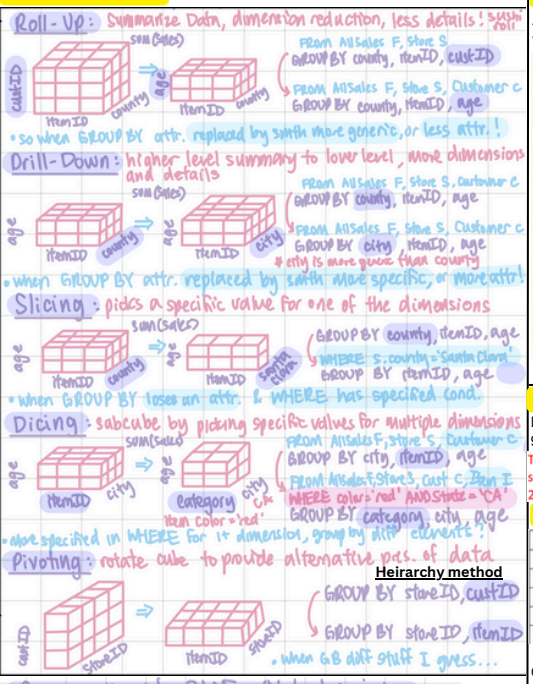
CPSC 304 FINAL EXAM



Decomposing into 3NF: #1 lossless join

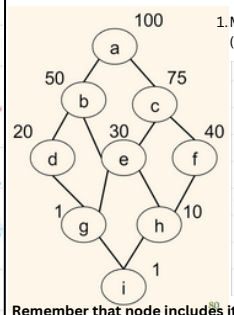
1. Given FDs F, compute F': minimal cover for F
2. Decompose using F' if violating 3NF similar to BCNF - all the way to BCNF
3. After each decomp, identify set of dependencies N in F' that are not preserved by the decomp.
4. At end, for each X→b in N create a relation R_n (X ∪ b) & add it to the decomposition

Valid: The patterns hold in general.
Novel: We did not know the pattern beforehand
Useful: We can devise actions from the patterns (business intelligence).
Understandable: We can interpret and comprehend the patterns

Data mining is the exploration and analysis of large quantities of data in order to discover valid, novel, potentially useful, and ultimately understandable patterns in data.

HRU Algorithm

Assuming 'a' is already materialized, what are the best 3 other views that we should materialize?



1. Make table of values. "Gain" is the top (a) subtracted with the current one.

	First Choice	Second Choice	Third Choice
b	**50*6=300**	-	-
c	25*6=150	25*2=50	25*1=25
d	80*3=240	30*3=90	30*3=90
e	70*4=280	20*4=80	20*4=80
f	60*3=180	**60*3=120**	-
g	99*2=198	49*2=98	**49*2=98**
h	90*2=180	40*2=80	30*2=60
i	99	49	39

Remember that node includes itself, and once you've made a choice, you count the subtraction from that point onwards.

Data warehousing: Finding number of tuples

Data warehouse with dimensions D1, D2, D3, D4, D5. They have 999, 99, 24, 4, and 19 values, . Sparsity is 10%. What's est. number of tuples?
The size of the full cube is (999+1)(99+1)(24+1)(4+1)(19+1) = 250,000,000 tuples. With a sparsity factor of 10%, the size of the sparse cube is about 250,000,000 x 0.1 = 25,000,000 tuples.

Apriori Algorithm

minimum support: 33%.

T1	tree, cup, paper
T2	book, tomato, pen
T3	pen, book, tree
T4	tomato, pen, cup
T5	tree, paper, book
T6	pen, book

1. **Find possible itemsets, and support of size C_k denoted as**

C_1 = {tree}: 3, {cup}: 2, {paper}: 2, {book}: 4, {tomato}: 2 {pen}: 4 }

2. **Find frequent itemsets of size k, denoted as F_k**

F_1 = {{tree},{cup}, {paper}, {book}, {tomato}, {pen}}
It's these values because they occur more then 33% (2/6 times).

2. **Go up one k, and ONLY use possible combinations from F_{k-1}.**

C_2 = {{tree, cup}: 1, **{tree, paper}: 2**, **{tree, book}: 2**, {tree, tomato}: 0, {tree, pen}: 1, {cup, paper}: 1, {cup, book}: 0, {cup, tomato}: 1, {cup, pen}: 1, {paper, book}: 1, {paper, tomato}: 0, {paper, pen}: 0, {book, tomato}: 1, **{book, pen}: 3**, **{tomato, pen}: 2}**

F_2 = {{tree, paper}, {tree, book}, {book, pen}, {tomato, pen}}

repeat until it's not possible anymore. Apriori will finally return all the items meeting the minimum support.

Apriori speeds up calculating association rules based on the observation that each subset of a frequent itemset must also be a frequent itemset Ex. rice only appears one time, it can't appear two or more times with anything else.

Relational Algebra Examples

Find name of actors in Jaws and Spongebob "Joining with Relation"

Jaws ← $\pi_{name}((\sigma_{title="Jaws"}(Movie)) \bowtie StarsIn) \bowtie MovieStar$
Spongebob ← $\pi_{name}((\sigma_{title="Spongebob"}(Movie)) \bowtie StarsIn) \bowtie Moviestar$
Res ← Jaws ∩ Spongebob

Find name of actors in all movies "Has to be connected in some way"

$\pi_{name}(\pi_{starID, movieID}(StarsIn) / \pi_{movieID}(Movie)) \bowtie Moviestar$
"all"

Find all profs in exactly all the commitees prof piper is in

R2 ← $\pi_{commname}(\sigma_{profname='piper'}(committee))$ committee piper is in
R3 ← $\pi_{commname}(committee) - R2$ committee piper is not in
$committee / R2 - \pi_{profname}(committee \bowtie R3)$
profs touching piper's comitee — determine which profs are on committees from R3

Find all profs who have offices in at least all the buildings that Piper has offices in

$\pi_{profname, building}(professor \bowtie department) / \pi_{building}(\sigma_{profname='piper'}(professor) \bowtie department)$
"all"

For each person, find all pizzas the person eats that aren't served by any pizzeria the Person frequents

Eats ← $\pi_{name, pizza}(Frequents \bowtie Serves)$
IF for each, has to do with domain — Eats(name, pizza) — same domain — Pizzerias the person frequents — Frequents(name, pizzaria) — Serves(pizza, pizzaria, price)

Find names of people who frequent only pizzarias serving at least one pizza they eat.

all the pizzarias someone frequents − without all the pizzarias serving pizza someone eats

$\pi_{name}(Person) - \pi_{name}(Frequents - \pi_{name, pizzaria}(Eats \bowtie Serves))$

Find names of people who frequent all the pizzarias serving at least one pizza they eat.

$\pi_{name}(Person) - \pi_{name}(\pi_{name, pizzaria}(Eats \bowtie Serves) - Frequents)$

The most expensive pass cost

$\pi_{cost}(Pass) - \pi_{cost}(P_{cost \to newcost}(Pass) \bowtie newcost < cost(Pass))$

The min price book

$\pi_{isbn}(Book) - \pi_{b1.isbn}(\sigma_{b1.price > b2.price}(\rho(b1, Book) \times \rho(b2, Book)))$
Book w/ smallest price — books with a greater price then at least one other book

Find Pizzaria serving cheapest pepperoni pizza. For ties, return all of cheapest pepperoni pizzas.

izzaria ($\sigma_{pizza=pepperoni(Serves)}$) − $\pi_{s1.pizzaria}(\sigma_{Note}(P(serves, s1) \times P(serves, s2))$

SQL Examples

Find students who've taken all classes (division example)

1) With EXCEPT & NOT EXISTS

```
SELECT sname
FROM student s
WHERE NOT EXISTS ((SELECT c.name          -- all course names
                   FROM class c)
                  EXCEPT
                  (SELECT e.cname         -- courses student is enrolled in
                   FROM enrolled e
                   WHERE e.snum = s.snum))
```

courses student is not enrolled in

2) Without EXCEPT using NOT EXISTS

```
SELECT sname
FROM student s
WHERE NOT EXISTS(SELECT c.name
                 FROM class c
                 WHERE NOT EXISTS (SELECT e.snum
                                   FROM enrolled e
                                   WHERE c.name = e.cname
                                   and e.snum = s.snum))
```

Find those majors for which their average age is the minimum over all majors

```
CREATE VIEW Temp(major, average) AS
    SELECT S.major, AVG(s.age) AS average
    FROM Student S
    GROUP BY major
```
grouping by major, getting average age

```
SELECT major, average
FROM Temp
WHERE average = (SELECT min(average)
                 FROM Temp)
```
selecting minimum over all majors

Get all students using natural join

```
SELECT *
FROM student s, enrolled e
WHERE s.snum = e.snum
```

```
SELECT *
FROM Student S NATURAL LEFT OUTER JOIN
Enrolled E
```

Find the names of all movie stars who've been in a movie

```
SELECT DISTINCT Name
FROM StarsIn s, Moviestar m
WHERE s.starsid = m.starid
```
Need distinct or else can get movie star name duplicates!

Find the departments that have more then one faculty member

```
SELECT DISTINCT f1.deptid
FROM Faculty f1, Faculty f2
WHERE f1.fid <> f2.fid AND f1.deptid = f2.deptid
```
seeing if different faculties have same department.

Find IDs of MovieStars who've been in a movie in 1944 or 1974

```
SELECT StarId
FROM Movie M, StarsIn S
WHERE M.MovieID = S.MovieID AND Year = 1944
UNION ALL
SELECT StarId
FROM Movie M, StarsIn S
WHERE M.MovieID = s.movieID AND year=1974
```
For 1944 **and** 1974 use **INTERSECT**
UNION ALL keeps duplicates
equivalent to using year=1944 OR year=1974

Find the name & Age of the oldest student(s)

```
SELECT sname, age
FROM Student s2
WHERE NOT EXISTS (SELECT *
                  FROM student s1
                  WHERE s1. age > s2.age)
```
Find an s2 where there is no s1 older than them

Can also be written with any/all instead of exist/not exist

```
SELECT sname, age
FROM Student s2
WHERE s2.age >= ANY (SELECT age
                     FROM student s1)
```

Or with aggregate operators

```
SELECT sname, age
FROM Student s2
WHERE s.age = (SELECT MAX(s2.age) FROM Student S2)
```

Aggregate Operations

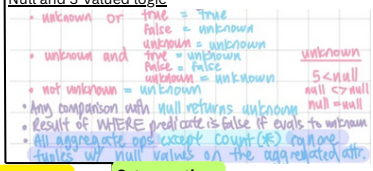
AVG, MIN, MAX, SUM, COUNT

```
SELECT country, city, count(*)
FROM customers
WHERE country LIKE '%a%'
GROUP BY country, city
HAVING count(*) < 3
ORDER BY count(*) DESC
```
this is for checking individual values
HAVING is for filtering sums/maxes/mins etc, is like a where condition for group by

Null Values tuples can have null values. unknown or doesn't exist

- can use IS NULL (IS NOT NULL) in WHERE or others

Null and 3-valued logic



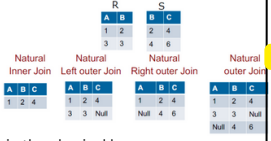
LIKE is used for string matching

- _ stands for any one character
- % is 0 or more arbitrary characters

Set operations

- **UNION** usually used for **or**
- **INTERSECT** usually used for **and**
- **EXCEPT** usually used for **A but didn't B**
- **'have not been in'**

Natural Join
Default is INNER JOIN – only include matches.
FROM Student S NATURAL LEFT OUTER JOIN Enrolled E



Views

Relations defined w/ a create table, statement existing in the physical layer

- Hide data from users, make queries easier, modularity

```
CREATE VIEW CourseWithFalls(dept, course #, mark) AS
    SELECT c.dept, c.course#, mark
    FROM Course C, Enrolled E
    WHERE c.course# = e.course# and mark < 50
```

View updates must occur at base tables.

- DBMS restrict view updates only to some simple views on single tables (updatable views)

DROP VIEW <view name> does NOT affect any tuples in underlying relation
We can assign commands to DROP table for view.
DROP TABLE student RESTRICT/CASCADE

- drop table unless there's a view on it
- drop table & recursively drops any view referencing it

Updating with condition

```
update SuppProd
  set Price = Price * 0.9
  where PID in
    (select PID
     from Product
      where Name = 'Coronary Stent');
```

Relational Algebra Part 2

Find all the professors who are in any of the committee's professor Piper is in

```
SELECT DISTINCT profname
FROM Committee
WHERE commname IN (SELECT *
                   FROM Committee
                   WHERE profname = 'Piper')
```

RA Statement: $\pi_{c2.profname}(\rho_{c1.profname='piper'}(committee) \times \rho_{c2.profname=c1.profname}(committee))$

Find all professors who are in at least all those commitees that professor Piper is i

```
SELECT DISTINCT c.profname
FROM Committee c
WHERE NOT EXISTS (SELECT commname
                  FROM Commitee a
                  WHERE profname = 'Piper')
                  EXCEPT
                  (SELECT commname
                   FROM committee a
                   WHERE a.profname=c.profname)
```

committee / $\pi_{commname}(\sigma_{profname='piper'}(committee))$

Find all enclosures that were never visited by the visitors born before 2000-01-01

```
SELECT E.id
FROM Employee E
EXCEPT
(SELECT EZ.id
 FROM Visitor v, Visits vs, Enclosure EZ
 WHERE v.visitorID = vs.passID AND vs.enclosureID = ez.id
 AND v.DateOfBirth < Date('2001-01-01')
```

Visitor(visitorID, firstName, lastName, dateofBirth)
Visits(passID, enclosureID)
Enclosure(id, latitude, longitude)

Find all the animals of species cat that are taken care of by 'Sam Smith'

```
SELECT employeeID
FROM TakesCareOf NATURAL INNER JOIN Employee
WHERE animalID IN (SELECT id from Animal WHERE Species=`Cat`) AND
                   firstName = 'Same' AND lastName = 'Smith'
```

Find the names of sailors who have reserved at least 2 boats

```
SELECT s.sname
FROM Sailors s, Reserves r1, Reserves r2
WHERE r1.sid=s.sid AND r2.sid=s.sid AND r1.bid <> r2.bid
```

Find the sailor id of the sailors with the highest rating

```
SELECT s.sid
FROM sailors s
WHERE s.rating >= (SELECT MAX(s2.rating) FROM Sailors s2)
```

Find names of sailors who have reserved all boats whose name starts w/ "typhoon'

```
SELECT s.sname
FROM Sailors s
WHERE NOT EXISTS (SELECT b.bid FROM Boats b WHERE name LIKE "%typhoon%"
                  EXCEPT
                  (SELECT r.bid FROM Reserves r WHERE r.sid = s.sid))
```
"no boat w/ typhoon that we have not reserved"

Find the name & age of the oldest sailor

```
SELECT s.sname, s.age
FROM Sailor s
WHERE s.age = (SELECT MAX(s2.age) FROM Sailor s2)
```

Loss-less join

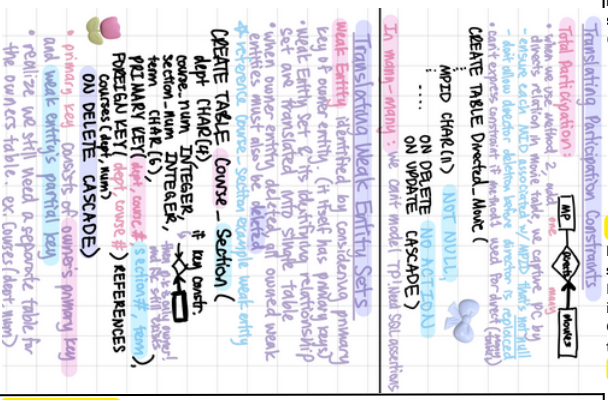
When we rejoin the decomposed tables, no loss of information (no more or less tuples)

How to check lossless: common attributes have to be a superkey in X1 or X2
- check common attr. of X1 & X2, ex. BC
- $BC^+ = \{BCS\}$ which is not superkey for X1 or X2 so this is lossy

Conceptual Design

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). MUST BE UNIQUE, and NON-NULL.
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
Cardinality ratio: the number of relationships in the set that an entity can participate in. These are called cardinality constraints:



Key Constraints

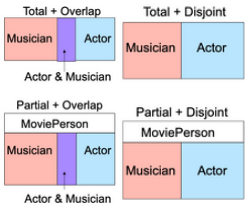
Key constraints are shown with **arrow** in ER diagram:

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 OR Primary Key of B
one-to-many: entity in A is associated with any numb
(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 m
Many to many is Primary Key of AND Primary Key of

Overlapping Constraints

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

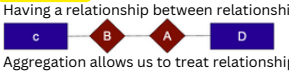


Participation Constraints

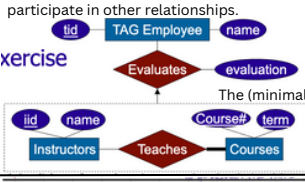
Whether a relationship has to be populated or not.
Important for updates.
Employees---->[Manages]<-----Departments
Ex. When deleting, a new department must still have a manager

Aggregation

Having a relationship between relationships is forbidden.

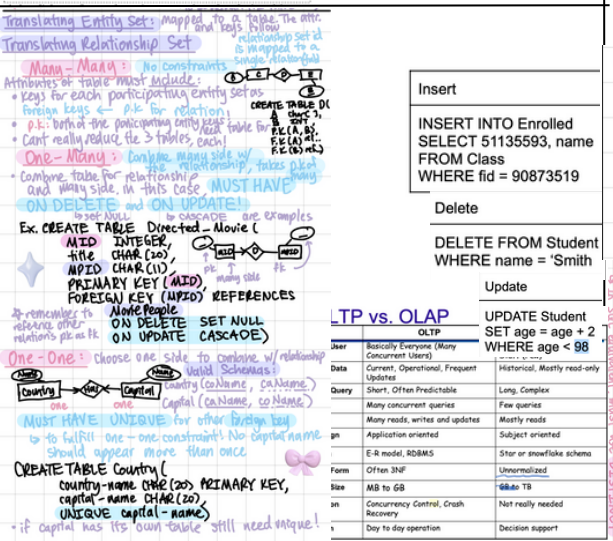


Aggregation allows us to treat relationship set as entity set, letting us participate in other relationships.



closure is that given a set, what else can it get?

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



Normalization

Removing redundancy from data.

First Normal Form (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
Codd's original version allowed multi-valued attributes.

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

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

In 1NF.

Second Normal Form (2NF)

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.**
e.g., the address relation is not in 2NF:
- House#, street, postal_code is a (minimal) key
- House#, street, postal_code → Province (Subset of X)
- Postal_code → Province

Not in 2NF

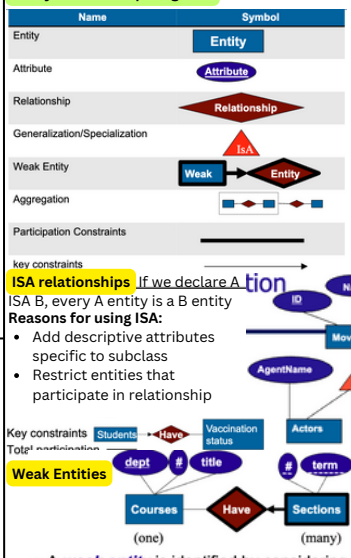
Boyce-Codd Normal Form (BCNF)

For all non-trivial functional dependencies X->b, X must be a superkey for a relation to be in BCNF.
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.

Third Normal Form (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:
Each CK has 3 attributes but each FD has 2, so it's not in BCNF.
3NF?
AB → C AB is not a CK but C is part of a CK; so, 3NF is good so far.
CD → E CD is not a CK, but E is part of a CK; therefore, 3NF, so far
DE → B DE is not a CK, but B is part of a CK
Conclusion: 3NF
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.**

Entity Relationship Diagrams

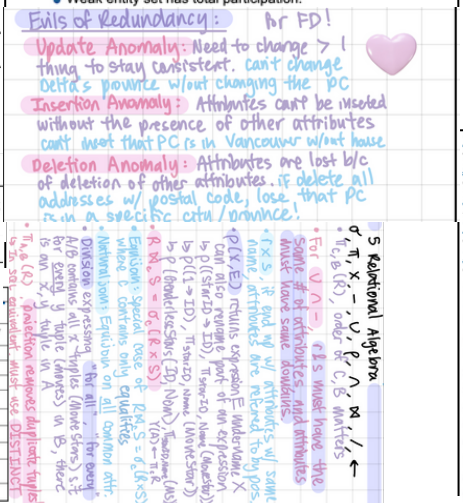


ISA relationships

If we declare A tion
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

Weak Entities

- 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.

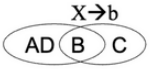


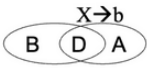
Decomposing

Decomposing into 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)

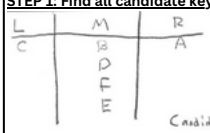




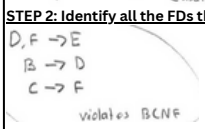
Consider the relation $R = (A, B, C, D, E, F)$ with the following functional dependencie
- (FD1) $C, D \rightarrow A$
- (FD2) $C, E \rightarrow B$
- (FD3) $D, F \rightarrow E$
- (FD4) $B \rightarrow D$
- (FD5) $C \rightarrow F$

STEP 1: Find all candidate keys.

Candidate keys is where you can get every letter.

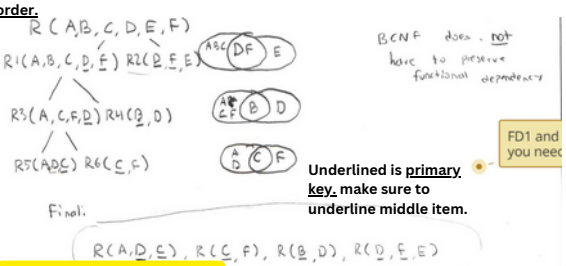


STEP 2: Identify all the FDs that violate the BCNF condition.



BCNF is that for X -> B, it has to be a superkey. Since {C, E}, {C, D}, {C, B} are keys, FD1 and FD2 are ruled out, meaning only 3 options left.

STEP 3: Decompose R so that everything is in BCNF. Choose FDs in ascending order.



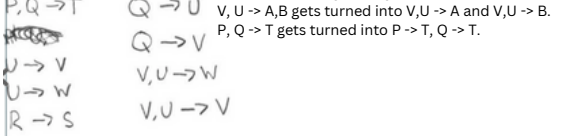
Underlined is primary key, make sure to underline middle item.

Decompose into minimal cover

Given the relation $A(P, Q, R, S, T, U, V, W)$ and set of FDs:
$P, Q \rightarrow T$
$U \rightarrow V, W$
$R \rightarrow S$
$Q \rightarrow U, V$
$V, U \rightarrow W, V$

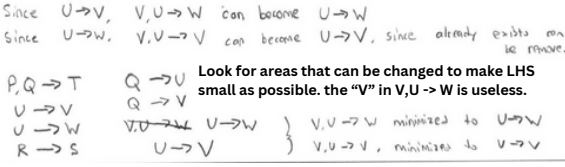
Cross or cartesian product is denoted by X * Y and returns a relation on tuples, whose schema contains all fields of X followed by all fields of Y.
Natural join is denoted by '⋈'. It is applicable only when there is at least one attribute common to R and S.

STEP 1: Put FDs into standard form.



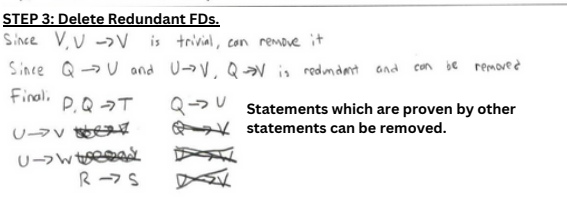
Standard form: split output into individual FDs.
V, U -> A,B gets turned into V,U -> A and V,U -> B.
P, Q -> T gets turned into P -> T, Q -> T.

STEP 2: Minimize the LHS (left-hand-side) of FDs.



Look for areas that can be changed to make LHS small as possible. the "V" in V,U -> W is useless.

STEP 3: Delete Redundant FDs.



Statements which are proven by other statements can be removed.

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 $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
e.g., if sid→acode and sid→city, then sid→acode,city

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

Functional dependency: one attribute determines another.
Ex. Department determines Address but not Name, we say that there's a functional dependency from Department to Address. But Department is NOT a key.

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

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

Normal Form, and proving decompositions