Relational Database Design

Relation *leaves*

Flight	Pilot	Date	From	To	Departs
83	Cushing	9 Aug	JFK	O'Hare	9:10a
116	Cushing	10 Aug	Denver	O'Hare	10:32a
281	Clark	8 Aug	Denver	JFK	9:30p
301	Clark	12 Aug	Atlanta	Houston	1:15p
83	Clark	11 Aug	JFK	O'Hare	9:10a
83	Chin	13 Aug	JFK	O'Hare	9:10a
116	Chin	12 Aug	Denver	O'Hare	10:32a
281	Copley	9 Aug	Denver	JFK	9:30p
281	Copley	13 Aug	Denver	JFK	9:30p
412	Copley	15 Aug	Denver	JFK	1:25p

Questions

- What is a minimal key for R?
  - Given a relation scheme R, a key on R is a subset K of R such that for any permissible relation $r(R)$, there are no two distinct tuples t_1 and t_2 in r such that $t_1[K] = t_2[K]$, and no proper subset K' of K has this property.
  - In terms of FDs, a key for a scheme R is a subset K of R such that any permissible relation $r(R)$ satisfies $K \rightarrow R$, but no proper subset of K has this property.
  - A superkey is a subset K of R such that $K \rightarrow R$ but there is no requirement for minimality.

Pitfalls in relational database design

- Repetition of information
- Inability to represent certain information
- Loss of information
- Representation of information
  - Add information on a new flight to the *leaves* relation
  - Problem
    * Addition of information that is already there
    * Wastage of space
    * Costly updates (in case of changes to flight schedule)
    * Integrity problem
    * Consistency problem
- Loss of information
  - Can we solve the problem by decomposing the relation into two?
  - Bad decomposition may lead to problems
  - Decompose on the basis of Pilot
  - Lossless join
    * Central to relational database design
    * A set of relation schemes $\{R_1, R_2, R_3, \ldots, R_n\}$ is a decomposition of R if

$$R = \bigcup_{i=1}^{n} R_i$$

Every attribute in R appears in at least one R_i

- Let r be a relation on scheme R
- Let $r_i = \Pi_{R_i}(r)$ for $1 \leq i \leq n$
- It is always the case that

$$r \subseteq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

- In general,

$$r \neq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

- Additional constraints for lossless decomposition
  - Decomposition based on FDs
  - Relation is *legal* if it satisfies all rules (constraints) imposed on the database
- A decomposition $\{R_1, R_2, \ldots, R_n\}$ of R is a *lossless-join decomposition* for R if for all relations r on R that are legal under a set of constraints C

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \ldots \bowtie \Pi_{R_n}(r)$$

Normalization using functional dependencies

- Define several normal forms to represent good database design
- First normal form
  - A relation scheme R is in first normal form (1NF) if the values in `dom`(A) are atomic for every attribute A in R
  - A value is non-atomic if the application deals with only a part of the value
  - Relation *born* to list birthdays

Name	Birthdate
Allen	June 7, 1949
Alfred	March 21, 1933
Alphonse	April 30, 1959
Alice	December 12, 1963

  Relation *born* is not in 1NF if we are only interested in the month and year of birth of a person

  - Relation *gender*

Name	Sex
{John, Jean, Ivan}	male
{Mary, Marie}	Female

  - Advantages of 1NF
    - It may not be possible to express FDs in as great detail as we would like without 1NF
      - Suppose we want to express astrological signs with birthdate
    - Problem with updates
      - Add another male or delete one
- Second normal form
  - Tries to avoid anomalies when updating relations
  - Relation *assign*

Flight	Day	Pilot	Gate
112	6 June	Bosley	7
112	7 June	Brooks	7
203	9 June	Bosley	12

  - Key for *assign* – (Flight Day)

- FD – Flight → Gate
- Update the relation with

  update *assign*
  set Gate = 8
  where Flight = 112 **and** Day = "6 June"

- Violation of FD (Flight → Gate)
- Flight number and Gate number information is duplicated in the relation
- To avoid redundancy, break the relation into two

Relation *passign*

Flight	Day	Pilot
112	6 June	Bosley
112	7 June	Brooks
203	9 June	Bosley

Relation *gassign*

Flight	Gate
112	7
203	12

- Original relation *assign* can be reconstructed by taking *passign* ⋈ *gassign*
- No update anomaly as only one tuple needs to be altered
- Less data redundancy
- Definitions
  * Given a relation scheme R, and attribute A in R, and a set of FDs F over R. Attribute A is *prime* in R with respect to F if A is contained in some key of R. Otherwise A is *nonprime* in R.
  * A relation scheme R is in *second normal form* (2NF) with respect to a set of FDs F if it is in 1NF and every nonprime attribute is fully dependent on every key of R. A database scheme $\mathbf{R}$ is in *second normal form* with respect to F if every relation scheme R in $\mathbf{R}$ is in 2NF with respect to F.

