Lecture 12: Design Theory: Functional Dependencies and Normal Forms, Part I

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems,
3rd edition, Chapter 3, Sections 3.1 - 3.5

Important Notices

- Lecture slides and other class information will be posted on Piazza (not Canvas).
  - Slides are posted under Resources→Lectures
  - Lecture Capture recordings are available to all students under Yuja.
    - That includes classes given over Zoom.
  - There's no Lecture Capture for Lab Sections.
- All Lab Sections (and Office Hours) normally meet In-Person, with rare exceptions announced on Piazza.
  - Some slides for Lab Sections have been posted on Piazza under Resources→Lab Section Notes
- Office Hours for TAs and me are posted in Syllabus and Lecture1, as well as in Piazza notice @7; that post also now includes hours for Group Tutors.
- Some suggestions about use (and non-use) of Generative AI systems such as ChatGPT were posted on Piazza in @41, with specific discussion about the Movies tables and the four Avatar queries in Lecture 4.

Important Notices

- Lab3 scores were unmuted on Monday, March 4.
  - Deadline for asking questions about your score is Friday, March 8.
- The Fifth Gradiance Assignment, CSE 180 Winter 2024 #5, will be assigned on Friday, March 8, after you've learned about Design Theory (Lectures 12 and 13).
- Explanation of two difficult Gradiance questions on transactions was posted on Piazza on Sunday, March 2 in @122.
- Answers for two of the "Practice" Relational Algebra queries (in magenta) were posted on Piazza on Sunday, March 2 in @126.
- Winter 2024 Student Experience of Teaching Surveys - SETs open on Monday, March 4.
  - SETs close on Sunday March 17 at 11:59pm.
  - Instructors are not able to identify individual responses.
  - Constructive responses help improve future courses.

Important Notices

- The subject of Lab4 is Real Application Programming, using libpq (Lecture 11) and a Stored Function (Lecture 10).
  - Lab4 was posted on Piazza on Wednesday, February 28.
  - Lab4 is hard, and many students will need help completing it.
    - Read the Lab4 announcement on Piazza as soon as possible!
    - We provided information files and examples, posted under Resources→Lab4, and described in the Lab4 pdf and announcement.
    - We have also provided load data that you can use to test your Lab4 solution.
      - But testing can prove that program is incorrect, <u>not</u> that it's correct.
  - Lab4 is due on **Tuesday, March 12 by 11:59pm**.
    - unix.ucsc.edu gets busy near the end of the quarter.
    - Please avoid Infinite Loops in your Stored Function.
  - All material needed for Lab4 was covered in Lecture by Friday, March 1.
  - However, there were some important clarifications and corrections which were made to Lab4.
    - These corrections, which are described in <u>@121</u> on Piazza, “Important updates of multiple Lab4 files”, are summarized on the next slide.

Important Updates of Multiple Lab4 Files Slide #1 (see @121)

- The Lab4 pdf is now correctly called CSE180_W24_Lab4.pdf
- A new version of the load file, load_lab4.sql, has ben posted.
- Several changes have been made in the runShakespeareApplication.c skeleton file.
  - The signature for the C function IncreaseSomeCastMemberSalaries in the runShakespeareApplication.c skeleton has been changed to:

    ```
    float increaseSomeCastMemberSalaries(PGconn *conn, char *thePlayTitle,
            int theProductionNum, float maxDailyCost)
    ```

    - That is, the maxDailyCost parameter is float, and the function returns a float value.
  - Minor: The name of the function begins with a lowercase "i", just as all the other C functions begin with lowercase letters.
    - Same is true for the Stored Function which is invokes, which is called increaseSomeCastMemberSalariesFunction (lowercase i, not uppercase).
  - Minor: Parameter which had been called theProdNum is now called theProductionNum, aligning with productionNum attribute name in tables.

