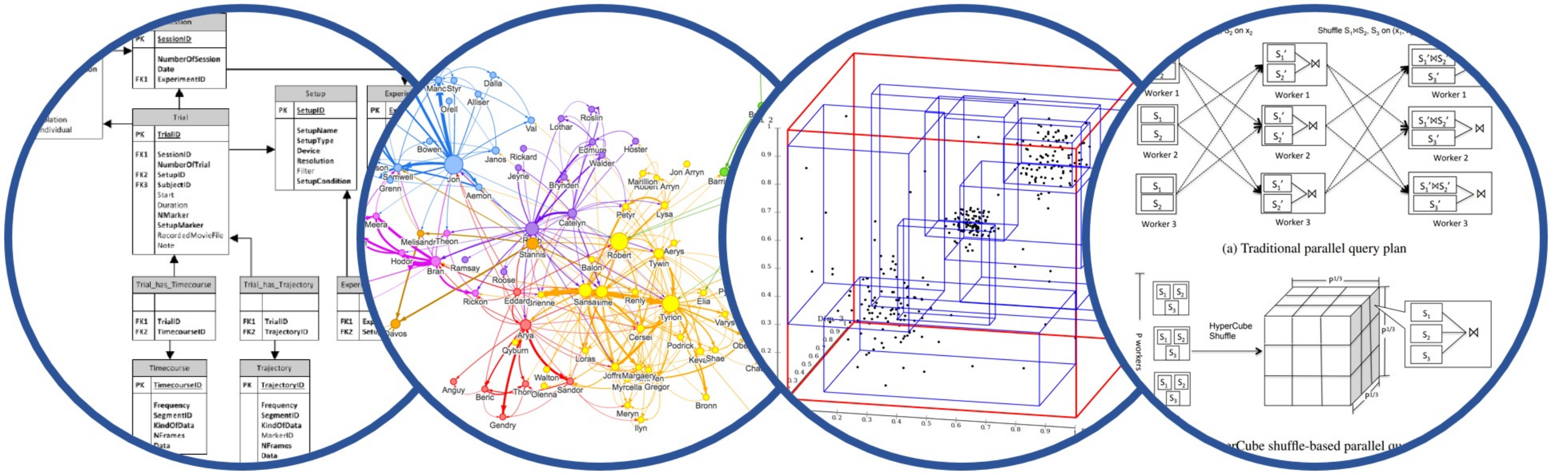


Introduction to Data Management

Functional Dependencies

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW3 due tonight
- HW4 posted, due on Friday, May 3rd

Announcements

Midterm:

- Next Friday, in class, closed books, no cheat sheet
- Some practice midterms on the course website

- Midterm has four parts:
  - SQL
  - Relational Algebra
  - Entity-Relationship Diagrams (ER)
  - Functional Dependencies

longest

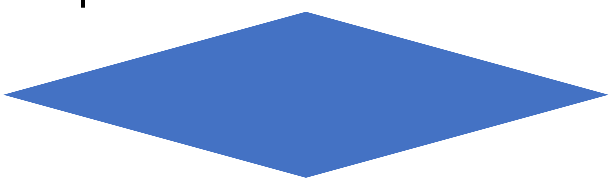
shortest

Recap: ER Diagrams

Entity set

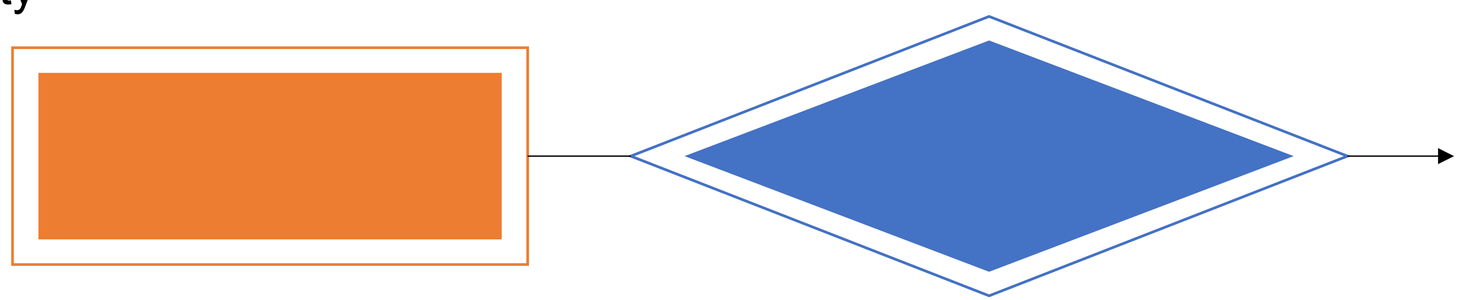
Attribute

Relationship



Subclass

Weak Entity



Agenda

Today:

