Functional Dependencies

CS 1270
Constraints

We use constraints to enforce **semantic** requirements on a DBMS.

Predicates that the DBMS must ensure to be always true.

Predicates are checked when the DBMS chooses to; most of the time this is on INSERT/DELETE.
Types of Constraints

1. Key Constraints (1 table)
2. Attribute Constraints (1 table)
3. Referential Integrity Constraints (2 tables)
4. Global Constraints ($n$ tables)
Attribute & Key Constraints

Key Constraints:
Mainly Primary and Candidate Keys
Defined with PRIMARY KEY, UNIQUE

Attribute Constraints:
Constraints on a single column. Mostly NOT NULL and basic constraints like > 0 etc.
Domains

Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK)

Eg: You want to create a table “bank_account” with “account_type” field having value either “Checking” or “Saving”:

```sql
CREATE DOMAIN account_type char(12)

CONSTRAINT acc_type_test
    check(value in(“Checking”,”Saving”));

CREATE TABLE bank_account( acc_no INT PRIMARY KEY,
    acc_holder_name VARCHAR(30),
    acc_type account_type);
```
Referential Constraints

Allows values associated with certain attributes to appear for certain attributes in another relation
Foreign key in the referencing (child) table should correspond to a Primary Key in the referenced (Parent) table
The main idea behind this is to avoid dangling tuples.

This affects update/delete operation in the following ways:
  a. Insertions/uploads in the child relation

  b. Delete/update in the parent relation:
Referential Constraints Continued

How to deal with dangling tuples:

a. Reject update/deletion
b. Set value in dangling tuple to NULL
c. Cascade the operation
   i. On update/delete to parent, update/delete child
Global Constraints

Constraints that the DB enforces across all relations (or a single relation). Can be very expensive.

1. Single Table: CHECK(savings + expenses > 0)
   a. Enforced at a single table level and may use multiple columns

2. Multiple Relations: Create Assertion
   a. Enforced on any DB change/update
   b. EX: CREATE ASSERTION loan-constraint
      CHECK (NOT EXIST (SELECT ...))
Functional Dependencies

Functional Dependencies are used to define constraints between two attributes of a given relation.

Given a relation $R$, a set of attributes $X$ in $R$ is said to functionally determine another set of attributes $Y$, also in $R$, (written $X \rightarrow Y$) if, and only if, each $X$ value in $R$ is associated with precisely one $Y$ value in $R$. 
Main Uses

- Selecting constraints to enforce
- Help in schema design
Closure

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

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 $X \rightarrow Z$

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 $WX \rightarrow Z$
Closure Algorithm 1

- Closure = S
- Loop
  - For each F in S, apply reflexivity and augmentation rules
  - Add the new FDs to the Closure
  - For each pair of FDs in S, apply the transitivity rule
  - Add the new Fd to Closure
- Until closure doesn't change any further
Closure Algorithm 2

```
algorithm (F)
/* F is a set of FDs */
1. \( F^+ = \emptyset \)
2. for each possible attribute set \( X \)
3. compute the closure \( X^+ \) of \( X \) on \( F \)
4. for each attribute \( A \in X^+ \)
5. add to \( F^+ \) the FD: \( X \rightarrow A \)
5. return \( F^+ \)
```
Example. Assume that there are 4 attributes \(A, B, C, D\), and that \(F = \{A \rightarrow B, B \rightarrow C\}\). To compute \(F^+\), we first get:

- \(A^+ = AB^+ = AC^+ = ABC^+ = \{A, B, C\}\)
- \(B^+ = BC^+ = \{B, C\}\)
- \(C^+ = \{C\}\)
- \(D^+ = \{D\}\)
- \(AD^+ = \{A, D\}\)
- \(BC^+ = \{B, C\}\)
- \(BD^+ = BCD^+ = \{B, C, D\}\)
- \(ABD^+ = ABCD^+ = \{A, B, C, D\}\)
- \(ACD^+ = \{A, C, D\}\)

It is easy to generate the FDs in \(F^+\) from the closures of the above attribute sets.
Closure Example

\[ R = (A, B, C, D) \]

\[ F = \{ A \rightarrow BC; C \rightarrow D \} \]
\{A\}^+ = \{A, B, C, D\} \quad \text{Minimum candidate key}
\{B\}^+ = \{B\}
\{C\}^+ = \{C, D\}
\{D\}^+ = \{D\}
\{A, B\}^+ = \{A, B, C, D\} \quad \text{Superkey}
\{A, C\}^+ = \{A, B, C, D\} \quad \text{Superkey}
\{B, C\}^+ = \{B, C, D\}
\{A, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}
\{B, D\}^+ = \{B, D\}
\{C, D\}^+ = \{C, D\}
\{A, B, C\}^+ = \{A, B, C, D\} \quad \text{Superkey}
\{A, B, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}
\{A, C, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}
\{B, C, D\}^+ = \{B, C, D\}
\{A, B, C, D\}^+ = \{A, B, C, D\} \quad \text{Superkey}

To practice: http://raymondcho.net/RelationalDatabaseTools/RelationalDatabaseTools
Canonical Cover

A minimal set of dependencies \( C \) which imply every FD defined in the closure of \( F \)

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 \( A \rightarrow BC \)
                  (B extraneous if \( (A \rightarrow B) \in (F - \{A \rightarrow BC\} U \{A \rightarrow C\})^+ \) = \( F^+ \)
               b. Test if B extraneous in \( AB \rightarrow C \)
                  (B extraneous if \( (A \rightarrow C) \in F^+ \))
    END
Canonical Cover Example

$F = \{ A \rightarrow BC; B \rightarrow C; A \rightarrow B; AB \rightarrow C \}$

$F = \{ A \rightarrow BC; B \rightarrow C; AB \rightarrow C \}$  \hspace{1cm} \text{Combine } A \rightarrow B \text{ and } A \rightarrow BC, \text{ since they share information}

$F = \{ A \rightarrow BC; B \rightarrow C \}$  \hspace{1cm} \text{Knowing } B \rightarrow C, \text{ then } AB \rightarrow C \text{ doesn’t add any information}

$F = \{ A \rightarrow B; B \rightarrow C \}$  \hspace{1cm} \text{Knowing } A \rightarrow B, \text{ with } B \rightarrow C, \text{ implies that } A \rightarrow C, \text{ so no need to keep } A \rightarrow BC$
Another Example

\[ F = \{ A \to BC; AC \to D, B \rightarrow D, A \rightarrow D \} \]
\{ A \rightarrow B, C; B \rightarrow D \}
When FD's are used for decomposition, the decomposed relations should have the following characteristics:

1. **Lossless Joins**: No information should be lost
   a. i.e if R is decomposed into R1 and R2 then the natural join should result in R and no extra records should be added
2. **Dependency Preservation** (Time Efficiency)
3. **Redundancy Avoidance** (Space Efficiency)
Decomposition Example

\[ R = (\text{SSN, Name, Address}) \]

\[ R_1(\text{SSN, Name}) \]
\[ R_2(\text{Name, Address}) \]
### LOSSY DECOMPOSITION:

<table>
<thead>
<tr>
<th>SNN</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>J</td>
<td>MN</td>
</tr>
<tr>
<td>22</td>
<td>A</td>
<td>OP</td>
</tr>
<tr>
<td>33</td>
<td>A</td>
<td>OQ</td>
</tr>
</tbody>
</table>

**Subtree 1:**

<table>
<thead>
<tr>
<th>SSN</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>J</td>
</tr>
<tr>
<td>22</td>
<td>A</td>
</tr>
</tbody>
</table>

**Subtree 2:**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>MN</td>
</tr>
<tr>
<td>A</td>
<td>OP</td>
</tr>
<tr>
<td>A</td>
<td>OQ</td>
</tr>
</tbody>
</table>

**Subtree 3:**

<table>
<thead>
<tr>
<th>SNN</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>J</td>
<td>MN</td>
</tr>
<tr>
<td>22</td>
<td>A</td>
<td>OP</td>
</tr>
<tr>
<td>22</td>
<td>A</td>
<td>OQ</td>
</tr>
<tr>
<td>33</td>
<td>A</td>
<td>OP</td>
</tr>
<tr>
<td>33</td>
<td>A</td>
<td>OQ</td>
</tr>
</tbody>
</table>
Example

Given $R(ABCDE)$ having following FD’s $AB \rightarrow C$, $C \rightarrow D$, $B \rightarrow E$

$D = R1(ABC), R2(CDE)$ ?

$D = \{ ABCD, BE \}$ ?
When is a decomposition Lossy?

A Decomposition of $R = R_1, R_2$ is Lossless iff

$$R_1 \cap R_2 \to R_1$$

or

$$R_1 \cap R_2 \to R_2$$

$$\Rightarrow$$ Intersecting atts must form a super key for one of the resulting smaller relations
Given R(ABCDE) having following FD’s $AB \rightarrow C$, $C \rightarrow D$, $B \rightarrow F$

$D = \{ABCD, ABE\}$?

$D = \{ABC, CD, DE\}$?

$D = \{ABC, CD, BE\}$?
BCNF

Boyce-Codd Normal Form

A type of schema where there is:

1. No redundancy
2. Lossless joins
3. Iff Dependency Preserving: every dependency represented in closure is a superkey
4. How to find all F in the closure? Use the canonical cover!!

Pseudo Algorithm:

5. Split R on some FD X \rightarrow Y in F_c into R_i(X,Y)
6. R <- R - \{Y\}
7. F_c <- F_c - \{(X \rightarrow R)\}
8. Split R on another FD X_i \rightarrow Y_i in F_c into R_k(X_i,Y_i)
9. Repeat 2-4 until no more non-trivial FD X \rightarrow Y that is not a super key for some R_j
BCNF Example Dependency Preserving

R = (A, B, C, D)

F = {AB -> C, A -> D}

R_1 = (A, B, C)  
R_2 = (A, D)  
R_3 = (A, B)

Trivial! We don’t need it!
BCNF Example Not DP

\( R = (A, B, C, D) \)

\( F = \{AB \rightarrow C, C \rightarrow A, B \rightarrow D\} \)

\( R_1 = (A, B, C) \quad R_2 = (C, A) \quad R_3 = (B, D) \)

Co-dependent, so how could we decompose on these non-trial dependencies
BCNF Example Not DP

\[ R = (A, B, C, D) \]
\[ F = \{ AB \rightarrow C, C \rightarrow A, B \rightarrow D \} \]

\[ R_1 = (A, B, C) \quad R_2 = (B, D) \]

Yes, it is in BCNF!
But missing DP \( C \rightarrow A \)