

Basi di dati

Fabio Ferrario

Gennaio 2022

# Indice

<b>1</b>	<b>Schema Entity-Relationship</b>	<b>4</b>
1.1	Introduzione . . . . .	4
1.2	I costrutti di ER . . . . .	4
1.2.1	Entità . . . . .	5
1.2.2	Relazione . . . . .	6
1.2.3	Attributi . . . . .	6
1.2.4	Cardinalità delle relazioni . . . . .	6
1.2.5	Cardinalità di un attributo . . . . .	7
1.2.6	Identificatori . . . . .	7
1.2.7	Generalizzazione . . . . .	7
1.3	Qualità degli schemi ER . . . . .	8
<b>2</b>	<b>Modello Relazionale</b>	<b>9</b>
2.1	Introduzione . . . . .	9
2.2	I vincoli di integrità . . . . .	9
2.2.1	Vincoli intrarelazionali . . . . .	10
<b>3</b>	<b>Progettazione Logica</b>	<b>11</b>
3.1	Introduzione . . . . .	11
3.2	Ristrutturazione di ER . . . . .	11
3.2.1	Analisi delle ridondanze . . . . .	12
3.2.2	Eliminazione delle Generalizzazioni . . . . .	12
<b>4</b>	<b>Algebra Relazionale</b>	<b>14</b>
4.1	Introduzione . . . . .	14
4.2	Gli operatori di AR . . . . .	14
4.2.1	Operatori insiemistici . . . . .	15
4.2.2	$\rho$ Ridenominazione . . . . .	15
4.2.3	$\pi$ Proiezione . . . . .	16
4.2.4	$\sigma$ Selezione . . . . .	16
4.3	$\bowtie$ Join . . . . .	16

4.3.1	Join Naturale . . . . .	17
4.3.2	Theta Join . . . . .	17
4.3.3	Join Esterno . . . . .	17
4.3.4	Self Join . . . . .	18
4.4	Gestione dei valori NULL . . . . .	18
4.5	Le Viste . . . . .	19
4.6	Produrre interrogazioni in AR . . . . .	19
4.7	Esercizi . . . . .	20
<b>5</b>	<b>SQL</b>	<b>21</b>
5.1	Introduzione . . . . .	21
5.2	SQL-DDL . . . . .	22
5.2.1	L'istruzione CREATE TABLE . . . . .	22
5.3	Istruzione SELECT . . . . .	23
5.3.1	Le clausole e condizioni di Select . . . . .	23
5.4	Operatori Aggregati . . . . .	26
5.4.1	COUNT . . . . .	26
5.4.2	raggruppamenti GROUP BY . . . . .	27
5.5	Riassumendo SELECT . . . . .	28
5.6	Operatori Insiemistici in SQL . . . . .	28
5.7	JOIN . . . . .	29
5.8	Le subquery . . . . .	30
5.9	VISTE in SQL . . . . .	31
5.10	Esercizi . . . . .	32
<b>6</b>	<b>Esame</b>	<b>35</b>
6.1	Schema ER . . . . .	35
6.2	Modello Relazionale . . . . .	35
6.3	SQL . . . . .	35
6.4	AR . . . . .	36
6.5	PL . . . . .	36

# Capitolo 1

## Schema Entity-Relationship

### 1.1 Introduzione









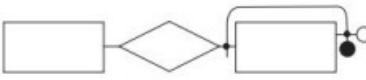


Lo schema ER, o Entity Relationship, è un modello intermedio per la produzione di un Database che serve a rappresentare in modo semplice e comprensibile la struttura di una base di dati.

Lo schema ER è un modello concettuale che rappresenta solo lo "schema", quindi la struttura, del database ma non può rappresentarne le istanze ed è indipendente da ogni DBMS.

Uno schema ER è formato da vari costrutti che rappresentano l'intera struttura della base di dati.

### 1.2 I costrutti di ER

Come detto in precedenza ER ha molti costrutti che vengono rappresentati in maniera varia, che però possiamo generalizzare come in figura:

Construct	Graphical representation
Entity	
Relationship	
Simple attribute	
Composite attribute	
Cardinality of a	
Cardinality of an attribute	
Internal identifier	 
External identifier	
Generalization	
Subset	

Di seguito elenchiamo i vari costrutti.

### 1.2.1 Entità

Un'entità è una classe di oggetti con proprietà comuni ed esistenza autonoma della quale si registrano fatti specifici.

**Nome** Ogni entità ha un nome univoco, espressivo e al **singolare**.

**Rappresentazione** Sono rappresentate da un rettangolo con all'interno il nome dell'entità.

**Caratteristiche** Ogni entità ha vari attributi, di (almeno) uno è necessariamente identificatore (chiave primaria).

### 1.2.2 Relazione

Una relazione descrive un'azione o una situazione e stabilisce legami logici tra istanze di entità; possono essere tra più di due entità e il numero di esse coinvolte determina il grado della relazione, che può essere binaria, ternaria o n-aria.

**Nome** Una relazione è identificata univocamente da un sostantivo.

**Rappresentazione** Vengono appresentate da un rombo con all'interno il nome della relazione, è collegato alle entità da dei connettori con indicata la cardinalità della relazione.

**Caratteristiche** Una relazione zero o più attributi ma NON ha identificatori.

### 1.2.3 Attributi

Gli attributi associano ad ogni istanza di entità o relazione un valore, definito su un dominio di valori, specificato nella documentazione associata, con il fine di descrivere le proprietà elementari di entità/relazioni disegnate per rappresentare la realtà di interesse

#### Attributo semplice

Un attributo è semplice se è formato da un solo elemento (attributo).

**Rappresentazione** Sono rappresentate da un'asticella con un pallino ad una estremità

#### Attributo composto

Raggruppamento di attributi di una medesima entità/relazione con affinità di significato e/o uso.

### 1.2.4 Cardinalità delle relazioni

Vengono specificate per ogni relazione e descrivono il numero minimo e massimo di occorrenza di relazione, a cui una occorrenza dell'entità può partecipare alla relazione, ossia quante volte un'occorrenza di un'entità può essere legata ad occorrenza delle altre entità coinvolte.

