CSCI 127
Introduction to Database Systems

Integrity Constraints and Functional Dependencies
Integrity Constraints

Purpose:

*Prevent* semantic inconsistencies in data

**e.g.:**

<table>
<thead>
<tr>
<th>cname</th>
<th>svngs</th>
<th>check</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>100</td>
<td>200</td>
<td>250</td>
</tr>
</tbody>
</table>

**e.g.:**

<table>
<thead>
<tr>
<th>cname</th>
<th>bname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Waltham</td>
</tr>
</tbody>
</table>

total ≠ savings + checking

**No entry for Waltham**
Integrity Constraints

What Are They?

- *Predicates on the database*
- *Must always be true (checked whenever db gets updated)*

The 4 Kinds of IC’s:

1. **Key Constraints (1 table)**
   e.g.: 2 *accts* can’t share same *acct_no*

2. **Attribute Constraints (1 table)**
   e.g.: *accts* must have nonnegative balance

3. **Referential Integrity Constraints (2 tables)**
   e.g.: *bnames* associated with loans must be names of real branches

4. **Global Constraints (n tables)**
   e.g.: all *loans* must be carried by at least 1 customer with a savings account
Key Constraints

Idea:

Specifies that a relation is a set, not a bag

SQL Examples:

1. Primary Key

   CREATE TABLE branch(
       bname  CHAR(15) PRIMARY KEY,
       bcity  CHAR (50),
       assets INTEGER);

   OR

   CREATE TABLE depositor(
       cname  CHAR(15),
       acct_no CHAR(5),
       PRIMARY KEY (cname, acct_no));
Key Constraints (cont.)

Idea:

Specifies that a relation is a set, not a bag

SQL Examples (cont.):

2. Candidate Key

```
CREATE TABLE customer(
    ssn CHAR(19),
    cname CHAR(15),
    address CHAR(30),
    city CHAR(10),
    PRIMARY KEY (ssn),
    UNIQUE (cname, address, city));
```
Key Constraints (cont.)

Effect of SQL Key Declarations

\[
\text{PRIMARY } (A_1, \ldots, A_n) \text{ OR UNIQUE } (A_1, \ldots, A_n)
\]

1. Insertions:

Check if inserted tuple has same values for \(A_1, \ldots, A_n\) as any previous tuple. If found, reject insertion

2. Updates to any of \(A_1, \ldots, A_n\):

Treat as insertion of entire tuple
Key Constraints (cont.)

Effect of SQL Key Declarations (cont.)

\[
\text{PRIMARY } (A_1, ..., A_n) \text{ OR UNIQUE } (A_1, ..., A_n)
\]

Primary vs. Unique (candidate):

1. **One primary key per table.**
   Several unique keys allowed.

2. **Only primary key can be referenced by “foreign key”**
   *(Referential integrity)*

3. **DBMS may treat these differently**
   *(e.g.: Putting index on primary key)*
Attribute Constraints

Idea:

- Attach constraints to value of attribute
- “Enhanced” type system
  
  \( \text{e.g.: } > 0 \text{ rather than integer} \)

In SQL:

1. `NULL`

```sql
CREATE TABLE branch(
    bname CHAR(15) NOT NULL
    ...
)
```

2. `CHECK`

```sql
CREATE TABLE depositor(
    ...(balance integer NOT NULL
    CHECK (balance \geq 0)
    ...
)
```

⇒ affect insertions, updates in affected columns
Domains:

*Can associate constraints with DOMAINS rather than attributes*

e.g.: *Instead of:*

```sql
CREATE TABLE depositor(
    ...
    balance integer NOT NULL
    CHECK (balance ≥ 0)
    ...
)
```

*One can write...*
Attribute Constraints (cont.)

Domains (cont):

CREATE DOMAIN bank-balance integer(
    CONSTRAINT not-overdrawn
    CHECK (value ≥ 0),
    CONSTRAINT not-null-value
    CHECK (value NOT NULL)
)

CREATE TABLE depositor(
    ... balance bank-balance ...
)

Q: What are the advantages of associating constraints w/ domains?
Attribute Constraints (cont.)

Advantages of Associating Constraints with Domains:

1. *Can avoid repeating specification of same constraint for multiple columns*

2. *Can name constraints*

   e.g.:  
   ```
   CREATE DOMAIN bank-balance integer(
   CONSTRAINT not-overdrawn
   CHECK (value ≥ 0),
   CONSTRAINT not-null-value
   CHECK (value NOT NULL))
   ```

Allows One To:

1. *Add or remove:*

   ```
   ALTER DOMAIN bank-balance
   ADD CONSTRAINT caped
   (CHECK value ≤ 10000)
   ```

2. *Report better errors (know which constraint violated)*
Referential Integrity Constraints

Idea:

Prevent “dangling tuples” (e.g.: A loan with bname, Waltham when no Waltham tuple in branch)

Illustrated:

Referential Integrity:

Ensure that: Foreign Key \rightarrow Primary Key value

Note: Need not ensure (i.e.: Not all branches have to have loans)
Referential Integrity Constraints

Q: Why are dangling references bad?

