Database Modifications

- A *modification* command does not return a result (as a query does), but changes the database in some way
- Three kinds of modifications:
  1. *Insert* a tuple or tuples
  2. *Delete* a tuple or tuples
  3. *Update* the value(s) of an existing tuple or tuples

Insertion

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);

- Example: add to Likes(drinker, beer) the fact that Lars likes Odense Classic.

```
INSERT INTO Likes
  VALUES('Lars', 'Od.Cl.');
```

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes
- Two reasons to do so:
  1. We forget the standard order of attributes for the relation
  2. We don't have values for all attributes, and we want the system to fill in missing components with NULL or a default value

Example: Specifying Attributes

- Another way to add the fact that Lars likes Odense Cl. to Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
VALUES('Od.Cl.', 'Lars');
```

Adding Default Values

- In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value
- When an inserted tuple has no value for that attribute, the default will be used

Example: Default Values

```
CREATE TABLE Drinkers (
    name CHAR(30) PRIMARY KEY,
    addr CHAR(50)
        DEFAULT 'Vestergade',
    phone CHAR(16)
);
```

Example: Default Values

```
INSERT INTO Drinkers(name)
  VALUES('Lars');
```

Resulting tuple:

name	address	phone
Lars	Vestergade	NULL

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

  INSERT INTO <relation>

  (<subquery>);

Example: Insert a Subquery

- Using Frequents(drinker, bar), enter into the new relation PotBuddies(name) all of Lars "potential buddies", i.e., those drinkers who frequent at least one bar that Lars also frequents

The other
drinker

Solution

Pairs of Drinker
tuples where the
first is for Lars,
the second is for
someone else,
and the bars are
the same

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = ’Lars’ AND
   d2.drinker <> ’Lars’ AND
   d1.bar = d2.bar
);
```

Deletion

- To delete tuples satisfying a condition from some relation:

  DELETE FROM <relation>

  WHERE <condition>;

Example: Deletion

- Delete from Likes(drinker, beer) the fact that Lars likes Odense Classic:

```
DELETE FROM Likes
    WHERE drinker = 'Lars' AND
          beer = 'Od.Cl.';
```

Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

Example: Delete Some Tuples

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
DELETE FROM Beers b
WHERE EXISTS (
  SELECT name FROM Beers
  WHERE manf = b.manf AND
    name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b

Semantics of Deletion

- Suppose Albani makes only Odense Classic and Eventyr
- Suppose we come to the tuple *b* for Odense Classic first
- The subquery is nonempty, because of the Eventyr tuple, so we delete Od.Cl.
- Now, when *b* is the tuple for Eventyr, do we delete that tuple too?

Semantics of Deletion

- Answer: we *do* delete Eventyr as well
- The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied
  2. Delete the marked tuples

Updates

- To change certain attributes in certain tuples of a relation:

  UPDATE <relation>

  SET <list of attribute assignments>

  WHERE <condition on tuples>;

Example: Update

- Change drinker Lars's phone number to 47 11 23 42:

```
UPDATE Drinkers
  SET phone = '47 11 23 42'
  WHERE name = 'Lars';
```

Example: Update Several Tuples

- Make 30 the maximum price for beer:

```
UPDATE Sells
  SET price = 30
  WHERE price > 30;
```

Summary 4

More things you should know:

- More joins
  - OUTER JOIN, NATURAL JOIN
- Aggregation
  - COUNT, SUM, AVG, MAX, MIN
  - GROUP BY, HAVING
- Database updates
  - INSERT, DELETE, UPDATE

Functional Dependencies

Functional Dependencies

- $X \rightarrow Y$ is an assertion about a relation R that whenever two tuples of R agree on all the attributes of X, then they must also agree on all attributes in set Y
  - Say "$X \rightarrow Y$ holds in R"
  - Convention: …, X, Y, Z represent sets of attributes; A, B, C,… represent single attributes
  - Convention: no set formers in sets of attributes, just ABC, rather than $\{A,B,C\}$

Splitting Right Sides of FD's

- $X \rightarrow A_1A_2 \ldots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2, \ldots,$ $X \rightarrow A_n$ hold for R
- Example: $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$
- There is no splitting rule for left sides
- We'll generally express FD's with singleton right sides

Example: FD's

Drinkers(name, addr, beersLiked, manf, favBeer)

- Reasonable FD's to assert:
  1. name → addr favBeer
     - Note: this FD is the same as name → addr and name → favBeer
  2. beersLiked → manf

Example: Possible Data

name	addr	beersLiked	manf	favBeer
Peter	Campusvej	Odense Cl.	Albani	Erdinger W.
Peter	Campusvej	Erdinger W.	Erdinger	Erdinger W.
Lars	NULL	Odense Cl.	Albani	Odense Cl.

