Advanced topics in Databases

Hellenic Mediterranean University

Prof. Demos Akoumianakis (da@hmu.gr)

Agenda

- ✓ *Functional dependencies between attributes*
  - *Definition*
  - *Representation, sources and identification*
- This week
  - Types and properties
  - Amstrong's axioms
  - Relational decomposition
    - Characteristics
    - Validating decompositions

Types of functional dependencies

Primary types of dependencies

- Trivial / non-trivial dependencies
- Partial / full functional dependencies
- Transitive functional dependencies
- Multivalued dependencies

Trivial dependencies

- Trivial dependency occurs if the dependent is a subset of the determinant
  - ✓ For $X \rightarrow Y$, if $Y \subseteq X$, then $X \rightarrow Y$ is trivial
- Examples of trivial dependencies for R = {A, B}

  $\{A\} \rightarrow \{A\}$

  $\{A,B\} \rightarrow \{A\}$

  $\{A,B\} \rightarrow \{B\}$

  $\{A,B\} \rightarrow \{A,B\}$
- In the Netflix dataset
  - Show_id, title, release_year → title

Non-trivial dependencies

- Non-trivial dependencies occur if the dependent is not a subset of the determinant
  - ✓ $X \rightarrow Y$ is non-trivial if $Y \not\subset X$
- In the Netflix dataset
  - Show_id → rating, release_year

Full functional dependencies

- Full functional dependency occurs when the dependent depends on the entire set and not any part of the determinant
  - ✓ $X \rightarrow Y$ is full if there is *no* W so that $W \subset X$ and $W \rightarrow Y$
- In the Netflix dataset
  - Show_id, title, release_year → duration

Partial functional dependencies

- Partial functional dependency $X \rightarrow Y$ occurs when the dependent attribute depends on only a portion rather than the whole of the determinant
  - ✓ $X \rightarrow Y$ is partial if there is $W \subset X$ so that $W \rightarrow Y$
- In the Netflix dataset
  - show_id, director_id → director_name
    - because director_id → director_name

Transitive dependencies

- A functional dependency $X \rightarrow Y$ is transitive if both X and *Y* are non-key attributes
  - ✓ If A→B and B→C, then A→C is transitive provided that B → A
- In the Netflix dataset
  - Since show_id → title and title → genre_name, show_id → genre_name is transitive

Multivalued dependencies

- Multivalued dependency $X \twoheadrightarrow Y$ designates a constraint where X uniquely determines a *set* of values for Y
  - ✓ If $X \twoheadrightarrow Y$, then $X[i] \rightarrow Y[j]$ and $X[i] \rightarrow Y[k]$
- Properties
  - If $\alpha \twoheadrightarrow \beta$, then hen $\alpha \twoheadrightarrow R - \beta$
  - If $\alpha \twoheadrightarrow \beta$ and $\gamma \subseteq \delta$, then $\alpha\delta \twoheadrightarrow \beta\gamma$
  - If $\alpha \twoheadrightarrow \beta$ and $\beta \twoheadrightarrow \gamma$, then $\alpha \twoheadrightarrow \gamma - \beta$
- In the Netflix dataset
  - title ↠ listed_in
    - e.g. a title may be listed in drama and documentaries
  - title ↠ country
    - e.g. a title may be produced in UK and Greece

Amstrong’s axioms

What they are

- Often, there are functional dependencies that can be derived from other functional dependencies
- It is therefore useful to determine inference rules which reveal all functional dependencies F^+ holding over a given set F of functional dependencies
- Amstrong established a small set of such rules and proved they are
  - Soundness
    - Every dependency revealed by these axioms is guaranteed to be valid in any database state that satisfies the initial rules
  - Completeness
    - Repeatedly applying these rules will eventually reveal every possible functional dependency that can be logically inferred—nothing is missed

What the axioms offer

- Amstrong's axioms are used to reveal
  - All functional dependencies (F^+) that can be derived from a given set F;
    - the result is known as closure of F
  - Canonical Covers which are reduced sets of functional dependencies that still hold the same information;
    - also known as minimal cover
  - Candidate Keys (and therefore primary keys) by revealing which attributes can determine all other attributes in a relation
  - Normalization Requirements which determine attributes that are fully or partially dependent, enabling proper decomposition into 2NF, 3NF, and BCNF to remove redundancy

Amstrong's basic axioms

- Let X, Y and Z be attribute subsets of a relation R, then the following hold:
  - Reflexivity (Ανακλαστική ιδιότητα)
    - If $Y \subseteq X$ then $X \rightarrow Y$
  - Augmentation (Επαυξητική ιδιότητα)
    - If $X \rightarrow Y$ then $XZ \rightarrow YZ$
  - Transitivity (Μεταβατική ιδιότητα)
    - If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Additional axioms (cont.)

- We can also prove that the following hold (on the basis of the basic axioms)
  - Additivity or Union (Ενωτική ιδιότητα)
    - If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
  - Decomposition (Διασπαστική ιδιότητα)
    - If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$
  - Pseudotransitivity (Ψευδομεταβατική ιδιότητα)
    - If $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$
  - Composition (Συνθετική ιδιότητα)
    - If $X \rightarrow Y$ and $Z \rightarrow W$ then $XZ \rightarrow YW$

Consolidation

- So far we have seen
  - How we can identify various types of dependencies in a data set
  - Axioms that allow inference
- Question
  - How do the above contribute to designing databases schemas?

