Design Theory for Relational Databases

Thomas Schwarz, SJ

Contents

- There are many ways a database scheme can be constructed
  - A poorly designed scheme:
    - Has problems with checking constraints
    - Has problems with data coherence
      - E.g. two different spellings of the same person's first name
    - Has problems with performance

Contents

- Design theory helps to design efficient schemes
  - Functional dependencies
    - Used in the definition of a key
    - Used for flagging potentially bad records
  - Normal forms
    - Get rid of anomalies
    - Get rid of redundant storage of data
  - Expand to multivalued dependencies

Functional Dependencies

- About the nature of data
  - Think about it as the potential contents of a table
  - Instead of the actual contents
- Example:
  - Students can have a double major
  - But an actual set of students might not include a student with double major

Functional Dependencies

- A form of constraint for a relation
  - **Functional Dependency** (FD) for table $R(\mathbb{X})$
    - FD $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots, B_m$
      - with $A_1, \ldots, A_n, B_1, \ldots, B_m \in \mathbb{X}$
    - If a tuple's values agree for attributes $A_1, \ldots, A_n$
    - Then they agree for attributes $B_1, B_2, \ldots, B_m$

Functional Dependencies

- Only consider FD with one attribute on the right
  - Because FD $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots, B_m$ is equivalent to all of:
    - $A_1, A_2, \ldots A_n \longrightarrow B_1$
    - $A_1, A_2, \ldots A_n \longrightarrow B_2$
    - $\vdots$
    - $A_1, A_2, \ldots A_n \longrightarrow B_m$

Functional Dependencies

- Example:
  - `Movies1( title, year, length, genre, studioName, starName)`
- Find all FDs

Functional Dependencies

- `title, year -> length`
- `title, year -> genre`
- `title, year -> studio`
- However:
  - `title, year` $\nrightarrow$ `starName`
    - is not an FD

Keys

- A **superkey** is a set of attributes in a table that determines all attributes
  - $R(A_1, A_2, \ldots A_n)$
    - $A_{i_1}, A_{i_2}, \ldots A_{i_m}$ is a superkey if
      - $\forall j : \quad A_{i_1}, A_{i_2}, \ldots A_{i_m} \longrightarrow A_j$

Keys

- A key is a minimal superkey with respect to set inclusion
  - I.e. A superkey so that no attribute in it can be reomved

- If a key consists of a single attribute, then we call the attribute the key instead of the set with only element this attribute

Functional Dependencies

- Quiz: Given $R(A, B, C)$ and FDs $A \rightarrow B$ and $B \rightarrow C$,
- Does this mean $A \rightarrow C$?

Functional Dependencies

- Answer: Yes.
- Show that all tuples that agree on attribute A also agree on attribute C
    - Called transitivity

Functional Dependencies

- A set S of FDs follows from a set T of FDs if every relation instance satisfying all FDs in T also satisfies all FDs in S
- Sets of FDs are equivalent if the set of relation instances satisfying one is equal to the set of relation instances satisfying the other one.

Functional Dependencies

- The splitting rule:
  - $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots, B_m$ is equivalent to
    - $A_1, A_2, \ldots A_n \longrightarrow B_1$
    - $A_1, A_2, \ldots A_n \longrightarrow B_2$
    - $\vdots$
    - $A_1, A_2, \ldots A_n \longrightarrow B_m$

Functional Dependencies

- The combining rule:
  - The set of FDs
    - $A_1, A_2, \ldots A_n \longrightarrow B_1$
    - $A_1, A_2, \ldots A_n \longrightarrow B_2$
    - $\vdots$
    - $A_1, A_2, \ldots A_n \longrightarrow B_m$
  - is equivalent to $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots, B_m$

Functional Dependencies

- Quiz: Does $A_1, A_2, \ldots, A_n \longrightarrow B$ imply $X, A_1, A_2, \ldots, A_n \longrightarrow B$?

Functional Dependencies

- Quiz: Does $A_1, A_2, \ldots, A_n \longrightarrow B$ imply $X, A_1, A_2, \ldots, A_n \longrightarrow B$?
- Yes:
  - Called augmentation

Functional Dependencies

- Trivial FDs
  - $A_i \to A_i$ (Reflexivity)
  - $A, B \to A$ (Reflexivity & Augmentation)
- Trivial Dependency Rule:
  - $A_1A_2 \ldots A_n \to B_1B_2 \ldots B_m$ is equivalent to
    - $A_1A_2 \ldots A_n \to C_1C_2 \ldots C_r$
      - where the C_i are those of the B_i that are not among the A_i

Functional Dependencies

- Closure:
  - Let $\mathbb{S}$ be a set of functional dependencies
  - Let $\mathbb{A} = \{A_1, A_2, \ldots, A_n\}$ be a set of attributes
  - The **<u>closure</u>** of $\mathbb{A}$ is the set $\mathbb{A}^+$ of attributes B such that every relation that satisfies all the FDs in $\mathbb{S}$ also satisfies $A_1, A_2, \ldots, A_n \rightarrow B$.

Functional Dependencies

- Closure calculation algorithm
  - Input: a set of attributes $\mathbb{A}$ and a set of functional dependencies $\mathbb{S}$.
  - Output: $\mathbb{A}^+$

1. Split all FDs in $\mathbb{S}$ so that there is only a single attribute on the right
2. Set $\mathbb{X}$ to be $\mathbb{A}$.
3. Repeatedly search for some FD $B_1, B_2, \ldots, B_m \rightarrow C \in \mathbb{S}$ such that $B_1, B_2, \ldots, B_m \in \mathbb{A}$ and $C \notin \mathbb{A}$. Then add C to $\mathbb{X}$
4. Stop when the search fails and output $\mathbb{X} = \mathbb{A}^+$.