Because name → addr

Because name → favBeer

Because beersLiked → manf

Keys of Relations

- *K* is a *superkey* for relation *R* if *K* functionally determines all of *R*
- *K* is a *key* for *R* if *K* is a superkey, but no proper subset of *K* is a superkey

Example: Superkey

Drinkers(name, addr, beersLiked, manf, favBeer)

- {name, beersLiked} is a superkey because together these attributes determine all the other attributes
  - name $\rightarrow$ addr favBeer
  - beersLiked $\rightarrow$ manf

Example: Key

- {name, beersLiked} is a key because neither {name} nor {beersLiked} is a superkey
  - name doesn't $\rightarrow$ manf
  - beersLiked doesn't $\rightarrow$ addr
- There are no other keys, but lots of superkeys
  - Any superset of {name, beersLiked}

Where Do Keys Come From?

1. Just assert a key K
   - The only FD's are $K \rightarrow A$ for all attributes A
2. Assert FD's and deduce the keys by systematic exploration

More FD's From "Physics"

- Example:
  "no two courses can meet in the same room at the same time" tells us:
  - hour room $\rightarrow$ course

Inferring FD's

- We are given FD's $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$,…, $X_n \rightarrow A_n$, and we want to know whether an FD $Y \rightarrow B$ must hold in any relation that satisfies the given FD's
  - Example:
    If $A \rightarrow B$ and $B \rightarrow C$ hold, surely $A \rightarrow C$ holds, even if we don't say so
- Important for design of good relation schemas

Inference Test

- To test if $Y \rightarrow B$, start by assuming two tuples agree in all attributes of Y

$$\overleftrightarrow{\;Y\;}$$

0000000. . . 0

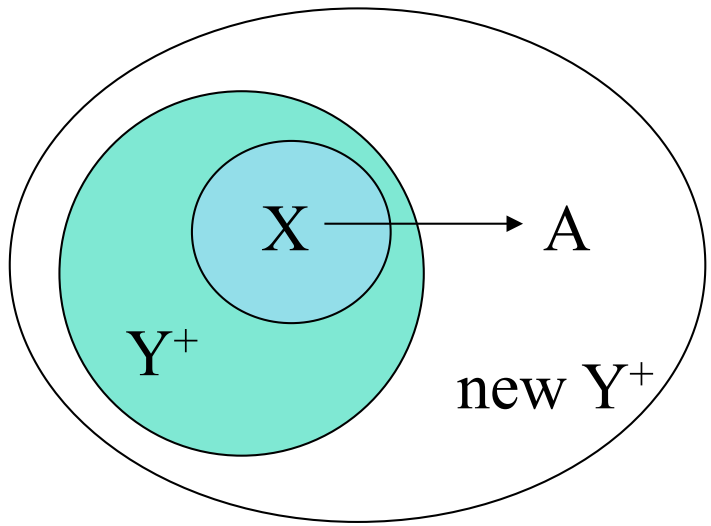
00000?? . . . ?

Inference Test

- Use the given FD's to infer that these tuples must also agree in certain other attributes
  - If B is one of these attributes, then $Y \rightarrow B$ is true
  - Otherwise, the two tuples, with any forced equalities, form a two-tuple relation that proves *Y -> B* does not follow from the given FD's

Closure Test

- An easier way to test is to compute the *closure* of Y, denoted Y^+
- Basis: $Y^+ = Y$
- Induction: Look for an FD's left side X that is a subset of the current Y^+
- If the FD is $X \rightarrow A$, add A to Y^+


X
A
Y+
new Y+

Finding All Implied FD's

- Motivation: "normalization," the process where we break a relation schema into two or more schemas
- Example: *ABCD* with FD's $AB \rightarrow C$, $C \rightarrow D$, and $D \rightarrow A$
  - Decompose into *ABC*, *AD*. What FD's hold in *ABC*?
  - Not only $AB \rightarrow C$, but also $C \rightarrow A$!