Decomposing relations using functional dependencies

Task

- Given a universal relation and a set of functional dependencies, determine a suitable relational decomposition
- We will make use of the Kaggle dataset about netflix titles

The single relation schema

- Universal relation
  - R(show_id, title, type, release_year, rating, duration, description, date_added, director_id, director_name, actor_id, actor_name, country_id, country_name, genre_id, genre_name)

Analysis

- Universal relation
  - R(show_id, title, type, release_year, rating, duration, description, date_added, director_id, director_name, actor_id, actor_name, country_id, country_name, genre_id, genre_name)
- Functional dependencies
  - ✓ show_id $\rightarrow$ title, type, release_year, rating, duration, description, date_added
  - ✓ genre_id $\rightarrow$ genre_name
  - ✓ actor_id $\rightarrow$ actor_name
  - ✓ director_id $\rightarrow$ director_name
  - ✓ country code $\rightarrow$ country name

Analysis

- Universal relation
  - R(show_id, title, type, release_year, rating, duration, description, date_added, director_id, director_name, actor_id, actor_name, country_id, country_name, genre_id, genre_name)
- Multivalued dependencies
  - ✓ show_id ↠ genre_id
  - ✓ show_id ↠ actor_id
  - ✓ show_id ↠ director_id
  - ✓ show_id ↠ country_code

Decomposition

R = {show_id, title, type, release_year, rating, duration, description, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

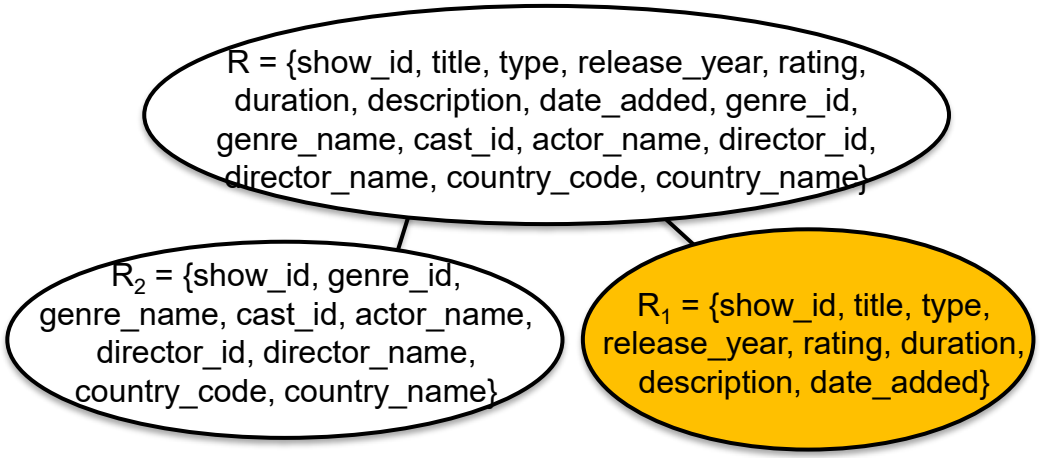
R = {show_id, title, type, release_year, rating, duration, description, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Decomposition

R = {show_id, title, type, release_year, rating, duration, description, date_added, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Full (non-trivial) functional dependencies

show_id → title, type, release_year, rating, duration, description, date_added



- For $X \rightarrow Y$ split the schema into $R_1 = X \cup Y$ & $R_2 = R - Y$

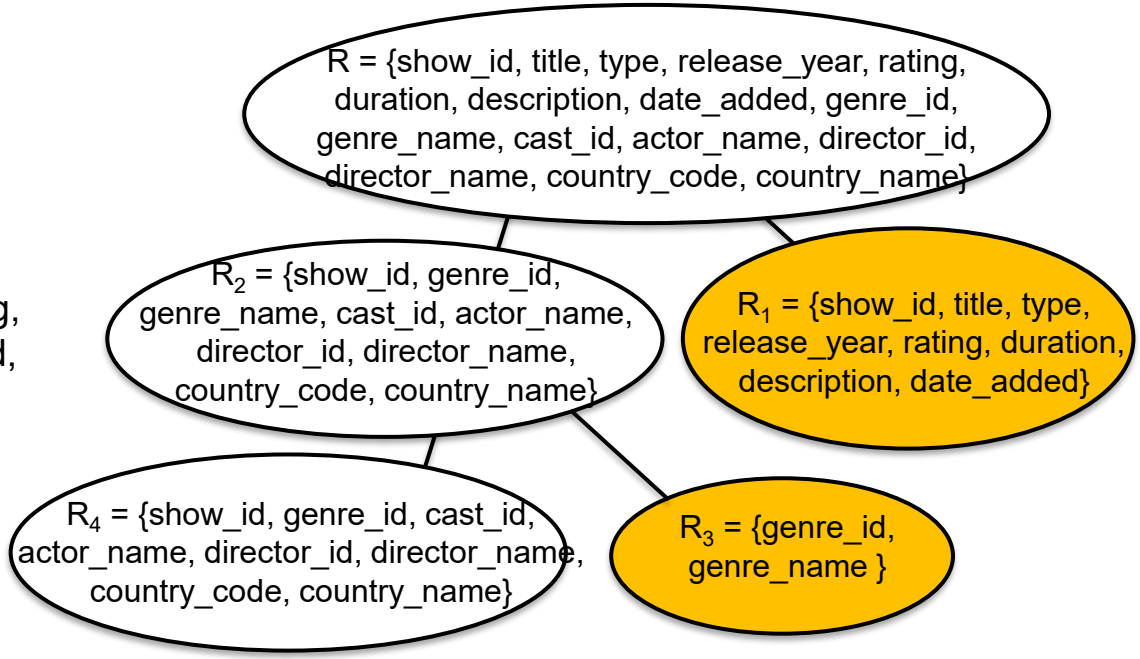
Decomposition

R = {show_id, title, type, release_year, rating, duration, description, date_added, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Full (non-trivial) functional dependencies