Functional Dependencies

- Consider the relation scheme R = {E, F, G, H, I, J, K, L, M} and the set of functional dependencies {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N} on R. What is the key for R?:
  - {E}
  - {E,F}
  - {E,F,H}
  - {E,F,H,K,L}
    - Hint: calculate the closure of all possible answers

Functional Dependencies

- First, normalize {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N}}
  - $\{\{E, F\} \to G, \{F\} \to I, \{F\} \to J, \{E, H\} \to K, \{E, H\} \to L, K \to M, L \to N\}$
  - Start with $\{E\}$.
    - There is no FD that has only E on the left side
    - $\{E\}^+ = \{E\}$

Functional Dependencies

- $\{\{E, F\} \rightarrow G, \{F\} \rightarrow I, \{F\} \rightarrow J, \{E, H\} \rightarrow K, \{E, H\} \rightarrow L, K \rightarrow M, L \rightarrow N\}$
  - Now try $\{E, F\} = \mathbb{X}$
  - We can add G to $\mathbb{X}$.
  - We can add I to $\mathbb{X}$.
  - We can add J to $\mathbb{X}$.
  - Then we are stuck: $\{E, F\}^+ = \{E, F, G, I, J\}$

Functional Dependencies

- $\{\{E, F\} \rightarrow G, \{F\} \rightarrow I, \{F\} \rightarrow J, \{E, H\} \rightarrow K, \{E, H\} \rightarrow L, K \rightarrow M, L \rightarrow N\}$
  - Now try $\{E, F, H\} = \mathbb{X}$
  - We can add G to $\mathbb{X}$ because of (1).
  - We can add I to $\mathbb{X}$ because of (2): $\mathbb{X} = \{E, F, G, H, I\}$
  - We can add J to $\mathbb{X}$ because of (3): $\mathbb{X} = \{E, F, G, H, I, J\}$
  - (4) gives $\mathbb{X} = \{E, F, G, H, I, J, K\}$
  - (5) gives $\mathbb{X} = \{E, F, G, H, I, J, K, L\}$
  - (6) gives $\mathbb{X} = \{E, F, G, H, I, J, K, L, M\}$
  - (7) gives $\mathbb{X} = \{E, F, G, H, I, J, K, L, M, N\}$
- Therefore $\{E, F, H\}^+$ contains all the attributes.
  - Since $\{F, H\}^+ = \{F, H, I, J\}$, $\{E, F, H\}$ is a minimal candidate key and therefore a key.

Functional Dependencies

- Why does closure work
  - Need to show equivalency of :
    - $B \in \{A_1, A_2, \ldots, A_n\}^+$ with regards to $\mathbb{S}$
    - Every relation fulfilling $\mathbb{S}$ fulfills $A_1 A_2 \ldots A_n \to B$

Functional Dependencies

- Why does
    - $B \in \{A_1, A_2, \ldots, A_n\}^+$ with regards to $\mathbb{S}$
  - imply
    - Every relation fulfilling $\mathbb{S}$ fulfills $A_1A_2\ldots A_n \to B$

- Look at the first time adding an attribute to $\mathbb{X}$ leads to an FD $A_1A_2\ldots A_n \to B$ that is **not** true.
  - But B was added using a FD $X_1, X_2, \ldots X_m \to B$
  - Because this is the first time and $X_1, X_2, \ldots X_m$ follow from the $A_1A_2\ldots A_n$ in all relations, $A_1A_2\ldots A_n \to X_1, \ldots X_m$
  - Thus, a tuple equal in $A_1A_2\ldots A_n$ is also equal in all $X_1, \ldots X_m$ and hence equal in B.
  - Therefore $A_1A_2\ldots A_n \to B$ has to be true and we have a contradiction

Functional Dependencies

- Why does
    - Every relation fulfilling $\mathbb{S}$ fulfills $A_1A_2\ldots A_n \to B$
  - imply
    - $B \in \{A_1, A_2, \ldots, A_n\}^+$ with regards to $\mathbb{S}$

- Assume $B \notin \{A_1, A_2, \ldots, A_n\}^+$ with regards to $\mathbb{S}$, but $A_1A_2\ldots A_n \to B$ holds in all relations that also fulfill $\mathbb{S}$.
  - Create a simple table:
    - ```
      {A1 A2 ... An}+  every thing else
       0   0       0      0    0   ... 0
       0   0       0      1    1   ... 1
      ```
      ```

Functional Dependencies

- Does this instance satisfy $\mathbb{S}$?
  - Assume an FD $C_1C_2\ldots C_r \to D$ in $\mathbb{S}$ is violated
    - For a violation to occur, the C_i need to be on the left side, i.e. in $\{A_1, A_2, \ldots, A_n\}^+$ and the D on the right side of the table.

```
{A1 A2 ... An}+  every thing else
 0  0      0      0   0  ... 0
 0  0      0      1   1  ... 1
```

    - But then we did not calculate the closure correctly and D should have been in $\{A_1, A_2, \ldots, A_n\}^+$

Functional Dependencies

- Does this instance not satisfy $A_1A_2\ldots A_n \rightarrow B$

```
{A1  A2  ...  An}+   every thing else
 0   0         0       0    0   ... 0
 0   0         0       1    1   ... 1
```

- Yes!