è possibile assegnare alla cardinalità un qualunque intero non negativo, con l'unico vincolo che la cardinalità minima sia minore o uguale alla cardinalità massima e di solito si usano i valori 0, 1 e N.

### 1.2.5 Cardinalità di un attributo

Descrive il numero minimo e massimo di valori dell'attributo associati all'entità e/o relazione, con la cardinalità (1,1) stabilita come default, che può essere vista come funzione che associa ad ogni occorrenza di entità un solo valore dell'attributo; si hanno le stesse consuetudini delle cardinalità delle relazioni.

### 1.2.6 Identificatori

Gli identificatori sono attributi che permettono di identificare in maniera univoca un'entità. Possono essere interni ed esterni e formati da un'attributo, un gruppo di attributi o un insieme di più attributi separati

#### Identificatore Interno

Un identificatore è interno in caso sia uno più attributi dell'entità stessa che identificano, tutti con cardinalità (1,1)

#### Identificatore Esterno

Un identificatore è esterno se è un attributo di un'entità diversa da quella che identifica, in cui esiste una relazione uno a uno tra le due entità.

### 1.2.7 Generalizzazione

Anche dette Is-a, una generalizzazione è un'entità che eredita tutti gli attributi e le relazioni di un'altra entità (detta padre), rappresentata da una freccia (ramata) che va dai figli verso il padre.

**Generalizzazione Totale** Una generalizzazione viene detta totale se ogni occorrenza del genitore è un'occorrenza di almeno uno dei figli, altrimenti è parziale

**Sottoinsieme** è una generalizzazione con soltanto un'entità figlia, di cui solitamente rappresenta una parte dell'entità genitore.

### 1.3 Qualità degli schemi ER

Come già detto, uno schema ER deve rappresentare completamente la struttura di una base di dati, e deve farlo in modo efficiente e comprensibile. Una determinata richiesta per una base di dati può essere soddisfatta da più schemi ER, alcuni però lo fanno meglio di altri.

Quindi possiamo definire alcuni indicatori della qualità degli schemi ER:

- Correttezza rispetto al modello
- Correttezza rispetto ai requisiti
- Minimalità (e ridondanza)
- Competenza e pertinenza
- Leggibilità grafica
- Compattezza



# Capitolo 2

## Modello Relazionale

### 2.1 Introduzione

Dove lo schema ER è un modello concettuale per rappresentare una base di dati, il Modello Relazionale è il **Modello Logico** necessario per rappresentare in maniera efficace ed efficiente la base dati modellata. Il modello relazionale è utilizzato in molti DBMS, e in esso ogni tabella è formata da Relation Schema e Relation Instance. Questo modello presenta una notevole differenza da ER, dato che i riferimenti fra dati in strutture diverse sono rappresentati per mezzo dei valori stessi, garantendo così l'indipendenza dei dati.

**Relation Schema e Relation Instance** Nel modello relazionale ogni relazione può essere rappresentata in due modi, lo schema e l'istanza.

Lo **schema di una relazione** è invariante nel tempo e descrive la struttura, quindi l'aspetto intensionale della relazione. Viene rappresentato con il nome della tabella e l'elenco degli attributi.

L'**istanza di una relazione** invece sono i valori effettivi della relazione che possono cambiare nel tempo (aspetto estensionale). Viene rappresentato come la tabella della relazione con le istanze.

### 2.2 I vincoli di integrità

Una relazione è una rappresentazione di un frammento di realtà osservata, quindi possono esistere istanze di basi di dati che, pur sintatticamente corrette, non rappresentano stati possibili nella realtà.

Introduciamo quindi il concetto di vincolo di integrità, una proprietà che

deve essere soddisfatta da tutte le istanze di uno schema che rappresentano informazioni corrette per l'applicazione

### 2.2.1 Vincoli intrarelazionali

**Vincoli di Tupla** I vincoli che vengono valutati su una sola Tupla di una relazione sono detti vincoli di tupla, e vengono espressi con sintassi booleana. Se questo vincolo viene valutato su più attributi di una sola tupla, quindi limita il valore di un attributo in base al valore di un altro attributo della stessa ennupla, allora è detto vincolo di **Tupla**.

Se un vincolo di tupla è valutato su un singolo attributo è detto di **Dominio**, che quindi limita il dominio di quel singolo attributo (per esempio voto  $\leq 30$ )

**Vincolo di Relazione** Vincolo relativo all'insieme di Ennuple di una relazione

**Vincolo di Chiave** Vincolo che permette di avere la certezza dell'unicità di uno schema di relazione. I vincoli di chiave nascono dalla necessità di individuare informazioni che ci permettano di rappresentare ogni oggetto di interesse con una ennupla differente e identificarlo quando se ne abbia necessità.

Una **Superchiave** è un insieme di attributi che identificano le ennuple di una relazione, quindi è un insieme di attributi per cui nella relazione non possono esistere due tuple che hanno gli stessi valori degli attributi contenuti nella superchiave.

Se una superchiave non contiene propriamente un'altra superchiave, viene detta **Chiave** oppure **Superchiave Minimale**

# Capitolo 3

## Progettazione Logica

### 3.1 Introduzione

L'obiettivo della progettazione logica è quello di costruire uno schema logico in grado di descrivere, in maniera corretta ed efficiente, tutte le informazioni contenute nello schema ER prodotto nella fase di progettazione concettuale. Nella Progettazione Logica si hanno due necessità:

- Semplificare la traduzione
- Ottimizzare il progetto

La semplificazione dello schema si rende necessaria perchè *non tutti i costrutti del modello ER hanno una traduzione naturale nei modelli logici*, vedisi la generalizzazione.

Inoltre, mentre la progettazione concettuale ha come obiettivo la rappresentazione accurata e naturale dei dati dal punto di vista del significato che hanno nell'applicazione, quindi in modo da renderli comprensibili nel mondo reale, **la progettazione logica costituisce la base per l'effettiva realizzazione dell'applicazione** e deve tenere conto, per quanto possibile, delle sue prestazioni: questa necessità può portare a una ristrutturazione dello schema concettuale che renda più *efficiente* l'esecuzione delle operazioni previste.