show_id → title, type, release_year, rating, duration, description, date_added

genre_id → genre_name

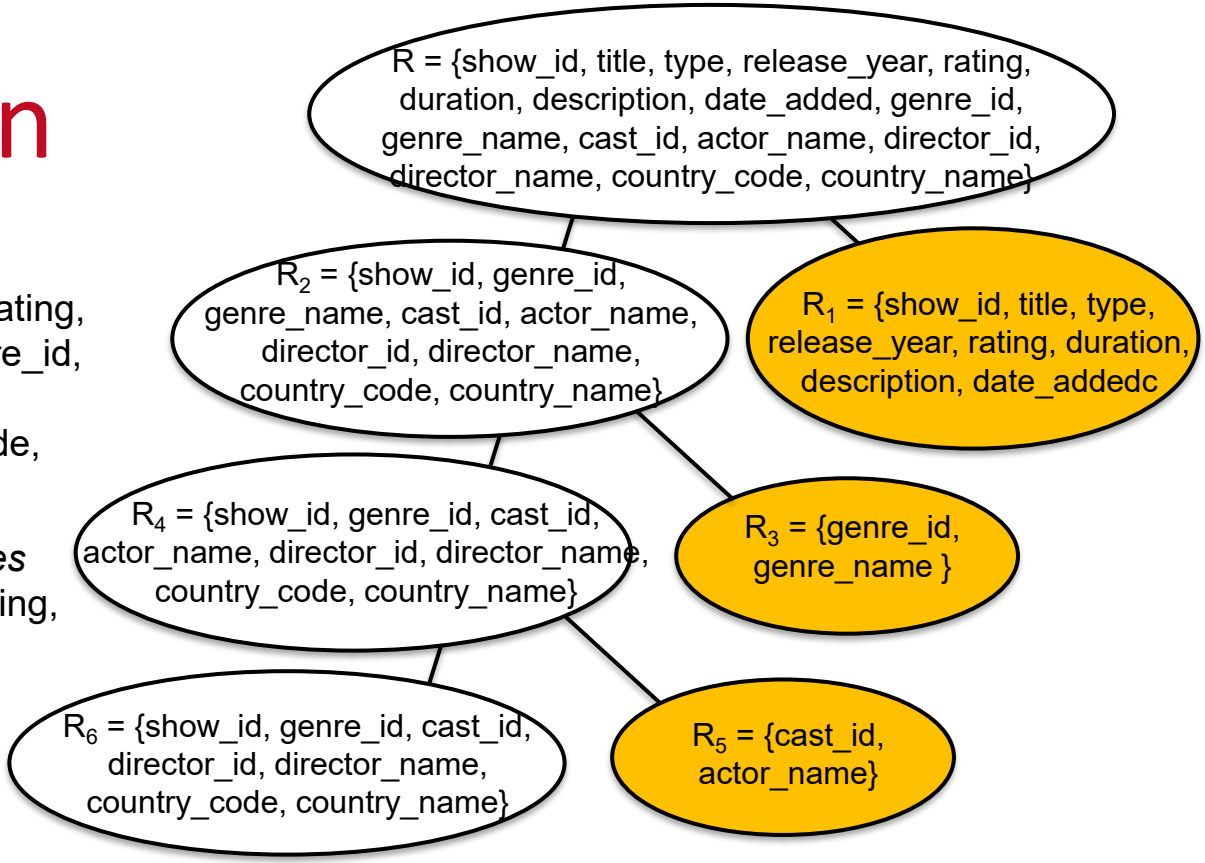


- For $X \rightarrow Y$ split the schema into $R_1 = X \cup Y$ & $R_2 = R - Y$

Decomposition

R = {show_id, title, type, release_year, rating, duration, description, date_added, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Full (non-trivial) functional dependencies
show_id → title, type, release_year, rating, duration, description, date_added
genre_id → genre_name
cast_id → actor_name