- Database Constraints (finish)
- Anomalies and Functional Dependencies

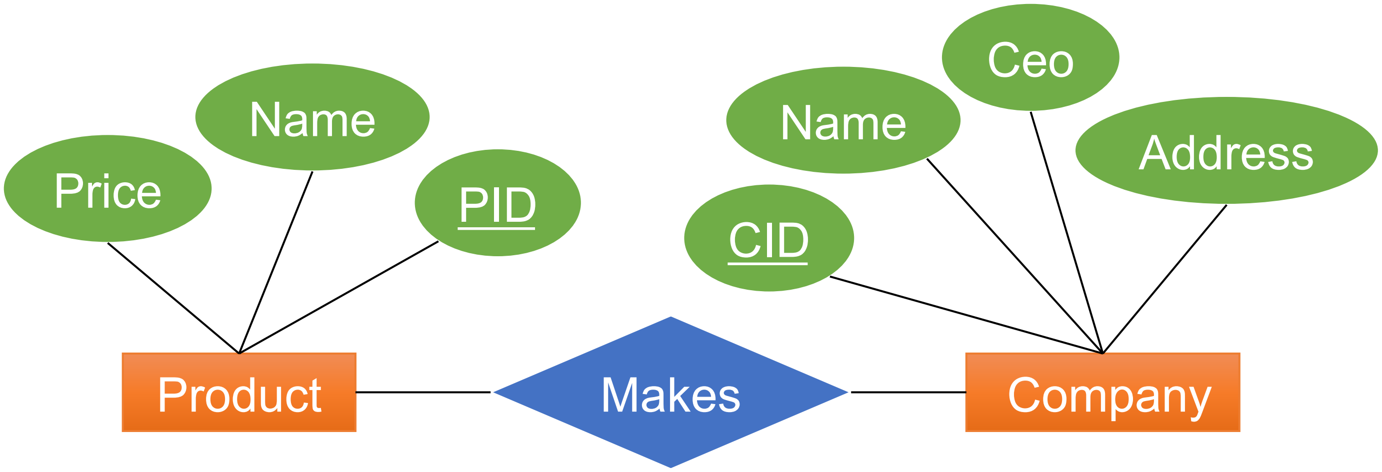
Next lecture:

- Schema Normalization

Database Constraints

- A **constraint** is an assertion that must always hold on the data
- Defining constraints is part of conceptual design
- SQL supports several constraints:
  - Keys and Foreign Keys
  - Attribute-level constraints
  - Tuple-level constraints
  - General assertions

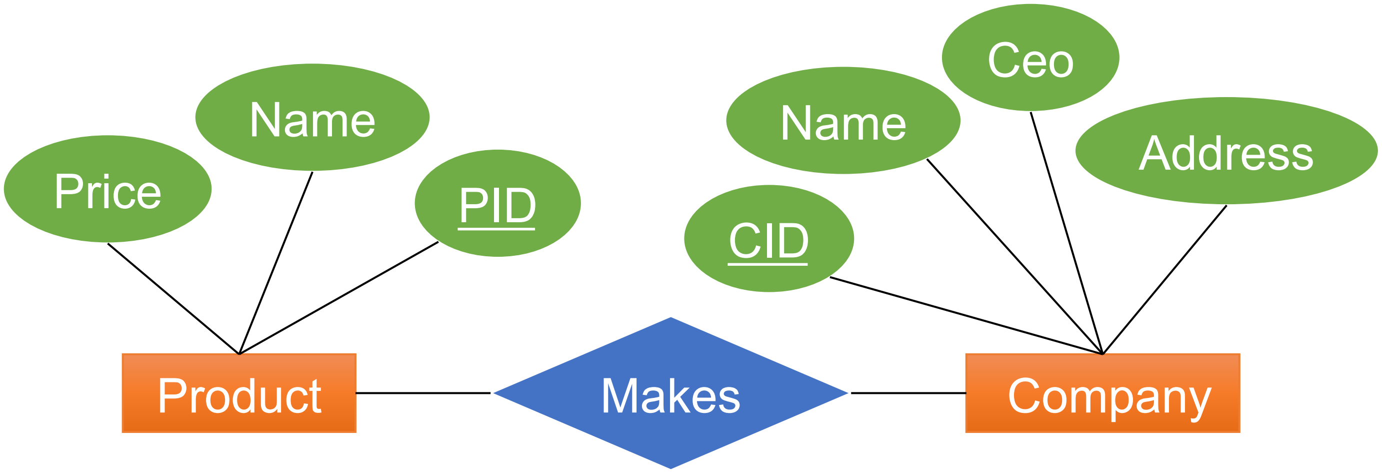
Keys and Foreign Keys




```
CREATE TABLE
  Product (
    PID INT PRIMARY KEY,
    name TEXT,
    Price int);
```

```
CREATE TABLE
  Makes (
    PID INT References Product,
    CID INT References Company);
```

