Introduction to Data Management
CSE 414

Unit 6: Conceptual Design
E/R Diagrams
Integrity Constraints
BCNF

(3 lectures)

Introduction to Data Management
CSE 414

Integrity Constraints

Integrity Constraints Motivation

An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

- ICs help prevent entry of incorrect information
- How? DBMS enforces integrity constraints
  - Allows only legal database instances (i.e., those that satisfy all constraints) to exist
  - Ensures that all necessary checks are always performed and avoids duplicating the verification logic in each application

Constraints in E/R Diagrams

Finding constraints is part of the modeling process.
Commonly used constraints:

Keys: social security number uniquely identifies a person.

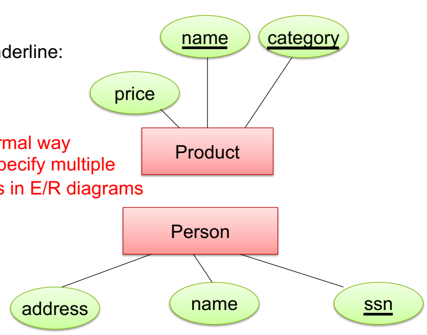
Single-value constraints: a person can have only one father.

Referential integrity constraints: if you work for a company, it must exist in the database.

Other constraints: peoples' ages are between 0 and 150.

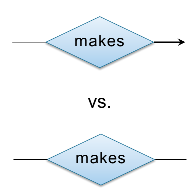
Keys in E/R Diagrams

Underline:

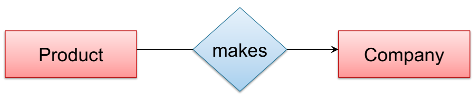


No formal way to specify multiple keys in E/R diagrams

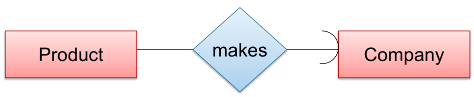
Single Value Constraints



Referential Integrity Constraints

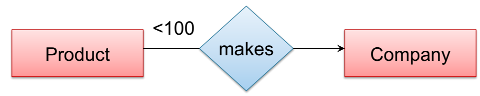


Each product made by at most one company.
Some products made by no company



Each product made by ***exactly*** one company.

Other Constraints



Q: What does this mean ?
A: A Company entity cannot be connected
by relationship to more than 99 Product entities

Constraints in SQL

Constraints in SQL:

- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

simplest

Most complex

- The more complex the constraint, the harder it is to check and to enforce

Key Constraints

Product(<u>name</u>, category)

```
CREATE TABLE Product (
        name CHAR(30) PRIMARY KEY,
        category VARCHAR(20))
```

OR:

```
CREATE TABLE Product (
        name CHAR(30),
        category VARCHAR(20),
PRIMARY KEY (name))
```


Keys with Multiple Attributes

Product(<u>name, category</u>, price)

```
CREATE TABLE Product (
        name CHAR(30),
        category VARCHAR(20),
        price INT,
   PRIMARY KEY (name, category))
```

Name	Category	Price
Gizmo	Gadget	10
Camera	Photo	20
Gizmo	Photo	30
~~Gizmo~~	~~Gadget~~	~~40~~

Other Keys

```
CREATE TABLE Product (
        productID  CHAR(10),
        name CHAR(30),
        category VARCHAR(20),
        price INT,
        PRIMARY KEY (productID),
        UNIQUE (name, category))
```

There is at most one PRIMARY KEY;
there can be many UNIQUE

Foreign Key Constraints

```
CREATE TABLE Purchase (
        prodName CHAR(30)
        REFERENCES Product(name),
        date DATETIME)
```

Referential integrity constraints

May write just Product if name is PK

prodName is a **foreign key** to Product(name)
name must be a **key** in Product

Foreign Key Constraints

- Example with multi-attribute primary key

```
CREATE TABLE Purchase (
        prodName CHAR(30),
        category VARCHAR(20),
        date DATETIME,
        FOREIGN KEY (prodName, category)
           REFERENCES  Product(name, category)
```

- (name, category) must be a KEY in Product

What happens when data changes?

Types of updates:
- In Purchase: insert/update
- In Product: delete/update

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

What happens when data changes?

- SQL has three policies for maintaining referential integrity:
- NO ACTION reject violating modifications (default)
- CASCADE after delete/update do delete/update
- SET NULL set foreign-key field to NULL
- SET DEFAULT set foreign-key field to default value
  - need to be declared with column, e.g., `CREATE TABLE Product (pid INT DEFAULT 42)`

Maintaining Referential Integrity

```
CREATE TABLE Purchase (
        prodName CHAR(30),
        category VARCHAR(20),
        date DATETIME,
        FOREIGN KEY (prodName, category)
           REFERENCES  Product(name, category)
           ON UPDATE CASCADE
           ON DELETE SET NULL   )
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Category
Gizmo	Gizmo
Snap	Camera
EasyShoot	Camera

Constraints on Attributes and Tuples

- Constraints on attributes:
  - NOT NULL -- obvious meaning...
  - CHECK condition -- any condition !
- Constraints on tuples
  - CHECK condition

Constraints on Attributes and Tuples

```
CREATE TABLE R (
        A int NOT NULL,
        B int CHECK (B > 50 and B < 100),
        C varchar(20),
        D int,
        CHECK (C >= 'd' or D > 0))
```


Constraints on Attributes and Tuples

```
CREATE TABLE Product (
        productID  CHAR(10),
        name CHAR(30),
        category VARCHAR(20),
        price INT CHECK (price > 0),
        PRIMARY KEY (productID),
        UNIQUE (name, category))
```


Constraints on Attributes and Tuples

What does this constraint do?

What is the difference from Foreign-Key ?