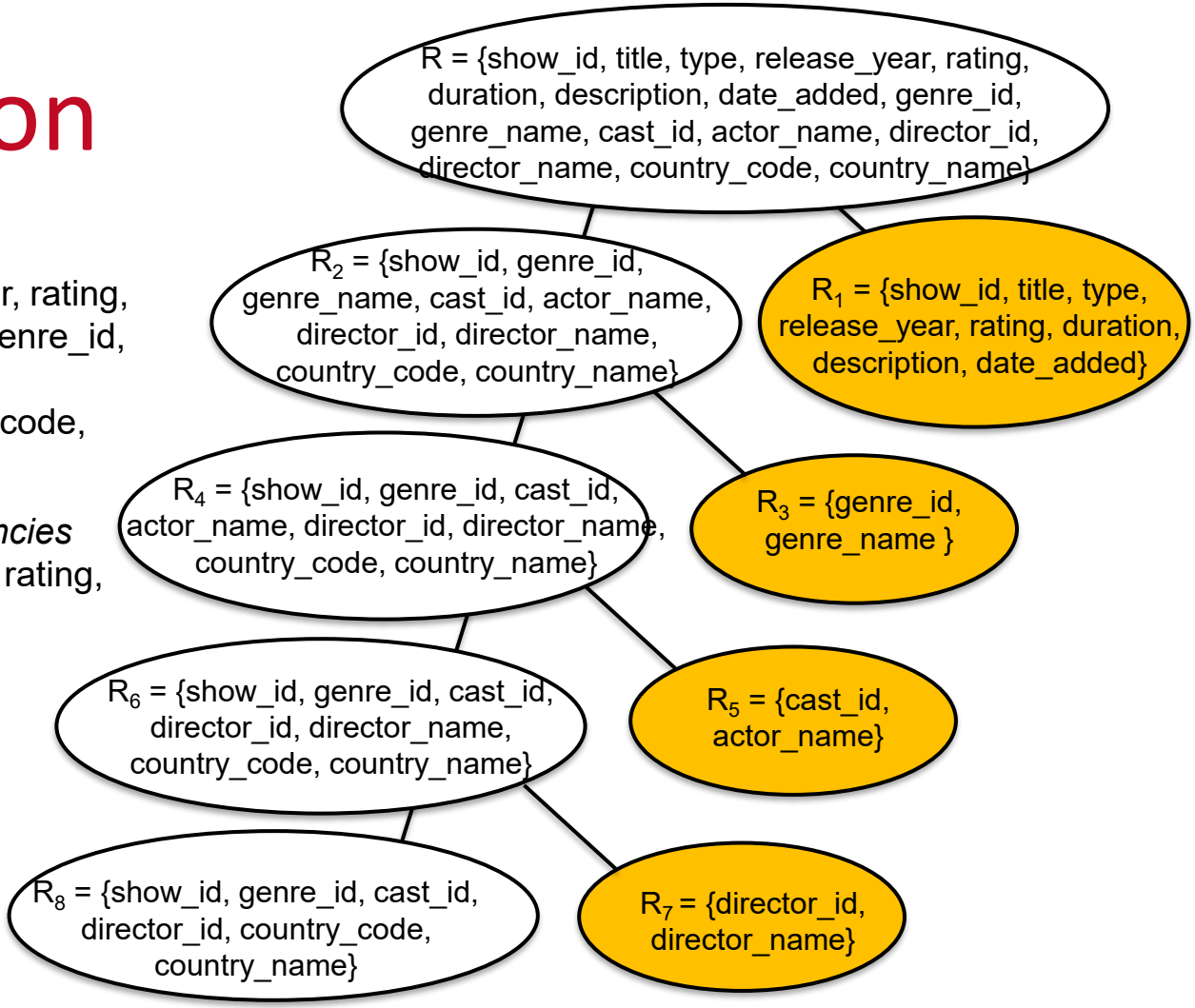


- For $X \rightarrow Y$ split the schema into $R_1 = X \cup Y$ & $R_2 = R - Y$

Decomposition

R = {show_id, title, type, release_year, rating, duration, description, date_added, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Full (non-trivial) functional dependencies
show_id → title, type, release_year, rating, duration, description, date_added
genre_id → genre_name
cast_id → actor_name
director_id → director_name



- For $X \rightarrow Y$ split the schema into $R_1 = X \cup Y$ & $R_2 = R - Y$

Decomposition

R = {show_id, title, type, release_year, rating, duration, description, date_added, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Full (non-trivial) functional dependencies

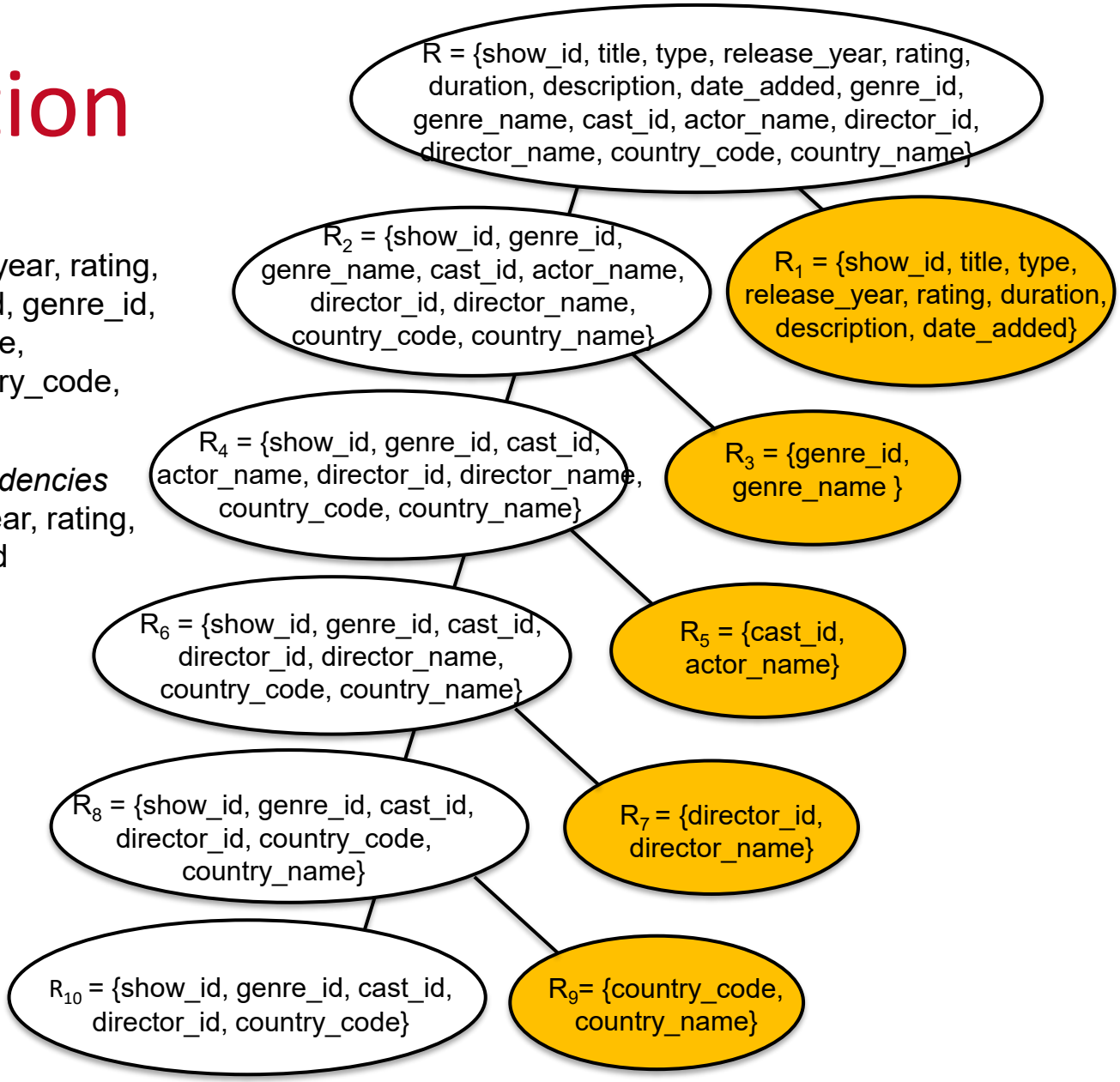
show_id → title, type, release_year, rating, duration, description, date_added

