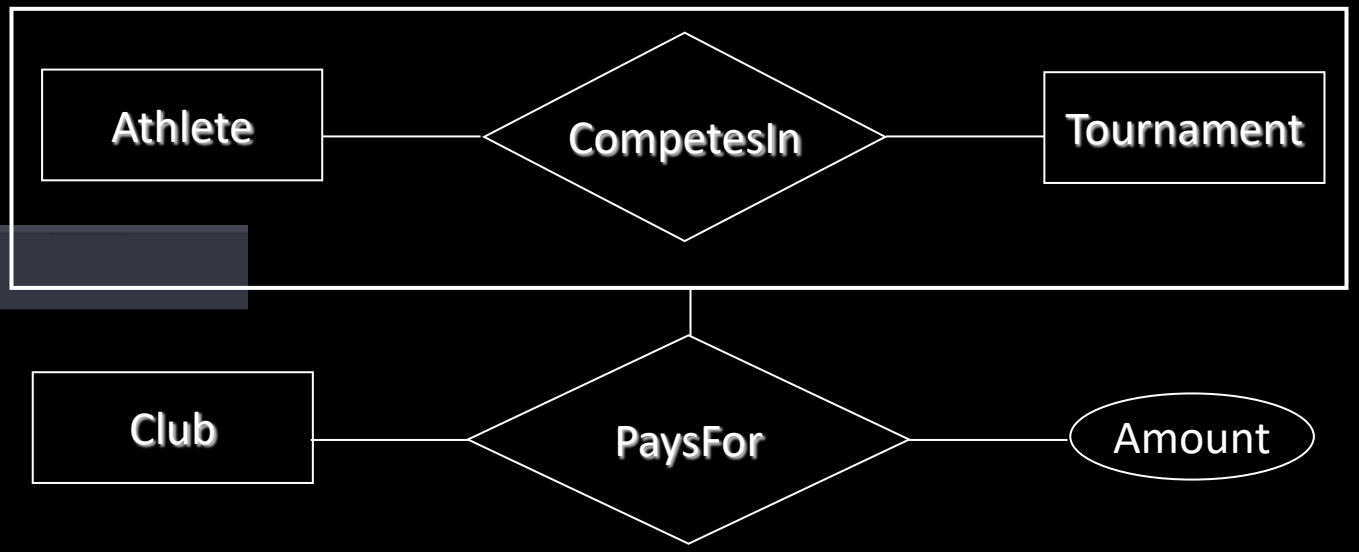
+ Check application design requirements
+ Check reality

Person

ID	Name	ZIP	City
1	Björn	2100	København Ø
2	Johan	2300	København S
3	Peter	2100	København Ø

Another Example Revisited (slides 15-16)

- **Consider the PaysFor relationship from ER slides**




```sql
CREATE TABLE PaysFor (
    AID INTEGER,
    TID INTEGER,
    CID INTEGER REFERENCES Clubs,
    amount INTEGER NOT NULL,
    FOREIGN KEY (AID, TID) REFERENCES CompetesIn (AID, TID),
    PRIMARY KEY (AID, TID, CID)
);
```

Another Example: Add Data

- **Perhaps AID and CID are related?**
  - What if AID ⟶ CID – athlete belongs to one club!
  - Means that the key is in fact only AID, TID

```sql
CREATE TABLE PaysFor (
  AID INTEGER,
  TID INTEGER,
  CID INTEGER REFERENCES Clubs,
  amount INTEGER NOT NULL,
  FOREIGN KEY (AID, TID) REFERENCES CompetesIn (AID, TID),
  PRIMARY KEY (AID, TID, CID)
);
```

