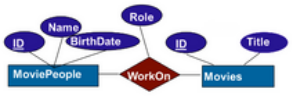
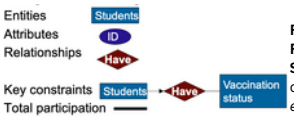
Conceptual Design

Entity: real-world object (ex. cat) described by attributes
Entity: An entity which can only be identified by considering key of another entity (a belongs to) relationship. In ER diagram, has THICK lines.
Entity Set: Collection of entities (ex. All cats)
- All entities in entity set have same set of attributes
- All entity sets have keys

Domain: value type (ex. float, date, int)
Key: minimal set of attributes which can identify an entity in entity set.
Primary Key: main key to identify entity in entity set. has to be as minimal as possible. (ex. cat_id)
Candidate key: One more kore keys in a relation
Super key: A key + zero or more additional attributes. (Ex. {sid, name}, or {CWL, major}, or {name, major, age})
Minimal Key: Smallest set of keys to identify entity in entity set.
Schema: The logic structure of database
Instance: The actual content of the database



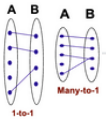


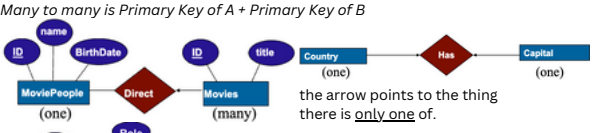
Relationship: association between two or more entities (ex. Tom Holland worked on Spiderman)
Relationship Set: collection of similar relationships. (ex. collection of all movie people who have worked in movies)
- Entities can be pat of multiple entity sets (ex. Tom Holland worked on Spiderman and isSonOf Tony Stark)
- Can have attributes (ex. Role)

N-ary relationship set: set **R** which relates n entity sets from E_1 ... E_n;
each relationship in R involves entities e_1 ∈ E_1, ..., e_n ∈ E_n
- Degree or "arity": # of entity sets in relationship (binary, ternary, etc)
- Ex. ternary relationship set (n=3). a "Supplies" relationship connects a Supplier entity, a Product entity, and a Store entity

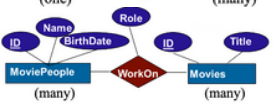
Cardinality ratio: the number of relationships in the set that an entity can participate in. These are called cardinality constraints:
Total Participation: every instance of entity must be involved in relationship
Ex. If every employee must belong to exactly one department, then we say that the "Employee" entity has total participation in the "Employee-Department" relationship. a double line connects the two entities.

Key Constraints
Key constraints are shown with **arrow** in ER diagram:
one-to-one: entity in A is associated at most once with one entity in B (ex. A: Student, B: student ID #)
The primary key of R (relation) is Primary Key of A + Primary Key of B
one-to-many: entity in A is associated with any number of entities in B (ex. A: Mom, B: children)
One to many (A is one, B is many) is primary key of B
many-to-one: opposite of one-to-many
One to many (B is one, A is many) is primary key of A
many-to-many: entities in A can be associated with many entities in B
Many to many is Primary Key of A + Primary Key of B





the arrow points to the thing there is only one of.

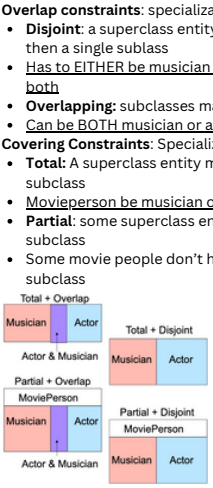


you can figure out the one from the many, but not the many from the one.

Participation Constraints: Whether a relationship has to be populated or not. Important for updates.
Employees----[Manages]<-----Departments
Ex. When deleting, a new department must still have a manager
ISA Relationships: If we declare A ISA B, every A entity is a B entity
Reasons for using ISA:
- Add descriptive attributes specific to subclass
- Restrict entities that participate in relationship

Some ISA constraints we can't express in ER diagram.
Overlap constraints: specializations can be
- **Disjoint**: a superclass entity belongs to no more then a single sublass
- Has to EITHER be musician or actor, cannot be both
- **Overlapping:** subclasses may overlap
- Can be BOTH musician or actor

