Introduzione

In queste dispense, dopo aver riportato una sintesi del concetto di Dipendenza Funzionale e di Normalizzazione estratti dal libro Progetto di Basi di Dati Relazionali (Beneventano, Bergamaschi, Guerra, Vincini), viene introdotto il concetto di Denormalizzazione.

♢ Dato uno schema di relazione $R(X)$, una *dipendenza funzionale* su R è un vincolo di integrità espresso nella forma $Y \longrightarrow Z$, dove Y e Z sono sottoinsiemi di X; in tal caso si dice che Y *determina funzionalmente* Z

♢ Un'istanza r di R soddisfa la dipendenza funzionale $Y \longrightarrow Z$ se in ogni tupla di r il valore di Y determina univocamente il valore di Z.

♢ Formalmente, dato uno schema di relazione $R(X)$, un'istanza r di R soddisfa il vincolo di dipendenza funzionale $Y \longrightarrow Z$ su R se e solo se

$$\forall t_1, t_2 \in r, t_1[Y] = t_2[Y] \Longrightarrow t_1[Z] = t_2[Z]$$

♢ È facile verificare che se $Z \subseteq Y$ allora $Y \longrightarrow Z$.

Queste dipendenze funzionali vengono dette **banali**.

♢ È facile verificare che una chiave K di uno schema $R(X)$ determina funzionalmente tutti gli attributi di X:

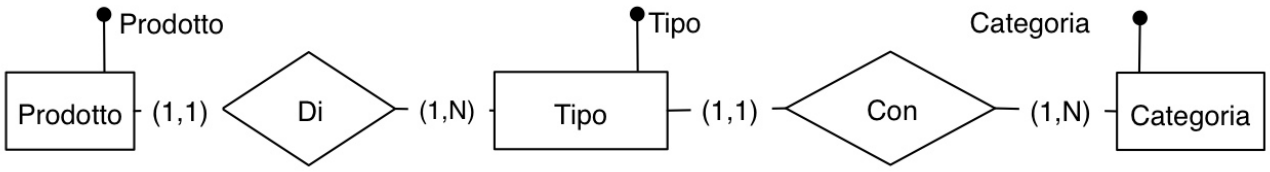
$$\texttt{se } K \subseteq X \texttt{ è chiave di } R(X) \texttt{ allora } K \longrightarrow X$$

♢ Si può verificare che una dipendenza funzionale $Y \longrightarrow Z$,
con $Z = \{A_1, \ldots, A_n\}$ è equivalente ad un insieme di dipendenze funzionali
$Y \longrightarrow A_i$, con $1 \leq i \leq n$.

Quindi nel seguito sono considerate solo dipendenze funzionali $Y \longrightarrow A$, dove A è un singolo attributo.

Esempi di Dipendenze Funzionali

♢ Consideriamo il seguente schema E/R:



♢ Con le **Regole di Progettazione logica relazionale** (vedi capitolo 3.2 del libro Progetto di Basi di Dati Relazionali) questo schema E/R viene tradotto nel seguente schema relazionale:

`CATEGORIA(Categoria)`

`TIPO(Tipo,Categoria)`
`FK : Categoria REFERENCES Categoria`

`PRODOTTO(Prodotto,Tipo)`
`FK : Tipo REFERENCES Tipo`

♢ Le dipendenze funzionali (ricordiamo che per definizione una dipendenza funzionale è relativa ad una singola relazione) sono solo quelle dichiarate dalle chiavi:

Nella relazione `PRODOTTO`: `Prodotto` $\longrightarrow$ `Tipo`

Nella relazione `TIPO`: `Tipo` $\longrightarrow$ `Categoria`

♢ Cosa succede se traduciamo lo schema E/R in relazionale, senza seguire le **Regole di Progettazione logica relazionale**? Ad esempio, riportiamo tutto lo schema E/R in un'unica relazione:

`PRODOTTO(Prodotto,Tipo,Categoria)`

♢ Nella nuova relazione `PRODOTTO` si hanno tre dipendenze funzionali non banali :

`Prodotto` $\longrightarrow$ `Tipo` (dichiarata tramite la chiave)

`Prodotto` $\longrightarrow$ `Categoria` (dichiarata tramite la chiave)

`Tipo` $\longrightarrow$ `Categoria`

♢ Per indicare la terza dipendenza funzionale, che non viene dichiarata dalla chiave, useremo questa notazione:

`PRODOTTO(Prodotto,Tipo,Categoria)`
`FD:` `Tipo` $\longrightarrow$ `Categoria`

♢ Perchè abbiamo sempre utilizzato, nella progettazione di una base di dati, le **Regole di Progettazione logica relazionale**? Perchè non tradurre con una sola relazione `PRODOTTO`? La risposta è nel concetto di normalizzazione.