Functional Dependencies

- Therefore the assumption is violated and this finishes the proof

Functional Dependencies

- With the closure calculation, we can prove
  - If in a relation R $A_1, A_2, \ldots, A_m \to B_1, B_2, \ldots, B_n$ and $B_1, B_2, \ldots, B_n \to C_1, C_2, \ldots C_t$ then $A_1, A_2, \ldots, A_m \to C_1, C_2, \ldots C_t$
  - Transitivity

Functional Dependencies

- We sometimes have a choice in the minimal set of FDs that describe a relation
  - A set of FD is called a <u>*basis*</u> if all FDs holding in the relation can be derived from the basis
  - A <u>*minimal basis*</u> $\mathbb{B}$:
    - All FDs in $\mathbb{B}$ have singleton right sides
    - Removing any FD from $\mathbb{B}$ is no longer a basis
    - If in any FD from $\mathbb{B}$ we drop an attribute from the right side, then the result is no longer a basis

Functional Dependencies

- Example:
  - A relation with three attributes such that each attribute determines the other attributes
  - What are the FDs?
  - Find a minimal basis

Functional Dependencies

- Answer: FDs are
  - $A \rightarrow B, A \rightarrow C$ and all augmentations $A \rightarrow B, C$ including the trivial ones $A \rightarrow A, B, A \rightarrow A, C$ and $A \rightarrow A, B, C$
  - $B \rightarrow A, \quad B \rightarrow C$ plus all augmentation
  - $C \rightarrow A, \; C \rightarrow B$ plus all augmentations

Functional Dependencies

- Answer: To obtain a bases, we can look at all subsets of right side singleton
  - $\{A \to B, A \to C, B \to A, B \to C, C \to A, C \to B\}$
  - For example:
    - We try to remove from left
    - $A \to B$ follows from $A \to C \quad \& \quad C \to B$
    - Left with
      $\{A \to C, B \to A, B \to C, C \to A, C \to B\}$

Functional Dependencies

- Left with $\{A \to C, B \to A, B \to C, C \to A, C \to B\}$
  - Now can get rid of $B \to A$
- Left with $\{A \to C, B \to C, C \to A, C \to B\}$

Functional Dependencies

- Another possibility:
  - $\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

Functional Dependencies

- Projecting Functional Dependencies
  - Given a relation R with a set of FDs $\mathbb{S}$ and a subset L of attributes of R:
    - What are the FDs induced in $\pi_L(R)$?
  - FDs can only involve attributes from L
  - But restricting $\mathbb{S}$ to those is not enough

Functional Dependencies

- Algorithm:
  - Start out with an empty set $\mathbb{T}$ of FDs
  - For each set M of attributes $\subset L$ calculate the closure M^+ in R
    - If $M \rightarrow X$ is a FD calculated this way and $X \in L$, add the FD to $\mathbb{T}$
  - Modify $\mathbb{T}$ to become a minimal basis
    - Remove all FDs that follow from others in $\mathbb{T}$
    - Test whether an attribute on the left of a FD in $\mathbb{T}$ can be removed

Functional Dependencies

- Example: $R(A, B, C, D)$ with $\mathbb{S} = \{A \to B, B \to C, C \to D\}$ projected on $L = \{A, C, D\}$
- Calculate first closures
  - $\{A\}^+ = \{A, B, C, D\}$
  - $\{B\}^+ = \{B, C, D\}$
  - $\{C\}^+ = \{C, D\}$
  - $\{D\}^+ = \{D\}$

Functional Dependencies

- We really do not need any more because those with two attributes on the left would follow trivially
  - Now we add the FDs derived from the closure, if all attributes are in L
    - $\mathbb{T} = \{A \to C, A \to D, C \to D\}$
  - This is not a base, because $A \to D$ follows from the other ones.
- The induced FDs have base $\mathbb{T} = \{A \to C, C \to D\}$

Anomalies

- Take

`movies = (title, year, length, genre, studioName, starName)`

- ***Redundancy*** : The studioName for Star Wars is repeated for every star
    - This implies:
- ***Update anomaly*** : If we update the length of the movie, we need to repeat this update operation for every star or we get incoherent information
- ***Delete anomaly*** : If we delete all stars from an animation cartoon, we have no information left on the movie!

Decomposition

- Divide the information over two tables

```
movies = (title, year, length, genre, studioName, starName)
```

- becomes

```
movies1=(title, year, length, genre, studioName)
movies2=(title, year, starName)
```

Boyce Codd Normal Form

- Relation in BCNF if and only if:
    - Whenever there is a non-trivial FD $A_1 \ldots A_n \rightarrow B$ then $A_1 \ldots A_n$ is a superkey

Boyce Codd Normal Form

- Example
  - `movies1(title, year, length, genre, studio, star)`
  - Has FD `title, year --> studio`
    - but because of the star attribute, `title, year` is not a key.
  - We can decompose:
    - Take the left side of the FD
    - Calculate its closure
      - `{title, year}+ = {title, year, length, genre, studio}`
      - Decompose into closure and right side
      - movies(<u>title</u>, <u>year</u>, length, genre, studio)
        starsIn(title, year, star)

Boyce Codd Normal Form

- What is good about BCNF?
  - Update anomaly
    - Decomposition prevents having to enter the same information multiple times
  - Delete anomaly
    - Can now have movies without stars
  - Can we do better?
    - Yes, sometimes. starsIn has still a two-attribute key

Boyce Codd Normal Form

- Any two attribute table $R(A, B)$ is in BCNF
  - Proof by case distinction:
    - Case 1: $A \nrightarrow B, \qquad B \nrightarrow A$
      - No nontrivial FDs exists, R is in BCNF
    - Case 2: $A \rightarrow B, B \nrightarrow A$
      - A is the only key and it is on the right of the only non-trivial FD. So BNCF.
    - Case 3: $A \nrightarrow B, B \rightarrow B$
      - Same as before
    - Case 4: $A \rightarrow B, \quad B \rightarrow A$
      - Both A, B are keys. So, BCNF