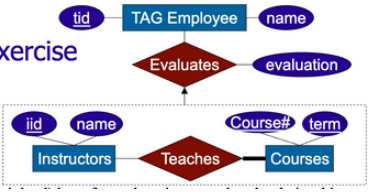
Covering Constraints: Specializations can be
- **Total:** A superclass entity must belong to some subclass
- Movieperson be musician or actor
- **Partial**: some superclass entity may not be in any subclass
- Some movie people don't have to be a part of a subclass



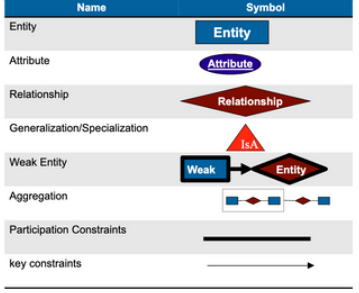
Choices for Good database design
entity vs. attribute
entity vs. relationship
binary or n-ary relationship
whether or not to use ISA hierarchies
whether or not to use aggregation

Aggregation:
Having a relationship between relationships is forbidden.

Aggregation allows us to treat relationship set as entity set, letting us participate in other relationships.



The (minimal) key of Evaluates is iid + course# + term.



Relational Models

Relational database: a set of relations
Relation: made up of 2 parts:
Schema : specifies name of relation, plus name and domain (type) of each attribute.
e.g., Student (sid: char[20], name: char[20], address: char[20], phone: char[20], major: char[20]).
Instance : a table, with rows and columns.
#Rows = cardinality
#Columns = arity / degree
Relational Database Schema: collection of schemas in the database
Database Instance: a collection of instances of its relations

Example of a Relation Instance

attribute, column name

relation name: Student

sid	name	address	phone	major
99111120	G. Jones	1234 W. 12th Ave., Van.	889-4444	CPSC
92001200	G. Smith	2020 E. 18th St., Van	409-2222	MATH
94001020	A. Smith	2020 E. 18th St., Van	222-2222	CPSC
94001150	S. Wang	null	null	null

tuple, row, record

domain value

- degree/arity = 5; Cardinality = 4,
- Order of rows isn't important
- Order of attributes isn't important (except in some query languages)

Student(sid: integer, name: char[20], address: char[20], phone: char[20], major: char[20])
Or, without the domains:
Student (sid, name, address, phone, major)

SQL Statements

```
CREATE TABLE Student
    (sid INTEGER ,
     name CHAR(20),
     address CHAR(30),
     phone CHAR(13),
     major CHAR(4))
```

```
CREATE TABLE Grade
    (sid INTEGER,
     dept CHAR(4),
     course# CHAR(3),
     mark INTEGER)
```

```
DROP TABLE Student
```
- Destroys relation student, schema AND tuples are deleted

```
ALTER TABLE STUDENT
    ADD COLUMN gpa REAL;
```
- Schema of students is altered with new attribute- every tuple in current instance is extended with null value in new attribute

```
INSERT
INTO Student (sid, name, address, phone, major)
VALUES ('52033688', 'G. Chan', '1235 W. 33, Van', '882-4444', 'PHYS')
```
- Insert a single Tuple

```
DELETE
FROM Student
    WHERE name = 'Smith'
```
- Delete tuples satisfying condition

Integrity Constraints

IC: condition that must be true for ANY instance of database; ex. domain constraints
- IC's specified when schema is defined
- IC's checked when relations are modified

Legal Instance of Relation satisfies all specified IC's
- DBMS does NOT allow illegal instances, can avoid data entry errors with this

Candidate keys are possible when
1. No distinct tuples can have the same values for all attributes in the key, and
2. No subset of S is itself a key (according to (1)).

- One of the possible keys is chosen (by the DBA) to be the ***primary key*** (PK).
- e.g.
  - {*sid, name*} is a superkey
  - ***sid*** is the primary key for Students

```
CREATE TABLE Student
    (sid     INTEGER PRIMARY KEY,
     name    CHAR(20),
     address CHAR(30),
     phone   CHAR(13),
     major   CHAR(4))
```

Key Constraints

Primary Key Constraint:
- values for primary key must be unique
- a primary key attributes cannot be null

For Unique Constraint:
- values for a group of attributes must be unique (if they are not null)
- these attributes can be null

Key constraints are checked when new values are inserted, or values are modified

(Ex.1- Normal) "For a given student and course, there is a single grade."

```
CREATE TABLE Grade
    (sid     INTEGER,
     dept    CHAR(4),
     course# CHAR(3),
     mark    INTEGER,
     PRIMARY KEY (sid,dept,course#) )
```

vs.