• Lossless-Join Decomposition
  - Decomposition of a relation must be lossless
  - Relation scheme R
  - F is a set of FDs on scheme R
  - Decompose R into R_1 and R_2
  - Decomposition is lossless if at least one of the following FDs are in F^+
    * $R_1 \cap R_2 \rightarrow R_1$
    * $R_1 \cap R_2 \rightarrow R_2$

• Dependency preservation
  - Basic goal of 2NF
  - System should be able to check that the update will not create an illegal relation
    * All the functional dependencies must be satisfied by the updated relation
    * For efficiency, the database schema should be designed so as to allow update validation without computation of joins
  - *How can you decide whether joins must be computed?*
    * Determine the functional dependencies to be tested by checking each relation individually
    * Let F be the set of FDs on schema R and let $R_1, R_2, \ldots, R_n$ be a decomposition of R
    * *Restriction* of F to R_i
      · Set F_i of all FDs in F^+ that include *only* attributes of R_i
    * Since all FDs in a restriction involve attributes of only one relation schema, it is possible to test satisfaction of dependency by checking only one relation

- Set of restrictions $F_1, F_2, \ldots, F_n$ is the set of dependencies that can be checked efficiently; now the question is *whether testing only the restrictions is sufficient*
  - Let $F' = F_1 \cup F_2 \cup \cdots \cup F_n$ be the set of FDs on schema R
  - In general, $F' \neq F$ but F'^+ may be equal to F^+, that is, closure of F' and F may be the same
  - Even if $F' \neq F$, it may be possible that $F'^+ = F^+$, in which case, every dependency of F is logically implied by F'
  - If we verify that F' is satisfied, we have already verified that F is satisfied as well

– Dependency-preserving decomposition
- A decomposition having the property $F'^+ = F^+$ is said to be dependency-preserving
- Dependency preservation testing algorithm

```
compute F+
foreach scheme Ri ∈ D
   compute Fi
end
F' = ∅
foreach restriction Fi
   set F' = F' ∪ Fi
end
compute F'+
if ( F'+ == F+ ) then
   return (true)
else
   return (false)
endif
```

Third normal form

- Allows you to find a lossless-join dependency-preserving decomposition but is more restrictive than 2NF
- Consider new version of relation *assign*

Flight	Day	Pilot-id	Name
112	6 June	31174	Bosley
112	7 June	30046	Brooks
203	9 June	31174	Bosley

- Key for *assign* – (Flight Day)
- FDs – (Pilot-id $\rightarrow$ Name) and (Name $\rightarrow$ Pilot-id)
- Update the relation with

update *assign*
set Pilot-id = 31039
where Flight = 112 **and** Day = "6 June"

- Violation of FD (Name $\rightarrow$ Pilot-id)
- Redundant Pilot-id–Name pairs
- Solution – Break up the relation into two with schemes