A: Think E/R Diagrams. In what situation do we create table A (with column containing keys of table B)

1. A represents a relationship with B, or is an entity set with an n:1 relationship with B
2. A is a weak entity dominated by B (d.r. violates weak entity condition)
3. A is a specialization of B (dang.ref. violates inheritance tree)
Referential Integrity Constraints

In SQL, Declare:

```
CREATE TABLE branch(
  bname CHAR(15) PRIMARY KEY
  ...
)
CREATE TABLE loan(
  ...
  FOREIGN KEY bname REFERENCES branch)
```

Affects:

1. Insertions, updates of **referencing** relations

2. Deletions, updates of **referenced** relation

Ensure no tuples in referencing relation left dangling

CSCI 127: Introduction to Database Systems
Referential Integrity Constraints

Q: What happens to tuples left dangling as a result of deletion/update of referenced relation?

A: 3 Possibilities

1. Reject deletion/update
2. Set \( t_i[c] \) and \( t_j[c] = \text{NULL} \)
3. Propagate deletion/update

DELETE: delete \( t_i, t_j \)
UPDATE: set \( t_j[c], t_j[c] \) to updated value
Referential Integrity Constraints

Resolving Dangling Tuples

In SQL:

```
CREATE TABLE A (...  
    FOREIGN KEY C REFERENCES B <action>  
    ...) 
```
Referential Integrity Constraints

Resolving Dangling Tuples (cont.)

Deletion:

1. *(Left blank): Deletion/update rejected*

2. **ON DELETE SET NULL / ON UPDATE SET NULL**
   
   "sets \( t_i[c] = \text{NULL} \), \( t_j[c] = \text{NULL} \)"

3. **ON DELETE CASCADE**
   
   "delete \( t_i \), delete \( t_j \)"

   **ON UPDATE CASCADE**

   "sets \( t_i[c] \), \( t_j[c] \) to new Key value"
Global Constraints

Idea:

1. **Single relation (constraint spans multiple columns)**
   
   e.g.: \( \text{CHECK (total = svngs + check)} \)
   
   declared in \( \text{CREATE TABLE} \) for relation

2. **Multiple relations**

   \( \text{CREATE ASSERTIONS} \)
Global Constraints (cont.)

SQL Example (cont.):

*Multiple relations: Every loan has a borrower with a savings account*

```
CHECK (NOT EXISTS(
  SELECT *
  FROM loan AS l
  WHERE NOT EXISTS(
    SELECT *
    FROM borrower AS b, depositor AS d, account AS a,
    WHERE b.cname = d.cname AND d.acct_no = a.acct_no
    AND l.lno = b.lno))

CHECK (NOT EXISTS(
  SELECT *
  FROM loan AS l
  WHERE NOT EXISTS(
    SELECT *
    FROM borrower AS b, depositor AS d, account AS a,
    WHERE b.cname = d.cname AND d.acct_no = a.acct_no
    AND l.lno = b.lno)))
```
Global Constraints (cont.)

SQL Example (cont.):

Multiple relations: Every loan has a borrower with a savings account (cont.)

Problem:

With which table’s definition does this go?
(loan?, depositor?, ...)

A: None of the above

CREATE ASSERTION loan-constraint
CHECK (NOT EXISTS...)

Checked with EVERY DB update! VERY EXPENSIVE...
# Integrity Constraints: Summary

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Where Declared</th>
<th>Affects…</th>
<th>Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Constraints</td>
<td><code>CREATE TABLE (PRIMARY KEY, UNIQUE)</code></td>
<td><em>Insertions, updates</em></td>
<td><em>Moderate</em></td>
</tr>
</tbody>
</table>
## Integrity Constraints: Summary

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Where Declared</th>
<th>Affects…</th>
<th>Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Constraints</strong></td>
<td>CREATE TABLE (PRIMARY KEY, UNIQUE)</td>
<td><em>Insertions, updates</em></td>
<td>Moderate</td>
</tr>
<tr>
<td><strong>Attribute Constraints</strong></td>
<td>CREATE TABLE CREATE DOMAIN (NOT NULL, CHECK)</td>
<td><em>Insertions, updates</em></td>
<td>Cheap</td>
</tr>
</tbody>
</table>
# Integrity Constraints: Summary