(Ex.2 - Silly) "Students can take a course once, and receive a single grade for that course; further, no two students in a course receive the same grade."

```
CREATE TABLE Grade2
    (sid     INTEGER,
     dept    CHAR(4),
     course# CHAR(3),
     mark    CHAR(2),
     PRIMARY KEY (sid,dept,course#),
     UNIQUE (dept,course#,mark) )
```

Referential Integrity

SQL/92 supports all 4 options on deletes and updates.
- Default is **NO ACTION** (*delete/update is rejected*)
- **CASCADE** (also updates/deletes all tuples that refer to the updated/deleted tuple)
- **SET NULL / SET DEFAULT** (referencing tuple value is set to the default foreign key value)

```
CREATE TABLE Grade
    (sid CHAR(8), dept CHAR(4),
     course# CHAR(3), mark INTEGER,
     PRIMARY KEY (sid,dept,course#),
     FOREIGN KEY (sid)
        REFERENCES Student(sid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
     FOREIGN KEY (dept, course#)
        REFERENCES
        Course(dept,course#)
        ON DELETE SET DEFAULT
        ON UPDATE CASCADE );
```

Translating Key Constraints

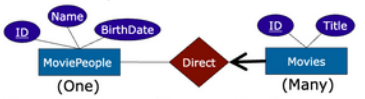
For **many-to-many** relationships, you create separate tables for each entity, and the relationship.
For **many-to-one** relationships you combine the tbale for the relationship on the many side with the table for the entity.
For **one-to-one** relationships, you pick one and combine it.

Assume you went with Country(coName, caName) and all attributes have type Char(20) and we're not creating a separate relation for Capital. Write the SQL DDL that you would need for this relation.

```
CREATE TABLE Country(
    country-name CHAR(20) PRIMARY KEY,
    capital-name CHAR(20),
    UNIQUE capital-name ← needed for one-to-one constraint
)
```

coName	caName
Canada	Ottawa
United States	Ottawa
Mexico	Ottawa

Translating Participation Constraints

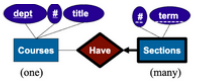


- Every movie must have a director.
  - Every tuple in the Movie table must appear with a non-null *MoviePeople ID* value
- How can we express that in SQL?

Using method 2 (add Directs relation in the Movie table), we can capture participation constraints by
- ensuring that each **MID** is associated with a **MPID** that is not null
- not allowing deletion of a director before the director is replaced

```
CREATE TABLE Directed_Movie(
    MID     INTEGER,
    title   CHAR(20),
    MPID    CHAR(11) NOT NULL,
    PRIMARY KEY (MID),
    FOREIGN KEY (MPID) REFERENCES
    MoviePeople
            ON DELETE NO ACTION
            ON UPDATE CASCADE)
```

Translating Weak Entity Sets



- A ***weak entity*** is identified by considering the primary key of the *owner* (strong) entity.
  - Owner entity set and weak entity set participate in a one-to-many identifying relationship set.
  - Weak entity set has total participation.

- Weak entity set and its identifying relationship set are translated into a single table (like many to one anyway)
  - Primary key would consist of the owner's primary key and weak entity's partial key
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE Course_Section (
    dept        CHAR(4),
    course_num  INTEGER,
    section_num INTEGER,
    term        CHAR(6)
    PRIMARY KEY (dept, course_num, section_num, term),
    FOREIGN KEY (dept, course_num) REFERENCES
                Courses(dept, num),
                ON DELETE CASCADE)
```

- What should be done if a **Student tuple** is deleted?
  - Also delete all Grade tuples that refer to it?
  - Disallow deletion of this particular Student tuple?
  - Set sid in Grade tuples that refer to it, to *null*, (the special value denoting *'unknown'* or *'inapplicable'*.)
    - problem if sid is part of the primary key
  - Set sid in Grade tuples that refer to it, to a *default sid*.

Only students listed in the Student relation should be allowed to have grades for courses that are listed in the Course relation.

