Chapter 3
Design theory for Relational Databases

Université Grenoble Alpes

09/03/2023

Bahareh Afshinpour

bahareh.afshinpour@univ-grenoble-alpes.fr

Main reference:
A First Course in Database Systems (and associated material) by
J. Ullman and J. Widom, Prentice-Hall

Left outer join- Right outer join

There are many variants of the basic (natural) outerjoin idea. The *left outerjoin* $R \overset{\circ}{\bowtie}_L S$ is like the outerjoin, but only dangling tuples of the left argument R are padded with $\perp$ and added to the result. The *right outerjoin* $R \overset{\circ}{\bowtie}_R S$ is like the outerjoin, but only the dangling tuples of the right argument S are padded with $\perp$ and added to the result.

Examples

- **lives**(person-name,street,city)
- **works**(person-name, company-name,salary)
- **located-in**(company-name,city)
- **manages**(person-name,manager-name)

For the above schema (the primary key for each relation is denoted by the underlined attribute), provide relational algebra expressions for the following queries:

1. Find all tuples in works of all persons who work for the City Bank company (which is a specific company in the database).
   (a) $\sigma_{(cname='City\,Bank')}(works)$

2. Find the name of persons working at City Bank who earn more than $50,000.
   (a) $\pi_{pname}(\sigma_{(cname='City\,Bank')\wedge(salary>50000)}(works))$

3. Find the name and city of all persons who work for City Bank and earn more than 50,000. Similar to previous query, except we have to access the lives table to extract the city of the employee . Note the join condition in the query.
   (a) $\pi_{lives.pname,lives.city}(\sigma_{((cname='City\,Bank')\wedge(salary>50000)\wedge(lives.pname=works.pname))})(lives \times works)$

Find names of all persons who do not work for City Bank. Can write this in multiple ways - one solution is to use set difference:

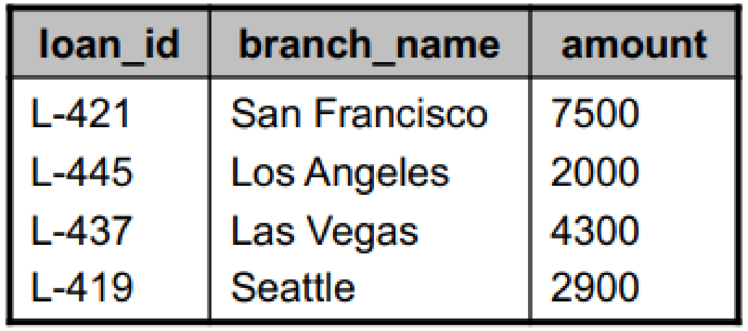
(a) $(\pi_{pname}(works)) - (\pi_{pname}(\sigma_{cname='City\,Bank'}(works)))$

6. Find the name of all persons who work for City Bank and live in DC. Similar to query 3, but select only with tuples where person city is DC.

(a) $\pi_{lives.pname}(\sigma_{((cname='City\,Bank')\wedge(lives.city='DC')\wedge(lives.pname=works.pname))})(lives \times works)$

Example

- Find the ID of the loan with the largest amount.
  - Hard to find the loan with the largest amount! (At least, with the tools we have so far…)
  - Much easier to find all loans that have an amount smaller than some other loan
  - Then, use set-difference to find the largest loan



loan_id	branch_name	amount
L-421	San Francisco	7500
L-445	Los Angeles	2000
L-437	Las Vegas	4300
L-419	Seattle	2900

How to find all loans with an amount smaller than some other loan?

Use Cartesian Product of *loan* with itself:

loan × *loan*

Compare each loan's amount to all other loans

Problem: Can't distinguish between attributes of left and right *loan* relations!

Solution: Use rename operation

$loan \times \rho_{test}(loan)$

Now, right relation is named *test*

Find IDs of all loans with an amount smaller than some other loan:

$\Pi_{loan.loan\_id}(\sigma_{loan.amount<test.amount}(loan \times \rho_{test}(loan)))$

Finally, we can get our result:

$\Pi_{loan\_id}(loan) -$
$\Pi_{loan.loan\_id}(\sigma_{loan.amount<test.amount}(loan \times \rho_{test}(loan)))$

loan_id
L-421

loan_id	branch_name	amount
L-421	San Francisco	7500
L-445	Los Angeles	2000
L-437	Las Vegas	4300
L-419	Seattle	2900