Keys and Foreign Keys



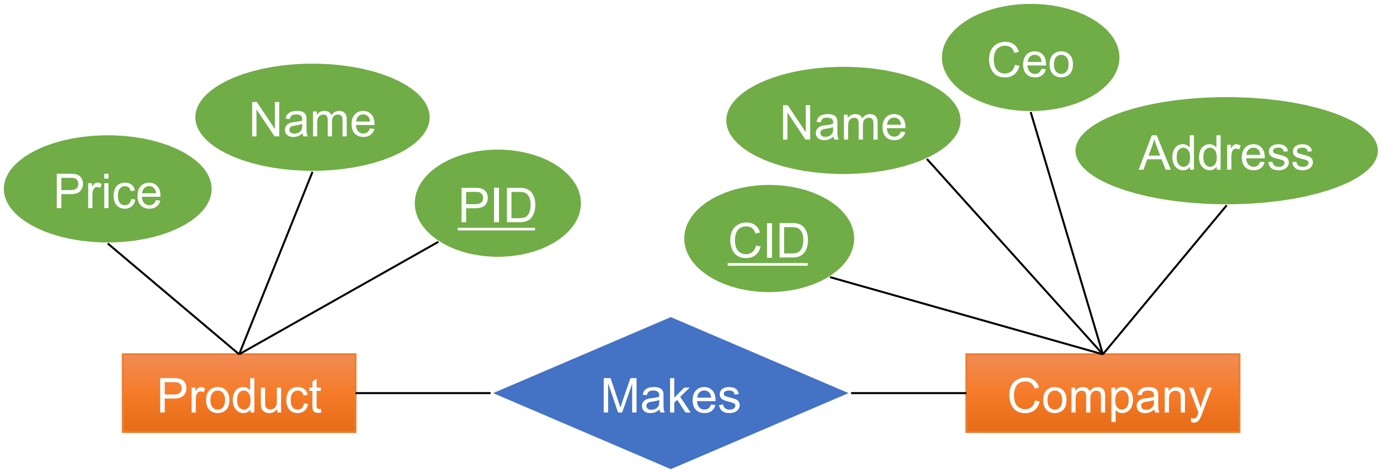

```
CREATE TABLE
  Product (
    PID INT PRIMARY KEY,
    name TEXT,
    Price int);
```

```
CREATE TABLE
  Makes (
    PID INT References Product,
    CID INT References Company);
```

What does system check when…

- …we insert a Product?
- …we delete a Product?

Keys and Foreign Keys




```
CREATE TABLE
  Product (
    PID INT PRIMARY KEY,
    name TEXT,
    Price int);
```

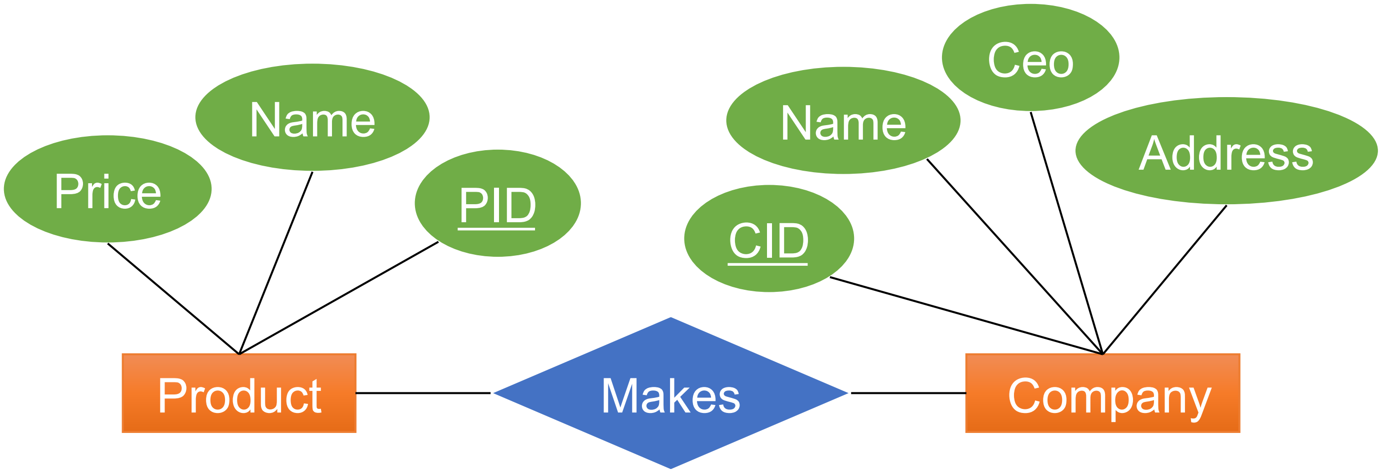
```
CREATE TABLE
  Makes (
    PID INT References Product,
    CID INT References Company);
```

Check PID doesn't exist

What does system check when…

- …we insert a Product?
- …we delete a Product?

Keys and Foreign Keys




```
CREATE TABLE
  Product (
    PID INT PRIMARY KEY,
    name TEXT,
    Price int);
```

```
CREATE TABLE
  Makes (
    PID INT References Product,
    CID INT References Company);
```

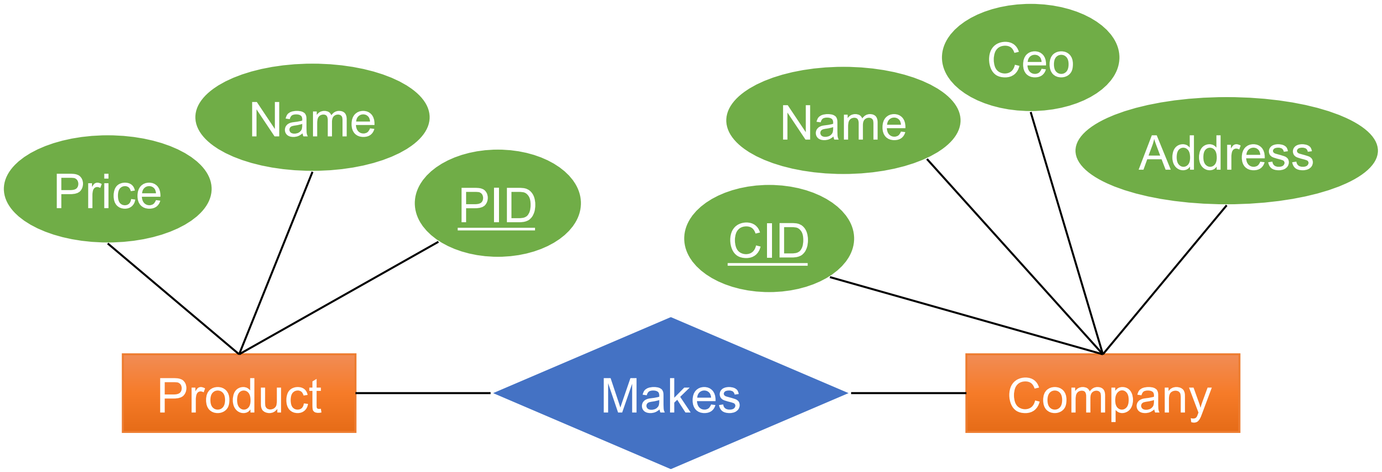
Check PID doesn't exist

What does system check when…

- …we insert a Product?
- …we delete a Product?