```
CREATE TABLE Grade
    (sid INTEGER, dept CHAR(4), course# CHAR(3), mark INTEGER,
     PRIMARY KEY (sid,dept,course#),
     FOREIGN KEY (sid) REFERENCES Student,
     FOREIGN KEY (dept, course#) REFERENCES Course(dept, cnum))
```

Sometimes you can not specify which attributes are referenced, but in this case they are needed. Never hurts to include them!

Grade

sid	dept	course#	mark
53666	CPSC	101	80
53666	RELG	100	45
53650	MATH	200	null
53666	HIST	201	60

Student

sid	name	address	Phone	major
53666	G. Jones	....	...	...
53688	J. Smith	....	...	...
53650	G. Smith	....	...	...

```
CREATE TABLE Student
    (sid     INTEGER PRIMARY KEY,
     name    CHAR(20),
     address CHAR(30),
     phone   CHAR(13),
     major   CHAR(4))
```

Foreign Key Constraint
Foreign key: Set of attributes in one relation used to 'reference' a tuple in another relation.
- Must correspond to the primary key of the other relation.
- Like a 'logical pointer'.

Grade

sid	dept	cnum	grade
2	CPSC	304	90
2	MATH	221	90
2	EPSE	223	90
1	MUSC	103	90

```
CREATE TABLE Grade (
    sid INTEGER,
    dept  CHAR(4),
    course# CHAR(3),
    mark INTEGER,
    PRIMARY KEY (sid,dept,course#),
    FOREIGN KEY (sid) REFERENCES
        Student,
    FOREIGN KEY (dept, course#)
        REFERENCES Course(dept, cnum)
)
```

E.g.:
Grade(*sid, dept, course#, grade*)
- *sid* is a foreign key referring to Student:
- *(dept, course#)* is a foreign key referring to Course

Referential integrity: All foreign keys reference existing entities.
- i.e. there are no dangling references
- all foreign key constraints are enforced

Normal Forms

Functional dependency: one attribute determines another through a functional dependency.
Ex. If Department determines Address but not Name, we say that there's a functional dependency from Department to Address. But Department is NOT a key.
X determines Y.
if t1.X = t2.X then t1.Y = t2.Y
It is a statement about all allowable instances.

- Must be identified by application semantics
- Given some instance r1 of R, we can check if r1 violates some FD f, but cannot tell if f holds over R.

Examples of Functional Dependencies

Trivial case is where:
FD: PostalCode -> Province
So PostalCode, House # -> Postal Code

Given some FDs, we can often infer additional FDs:

Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
e.g., city,major→city
Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
e.g., if sid→city, then sid,major → city,major
Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
sid→ city, city→ areacode implies *sid→ areacode*
Union: If X→Y and X→Z, then X→Y Z
e.g., if sid→acode and sid→city, then sid→acode,city
Decomposition: If X→Y Z, then X→Y and X→Z
e.g., if sid→acode,city then sid→acode, and sid→city

A couple of additional rules (that follow from axioms):

- ***Union:*** If X→Y and X→Z, then X→Y Z
  e.g., if sid→acode and sid→city, then sid→acode,city
- ***Decomposition***: If X→Y Z, then X→Y and X→Z
  e.g., if sid→acode,city then sid→acode, and sid→city

Union: If X→Y and X→Z, then X→Y Z
e.g., if sid→acode and sid→city, then sid→acode,city

1. X→Y Given
2. X→Z Given
3. X→XY 1, augmentation
4. XY→ZY 2, augmentation
5. X→ZY 3, 4, transitivity

Decomposition: If X→Y Z, then X→Y and X→Z
e.g., if sid→acode,city then sid→acode, and sid→city

1. X→YZ Given
2. YZ→Y Reflexivity
3. YZ→Z Reflexivity
4. X→Y 1, 2, transitivity
5. X→Z 1, 3, transitivity

Show that (sname, p#) is a superkey of SupplierPart(sname,city,status,p#,pname,qty)
Proof has two parts:

fd1: sname → city
fd2: city → status
fd3: p# → pname
fd4: sname, p# → qty

a. Show: sname, p# is a (super)key

1.	sname, p# → sname, p#	reflex
2.	sname → city	fd1
3.	sname → status	2, fd2, trans
4.	sname,p# → city, p#	2, aug
5.	sname,p# → status, p#	3, aug
6.	sname,p# → sname, p#, status	1, 5, union
7.	sname,p# → sname, p#, status, city	4, 6, union