PaysFor

AID	TID	CID	amount
1	1	1	1000
1	2	1	5000
1	3	1	600
2	1	1	1000
3	2	2	1000

- **This table is in 1NF**
  - AID, TID ⟶ Amount
  - AID ⟶ CID

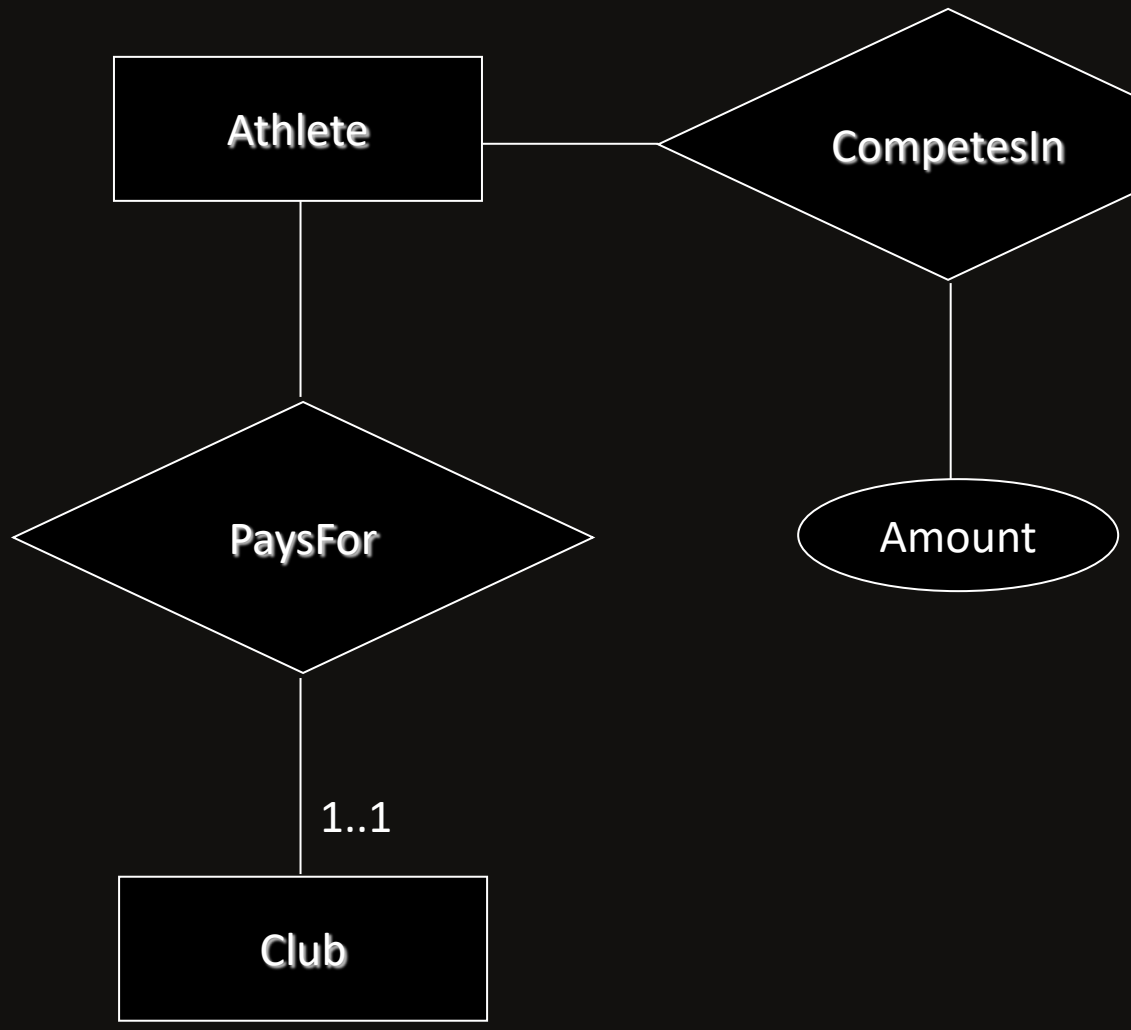
Another Example: Correct Tables

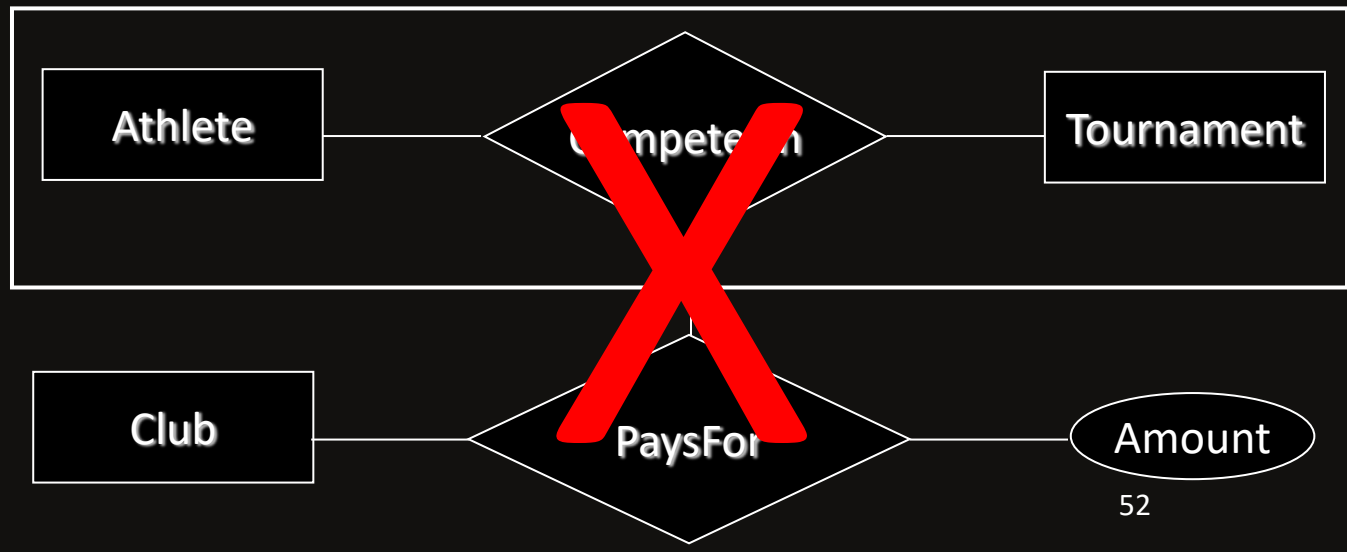
```sql
CREATE TABLE AthleteClub (
    AID INTEGER PRIMARY KEY,
    CID INTEGER NOT NULL REFERENCES Clubs
);

CREATE TABLE PaysFor (
    AID INTEGER,
    TID INTEGER,
    CID INTEGER REFERENCES Clubs,
    amount INTEGER NOT NULL,
    FOREIGN KEY (AID, TID) REFERENCES CompetesIn (AID, TID),
    PRIMARY KEY (AID, TID, CID)
);
```

Another Example: Really Correct Tables

```sql
CREATE TABLE Athlete (
    AID INTEGER PRIMARY KEY,
    ... -- Other columns here
    CID INTEGER NOT NULL REFERENCES Clubs
);

CREATE TABLE CompetesIn (
    AID INTEGER REFERENCES Athlete,
    TID INTEGER REFERENCES Tournament,
    amount INTEGER NULL,
    PRIMARY KEY (AID, TID)
);
```

Another Example: Corrected ER Diagram





Takeaways

- **Functional Dependencies**
  - With flash-back to keys
  - Unavoidable, trivial, and redundant FDs
  - How to detect potential FDs?
- **Normal forms**
  - Focus on BCNF and 3NF
  - Always decompose 1NF and 2NF!
  - Never decompose 3NF!
- **Normalization process**
  - Decomposition of relations to BCNF (or 3NF)

Note

Contrary to popular belief, practical normalisation is not hard!

What is next?

- **Next Lecture:**
  - Storage Hierarchy
  - Multicore Processing
  - OS
- **Lecturer:**
  - Pınar Tözün