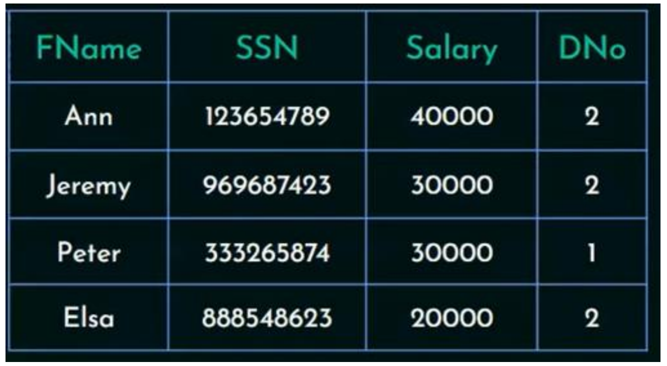
loan_id	branch_name	amount
L-421	San Francisco	7500
L-445	Los Angeles	2000
L-437	Las Vegas	4300
L-419	Seattle	2900

loan_id	branch_name	amount	loan_id	branch_name	amount
L-421	San Francisco	7500	L-421	San Francisco	7500
L-421	San Francisco	7500	L-445	Los Angeles	2000
L-421	San Francisco	7500	L-437	Las Vegas	4300
L-421	San Francisco	7500	L-419	Seattle	2900
L-445	Los Angeles	2000	L-421	San Francisco	7500
L-445	Los Angeles	2000	L-445	Los Angeles	2000
L-445	Los Angeles	2000	L-437	Las Vegas	4300
L-445	Los Angeles	2000	L-419	Seattle	2900

Aggregation operators (summarize)

- Many operators we can apply to set or bags of numbers or strings.
- They are used to **summarize or aggregate** the values in <u>one column </u>of relation

- For example:
- SUM
- AVG
- MIN and MAX (numerical values and character-string values)
- COUNT



FName	SSN	Salary	DNo
Ann	123654789	40000	2
Jeremy	969687423	30000	2
Peter	333265874	30000	1
Elsa	888548623	20000	2

To retrieve the number of person and their salary

Count(ISBN) R   AVERAGE(SALARY) R

Chapter 3

Design Theory for Relational Databases

Chapter 3

- We can examine the requirements for a database and define relations directly, without going through a high-level intermediate stage.
- In this chapter:
  - Identify the problems that are caused in some relation schemas
  - Normalization

Functional Dependencies

Determinant $X \rightarrow y$ Dependend

Y is determined by x

X=2 y=?

X=2 y=7

x	y
1	6
2	7
5	10
3	22

In the first relation, If I tell you the value of X you can find the value of Y

X=2 y=?

x	y
1	6
2	7
5	10
2	22

So, based on relation we can find the FD.

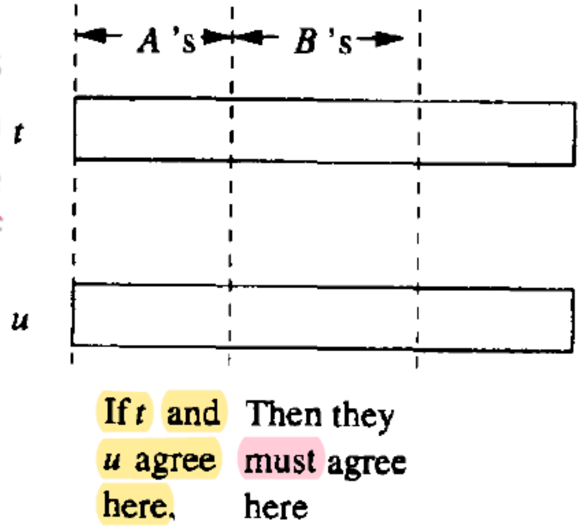
X and Y can be a set of attributes.

Functional Dependencies

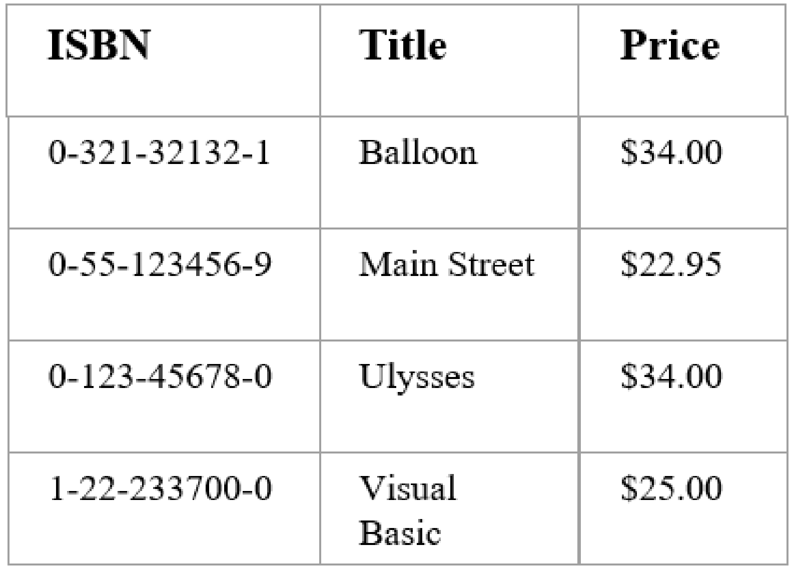
- If two tuples of R agree on all the attributes A1,A2,A3, …, An then they must also agree on all of another list of attributes B1,B2,…,Bm.
- We write this FD formally as A1A2…An → B1B2… Bm and say that

“A1,A2,….,An functionally determine B1,B2,…,Bm”

If one set of attributes in a table determines another set of attributes in the table, then the second set of attributes is said to be functionally dependent on the first set of attributes.