Boyce Codd Normal Form

- Decomposition:
  - Does decomposition loose information or add spurious information?
  - Does decomposition preserve dependencies
  - How do we do decomposition

Boyce Codd Normal Form

- Finding decompositions
  - Look for a non-trivial FD.
    - If the right side is not a superkey:
    - Expand the right side as much as possible
      - $A_1A_2\ldots A_n \to B_1\ldots B_m$
      - Right side are <u>all</u> attributes that are dependent on $A_1\ldots A_n$

Boyce Codd Normal Form

- Example:
  - `prod(title, year, studio, president, presAddr)`
  - with FD
    ```
    title year -->studio
    studio --> president
    president --> presAddr
    ```
  - Question: What are possible keys?

Boyce Codd Normal Form

- Only key is `title, year`
  - Just look at the closures of all subsets of attributes

- Which FDs violate BCNF?

Boyce Codd Normal Form

- Two FDs:
  - `studio --> president`
  - `president --> presAddr`

- What happens with `studio --> president`

Boyce Codd Normal Form

- We calculate the closure of the right side
  - `studio -> president`
  - `{studio}+ = {president, presAddr}`
  - This gives a decomposition
    - `(title, year, studio)  (studio, president, presAddr)`
  - Using projection of FDs, we get
    - `title, year -> studio`
    - `studio -> president, president -> presAddr`
      - so second relation is not in BCNF (studio is the only key)

Boyce Codd Normal Form

- Now we decompose the second relation again:
  - `(studio, president)`

    `(president, presAddr)`

Boyce Codd Normal Form

- Decomposition algorithm
  - If there is an FD $X \rightarrow Y$ that violates BCNF
    - Calculate X^+
    - Choose X^+ as one relation and $X \cup \complement(X^+)$ as the other
      - All attributes in X and all attributes not in X^+
    - Calculate the projected FDs
    - Continue

Boyce Codd Normal Form

- In class exercise.
  - Find all BNCF violations (including those following from the FDs given)
  - Decompose the relation, if possible
  - $R(A, B, C, D);\ AB \rightarrow C;\ C \rightarrow D;\ D \rightarrow A$

Answer

- $R(A, B, C, D)$; $AB \to C$; $C \to D$; $D \to A$
- Keys are (A,B), (C,B), (D,B}
- $C \to D$ violates Boyce Codd
  - $\{C\}^+ = \{C, D, A\}$
  - $X \cup \complement(X^+)$ = $\{C, B\}$
    - We find that $D \to A$ is still a violation is R(A,C,D).
      - $\{D\}^+ = \{D, A\}$
      - Complement is $\{D, C\}$

Boyce Codd Normal Form

- In class exercise.
  - Find all BNCF violations (including those following from the FDs given)
  - Decompose the relation, if possible

$$R(A, B, C, D);\ \ AB \to C;\ \ BC \to D;\ \ CD \to A;\ \ AD \to B$$

Boyce Codd Normal Form

- In class exercise.
  - Find all BNCF violations (including those following from the FDs given)
  - Decompose the relation, if possible

$$R(A, B, C, D);\ \ AB \rightarrow C;\ \ BC \rightarrow D;\ \ CD \rightarrow A;\ \ AD \rightarrow B$$

Decomposition

- Recovering data from decomposition
  - Assume a relation $R(A, B, C)$ with FD $B \to C$, where B is not a key
    - Decomposition is then $R_1(A, B)$ and $R_2(B, C)$
  - Assume $t = (a, b, c)$ is a tuple. It is projected as $t_1 = (a, b)$ and $t_2 = (b, c)$
    - Thus, $t \in R_1 \bowtie R_2$.
  - Assume $t_1 = (a, b) \in R_1$ and $t_2 = (b, c) \in R_2$, i.e. $t \in R_1 \bowtie R_2$
    - There is a tuple $(a, b, x) \in R$ because R_1 is a projection.
    - (Similarly, there is a tuple $(a, y, c) \in R$.)
    - Because of the FD $B \to C$ there is only one value for x
    - Hence, the tuple must have been $(a, b, x = c)$

Decomposition

- This argument generalizes to sets A, B
  - This means: Boyce Codd decomposition is recoverable
  - Since natural joins are associative and commutative, the BCNF decomposition algorithm cannot loose information

Decomposition

- Dependency preservation
  - Assume a table

    `bookings(title, theater, city)`

  - FDs `theater --> city`

    `title, city --> theater`

  - Keys are: `title, city` and `title, theater`

Decomposition

- The existence of the FDs is important
  - Assume a similar decomposition of $R(A, B, C)$ but without the FDs $B \to A, \quad B \to C$
  - Example instance:

A	B	C
1	2	3
4	2	5

  - Split into $R_1(A, B)$ and $R_2(B, C)$

Decomposition

- Result of projection

A	B	C
1	2	3
4	2	5

A	B
1	2
4	2

B	C
2	3
2	5

- What is the join of the two tables on the right?

Decomposition

- Result

A	B
1	2
4	2

B	C
2	3
2	5

A	B	C
1	2	3
4	2	3
1	2	5
4	2	5

- which introduces spurious records.
- Of course, attribute B was not a key for the second relation!