Check no Makes has that PID

Keys and Foreign Keys




```
CREATE TABLE
  Product (
    PID INT PRIMARY KEY,
    name TEXT,
    Price int);
```

```
CREATE TABLE
  Makes (
    PID INT References Product,
    CID INT References Company);
```

Check PID doesn't exist

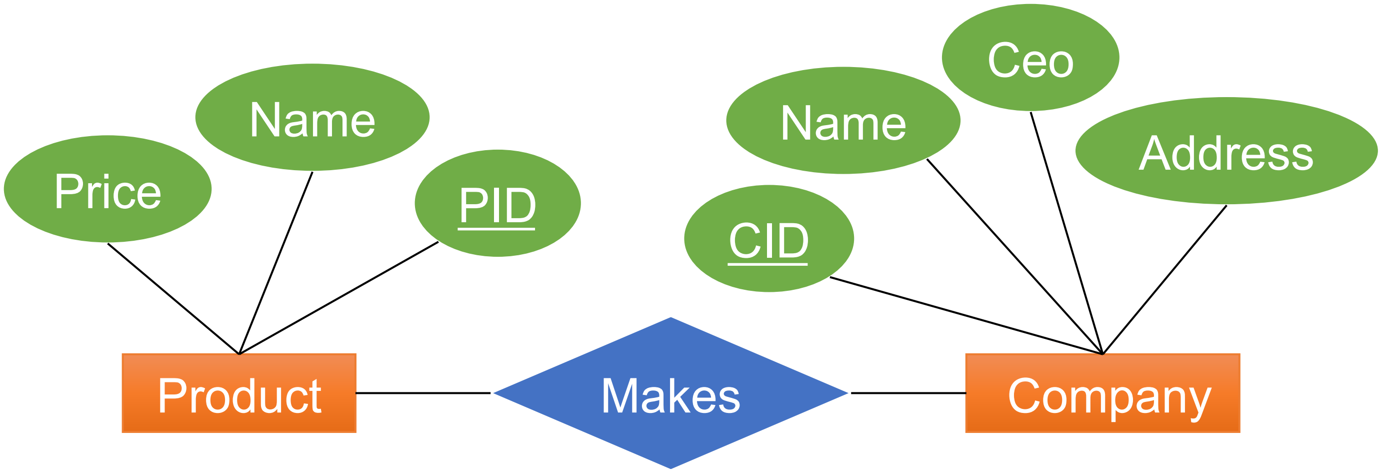
What does system check when…

- …we insert a Product?
- …we delete a Product?

- …we insert a Makes tuple?
- …we delete a Makes tuple?

Check no Makes has that PID

Keys and Foreign Keys




```
CREATE TABLE
  Product (
    PID INT PRIMARY KEY,
    name TEXT,
    Price int);
```

```
CREATE TABLE
  Makes (
    PID INT References Product,
    CID INT References Company);
```

Check PID doesn't exist

Check PID,CID exist

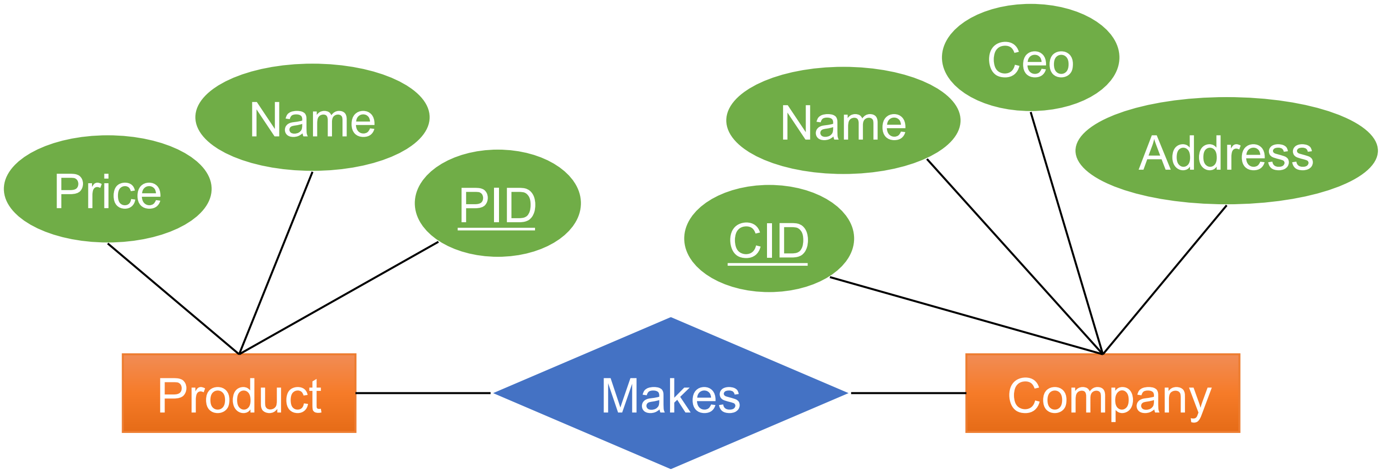
What does system check when…

- …we insert a Product?
- …we delete a Product?