♢ **NOTA:** Le definizioni e la teoria delle Dipendenza Funzionali è stata estesa anche al modello E/R. Noi non introduciamo questa teoria ma parleremo di dipendenze funzionali negli schemi E/R in modo *intuitivo*, sulla base delle dipendenze funzionali che si ottengono nello schema relazionale tradotto.

Quindi diremo che nello schema E/R `PRODOTTO` determina `TIPO` e `TIPO` determina `CATEGORIA`, ovvero Un'associazione binaria uno-a-molti comporta una dipendenza funzionale.

Forme Normali

♢ Le Forme Normali sono alla base della *teoria della normalizzazione*, il cui obiettivo principale è quello di definire formalmente la qualità degli schemi

♢ La qualità di uno schema viene essenzialmente definita come l'**assenza** di

- **ridondanza nei dati**
- **anomalie di aggiornamento dei dati**

♢ **Esempio 1:** Consideriamo la relazione `FREQUENZA`

`FREQUENZA(MATR,CODCOR,NUMEROORE,CODDOC)`
FD: `CODCOR` $\longrightarrow$ `CODDOC`

MATR	CODCOR	NUMEROORE	CODDOC
M1	C1	102	D1
M1	C3	98	D1
M2	C1	111	D1
M2	C2	101	D2

Problemi:
ridondanza: in tutte le frequenze di un corso si ripete lo stesso docente

anomalia di modifica: se un corso cambia docente si devono modificare tutte le tuple relative alla frequenza di quel corso

anomalia di inserimento: non si può inserire un corso senza frequenze

anomalia di cancellazione: se vengono cancellate tutte le frequenze relative ad un corso si perde anche il docente del corso

- I problemi derivano da `CODCOR` $\longrightarrow$ `CODDOC` che stabilisce la dipendenza di un attributo (`CODDOC`) **solo da una parte** (`CODCOR`) della chiave.

♢ **Esempio 2:** Problemi analoghi si hanno se consideriamo lo schema con una dipendenza funzionale:

`Prodotto(Prodotto,Tipo,Categoria)`
FD: `Tipo` $\longrightarrow$ `Categoria`

- I problemi derivano da `Tipo` $\longrightarrow$ `Categoria` che stabilisce che un attributo *non* chiave (`Tipo`) determina funzionalmente un altro attributo.

Seconda e Terza Forma Normale

Prima Forma Normale - 1NF :

Uno schema $R(X)$ è in **1NF** se e solo se i valori di tutti i domini degli attributi $A \in X$ sono atomici.

Seconda Forma Normale - 2NF :

♢ Informalmente, la 2NF serve per definire schemi esenti dai problemi derivanti dalla dipendenza *parziale* di un attributo dalla chiave.

♢ Dato uno schema di relazione $R(X)$, un attributo $A \in X$ e un insieme di attributi $Y \subseteq X$, si dice che A *dipende completamente* da Y se $Y \longrightarrow A$ e non esiste nessun sottoinsieme proprio Z di Y, $Z \subset Y$, tale che $Z \longrightarrow A$.

♢ Un attributo che appartiene ad una chiave è detto **attributo primo**.

Definizione di Seconda Forma Normale - 2NF :

Uno schema $R(X)$ è in **2NF** se e solo se ogni attributo non primo $A \in X$ *dipende completamente* da ognuna delle chiavi di R.

Terza Forma Normale - 3NF :

♢ Informalmente, la 3NF serve per definire schemi esenti dai problemi derivanti da attributi non chiave che determinano funzionalmente altri attributi.

Definizione di Terza Forma Normale - 3NF :

Uno schema R è in **3NF** se e solo se per ogni dipendenza funzionale non banale $X \longrightarrow A$, o X è una superchiave oppure A è primo.

♢ Si può verificare che uno schema R che è in **3NF** è anche in **2NF**.

♢ Uno schema in **3NF** non è esente da problemi di ridondanza e anomalie:

- **Esempio 3:** `CAMPIONATO(SQUADRA,PARTITA,GIOCATORE,RUOLO)`
  con la dipendenza funzionale `GIOCATORE` → `SQUADRA`

  `CAMPIONATO` è in **3NF**: infatti in `GIOCATORE` → `SQUADRA`, `SQUADRA` è primo.

  Tuttavia la dipendenza funzionale `GIOCATORE` → `SQUADRA` genera problemi di ridondanza e anomalie simili a quelli visti negli esempi precedenti.

♢ La **3NF** definisce comunque schemi con un buon livello di qualità e costituisce quindi l'obiettivo normalmente adottato per il progetto logico di un database.

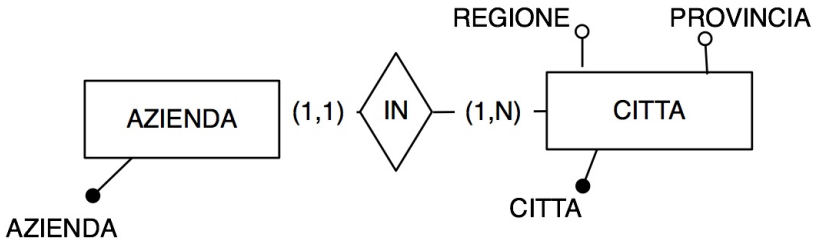
Progettazione Logica e Normalizzazione

♢ La traduzione di uno schema E/R in schema relazionale secondo le **Regole di Progettazione logica relazionale** produce uno schema relazionale in **3NF**.

♢ Intuitivamente: le **Regole di Progettazione logica relazionale** *non introducono* problemi di normalizzazione: dato uno schema E/R *normalizzato*, lo schema relazionale ottento è **3NF**.

♢ La precedente affermazione è solo a livello intuitivo, in quanto usa il concetto di schema E/R *normalizzato*, che non è stato formalmente definito.

♢ Esempio: uno schema E/R con l'entità `CITTA` che ha come attributi `PROVINCIA` e `REGIONE`, è non normalizzato in quanto `PROVINCIA` determina funzionalmente `REGIONE`:



♢ Per tradurre questo schema E/R in relazionale, applichiamo le **Regole di Progettazione logica relazionale**, ottenendo il seguente schema relazionale:

`CITTA(`<u>`Citta`</u>`,Provincia,Regione)`
`FD:` `Provincia` ⟶ `Regione`

`AZIENDA(`<u>`Azienda`</u>`,Citta)`
`FK :` `Citta` **`REFERENCES`** `CITTA`

La relazione `CITTA` non è in **3NF** a causa della **`FD:`** `Provincia` ⟶ `Regione`: questa **FD** non è stata introdotta durante la traduzione, ma era già presente nello schema E/R.

♢ Se invece traduciamo lo schema E/R in relazionale senza seguire le **Regole di Progettazione logica relazionale**, si possono introdurre altre **FD**:

`AZIENDA(`<u>`Azienda`</u>`,Citta,Provincia,Regione)`
`FD:` `Citta` ⟶ `Provincia`
`FD:` `Citta` ⟶ `Regione`
`FD:` `Provincia` ⟶ `Regione`

La relazione `AZIENDA` non è in **3NF** sia a causa della **`FD:`** `Provincia` ⟶ `Regione`, ma anche a causa di **`FD:`** `Citta` ⟶ `Provincia` e **`FD:`** `Citta` ⟶ `Regione`.

Progettazione logica da schemi E/R nel compito scritto

♢ Il principale esercizio del compito scritto consiste nella progettazione di un Data Warehouse, partendo da un *DBO* (DataBase Operazionale) di cui viene fornito lo schema E/R ed il corrispondente schema relazionale.

♢ **Esempio di compito scritto:** "Consideriamo un DBO con il seguente schema E/R ed il corrispondente schema relazionale (nello schema relazionale ci possono essere vincoli di integrità aggiuntivi):"

♢ **Nuova notazione:**
Negli schemi E/R considerati, la maggior parte degli identificatori sono semplici, cioè composti da un solo attributo: di conseguenza nello schema relazionale la maggior parte delle **FK** sarà su un attributo singolo.

Per semplicità introduciamo una notazione semplificata di **FK**, riportando direttamente la relazione riferita dopo il nome della **FK**.

Ad esempio, nella relazione `AZIENDA`:

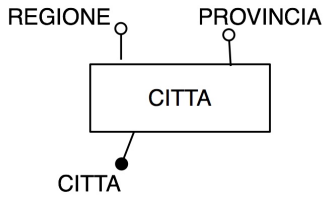
`AZIENDA(`<u>`AZIENDA`</u>`,CITTA)`
`FK` `: CITTA` **`REFERENCES`** `CITTA`

la **FK CITTA** riferita alla relazione `CITTA`, verrà scritta come:

`AZIENDA(`<u>`AZIENDA`</u>`,CITTA:CITTA)`

♢ **Semplificazione:**
Nello schema E/R del compito scritto, la maggior parte delle associazioni sono associazioni binarie uno-a-molti che servono per specificare delle dipendenze funzionali e quindi una gerarchia di una dimensione.