genre_id → genre_name

cast_id → actor_name

director_id → director_name

country code → country name



- For $X \rightarrow Y$ split the schema into $R_1 = X \cup Y$ & $R_2 = R - Y$

Decomposition

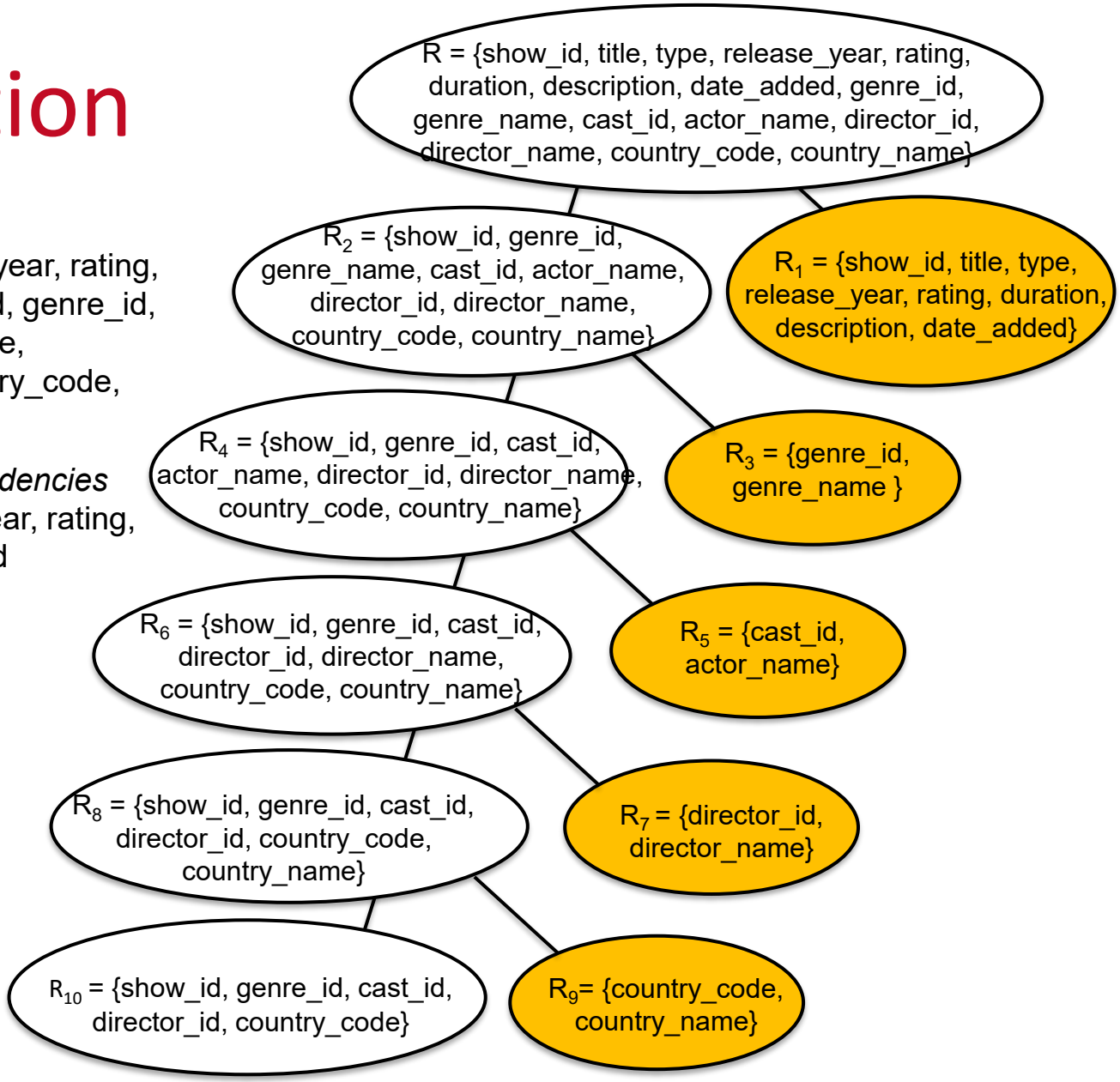
R = {show_id, title, type, release_year, rating, duration, description, date_added, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Full (non-trivial) functional dependencies

show_id → title, type, release_year, rating, duration, description, date_added
genre_id → genre_name
cast_id → actor_name
director_id → director_name
country code → country name

Multi-valued dependencies

show_id ↠ genre_id
show_id ↠ actor_id
show_id ↠ director_id
show_id ↠ country_code



- For X ↠ Y create new schema R′ = {X , Y}

Decomposition

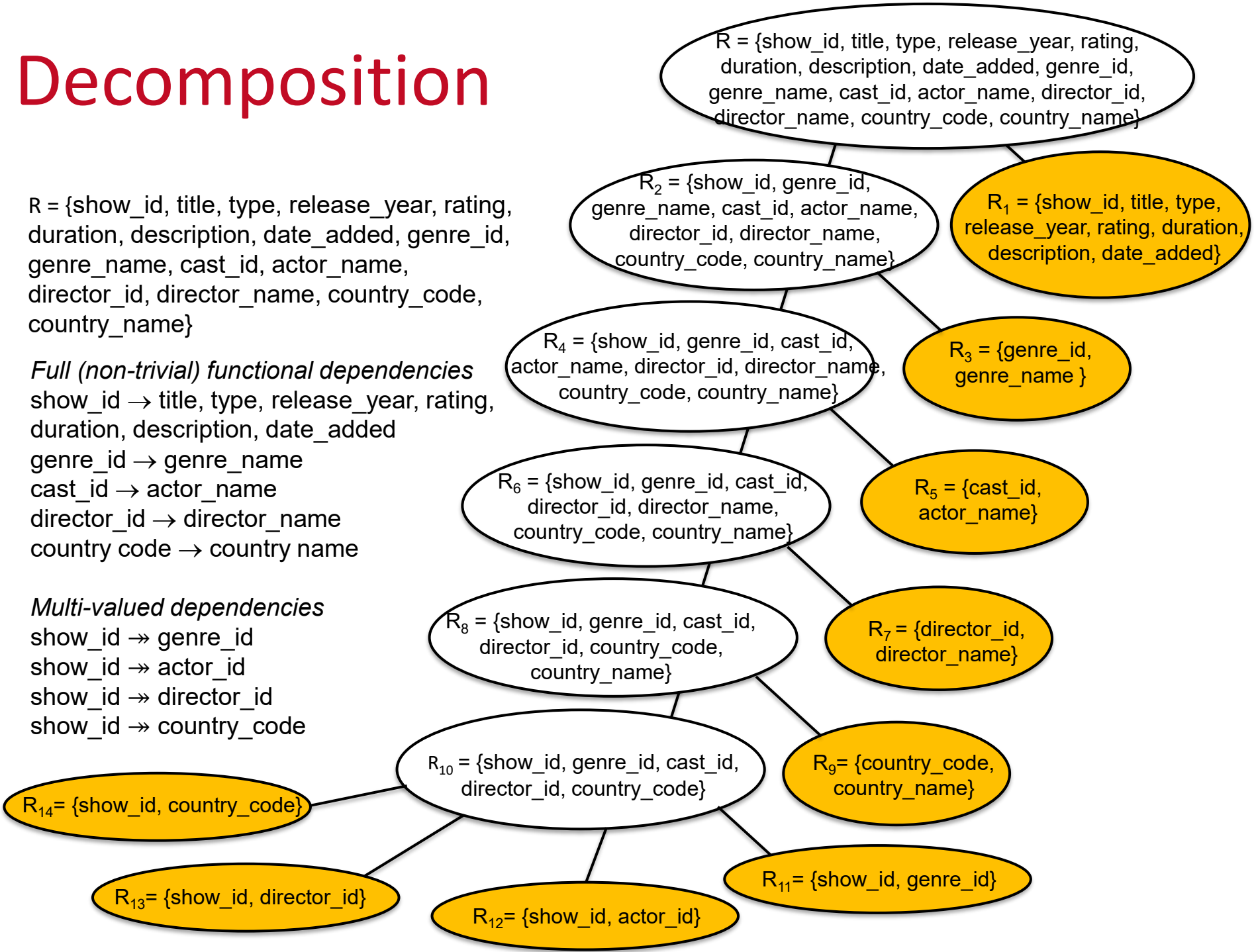
R = {show_id, title, type, release_year, rating, duration, description, date_added, genre_id, genre_name, cast_id, actor_name, director_id, director_name, country_code, country_name}