Pertanto è necessario prevedere sia a un'attività di **riorganizzazione** del modello concettuale, sia un'attività di **traduzione** da concettuale a logico.

### 3.2 Ristrutturazione di ER

Come già detto, la progettazione logica prevede una parte di riorganizzazione (ristrutturazione) dello schema ER.

Questa fase si può suddividere in una serie di passi da effettuare in sequenza:

1. **Analisi delle ridondanze**, dove si decide se eliminare o mantenere eventuali ridondanze presenti nello schema
2. **Eliminazione delle generalizzazioni**, dove tutte le generalizzazioni presenti nello schema vengono analizzate e sostituite da altri costrutti
3. **Partizionamento o accorpamento** di entità e associazioni, dove si decide se è opportuno partizionare concetti dello schema (entità e/o associazioni) in più concetti o, viceversa, accorpare concetti separati in un unico concetto
4. **Scelta degli identificatori principali**, dove si seleziona un identificatore per quelle entità che ne hanno più di uno

### 3.2.1 Analisi delle ridondanze

Un attributo ridondante è un attributo che può essere derivato da altri attributi. Ci sono varie forme di ridondanza ma riportiamo alcuni casi frequenti:

- Attributi derivabili, occorrenza per occorrenza, da altri attributi della stessa entità o associazione
- Attributi derivabili da attributi di altre entità (o associazioni), di solito attraverso funzioni aggregative.

In generale la presenza di un dato derivato presenta un vantaggio e alcuni svantaggi, quindi la decisione di mantenere o eliminare una ridondanza va presa confrontando costo di esecuzione delle operazioni che coinvolgono il dato ridondante e la relativa occupazione di memoria.

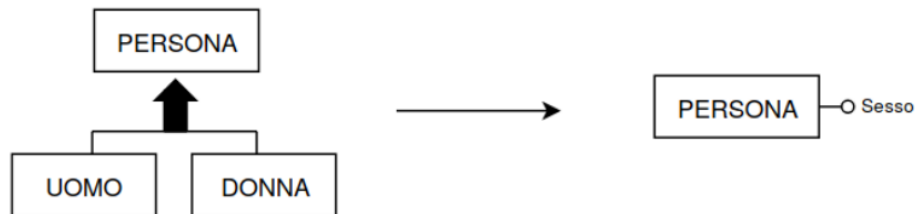
### 3.2.2 Eliminazione delle Generalizzazioni

Le generalizzazioni non sono rappresentabili nel Modello Relazionale, quindi è compito della Progettazione Logica di eliminarle mantenendo la correttezza dello schema.

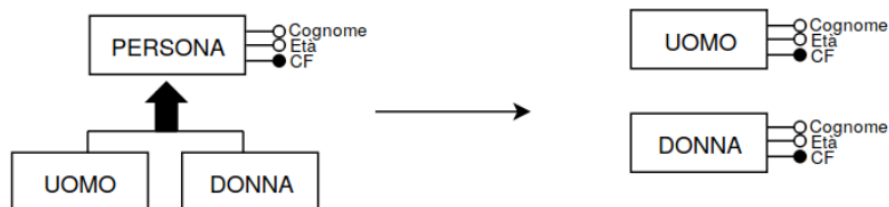
Per fare questo ci sono tre modi:

**Accorpamento dei figli nel genitore**

Viene usato quando l'accesso alle entità è contestuale

**Accorpamento del genitore nei figli**

Possibile solo se la generalizzazione è totale

**Sostituzione di generalizzazione con associazioni**

Si usa quando l'accesso alle due entità è separato. Conveniente con generalizzazione parziale



# Capitolo 4

## Algebra Relazionale

### 4.1 Introduzione

L'algebra relazionale è un linguaggio per basi di dati di tipo Procedurale. AR è un linguaggio prettamente formale che forma la base per linguaggi 'reali'. È di tipo procedurale, quindi viene specificato l'algoritmo con cui ottenere il risultato. In AR istruzioni equivalenti possono differire in termini di efficienza.

Le relazioni vengono intese in senso matematico, quindi insieme di tuple definite su attributi. In un insieme non ci possono essere elementi uguali.

### 4.2 Gli operatori di AR

L'algebra relazionale ha 7 operatori che danno l'intero potere espressivo del linguaggio e grazie a una combinazione di questi è possibile interrogare la base di dati

- Operatori insiemistici
  - $\cup$  Unione
  - $\cap$  Intersezione
  - $-$  Differenza
- $\rho$  Ridenominazione
- $\pi$  Proiezione
- $\sigma$  Selezione
- $\bowtie$  Join

il Join lo considereremo come un operatore "a parte"

### 4.2.1 Operatori insiemistici

Una relazione (in matematica) è un sottoinsieme del prodotto cartesiano di due o più insiemi. Una relazione è un insieme, quindi: non c'è ordinamento tra le tuple, ogni tupla è distinta e al suo interno è ordinata.

Visto che le relazioni sono insiemi, è possibile applicare operatori tra insiemi che producono insiemi.

Nota bene: Le operazioni di Unione, Intersezione e differenza sono applicabili solo a relazioni definite sugli stessi attributi. Visto che i risultati devono essere relazioni, le n-ple identiche nel risultato vengono unite in una unica n-ple.

#### Unione

$$r_1 \cup r_2$$

Produce tutte le n-ple delle relazioni  $r_1$  e  $r_2$

L'unione è commutativa e associativa

#### Intersezione

$$r_1 \cap r_2$$

Produce tutte le n-ple che appartengono sia a  $r_1$  che  $r_2$

L'intersezione è commutativa e associativa

#### Differenza

$$r_1 - r_2$$

Produce tutte le n-ple di  $r_1$  che non compaiono anche in  $r_2$

La differenza non è ne commutativa, ne associativa

### 4.2.2 $\rho$ Ridenominazione

$$\rho_{y \leftarrow x}(r)$$

Cambia il nome dell'attributo  $x$  della relazione  $r$  in  $y$

ATTENZIONE: la ridenominazione modifica lo schema lasciando inalterata l'istanza di  $r$

è possibile rinominare più attributi contemporaneamente usando  $\rho_{y,z \leftarrow x,w}(r)$

**REMEMBER** la ridenominazione è in ordine  $nuovoNome \leftarrow vecchioNome$

### 4.2.3 $\sqcap$ Proiezione

$$\sqcap_{\text{listaAttributi}}(r)$$

La proiezione "decompone verticalmente" la tabella, riportando solo gli attributi contenuti in listaAttributi di tutte le n-ple.

Gli attributi in listaAttributi sono separati da una virgola

### 4.2.4 $\sigma$ Selezione

$$\sigma_{\text{condizione}}(r)$$

La selezione è un operatore ortogonale alla proiezione, quindi "decompone orizzontalmente" una tabella. Produce una relazione che ha lo stesso schema dell'operando e contiene un sottoinsieme delle ennuple dell'operando, quelle che soddisfano la condizione espressa dall'operatore

**Condizione** La condizione è una espressione booleana ottenuta combinando con i connettivi OR, AND e NOT condizioni atomiche. Le condizioni atomiche sono del tipo confronto tra attributi o tra un attributo e una costante

## 4.3 $\bowtie$ Join

Il join è un operatore fondamentale dell'AR che ci permette di combinare tuple appartenenti a relazioni diverse. Il Join produce il sottoinsieme del prodotto cartesiano di due relazioni in cui il valore di determinati attributi coincide

### Varianti del join

- Join Naturale
- Prodotto cartesiano
- Theta Join
- Join esterno (Destro e Sinistro)



### 4.3.1 Join Naturale

$$r_1(X_1) \bowtie r_2(X_2)$$

Il Join Naturale combina le tuple delle due relazioni sulla base dell'uguaglianza dei valori degli attributi comuni, quindi se faccio il join di due relazioni cercherà una corrispondenza tra gli attributi con lo stesso nome e produrrà una tupla corrispondente.

Date due relazioni  $r_1$  e  $r_2$ , ogni tupla che compare nel risultato del join naturale è ottenuta come combinazione di una tupla di  $r_1$  con una di  $r_2$  sulla base dell'uguaglianza dei valori degli attributi comuni alle due relazioni.

Lo schema del risultato è l'unione degli schemi degli operandi  $X_1 \cup X_2$

Quando una tupla non partecipa al risultato (per esempio un valore appare in una tabella ma non nel risultato) viene detto Dangling e il Join diventa incompleto

Se due relazioni sono definite sugli stessi attributi il join naturale equivale all'intersezione delle due relazioni

**Attenzione** se facciamo il join naturale di due relazioni che non hanno attributi comuni (con lo stesso nome) il risultato sarà il prodotto cartesiano delle due!

### 4.3.2 Theta Join

$$r_1(X_1) \bowtie_{\text{condizione}} r_2(X_2)$$

Succede spesso che due relazioni non hanno attributi comuni, quindi il join genererebbe il prodotto cartesiano che ha senso solo se è seguito da una selezione.

Il theta join effettua un Join con una immediata selezione, di modo da dar senso al prodotto cartesiano.

**Attenzione** Il theta join si applica SOLO quando  $X_1$  e  $X_2$  sono insiemi disgiunti, altrimenti un join normale sarebbe sufficiente.

### 4.3.3 Join Esterno

$$r_1 \text{ JOIN}_{\text{LEFT/RIGHT/FULL}} r_2$$

Il join esterno è un tipo di join naturale che mantiene tutte le tuple di uno o entrambi gli operandi, estendendo le tuple che rimarrebbero dangling con dei valori nulli, eliminando quindi il tuple dangling. Quando facciamo un join

naturale stiamo facendo un join interno, quindi se esistono delle tuple che non appartengono al risultato rimangono dangling.

In caso volessimo mantenere tutte le tuple di una o entrambe le operazioni, usiamo il join esterno.

#### Tipi di join esterno

- LEFT, che mantiene tutte le tuple del primo operando
- RIGHT, che mantiene tutte le tuple del secondo operando
- FULL, che mantiene tutte le tuple di entrambi gli operandi

#### 4.3.4 Self Join

In alcuni casi bisogna fare un join di una relazione con (una copia di) se stessa, chiamiamo questo tipo di Join "self join".

Generalmente viene usato per:

- Confrontare tuple di una relazione tra loro
- trovare elementi duplicati
- Quando una tuple fa riferimento a dati della stessa tabella

Un modo per fare un self join è (per esempio) un join tra una relazione (con opportuno select etc) e una proiezione di un attributo di se stessa

### 4.4 Gestione dei valori NULL

Nelle basi di dati possiamo incontrare dei valori nulli, quindi AR utilizza una logica a tre valori: True, False e NULL.

In generale l'introduzione di un NULL nella logica classica non varia il risultato, tranne in due casi:

- NULL AND True  $\rightarrow$  NULL
- NULL OR False  $\rightarrow$  NULL

#### condizione IS NULL

Quando si necessita una condizione si possono usare IS NULL e IS NOT NULL, che diventano true rispettivamente se il dato è Nullo e se non è Nullo

## 4.5 Le Viste

Per semplificare alcune espressioni è utile avere delle relazioni derivate a partire dalle relazioni definite nello schema di base di dati.