Examples



ISBN	Title	Price
0-321-32132-1	Balloon	$34.00
0-55-123456-9	Main Street	$22.95
0-123-45678-0	Ulysses	$34.00
1-22-233700-0	Visual Basic	$25.00

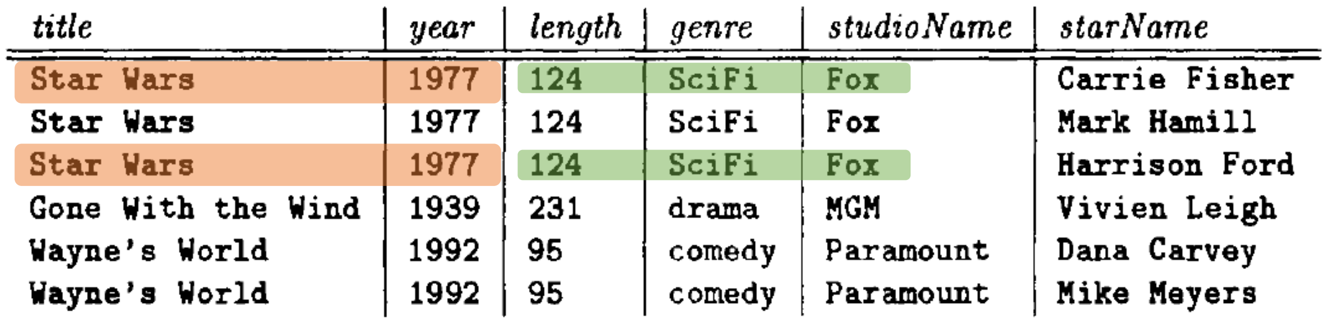
Table Scheme: {ISBN, Title, Price}

Functional Dependencies: **$\{ISBN\} \rightarrow \{Title\}$**

$\{ISBN\} \rightarrow \{Price\}$

Example: “no two courses can meet in the same room at the same time” tells us: **hour, room -> course**

Example2



title	year	length	genre	studioName	starName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure 3.2: An instance of the relation Movies1(title, year, length, genre, studioName, starName)

Thus, we expect that given a title and year, there is a unique movie.

🚫 Title, year→ starName

As we shall see, the schema for MOVIES1 is not a good design.

Title , year → length, genre, StudioName

This FD says that two tuples have the same value in their TITLE and Year components, the these two tuples must also have

- the same values in their Length component,
- the same values in their genre components,
- the same values in their studioName components

Functional Dependencies

- FD says something about all possible instances of the relation, not about one of its instances.

X→Y
IF tuple(i).x=tuple(j).x then
tuple(i).y=tuple(j).y

X	y
1	6
2	7
1	6
3	22

x	y
1	6
2	7
1	6
2	22

🚫

Only we need to find tuples with two equal value, then check the IF statement

If y for both of them is same so x->y is FD.

Keys of Relations

- We say a set of one or more attributes {A1,A2,A3,…An} is a key for a relation R if:

1. Those attributes functionally determine all other attributes of a relation .

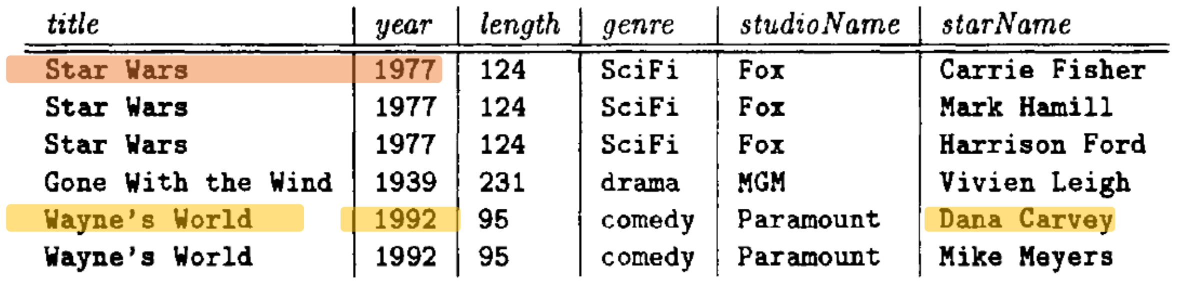
   It is impossible for two distinc tuples of R to agree on all A1,A2,A2,….,An

2. No proper subset of {A1,A2,A3,…An} functionally determines all other attributes of R;

   a key must be minimal.

