Appunti di Basi di Dati 1

Andrea Donati

A.A. 2017/2018 - Prof. Stefano Ceri

Contents

1 Introduzione

I dati sono il cuore dei sistemi informatici, nessuna impresa può operare senza aver sviluppato e formalizzato la modalità di accesso e gestione dei propri dati. In quanto "patrimonio aziendale", essi devono essere gestiti e protetti.
Un **dato** a se stante è poco utile, è l'unità elementare grezza dell'informazione. Quest'ultima può essere definita come l'**elaborazione** dei dati per rispondere a esigenze specifiche dell'impresa.

Un esempio di **dato** può essere: "Stefano Ceri, Basi di dati, IIIA, I Sem". Da questo dato possiamo estrarre le seguenti informazioni:

- Chi insegna questa materia? Stefano Ceri
- Quando si tiene il corso? Al I semestre del III anno.

Questa distinzione swag.

2 Modelli dei Dati

I modelli costituiscono una strutturazione semplificata della realtà, che ne coglie gli aspetti specifici ed aiuta a comprenderla meglio. Esempi dei **modelli logici** più comuni sono: il modello Gerarchico, il modello Reticolare e il modello Relazionale, sul quale concentreremo la nostra attenzione.

2.0.1 Il Modello Gerarchico e il Modello Reticolare

Queste due strutture sono molto simili, nel senso che in entrambe i dati sono rappresentati in raggruppamenti come **record** , mentre le associazioni tra i dati sono differenti per i due modelli.
Nel modello **Gerarchico** sono rappresentate con **puntatori** in una struttura ad albero, mentre nel modello **Reticolare** come **puntatori** in una struttura a grafo complesso.

2.0.2 Il Modello Relazionale

nel modello relazionale i dati sono invece rappresentati come **tabelle** , mentre le associazioni tra i dati sono ottenute associando i **valori** di attributi in tabelle diverse.
Una definizione formale del modello Relazionale può essere la seguente:

Dominio qualunque insieme di valori;

Prodotto Cartesiano su n domini $D_1 \times ... \times D_n$ non necessariamente distinti insieme delle tuple "$d_1, ..., d_n$" con $d_i \in D_i$ ed $1 \leq i \leq n$;

Relazione su $D_1 \times ... \times D_n$ un qualunque sottoinsieme di D_1 x ... x D_n.

Le **Relazioni** hanno diverse proprietà utili:

Grado della Relazione numero di domini coinvolti;

Cardinalità della Relazione numero di tuple che la compongono;

Attributo nome dato ad un dominio in una relazione. I nomi di attributi devono essere tutti distinti tra loro.

Lo **schema di una Relazione** è il modo in cui la si rappresenta formalmente.
Si scrive come $R(attributo1, ..., attributoN)$ ed è importante ricordare che anche i nomi delle relazioni in uno stesso schema devono essere univoci.
Queste definizioni ad un primo impatto possono risultare fuorvianti per uno studente alle prime armi, si può però fare un esempio di corrispondenza tra definizioni formali e definzioni informali (usate nel linguaggio comune).
Una **Relazione** è comunemente vista come una Tabella, mentre un **attributo** come la Colonna di una Tabella. Una **Tupla** è una riga della Tabella, il **Dominio** è il tipo del dato e la **Cardinalità** è il numero di righe, mentre il **grado** è il numero di colonne.

Interrogazioni Le interrogazioni (eseguite tramite un *Query Language*) sono ciò che permette di passare dal dato all'informazione. Una volta ottenuto uno schema di organizzazione dei dati, come esemplificato nell'introduzione, è possibile ricavarne informazione tramite interrogazioni. Le interrogazioni dovranno ovviamente essere coerenti con i dati raccolti.
Al fine di eseguire in modo ottimale le interrogazioni su un *database* conviene arricchire lo schema, aggiungendo dei **Vincoli di Integrità**.

Vincoli di Integrità Escludono alcune possibili istanze della relazione in quanto non rappresentano concretamente il mondo applicativo. Infatti, è possibile che determinate combinazioni di dati dei diversi domini che compongono la relazione non abbiano "senso" nella realtà che stiamo considerando.
Tali vincoli possono essere, ad esempio:

- **Chiavi**
- **Vincoli su valori nulli**
- **Integrità Referenziale**
- **Vincoli Generici**

In particolare, la **Chiave** ha un ruolo importante nel database, è definita come sottoinsieme di attributi dello schema con le proprietà di *Unicità*, ossia non possono esistere due tuple con l'attributo chiave uguale, e *Minimalità*, ossia, sottraendo un qualunque attributo alla chiave si perde la proprietà di unicità.
Di seguito un esempio, più o meno, riassuntivo.