Flight Day Pilot-id
Pilot-id Name

- Difference from 2NF – Total dependence v Partial dependence
- Definitions
  - Given a relation scheme R, a subset R_i of R, an attribute $A \in R$, and a set of FDs F. A is *transitively dependent* upon R_i in R if there is a subset R_j of R with $R_i \rightarrow R_j$, $R_j \not\rightarrow R_i$, and $R_j \rightarrow A$ under F and $A \notin R_i R_j$.
  - Example: Let $R =$ (Flight Day Pilot-id Name).
    Let $F = \{$ (Flight Day $\rightarrow$ Pilot-id Name), (Pilot-id $\rightarrow$ Name), (Name $\rightarrow$ Pilot-id) $\}$.
    (Name) is transitively dependent upon (Flight Day).
    (Flight Day $\rightarrow$ Pilot-id), (Pilot-id $\not\rightarrow$ Flight Day), and (Pilot-id $\rightarrow$ Name)
  - A relation scheme R is in *third normal form* (3NF) with respect to a set of FDs F if it is in 1NF and no nonprime attribute in R is transitively dependent upon a key of R.
  - Another definition for 3NF: A relation scheme R is in 3NF with respect to a set of functional dependencies F if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
    * $\alpha \rightarrow \beta$ is a trivial functional dependency
    * α is a superkey for R
    * Each attribute $A \in (\beta - \alpha)$ is contained in a candidate key for R
- **Lemma**. Any relation scheme R that is in 3NF with respect to F is in 2NF with respect to F
- Algorithm to find a dependency-preserving lossless-join decomposition into 3NF

```
Fc = canonical cover for F
i = 0
foreach FD α → β ∈ Fc
   if none of the schemes Rj contain αβ then # 1 ≤ j ≤ i
      i++
      Ri = αβ
   endif
   if none of the schemes Rj contain a candidate key for R then # 1 ≤ j ≤ i
      i++
      Ri = any candidate key for R
   endif
end
return (R1, R2, ..., Ri)
```

  - The fact that each relation schema R_i is in 3NF follows directly from the requirement that the set F of FDs be in canonical form
  - The algorithm ensures preservation of dependenciesby building explicitly a schema for each given dependency
  - Decomposition is a lossless-join decomposition because of the guarantee that at least one schema contains a candidate key for the schema being decomposed
- Example following the algorithm
  - Consider the following schema

    `Banker-info = ( branch_name, customer_name, banker_name, office_number )`

  - We have the following FDs:
    * `banker_name` $\rightarrow$ `branch_name office_number`
    * `customer_name branch_name` $\rightarrow$ `banker_name`
  - The `for` loop causes us to include the following schemas in the decomposition

    * banker_office = (banker_name, branch_name, office_number)
    * banker = (customer_name, branch_name, banker_name)

  - Since banker contains a candidate key for banker_info, we are finished with the decomposition process

Boyce-Codd Normal Form

- Problem with 3NF
  - Consider the relation *billto*(Airport Company Office)
  - Tuple $\langle a \ c \ f \rangle$
    * If someone from a company c charges a ticket at airport a, the bill should be sent to office f of the company
  - Two FDs
    * Airport Company $\rightarrow$ Office
    * Office $\rightarrow$ Company
  - Key for relation – (Airport Office)
  - Company is partially [transitively] dependent upon (Airport Office)
  - Company is a prime attribute but desirable to remove it from relation because of duplication of (Company Office) pairs
- Definition 1. A relation scheme R is in Boyce-Codd normal form (BCNF) with respect to a set of FDs F if it is in 1NF and no attribute in R is transitively dependent upon any key of R
- Definition 2. A relation scheme R is in Boyce-Codd normal form (BCNF) with respect to a set of FDs F if for every subset R_j of R, and for every attribute $A \in R - R_j$, if $R_j \rightarrow A$, then $R_j \rightarrow R$ under F. That is, if R_j non-trivially determines any attribute of R, then R_j is a superkey for R
- Example using the above scheme
  - Use the FD (Office $\rightarrow$ Company) to decompose R into $R_1 =$ (Airport Office) with designated key $K_1 =$ {Airport Office} and $R_2 =$ (Company Office) with designated key $K_2 =$ {Office}
- Algorithm to compute BCNF

```
result = {R}
done = false
compute F+
while (not done)
   if ∃ scheme Ri ∈ result that is not in BCNF then
      let α → β be a nontrivial FD that holds on Ri such that
         α → Ri ∉ F+, and
         α ∩ β = ∅
      result = (result - Ri) ∪ (Ri − β) ∪ (α, β)
   else
      done = true
   endif
end
```

- Problems with BCNF
  - Given a set of FDs over R, it is possible to find a 3NF scheme that completely characterizes F
  - Not every BCNF decomposition preserves FDs
  - Choice between BCNF and enforceable FDs

- NP-complete problem to decide if a DB is in BCNF
- Example
  * Consider the relation scheme

    banker-scheme(branch-name, customer-name, banker-name)
  * Set of FDs

    banker-name $\rightarrow$ branch-name
    customer-name branch-name $\rightarrow$ banker-name
  * *banker-scheme* is not in BCNF since banker-name is not a superkey
  * With the algorithm, the following decomposition occurs

    banker-branch-scheme = (banker-name, branch-name)
    customer-banker-scheme = (customer-name, banker-name)

