DATABASE RECAP

A functional dependency is a constraint between two sets of attribute inside the same relation.

Def

Given $R(X,Y)$ (R: Relation, X: set, Y: set) X functionally determines Y (written $X \to Y$) $X \subseteq X$, $Y \subseteq Y$ // The values actually stored

// iff

$x_i \in X \mapsto y_i \in Y \iff \Pi_{X,Y} R$ is a function (determined by x_i)

$\iff$ Two tuples sharing the same values x_i's have to share values for y_i's for function

Heath's Theorem: lossless decomposition

A relation R over the set of attributes U that satisfy the functional dependency $X \to Y$ can be losslessly joined decomposed as

$$\Pi_{XY}(R) \bowtie \Pi_{XZ}(R) = R$$

where $Z = U - XY$

Candidate Key

A candidate key is the minimal set of attributes that functionally determine all the attributes in a relation

Functional dependencies and attribute domains are selected so as to generate constraint that would exclude all the inappropriate data items for the user domain from the system

Logical Implication

A set Σ of functional dependencies logically implies another set Γ of dependencies, is called $\Sigma \models \Gamma$ logical implication.

Armstrong's axiomatization: a sound & for logical implication

R relation.

X, Y, Z sets of attributes.

Primary Rules

- $Y \subseteq X \Rightarrow X \to Y$ Reflexivity
- $X \to Y \Rightarrow XZ \to YZ$ Augmentation
- $X \to Y,\ Y \to Z \Rightarrow X \to Z$ Transitivity

Secondary Rules: (obtainable from the former)

- Union: $X \to Y \wedge X \to Z \Rightarrow X \to YZ$

Contrast using FD:

- Decomposition: $X \to YZ \Rightarrow X \to Y \wedge X \to Z$
- Pseudotransitivity: $X \to Y \wedge WY \to Z \Rightarrow WX \to Z$

Another handy rule:

- Composition: $X \to Y,\ Z \to V \Rightarrow XZ \to YV$

Closure of functional dependency

R relation. F set of FDs on R.

The closure F^+ is the set of all attributes determined by X using F.

• Concept: to pave the way through 3NF

• Atomicity: No columns have more than a value (1NF) (maybe a long text separated by commas)

• 2NF: If we have a candidate key {A, B} and there are columns that don't need either A or B for their functional dependency; you're not in 2NF.

SPLIT. Ex: Suppose a book has only an author for the sake of the example:

Book

Title	Format	Author	Price	Pages
Python HS	Hardcover	Giovanni	60 $	520
Python HS Ebook	Ebook	Giovanni	12 $	520

Note that: ~~Title → Author and Format → Price~~

Title, Format → Author, Price, Pages

BUT ~~Title → Author only~~

Format → Price only

Book

Title	Author	Pages
Python HS	Giovanni	520
Python HS	Giovanni	520

Book-Price

Title	Format	Price
Python HS	Hardcover	60 $
Python HS	Ebook	12 $

2NF ✓

• 3NF: 2NF ⊕ No Transitive dependencies ($X \to Y, Y \to Z \Rightarrow X \to Z$)

Example:

Title → Genre ID, Genre Name

BUT this can be deduced by

Title → (Genre ID, Genre ID → Genre Name)

→ Redundancy (Genre ID | Genre Name is a new table)

Example

FDs:

1. $A \to B$
2. $B \to C$
3. $AB \to D$

A^+:

$A \to A$ by reflexivity.

$A \to AB$ by 1.

$A \to ABC$ by 2

$ABC \to ABCD$ by 3.

Def Cover (of FDs)

F covers G if every iff

$F \Rightarrow fd \in FD_G \iff G^+ \subseteq F^+$

Def Equivalence (of FDs)