Important Updates of Multiple Lab4 Files Slide #2 (see @121)

- The first lines of the Stored Function increaseSomeCastMemberSalariesFunction (which we did not give you in the Lab4 pdf) should be:

```
CREATE OR REPLACE FUNCTION
    increaseSomeCastMemberSalariesFunction(thePlayTitle VARCHAR(40),
            theProductionNum INTEGER, maxDailyCost NUMERIC(7,2))
 RETURNS NUMERIC(7,2) AS $$
```

  - Note that C does not have a type corresponding to NUMERIC(7,2).
- The value returned by increaseSomeCastMemberSalariesFunction when you execute that Stored Function in your C program (and then use PQgetvalue(res,0,0) to get get the result of the Stored Function) is a string.
  - Just as you can convert an appropriate character string to an integer using C's atoi function, you can convert a character string to a float using C's atof function, and increaseSomeCastMemberSalaries needs to return a float.
- The value that you print out in your tests of increaseSomeCastMemberSalaries should be a number with 2 decimal places. You can print out a floating point value in that format using the %7.2f format string (instead of using %f or %d).
- Per @135, you should assume that salaryPerDay, theaterFeePerDay (and also theaterID in Productions) can't be NULL.

Important Notices

- Lab3 scores were unmuted on Monday, November 28.
  - Deadline for asking questions about Lab3 grades is Saturday, December 2.
  - Your questions don't have to be resolved by that date, but you must submit them by that date.
- The subject of Lab4 is Real Application Programming, using libpq (Lecture 11) and a Stored Function (Lecture 10).
  - Lab4 was posted on Piazza on Wednesday, November 22.
  - Lab4 is hard, and many students will need help completing it.
    - Read the Lab4 announcement on Piazza as soon as possible!
    - We provided information files and examples, posted under Resources→Lab4, and described in the Lab4 pdf and announcement.
    - We have also provided load data that you can use to test your Lab4 solution.
      - But testing can prove that program is incorrect, not that it's correct.
    - Using views isn't required for Lab4, but views may help you, particularly for the countCoincidentSubscriptions C function.
      - Include a createNewspaperViews.sql file in your solution if you use views.
  - Lab4 is due on **Tuesday, December 5 by 11:59pm**.
    - unix.ucsc.edu gets busy near the end of the quarter.
- You won't be asked to write PL/pgSQL or C code on the Final, but you might be asked about concepts, and you might be asked what code does.

Important Notices: Final

- CSE 180 Final is on **Wednesday, March 20, 8:00-11:00am, and it will be given in-person in our classroom, except for students who receive Remote Exam permission.**
  - **No** early/late Exams. **No** make-up Exams. **No** devices (except for Remote students).
  - 3 hours, extended for DRC students, covering the <u>entire quarter.</u>
    - All DRC students should have recently received email about the final.
    - Final will be harder than the Midterm.
  - **You may bring one double-sided 8.5 x 11 sheet to the Final, with anything that you want written or printed on it that you can read unassisted) …**
    - (Okay, you may bring two sheets with writing only on one side of each sheet.)
    - **But you must not use any other material or receive any help during the Final, whether you're in the classroom, remote, or in a DRC room**
    - As the Syllabus emphasizes, Academic Integrity violations have serious consequences!
  - We will assign seats as you enter the room. You may not choose your own seat.
- CSE 180 Final from Winter 2023 was posted on Piazza under Resources→Exams on Sunday, March 3.
  - Solution to that Final will be posted on Piazza by Monday, March 11 … but take it yourself first, rather than just reading the solution.

Important Notices: In-Person Final

Final includes a Multiple Choice Section and a Longer Answers Section.

- In-Person students should bring a Red Scantron sheet (ParSCORE form number f-1712) sold at Bookstore for about 25 cents, and #2 pencils for Multiple Choice Section.
- You'll answer Multiple Choice Section on your Scantron sheet, entering your name, your student ID, and which version ("Test Form Letter") of the Multiple Choice Section you're answering.
  - Write your Multiple Choice answers on the Scantron sheet using a #2 pencil.
  - Ink and #3 pencils don't work.
- You'll answer Longer Answers Section on the Long Answers Section, using either ink or #2 pencil (as on the Midterm).
- Be sure to answer all questions readably!!
- Do not hand in your 8.5 x 11 sheet or your Multiple Choice Section. Just hand in your Scantron Sheet and your Longer Answers Section.
  - But show us your Multiple Choice Section, so that we can check that you filled in the Test Form Letter (A, B, C or D) for that Section on your Scantron Sheet.