Why?

ABCD $a_1b_1cd_1$ $a_2b_2cd_2$

comes from

ABC a_1b_1c a_2b_2c

$d_1=d_2$ because $C \rightarrow D$

$a_1=a_2$ because $D \rightarrow A$

Thus, tuples in the projection with equal C's have equal A's $C \rightarrow A$

Basic Idea

1. Start with given FD's and find all *nontrivial* FD's that follow from the given FD's
   - Nontrivial = right side not contained in the left
2. Restrict to those FD's that involve only attributes of the projected schema

Simple, Exponential Algorithm

1. For each set of attributes X, compute X^+
2. Add $X \rightarrow A$ for all A in $X^+ - X$
3. However, drop $XY \rightarrow A$ whenever we discover $X \rightarrow A$
   - Because $XY \rightarrow A$ follows from $X \rightarrow A$ in any projection
4. Finally, use only FD's involving projected attributes

A Few Tricks

- No need to compute the closure of the empty set or of the set of all attributes
- If we find X^+ = all attributes, so is the closure of any superset of X

Example: Projecting FD's

- ABC with FD's $A \rightarrow B$ and $B \rightarrow C$
  Project onto AC:
  - $A^+ = ABC$; yields $A \rightarrow B$, $A \rightarrow C$
    - We do not need to compute AB^+ or AC^+
  - $B^+ = BC$; yields $B \rightarrow C$
  - $C^+ = C$; yields nothing
  - $BC^+ = BC$; yields nothing

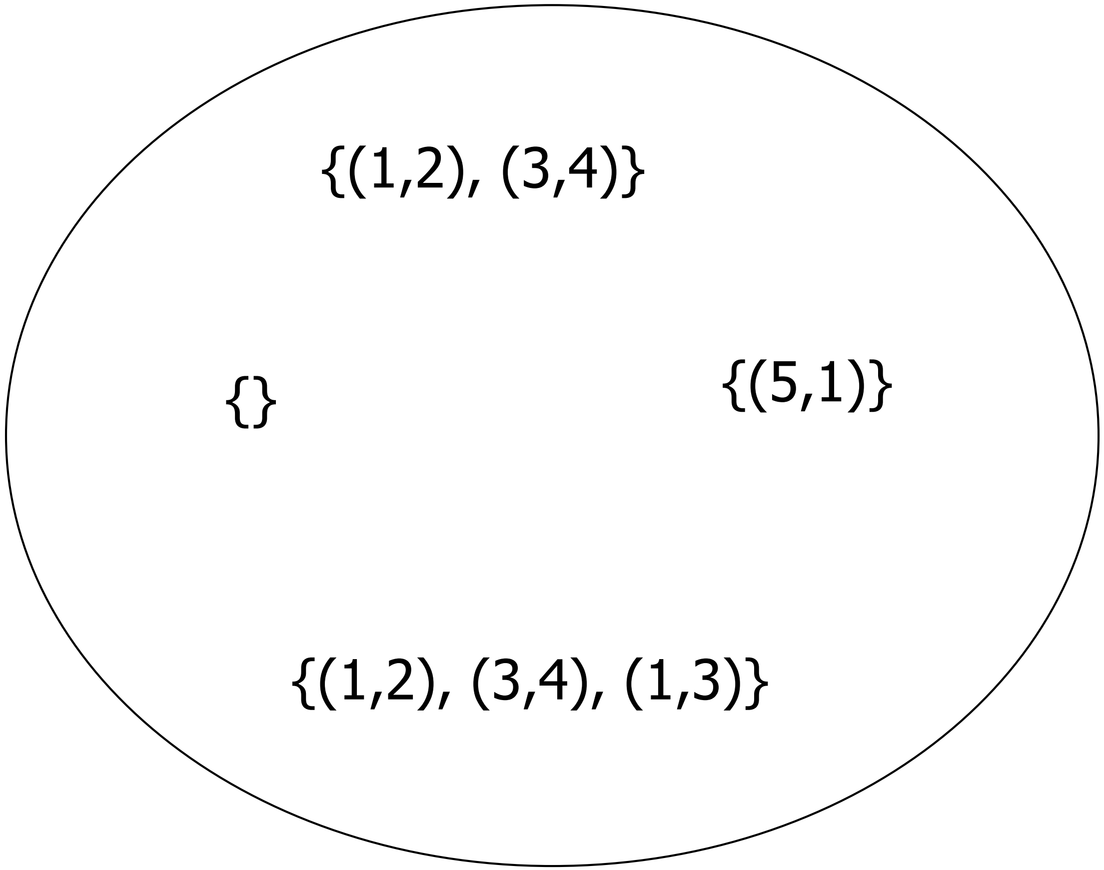
Example: Projecting FD's

- Resulting FD's: $A \rightarrow B$, $A \rightarrow C$, and $B \rightarrow C$
- Projection onto *AC*: $A \rightarrow C$
  - Only FD that involves a subset of $\{A,C\}$

A Geometric View of FD's

- Imagine the set of all *instances* of a particular relation
- That is, all finite sets of tuples that have the proper number of components
- Each instance is a point in this space

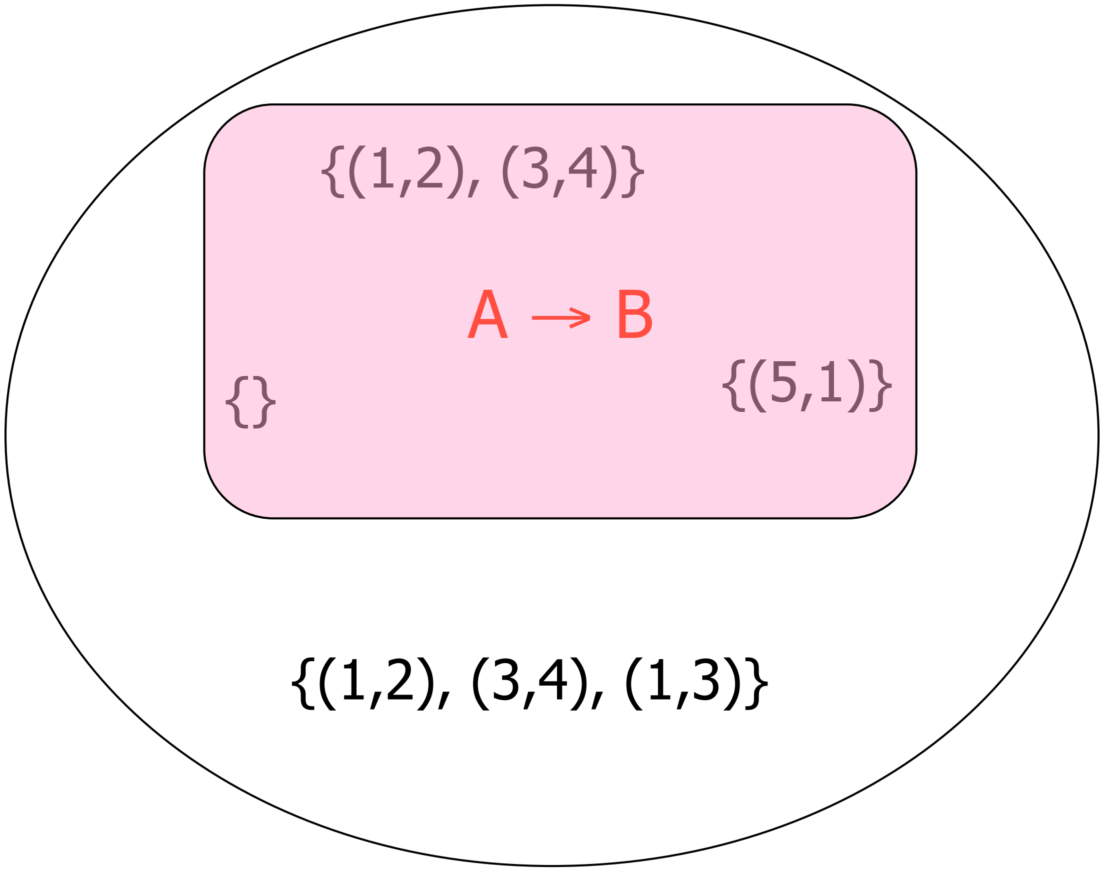
Example: R(A,B)



An FD is a Subset of Instances

- For each FD $X \rightarrow A$ there is a subset of all instances that satisfy the FD
- We can represent an FD by a region in the space
- Trivial FD = an FD that is represented by the entire space
  - Example: $A \rightarrow A$

Example: A → B for R(A,B)

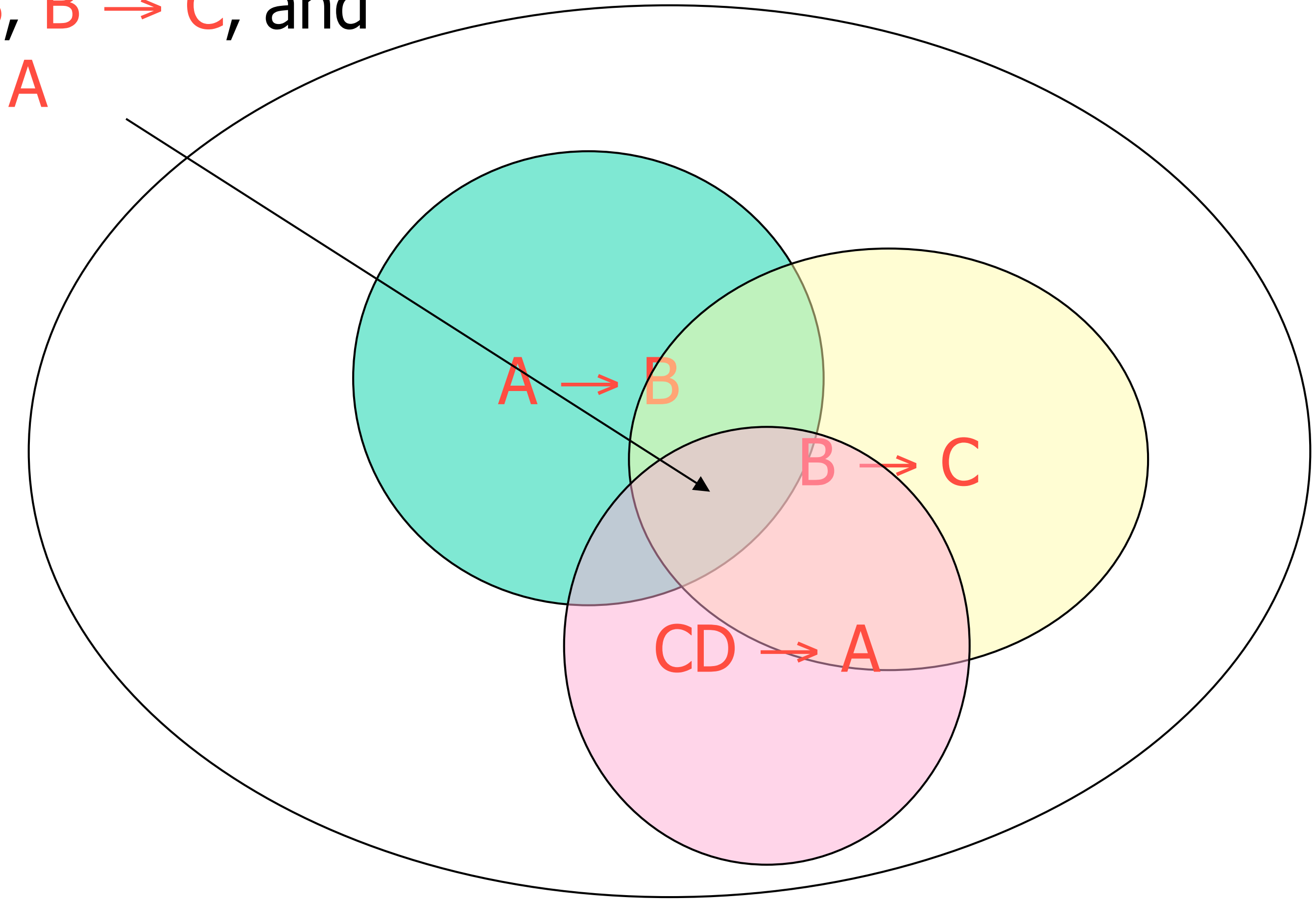


Representing Sets of FD's

- If each FD is a set of relation instances, then a collection of FD's corresponds to the intersection of those sets
  - Intersection = all instances that satisfy all of the FD's

Example

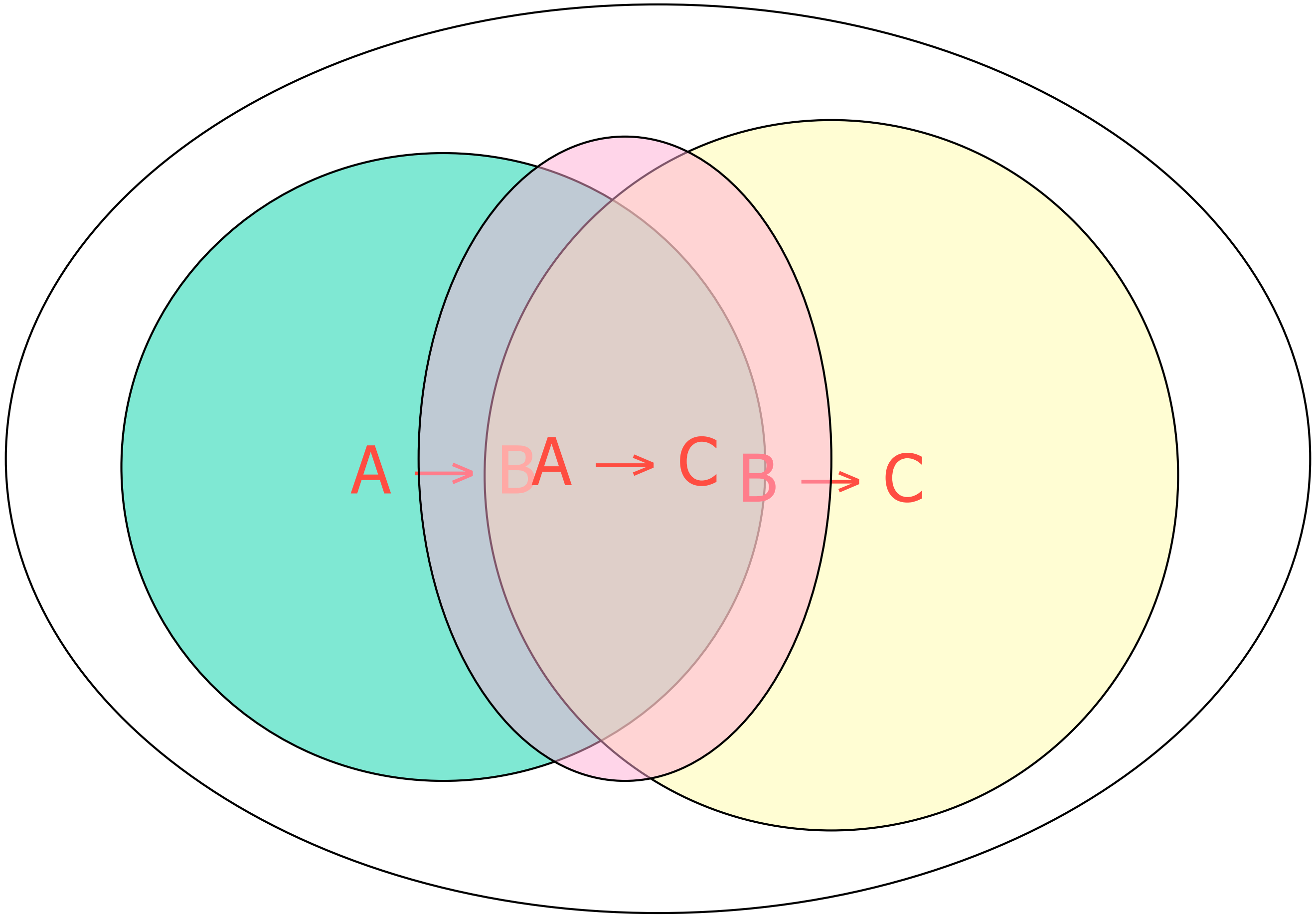
Instances satisfying
A → B, B → C, and
CD → A



Implication of FD's

- If an FD $Y \rightarrow B$ follows from FD's $X_1 \rightarrow A_1$, …, $X_n \rightarrow A_n$, then the region in the space of instances for $Y \rightarrow B$ must include the intersection of the regions for the FD's $X_i \rightarrow A_i$
  - That is, every instance satisfying all the FD's $X_i \rightarrow A_i$ surely satisfies $Y \rightarrow B$
  - But an instance could satisfy $Y \rightarrow B$, yet not be in this intersection

Example



Relational Schema Design

- Goal of relational schema design is to avoid anomalies and redundancy
  - *Update anomaly:* one occurrence of a fact is changed, but not all occurrences
  - *Deletion anomaly:* valid fact is lost when a tuple is deleted

Example of Bad Design

Drinkers(name, addr, beersLiked, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Peter	Campusvej	Odense Cl.	Alb.	Erdinger W.
Peter	???	Erdinger W.	Erd.	???
Lars	NULL	Odense Cl.	???	Odense Cl.

Data is redundant, because each of the ???'s can be figured out by using the FD's name $\rightarrow$ addr favBeer and beersLiked $\rightarrow$ manf

This Bad Design Also Exhibits Anomalies

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Peter	Campusvej	Odense Cl.	Alb.	Erdinger W.
Peter	Campusvej	Erdinger W.	Erd.	Erdinger W.
Lars	NULL	Odense Cl.	Alb.	Odense Cl.

- Update anomaly: if Peter moves to Niels Bohrs Alle, will we remember to change each of his tuples?
- Deletion anomaly: If nobody likes Odense Classic, we lose track of the fact that Albani manufactures Odense Classic

Boyce-Codd Normal Form

- We say a relation *R* is in *BCNF* if whenever $X \rightarrow Y$ is a nontrivial FD that holds in *R*, *X* is a superkey
  - Remember: *nontrivial* means *Y* is not contained in *X*
  - Remember, a *superkey* is any superset of a key (not necessarily a proper superset)

Example

Drinkers(name, addr, beersLiked, manf, favBeer)

FD's: name → addr favBeer, beersLiked → manf

- Only key is {name, beersLiked}
- In each FD, the left side is *not* a superkey
- Any one of these FD's shows *Drinkers* is not in BCNF

Another Example

Beers(<u>name</u>, manf, manfAddr)

FD's: name $\rightarrow$ manf, manf $\rightarrow$ manfAddr

- Only key is {name}
- Name $\rightarrow$ manf does not violate BCNF, but manf $\rightarrow$ manfAddr does

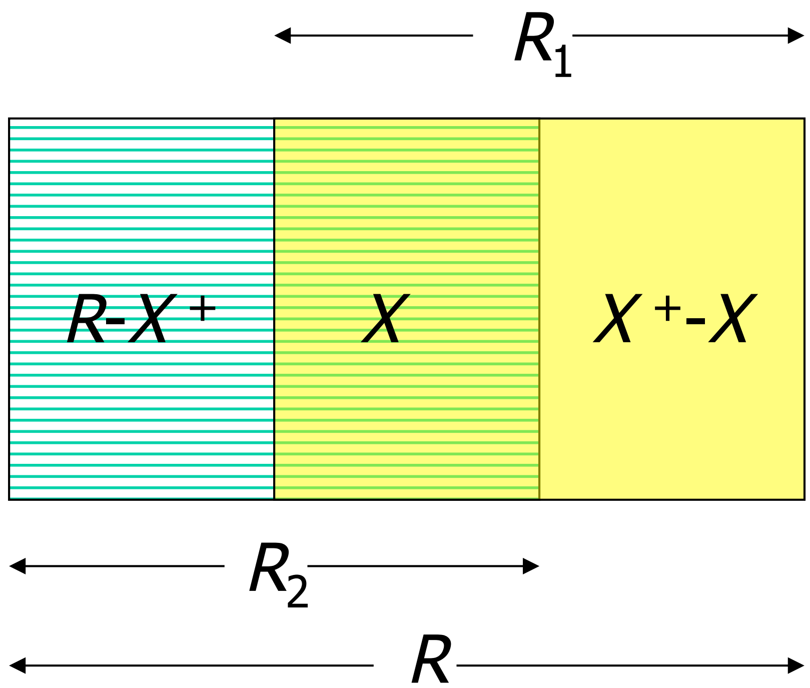
Decomposition into BCNF

- Given: relation R with FD's F
- Look among the given FD's for a BCNF violation $X \rightarrow Y$
  - If any FD following from F violates BCNF, then there will surely be an FD in F itself that violates BCNF
- Compute X^+
  - Not all attributes, or else X is a superkey