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Where Declared</th>
<th>Affects…</th>
<th>Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Constraints</strong></td>
<td>CREATE TABLE (PRIMARY KEY, UNIQUE)</td>
<td>Insertions, updates</td>
<td><em>Moderate</em></td>
</tr>
<tr>
<td><strong>Attribute Constraints</strong></td>
<td>CREATE TABLE CREATE DOMAIN (NOT NULL, CHECK)</td>
<td>Insertions, updates</td>
<td><em>Cheap</em></td>
</tr>
</tbody>
</table>
| **Referential Integrity** | Table tag (FOREIGN KEY REFERENCES ...)              | 1. Insertions into referencing relation  
2. Updates of referencing relation of relevant att’s  
3. Deletions from referenced relations  
4. Updates of referenced relations | 1,2: Like key constraints. Another reason to index/sort on primary keys  
3,4: Depends on  
a. update/delete policy chosen  
b. Existence of indexes on foreign keys |
## Integrity Constraints: Summary

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Where Declared</th>
<th>Affects…</th>
<th>Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Constraints</strong></td>
<td><code>CREATE TABLE PRIMARY KEY, UNIQUE</code></td>
<td><code>Insertions, updates</code></td>
<td>Moderate</td>
</tr>
<tr>
<td><strong>Attribute Constraints</strong></td>
<td><code>CREATE TABLE CREATE DOMAIN NOT NULL, CHECK</code></td>
<td><code>Insertions, updates</code></td>
<td>Cheap</td>
</tr>
<tr>
<td><strong>Referential Integrity</strong></td>
<td><code>FOREIGN KEY REFERENCES</code></td>
<td>1. Insertions into referencing relation</td>
<td>1,2: Like key constraints.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Updates of referencing relation of relevant att’s</td>
<td>Another reason to index/sort on primary keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Deletions from referenced relations</td>
<td>3,4: Depends on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Updates of referenced relations</td>
<td>a. update/delete policy chosen</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b. Existence of indexes on foreign keys</td>
</tr>
<tr>
<td><strong>Global Constraints</strong></td>
<td><code>Outside tables create assertion</code></td>
<td>1. For single relation constraint, with insertions, updates of relevant att’s</td>
<td>1. Cheap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. For assertions, with every database modification</td>
<td>2. Very Expensive</td>
</tr>
</tbody>
</table>

CSCI 127: Introduction to Database Systems
An Example:

\[ \text{loan-info} = \]

<table>
<thead>
<tr>
<th>bname</th>
<th>lno</th>
<th>cname</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>L-17</td>
<td>Jones</td>
<td>1000</td>
</tr>
<tr>
<td>Dntn</td>
<td>L-17</td>
<td>Williams</td>
<td>1000</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-23</td>
<td>Smith</td>
<td>1000</td>
</tr>
<tr>
<td>Perry</td>
<td>L-15</td>
<td>Hayes</td>
<td>1500</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-23</td>
<td>Johnson</td>
<td>1000</td>
</tr>
</tbody>
</table>

**True or False?**

\[
\begin{align*}
\text{amt} & \rightarrow \text{lno} \? \\
\text{lno} & \rightarrow \text{cname} \? \\
\text{lno} & \rightarrow \text{lno} \? \\
\text{bname} & \rightarrow \text{lno} \? \\
\end{align*}
\]

*Can’t always decide by looking at populated db’s*

Observe:

*Tuples with the same value for lno will always have the same value for amt*

*We write: lno \(\rightarrow\) amt*  
(lno “determines” amt, or amt is “functionally determined” by lno)
Functional Dependencies

In general:

\[ A_1, \ldots, A_n \rightarrow B \]

Informally:

*If 2 tuples “agree” on their values for \( A_1, \ldots, A_n \), they will also agree on their values for \( B \).*

Formally:

\[ \forall t, u \ (t[A_1] = u[A_1] \land t[A_2] = u[A_2] \land \ldots \land t[A_n] = u[A_n] \Rightarrow t[B] = u[B]) \]
Functional Dependencies

Another Example:

*Drinkers*

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>likes</th>
<th>lmanf</th>
<th>fave</th>
<th>fmanf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homer</td>
<td>WS</td>
<td>Bud</td>
<td>AB</td>
<td>Duff</td>
<td>SB</td>
</tr>
<tr>
<td>Homer</td>
<td>WS</td>
<td>Duff</td>
<td>SB</td>
<td>Duff</td>
<td>SB</td>
</tr>
<tr>
<td>Apu</td>
<td>ES</td>
<td>Bud</td>
<td>AB</td>
<td>Bud</td>
<td>AB</td>
</tr>
</tbody>
</table>

What are the FD’s?

