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

makes

vs.

makes
Referential Integrity Constraints

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

Each product made by exactly one company.
Q: What does this mean?
A: A Company entity cannot be connected by relationship to more than 99 Product entities
Constraints in SQL:

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

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

Product(name, 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(name, category, price)

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>10</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>20</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Photo</td>
<td>30</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>40</td>
</tr>
</tbody>
</table>
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
CREATE TABLE Purchase (  
prodName CHAR(30)  
REFERENCES Product(name),  
date DATETIME)  

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

Referential integrity constraints  
May write just Product if name is PK
Foreign Key Constraints

- Example with multi-attribute primary key

```sql
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

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>ProdName</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
</tbody>
</table>
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

```sql
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

### Purchase

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Snap</td>
<td>Camera</td>
</tr>
<tr>
<td>EasyShoot</td>
<td>Camera</td>
</tr>
</tbody>
</table>
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))
```
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?

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

What is the difference from Foreign-Key?
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

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

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

Anomalies:

• **Redundancy** = repeat data
• **Update anomalies** = what if Fred moves to “Bellevue”?  
• **Deletion anomalies** = what if Joe deletes his phone number?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>
Relation Decomposition

Break the relation into two:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

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:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
Definition  \( A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \) holds in \( R \) if:

\[
\forall t, t' \in R, \\
(t.A_1 = t'.A_1 \land \ldots \land t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \land \ldots \land t.B_n = t'.B_n)
\]

<table>
<thead>
<tr>
<th>( R )</th>
<th>( A_1 )</th>
<th>( \ldots )</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>( \ldots )</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t' )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if \( t, t' \) agree here then \( t, t' \) agree here
Example

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

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

EmpID → Name, Phone, Position
Position → Phone
but not Phone → Position
## Example

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

*Position ➔ Phone*
### Example

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

**But not Phone ➔ Position**
Example

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Do all the FDs hold on this instance?
Example

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

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:

\[
\begin{align*}
\text{name} & \rightarrow \text{color} \\
\text{category} & \rightarrow \text{department} \\
\text{color}, \text{category} & \rightarrow \text{price}
\end{align*}
\]

Then this FD also holds:

\[
\begin{align*}
\text{name, category} & \rightarrow \text{price}
\end{align*}
\]
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 $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price

Closures:
$name^+ = \{\text{name, color}\}$
$color^+ = \{\text{color}\}$
Closure Algorithm

\[ X = \{A_1, \ldots, A_n\}. \]

**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 \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

\[ \{\text{name, category}\}^+ = \{ \text{name, category}, \} \]
Closure Algorithm

\( X = \{A_1, \ldots, A_n\} \).

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 \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

\( \{\text{name, category}\}^+ = \{\text{name, category, color,} \} \)
Closure Algorithm

\[ X = \{A_1, \ldots, A_n\}. \]

Repeat until \( X \) doesn’t change do:

\[
\text{if } B_1, \ldots, B_n \rightarrow C \text{ is a FD and } \\
B_1, \ldots, B_n \text{ are all in } X \\
\text{then add } C \text{ to } X.
\]

Example:

1. name \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

\[ \{\text{name, category}\}^+ = \\
\{ \text{name, category, color, department} \} \]
Closure Algorithm

\[ X = \{ A_1, \ldots, A_n \} \]

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 \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

\[ \{ \text{name, category} \}^+ = \{ \text{name, category, color, department, price} \} \]
Closure Algorithm

\[ X = \{ A_1, \ldots, A_n \} \]

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 \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

\( \{ \text{name, category} \}^+ = \{ \text{name, category, color, department, price} \} \)

Hence: \( \text{name, category} \rightarrow \text{color, department, price} \)
Example

In class:

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

A, $B \rightarrow C$
A, $D \rightarrow E$
B $\rightarrow D$
A, $F \rightarrow B$

Compute $\{A, B\}^+$ $X = \{A, B, \}$

Compute $\{A, F\}^+$ $X = \{A, F, \}$
Example

In class:

\[ R(A, B, C, D, E, F) \]

\[
\begin{array}{c}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B
\end{array}
\]

Compute \( \{A, B\}^+ \) \hspace{1cm} X = \{A, B, C, D, E\}

Compute \( \{A, F\}^+ \) \hspace{1cm} X = \{A, F, \}
Example

In class:

\[ R(A,B,C,D,E,F) \]

\[
\begin{array}{c}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B \\
\end{array}
\]

Compute \( \{A,B\}^+ \) \hspace{1cm} X = \{A, B, C, D, E\}

Compute \( \{A, F\}^+ \) \hspace{1cm} 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

What is the key of R?

Compute \( \{A, B\}^+ \) \( X = \{A, B, C, D, E\} \)

Compute \( \{A, F\}^+ \) \( X = \{A, F, B, C, D, E\} \)
Practice at Home

Find all FD’s implied by:

\[
\begin{align*}
& A, B \rightarrow C \\
& A, D \rightarrow B \\
& B \rightarrow D
\end{align*}
\]
Practice at Home

Find all FD’s implied by:

\[
\begin{align*}
A, B & \rightarrow C \\
A, D & \rightarrow B \\
B & \rightarrow D \\
\end{align*}
\]

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

\[
\begin{align*}
A^+ &= A, \quad B^+ = BD, \quad C^+ = C, \quad D^+ = D \\
AB^+ &= ABCD, \quad AC^+ = AC, \quad AD^+ = ABCD, \\
&\hspace{1cm} BC^+ = BCD, \quad BD^+ = BD, \quad CD^+ = CD \\
ABC^+ &= ABD^+ = ACD^+ = ABCD \quad \text{(no need to compute– why ?)} \\
BCD^+ &= BCD, \quad ABCD^+ = ABCD
\end{align*}
\]

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

\[
\begin{align*}
AB \rightarrow CD, \quad AD \rightarrow BC, \quad ABC \rightarrow D, \quad ABD \rightarrow C, \quad ACD \rightarrow B
\end{align*}
\]