Important Notices: Remote Final

CSE 180 Final **will be given in-person in our classroom, except for students who receive Remote Exam permission.**

- If you need to take the Final Remotely, send me an email whose subject is "Taking the CSE 180 Final Remotely" by Tuesday, March 19 at 6:00pm that includes your strong justification for that request.
  - Only students whose requests are approved may take the Final Remotely.
    - If you don't receive a response from me by 9:00pm, please send email again!
- I'll send instructions to you for taking the Final before 8:00am on Wednesday, March 20 which include a Zoom link
  - You must be on Zoom while you are taking the Final.
    - Please make sure that your face is visible on Zoom video, but that your microphone is muted.
    - Please do not have headphones on during the Final.
    - If there are any bugs on the Final, they will be conveyed to all Remote students via Zoom Chat.
    - If you have a question during the Final, please send it to me (just to me) directly using Zoom Chat, not by speaking.
  - You'll have to complete the exam by 11:00am, just like all other students …
  - … except for DRC students, who will receive extra time, whether they take the exam In-Person (in a different room) or Remotely.
  - You <u>won't</u> need a red Scantron Sheet if you're taking the Final Exam Remotely.

A Word to the Unwise

- This is a tough class for some students since it involves a combination of theory and practice.
  - The second half of CSE 180 is <u>much harder</u> than the first half of the course.
- Students who regularly attend Lectures and Lab Sections (and Office Hours and Tutoring) often do well; students who don't regularly attend often do poorly.
  - We don't take attendance; you're responsible for your own choices.
- After the course ends, your course grade will be determined by your scores on Exams, Labs and Gradiance, as described on the "Course Evaluation" and "Grading" Slides in the Syllabus.
  - You won't be able to do any additional work to improve your grade.

Database Schema Design

- So far, we have learned database query languages:
  - SQL, Relational Algebra
- How can you tell whether a given database schema is “good” or “bad”?
- Design theory:
  - A set of design principles that allows one to decide what constitutes a “good” or “bad” database schema design.
  - A set of algorithms for modifying a “bad” design to a “better” one.

Example

- If we know that rank determines the salary scale, which is a better design? Why?

- Employees(eid, name, addr, rank, salary_scale)

OR

- Employees2(eid, name, addr, rank)
  Salary_Table(rank, salary_scale)

Lots of Duplicate Information

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Lots of duplicate information
  - Employees who have the same rank have the same salary scale.

Update Anomaly

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Update anomaly
  - If one copy of salary scale is changed, then all copies of that salary scale (of the same rank) have to be changed.

Insertion Anomaly

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Insertion anomaly
  - How can we store a new rank and salary scale information if currently, no employee has that rank?

Deletion Anomaly

eid	name	addr	rank	salary_scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

- Deletion anomaly
  - If Hugh is deleted, how can we retain the rank and salary scale information?

So What Would Be a Good Schema Design for this Example?

- salary_scale is dependent only on rank
  - Hence associating employee information such as name, addr with salary_scale causes redundancy.

- Based on the constraints given, we would like to refine the schema so that such redundancies **cannot** occur.

- Note however, that sometimes database designers **may choose** to live with redundancy in order to improve query performance.
  - Ultimately, a good design depends in part on the query workload.
  - But understanding anomalies and how to deal with them is still important.

Functional Dependencies

- The information that rank determines salary_scale is a type of integrity constraint known as a *functional dependency (FD).*

- Functional dependencies can help us detect anomalies that may exist in a given schema.

- The FD “rank → salary_scale” suggests that

  Employees(<u>eid</u>, name, addr, rank, salary_scale)

  should be *decomposed* into two relations:

  Employees2(<u>eid</u>, name, addr, rank)

  Salary_Table(<u>rank</u>, salary_scale).

Meaning of an FD

We have seen a Functional Dependency before.

Keys:

- Emp(<u>ssn</u>, name, addr)
- If two tuples agree on the ssn value, then they must also agree on the name and address values. (ssn → name, addr).

Let **R** be a relation schema. A *Functional Dependency (FD)* is an integrity constraint of the form:

X → Y (read as "X *determines* Y, or X *functionally determines* Y")

where X and Y are non-empty subsets of attributes of **R**.