Full (non-trivial) functional dependencies

show_id → title, type, release_year, rating, duration, description, date_added
genre_id → genre_name
cast_id → actor_name
director_id → director_name
country code → country name

Multi-valued dependencies

show_id ↠ genre_id
show_id ↠ actor_id
show_id ↠ director_id
show_id ↠ country_code



The final decomposition

- Decomposition into tables
  - Shows (**show_id**, title, type, release_year, rating, duration, description, date_added)
  - Genres (**genre_id**, genre_name)
  - Cast (**cast_id**, actor_name)
  - Director (**director_id**, director_name)
  - Country (**country_code**, country_name)
  - Show_genre (**show_id**, **genre_id**)
  - Show_actor (**show_id**, **actor_id**)
  - Show_director (**show_id**, **director_id**)
  - Show_country (**show_id**, **country_code**)

Another problem

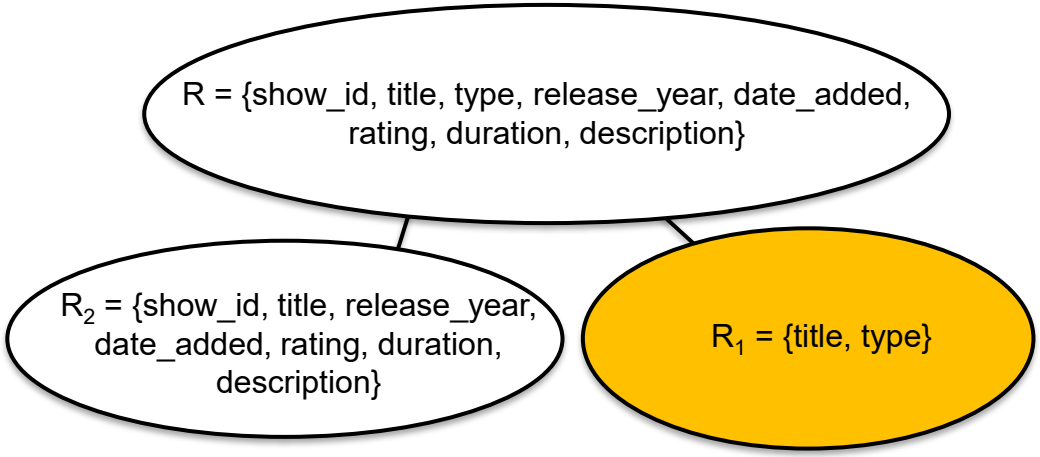
Example with no MVDs

A subset of the Netflix data set – No MVDs

- The same relation
  - R = {show_id, title, type, release_year, date_added, rating, duration, description}
- Two functional dependencies
  - title $\rightarrow$ type
  - show_id $\rightarrow$ title, release_year, date_added, rating, duration, description

Decomposition

R = {show_id, title, type, release_year, date_added, rating, duration, description}



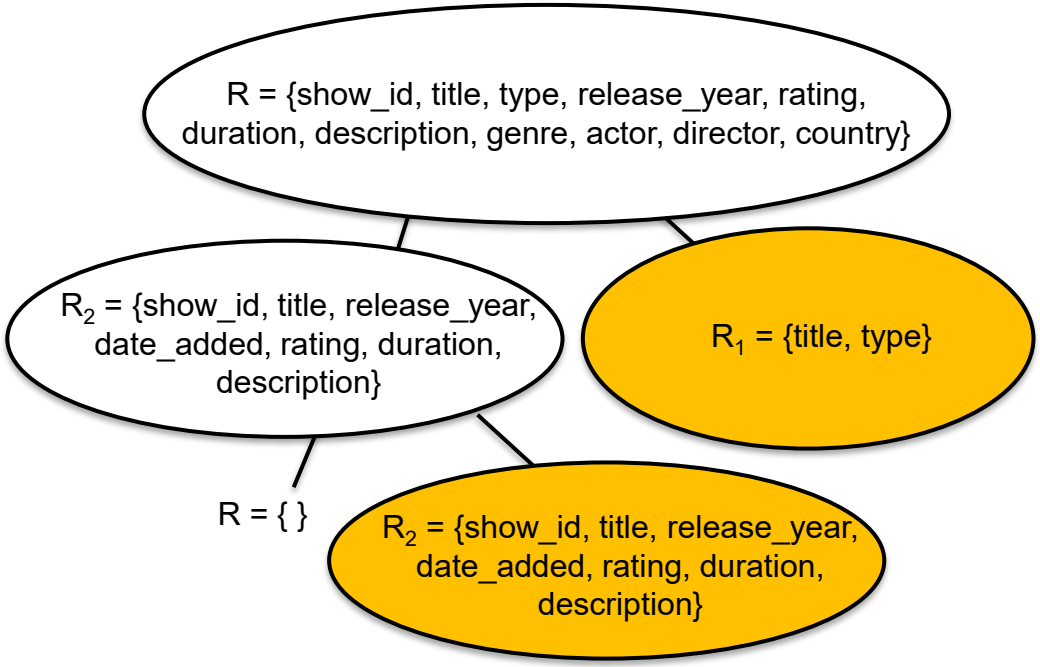
Full (non-trivial) functional dependencies
title → type