    It preserves the FD banker-name $\rightarrow$ branch-name but the closure of {banker-name $\rightarrow$ branch-name} does not include (customer-name branch-name $\rightarrow$ banker-name)
  * Dependency preservation
    · Apply the algorithm to check dependency preservation
    · Canonical cover for F

    $$\begin{array}{rcl} F_1 & = & \{\text{banker-name} \rightarrow \text{branch-name}\} \\ F_2 & = & \emptyset \text{ (only trivial dependencies hold on } \textit{customer-banker-scheme}) \end{array}$$

    · Canonical cover for the set F' is F_1
    · Dependency (customer-name branch-name $\rightarrow$ banker-name) is not in F'^+ even though it is in F^+
    · Decomposition is not dependency-preserving

- Comparison of BCNF and 3NF
  - Always possible to obtain a 3NF design without sacrificing lossless join or dependency preservation
  - If transitive dependencies are not eliminated, it may be necessary to use null values to represent some of the possible meaningful relationships among data items, plus problem of repetition of information

Normalization using multivalued dependencies

- Presence of certain FDs in a scheme means that the scheme can be decomposed to eliminate redundancy while preserving information
- Such an FD is not necessary for decomposition
- Relation *service*

Flight	Day	Plane-Type
106	Monday	747
106	Thursday	747
106	Monday	1011
106	Thursday	1011
204	Wednesday	707
204	Wednesday	727

- A tuple $\langle f \;\; d \;\; p \rangle$ means that the flight number f flies on day d and can use plane type p on that day
- No FD (Flight $\rightarrow$ Day) or (Flight $\rightarrow$ Plane-Type) in the relation
- Decomposed relation

Relation: *servday*

Flight	Day
106	Monday
106	Thursday
204	Wednesday

Relation: *servtype*

Flight	Plane-Type
106	747
106	1011
204	707
204	727

- Another instance of relation *service*

Flight	Day	Plane-Type
106	Monday	747
106	Thursday	747
106	Thursday	1011
204	Wednesday	707
204	Wednesday	727

- Decomposition gives us the same two relations *servday* and *servtype* but join does not return the original relation
- Multivalued dependencies
  - Problem with second instance of relation *service*
  - In first instance, if we know Flight, Day does not give us any information about Plane-Type
  - If we have tuples $\langle f \;\; d \;\; p \rangle$ and $\langle f \;\; d' \;\; p' \rangle$ in the relation, we must also have tuple $\langle f \;\; d' \;\; p \rangle$
  - **Definition:** Let R be a relation scheme, let R_1 and R_2 be two disjoint subsets of R, and let $R_3 = R - (R_1 R_2)$. A relation $r(R)$ satisfies the *multivalued dependency* (MVD) $R_1 \twoheadrightarrow R_2$ if, for any two tuples t_1 and t_2 in r with $t_1(R_1) = t_2(R_1)$, there exists a tuple t_3 in r with $t_3(R_1) = t_1(R_1)$, $t_3(R_2) = t_1(R_2)$, and $t_3(R_3) = t_2(R_3)$ The symmetry of t_1 and t_2 in this definition implies there is also a tuple t_4 in r with $t_4(R_1) = t_1(R_1)$, $t_4(R_2) = t_2(R_2)$, and $t_4(R_3) = t_1(R_3)$
  - **Example**: The MVD Flight $\twoheadrightarrow$ Day holds on the first instance of *service* but not on the second instance. The first instance also satisfies the MVD Flight $\twoheadrightarrow$ Plane-Type
  - **Lemma**: If relation r on scheme R satisfies the MVD $R_1 \twoheadrightarrow R_2$ and $R_3 = R - (R_1 \; R_2)$, then r satisfies $R_1 \twoheadrightarrow R_3$
  - The definition of MVD requires that R_1 and R_2 be disjoint in $R_1 \twoheadrightarrow R_2$
    * Suppose the disjoint condition is removed from the definition
    * Let relation $r(R)$ satisfy the MVD $R_1 \twoheadrightarrow R_2$
    * Let $R_2' = R_2 - R_1$
    * Therefore, r satisfies $R_1 \twoheadrightarrow R_2'$
    * Let $R_3 = R - (R_1 \;\; R_2) = R - (R_1 \;\; R_2')$
    * Let t_1 and t_2 be tuples in r with $t_1(R_1) = t_2(R_1)$
    * Since $R_1 \twoheadrightarrow R_2$, there must be a tuple t_3 in r with $t_3(R_1) = t_1(R_1)$, $t_3(R_2) = t_2(R_1)$, and $t_3(R_3) = t_2(R_3)$
    * If $t_3(R_2) = t_1(R_2)$, then $t_3(R_2') = t_1(R_2')$ since $R_2' \subseteq R_2$
    * So, r satisfies $R_1 \twoheadrightarrow R_2'$
  - Suppose that R_1 and R_2 are disjoint and relation $r(R)$ satisfies $R_1 \twoheadrightarrow R_2$
    * If $R_1' \subseteq R_1$, then $R_1 \twoheadrightarrow R_2 R_1'$
    * If tuples t_1 and t_2 are in r, and $t_1(R_1) = t_2(R_1)$, then $\exists$ a tuple $t_3 \in r$ with $t_3(R_1) = t_1(R_1)$, $t_3(R_2) = t_1(R_2)$, and $t_3(R_3) = t_2(R_3)$.
    * Therefore, $t_3(R_2 R_1') = t_1(R_2 R_1')$
  - **Example**: The relation r shown below satisfies the MVD $AB \twoheadrightarrow BC$, hence it satisfies the MVD $AB \twoheadrightarrow C$