A relation instance r of **R** *satisfies* the FD X → Y if

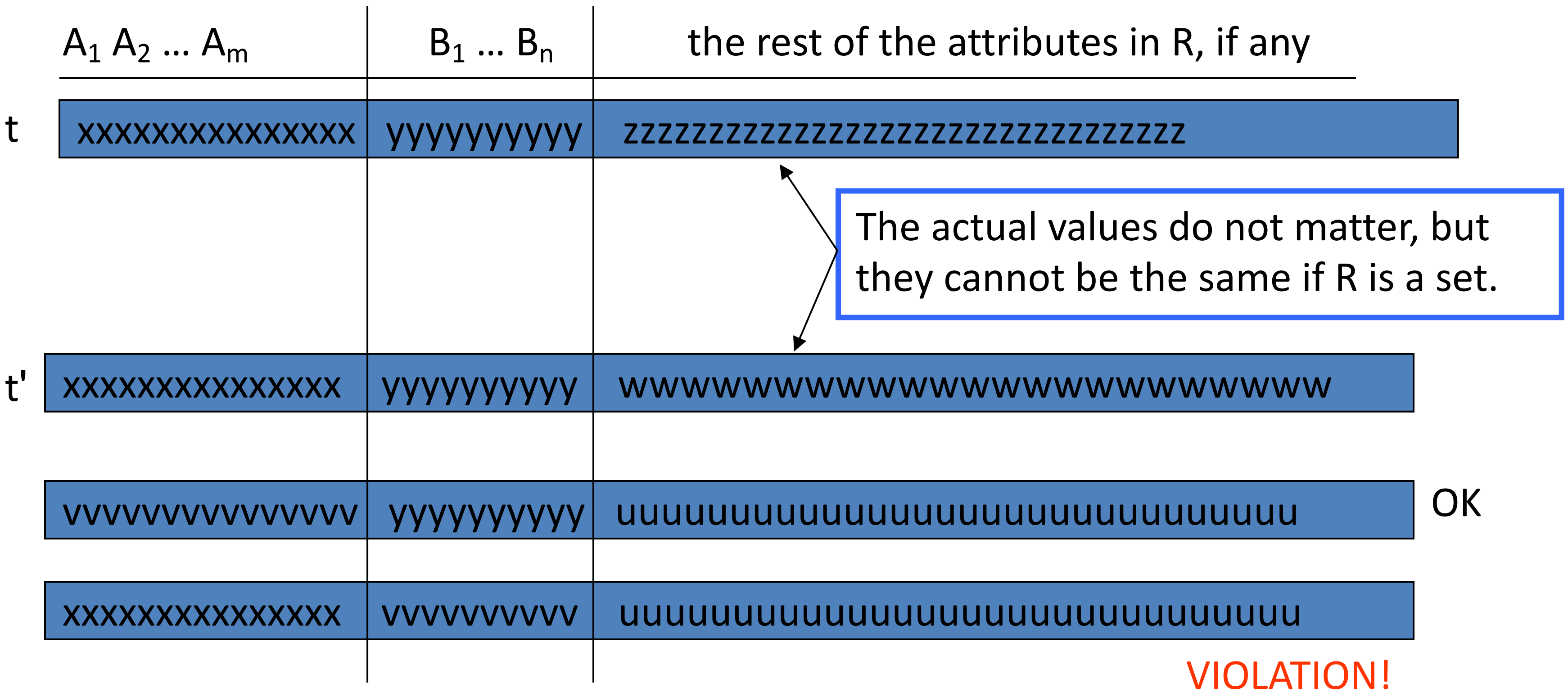
for every pair of tuples t and t' in r:

If $\pi_X(t) = \pi_X(t')$ holds, then $\pi_Y(t) = \pi_Y(t')$ also holds.

That is, if the two tuples t and t' agree on the the values of all the attributes in X, then the two tuples t and t' must also agree on the values of all the attributes in Y.

Illustration of the Semantics of an FD

- Relation schema R with the FD $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ where $\{ A_1, \ldots, A_m, B_1, \ldots, B_n \} \subseteq$ attributes(R).



More on Meaning of an FD

- Relation R satisfies $X \rightarrow Y$
  - Pick any two (not necessarily distinct) tuples t and t' of an instance r of R. If t and t' agree on the X attributes, then they must also agree on the Y attributes.
  - The above must hold for *every possible instance* r of R.

- An FD is a statement about *all possible legal instances* of a schema. We <u>cannot</u> just look at an instance (or even at a set of instances) to determine which FDs hold.
  - Looking at an instance may enable us to determine that some FDs are <u>not satisfied</u>.