8.	sname,p# → sname, p#, status, city, qty	7, fd4, union
9.	sname,p# → sname, p#, status, city, qty, pname	8, fd3, union

b. Show: (sname, p#) is a *minimal* key of SupplierPart(sname,city,status, p#,pname,qty)

1. p# does not appear on the RHS of any FD therefore except for p# itself, nothing else determines p#
3. specifically, sname → p# does not hold
4. therefore, sname is not a key
5. similarly, p# is not a key

fd1: sname → city
fd2: city → status
fd3: p# → pname
fd4: sname, p# → qty

Specifically, for combinations of sname and p#:

- sname,p# →sname, p#, city, status, pname, qty
- sname → sname, city, status
- p# → p#, pname

30

- Scared you're going to mess up? *Closure* is a fool-proof method of checking FDs and finding implicit FDs.
- Closure for a set of attributes X is denoted X^+
- X^+ includes all attributes of the relation IFF X is a (super)key
- Algorithm for finding Closure of X:
  Let Closure = X
  Until Closure doesn't change do
  if $a_1, \ldots, a_n \rightarrow C$ is a FD and $\{a_1, \ldots, a_n\} \in$ Closure
  Then add C to Closure

fd1: sname → city
fd2: city → status
fd3: p# → pname
fd4: sname, p# → qty

SupplierPart(sname,city,status,p#,pname,qty)

Ex: sname,p#⁺ = {sname, p#, city, status, pname, qty} <-- superkey
sname⁺ = {sname, city, status}
p#⁺ = {p#, pname}

So seeing if a set of attributes is a key means checking to see if it's closure is all the attributes – pretty simple

31

Normalization

Normalization is the process of removing redundancy from data
Four important normal forms:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Boyce-Codd Normal Form (BCNF)
- Third Normal Form (3NF)
- If a relation is in a certain normal form, certain problems are avoided/minimized
- Normal forms can help decided whether decomposition (splitting tables) will help

1NF
Each attribute in a tuple has only one value (can't be an array).
E.g., for "postal code" you can't have both V6T 1Z4 and V6S 1W6

Why do we need it? Codd's original version allowed multi-valued attributes

Client ID	Postal Code
1	V6T 1Z4, V6S 1W6

Can't have multiple values in a single attribute

Normalize to 1NF

Client ID	Postal Code
1	V6T 1Z4
1	V6S 1W6

2NF
There are no partial key dependencies.
A relation is in 2NF if it is in 1NF and for every FD, X -> Y where X is a (minimal) key and Y is a non-key attribute, then no proper subset of X determines Y.

Ex. This address relation is not in 2NF

e.g., the address relation is not in 2NF:

- House#, street, postal_code is a (minimal) key
- House#, street, postal_code → Province
- Postal_code → Province

X, Y, Subset of X

Not in 2NF

2NF
There are no partial key dependencies.
A relation is in 2NF if it is in 1NF and for every FD, X -> Y where X is a (minimal) key and Y is a non-key attribute, then no proper subset of X determines Y.

Ex. This address relation is not in 2NF

Boyce-Codd Normal Form (BCNF)
A relation R is in BCNF if X -> b is a non-trivial dependency in R, then X is a superkey for R.

Ex. Whenever a set of attributes R determine another attribute, it should determine all the attributes of R.

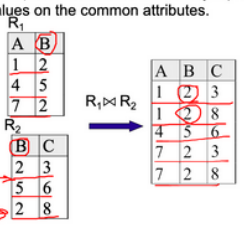
Check if all the left parts of the FD are a superkey. If they are, then it is in BCNF, and if not, they need to be decomposed.

Decomposition

Decomposition of R replaces R by two or more relations.

- Each new relation contains a subset of the attributes of R (and no attributes not appearing in R)
- Every attribute of R appears in at least one new relation.

Definition: $R_1 \bowtie R_2$ is the (natural) join of the two relations; i.e., each tuple of R_1 is concatenated with every tuple in R_2 having the same values on the common attributes.



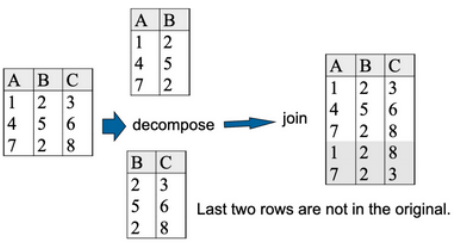
67

Lossless-Join Decomposition