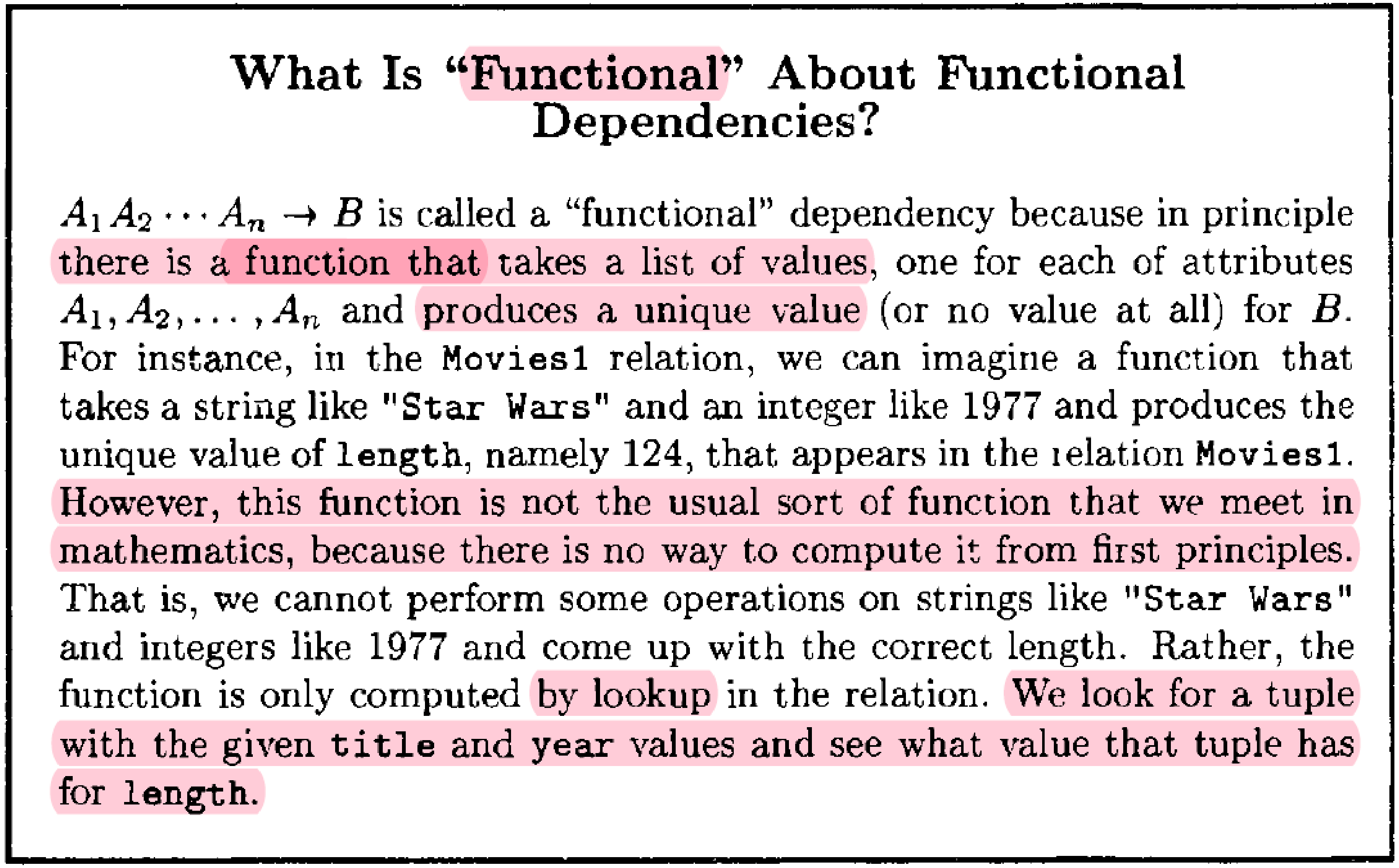
-{title, year, starname} form a key
1- two different tuples can not agree on all of title, year and starname
2-no proper subset of it functionally determines all other attributes ({title , year} is not a key)

Sometimes a relation has more than one key.



title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure 3.2: An instance of the relation Movies1(title, year, length, genre, studioName, starName)

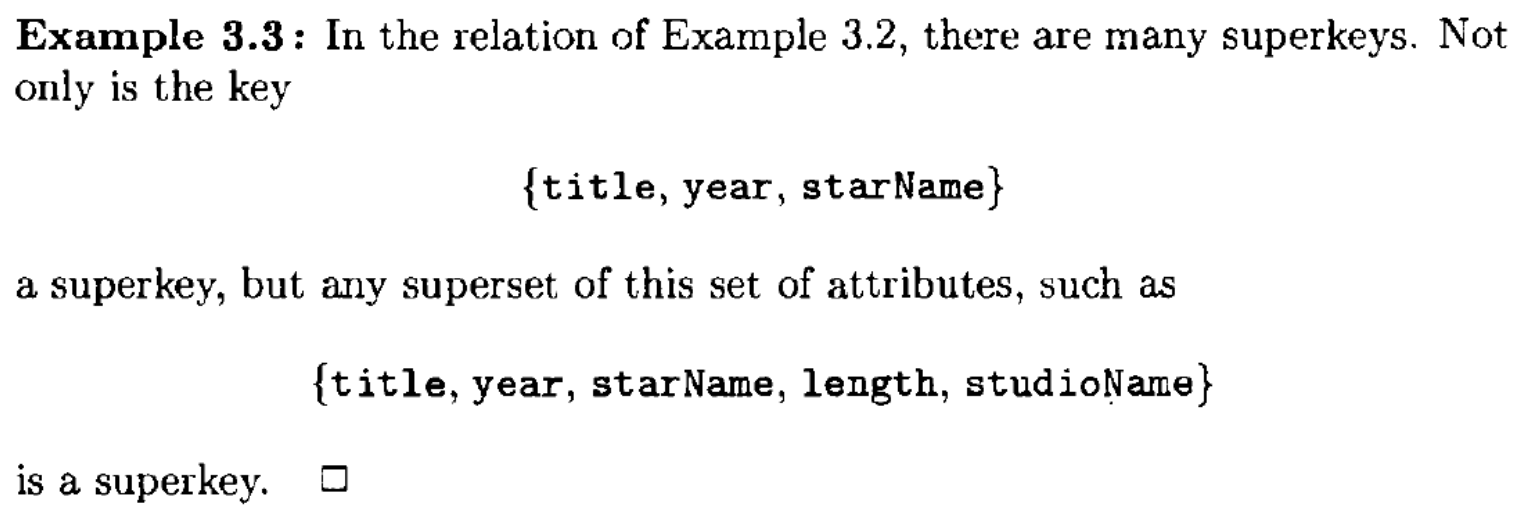


What Is "Functional" About Functional Dependencies?

$A_1A_2 \cdots A_n \rightarrow B$ is called a "functional" dependency because in principle there is a function that takes a list of values, one for each of attributes $A_1, A_2, \ldots, A_n$ and produces a unique value (or no value at all) for B. For instance, in the `Movies1` relation, we can imagine a function that takes a string like `"Star Wars"` and an integer like 1977 and produces the unique value of `length`, namely 124, that appears in the relation `Movies1`. However, this function is not the usual sort of function that we meet in mathematics, because there is no way to compute it from first principles. That is, we cannot perform some operations on strings like `"Star Wars"` and integers like 1977 and come up with the correct length. Rather, the function is only computed by lookup in the relation. We look for a tuple with the given `title` and `year` values and see what value that tuple has for `length`.

SuperKey

- A set of attributes that contain a key is called superkey.
- Every superkey satisfies the **first condition of a key**.
- Thus every key is a superkey.
- However, some superkeys are not (minimal)keys.
- **A superkey need not satisfy the second condition :minimality**.



Example 3.3: In the relation of Example 3.2, there are many superkeys. Not only is the key

{`title`, `year`, `starName`}

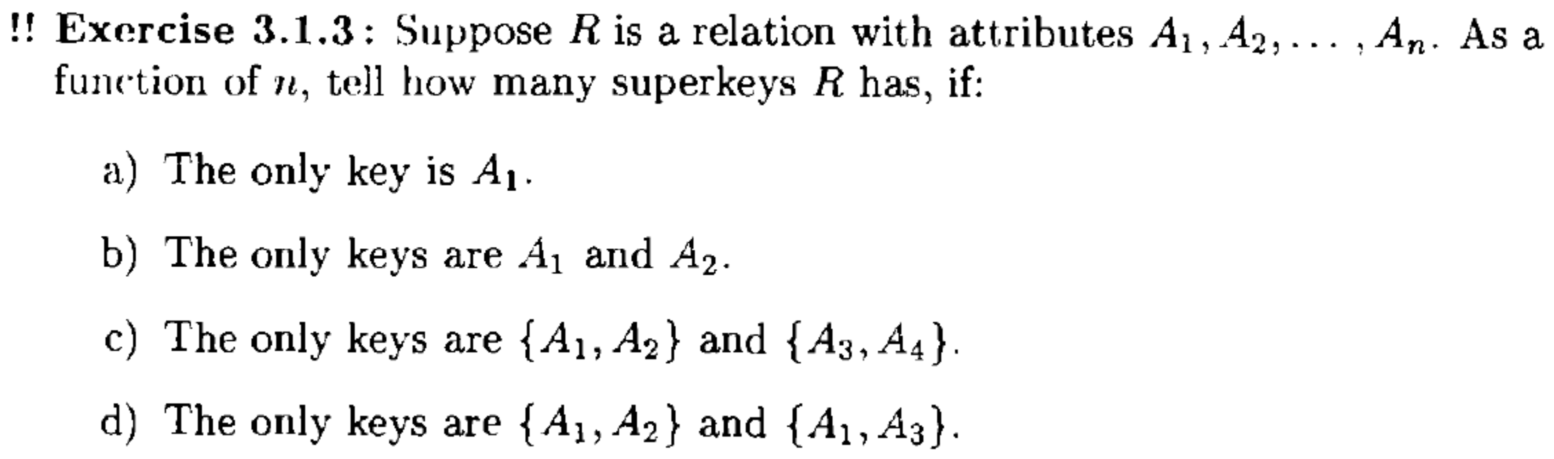
a superkey, but any superset of this set of attributes, such as