Reasoning about FDs

R(A,B,C,D,E)

Suppose A → C and C → E. Is it also true that A → E ?

In other words, suppose an instance r satisfies A → C and C → E, is it true that r must also satisfy A → E ?

YES

Proof: ?

Implication of FDs

- We say that a set $\mathcal{F}$ of FDs *implies* an FD F if for every instance r that satisfies $\mathcal{F}$, it must also be true that r satisfies F.
- Notation: $\mathcal{F} \models F$

- Note that just finding some instance(s) r such that r satisfies $\mathcal{F}$ and r also satisfies F is not sufficient to prove that $\mathcal{F} \models F$.

- How can we determine whether or not $\mathcal{F}$ implies F?

Armstrong's Axioms

- Let X, Y, and Z denote sets of attributes over a relation schema R.

- **Reflexivity**: If Y ⊆ X, then X → Y.

  ssn, name → name

  – FDs in this category are called *trivial FDs.*
- **Augmentation**: If X → Y, then XZ → YZ for any set Z of attributes.

  ssn, name, addr → name addr
- **Transitivity**: If X → Y and Y → Z, then X → Z.

  If ssn → rank, and rank → sal_scale,

  then ssn → sal_scale.

Union and Decomposition Rules

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

- Union and Decomposition rules are not essential. In other words, they can be derived using Armstrong's axioms.

- Derivation of the Union rule:

  (to fill in)

Union and Decomposition Rules

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

- Union and Decomposition rules are not essential. In other words, they can be derived using Armstrong's axioms.

- Derivation of the Union rule:

  Since $X \rightarrow Z$, we get $XY \rightarrow YZ$ (augmentation)

  Since $X \rightarrow Y$, we get $X \rightarrow XY$ (augmentation)

  Therefore, $X \rightarrow YZ$ (transitivity)

Additional Rules

- Derivation of the Decomposition rule:
  (to fill in)

Additional Rules

- Derivation of the Decomposition rule:

  $X \rightarrow YZ$ (given)

  $YZ \rightarrow Y$ (reflexivity)

  $YZ \rightarrow Z$ (reflexivity)

  Therefore, $X \rightarrow Y$ and $X \rightarrow Z$ (transitivity).

- We use the notation $\mathcal{F} \vdash F$ to mean that *F can be derived from $\mathcal{F}$ using Armstrong's axioms*.
  – What was the meaning of $\mathcal{F} \vDash F$ ($\mathcal{F}$ implies F)?

Pseudo-Transitivity Rule

- **Pseudo-Transitivity**: If $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$.
- Can you derive this rule using Armstrong's axioms?
- Derivation of the Pseudo-Transitivity rule:

  (to fill in)

Pseudo-Transitivity Rule

- **Pseudo-Transitivity**: If X $\rightarrow$ Y and WY $\rightarrow$ Z, then XW $\rightarrow$ Z.

- Can you derive this rule using Armstrong's axioms?

- Derivation of the Pseudo-Transitivity rule:

  X -> Y and WY -> Z

  XW -> WY (augmentation)

  WY -> Z (given)

  Therefore XW -> Z (transitivity)

Completeness of Armstrong's Axioms

- **Completeness**: If a set $\mathcal{F}$ of FDs implies F, then F can be derived from $\mathcal{F}$ using Armstrong's axioms.
  - If $\mathcal{F} \vDash F$, then $\mathcal{F} \vdash F$.

    semantic

    syntactic

  - If $\mathcal{F}$ implies F, then we can derive F from $\mathcal{F}$ using Armstrong's axioms.

For those familiar with Mathematical Logic:

- $\mathcal{F} \vDash F$ is "model-theoretic"
- $\mathcal{F} \vdash F$ is "proof-theoretic"

Soundness of Armstrong's Axioms

- **Soundness**: If F can be derived from a set of FDs $\mathcal{F}$ using Armstrong's axioms, then $\mathcal{F}$ implies F.
  - If $\mathcal{F} \vdash$ F, then $\mathcal{F} \vDash$ F.
    - That is, if we can derive F from $\mathcal{F}$ using Armstrong's axiom, then $\mathcal{F}$ implies F.
  - Handwaving proof: If we can derive F from $\mathcal{F}$ using Armstrong's axioms, then surely $\mathcal{F}$ implies F. (<u>Why?</u>)

- With Completeness and Soundness, we know that
  $\mathcal{F} \vdash$ F if and only if $\mathcal{F} \vDash$ F
- In other words, the FDs that we can derive from $\mathcal{F}$ using Armstrong's axioms are <u>precisely</u> all the FDs that must hold given $\mathcal{F}$ (that is, all the axioms that $\mathcal{F}$ implies).