Informally: If we break a relation, R, into bits, when we put the bits back together, we should get exactly R back again

Formally: Decomposition of R into X and Y is lossless-join w.r.t. a set of FDs F if, for every instance r that satisfies F:

- If we JOIN the X-part of r with the Y-part of r the result is exactly r
- It is *always* true that r is a subset of the JOIN of its X-part and Y-part, even if the join isn't lossless
- In general, the other direction does not hold – you may get back additional information! If it does hold, the decomposition is a lossless-join.
- Note: The word loss in lossless refers to loss of information, not to loss of tuples. In fact, for "loss of information" a better way to refer to it might be "addition of spurious information".



Finding Minimal Keys

Ex. Find all minimal keys for R(ABCD) where FDs are AB -> C and C -> BD.

1. List only attributes that appear on the RHS of the FDs (only what can be derived). Put on the right.

Left	Middle	Right
		D

2. List all the attributes that only ever appear on the LHS of an FD (or do not appear in the FD at all, ex. things that have to be a part of any key). put on the left.

Left	Middle	Right
A		D

3. List attributes that appear on the LHS and the RHS of the FDs (not obviously required and may help you derive). Put in middle

Left	Middle	Right
A	BC	D

4. Take the closure of the attributes in the left column.
{A}+ = {A}
Are all of the attributes there? If so, you have found a minimal key. If not, start adding in attributes from the middle column to see if you can determine all the attributes of the relation.

Examples of 3NF

A Relation R is in 3NF if:
If X -> b is a non-trivial dependency in R, then X is a superkey for R **or B is part of a (minimal) key.**

Note: b must be part of a key not part of a superkey (if a key exists, all attributes are part of a superkey)
Example:
R(Unit, Company, Product)
Unit -> Company
Company, Product -> Unit
Keys: {Company, Product}, {Unit, Product}

Rule: for all non-trivial functional dependencies in a relation R of the form X->b, it must be the case that X is a superkey of R or **b is part of a key.**

Minimal Cover G for a set of FDs F:

- Closure of F = closure of G (i.e., imply the same FDs)
- Right hand side of each FD in G is a single attribute
- If we delete an FD in G or delete attributes from an FD in G, the closure changes

e.g., A→B, ABCD→E, EF→GH, ACDF→EG has the following minimal cover:

- A→B, ACD→E, EF→G and EF→H

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Example:
A→B, ABCD→E, EF→G, EF→H, ACDF → EG

- Replace last rule with
  - ACDF → E
  - ACDF → G

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Example:
A→B, ABCD→E, EF→G, EF→H, ACDF → E, ACDF → G

- Can we take anything away from the LHS?
  - ACD^+= ABCDE, (crucially includes B) so remove B from the FD

1. Put FDs in standard form (have only one attribute on RHS)
2. Minimize LHS of each FD
3. Delete Redundant FDs

Example:
A→B, ACD→E, EF→G, EF→H, ACDF → E, ACDF → G

- Let's find $ACDF^+$ **without** considering the highlighted FDs
  - $ACDF^+$= ACDFEBGH, so I can remove the highlighted rules
- Final answer: A→B, ACD→E, EF→G, EF→H

Examples of BCNF

Remember BCNF def: For all non-trivial functional dependencies X→b, X must be a superkey for a relation to be in BCNF
Relation: R(ABCD) FD: B→C, D→A
Keys?
A^+ = {A}
B^+ = {B,C}
C^+ = {C}
D^+ = {A,D}
BD^+ = {B,D,C,A}
BD is the only key
Look at FD B→ C. Is B a superkey?
No. Decompose
R1(B,C), R2(A,B,D)
Look at FD D→ A. Is D a superkey for R2?
No. Decompose
R3(D,A), R4(D,B)
Final answer: R1(B,C), R3(D,A), R4(D,B)

X→b
AD B C

X→b
B D A

Let R be a relation with attributes A, and FD be a set of FDs on R s.t. all FDs determine a single attribute

Pick any f ∈ FD that violates BCNF of the form X→b
Decompose R into two relations: R_1(A-b) & R_2(X ∪ b)
Recurse on R_1 and R_2 using FD

Pictorally: