Functional Dependencies
Describe “Key-Like” Relationships

A key is a set of attributes where:
If keys match, then the tuples match

A functional dependency (FD) is a generalization:
If an attribute set determines another, written $A \rightarrow B$
then if two tuples agree on attribute set $A$, they must agree on $B$:

\[ \text{id} \rightarrow \text{name} \]

- FDs are independent of our schema design choice

---

Example FDs

- Enrollment Information/Stuff
  - (\text{id, name, cid, subj, crnum, exp-grade})
    - sid: student id, cid is course id, .....

- Key is (\text{sid,cid})

- $\text{sid} \rightarrow \text{name}$
- $\text{cid} \rightarrow \text{crnum}$
- $\{\text{sid,cid}\} \rightarrow \text{exp-grade}$
- $\text{crnum} \rightarrow \text{subj}$
- $\text{cid} \rightarrow \text{subj}$
Formal Definition of FD’s

Given a relation schema R and subsets X, Y of R:
Given a relation schema R and subsets X, Y of R:

An instance r of R satisfies FD \( X \rightarrow Y \) if,

for any two tuples \( t_1, t_2 \in r \),

\[
\text{if } t_1[X] = t_2[X] \text{ implies } t_1[Y] = t_2[Y]
\]

if they have the same values in X attributes/columns, they have the same values in Y attributes/columns

- For an FD to hold for schema R, it must hold for every possible instance of r
  - (Can a DBMS verify this? Can we determine this by looking at an instance?)

What do FDs tell us..

- What is FD \( X \rightarrow Y \)
  - X and Y are “related”
  - If we know value of X then values of Y can be determined
  - X is a ‘unique’ attribute of the entity
  - ...

General Thoughts on Good Schemas

We want all attributes in every tuple to be determined only by the tuple’s key attributes, i.e. part of a superkey (for key $X \rightarrow Y$, a superkey is a “non-minimal” $X$).

What does this say about redundancy?

But:

- What about tuples that don’t have keys (other than the entire value)?

Sets of Functional Dependencies

- Relation EMP-DEPT($\text{SSN, NAME, ADDRESS, DNUMBER, DNAME, MGRSSN}$)
  - Employee info; the dept they are assigned to; their manager’s ssn
  - Key is SSN

- Some obvious functional dependencies
  - $\{\text{SSN}\} \rightarrow \{\text{NAME, ADDRESS, DNUMBER}\}$
  - $\{\text{DNUMBER}\} \rightarrow \{\text{DNAME, MGRSSN}\}$
Sets of Functional Dependencies

- Some obvious functional dependencies
  - \{SSN\} \rightarrow \{NAME, ADDRESS, DNUMBER\}
  - \{DNUMBER\} \rightarrow \{DNAME, MGRSSN\}
- From above dependencies, we can infer
  - \{SSN\} \rightarrow \{DNAME, MGRSSN\}
- Concept of a set of dependencies that can be inferred from the given set
  - Inference rules?
  - Closure: \( F^+ \) is all dependencies that can be inferred from \( F \)

Some Questions:

- Given a set of functional dependencies (properties on the data), what other properties can we infer?
- What is the formal definition of a key?
- How can we use the formal framework of Functional dependencies to define a ’good schema design’?
- Can we automate the process (develop algorithms)?
Armstrong’s Axioms: Inferring FDs

Some FDs exist due to others; can compute using Armstrong’s axioms:

- **Reflexivity:** If \( Y \subseteq X \) then \( X \rightarrow Y \) *(trivial dependencies)*
  - name, sid \( \rightarrow \) name

- **Augmentation:** If \( X \rightarrow Y \) then \( XW \rightarrow YW \)
  - cid \( \rightarrow \) subj so cid, exp-grade \( \rightarrow \) subj, exp-grade