Per semplificare la struttura dello schema E/R, nei casi semplici queste dipendenze funzionali vengono aggiunte soltanto nello schema relazionale e non nello schema E/R, come ad esempio nell'entità `CITTA` dell'esempio precedente:



La dipendenza funzionale **`FD:`** `Provincia` $\longrightarrow$ `Regione`

viene riportata solo nello schema relazionale

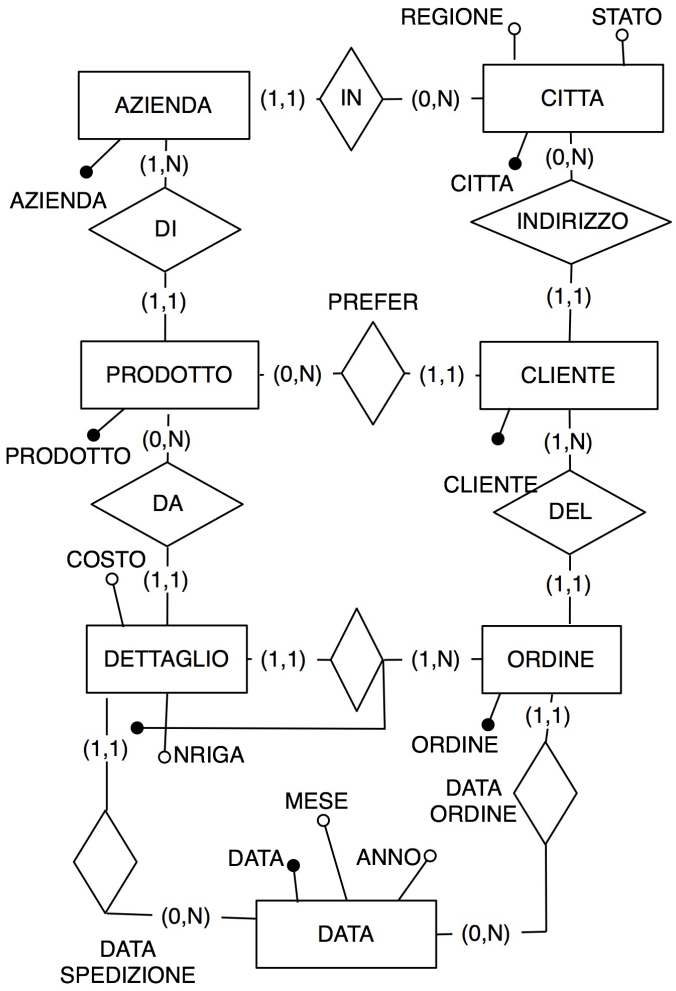
`CITTA(`<u>`Citta`</u>`,Provincia,Regione)`
`FD:` `Provincia` $\longrightarrow$ `Regione`

♢ Per riassumere: nel testo del compito scritto viene dato un *DBO* (DataBase Operazionale), specificandone sia il suo schema E/R che lo schema relazionale, con eventuali vincoli di integrità aggiuntivi quali *semplici* dipendenze funzionali.

Quindi nel compito scritto non viene richiesto di effettuare la progettazione logica relazionale da schemi E/R, ma viene già dato lo schema relazionale corrispondente.

Esempio di specifiche nel compito scritto

♢ "Consideriamo un DBO con il seguente schema E/R ed il corrispondente schema relazionale (nello schema relazionale ci possono essere vincoli di integrità aggiuntivi):"




```
CITTA(CITTA,REGIONE,STATO)
     F'D: REGIONE ⟶ STATO

DATA(DATA,MESE,ANNO)
     F'D: MESE ⟶ ANNO

AZIENDA(AZIENDA,CITTA:CITTA)

PRODOTTO(PRODOTTO,AZIENDA:AZIENDA)

CLIENTE(CLIENTE,CITTA:CITTA,PREFER:PRODOTTO)

ORDINE(ORDINE,CLIENTE:CLIENTE,DATA:DATA)

DETTAGLIO(NRIGA,ORDINE:ORDINE,
            PRODOTTO:PRODOTTO, DATASPEDIZIONE:DATA, COSTO)
```

Normalizzazione e Denormalizzazione

♢ **Normalizzazione**

Una relazione che non è nella forma normale desiderata viene *decomposto in sottoschemi* allo scopo di raggiungere tale forma normale, tramite un procedimento chiamato appunto *normalizzazione*.