- For $X \rightarrow Y$ split the schema into $R_1 = X \cup Y$ & $R_2 = R - Y$

Decomposition

R = {show_id, title, type, release_year, date_added, rating, duration, description}

Full (non-trivial) functional dependencies
title → type
show_id → title, release_year, date_added, rating, duration, description



- Decomposition
  - R2 (show_id, title, release_year, date_added, rating, duration, description)
  - R1 (title, type)

Conclusion

- There may be several decompositions resulting and therefore we need a method to assess which is the best

Validating decompositions

Properties of good decompositions

What is the issue

- Identifying optimal decompositions
  - ✓ Attribute – preserving decompositions (διατήρηση γνωρισμάτων)
    - Δεν πρέπει να υπάρχει γνώρισμα στο καθολικό σχήμα που να μην εμφανίζεται σε μια (τουλάχιστον) από τις παραγόμενες σχέσεις
  - ✓ Dependency – preserving decompositions (διατήρηση εξαρτήσεων)
    - Όποιες συναρτησιακές εξαρτήσεις ισχύουν ή είναι παραγόμενες στο καθολικό σχήμα πρέπει να ισχύουν και στην αποσύνθεση
  - ✓ Lossless-Join decomposition (συνένωση άνευ απωλειών)
    - Από μια δεδομένη αποσύνθεση θα πρέπει να μπορούμε να παράγουμε ακριβώς το αρχικό καθολικό σχήμα

Attribute – preserving decomposition

- Given a relation R which is decomposed into R_1 , R_2 , R_3 , R_v, the decomposition is attribute-preserving if $R = R_1 \cup R_2 \cup R_3 \cup \ldots \cup R_v$
- It ensures that no data fields are "lost" during the decomposition and therefore

Dependency – preserving decomposition

- A decomposition ensuring all original functional dependencies (FDs) are still enforceable locally within the new tables
- Original functional dependencies are all those identified or those implied too?
  - The full set
- We need a method to compute all functional dependencies implied !

Closure

- Η απόδειξη της ιδιότητας της διατήρησης των εξαρτήσεων βασίζεται και αξιοποιεί την έννοια της κλειστότητας (ή θήκης) συνόλου συναρτησιακών εξαρτήσεων
- Κλείσιμο ή θήκη συνόλου F
  - Αν F είναι ένα σύνολο συναρτησιακών εξαρτήσεων τότε το κλείσιμο (closure) του F συμβολίζεται με F+ και περιλαμβάνει όλες τις συναρτησιακές εξαρτήσεις που *υπονοούνται* ή *παράγονται* από το F με τους κανόνες συμπερασμού

$$(R_1 \cup R_2 \cup \ldots \cup R_v)^+ = F^+$$

Lossless-Join decomposition

- Η αποσύνθεση πρέπει να είναι χωρίς απώλειες στη συνένωση
  - Μια αποσύνθεση της σχέσης *R = (A, B)* σε R_1 *= (A)* και R_2 *= (B)* είναι χωρίς απώλειες στη συνένωση όταν η φυσική συνένωση των R_1 και R_2 *ανακατασκευάζει το αρχικό σχήμα της R*