é quindi possibile in AR definire delle viste, che altro non sono che delle espressioni a cui viene assegnato un nome che possono essere usate all'interno di altre espressioni

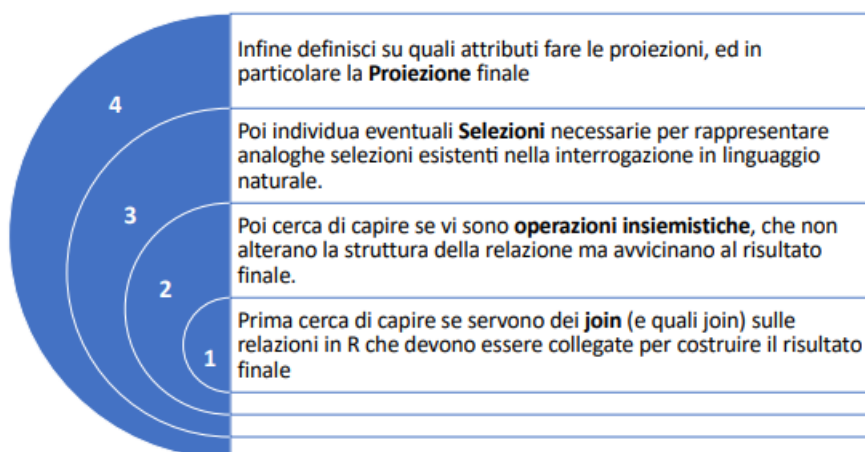
`nome_vista = ESPRESSIONE AR`

**Attenzione** Le viste sono relazioni virtuali, quindi non vengono mai salvate nella base di dati, quindi una interrogazione su una vista viene eseguita "ricalcolando" la vista

## 4.6 Produrre interrogazioni in AR

Esiste un metodo "generale" per produrre interrogazioni in algebra relazionale:

1. Dalla "domanda" trova l'insieme di relazioni su cui applicare gli operatori
2. Procedi secondo la strategia "dall'interno all'esterno"



**In parole povere** Per produrre con efficienza delle interrogazioni i AR bisogna procedere a strati, partendo da quello più interno (il Join) e finendo con la proiezione degli attributi

## 4.7 Esercizi

Dato il seguente schema relazionale:

Personale\_non\_docente ( Matricola\_d , Cognome , Nome , Ruolo , Classe\_stipendio )  
 Personale\_docente ( Matricola\_d , Cognome , Nome , Ruolo , Classe\_stipendio )  
 Stipendio ( Classe , Valore )

Studente ( matricola\_st , Cognome , Nome , corso\_di\_Laurea )

**es16** Formulare l'espressione in AR che produca le classi di stipendio che non sono attribuite a nessun personale docente - senza usare l'operatore differenza

```

PROJclasse,stipendio (
  SELmatricola_d ISNULL (
    personale_docente  $\bowtie_{RIGHT JOIN}$  RIDclasse_stipendio  $\leftarrow$  classe ( stipendio )
  )
)

```

**Note su es16** la rinominazione di "classe\_stipendio" in "classe" può essere evitata facendo un THETA RIGHT JOIN, quindi con  $\bowtie_{RIGHT classe=classestipendio}$

**es17** Formulare una espressione in AR che produca il personale non docente che è anche iscritto ad un corso di laurea senza usare l'operatore intersezione o differenza

```

PROJmatricola_d,Cognome_nome (
  PROJmatricola_st,cognome,nome ( personale\_non\_docente )
  JOINmatricola_d=matricola_st
  PROJmatricola_st,cognome,nome ( studente )
)

```

**Note su es17** La prima proiezione serve a eliminare il doppione "matricola\_s" che verrebbe generato dal join

Le due piccole proiezioni invece servono a eliminare gli attributi non comuni perchè il join in AR va sempre fatto tra relazioni con gli stessi attributi

# Capitolo 5

## SQL

### 5.1 Introduzione

SQL è il linguaggio per la definizione e la manipolazione dei dati in database relazionali adottato da tutti i principali DBMS.

Nonostante sia uno standard solo parte dei DBMS lo utilizza, ed esistono tre livelli di conformità che differiscono a seconda delle features implementate:

- Entry
- Intermediate
- Full

La maggior parte dei database è conforme all'Entry, ma Intermediate è la versione che più soddisfa le esigenze di mercato

### SQL e AR

SQL è "relazionalmente completo", quindi ogni espressione di AR può essere tradotta in SQL, però quest'ultimo può fare molto di più.

Come AR SQL adotta una logica a tre valori (T,F,?), e la gestione dei valori NULL è la medesima. Inoltre SQL è un linguaggio computazionalmente completo. A differenza di AR in SQL le relazioni sono intese come tabelle e possono esserci righe uguali

### Funzionalità

SQL estende molto AR, in particolare contiene:

- DDL(Data definition language)  
Operazioni di definizione e modifica dello schema
- DML(Data Manipulation Language)  
Operazioni di interrogazione  
Operazioni di Aggiornamento

## 5.2 SQL-DDL

SQL-DDL, o data definition language è la parte di SQL che ti permette di effettuare operazioni di definizione e modifica degli schemi di una base di dati. Con SQL-DDL si possono descrivere tre aspetti di uno schema relazionale: Schema, Table e Domain.

### 5.2.1 L'istruzione CREATE TABLE

```
CREATE TABLE TableName  
(AttributeName Domain [DefaultValue] [Constraints]  
, ... ,  
[OtherConstraints])
```

L'istruzione CREATE TABLE definisce uno schema di relazione e ne crea un'istanza vuota, cioè senza tuple, di cui specifica attributi, domini e vincoli

**Domini o data types** esistono due tipi di domini: elementari e definiti dall'utente.

Tra i domini elementari troviamo: Testuali, Numerici, Data e ora, Booleani, BLOB e CLOB.

I domini definiti dall'utente invece sono creati dall'istruzione CREATE DOMAIN che definisce un dominio elementare utilizzabile in definizioni di relazioni, anche con vincoli e valori di default

#### Vincoli

SQL-DDL può anche definire i vincoli di integrità intra-relazionali e inter-relazionali.

**Vincoli Intrarelazionali** Definiti tramite: NOT NULL, UNIQUE, PRIMARY KEY, CHECK

**Vincoli Interrelazionali** FOREIGN KEY e REFERENCES, che sono vincoli di integrità referenziale

## 5.3 Istruzione SELECT

In SQL ogni query di interrogazione inizia con l'istruzione SELECT che select non ha niente (circa) a che fare con la selezione di AR. Il risultato di una SELECT è sempre una tabella e una tabella è sempre ottenuta tramite una Select.