- `likes \rightarrow lmanf`
- `fave \rightarrow fmanf`
- `name \rightarrow fave`
- `name \rightarrow addr (?)`
Back to Global Integrity Constraints

How Do We Decide What Constraints to Impose?

Consider Drinkers \((\text{name, addr, likes, lmanf, fave, fmanf})\) with FD’s: \(\text{name} \rightarrow \text{addr}, \ldots\)

Q: How do we ensure that \(\text{name} \rightarrow \text{addr}\)?

A: CREATE ASSERTION name-addr
   CHECK (NOT EXISTS
            (SELECT *
             FROM Drinkers AS \(d_1\), Drinkers AS \(d_2\)
             WHERE ?))

\(? \equiv d_1.\text{name} = d_2.\text{name} \text{ AND } d_1.\text{addr} \not= d_2.\text{addr}\)
Back to Functional Dependencies

How to derive them?

1. **Key Constraints**
   (e.g.: `bname` a key for `branch`)

   \[
   \text{Therefore: } \begin{align*}
   bname & \rightarrow bname \\
   bname & \rightarrow \text{city} \\
   bname & \rightarrow \text{assets}
   \end{align*}
   \]

   \[
   \text{will instead write: } \begin{align*}
   bname & \rightarrow bname \text{ bcity assets}
   \end{align*}
   \]

Q: **Define “Super Keys” in terms of FD’s**

A: Any set of attributes in a relation that functionally determines all attributes in the relation

Q: **Define “Candidate Key” in terms of FD’s**

A: Any super key such that the removal of any attribute leaves a set that does not functionally determine all attributes
## Functional Dependencies

### How to Derive Them?

1. **Key Constraints**
   
2. **n:1 relationships**
   
3. **Laws of Physics**

4. **Trial-and-error**

*Given* $R = (A, B, C)$, *try each of the following to see if they make sense.*

<table>
<thead>
<tr>
<th>$A \rightarrow B$</th>
<th>$C \rightarrow A$</th>
<th>$BC \rightarrow A$</th>
<th>$A \rightarrow C$</th>
<th>$C \rightarrow B$</th>
<th>$B \rightarrow A$</th>
<th>$AB \rightarrow C$</th>
<th>$B \rightarrow C$</th>
<th>$AC \rightarrow B$</th>
<th>$AB \rightarrow A$</th>
<th>$C \rightarrow C$</th>
</tr>
</thead>
</table>

*Just write: “... plus all of the trivial dependencies”*
2. Avoiding the Expense

**Recall:** \( \text{name} \rightarrow \text{addr} \) preserved by

```
CHECK (NOT EXISTS
  (SELECT *
   FROM Drinkers AS d1, Drinkers AS d2
   WHERE d1.name = d2.name AND d1.addr <> d2.addr))
```

**Q:** Is it necessary to have an assertion for every FD?

**A:** Luckily, no. Can preprocess FD set

- Some FD’s can be eliminated
- Some FD’s can be combined
Functional Dependencies

Combining FD’s:

a. name \rightarrow addr

CREATE ASSERTION name-addr
CHECK (NOT EXISTS
  (SELECT *
  FROM Drinkers AS d_1, Drinkers AS d_2
  WHERE d_1.name = d_2.name AND d_1.addr <> d_2.addr))

b. name \rightarrow fave

CREATE ASSERTION name-fave
CHECK (NOT EXISTS
  (SELECT *
  FROM Drinkers AS d_1, Drinkers AS d_2
  WHERE d_1.name = d_2.name AND d_1.fave <> d_2.fave))
Combining FD’s (cont.):

Combine into: \(\text{name} \rightarrow \text{addr fave}\)

CREATE ASSERTION name-addr
    CHECK (NOT EXISTS(SELECT *
                        FROM Drinkers AS d_1, Drinkers AS d_2
                        WHERE d_1.name = d_2.name AND ?))

? \equiv (d_1.addr \not\equiv d_2.addr) \text{ OR } (d_1.fave \not\equiv d_2.fave)
Determining Unnecessary FD’s

Consider: name → name

CREATE ASSERTION name-name
CHECK(NOT EXISTS
(SELECT *
FROM Drinkers AS d₁, Drinkers AS d₂
WHERE d₁.name = d₂.name AND d₁.name <> d₂.name))

Cannot possibly be violated!
Functional Dependencies

Note:

\[ X \rightarrow Y \text{ s.t. } Y \supseteq X \text{ is a "trivial dependency" } \]
\[ (true, \text{ regardless of attributes involved}) \]

Moral:

Don’t create assertions for trivial dependencies
Functional Dependencies

Determining Unnecessary FD’s

Even non-trivial FD’s can be unnecessary

e.g.:

1. name → fave
   CREATE ASSERTION name-fave
     CHECK (NOT EXISTS
       SELECT *
       FROM Drinkers AS d_1, Drinkers AS d_2
       WHERE d_1.name = d_2.name AND d_1.fave <> d_2.fave)

2. fave → fmanf
   CREATE ASSERTION fave-fmanf
     CHECK (NOT EXISTS
       SELECT *
       FROM Drinkers AS d_1, Drinkers AS d_2
       WHERE d_1.fave = d_2.fave AND d_1.fmanf <> d_2.fmanf)
Determining Unnecessary FD’s (cont.)