- …we insert a Makes tuple?
- …we delete a Makes tuple?

Check no Makes has that PID

Keys and Foreign Keys




```
CREATE TABLE
  Product (
    PID INT PRIMARY KEY,
    name TEXT,
    Price int);
```

```
CREATE TABLE
  Makes (
    PID INT References Product,
    CID INT References Company);
```

What does system check when…

- …we insert a Product? — Check PID doesn't exist
- …we delete a Product? — Check no Makes has that PID
- …we insert a Makes tuple? — Check PID,CID exist
- …we delete a Makes tuple? — Nothing

Keys and Foreign Keys

```
CREATE TABLE Product (pid INT PRIMARY KEY, ...);
```

OK

```
CREATE TABLE Makes(fk_pid INT References Product,…);
```

OK

```
CREATE TABLE Makes(fk_pid INT References Product(PID),…);
```

Error

```
CREATE TABLE Makes(fk_pid INT References Product(price),…)
```

Attribute- and Tuple-level Constraints

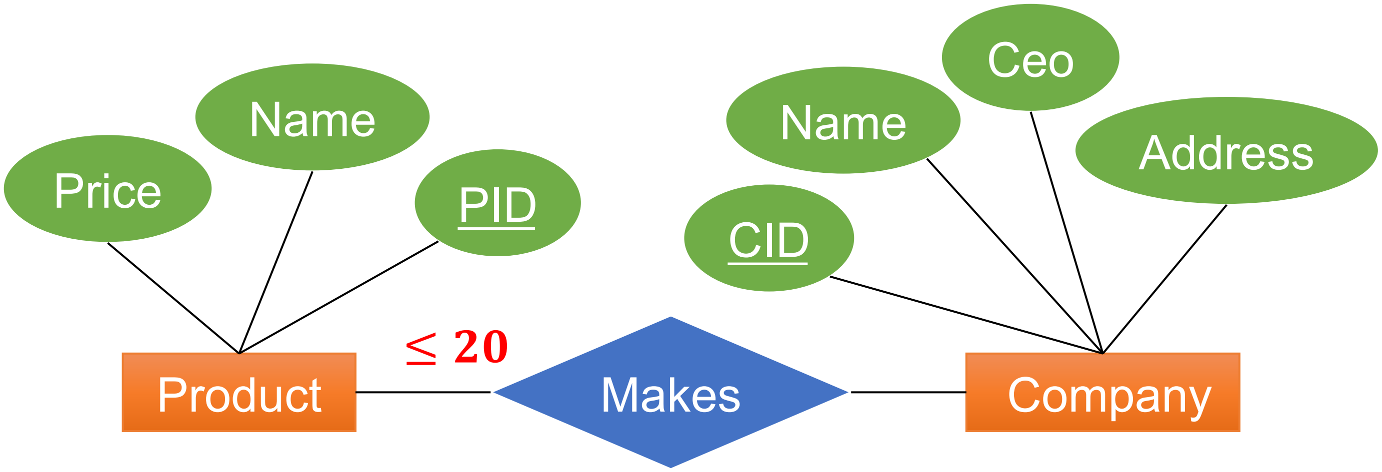
```
CREATE TABLE User (
    uid INT PRIMARY KEY,
    name TEXT,
    age INT CHECK (age > 12 AND age < 120),
    email TEXT,
    phone TEXT,

    CHECK (email IS NOT NULL OR phone IS NOT NULL)
);
```

Attribute constraint

Tuple-level constraint

What happens when we insert a User?

Global Assertions




```
CREATE ASSERTION myAssert CHECK
    (NOT EXISTS (
      SELECT Makes.PID
      FROM Makes
      GROUP BY Make.PID
      HAVING COUNT(*) > 20)
     );
```

Expensive.