Nella sua forma più base

```
SELECT listaAttributi <— target list
FROM tabella
[WHERE condizione]
```

Seleziona tra le n-pie del prodotto cartesiano delle tabelle citate nella FROM, quelle che soddisfano la condizione presente nella WHERE, e di esse ne rappresenta solo gli attributi nella ListaAttributi.

### 5.3.1 Le clausole e condizioni di Select

L'istruzione select ha moltissime possibili clausole o condizioni che modificano o estendo il risultato di una query.

#### SELECT DISTINCT

In generale, in una query SELECT i duplicati vengono mantenuti. *Se Si vogliono eliminare i duplicati* si può usare l'istruzione SELECT DISTINCT.

```
SELECT DISTINCT attributo1, attributo2, ...
```

La clausola distinct elimina i duplicati *considerando tutta la ennupla e non solo un singolo attributo*.

**Attenzione** DISTINCT da solo non esiste, se si usa va sempre indicato come SELECT DISTINCT

#### AS

La clausola AS serve a rinominare gli attributi in modo da semplificarne la comprensione o velocizzare la scrittura

```
SELECT attributo1 AS nuovonome, attributo2 AS ...
```

AS può anche essere utilizzato per rinominare le tabelle, però può anche essere omesso. Quindi:

```
SELECT ...  
FROM tabella AS T
```

equivale a dire

```
SELECT ...  
FROM tabella T
```

### Clausola . (punto)

Quando nella SELECT si considerano più tabelle (con un JOIN o un prodotto cartesiano) si potrebbe aver bisogno di specificare a quale tabella appartiene l'attributo che si sta considerando per una condizione (quindi in WHERE o HAVING). Per fare ciò si può usare la clausola . (punto), sia in caso la tabella è stata rinominata sia in caso contrario.

```
SELECT ...  
FROM tabella T  
WHERE T.attributo1 ...
```

### Aritmetica nella target list

La SELECT ci permette di utilizzare degli operatori aritmetici nella target list.

gli operatori aritmetici consentiti sono i classici +, -, \*, / e la priorità è quella standard e alterabile con delle parentesi.

```
SELECT attr1 + attr2  
FROM ...
```

Questa query produrrà una tabella che avrà come attributo "attr1 + attr2" e conterrà come enuple l'elenco delle somme dei due per ogni enupla della tabella

### LIKE

LIKE è un tipo di condizione che ti permette di selezionare stringhe in base alla loro struttura.

Una condizione like è così formata:



- **A**: Una lettera significa che nella stringa ci deve essere quella determinata lettera in quella posizione (che può essere come prima, ultima o tra dei caratteri arbitrari)
- **%**: il simbolo percentuale significa "un numero arbitrario di caratteri"
- **\_**: il trattino basso significa "un solo carattere arbitrario"

quindi la query:

```
SELECT ...  
FROM ...  
WHERE nome LIKE F_b%
```

significa "dove nome inizia per 'F', seguito da un solo carattere arbitrario, seguito da 'b' e che finisce in un numero arbitrario di caratteri"

## BETWEEN

L'operatore BETWEEN nella clausola WHERE seleziona dei valori all'interno di un range

```
SELECT ...  
FROM ...  
WHERE att1 BETWEEN 1 AND 5 ...
```

## ORDER BY

L'operatore ORDER BY, inserito in fondo alla query, specifica l'ordinamento delle tuple del risultato secondo l'attributo (o attributi) selezionato

L'ordinamento può essere:

- ascendente utilizzando *asc*
- discendente utilizzando *desc*

ascendente o discendente va dichiarato dopo l'attributo

```
ORDER BY attr1 asc, attributo2 desc, ...
```

Se vengono indicati più attributi, l'ordinamento avverrà dando priorità al primo attributo.

## 5.4 Operatori Aggregati

Le espressioni aritmetiche (come le condizioni) sono sempre valutate **singolarmente su una riga alla volta**, quindi se devo fare delle operazioni le posso fare solo riga per riga e non posso valutarli su insiemi di tuple.

Quindi SQL mette a disposizione degli operatori che sono valutati su insiemi di tuple:

- COUNT, conteggio
- MIN, minimo
- MAX, massimo
- AVG, media
- SUM, somma

Questi eseguono operazioni considerando tutte le tuple della tabella (selezionate opportunamente nella query).

Gli operatori Aggregati vengono valutati **solo dopo aver eseguito la interrogazione di base**, quindi sull'insieme delle n-ple prodotte dall'interrogazione di base.

### 5.4.1 COUNT

COUNT restituisce il numero di righe o il numero di valori distinti di un particolare attributo e si può applicare a qualsiasi tipo di attributo o lista di attributi

```
SELECT COUNT *  
FROM ...
```

oppure

```
SELECT COUNT [DISTINCT] ListaAttributi  
FROM ...
```

**esempio di COUNT \*** la query:

```
SELECT count(*) AS numero_ordinari  
FROM Personale_docente  
WHERE ruolo = 'Ordinario'
```

produrrà una tabella che contiene semplicemente 'numero\_ordinari' con il numero dei prof. ordinari. Questa query viene realizzata prima eseguendo la query di base (quindi la selezione di tutti gli attributi da personale docente dove il ruolo è ordinario) e poi conta le righe dei risultati ottenuti

**esempio di COUNT attributo** la query

```
SELECT count(distinct Classe_stipendio)
FROM Personale_docente
WHERE ruolo = 'Ordinario'
```

andrà a contare il numero delle classi di stipendio per i professori ordinari,  
**non contando NULL**

### Ricapitolando COUNT

- numero di tuple: `SELECT count(*) FROM ...`
- numero di volte in cui "attributo" non è NULL: `SELECT count(attributo) FROM ...`
- numero di valori distinti di "attributo" (senza i NULL): `SELECT count(distinct attributo) FROM ...`

## 5.4.2 raggruppamenti GROUP BY

Gli operatori aggregati si possono applicare sia all'intero risultato dell'operazione che su partizioni delle relazioni, ovvero gruppi di tuple, usando GROUP BY

```
SELECT operatoreAggregato ...
FROM ...
WHERE ...
GROUP BY listaAttributi
```

Group by va inserito dopo la clausola WHERE e **ha senso SOLO nell'ambito degli operatori aggregati**