{`title`, `year`, `starName`, `length`, `studioName`}

is a superkey. □

Maximum number of super key: 2^N-1

Example3



!! **Exercise 3.1.3 :** Suppose R is a relation with attributes $A_1, A_2, \ldots, A_n$. As a function of n, tell how many superkeys R has, if:

a) The only key is A_1.

b) The only keys are A_1 and A_2.

c) The only keys are $\{A_1, A_2\}$ and $\{A_3, A_4\}$.

d) The only keys are $\{A_1, A_2\}$ and $\{A_1, A_3\}$.

In general, if we have 'N' attributes with one candidate key then the number of **possible** superkeys is $2^{(N-1)}$.

Let a Relation R have attributes {a1, a2, a3,…, an} and the candidate keys are "a1 a2", "a3 a4" then the possible number of super keys?
Super keys of(a1 a2) + Super keys of(a3 a4) – Super keys of(a1 a2 a3 a4)
$\Rightarrow 2^{(N-2)} + 2^{(N-2)} - 2^{(N-4)}$

Let a Relation R have attributes {a1, a2, a3,…, an} and the candidate keys are "a1 a2", "a1 a3" then the possible number of
Super keys of (a1 a2) + Super keys of (a1 a3) – Super keys of(a1 a2 a3)
$\Rightarrow$ 2^(N – 2) + 2^(N – 2) – 2^(N – 3)

Example

How many possible superkeys do we have in this example?

- {A} is a super key. (the values are not repeated)
- So , {A,B} is superkey, since A is a superkey.
- {A,C} , {A,D},{A,B,C},{A,C,D},{A,B,D},{A,B,C,D}
- Order does not matter
- Is {B}is a super key? No {C}No {D}No
- {B,C,D} No. So subset of {B,C,D} can not be a superkey
- Answer is 8

A	B	C	D
1	2	3	1
2	2	7	1
3	2	7	1
4	7	7	1
5	7	3	1
6	7	3	8

Candidate Key

- Is a superkey whose proper subset is **not** a superkey. (minimal super key)

SK= {A},{A,B},{A,C},{A,B,C}
{B} OR {C} NO
SK={B,C}
SK= {A},{A,B},{A,C},{A,B,C}, {B,C}

A	B	C
1	6	3
2	6	5
3	1	3
4	1	5

proper subset :
Suppose X1={1,2,3} and X2={1,2}
X2 is **subset** of x1 if every member of X2 must be member of X1
X2 is **proper subset** of x1
First x2 is subset of x1
But x1 is not subset of x2

So every CK is a SK
But every SK is not a CK

{A,B,C} : WHOSE proper subset are {A,B}, {B,C},{A,C},{A}, {B}, {C} CK=NO SOME ARE SUPERKEYS
{A,C}: WHOSE proper subset are {A},{C} CK=NO SOME ARE SUPERKEYS
{A}: CK=YES {B,C}: WHOSE proper subset are{B} ,{C} none of its proper subset is sk CK=YES

Rules about Functional Dependencies

- The ability to discover additional FD ‘s is essential when we discuss the design of good relation schemas

1- Reasoning about Functional Dependencies

Example 3.4: If we are told that a relation $R(A, B, C)$ satisfies the FD’s $A \rightarrow B$ and $B \rightarrow C$, then we can deduce that R also satisfies the FD $A \rightarrow C$. How does that reasoning go? To prove that $A \rightarrow C$, we must consider two tuples of R that agree on A and prove they also agree on C.