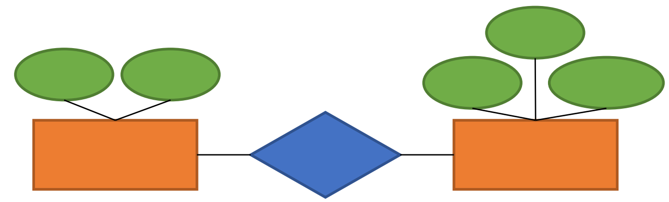
Very few systems support it

Database Normalization

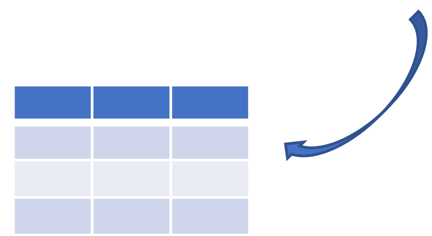
The Database Design Process

Done

Conceptual Model

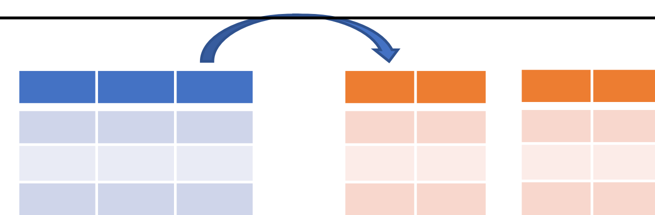


Relational Model
+ Schema
+ Constraints



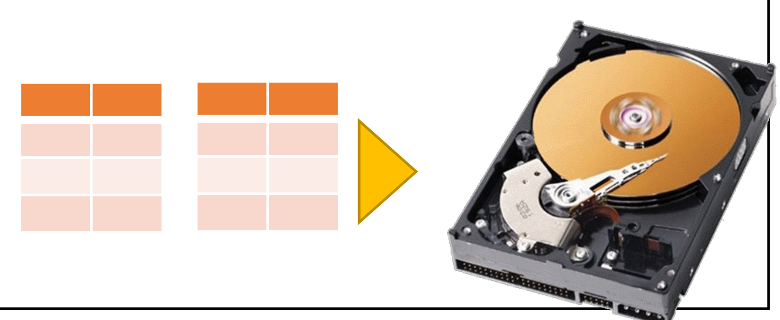
Starting today

Conceptual Schema
+ Normalization



Later…

Physical Schema
+ Partitioning
+ Indexing



Outline

- A poorly designed table may exhibit **anomalies**
- **Database normalization:** remove them by splitting the table
- **Functional Dependencies** (FD): mathematical tool for database normalization

Example

Simple directory of people, their phone number, and their city

UID	Name	Phone	City
234	Fred	206-555-9999	Seattle
234	Fred	206-555-8888	Seattle
987	Joe	415-555-7777	SF

Notice that UID is not a key – **why?**

Example

Simple directory of people, their phone number, and their city

UID	Name	Phone	City
234	Fred	206-555-9999	Seattle
234	Fred	206-555-8888	Seattle
987	Joe	415-555-7777	SF

Notice that UID is not a key – **why?**

Anomalies:

- **Redundancy anomaly**: Fred, Seattle repeated

Example

Simple directory of people, their phone number, and their city

UID	Name	Phone	City
234	Fred	206-555-9999	Seattle
234	Fred	206-555-8888	Seattle
987	Joe	415-555-7777	SF

Notice that UID is not a key – **why?**

Anomalies:

- **Redundancy anomaly**: Fred, Seattle repeated
- **Update anomaly**: Fred to Portland needs multiple updates

Example

Simple directory of people, their phone number, and their city

UID	Name	Phone	City
234	Fred	206-555-9999	Seattle
234	Fred	206-555-8888	Seattle
987	Joe	415-555-7777	SF

Notice that UID is not a key – **why?**

Anomalies:

- **Redundancy anomaly**: Fred, Seattle repeated
- **Update anomaly**: Fred to Portland needs multiple updates
- **Deletion anomaly**: deleting Joe’s phone number loses Joe

Example

Simple directory of people, their phone number, and their city

UID	Name	Phone	City
234	Fred	206-555-9999	Seattle
234	Fred	206-555-8888	Seattle
987	Joe	415-555-7777	SF

How do we remove anomalies?

Example

Simple directory of people, their phone number, and their city

UID	Name	Phone	City
234	Fred	206-555-9999	Seattle
234	Fred	206-555-8888	Seattle
987	Joe	415-555-7777	SF

How do we remove anomalies?

UID	Name	City
234	Fred	Seattle
987	Joe	SF

UID	Phone
234	206-555-9999
234	206-555-8888
987	415-555-7777

Example

Simple directory of people, their phone number, and their city

UID	Name	Phone	City
234	Fred	206-555-9999	Seattle
234	Fred	206-555-8888	Seattle
987	Joe	415-555-7777	SF

How do we remove anomalies?

UID	Name	City
234	Fred	Seattle
987	Joe	SF

UID	Phone
234	206-555-9999
234	206-555-8888
987	415-555-7777

No more anomalies (In class)

Discussion

- We need a systematic way to reason about, detect, and remove anomalies
- Main theoretical tool: **Functional Dependencies**

Functional Dependencies

Overview

Fix a relation $\mathbf{R(A_1, A_2, \ldots, A_n)}$:

- A Functional Dependency asserts that some attributes uniquely determine other attributes

Overview

Fix a relation $\mathbf{R(A_1, A_2, \ldots, A_n)}$:

- A Functional Dependency asserts that some attributes uniquely determine other attributes

Directory(UID, Name, Phone, City)

- UID uniquely determines Name, City (not Phone)
- We write: **UID → Name, City**

Definition: Informal

A functional dependency is an assertion:

$$A_1, A_2, \ldots \rightarrow B_1, B_2, \ldots$$

Definition: Informal

A functional dependency is an assertion:

$$A_1, A_2, \ldots \rightarrow B_1, B_2, \ldots$$

It says:

If two tuples have same values for attributes $A_1, A_2, \ldots$, then they have the same values for attributes $B_1, B_2, \ldots$

We say that $A_1, A_2 \ldots$ **determine** $B_1, B_2 \ldots$

Definition: Informal

A functional dependency is an assertion:

$$A_1, A_2, \ldots \rightarrow B_1, B_2, \ldots$$

antecedent

consequent

It says:

If two tuples have same values for attributes $A_1, A_2, \ldots$, then they have the same values for attributes $B_1, B_2, \ldots$

We say that $A_1, A_2 \ldots$ **determine** $B_1, B_2 \ldots$

Example

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Example

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Examples:

EID → Name, Email, Dept
Dept → Email

Example

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Examples:

EID → Name, Email, Dept
Dept → Email

Non-Examples:

Name → Dept
Email → Dept

Example

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Examples:

EID → Name, Email, Dept
Dept → Email

Non-Examples:

Name → Dept
Email → Dept

Maybe Examples:

Name, Email → Dept

Example

If two tuples have the same values of $A_1 A_2$ …, then they have the same values of $B_1 B_2$ …

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Examples:

EID → Name, Email, Dept
Dept → Email

Non-Examples:

Name → Dept
Email → Dept

Maybe Examples:

Name, Email → Dept

Example

If two tuples have the same values of $A_1 A_2$ …, then they have the same values of $B_1 B_2$ …

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Examples:

EID → Name, Email, Dept
Dept → Email

Non-Examples:

Name → Dept
Email → Dept

Maybe Examples:

Name, Email → Dept

Example

If two tuples have the same values of $A_1 A_2 \ldots$, then they have the same values of $B_1 B_2 \ldots$

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Examples:

EID → Name, Email, Dept
Dept → Email

Non-Examples:

Name → Dept
Email → Dept

Maybe Examples:

Name, Email → Dept

Example

If two tuples have the same values of $A_1 A_2$ …, then they have the same values of $B_1 B_2$ …

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Name,Email happen to have unique values

Examples:

EID → Name, Email, Dept
Dept → Email

Non-Examples:

Name → Dept
Email → Dept

Maybe Examples:

Name, Email → Dept

Example

If two tuples have the same values of $A_1 A_2$ …, then they have the same values of $B_1 B_2$ …

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer
0999	Alice	clr@abc.com	Clerk 2

Name,Email happen to have unique values

No more

Examples:

EID → Name, Email, Dept
Dept → Email

Non-Examples:

Name → Dept
Email → Dept

Maybe Examples:

Name, Email → Dept

Discussion

Two ways to interpret an FD A→B:

- Given a concrete instance R(A,B,…) we can **check** whether A→B holds or not.
- We **assert** that A→B shall hold on R, and will reject updates that violate this FD

Example

Which of these FDs hold?

Name → Color
Category → Dept
Color, Dept → Price

Name	Category	Color	Dept	Price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Example

Which of these FDs hold?

Name → Color
Category → Dept
Color, Dept → Price

yes

Name	Category	Color	Dept	Price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Example

Which of these FDs hold?

Name → Color yes
Category → Dept yes
Color, Dept → Price

Name	Category	Color	Dept	Price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Example

Which of these FDs hold?

Name → Color	yes
Category → Dept	yes
Color, Dept → Price	no

Name	Category	Color	Dept	Price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	**99**

Example

Which of these FDs hold?

Name → Color	yes
Category → Dept	no
Color, Dept → Price	no

Name	Category	Color	Dept	Price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99
Grill	Gadget	Black	Kitchen	199

Example

Which of these FDs hold?

Name → Color no
Category → Dept no
Color, Dept → Price no

Name	Category	Color	Dept	Price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99
Grill	Gadget	Black	Kitchen	199
Grill	Gadget	Brown	Kitchen	199

The more tuples we add, the fewer FDs hold

Checking an FD in SQL

Employees

EID	Name	Email	Dept
0345	Alice	clr@abc.com	Clerk 1
0456	Bob	clr@abc.com	Clerk 2
0567	Alice	sales@abc.com	Sales rep
0678	Carol	sales@abc.com	Sales rep
0789	David	law@abc.com	Lawyer

Check this in SQL

Name → Dept

```
SELECT *
FROM Employees E1, Employees E2
WHERE E1.Name = E2.Name
  and E1.Dept != E2.Dept;
```

We will improve this query in class

Inference

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
a	b	c	d	e	…
a	b	?	?	?	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
a	b	c	d	e	…
a	b	?	?	?	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
a	b	c	d	e	…
a	b	c	?	?	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
a	b	c	d	e	…
a	b	c	?	?	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
a	b	c	d	e	…
a	b	c	d	?	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
a	b	c	d	e	…
a	b	c	d	?	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Proof: tuples with same Name, Category must have same Price

Name	Category	Color	Dept	Price	…
…	…	…	…	…	…
a	b	c	d	e	…
a	b	c	d	e	…
…	…	…	…	…	…
…	…	…	…	…	…

An Interesting Observation

If all these FDs are true:

Name → Color
Category → Dept
Color, Dept → Price

Then this FD is also true:

Name, Category → Price

Yes

Proof: tuples with same Name, Category must have same Price

Name	**Category**	**Color**	**Dept**	**Price**	**…**
…	…	…	…	…	…
a	b	c	d	e	…
a	b	c	d	e	…
…	…	…	…	…	…
…	…	…	…	…	…

Discussion

Two ways to infer new FDs:

- Armstrong axioms
- The closure operator

Armstrong’s Axioms

Armstrong's Axioms

Reflexivity: if $Y \subseteq X$ then $X \rightarrow Y$

Augmentation: if $X \rightarrow Y$ then $XZ \rightarrow YZ$

Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Using Armstrong's Axioms

Reflexivity: if $Y \subseteq X$ then $X \rightarrow Y$
Augmentation: if $X \rightarrow Y$ then $XZ \rightarrow YZ$
Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