A	B	C	D
a	b	c	d
a	b	c’	d’
a	b	c	d’
a	b	c’	d
a	b’	c’	d
a’	b	c	d’

- Theory of multivalued dependencies
  - Reflexivity rule. If R_1 is a set of attributes and $R_2 \subseteq R_1$, then $R_1 \rightarrow R_2$ holds. Furthermore, relation r satisfies $R_1 \rightarrow\rightarrow R_1$.
  - Augmentation rule. If $R_1 \rightarrow R_2$ holds and R_3 is a set of attributes, then $R_3R_1 \rightarrow R_3R_2$ holds. Moreover, if r satisfies $R_1 \rightarrow\rightarrow R_2$, then r also satisfies $R_1R_3 \rightarrow\rightarrow R_2$.
  - Transitivity rule. If $R_1 \rightarrow R_2$ holds and $R_2 \rightarrow R_3$ holds, then $R_1 \rightarrow R_3$ holds. If r satisfies $R_1 \rightarrow\rightarrow R_2$ and $R_2 \rightarrow\rightarrow R_3$ then r also satisfies $R_2 \rightarrow\rightarrow R_3 - R_2$.
  - Complementation rule. If $R_1 \rightarrow\rightarrow R_2$ holds then $R_1 \rightarrow\rightarrow R - R_1 - R_2$ holds
  - Multivalued augmentation rule. If $R_1 \rightarrow\rightarrow R_2$ holds and $R_3 \subseteq R$ and $R_3' \subseteq R_3$, then $R_1R_3 \rightarrow\rightarrow R_2R_3'$ holds
  - Multivalued transitivity rule. If $R_1 \rightarrow\rightarrow R_2$ holds and $R_2 \rightarrow\rightarrow R_3$ holds, then $R_1 \rightarrow\rightarrow R_3 - R_2$ holds
  - Replication rule. If $R_1 \rightarrow R_2$ holds, then $R_1 \rightarrow\rightarrow R_2$
  - Coalescence rule. If $R_1 \rightarrow\rightarrow R_2$ holds and $R_3 \subseteq R_2$ and $\exists\ R_4$ such that $R_4 \subseteq R$ and $R_4 \cap R_2 = \emptyset$ and $R_4 \rightarrow R_3$, then $R_1 \rightarrow R_3$ holds

  The above rules are sound (the rules do not generate any dependencies that are not logically implied by the set of FDs and MVDs) and complete (the rules allow us to generate all depenedncies in D^+)