2- The splitting/combining rule

We can replace an FD $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$ by a set of FD’s $A_1A_2 \cdots A_n \rightarrow B_i$ for $i = 1, 2, \ldots, m$. This transformation we call the *splitting rule.*

We can replace a set of FD’s $A_1A_2 \cdots A_n \rightarrow B_i$ for $i = 1, 2, \ldots, m$ by the single FD $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$. We call this transformation the *combining rule.*

Example 3.5: In Example 3.1 the set of FD’s:

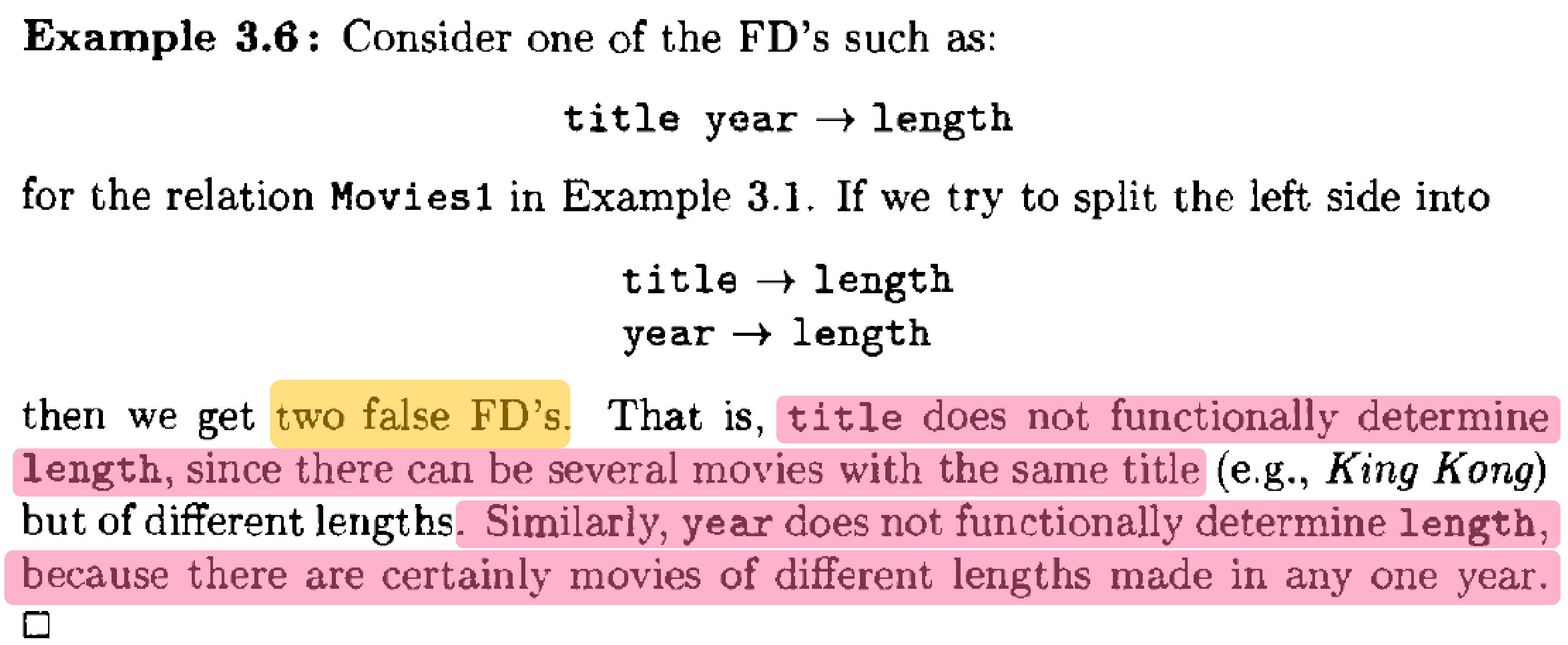
```
title year → length
title year → genre
title year → studioName
```

is equivalent to the single FD:

```
title year → length genre studioName
```

that we asserted there. □

- However, there is no splitting rule for left sides



Example 3.6: Consider one of the FD's such as:

$$\texttt{title year} \rightarrow \texttt{length}$$

for the relation `Movies1` in Example 3.1. If we try to split the left side into

$$\texttt{title} \rightarrow \texttt{length}$$
$$\texttt{year} \rightarrow \texttt{length}$$

then we get two false FD's. That is, `title` does not functionally determine `length`, since there can be several movies with the same title (e.g., *King Kong*) but of different lengths. Similarly, `year` does not functionally determine `length`, because there are certainly movies of different lengths made in any one year.
□