1. Name → Color
2. Category → Dept
3. Color, Dept → Price

⇒

Name, Category → Price

Using Armstrong's Axioms

Reflexivity: if $Y \subseteq X$ then $X \rightarrow Y$
Augmentation: if $X \rightarrow Y$ then $XZ \rightarrow YZ$
Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

1. Name → Color
2. Category → Dept
3. Color, Dept → Price

⇒ Name, Category → Price

4. Name, Category → Color, Category (Augmentation of 1)

Using Armstrong's Axioms

Reflexivity: if $Y \subseteq X$ then $X \rightarrow Y$
Augmentation: if $X \rightarrow Y$ then $XZ \rightarrow YZ$
Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

1. Name → Color
2. Category → Dept
3. Color, Dept → Price

⇒ Name, Category → Price

4. Name, Category → Color, Category (Augmentation of 1)
5. Color, Category → Color, Dept (Augmentation of 2)

Using Armstrong's Axioms

Reflexivity: if $Y \subseteq X$ then $X \to Y$
Augmentation: if $X \to Y$ then $XZ \to YZ$
Transitivity: if $X \to Y$ and $Y \to Z$ then $X \to Z$

1. Name → Color
2. Category → Dept
3. Color, Dept → Price

⇨ Name, Category → Price

4. Name, Category → Color, Category (Augmentation of 1)
5. Color, Category → Color, Dept (Augmentation of 2)
6. Color, Category → Price (Transitivity 5 and 3)

Using Armstrong's Axioms

Reflexivity: if $Y \subseteq X$ then $X \rightarrow Y$
Augmentation: if $X \rightarrow Y$ then $XZ \rightarrow YZ$
Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

1. Name → Color
2. Category → Dept
3. Color, Dept → Price

⇒ Name, Category → Price

4. Name, Category → Color, Category (Augmentation of 1)
5. Color, Category → Color, Dept (Augmentation of 2)
6. Color, Category → Price (Transitivity 5 and 3)
7. Name, Category → Price (Transitivity 4 and 6)

Discussion

- Armstrong's Axioms were introduced in the 70s, shortly after Codd's relational model
- They are widely known today
- But they are cumbersome to use for inference
- Instead, the efficient inference method uses the **closure operator**: next.

The Closure Operator

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a
set of attributes X
is the set of attributes A
such that $X \to A$.

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \to A$.

```
Closure(X):
  Repeat:
      find a FD Y → A
      such that Y ⊆ X and A ⊄ X
      X ≔ X ∪ {A}
  Until "no more change"
```

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \rightarrow A$.

```
Closure(X):
  Repeat:
      find a FD Y → A
      such that Y ⊆ X and A ⊈ X
      X := X ∪ {A}
  Until "no more change"
```

Name → Color Category → Dept Color, Dept → Price

$\{\text{Name}, \text{Category}\}^+ =$

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \to A$.

```
Closure(X):
  Repeat:
     find a FD Y → A
     such that Y ⊆ X and A ⊄ X
     X := X ∪ {A}
  Until "no more change"
```

Name → Color
Category → Dept
Color, Dept → Price

$\{Name, Category\}^+ =$

$= \{Name, Category, \qquad \}$

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \rightarrow A$.

```
Closure(X):
  Repeat:
      find a FD Y → A
      such that Y ⊆ X and A ⊄ X
      X := X ∪ {A}
  Until "no more change"
```

Name → Color Category → Dept Color, Dept → Price

$\{Name, Category\}^+ =$

$= \{Name, Category, Color, \quad \}$

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \rightarrow A$.

```
Closure(X):
  Repeat:
     find a FD Y → A
     such that Y ⊆ X and A ⊄ X
     X := X ∪ {A}
  Until "no more change"
```

Name → Color
Category → Dept
Color, Dept → Price

$\{Name, Category\}^+ =$
$= \{Name, Category, Color, Dept, \quad \}$

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \to A$.

```
Closure(X):
  Repeat:
      find a FD Y → A
      such that Y ⊆ X and A ⊄ X
      X := X ∪ {A}
  Until "no more change"
```

Name → Color
Category → Dept
Color, Dept → Price

$\{\text{Name}, \text{Category}\}^+ =$

$= \{\text{Name}, \text{Category}, \text{Color}, \text{Dept}, \text{Price}\}$

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \rightarrow A$.

```
Closure(X):
  Repeat:
     find a FD Y → A
     such that Y ⊆ X and A ⊄ X
     X := X ∪ {A}
  Until "no more change"
```

Name → Color Category → Dept Color, Dept → Price

$\{\text{Name}, \text{Category}\}^+ =$
$= \{\text{Name}, \text{Category}, \text{Color}, \text{Dept}, \text{Price}\}$

$\{\text{Color}\}^+ =$

The Closure of a set X

Fix a set of Functional Dependencies

The closure X^+ of a set of attributes X is the set of attributes A such that $X \to A$.

```
Closure(X):
   Repeat:
      find a FD Y → A
      such that Y ⊆ X and A ⊄ X
      X := X ∪ {A}
   Until "no more change"
```

Name → Color
Category → Dept
Color, Dept → Price

$\{\text{Name, Category}\}^+ =$
$= \{\text{Name, Category, Color, Dept, Price}\}$

$\{\text{Color}\}^+ = \{\text{Color}\}$

Discussion so Far

- Goal is to detect/remove anomalies

- Anomalies are caused by unwanted FDs
  E.g. UID → Name, City; but UID not a key

- Next lecture: use FDs to decompose table
  Database Normalization