- Additional rules to compute D^+, the closure of FDs and MVDs
  - Multivalued union rule. If $R_1 \rightarrow\rightarrow R_2$ holds and $R_1 \rightarrow\rightarrow R_3$ holds, then $R_1 \rightarrow\rightarrow R_2R_3$ holds
  - Intersection rule. If $R_1 \rightarrow\rightarrow R_2$ holds and $R_1 \rightarrow\rightarrow R_3$ holds, then $R_1 \rightarrow\rightarrow R_2 \cap R_3$ holds
  - Difference rule. If $R_1 \rightarrow\rightarrow R_2$ holds and $R_1 \rightarrow\rightarrow R_3$ holds, then $R_1 \rightarrow\rightarrow R_2 - R_3$ holds and $R_1 \rightarrow\rightarrow R_3 - R_2$ holds

- Fourth Normal Form
  - Any relation $r(R)$ that satisfies the MVD $R_1 \rightarrow\rightarrow R_2$ decomposes losslessly onto the relation schemes R_1R_2 and R_1R_3, where $R_3 = R - (R_1R_2)$
  - If $R_1 \rightarrow\rightarrow R_2$ is the only dependency on R, then R is in 3NF
  - Therefore, 3NF is not guaranteed to find all possible decompositions
  - **Definition.** An MVD $R_1 \rightarrow\rightarrow R_2$ is *trivial* if for any relation scheme R with $R_1R_2 \subseteq R$, any relation $r(R)$ satisfies $R_1 \rightarrow\rightarrow R_2$
  - The trivial MVDs on a relation $r(R)$ are exactly those of the form $R_1 \rightarrow\rightarrow R_2$ where $R_2 \subseteq R_1$ or $R_1R_2 = R$
  - If $R_1 \rightarrow\rightarrow R_2$ is trivial, and we attempt to decompose a relation $r(R)$ using it, one of the projected relations will be all of r
  - **Definition.** An MVD $R_1 \rightarrow\rightarrow R_2$ *applies* to a relation scheme R if $R_1R_2 \subseteq R$
  - **Definition.** Let F be a set of FDs and MVDs over $\mathbf{U}$. A relation scheme $R \subseteq \mathbf{U}$ is in *fourth normal form* (4NF) with respect to F if for every MVD $R_1 \rightarrow\rightarrow R_2$ implied by F that applies to R either the MVD is trivial or R_1 is a superkey for R
  - **Example.** Let relation scheme $R =$ (Flight Day-of-week Plane-type) and let $F = \{$ Flight $\rightarrow\rightarrow$ Day-of-week $\}$. R is not in 4NF with respect to F. The data-base scheme $\mathbf{R} =$ (Flight Day-of-week, Flight Plane-type) is in 4NF with respect to F. Any relation $r(R)$ that satisfies F decomposes losslessly onto the relation schemes in $\mathbf{R}$.
  - **Example.** Let $F = \{A \rightarrow BC, C \rightarrow\rightarrow DE\}$ be a set of dependencies over the relation scheme $R = ABCDE$. R is not in 4NF with respect to F because of the MVD $C \rightarrow\rightarrow DE$. The database scheme $\mathbf{R}$ consisting of the two relation schemes $R_1 = ABC$, and $R_2 = CDE$ is in 4NF with respect to F, even though the MVD $A \rightarrow\rightarrow B$ is implied by F and applies to R_1. $A \rightarrow\rightarrow B$ is not trivial, but A is a key for R_1.

- **Example.** Let $F = \{A \to\to BCD, B \to AC, C \to D\}$ be a set of dependencies over the relation scheme $R = ABCDEI$. Since $A \to\to BCD$ is a nontrivial MVD and A is not a key for R, we decompose R into relation schemes $R_1 = ABCD$ and $R_2 = AEI$. R_2 is in 4NF with respect to F. F implies the MVD $B \to\to AC$ on R, but this MVD is not a candidate for use in decomposition because B is a key for R_1, since $C \to D$. However $C \to D$ implies the MVD $C \to\to D$, which we can use to decompose R_1. The result is the relation schemes $R_{11} = ABC$ and $R_{12} = CD$. Both of these schemes are in 4NF with respect to F. The database scheme $\mathbf{R} = R_{11}, R_{12}, R_2$ is in 4NF with respect to F.
- **Lemma.** If a relation scheme R is in 4NF with respect to a set F of FDs and MVDs, then R is in BCNF with respect to the set of FDs implied by F.