♢ **Esempio :** Lo schema precedente non in **3NF**:

`PRODOTTO(Prodotto,Tipo,Categoria)`
FD: `Tipo` $\longrightarrow$ `Categoria`

viene intuitivamente normalizzato tramite decomposizione come segue: si considera la **FD:** `Tipo` $\longrightarrow$ `Categoria` che causa la violazione della **3NF**

1. gli attributi di tale **FD** si *riportano* in una nuova relazione
   `TIPO(Tipo,Categoria)`
2. nella relazione originale si toglie l'attributo determinato funzionalmente e si mette una **FK**:
   `PRODOTTO(Prodotto,Tipo)`
   FK : `Tipo` **REFERENCES** `Tipo`

♢ **DeNormalizzazione**

Una relazione non in forma normale è detto *denormalizzato*.

La *denormalizzazione* è il procedimento inverso della normalizzazione: denormalizzare lo schema

`TIPO(Tipo,Categoria)`

`PRODOTTO(Prodotto,Tipo)`
FK : `Tipo` **REFERENCES** `Tipo`

significa riportarlo nella forma:

`PRODOTTO(Prodotto,Tipo,Categoria)`
FD: `Tipo` $\longrightarrow$ `Categoria`

La denormalizzazione è una delle tecniche più utilizzate, soprattutto nel contesto dei datawarehouse, per aumentare le performance delle interrogazioni evitando di fare join tra relazioni. Ad esempio, nello schema denormalizzato, l'interrogazione "selezionare, per ogni prodotto il tipo e la categoria del " può essere eseguita considerando solo la relazione `PRODOTTO`

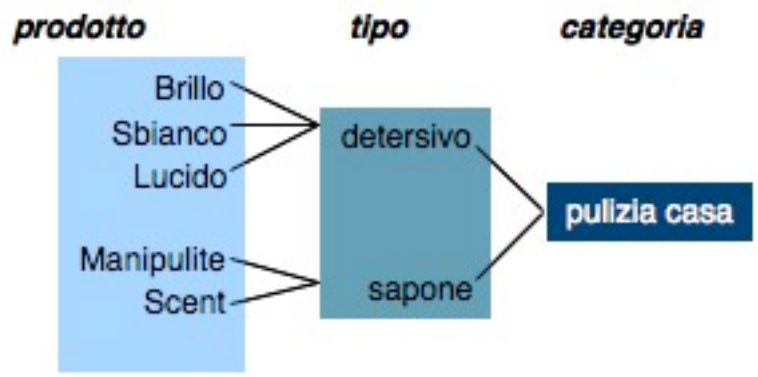
```
SELECT Prodotto,Tipo,Categoria
FROM PRODOTTO
```

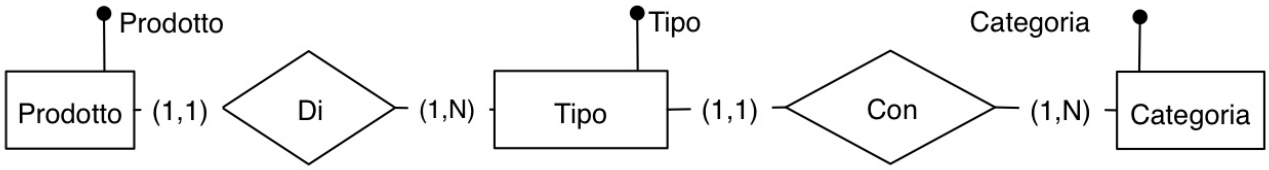
mentre nel caso dello schema normalizzato dobbiamo eseguire il join:

```
SELECT Prodotto,Tipo,Categoria
FROM TIPO, PRODOTTO
WHERE TIPO.Tipo=PRODOTTO.Tipo
```

Denormalizzazione nel progetto di un Data Warehouse

♢ Nella progettazione di un DW la maggior parte delle associazioni sono associazioni del tipo uno-a-molti che rappresentano gerarchie di attributi dimensionali





♢ La **progettazione logica di un Data Warehouse** viene generalmente effettuata secondo il cosiddetto **Star Schema**, che prevede relazioni completamente denormalizzate, in quanto una intera gerarchia viene riportata in un'unica relazione (chiamata *Dimension Table*)

```
Prodotto(Prodotto,Tipo,Categoria)
```

♢ Un'altra possibile tecnica di **progettazione logica di un Data Warehouse** è quella secondo lo **Snowflake Schema**, che prevede relazioni normalizzate, che si ottengono come normalizzazione delle corrispondenti relazioni dello **Star Schema**:

```
Prodotto(Prodotto,Tipo)
     FK : Tipo REFERENCES Tipo

Tipo(Tipo,Categoria)
```