F, G equivalent

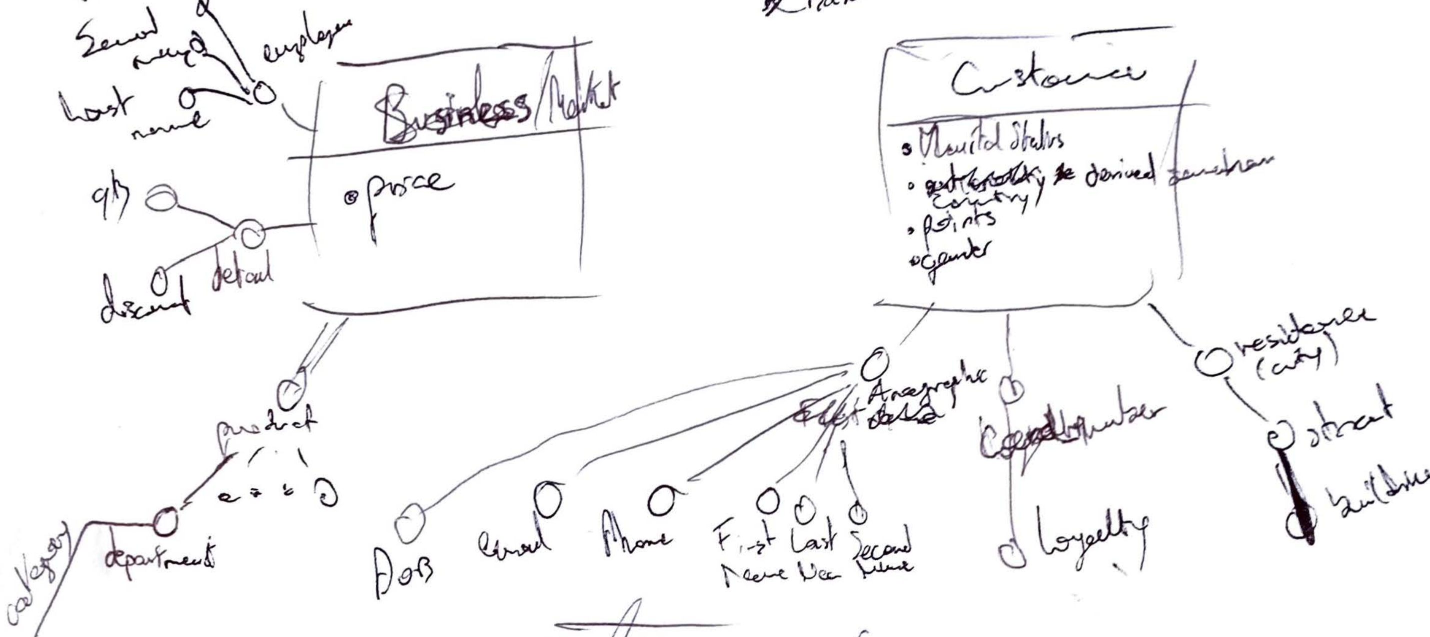
$F \equiv G \iff F^+ = G^+$

SNOW FLAKE

- CONSTELLATION
- 2 VIEWS ARCHITECTURE

ASSUMPTIONS:

- From previous analyses Married > Divorced > Single > Widowed
- " " " Country ⇒ Italian > European > World
  Gender ⇒ Female > NA > Male
  based on makeup and women do the shopping



- Building is an informative attr because the market don't do shipments service (It's a physical store)

- product dimension: now we can count different items not [illegible]

• BCNF

Considered the relation R and FDs, R is in BCNF

iff: ∀ f ∈ FDs

1. f is trivial functional dependency: f X → Y where Y ⊆ X

2. f: X → Y and X is a superkey (primary key or candidate key)

Ex: Title, Author, AuthorNationality, EverythingElse

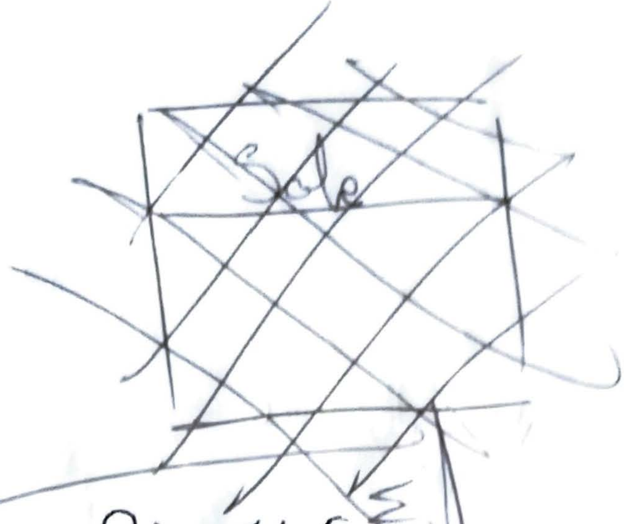
f: Author → AuthorNationality

f doesn't meet 2. so we have redundancy

and we can split

R1: Title, EverythingElse, Author (FK)

R2: Author, AuthorNationality



IN PACTICE

Materialized View

1. Create a normal table A as a result of a query
2. Create an index on it I_A
3. Rename the table A
4. Create view come risultato della query SELECT * FROM A