**Funzionamento** GROUP BY funziona così: prima esegue l'interrogazione base senza GROUP BY e gli operatori aggregati, raggruppa le righe secondo il GROUP BY e infine applica l'operatore aggregato a ciascun gruppo di righe

**Attenzione** con il group by il NULL è un gruppo a se (quindi viene contato)

**Regola** In una interrogazione che fa uso della GROUP BY può comparire come argomento della SELECT, e quindi come insieme di attributi nella target list, solamente un sottoinsieme degli attributi che compaiono nella clausola GROUP BY

## HAVING

HAVING permette di esprimere condizioni sui gruppi, applicate a ogni insieme di n-ple risultato della applicazione del GROUP BY. Va inserito dopo group by (HAVING condizione)

## 5.5 Riassumendo SELECT

```
SELECT ListaAttributoOEspressioni
FROM ListaTabelle
[ WHERE CondizioneSelezioneTuple ]
[ GROUP BY ListaAttributiDiRaggruppamento ]
[ HAVING CondizioniSelezioneGruppi ]
[ ORDER BY ListaAttributiDiOrdinamento ]
```

## 5.6 Operatori Insiemistici in SQL

SQL prevede i normali operatori insiemistici, con semantica uguale a AR ma regole diverse.

**attenzione** Con gli operatori insiemistici in SQL i duplicati sono esclusi di default!

- UNION
- INTERSECT
- MINUS

**Regole** in SQL gli operatori insiemistici si applicano solo a relazioni definite sullo stesso numero di attributi (non necessariamente devono essere uguali!) e l'ordine non è importante. La tabella risultante avrà gli attributi della prima tabella nella query

## 5.7 JOIN

Si può indicare il costrutto JOIN nel FROM in modo da rendere il tutto più leggibile e poter specificare il tipo di join

```
SELECT ...  
FROM tabella1 JOIN tabella2 ON att1=att2
```

è possibile specificare il tipo di join:

- NATURAL JOIN
- INNER JOIN
- OUTER JOIN

LEFT/RIGHT/FULL OUTER JOIN

**Self Join** Anche in SQL il self join è possibile e viene usato per confrontare le righe di una tabella, trovare elementi duplicati o quando una riga fa riferimento a dati nella stessa tabella  
in SQL si possono ridenominare sia attributi che tabelle, quest'ultima è necessaria per il self join.

### Prodotto cartesiano

Nonostante il prodotto cartesiano sia un tipo di JOIN, la sua sintassi è molto diversa dagli altri.

In SQL un prodotto cartesiano si fa semplicemente indicando più tabelle nella clausola FROM separate da una virgola

```
SELECT ...  
FROM tabella1, tabella2
```

Il prodotto cartesiano ha raramente senso, quindi generalmente è seguito da una selezione nella clausola where che lo trasforma in un THETA JOIN

**NB** in AR il theta join ha senso solo per relazioni che non hanno attributi in comune, mentre in sql ha sempre senso

## 5.8 Le subquery

Ci sono alcuni casi in cui una query "semplice" non è sufficiente per rispondere a una richiesta, SQL quindi dà la possibilità di creare delle Subquery che ci permettono di confrontare valori elementari con il risultato di istruzioni SELECT. Una subquery è una SELECT nidificata all'interno di una clausola WHERE o HAVING

```
SELECT ...
FROM ...
WHERE att1 *op* (SELECT ...)
```

\*op\* può essere un qualunque operatore di confronto, sia semplice che quantitativo.

**Visibilità delle variabili** La creazione di subquery crea un dubbio: come funzionano le variabili tra query e relativa subquery?

SQL segue una semplice regola: una variabile è visibile nella query che l'ha definita o in una subquery di essa. Non è quindi possibile fare riferimenti da blocchi esterni a variabili definite in blocchi più interni o da altri blocchi di pari livello.

**REGOLE di Subquery** Il numero e il dominio dell'attributo della query esterna deve necessariamente essere compatibile con numero e dominio dell'attributo della query interna

EG non devo confrontare "matricola" con "data" ma neanche "matricola" con "matricola e data"

**Semantica bottom up** Quando bisogna eseguire una query con una subquery, SQL esegue prima la subquery e poi la query esterna.

**Gli operatori di confronto** Gli operatori di confronto semplici (=, <, >, <=, >=, ...) si possono usare solo per subquery che restituiscono un singolo valore che si può usare come espressione scalare.

Per usare predicati di confronto con subquery che possono restituire più di una riga, occorre usare le quantificazioni ALL, ANY o IN/NOT IN.

Il predicato EXISTS/NOT EXISTS è invece un quantificatore esistenziale che permette di verificare se la query restituisce o meno una tupla

## ALL e ANY

ALL e ANY sono due operatori di confronto (che estendono quelli normali) utilizzati per confrontare un valore atomico con il risultato di una subquery che restituisce più di una tupla.

- ALL specifica che la condizione è vera se il confronto è vero rispetto ad **almeno uno** degli elementi di una lista.

```
WHERE attributo = ANY(SELECT ...)
```

- ANY invece specifica che la condizione è vera se il confronto è vero rispetto a **tutti** gli elementi di una lista

```
WHERE attributo <> ALL(SELECT ...)
```

## IN e NOT IN

IN e NOT IN invece sono usati per selezionare righe che hanno , o non hanno, un attributo che assume valori contenuti in una lista.

- IN ha valore vero se il valore **compare** nell'insieme risultato della interrogazione

```
WHERE attributo IN(SELECT ...)
```

- NOT IN invece è il contrario, quindi è vero se il valore non compare nell'insieme risultato.

```
WHERE attributo NOT IN(SELECT ...)
```

**NB** IN e NOT IN possono confrontare più attributi contemporaneamente usando le parentesi.

**ESERCIZIO RIEPILOGATIVO** esercizio riepilogativo a pagina 35 della presentazione SQL4

## 5.9 VISTE in SQL

In SQL è possibile creare viste usando l'istruzione CREATE VIEW. Le viste sono delle tabelle virtuali.