Decompose R Using $X \rightarrow Y$

- Replace R by relations with schemas:
  1. $R_1 = X^+$
  2. $R_2 = R - (X^+ - X)$
- *Project* given FD's F onto the two new relations

Decomposition Picture



Example: BCNF Decomposition

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

F = name → addr, name → favBeers
beersLiked → manf

- Pick BCNF violation name → addr
- Close the left side:
  $\{name\}^+ = \{name, addr, favBeer\}$
- Decomposed relations:
  1. Drinkers1(<u>name</u>, addr, favBeer)
  2. Drinkers2(<u>name</u>, <u>beersLiked</u>, manf)

Example: BCNF Decomposition

- We are not done; we need to check Drinkers1 and Drinkers2 for BCNF
- Projecting FD's is easy here
- For Drinkers1(<u>name</u>, addr, favBeer), relevant FD's are name $\rightarrow$ addr and name $\rightarrow$ favBeer
  - Thus, {name} is the only key and Drinkers1 is in BCNF

Example: BCNF Decomposition

- For Drinkers2(<u>name</u>, <u>beersLiked</u>, manf), the only FD is beersLiked $\rightarrow$ manf, and the only key is {name, beersLiked}
  - Violation of BCNF
- beersLiked$^+$ = {beersLiked, manf}, so we decompose *Drinkers2* into:

1. Drinkers3(<u>beersLiked</u>, manf)
2. Drinkers4(<u>name</u>, <u>beersLiked</u>)

Example: BCNF Decomposition

- The resulting decomposition of *Drinkers:*

  1. Drinkers1(name, addr, favBeer)
  2. Drinkers3(beersLiked, manf)
  3. Drinkers4(name, beersLiked)

  - Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like

- Compare with running example:

  1. Drinkers(name, addr, phone)
  2. Beers(name, manf)
  3. Likes(drinker,beer)

Third Normal Form – Motivation

- There is one structure of FD's that causes trouble when we decompose
- $AB \rightarrow C$ and $C \rightarrow B$
  - Example:
    A = street address, B = city, C = post code
- There are two keys, $\{A,B\}$ and $\{A,C\}$
- $C \rightarrow B$ is a BCNF violation, so we must decompose into AC, BC

We Cannot Enforce FD's

- The problem is that if we use AC and BC as our database schema, we cannot enforce the FD $AB \rightarrow C$ by checking FD's in these decomposed relations
- Example with A = street, B = city, and C = post code on the next slide

An Unenforceable FD

street	post
Campusvej	5230
Vestergade	5000

city	post
Odense	5230
Odense	5000

Join tuples with equal post codes

street	city	post
Campusvej	Odense	5230
Vestergade	Odense	5000

No FD's were violated in the decomposed relations and FD street city $\rightarrow$ post holds for the database as a whole

An Unenforceable FD

street	post
Hjallesevej	5230
Hjallesevej	5000

city	post
Odense	5230
Odense	5000

Join tuples with equal post codes

street	city	post
Hjallesevej	Odense	5230
Hjallesevej	Odense	5000

Although no FD's were violated in the decomposed relations, FD street city $\rightarrow$ post is violated by the database as a whole

3NF Let's Us Avoid This Problem

- 3rd Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation
- An attribute is *prime* if it is a member of any key
- $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime

Example: 3NF

- In our problem situation with FD's $AB \rightarrow C$ and $C \rightarrow B$, we have keys AB and AC
- Thus A, B, and C are each prime
- Although $C \rightarrow B$ violates BCNF, it does not violate 3NF

What 3NF and BCNF Give You

- There are two important properties of a decomposition:
  1. *Lossless Join:* it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original
  2. *Dependency Preservation:* it should be possible to check in the projected relations whether all the given FD's are satisfied

3NF and BCNF – Continued

- We can get (1) with a BCNF decomposition
- We can get both (1) and (2) with a 3NF decomposition
- But we can't always get (1) and (2) with a BCNF decomposition
  - street-city-post is an example