$$R = \Pi_A (R_1) \bowtie \Pi_B (R_2)$$

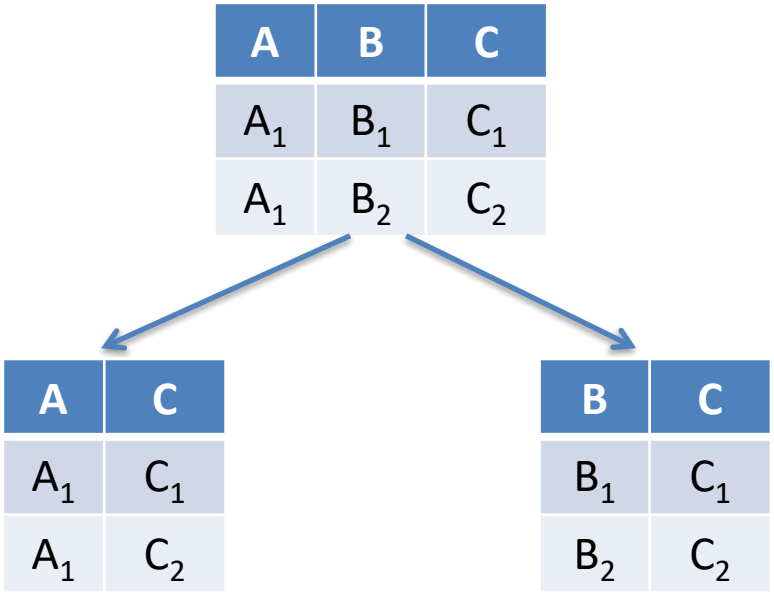
An example

- Let us assume the table R

A	B	C
A_1	B_1	C_1
A_1	B_2	C_2

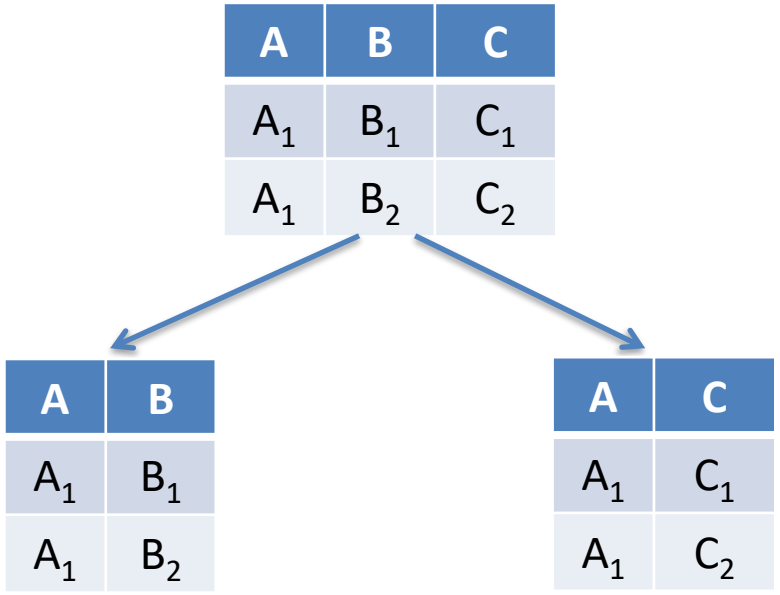
An example

- One possible decomposition



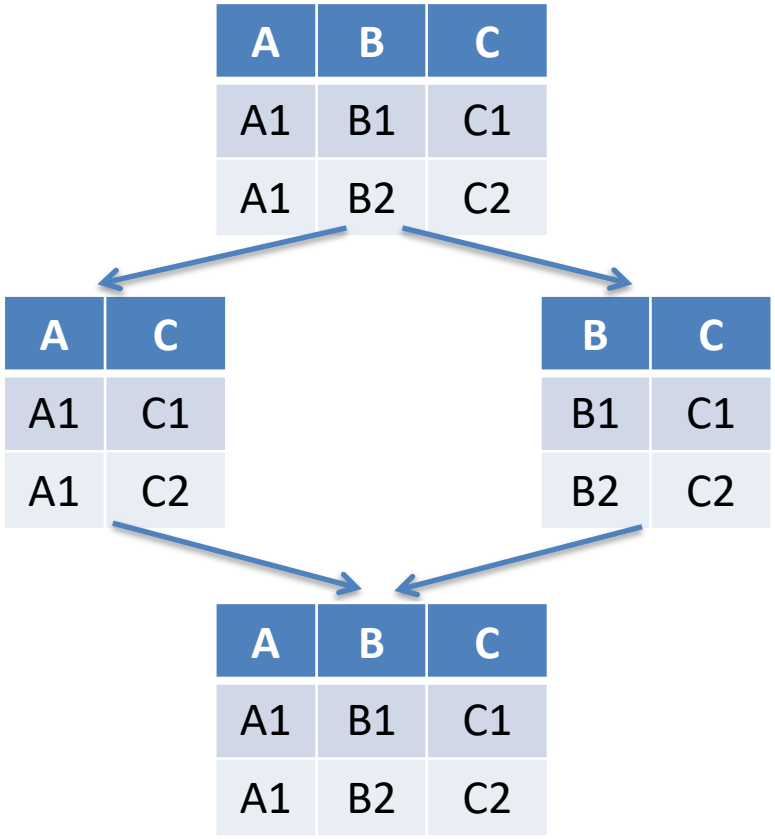
An example

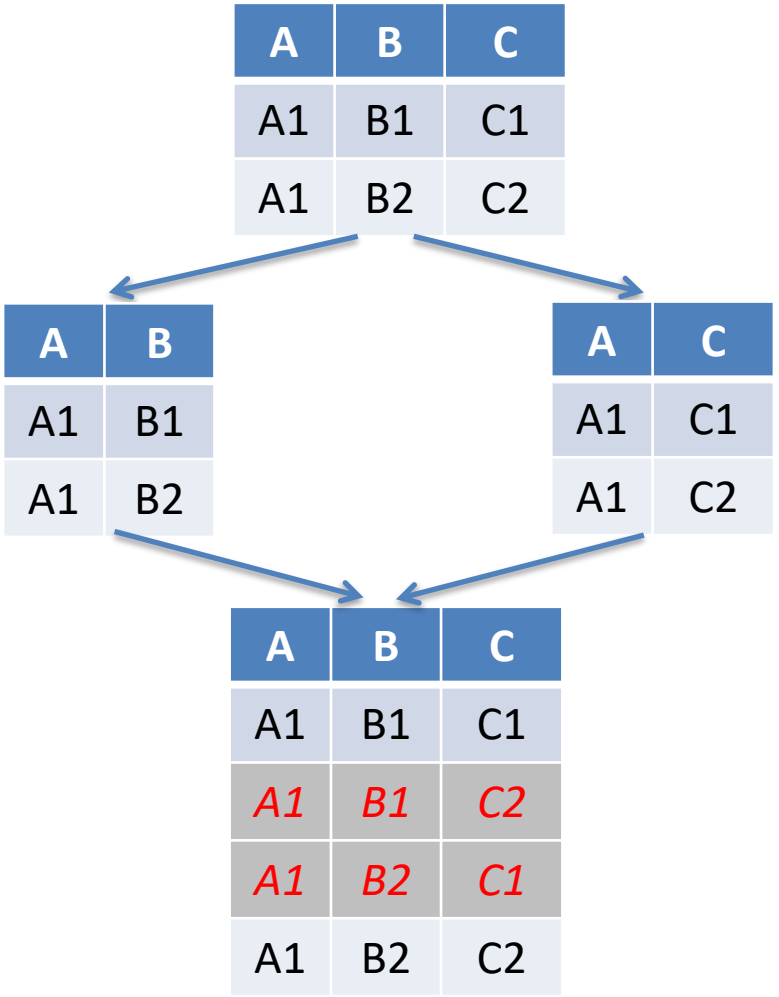
- Another possible decomposition



An example

- Only one is Lossless-Join decomposition





Lossless-Join decomposition theorem

- If R is a relation and F the set of functional dependencies on R.
- For any given decomposition into R_1 and R_2
  - If $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$ belongs to F^+ then the decomposition is a Lossless-Join decomposition
    - In other words, the common attributes of R_1 and R_2 need to be primary key for at least one of the R_1 and R_2

Conclusions

- Decompositions which are lossless-join and dependency preserving
  - 2NF
- Decompositions which are lossless-join may not be dependency preserving
  - BCNF

Τέλος για σήμερα - Ερωτήσεις