## 5.10 Esercizi

### Studente-Esame

Data la base di dati:

```
Studente(matricola_st, Cognome, Nome, corso_di_Laurea)
Esame (Data, matricola_studente,Voto)
```

**ES 25** Trova gli studenti che hanno preso più di 25 in tutti gli esami

```
SELECT DISTINCT matricola_st, cognome, nome
FROM studente S JOIN Esame E
ON E.matricola_st = E.matricola_studente
WHERE matricola_st NOT IN
      (SELECT matricola_studente
       FROM Esame
       WHERE voto < 25)
```

Abbiamo bisogno degli studenti che non anno mai preso meno di 25, di conseguenza se riusciamo a trovare con una subquery le matricole di tutti gli studenti che hanno preso meno di 25 e la escludiamo (con NOT IN) dalla tabella degli studenti troviamo il risultato. il Join serve a escludere quegli studenti che non hanno mai fatto un esame

**Variante** Risolviamo utilizzando NOT EXISTS

```
SELECT DISTINCT Matricola_St, Nome, Cognome
FROM Studente S JOIN Esame E
ON E.Matricola_studente = S.Matricola_St
WHERE NOT EXISTS (
      SELECT *
      FROM ESAME E2
      WHERE S.Matricola_st = E2.Matricola_studente and Voto < 25)
```

La soluzione è pressochè identica ma viene utilizzato NOT EXISTS per eliminare tutti gli esami in cui il voto è minore di 25.

L'operatore esistenziale EXISTS/NOT EXISTS ha senso solo nel caso in cui la query interna faccia riferimento alla query esterna (quindi se è correlata). Nota bene che *si valuta la query interna per ogni tupla della query esterna*



**Variante** Stessa domanda ma nel caso in cui l'attributo voto possa assumere valore NULL

```
SELECT DISTINCT matricola_st, cognome, nome
FROM studente S JOIN esame E
ON E.Matricola_studente = S.Matricola_St
WHERE matricola_st NOT in
      (SELECT matricola_studente
       FROM Esame
       WHERE voto < 25 or voto IS NULL)
```

## Disco-Cantante

Data la base di dati

```
DISCO(Nroserie, TitoloAlbum, Anno, Prezzo)
CONTIENE(NroSerieDisco, CodiceReg, NroProgr)
ESECUZIONE(CodiceReg, TitoloCanz, Anno)
AUTORE(Nome, TitoloCanzone)
CANTANTE(NomeCantante, CodiceReg)
```

**1** Trovare i cantautori (persone che hanno scritto e cantato la stessa canzone) il cui nome inizia per D.

```
Select distinct A.nome
FROM
      cantante C JOIN esecuzione E
      ON C.CodiceReg = E.CodiceReg
      JOIN autore A ON A.TitoloCanzone = E.TitoloCanz
WHERE NomeCantante LIKE 'D%'
```

Facendo il JOIN di Autore, Esecuzione e Cantante troviamo l'elenco di tutte le canzoni con relativi autori e cantanti. Non capisco come mai non fa un check per vedere se autore e cantante sono la stessa persona. Il distinct serve a eliminare ridondanze nel risultato

**2** I titoli dei dischi che contengono canzoni di cui non si conosce l'anno di registrazione

```
SELECT distinct D.TitoloAlbum
FROM disco D JOIN contiene C on NroSerieDisco = Nroserie
JOIN esecuzione E on E.CodiceReg = C.CodiceReg
WHERE E.Anno IS NULL
```

- 3** I pezzi del disco di serie 2, ordinati per numero progressivo, con indicazione dei cantanti

```
SELECT distinct C.NroProgr, E.TitoloCanz, Ca.NomeCantante
FROM contiene C
JOIN esecuzione E on E.CodiceReg = C.CodiceReg
JOIN cantante Ca on Ca.CodiceReg = C.CodiceReg
WHERE D.Nroserie = 2
ORDER BY C.NroProgr
```

Non bisogna raggruppare perchè ogni canzone ha un cantante (credo)

- 4** Gli autori e i cantanti puri, ovvero autori che non hanno mai registrato una canzone e cantanti che non hanno mai scritto una canzone  
Provo a cercare tutti i cantanti puri e unisco con gli autori puri

```
SELECT NomeCantante
FROM cantante
WHERE NomeCantante NOT IN (
    SELECT Nome
    FROM autore
)

UNION

SELECT Nome
FROM autore
WHERE NomeCantante NOT IN (
    SELECT Nome
    FROM Cantante
)
```

Questa soluzione me la sono inventata io, il ragionamento è: Se il nome di un cantante è anche nella tabella autore, allora vuol dire che ha anche scritto una canzone e viceversa

- 5** I cantanti del disco che contiene il maggior numero di canzoni

# Capitolo 6

## Esame

L'esame è strutturato in 5 esercizi, ognuno relativo a un macro argomento dell'insegnamento. Ad ogni esercizio viene dato un voto da 0 a 30, il voto finale dell'esame è la media dei 5 esercizi.

### 6.1 Schema ER

Nell'esame c'è un esercizio sullo schema ER, viene fornito un testo che definisce le specifiche per un DB, bisogna progettare lo schema ER partendo dal testo specificando le cardinalità minime e massime ed eventuali identificatori esterni

### 6.2 Modello Relazionale

Viene dato uno schema relazionale (e un testo di specifica). Vengono poi chieste alcune domande, come: -Definire tutte le chiavi primarie e i vincoli di integrità referenziale -Vincoli intrarelazionali di tupla -Una chiave alternativa per una determinata relazione -Una superchiave non minimale -Altri eventuali vincoli di integrità

### 6.3 SQL

Viene data una base di dati (rappresentata da uno schema relazionale) e si chiede di formulare 3 (o 2) query in linguaggio SQL

## 6.4 AR

Viene data una base di dati (rappresentata da uno schema relazionale) e si chiede di formulare 3 query in algebra relazionale

## 6.5 PL

Viene dato uno schema ER, viene chiesto di tradurlo prima in uno schema ER semplificato, poi nel modello relazionale indicando chiavi e vincoli di integrità relazionale