Even non-trivial FD’s can be unnecessary (cont.)

e.g.:

3. name → fmanf

CREATE ASSERTION name-fmanf
CHECK (NOT EXISTS
SELECT *
FROM Drinkers AS d1, Drinkers AS d2
WHERE d1.name = d2.name AND d1.fmanf <> d2.fmanf)

Note: If 1 and 2 succeed, 3 must also
Functional Dependencies

Using FD’s to Determine Global IC’s:

**Step 1:** Given schema \( R = \{ A_1, \ldots, A_n \} \)

*Use key constraints, n:1 relationships, laws of physics and trial-and-error to determine an initial FD set, \( F \)*

**Step 2:**

*Use FD elimination techniques to generate an alternative (but equivalent) FD set, \( F' \)*

**Step 3:**

*Write assertions for each \( f \in F' \) (for now)***
Functional Dependencies

Using FD’s to Determine Global IC’s (cont.):

**Issues:**

1. *How do we guarantee that* $F = F'$ ?
   
   **A:** Closures

2. *How do we find a “minimal”* $F = F'$ ?
   
   **A:** Canonical cover algorithm
Functional Dependencies

Example:

Suppose:

\[ R = \{A, B, C, D, E, H\} \text{ and we determine that:} \]

\[ F = \{A \rightarrow BC, B \rightarrow CE, A \rightarrow E, AD \rightarrow H, D \rightarrow B\} \]

Then we determine the canonical cover of \( F \):

\[ F_c = \{A \rightarrow BH, B \rightarrow CE, D \rightarrow B\} \]

ensuring that \( F \) and \( F_c \) are equivalent

Note:

\( F \) requires 5 assertions
\( F_c \) requires 3 assertions
Functional Dependencies

Equivalent of FD Sets:

*FD sets $F$, $G$ are equivalent if they *imply* the same set of FD’s*

*E.g.*:

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

*Implies* $A \rightarrow C$

*Equivalence usually expressed in terms of closures*

Closures:

*For any FD set, $F$, $F^+$ is the set of all FD’s implied by $F$.*

*Can calculate in 2 ways:*

1. Attribute closures
2. Armstrong’s axioms

*Both techniques are tedious $\rightarrow$ we will do only for toy examples*

Note: $F$ equivalent to $G$ if and only if $F^+ = G^+$
Functional Dependencies

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>α</td>
<td>1</td>
<td>u</td>
</tr>
<tr>
<td>a</td>
<td>α</td>
<td>1</td>
<td>u</td>
</tr>
<tr>
<td>a</td>
<td>β</td>
<td>5</td>
<td>w</td>
</tr>
<tr>
<td>b</td>
<td>β</td>
<td>3</td>
<td>w</td>
</tr>
<tr>
<td>b</td>
<td>β</td>
<td>3</td>
<td>w</td>
</tr>
</tbody>
</table>

Shorthand:

\[ C \rightarrow BD \quad \text{same as} \quad C \rightarrow B \]
\[ C \rightarrow D \]

Be Careful!

\[ AB \rightarrow C \quad \text{not the same as} \quad A \rightarrow C \]
\[ B \rightarrow C \]

true \quad \text{not true}
Attribute Closures

Given:

\[ R = \{A, B, C, D, E, H\} \]
\[ F = \{A \rightarrow BC, \]
\[ B \rightarrow CE, \]
\[ A \rightarrow E, \]
\[ AC \rightarrow H, \]
\[ D \rightarrow B\} \]

Q: What is the closure of \( CD \) (i.e., \( CD^+ \))?

A: The set of attributes that can be determined from \( CD \).
**Q:** What is the closure of \( CD(CD^+) \) ?