<u>Normalization using join dependencies</u>

- MVDs are an attempt to detect lossless decompositions that will work for all relations on a given relation scheme
- Consider the relation r

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_3	b_3	c_3
a_4	b_3	c_4
a_5	b_5	c_5
a_6	b_6	c_5

- The relation decomposes losslessly onto the schemes (AB), (A, C), and (B, C)

A	B
a_1	b_1
a_1	b_2
a_3	b_3
a_4	b_3
a_5	b_5
a_6	b_6

A	C
a_1	c_1
a_1	c_2
a_3	c_3
a_4	c_4
a_5	c_5
a_6	c_5

B	C
b_1	c_1
b_2	c_2
b_3	c_3
b_3	c_4
b_5	c_5
b_6	c_5

- r satisfies no nontrivial MVDs
- **Definition**: Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a set of relation schemes over $\mathbf{U}$. A relation $r(\mathbf{U})$ satisfies the *join dependency (jd)* $*[R_1, R_2, \ldots, R_p]$ if r decomposes losslessly onto $R_1, R_2, \ldots, R_p$. That is,

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \cdots \bowtie \pi_{R_p}(r)$$

- Relation r above satisfies the JD $*[AB, AC, BC]$
- A necessary condition for a relation $r(\mathbf{U})$ to satisfy the JD $*[R_1, R_2, \ldots, R_p]$ is that $\mathbf{U} = R_1, R_2, \ldots, R_p$
- JD is a special case of MVD
  - A relation $r(R)$ satisfies the MVD $X \to\to Y$ if and only if r decomposes losslessly onto XY and XZ, where $Z = R - (XY)$
  - Same thing as the JD $*[XY, XZ]$
  - JD $*[R_1, R_2]$ is the same as the MVD $R_1 \cap R_2 \to\to R_1$
- Definition along the lines of MVD definition: Let R satisfy $*[R_1, R_2, \ldots, R_p]$. If r contains tuples $t_1, t_2, \ldots, t_p$ such that

$$t_i(R_i \cap R_j) = t_j(R_i \cap R_j)$$

for all i and j, then r must contain a tuple t such that $t(R_i) = t_i(r_i)$, $1 \leq i \leq p$

- Example: Suppose relation $r(ABCDE)$ satisfies the JD $*[ABC, BD, CDE]$ and contains the three tuples shown below.

	A	B	C	D	E
t_1	a	b	c	d	e
t_2	a'	b	c'	d'	e''
t_3	a''	b'	c	d'	e'

  Using our alternative definition, r must also contain the tuple $t = \langle abcd'e' \rangle$

- There is no set of sound and complete inference rules for the join dependencies
- Project-Join Normal Form
  - We have lossless decompositions that do not correspond to MVDs
  - 4NF is not the ultimate in finding lossless decompositions
  - **Definition**: A JD $*[R_1, R_2, \ldots, R_p]$ over R is *trivial* if it is satisfied by every relation $r(R)$
  - **Definition**: A JD $*[R_1, R_2, \ldots, R_p]$ *applies to* a relation scheme R if $R = R_1, R_2, \ldots, R_p$
  - Let R be a relation scheme and let F be a set of FDs and JDs over R. R is in *project-join normal form* (PJNF) with respect to F if for every JD $*[R_1, R_2, \ldots, R_p]$ implied by F that applies to R, the JD is trivial, or every R_i is a superkey for R. A database scheme $\mathbf{R}$ is in PJNF with respect to F if every relation scheme $R \in \mathbf{R}$ is in PJNF with respect to F.

Domain-Key Normal Form

- Based on three notions
  1. Domain declaration
     - Requires that the attribute value of all tuples be in corresponding domain
  2. Key declaration
     - The declared key must be a superkey for the relation scheme
  3. General constraint
     - A predicate on the set of all relations on a given scheme
     - All dependencies
     - Preferable to express as a clause in first-order logic
     - "If the first digit of attribute `account-number` is 9, then the value of attribute `balance` is greater than or equal to 2500."
- DKNF allows the general constraints to be tested using only domain and key constraints
- **Definition:** Let $\mathbf{D}$ be a set of domain constraints and let $\mathbf{K}$ be a set of key constraints for a relation scheme R. Let $\mathbf{G}$ denote the general constraints for R. Scheme R is in domain-key normal form (DKNF) if $\mathbf{D} \cup \mathbf{K} \Rightarrow \mathbf{G}$.