```
CREATE TABLE Purchase (
        prodName CHAR(30)
            CHECK (prodName IN
                    (SELECT Product.name
                     FROM Product),
        date DATETIME NOT NULL)
```


General Assertions

```
CREATE ASSERTION myAssert CHECK
 (NOT EXISTS(
        SELECT Product.name
        FROM Product, Purchase
        WHERE Product.name = Purchase.prodName
        GROUP BY Product.name
        HAVING count(*) > 200) )
```

But most DBMSs do not implement assertions
Because it is hard to support them efficiently
Instead, they provide triggers

Introduction to Data Management CSE 414

Design Theory and BCNF

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

One person may have multiple phones, but lives in only one city

Primary key is thus (SSN, PhoneNumber)

What is the problem with this schema?

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to "Bellevue"?
- Deletion anomalies = what if Joe deletes his phone number?

Relation Decomposition

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

- No more repeated data
- Easy to move Fred to "Bellevue" (how ?)
- Easy to delete all Joe's phone numbers (how ?)

Relational Schema Design (or Logical Design)

How do we do this systematically?

- Start with some relational schema
- Find out its ***functional dependencies*** (FDs)
- Use FDs to ***normalize*** the relational schema

Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

$A_1, A_2, \ldots, A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots, B_m$

Formally:

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

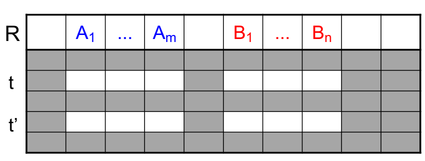
$A_1 \ldots A_n$ **determines** $B_1 \ldots B_m$

Functional Dependencies (FDs)

Definition $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **holds** in R if:

$\forall t, t' \in R,$

$(t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$



if t, t' agree here then t, t' agree here

Example

An FD holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

Position → Phone

but not Phone → Position

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

Position → Phone

Example

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

But not Phone → Position

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Do all the FDs hold on this instance?

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	49
Gizmo	Stationary	Green	Office-supp.	59

What about this one ?

Buzzwords

- FD **holds** or **does not hold** on an instance
- If we can be sure that *every instance of R* will be one in which a given FD is true, then we say that **R satisfies the FD**
- If we say that R satisfies an FD, we are **stating a constraint on R**

An Interesting Observation

If all these FDs are true:
name → color
category → department
color, category → price

Then this FD also holds: name, category → price

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds: name, category → price

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds: name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn't mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure** is the set of attributes B, notated $\{A_1, \ldots, A_n\}^+$, s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

1. name → color
2. category → department
3. color, category → price

Closures:
name$^+$ = {name, color}
color$^+$ = {color}

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and** $B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

{name, category}$^+$ =
{ name, category, }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and** $B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

{name, category}$^+$ =
{ name, category, color, }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and** $B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

{name, category}$^+$ =
{ name, category, color, department }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and** $B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

{name, category}+ =
{ name, category, color, department, price }

Closure Algorithm

X={A1, …, An}.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and** $B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

1. name → color
2. category → department
3. color, category → price

{name, category}+ =
{ name, category, color, department, price }

Hence: name, category → color, department, price

Example

In class:

R(A,B,C,D,E,F)

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}+ X = {A, B, }

Compute {A, F}+ X = {A, F, }

Example

In class:

R(A,B,C,D,E,F)

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}+ X = {A, B, C, D, E }

Compute {A, F}+ X = {A, F, }

Example

In class:

R(A,B,C,D,E,F)

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}+ X = {A, B, C, D, E }

Compute {A, F}+ X = {A, F, B, C, D, E }

Example

In class:

R(A,B,C,D,E,F)

A, B → C
A, D → E
B → D
A, F → B

Compute {A,B}+ X = {A, B, C, D, E }

Compute {A, F}+ X = {A, F, B, C, D, E }

What is the key of R?

Practice at Home

Find all FD's implied by:

A, B → C
A, D → B
B → D

Practice at Home

Find all FD's implied by:

A, B → C
A, D → B
B → D

Step 1: Compute X^+, for every X:

$A^+ = A$, $B^+ = BD$, $C^+ = C$, $D^+ = D$

$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$,
$BC^+ = BCD$, $BD^+ = BD$, $CD^+ = CD$

$ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute– why ?)

$BCD^+ = BCD$, $ABCD^+ = ABCD$

Step 2: Enumerate all FD's X → Y, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

AB → CD, AD→BC, ABC → D, ABD → C, ACD → B