**A:** Algorithm attr-closure (\( X: \) set of attributes)

\[
\text{result} \leftarrow X \\
\text{repeat until stable} \\
\quad \text{for each FD in } F, Y \rightarrow Z, \text{ do} \\
\quad \quad \text{if } Y \subseteq \text{result} \text{ then} \\
\quad \quad \quad \text{result} \leftarrow \text{result} \cap Z
\]

**e.g.:** attr-closure (CD)

\[
\begin{array}{|c|c|}
\hline
\text{Iteration} & \text{Result} \\
\hline
0 & CD \\
\hline
\end{array}
\]

\[
R = \{A, B, C, D, E, H\} \\
F = \{A \rightarrow BC, \ B \rightarrow CE, \ A \rightarrow E, \ AC \rightarrow H, \ D \rightarrow B\}
\]
Q: What is the closure of \( CD (CD^+) \)?

A: Algorithm attr-closure (\( X: \) set of attributes)

result \( \leftarrow X \)

repeat until stable

for each FD in \( F, Y \rightarrow Z \), do

if \( Y \subseteq \) result then

result \( \leftarrow \) result \( \cap Z \)

e.g.: attr-closure (CD)

\[
\begin{array}{|c|c|}
\hline
\text{Iteration} & \text{Result} \\
\hline
0 & CD \\
1 & CDB \\
\hline
\end{array}
\]

\[
R = \{A, B, C, D, E, H\}
\]

\[
F = \{A \rightarrow BC, \quad B \rightarrow CE, \quad A \rightarrow E, \quad AC \rightarrow H, \quad D \rightarrow B\}
\]
Q: What is the closure of $CD (CD^+)$?

A: Algorithm `attr-closure` ($X$: set of attributes)

```
result ← $X$
repeat until stable
  for each FD in $F$, $Y → Z$, do
    if $Y \subseteq$ result then
      result ← result ∩ $Z$
```

e.g.: `attr-closure` ($CD$)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CD</td>
</tr>
<tr>
<td>1</td>
<td>CDB</td>
</tr>
<tr>
<td>2</td>
<td>CDBE</td>
</tr>
</tbody>
</table>

$R = \{A, B, C, D, E, H\}$

$F = \{A → BC,$
  $B → CE,$
  $A → E,$
  $AC → H,$
  $D → B\}$
Attribute Closures

Q: What is $\text{ACD}^+$?

A: $\text{ACD}^+ \rightarrow R$

Q: How can you determine if $\text{ACD}$ is a super key?

A: *It is if* $\text{ACD}^+ \rightarrow R$

Q: How can you determine if $\text{ACD}$ is a candidate key?

A: *It is if:* $\text{ACD}^+ \rightarrow R$, and

None of ($\text{AC}^+ \rightarrow R$, $\text{AD}^+ \rightarrow R$, $\text{CD}^+ \rightarrow R$) are true.
Using Attribute Closures To Determine FD Set Closures

Given:

\[ F = \{ A \to BC, \quad B \to CE, \quad A \to E, \quad AC \to H, \quad D \to B \} \]

\[ F^+ = \{ A \to A^+, \quad B \to B^+, \quad C \to C^+, \quad D \to D^+, \quad E \to E^+, \quad H \to H^+, \quad AB \to AB^+, \quad AC \to AC^+, \quad AD \to AD^+, \quad AE \to AE^+, \quad AH \to AH^+, \quad BC \to BC^+, \quad BD \to BD^+, \quad \ldots \} \]

To Decide if \( F, \ G \) Are Equivalent:

1. \( \text{Compute } F^+ \)
2. \( \text{Compute } G^+ \)
3. \( \text{Is } F^+ = G^+ ? \)

Expensive:

\( F^+ \) has 63 rules (in general: \( O(2^{\mid R \mid}) \) rules)
FD Closures Using Armstrong’s Axioms

A. Fundamental Rules (W, X, Y, Z: sets of attributes)

1. Reflexivity
   If \( Y \subseteq X \) then \( X \rightarrow Y \)

2. Augmentation
   If \( X \rightarrow Y \) then \( WX \rightarrow WY \)

3. Transitivity
   If \( X \rightarrow Y \) and \( Y \rightarrow Z \) then \( X \rightarrow Z \)
B. Additional rules (can be proved from 1 through 3)

4. **Union**
   
   If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrow YZ \)

5. **Decomposition**
   
   If \( X \rightarrow YZ \) then \( X \rightarrow Y \) and \( X \rightarrow Z \)

6. **Pseudotransitivity**
   
   If \( X \rightarrow Y \) and \( WY \rightarrow Z \), then \( WX \rightarrow Z \)
FD Closures Using Armstrong’s Axioms

Given:

\[ F = \{ A \rightarrow BC, \ (1) \]
\[ B \rightarrow CE, \ (2) \]
\[ A \rightarrow E, \ (3) \]
\[ AC \rightarrow H, \ (4) \]
\[ D \rightarrow B \} \ (5) \]

Exhaustively Apply Armstrong’s Axioms to Generate \( F^+ \):

\[ F^+ = F \ U \]