Derivation rules

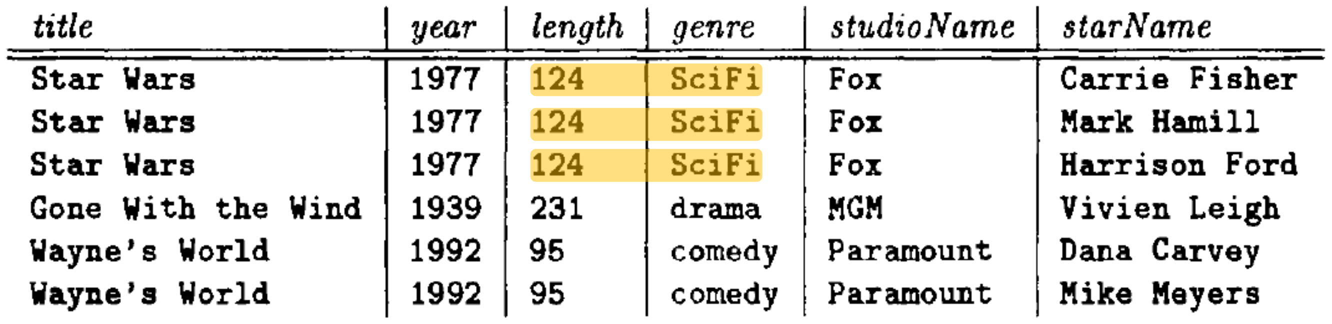
- X, Y, Z are subsets of U
- **Reflexivity**
  - if X $\subseteq$ Y $\subseteq$ U then Y --> X
- **Augmentation**
  - if X-->Y and Z $\subseteq$ U then X,Z -->Y,Z
- **Transitivity**
  - if X -->Y and Y --> Z then X --> Z
- **Pseudo - transitivity**
  - if X --> Y and Y,W --> Z then X,W --> Z
- **Union**
  - if X --> Y and X --> Z then X --> Y, Z
- **Decomposition**
  - if X --> Y and Z $\subseteq$ Y then X --> Z

Anomalies

- Problems such as redundancy that occur when we try to cram too much into a single relation are called ***anomalies***:

1. Redundancy
   - Information may be repeated unnecessarily in several tuples.
2. Update anomalies
   - We may change information in one tuple but leave the same information unchanged in another.
3. Deletion anomalies
   - If a set of values becomes empty, we may lose other information as a side effect.

Examples



title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Redundancy

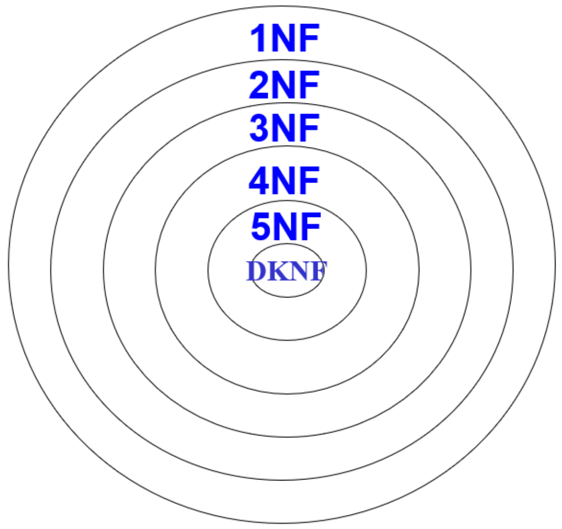
Figure 3.2: An instance of the relation Movies1(title, year, length, genre, studioName, starName)

- **Update anomaly**: If we found that star wars is really 125 minutes long, we might **carelessly** change the length in the first tuples but not in the second and third tuples.

Normalization Algorithms

Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain Key Normal Form (DKNF)



Each higher level is a subset of the lower level

Most databases should be 3NF or BCNF in order to avoid the database anomalies.

First normal form

- A relation schema is in first normal form (1 NF) if any attribute has an atomic value.
- **1:Atomic value :** It cannot be decomposed into two or more component

-Create separate column for each member of composite attribute
-Make two or more different tuples for each multi-value attributes
- Define Fk

SID	Name	Add	Tel
1	x1	F1,a1,b1	T1,T2
2	x2	F1,a2,b5	T5
3	x3	F2,a3,b4	T6,T3

It is not atomic

- **2:In first normal form : A column should contain values for the same domain.**
- **3:Each column should have unique name**
- **4: NO ordering, No duplicate rows**

Closure of a set of FDs (F^+)

- The closure of F, said $\mathbf{F^+}$, is the set of all FD that can be derived from F
- Using **attributes closure** can help to answer , it is a **candidate** key or not
- Then by finding a candidate key we can solve the 2NF , 3NF , ….

R(A,B,C,D,E,F) FD={A->B , B->C, C->D, D->E}

We can use rules and find more FD

A->B, B->C ➔ A->C

A->A (Reflexivity)

A->C, C->D ➔ A->D

A->D, D->E ➔ A->E

A->ABCDE (splitting/merge)

x is a set of attribute

X^+ contains set of attributes determined by X

A^+={A,B,C,D,E}

R(A,B,C,D,E)   FD={A->B , B->C, C->D, D->E}

AD^+=?

AD->A

AD->D

A->B ➔ AD->BD

AD->BD ➔ AD->B , AD->D

AD->B, B->C ➔ AD->C

AD^+={A,B,D,C,E}

CD^+

{C,D, ….}

D->E

CD^+={C,D, E}

B^+={B,C,D,E}

Superkey
Set of attributes whose closure contains all attributes of a given relation

A+ and AD+ are SK