- Great! But how can we decide whether or not $\mathcal{F}$ implies F?

Closure of a Set of FDs $\mathcal{F}$

- Let $\mathcal{F}^+$ denote the set of all FDs implied by a given set $\mathcal{F}$ of FDs.
  - Also called the closure of $\mathcal{F}$.
- To decide whether $\mathcal{F}$ implies F, first compute $\mathcal{F}^+$, then see whether F is a member of $\mathcal{F}^+$.
- Example: Compute $\mathcal{F}^+$ for the set { $A \rightarrow B$, $B \rightarrow C$} of FDs.
- Trivial FDs
  - $A \rightarrow A$, $B \rightarrow B$, $C \rightarrow C$, $AB \rightarrow A$, $AB \rightarrow B$, $BC \rightarrow B$, $BC \rightarrow C$, $AC \rightarrow A$, $AC \rightarrow C$, $ABC \rightarrow A$, $ABC \rightarrow B$, $ABC \rightarrow C$, $ABC \rightarrow AB$, $ABC \rightarrow AC$, $ABC \rightarrow BC$, $ABC \rightarrow ABC$
- Transitivity (non-trivial FDs)
  - $AC \rightarrow B$ $AC \rightarrow A$ (trivial), $A \rightarrow B$ (given), so $AC \rightarrow B$ (transitivity).
  - $AB \rightarrow C$ $AB \rightarrow B$ (trivial), $B \rightarrow C$ (given), so $AB \rightarrow C$ (transitivity).
  - $A \rightarrow C$ $A \rightarrow B$ (given), $B \rightarrow C$ (given), so $A \rightarrow C$ (transitivity).

Expensive and tedious! Let's find a better way.

<u>Attribute</u> Closure Algorithm

Let X be a set of attributes, and $\mathcal{F}$ be a set of FDs.
The Attribute Closure X^+ with respect to $\mathcal{F}$ is the set of all attributes A such that $X \rightarrow A$ is derivable from $\mathcal{F}$.

- That is, all the attributes A such that $\mathcal{F} \vdash X \rightarrow A$

<u>Input</u>: A set X of attributes and a set $\mathcal{F}$ of FDs.
<u>Output</u>: X^+

```
Closure = X;            // initialize Closure to equal the set X
Repeat until no change in Closure {
   If there is an FD  U → V in F such that U ⊆ Closure,
         then Closure = Closure ∪ V;
}
return Closure;
```

Should be clear that: If $A \in$ "Closure" (that is, if $A \in X^+$) , then $X \rightarrow A$.
More strongly: $\mathcal{F} \vdash X \rightarrow A$ <u>if and only if</u> $A \in X^+$.

FD Example 1 using Attribute Closure

- $\mathcal{F}$ = { A $\rightarrow$ B, B $\rightarrow$ C }.
- Question: Does A $\rightarrow$ C?
- Compute A^+

We'll write A^+
Instead of $\{A\}^+$

- Closure = { A }
- Closure = { A, B } (due to A $\rightarrow$ B)
- Closure = { A, B, C } (due to B $\rightarrow$ C)
- Closure = { A, B, C }
  – no change, stop
- Therefore A^+ = {A, B, C }
- Since C $\in$ A^+, answer YES.

FD Example 2 using Attribute Closure

- $\mathcal{F} = \{ AB \rightarrow E, B \rightarrow AC, BE \rightarrow C \}$
- Question: Does $BC \rightarrow E$?
- Compute $(BC)^+$

We'll write $(BC)^+$
Instead of $\{B,C\}^+$

- Closure = { B, C }
- Closure = { A, B, C } (due to $B \rightarrow AC$)
- Closure = { A, B, C, E } (due to $AB \rightarrow E$)
- Closure = { A, B, C, E } (due to $BE \rightarrow C$)
  - No change, so stop.
- Therefore $(BC)^+ = \{A,B,C,E\}$
- Since $E \in (BC)^+$, answer YES.