Dependency Preservation

- Decompose into BCNF
  - `(theater, city)` `(theater, title)`
  - Must be BCNF, because it only has two attributes
  - However, FD `title, city -> theater` cannot be derived

Decomposition

- Example:

Theater	City
AMC	Wauwatosa
Marcus 1	Milwaukee
Marcus 2	Wauwatosa

Theater	Title
Marcus 2	Doolittle
AMC	Doolittle

- Violates the FD
  - `title, city --> theater`

Chase Test

- We just saw: $R(A, B, C)$ with FD $B \to C$ has a lossless join into $R(A, B)$ and $R(B, C)$
- Without FD $B \to C$ or $B \to A$, the join is not loss-less

- Question: Given a set of FDs in R and a set of sets of attributes $S_1, S_2, \ldots S_n$:
  - Is decomposition by projection onto the S_i lossless?
  - i.e.: is $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \ldots \bowtie \pi_{S_n}(R) = R$?

Chase Test

- Two easy remarks:
  - Natural join is associative and commutative. The order in which we project is not important.
  - Certainly $R \subset \pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \ldots \bowtie \pi_{S_n}(R)$

Chase Test

- Chase Test:
  - **Task:** Show that given the FDs, we can prove that
    - $\pi_{S_1}(R) \bowtie \pi_{S_2}(R) \bowtie \ldots \bowtie \pi_{S_n}(R) \subset R$
  - Take a tuple $t \in R$
    - Use a *tableau* to determine the various versions this tuple could appear in the projections

Chase Test

- Tableau has one row for each decomposition
  - Put down unsubscripted letters for the attributes in the decomposed relationship
  - Put down subscripted letters for the attributes not in the decomposed relationship
    - Subscript is the number of the decomposed relationship

Chase Test

- Example: $R(A, B, C, D)$ with projections on $S_1 = \{A, D\}, S_2 = \{A, C\}$ and $S_3 = \{B, C, D\}$
  - A generic tuple in $S_1 \bowtie S_2 \bowtie S_3$ is then represented in the decomposition tableau

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

Chase Test

- The first row looks at the projection on A and D
  - From the projection, we know that a given tuple has certain *a* and *d* values, but the join might give some values for the b and c column

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

Chase Test

- Once given a tableau, we use the FDs in order to “chase down” identities between the elements in the tableau.
- We represent them by making subscripts equal or dropping them

Chase Test

- Example:
  - Assume the following FDs for the example:
    - $A \rightarrow B, B \rightarrow C, CD \rightarrow A$
  - Whenever we have tableau entries for attributes on the right side, we can use it to equalize the entries for attributes on the right of an FD

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

Chase Test

- Use $A \to B$:
  - First two rows, we have unsubscripted a.
    - Equalize the B column in these rows

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

Chase Test

- Use FD $B \to C$

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

Chase Test

- Now use $CD \rightarrow A$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

$\Rightarrow$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Chase Test

- Now we have one row that is equal to t
  - This means: any tuple of the join **has** to be equal to the original tuple

Chase Test

- What happens if after applying all FDs, we still are left with unsubscripted variables?
    - Then this gives us a value in the join that is not in the original relation

Chase Test

- Example:
  - $R(A, B, C, D)$ with FDs $B \to AD$
    - $\{B\}^+ = \{B, A, D\}$, so Boyce-Codd would split into
    - $R(B, C)$ and $R(A, B, D)$
  - But we decompose into $\{A, B\}, \{B, C\}, \{C, D\}$

Chase Test

- Example:
  - $R(A, B, C, D)$ with FDs $B \rightarrow AD$ and decomposition into $\{A, B\}, \{B, C\}, \{C, D\}$
  - Initial tableau is

A	B	C	D
a	b	c_1	d_1
a_2	b	c	d_2
a_3	b_3	c	d

Chase Test

- Example:
  - $R(A, B, C, D)$ with FDs $B \rightarrow AD$ and decomposition into $\{A, B\}, \{B, C\}, \{C, D\}$
  - Initial tableau is

A	B	C	D
a	b	c_1	d_1
a_2	b	c	d_2
a_3	b_3	c	d

  - After applying the FD, we get tableau

A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a_3	b_3	c	d

Chase Test

- Take this tableau and use it to construct a counter example
A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a_3	b_3	c	d
- Create tuples (a, b, c_1, d_1), (a, b, c, d_1), (a_3, b_3, c, d) in R.
- Fulfills the FD $B \to CD$
- Projections are

A	B
a	b
a_3	b_3

B	C
b	c_1
b	c
b_3	c

C	D
c_1	d_1
c	d_1
c	d

Chase Test

- Join these together:

A	B
a	b
a_3	b_3

B	C
b	c_1
b	c
b_3	c

C	D
c_1	d_1
c	d_1
c	d

- Result has two additional rows

A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a	b	c	d
a_3	b_3	c	d_1
a_3	b_3	c	d

- The decomposition is not loss-less!

Example

- Let $R(A, B, C, D, E)$ be decomposed into $\{A, B, C\}$, $\{B, C, D\}$, $\{A, C, E\}$. Assume FDs $A \to D$, $CD \to E$, $E \to D$. Is the decomposition lossless?

Example

- Let $R(A, B, C, D, E)$ be decomposed into $\{A, B, C\}$, $\{B, C, D\}$, $\{A, C, E\}$. Assume FDs $A \to D$, $CD \to E$, $E \to D$. Is the decomposition lossless?

A	B	C	D	E
a	b	c	d_1	e_1
a_2	b	c	d	e_2
a	b_3	c	d_3	e

Example

- Let $R(A, B, C, D, E)$ be decomposed into $\{A, B, C\}$, $\{B, C, D\}$, $\{A, C, E\}$. Assume FDs $A \rightarrow D$, $CD \rightarrow E$, $E \rightarrow D$. Is the decomposition lossless?
    - Use FD $A \rightarrow D$

A	B	C	D	E
a	b	c	d_1	e_1
a_2	b	c	d	e_2
a	b_3	c	d_3	e

$\Rightarrow$

A	B	C	D	E
a	b	c	d_1	e_1
a_2	b	c	d	e_2
a	b_3	c	d_1	e

Example

- Let $R(A, B, C, D, E)$ be decomposed into $\{A, B, C\}$, $\{B, C, D\}$, $\{A, C, E\}$. Assume FDs $A \to D$, $CD \to E$, $E \to D$. Is the decomposition lossless?
  - Use FD $CD \to E$

A	B	C	D	E
a	b	c	d_1	e_1
a_2	b	c	d	e_2
a	b_3	c	d_1	e

A	B	C	D	E
a	b	c	d_1	e
a_2	b	c	d	e_2
a	b_3	c	d_1	e

Example

- Let $R(A, B, C, D, E)$ be decomposed into $\{A, B, C\}$, $\{B, C, D\}$, $\{A, C, E\}$. Assume FDs $A \rightarrow D$, $CD \rightarrow E$, $E \rightarrow D$. Is the decomposition lossless?
  - Cannot use FD $E \rightarrow D, A \rightarrow D, CD \rightarrow E$

A	B	C	D	E
a	b	c	d_1	e
a_2	b	c	d	e_2
a	b_3	c	d_1	e

Example

- The tableau gives us tuples that satisfy the FDs
- Make the tableau into tuples
- Look at the projections

A	B	C	D	E
a	b	c	d_1	e
a_2	b	c	d	e_2
a	b_3	c	d_1	e

A	B	C
a	b	c
a_2	b	c
a	b_3	c

B	C	D
b	c	d_1
b	c	d
b_3	c	d_1

A	C	E
a	c	e
a_2	c	e_2
a	c	e

Example

- Join them

A	B	C	D	E
a	b	c	d_1	e
a_2	b	c	d	e_2
a	b_3	c	d_1	e

A	B	C
a	b	c
a_2	b	c
a	b_3	c

B	C	D
b	c	d_1
b	c	d
b_3	c	d_1

A	C	E
a	c	e
a_2	c	e_2

A	B	C	D	E
a	b	c	d_1	e
a	b	c	d	e
a_2	b	c	d_1	e_2
a_2	b	c	d	e_2
a	b_3	c	d_1	e
a	b_3	c	d_1	e_2

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Create a tableau with one row for each decomposition

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Create a tableau with one row for each decomposition
  - This represent potential values in the join

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_2	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_5	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies
  - Start with $A \to C$: Any tuple with the same A-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_2	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_5	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies
  - Start with $A \to C$: Any tuple with the same A-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_2	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_5	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Start with $A \rightarrow C$: Any tuple with the same A-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_2	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_5	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies
  - Start with $A \to C$: Any tuple with the same A-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_1	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
    - Start with $A \rightarrow C$: Any tuple with the same A-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_1	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies

- Apply $B \rightarrow C$: Any tuple with the same B-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_1	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies

- Apply $B \to C$: Any tuple with the same B-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d_2	e_2
a_3	b	c_3	d_3	e
a_4	b_4	c	d	e
a	b_5	c_1	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies

- Apply $B \to C$: Any tuple with the same B-value gets simplified

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d_2	e_2
a_3	b	c_1	d_3	e
a_4	b_4	c	d	e
a	b_5	c_1	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Apply $C \rightarrow D$:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d_2	e_2
a_3	b	c_1	d_3	e
a_4	b_4	c	d	e
a	b_5	c_1	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies

  - Apply $C \to D$:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d_2	e_2
a_3	b	c_1	d_3	e
a_4	b_4	c	d	e
a	b_5	c_1	d_5	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies
  - Apply $C \to D$:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c_1	d	e
a_4	b_4	c	d	e
a	b_5	c_1	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies
  - Apply $DE \to C$:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c_1	d	e
a_4	b_4	c	d	e
a	b_5	c_1	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Apply $DE \rightarrow C$: Unify the C-value if the D and E value are same

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c_1	d	e
a_4	b_4	c	d	e
a	b_5	c_1	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
    - Apply $DE \rightarrow C$: Unify the C-value if the D and E value are same

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c	d	e
a_4	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies

  - Apply $DE \rightarrow C$: Unify the C-value if the D and E value are same

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c	d	e
a_4	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Apply $CE \rightarrow A$

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c	d	e
a_4	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Apply $CE \rightarrow A$

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c	d	e
a_4	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Apply $CE \rightarrow A$

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a_3	b	c	d	e
a_4	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Apply $CE \rightarrow A$

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a	b	c	d	e
a	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, and $CE \rightarrow A$
- Now chase using functional dependencies
  - Apply $CE \rightarrow A$

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a	b	c	d	e
a	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies

- While we can continue, we do not have to:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a	b	c	d	e
a	b_4	c	d	e
a	b_5	c	d	e

Another Example

- $R(ABCDE)$ decomposed into $R_1(AD)$, $R_2(AB)$, $R_3(BE)$, $R_4(CDE)$, $R_5(AE)$
- Functional dependencies are $A \to C$, $B \to C$, $C \to D$, $DE \to C$, and $CE \to A$
- Now chase using functional dependencies
  - Middle row has no indexed values:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a	b	c	d	e
a	b_4	c	d	e
a	b_5	c	d	e

Another Example

- This shows that the decomposition has a lossless join

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c_1	d	e_2
a	b	c	d	e
a	b_4	c	d	e
a	b_5	c	d	e

Third Normal Form

- Decomposition
  - Avoid update and delete anomalies
  - Guarantee lossless joins
    - Decomposition **does not** generate spurious data
  - Want to check functional dependencies in the derived tables

Third Normal Form

- Sometimes, checking FDs is impossible
  - Example: `R(street, city, zip)`
    - `street, city —> zip`
    - `zip —> city`
  - Bring into BCNF: `{street, city}` and `{street, zip}` are keys, `zip —> city` violates BCNF
  - Decompose into `R1(zip, city)`, `R2(zip, street)`
  - But now we cannot check `street, city —> zip`

Third Normal Form

- Example:

street	zip
123 Wisconsin Avenue	53230
123 Wisconsin Avenue	53231

city	zip
Milwaukee	53230
Milwaukee	53231

  - looks fine

Third Normal Form

- But join gives:

street	city	zip
123 Wisconsin Avenue	Milwaukee	53230
123 Wisconsin Avenue	Milwaukee	53231

  - which violates a FD

Third Normal Form

- So we should not do this.
- “Elegant” solution: define the problem away
  - The original table needs to be “normal”
- A relation is in *third normal form* iff
  - For every non-trivial FD $X \rightarrow A$
    - X is a superkey
    - A is prime (member of at least one key

Third Normal Form

- A relation R is in ***third normal form***
  - If $A_1A_2\ldots A_n \rightarrow B_1B_2\ldots B_m$ is a non-trivial FD, then
    - either $\{A_1, A_2, \ldots, A_n\}$ is a superkey
    - or those of $\{B_1, B_2, \ldots, B_m\}$ not in $\{A_1, A_2, \ldots, A_n\}$ are each member of *some* key (not necessarily the same)

- Attributes that are part of some key are called ***prime***

Third Normal Form

- Example:
  - `bookings(title, theater, city)`

    ```
    theater --> city
    title, city --> theater
    ```

  - is in third normal form
    - city is part of a key

Third Normal Form

- Example:
  - `addresses(city, street, zip)`

    ```
    zip --> city
    city, street --> zip
    ```

  - is in third normal form
    - zip is prime

Third Normal Form

- Creation of 3NF Schemas
  - Want to decompose a relation R into a set of relations such that
    - All relations in the set are in 3NF
    - The decomposition has a lossless join
    - The decomposition preserves dependencies

Third Normal Form

- Synthesis Algorithm
  - Given a relation R and a set $\mathbb{F}$ of FDs
    - Find a minimal base $\mathbb{G}$ for $\mathbb{F}$
    - For all FD $X \to A \in \mathbb{G}$: use XA as a schema
    - If none of the relation schemas from previous step are a superkey for R, add another relation whose schema is a key for R

Third Normal Form

- Example:
  - $R(A, B, C, D, E)$ with FDs $AB \rightarrow C$, $C \rightarrow B$, $A \rightarrow D$

Third Normal Form

- Example:
  - The FDs are their own base:
    - Show: None of $AB \to C$, $C \to B$, $A \to D$ follows from the other two
    - Show: Cannot drop an attribute from a right side

Third Normal Form

- Example: $R(A, B, C, D, E)$ with FDs $AB \to C$, $C \to B$, $A \to D$
- This gives relations
    - $S_1(A, B, C)$, $S_2(B, C)$, $S_3(A, D)$
- Keys of R are A, B, E and A, C, E
    - Need to add one of them
    - $S_1(A, B, C)$, $S_2(B, C)$, $S_3(A, D)$, $S_4(A, C, E)$

Third Normal Form

- Why does this work
  - Lossless join:
    - We use the "Chase"
      - There is one subset of attributes in the decomposition that is a superkey $\mathbb{K}$.
      - The closure of $\mathbb{K}$ is all the attributes.
      - We start with a tableau

Third Normal Form

- Lossless join -- Chase
  - Use the FDs used in calculating the closure of $\mathbb{K}$.
  - We can assume that the FDs are in the base
  - Let the first FD be $\mathbb{X} \supset \mathbb{A} \to B$.
    - Tableau:

	$\mathbb{A}$	$\mathbb{X} \backslash \mathbb{A}$		B	rest of attributes
row $\mathbb{K}$	r,s,t,	e,	f,	b1	**
row FD	r,s	t1	e1 f1	b	**

    - The application of the FD sets `b1` to `b`

Third Normal Form

- Lossless join -- Chase
  - We continue the process.
    - Next FD might use column B or not, but because of it, we loose the subscript in the column corresponding to the right side
  - Eventually, we have removed all subscripts in the first row
  - Therefore, the decomposition is loss-less

Third Normal Form

- Dependency Preservation
  - Any FD is the consequence of the FDs in the base
  - Any FD in the base is represented by a relation in the decomposition
  - Therefore, we can first check those and as a consequence get all the FDs

Third Normal Form

- Is the decomposition in third normal form
  - If we add a relation that corresponds to a key, then this relation is by definition in third normal form
  - If we add a relation that corresponds to an FD in the basis:
    - **Can show**: If the relation is not in 3NF, then the basis is not minimal

Multivalued Dependencies

- First Normal Form: All values in a relation are atomic
  - This is removed by object-relational databases
- If the value of an attribute is a set, we represent it by using many relations

A	B	C
1	2	{3,4}
4	5	{3,4}

A	B	C
1	2	3
1	2	4
4	5	3
4	5	4

Multivalued Dependencies

- A more practical example
  - Relation `course(number, book, lecturer)`
- In this department, the books recommended and the lecturers are independent.
- ```
  calc 1 | Ross | Krenz
  calc 1 | Lang | Krenz
  calc 1 | Ross | Sanders
  calc 1 | Lang | Sanders
  calc 2 | Ash  | Gillen
  calc 2 | Ash  | Engbers
  calc 1 | Ross | Schwarz
  calc 1 | Lang | Schwarz
  ```
  ```

Multivalued Dependencies

- The same list can be expressed using sets more simply

```
calc 1 | Ross | Krenz
calc 1 | Lang | Krenz
calc 1 | Ross | Sanders
calc 1 | Lang | Sanders
calc 2 | Ash  | Gillen
calc 2 | Ash  | Engbers
calc 1 | Ross | Schwarz
calc 1 | Lang | Schwarz
```

```
calc1 | {Ross, Lang} | {Krenz, Sanders, Schwarz}
calc2 | {Ash}        | {Gillen, Engbers}
```

Multivalued Dependencies

- It would be an error to add a single tuple
  - `calc 1 | Burlow | Krenz`
- to the relation
  - indicating that an additional book is now recommended
- Instead, need to add:
  - `calc 1 | Burlow | Sanders`
  - `calc 1 | Burlow | Schwarz`
- as well

Multivalued Dependencies

- This gives rise to the definition of a multivalued dependency
    - Unlike before, we now demand that additional tuples exist in the relation.

Multivalued Dependencies

- Formally: $A_1, A_2, \ldots, A_n \twoheadrightarrow B_1, \ldots B_m$
- Whenever
  - two tuples agree on its values in $A_1, A_2, \ldots, A_n$
  - the tuples have values $b_1 \ldots b_m$ and $b'_1 \ldots b'_m$ in $B_1, B_2, \ldots, B_m$
  - the tuples have values $x_1 \ldots x_r$ and $x'_1 \ldots x'_r$ in the other attributes
  - then the tuples $a_1 \ldots a_n b'_1 \ldots b'_m x_1 \ldots x_r$ and $a'_1 \ldots a'_n b_1 \ldots b_m x'_1 \ldots x'_r$ also exist

Multivalued Dependencies

- For each pair of tuples *t* and *u* of a relation *R* that agree on all attributes $A_1, A_2, \ldots, A_n$:
  - We can find another tuple *v* such that *v* agrees :
    - With both *t* and *u* on $A_1, A_2, \ldots, A_n$
    - With *t* on $B_1, B_2, \ldots, B_m$
    - With *u* on all attributes that are not among the As and Bs

Multivalued Dependencies

- Example
  - Relation courses

```
calc 1 | Ross | Krenz
calc 1 | Lang | Krenz
calc 1 | Ross | Sanders
calc 1 | Lang | Sanders
calc 2 | Ash  | Gillen
calc 2 | Ash  | Engbers
calc 1 | Ross | Schwarz
calc 1 | Lang | Schwarz
```

  - has FD course $\twoheadrightarrow$ book and course $\twoheadrightarrow$ lecturer

Multivalued Dependencies

- Example `stars(name, address, movie)`
  - A star can have several address and can be in several movies
  - But: for each movie, all addresses of the star need to appear
  - But: for each address, all movies need to appear

Multivalued Dependencies

- Trivial MVD
  - If $\{B_1, \ldots, B_m\} \subset \{A_1, \ldots, A_n\}$ then
    - $A_1 \ldots A_n \twoheadrightarrow B_1 \ldots B_m$

Multivalued Dependencies

- Transitive MVDs
  - $A_1 \ldots A_n \twoheadrightarrow B_1 \ldots B_m$ and $B_1 \ldots B_m \twoheadrightarrow C_1 \ldots C_k$ implies $A_1 \ldots A_n \twoheadrightarrow C_1 \ldots C_k$
  - Provided that we remove any C-attributes that are also A-attributes

Multivalued Dependencies

- Splitting is **NOT** true
  - `stars(name, street, city, title, year)`
- has MVD
  - `name` $\twoheadrightarrow$ `street, city`
- However, `name` $\twoheadrightarrow$ `street` is not true.
  - `John Wayne, 123 Elm Street, Malibu, Hatari, 1962`
  - `John Wayne, 456 Overland, Culver City, Hatari, 1962`
  - `John Wayne, 123 Elm Street, Malibu, Shootist, 1976`
  - `John Wayne, 456 Overland, Culver City, Shootist, 1976`
  - DOES NOT HAVE
  - `John Wayne, 123 Elm Street, Culver City, Hatari, 1962`

Multivalued Dependencies

- Promotion
  - Any FD is also an MVD

Multivalued Dependencies

- Complementation
  - If $A_1 \ldots A_n \twoheadrightarrow B_1 \ldots B_m$ and $C_1 \ldots C_k$ are the attributes not in the As and Bs, then $A_1 \ldots A_n \twoheadrightarrow C_1 \ldots C_k$

Fourth Normal Form

- A relation is in fourth normal form if whenever $A_1 \ldots A_n \twoheadrightarrow B_1 \ldots B_m$ is a non-trivial MVD
- Then $A_1 \ldots A_n$ is a super-key

Normal Forms

- We have 4NF $\Rightarrow$ BCNF $\Rightarrow$ 3NF
-

	3NF	BCNF	4NF
eliminate redundancies due to FDs	no	yes	yes
eliminates redundancies due to MVDs	no	no	yes
preserves FDs	yes	no	no
preserves MVDs	no	no	no
lossless joins	yes	yes	yes