1. \( \{(6) \ A \rightarrow B, \ (7) \ A \rightarrow C\} \)
   ... decomposition on (1)
2. \( \{(8) \ A \rightarrow CE\} \)
   ... transitivity on (6),(2)
3. \( \{(9) \ B \rightarrow C, \ (10) \ B \rightarrow E\} \)
   ... decomposition on (2)
4. \( \{(11) \ A \rightarrow C, \ (12) \ A \rightarrow E\} \)
   ... decomposition on (8)
5. \( \{(13) \ A \rightarrow H\} \)
   ... pseudotransitivity on (1),(4)

...
Functional Dependencies

Our Goal:

Given FD set, \( F \), find an alternative FD set, \( G \), that is:

1. Smaller
2. Equivalent

Bad News:

Testing \( F \equiv G \ (F^+ = G^+) \) is computationally expensive

Good News: Canonical Cover Algorithm (CCA)

Given FD set, \( F \), CCA finds minimal FD set equivalent to \( F \)

Minimal: can’t find another equivalent FD set with fewer FD’s
Canonical Cover Algorithm

Given:

$$F = \{ A \rightarrow BC, B \rightarrow CE, A \rightarrow E, AC \rightarrow H, D \rightarrow B \}$$

Determine canonical cover of $F$:

$$F_{c} = \{ A \rightarrow BH, B \rightarrow CE, D \rightarrow B \}$$

No $G$ that is equiv. to $F$ is smaller than $F_{c}$

Another Example:

$$F = \{ A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C, AC \rightarrow D \}$$

$F_{c} = \{ A \rightarrow BH, B \rightarrow C \}$
Canonical Cover Algorithm

Basic Algorithm

ALGORITHM canonical-cover (X: FD Set)
BEGIN

REPEAT UNTIL STABLE
1. Where possible, apply UNION rule (A’s Axioms)
   (e.g.: $A \rightarrow BC, A \rightarrow CD$ becomes $A \rightarrow BCD$)

2. Remove “extraneous attributes” from each FD
   (e.g.: $AB \rightarrow C, A \rightarrow B$ becomes $A \rightarrow B, B \rightarrow C$
   i.e.: $A$ is extraneous in $AB \rightarrow C$)

END
Extraneous Attributes

1. Extraneous in RHS?
   
   e.g.: Can we replace $A \rightarrow BC$ with $A \rightarrow C$?
   (i.e.: Is $B$ extraneous in $A \rightarrow BC$?)

2. Extraneous in LHS?
   
   e.g.: Can we replace $AB \rightarrow C$ with $A \rightarrow C$?
   (i.e.: Is $B$ extraneous in $AB \rightarrow C$?)

Simple (but expensive) test:

1. Replace $A \rightarrow BC$ (or $AB \rightarrow C$) with $A \rightarrow C$ in $F$

Define $F_2 = F - \{A \rightarrow BC\} \cup \{A \rightarrow C\}$ OR
$F_2 = F - \{AB \rightarrow C\} \cup \{A \rightarrow C\}$

2. Test: Is $F_2^+ = F^+$? If yes, then $B$ was extraneous
Extraneous Attributes

A. RHS: Is $B$ extraneous in $A \rightarrow BC$?

Step 1: $F_2 = F - \{A \rightarrow BC\} \cup \{A \rightarrow C\}$

Step 2: $F^+ = F_2^+$?

To simplify step 2, observe that $F_2^+ \subseteq F^+$
(i.e.: no new FD’s in $F_2^+$)

Why? **Have effectively removed** $A \rightarrow B$ from $F$

When is $F^+ = F_2^+$?

A: When $(A \rightarrow B) \in F_2^+$

Idea: *If* $F_2^+$ *includes*: $A \rightarrow B$ and $A \rightarrow C$, *then it includes* $A \rightarrow BC$
Extraneous Attributes

B. LHS: Is $B$ extraneous in $AB \rightarrow C$?

*Step 1:* $F_2 = F \setminus \{AB \rightarrow C\} \cup \{A \rightarrow C\}$

*Step 2:* $F^+ = F_2^+$?

To Simplify step 2, observe that $F^+ \supseteq F_2^+$ (i.e.: there may be new FD’s in $F_2^+$)

**Why?**

$A \rightarrow C$ “implies” $AB \rightarrow C$.

Thus, all FD’s in $F^+$ also in $F_2^+$.

But $AB \rightarrow C$ does not “imply” $A \rightarrow C$.

Thus, all FD’s in $F_2^+$, not necessarily in $F^+$.

**When is $F^+ = F_2^+$?**

**A:** When $(A \rightarrow C) \in F^+$

**Idea:** If $(A \rightarrow C) \in F^+$, then it will include all FD’s of $F_2^+$

CSCI1270: Introduction to Database Systems
Extraneous Attributes

A. RHS:

\[
\text{Given } F = \{A \rightarrow BC, \ B \rightarrow C\},
\]

is C extraneous in A \rightarrow BC?

Why or why not?

A: Yes, because

\[(A \rightarrow C) \in \{A \rightarrow B, \ B \rightarrow C\}^+\]

Proof:

1. A \rightarrow B \quad \text{Given}
2. B \rightarrow C \quad \text{Given}
3. A \rightarrow C \quad \text{transitivity, (1) and (2)}
Canonical Cover Algorithm

ALGORITHM canonical-cover (X: FD Set)
BEGIN

REPEAT UNTIL STABLE
1. Where possible, apply UNION rule (A’s Axioms)

2. Remove all extraneous attributes:
   a. Test if B extraneous in A → BC
      (B extraneous if (A → B) ∈
      \( (F - \{A → BC\} U \{A → C\})^+ \)) = \( F_2^+ \)

   b. Test if B extraneous in AB → C
      (B extraneous if (A → C) ∈ \( F^+ \))

END
Canonical Cover Algorithm

Example:  Determine the canonical cover of
F = \{A \rightarrow BC, \ B \rightarrow CE, \ A \rightarrow E\}

Iteration 1:

a.  F = \{A \rightarrow BCE, \ B \rightarrow CE\}

b.  Must check for up to 5 extraneous attributes

•  B extraneous in A \rightarrow BCE?  \ No
•  C extraneous in A \rightarrow BCE?

Yes:  (A \rightarrow C) \in \{A \rightarrow BE, \ B \rightarrow CE\}^+

1.  A \rightarrow BE      Given
2.  A \rightarrow B      Decomposition (1)
3.  B \rightarrow CE     Given
4.  B \rightarrow C      Decomposition (3)
5.  A \rightarrow C      Trans (2,4)

•  E extraneous in B \rightarrow CE?   ...

CSCI1270: Introduction to Database Systems
Canonical Cover Algorithm

Example (cont.): \( F = \{ A \rightarrow BC, B \rightarrow CE, A \rightarrow E \} \)

Iteration 1:

a. \( F = \{ A \rightarrow BCE, B \rightarrow CE \} \)

b. Extraneous attrs:

- \( B \) extraneous in \( A \rightarrow BCE \)?  No
- \( C \) extraneous in \( A \rightarrow BCE \)?  Yes...
- \( E \) extraneous in \( A \rightarrow BCE \)?

Yes: \((A \rightarrow E) \in \{A \rightarrow B, B \rightarrow CE\}^+\)

1. \( A \rightarrow B \)   Given
2. \( B \rightarrow CE \)  Given
3. \( B \rightarrow E \)  Decomposition (2)
4. \( A \rightarrow E \)  Trans (1,3)

- \( E \) extraneous in \( B \rightarrow CE \)?  No
- \( C \) extraneous in \( B \rightarrow CE \)?  No
Canonical Cover Algorithm

Example (cont.): $F = \{A \rightarrow BC, \ B \rightarrow CE, \ A \rightarrow E\}$

Iteration 1:

a. $F = \{A \rightarrow BCE, \ B \rightarrow CE\}$

b. Extraneous attrs:
   - B extraneous in $A \rightarrow BCE$? No
   - C extraneous in $A \rightarrow BCE$? Yes...
   - E extraneous in $A \rightarrow BE$? Yes...
   - E extraneous in $B \rightarrow CE$? No
   - C extraneous in $B \rightarrow CE$? No

Iteration 2:

a. $F = \{A \rightarrow B, \ B \rightarrow CE\}$

b. Extraneous attrs:
   - E extraneous in $B \rightarrow CE$? No
   - C extraneous in $B \rightarrow CE$? No

DONE!
Functional Dependencies So Far…

1. Canonical Cover Algorithm

Result \((F_c)\) guaranteed to be minimal FD set equivalent to \(F\)

2. Closure Algorithms

   a. Armstrong’s Axioms:
      More common use: test for extraneous atts in CC algorithm

   b. Attribute closure:
      More common use: test if set of atts is a super key

3. Purpose

   a. Minimize cost of global integrity constraints
      So far: \(\text{min gic’s} = |F_c|\)
      In fact: \(\text{min gic’s} = 0\) (FD’s for “normalization”)
Functional Dependencies

So Far, have used for:

1. Determining global integrity constraints
2. Minimizing global integrity constraints (canonical cover)
3. Deciding if some attribute set is a key (attribute closure)

Next: Influencing schema design (normalization)