A Better Algorithm for FDs

It's much easier to compute Attribute Closure X^+, rather than FD Closure $\mathcal{F}^+$

- To determine if an FD $X \rightarrow Y$ is implied by $\mathcal{F}$, compute X^+ and check if $Y \subseteq X^+$.
- Notice that computing Attribute Closure X^+ is less expensive (and less tedious) to compute than is FD Closure $\mathcal{F}^+$.

Correctness of Attribute Closure Algorithm

If $A \in X^+$, then $\mathcal{F} \vdash X \rightarrow A$

- This is true because Armstrong's Axioms are "sound".

If $\mathcal{F} \vdash X \rightarrow A$, then $A \in X^+$

- Proof by contradiction. Won't go through proof details.
- Please let me know if you'd like to see the proof.

Reminder: Superkeys and Keys

- A **superkey S** for a relation schema R is a subset of the attributes of R such that:
   1. There can't be two different tuples in an instance of R that have the same value for all the attributes in S.

   So S is a superkey for relation R if and only if S^+ = attrib(R).

- A **key K** for a relation schema R is a subset of the attributes of R such that the following two properties hold:
   1. There can't be two different tuples in an instance of R that have the same value for all the attributes in K.
   ... that is, K is a superkey of R, which is true if and only if K^+ = atttrib(R).
   2. And K is Minimal: No proper subset of K is a superkey of R.

All keys are superkeys ... but some superkeys are not keys.

Using Attribute Closure Algorithm to Find All Superkeys/Keys for Relation R, Given Functional Dependencies $\mathcal{F}$

Attribute Closure algorithm can be modified to **find all superkeys and all candidate keys** for R, given Functional Dependencies $\mathcal{F}$.

- Compute the closure of a single attribute in attr(R). Then compute the closure of every 2 attribute set, 3 attribute set, and so on.
- If the closure of a set of attributes X contains <u>all</u> the attributes of relation R, then X is a *superkey* for R.
- If <u>no proper subset</u> of X is a superkey, then X is a *key*.
  - *Proper subset* of a set: A subset that isn't the entire set.

Here's an observation which makes it easier to find superkeys and keys:

If some attribute B doesn't appear on the right-hand side of any non-trivial FD for R, then that attribute B must in <u>every</u> superkey for R (why?), and hence must be in <u>every</u> key for R!

Using Attribute Closure Algorithm to Determine If a Set of Attributes X is a SuperKey/Key for Relation R, Given Functional Dependencies $\mathcal{F}$

Attribute Closure algorithm can be modified to determine **if a set of attributes X is a superkey/key** for R, given Functional Dependencies $\mathcal{F}$.

- How?
  - Compute the attribute closure of X^+.
  - If X^+ = attr(R), then X is a *superkey*.
  - If no proper subset of X is a superkey, then X is a *key*.
    - See whether X is still a superkey after it "goes on a diet"!
    - That is, try to find some attribute in X such that X is still a superkey if X loses that attribute?
    - If X still is a superkey after X loses some attribute ("goes on a diet"), then X is a superkey, but X is not a key.

Practice Homework 6

1. Let R(A,B,C,D,E) be a relation schema and let $\mathcal{F} = \{ AB \rightarrow E, B \rightarrow AC, BE \rightarrow C \}$ be a set of FDs that hold over R.

   a. Prove that $\mathcal{F} \models B \rightarrow E$ using Armstrong's axioms.
   b. Compute the closure of B. That is, compute B^+.
   c. Give a key for R. Justify why your answer is a key for R.
   d. Show an example relation that satisfies $\mathcal{F}$.
   e. Show an example relation that does not satisfy $\mathcal{F}$.

2. Let R(A,B,C,D,E) be a relation schema and let $\mathcal{F} = \{ A \rightarrow C, B \rightarrow AE, B \rightarrow D, BD \rightarrow C \}$ be a set of FDs that hold over R.

   a. Show that $B \rightarrow CD$ using Armstrong's axioms.
   b. Show a relation of R such that R satisfies $\mathcal{F}$ but R does not satisfy $A \rightarrow D$.
   c. Is AB a key for R?