- **Transitivity:** If \( X \rightarrow Y \) and \( Y \rightarrow Z \) then \( X \rightarrow Z \)
  - cid \( \rightarrow \) crnum and crnum \( \rightarrow \) subj
  - so cid \( \rightarrow \) subj

Armstrong’s Axioms Lead to…

- **Union:** If \( X \rightarrow Y \) and \( X \rightarrow Z \) then \( X \rightarrow YZ \)
- **Pseudotransitivity:** If \( X \rightarrow Y \) and \( WY \rightarrow Z \) then \( XW \rightarrow Z \)
- **Decomposition:** If \( X \rightarrow Y \) and \( Z \subseteq Y \) then \( X \rightarrow Z \)

Can prove these from Armstrong’s Axioms…

*homework ?? 😊*
Next question:

- Given a set $F$ of functional dependencies, what are all the properties we can infer?
- Do two sets of functional dependencies, $F$ and $G$, imply the same set of properties?
- How to formally define this property?

Closure of a Set of FD’s

Defn. Let $F$ be a set of FD’s. Its closure, $F^+$, is the set of all FD’s:

$\{X \rightarrow Y \mid X \rightarrow Y$ is derivable from $F$ by Armstrong’s Axioms$\}$

Which of the following are in the closure of our Student-Course FD’s?

- $\text{name} \rightarrow \text{name}$
- $\text{crnum} \rightarrow \text{subj}$
- $\text{cid} \rightarrow \text{subj}$
- $\text{crnum}, \text{sid} \rightarrow \text{subj}$
- $\text{crnum} \rightarrow \text{sid}$

$F=\{\text{sid} \rightarrow \text{name}, \text{cid} \rightarrow \text{crnum}, (\text{sid}, \text{cid}) \rightarrow \text{expgrade}, \text{crnum} \rightarrow \text{subject}\}$
**Why Armstrong’s Axioms?**

Why are Armstrong’s axioms (or an equivalent rule set) appropriate for FD’s? They are:

- **Consistent**: any relation satisfying FD’s in $F$ will satisfy those in $F^+$
- **Complete**: if an FD $X \rightarrow Y$ cannot be derived by Armstrong’s axioms from $F$, then there exists some relational instance satisfying $F$ but not $X \rightarrow Y$

- In other words, Armstrong’s axioms derive **all** the FD’s that should hold

---

**Equivalence of FD sets**

**Defn.** Two sets of FD’s, $F$ and $G$, are equivalent if their closures are equivalent, $F^+ = G^+$

e.g., these two sets are equivalent:

- $\{XY \rightarrow Z, X \rightarrow Y\}$ and
- $\{X \rightarrow Z, X \rightarrow Y\}$

- $F^+$ could contain a huge number of FD’s (exponential in the size of the schema?)
- Would like to have smallest “representative” FD set – the “cover” set for $F$
  - Why?
**Minimal Cover**

Defn. A FD set $F$ is *minimal* if:

1. Every FD in $F$ is of the form $X \rightarrow A$, where $A$ is a single attribute

2. For no $X \rightarrow A$ in $F$ is: $F - \{X \rightarrow A\}$ equivalent to $F$

3. For no $X \rightarrow A$ in $F$ and $Z \subseteq X$ is: $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ equivalent to $F$

Defn. $F$ is a *minimum cover* for $G$ if $F$ is minimal and is equivalent to $G$.

e.g.,

$$\{X \rightarrow Z, X \rightarrow Y\}$$ is a minimal cover for $$\{XY \rightarrow Z, X \rightarrow Y\}$$

---

**Minimal Cover**

- Let $F$ be:
  - $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow G$, $EF \rightarrow H$, $ACDF \rightarrow EG$

- First rewrite $ACDF \rightarrow EG$, so that every right hand side (RHS) is a single attribute:
  - $ACDF \rightarrow E$, $ACDF \rightarrow G$

- Consider $ACDF \rightarrow G$; is it implied by any dep?

- $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow G$
  - Therefore we can delete $ACDF \rightarrow G$

- Similarly can delete $ACDF \rightarrow E$

- Next consider $ABCD \rightarrow E$

- Since $A \rightarrow B$ holds we can replace with $ACD \rightarrow E$

- Thus, minimal cover for $F$ is:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$, $EF \rightarrow H$
Minimal Cover Algorithm

- Put the FDs of \( F \) in a standard form
  - Obtain collection \( G \) of equivalent FDs with single attribute on RHS (using decomposition axiom)
- Minimize the left hand side of each FD
  - For each FD in \( G \), check each attribute on LHS to see if it can be deleted while preserving equivalence to \( F^+ \)
- Delete redundant FDs
- There is a polynomial time algorithm to find min cover

Attribute Closures: Is Something Dependent on \( X \)?

Defn. The closure of an attribute set \( X \), \( X^+ \), is:

\[
X^+ = \bigcup \{Y \mid X \rightarrow Y \in F^+\}
\]

- This answers the question “is \( Y \) determined (transitively) by \( X \)?”;
  - *Given values for \( X \), the values are \( Y \) are fixed*

- *Does sid, cid \( \rightarrow \) subj, name, exp-grade?*
Computing Attribute Set Closure

- For attribute set $X$, compute closure $X^+$ by:

\[
\text{Closure } X^+ := X; \\
\text{repeat until no change in } X^+ \{ \\
\quad \text{if there is an } \text{FD } U \rightarrow V \text{ in } F \\
\quad \quad \text{such that } U \text{ is in } X^+ \\
\quad \quad \text{then add } V \text{ to } X^+ \}\]

Attribute Closure: Example

- Let $F$ be:
  - $SSN \rightarrow EName$
  - $PNUMBER \rightarrow PNAME, PLOCATION$
  - $SSN, PNUMBER \rightarrow HOURS$

  - What is the closure of \{SSN, PNUMBER\}
Attribute Set Closure and Keys

- If X is a key over relation scheme R, then what is $X^+$
  - Formal definition of a Key
- How to determine the keys for relation R?
  - R is a set of attributes $\{A_1, A_2, \ldots, A_n\}$
  - For each subset S of R, compute $S^+$
    - If $S^+ = R$ then S is Key
  - What is the “catch” here?
  - Can you improve this?

Example

- R= (C, T, H, R, S)
  - Course (C), Time (T), Hour (H), Room (R), Section (S), Grade (G)
  - $C \rightarrow T$  $CS \rightarrow G$
  - $HS \rightarrow R$  $HR \rightarrow C$
  - $HT \rightarrow R$

What is the smallest attribute set that must be part of the key?
Attribute Set Closures

- If attribute A does not appear on RHS of any FD, then any key must contain A
- If X is a key, then anything containing X is a superkey
- If X is a key, and Y → X is a FD then Y is a key

Relevance to schema design ?????
Schema Design:
Normal Forms, Functional Dependencies

Review

- Functional dependencies
- Closure of set of functional dependencies
  - Equivalence of FD sets
  - Minimal cover
- Armstrong’s Axioms
  - Guarantees Completeness and Consistency
- Attribute set closure algorithm
  - Can use to find all keys for a relation
Functional Dependencies and Schema Design: Normal Forms

- Normal forms are properties of relations
- We say a relation is in xNF if its attributes satisfy certain properties
  - Properties formally defined using functional dependencies
  - For example, test the relation to see if it is in 3NF
  - If not in 3NF, then change design...how?
    - Decomposition

How to go about designing a good schema?

- How to create a 3NF database schema? (i.e., a good design)?
- Ad-hoc approach
  - Create relations intuitively and hope for the best!
- Formal method – procedure (automated tool)
  - Start with single relation with all attributes
  - Systematically decompose relations that are not in the desired normal form
  - Repeat until all tables are in desired normal form
- Can decomposition create problems if we are not careful?
  - (i) Spurious tuples and (ii) lost dependencies
**Decomposition Problems**

S1: CAR (ID, Make, Color)

<table>
<thead>
<tr>
<th>ID</th>
<th>Make</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Toyota</td>
<td>Blue</td>
</tr>
</tbody>
</table>

S2: CAR1 (ID, Color)
CAR2 (Color, Make)

<table>
<thead>
<tr>
<th>ID</th>
<th>Color</th>
<th>Make</th>
</tr>
</thead>
<tbody>
<tr>
<td>456</td>
<td>Audi</td>
<td>Blue</td>
</tr>
<tr>
<td>789</td>
<td>Toyota</td>
<td>Red</td>
</tr>
</tbody>
</table>

What happens when we join CAR1 and CAR2?
Lossless Join Decomposition

$R_1, \ldots, R_k$ is a lossless join decomposition of $R$ w.r.t. an FD set $F$ if for every instance $r$ of $R$ that satisfies $F$,

$$\Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_k}(r) = r$$

Consider:

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>cid</th>
<th>subj</th>
<th>crnum</th>
<th>exp-grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sam</td>
<td>570103</td>
<td>SW</td>
<td>cs143</td>
<td>B</td>
</tr>
<tr>
<td>23</td>
<td>Dan</td>
<td>550103</td>
<td>DB</td>
<td>cs178</td>
<td>A</td>
</tr>
</tbody>
</table>

What if we decompose on
(sid, name) and (cid, subj, crnum, exp-grade)?
Testing for Lossless Join

R₁, R₂ is a lossless join decomposition of R with respect to F iff at least one of the following dependencies is in F⁺:

\[(R₁ \cap R₂) \rightarrow R₁ - R₂\]
\[(R₁ \cap R₂) \rightarrow R₂ - R₁\]

- Set of attributes common to the two tables are key to one of the two table.

So for the FD set:
- sid → name
- cid → crnum, exp-grade
- crnum → subj

Is (sid, name) and (crnum, subj, cid, exp-grade) a lossless decomposition?

Dependency Preservation

- Ensures we can “easily” check whether a FD X → Y is violated during an update to a database:
  - The projection of an FD set F onto a set of attributes Z, F_Z is:
    \[\{X \rightarrow Y \mid X \rightarrow Y \in F^+, X \cup Y \subseteq Z\}\]
    i.e., it is those FDs local to Z’s attributes
  - A decomposition R₁, …, Rₖ is dependency preserving if
    \[F^+ = (F_{R₁} \cup \ldots \cup F_{Rₖ})^+\]

- Why is this important/desirable?
- The decomposition hasn’t “lost” any essential FD’s, so we can check without doing a join
**Example of Lossless and Dependency-Preserving Decompositions**

Given relation scheme

\[ R(\text{name, street, city, st, zip, item, price}) \]

And FD set

- \( \text{name} \rightarrow \text{street, city} \)
- \( \text{street, city} \rightarrow \text{st} \)
- \( \text{street, city} \rightarrow \text{zip} \)
- \( \text{name, item} \rightarrow \text{price} \)

Consider the decomposition

\[ R_1(\text{name, street, city, st, zip}) \text{ and } R_2(\text{name, item, price}) \]

- Is it lossless?
- Is it dependency preserving?

What if we added FD \( \text{street, city} \rightarrow \text{item} \)?

---

**Dependency Preservation**

- Example:
  - FD set \( F = C \rightarrow \{\text{everything}\}, JP \rightarrow C, SD \rightarrow P, J \rightarrow S \)
  - Is decomposition of CSJDQV into SDP, JS and CJDQV dependency preserving

- It is a lossless join decomposition.
- In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  - JPC tuples stored only for checking FD! *(Redundancy!)*
FD's and Keys

- Ideally, we want a design s.t. for each nontrivial dependency $X \rightarrow Y$, $X$ is a superkey for some relation schema in $R$ and all dependencies are preserved
  - We just saw that this isn’t always possible
- What if a dependency is lost during decomposition, but we want to enforce the condition ?
  - Is there anything in SQL that can help us enforce this dependency condition?

Two Important Normal Forms

**Boyce-Codd Normal Form** (BCNF). For every relation scheme $R$ and for every $X \rightarrow A$ that holds over $R$,

- either $A \in X$ (it is trivial), or
- $X$ is a superkey for $R$

**Third Normal Form** (3NF). For every relation scheme $R$ and for every $X \rightarrow A$ that holds over $R$,

- either $A \in X$ (it is trivial), or
- $X$ is a superkey for $R$, or
- $A$ is a member of some key for $R$
Why these normal forms?

- What problems/anamolies exist?
- What problems are “removed” by a normal form?

- Start with Second Normal Form
  - Not used today
- Let’s work with a simplified model first
  - Called Primary Key version

Definitions

- Relation schema R
  - Superkey
  - Key
  - Candidate key – same as key
  - Primary key – a key designated for common use
  - Prime attribute – an attribute that belongs to some candidate key
  - Non-prime attribute – does not belong to any key
1NF and 2NF

- First Normal Form – assumed as condition
  - Value of any attribute is a single value
  - Domains of attribute contain only atomic values
    - Cannot be sets of values

Partial Dependency

- A FD $X \rightarrow Y$ is a partial dependency if there exists an attribute $A \in X$ such that $X - A \rightarrow Y$
  - Y is partially dependent on X
- Second Normal Form: Relation is in 2NF if no non-prime attribute is partially dependent on the primary key.
Problems with Partial Dependency

EMP_PROJ( SSN, PNUMBER, HOURS, ENAME, 
            PNAME, PLOCATION)

Some FDs:
{SSN, PNUMBER} → HOURS
{SSN, PNUMBER} → ENAME

Since SSN → ENAME, ENAME is partially dependent on the 
primary key {SSN, PNUMBER}
• So why is this a problem?

- Insert tuple
  • <987654321, 3, 12, Jones, Sprite, Atlanta>
- We have insertion anomaly
  • Check if 987654321 is Jones, project 3 is Sprite…
- We have deletion problem
  • If last tuple with Project #1 is deleted

- Similarly, we have modification anamoly
  • Smith changes name to Brown
Transitive Dependencies & 3NF

- FD $X \rightarrow Y$ is a transitive dependency in relation R if there exists set of attributes $Z \in R$ such that
  - $X \rightarrow Z$ and $Z \rightarrow Y$
  - $Z$ is not a subset of any key of R
- A relation R is in Third Normal Form if (1) it is in 2NF and (2) no non-prime attribute is transitorily dependent on any key.

Problem with Transitive Dependencies

- EMP_DEPT(ENAME, SSN, BDATE, ADDRESS, DNO, DNAME, MGRSSN)
  FDs in relation:
  - $\{SSN\} \rightarrow \{DNO\}$
  - $\{DNO\} \rightarrow \{MGRSSN\}$
  - $\{DNO\} \rightarrow \{DNAME\}$
- Insertion, Deletion, Modification anomalies in above schema
**Problem with 3NF?**

- ADDR_INFO( CITY, ADDRESS, ZIP)
  \{CITY, ADDRESS\} → ZIP
  \{ZIP\} → \{CITY\}

Possible keys: \{CITY, ADDRESS\} or
\{ADDRESS, ZIP\}
Is it in 3NF?

---

**Problems with the 3NF schema**

- Delete <Washington, 801 22nd St, 20052>
- What if this is the last 20052 tuple ?
  - We lose the info that 20052 is in Washington
  - We also have insert, modify anomalies
- Why the problem ?
  - Dependencies from an attribute to part of a key
- Solution?
  - Make all LHS of dependencies be key or superkey!
  - BCNF – Boyce Codd Normal Form: if all FDs are of the form X → Y where X is superkey.
General Definition of 3NF, BCNF

- Can simplify the 3NF definition to remove the reference to partial dependencies/2NF
- R is in 3NF if for every FD $X \rightarrow Y$, either
  - $X$ is a superkey or
  - $Y$ is a prime attribute
- R is in BCNF if for every FD $X \rightarrow Y$, $X$ is a superkey
  - R in BCNF $\Rightarrow$ R is in 3NF

Normal Forms Compared

- BCNF is preferable, but sometimes in conflict with the goal of dependency preservation
  - It’s strictly stronger than 3NF
- Let’s see algorithms to obtain:
  - A BCNF lossless join decomposition
  - A 3NF lossless join, dependency preserving decomposition
    - Read this on your own from the textbook
**BCNF Decomposition Algorithm**

Input: Relation R (consisting of all attributes), set of functional dependencies F
Output: BCNF schema result

\[ result := \{R\} \]

while there is a schema \( R_i \) in result that is not in BCNF

\{ 

let \( A \rightarrow B \) be a FD that violates BCNF in relation \( R_i \)

\[ result := (result - R_i) \cup \{(R_i - B), (A,B)\} \]

\}

---

**Example 1**

- **R** = \((C,T,H,R,S)\)
  - Course (C), Time (T), Hour (H), Room (R), Section (S), Grade (G)
  - \( C \rightarrow T \)
  - \( CS \rightarrow G \)
  - \( HS \rightarrow R \)
  - \( HR \rightarrow C \)
  - \( HT \rightarrow R \)

Key = \{HS\}
Prime attributes = \{H,S\}
Testing for 3NF, BCNF

- Is the schema in BCNF?
  - Check if there are non-BCNF dependencies

- Is the schema in 3NF?
  - Check if there are non-3NF dependencies
    - Is there a dependency to non-prime attribute from something that is not a key?

Normalization Procedure: Summary

- Input= (Set of dependencies F, Set of attributes - single table schema)
  1. Use attribute set closure algo to find (a) keys and (b) prime attributes
    - Prune the search using the various “tricks”
  2. Test each FD in F to see if it satisfies 3NF/BCNF properties
  3. Decompose into smaller relations using decomposition algorithm
  4. If BCNF is not dependency preserving, then go with a 3NF decomposition
Decomposition into BCNF

- Consider relation R with FDs F. If X → Y violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C (C → everything), JP → C, SD → P, J → S
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJDQV into JS and CJDQV

- In general, several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.

- Example: decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (Redundancy!)
Example 2:

- \( R = (A, B, C, D, E) \)
- \( A \rightarrow BC \)
- \( CD \rightarrow E \)
- \( B \rightarrow D \)
- \( E \rightarrow A \)

Summary

- We can always decompose into 3NF and get:
  - Lossless join
  - Dependency preservation
- But with BCNF we are only guaranteed lossless joins
- BCNF is stronger than 3NF: every BCNF schema is also in 3NF
- The BCNF algorithm is nondeterministic, so there is not a unique decomposition for a given schema \( R \)
Some Complexity results

- Testing if schema is in 3NF is NP-complete
- Testing for non-BCNF FD is NP-complete
- Finding all keys is NP-complete
- Polynomial time algorithms for computing all minimal covers
- Polynomial time algorithm to compute a lossless-join dependency preserving 3NF relation schema

So how do you design a schema from ground up?

- Go over the application specifications
  - Can represent as ER diagram
- Identify all required information
  - This will constitute your “data”/attributes
- Identify the ‘business rules’
  - This will define the functional dependencies
  - Will also define some of your application logic
  - This step may require you to “interact” with the “client” to clarify your questions
- Apply decomposition algorithm to get a good schema
- Now you are ready to start developing the application
  - Writing SQL queries
  - Writing the PHP/interface code