"Da Filippo"
Via Roma 23
9100 Chissadove
P.I. 012345678

Ricevuta n. 2369
del 12/5/2017

3	coperti	3,15
2	antipasti	6,22
3	primi	12,60
2	bistecche	19,00
	Totale	41,98

"Da Filippo"
Via Roma 23
9100 Chissadove
P.I. 012345678

Ricevuta n. 2456
del 16/5/2017

2	coperti	2,10
1	antipasti	3,11
2	primi	8,40
2	orate	25, 5
2	caffè	1,60
	Totale	39,41

Figure 1: Esempio di due istanze di ricevuta fiscale.

ricevute

NUMERO	DATA	TOTALE
2369	12/5/2017	41,98
2456	16/5/2017	39,41

dettaglio

NUMERO	QUANTITA'	DESCRIZIONE	IMPORTO
2369	3	coperti	3,15
2369	2	antipasti	6,22
2369	3	primi	12,60
2369	2	bistecche	19,00
2456	2	coperti	2,10
2456	1	antipasti	3,11
2456	2	primi	8,40
2456	2	orate	25, 5
2456	2	caffè	1,60

Menu'

PIATTO	COSTO
coperto	1,05
antipasto	3,11
primo	4,20
bistecca	9,50
orata	12,75
caffè	1,60

Figure 2: Esempio del **modello relazionale** derivante dalle ricevute fiscali.

3 Algebra Relazionale

L'Algebra Relazionale, definita da Codd nel 1970, è un utile strumento per imparare a formulare query corrette. Si compone di un insieme minimo di 5 operazioni, la cui combinazione dà l'intero potere espressivo del linguaggio. Le operazioni dell'Algebra Relazionale sono indicate di seguito. (Inserire immagine "Una visione d'insieme" delle operazioni.)

3.1 Selezione

Supponiamo di avere di avere una Relazione STUDENTE composta dei domini Matricola, Nome, Città, ecc..
Indicata come

$$\sigma_{\text{ Nome = "Paola"}} \ STUDENTE$$

, la selezione produce una Tabella, **priva di nome** che ha per Schema lo stesso Schema di STUDENTE, e per Istanza le Tuple di STUDENTE che soddisfano il predicato di selezione.
La sintassi del predicato di Selezione permette di utilizzare espressioni booleane di predicati semplici al suo interno.

3.2 Proiezione

Indicata con

$$\sqcap_{\text{ Nome, Città}} \ STUDENTE$$

la Proiezione produce una tabella, **priva di nome** con Schema gli attributi Nome e Città, ossia quelli indicati al pedice del predicato, e come Istanze la restrizione delle Tuple di STUDENTE (argomento del predicato) agli attributi (a.k.a. campi, domini) Nome e Città.
Va specificato che nel modello **formale** , la Proiezione elimina eventiuali duplicati prodotti implicitamente, ma nel modello **informale** (e nel modello effettivamente implementato nei Sistemi) l'eliminazione di duplicati va richiesta esplicitamente.

3.3 Assegnamento

Non è una vera e propria operazione algebrica dell'Algebra Relazionale, ma è comunque utile e molto usata dato che permette di dare un nome al risultato di un'espressione algebrica. Ad esempio

$$INFORMATICI = \sigma_{\text{CorsoStudi = "Informatica"}} \ STUDENTE.$$

In pratica al risultato prodotto dalle operazioni algebriche, che abbiamo detto essere senza nome, possiamo in questo modo assegnare un nome.

3.4 Unione

$$Tabella1 \cup Tabella2$$

è possibile eseguire l'unione di due tabelle solo se esse sono **compatibili**, ossia con lo stesso grado. Nella realtà applicativa è possibile unire due tabelle solamente se esse hanno domini *ordinatamente dello stesso tipo.*
L'operazione algebrica di Unione produce un risultato che è una Tabella (priva di nome) che ha per Schema lo stesso Schema di Tabella1, e per Istanze l'unione delle tuple di Tabella1 e Tabella2.

3.5 Differenza

Indicata con

$$Tabella1 - Tabella2$$

, si può fare solamente se le due tabelle argomento sono **compatibili**. Produce una tabella priva di nome che ha per Schema lo stesso Schema di Tabella1 e come Istanze la differenza delle tuple di Tabella1 e Tabella2.

3.6 Prodotto Cartesiano

Indicata con

$$R \times S$$

produce una tabella, piva di nome, con Schema l'unione degli attributi di R ed S (quindi il grado di $(R \times S) = grado(R) + grado(S)$), e come Istanze tutte le possibili coppie di tuple di R ed S (quindi la cardinalità di $(R \times S) = card(R) * card(S)$).

3.7 Intersezione

Indicata con

$$R \cap S$$

, l'operazione algebrica di intersezione ha senso solamente quando R ed S sono compatibili (come per gli altri operatori algebrici/insiemistici visti in precedenza).
L'intersezione è derivabile tramite la seguente formula: $R \cap S = R - (R - S)$ e produce come risultato una tabella con lo stesso Schema di R e come Istanze l'intersezione delle tuple di R ed S.

3.8 Join

Indicata con

$$STUDENTE \bowtie_{\text{STUDENTE.Matr = ESAME.Matr}} ESAME$$

ha come espressione equivalente (operatore derivato) la seguente

$$\sigma_{\text{STUDENTE.Matr = ESAME.Matr}} STUDENTE \times ESAME.$$

La notazione puntata è usata per rendere non ambigua l'espressione, quindi per differenziare attributi omonimi all'interno di tabelle diverse. Produce una tabelle, priva di nome, con Schema la **concatenazione** degli Schemi della prima e della seconda tabella e come Istanze le tuple ottenute concatenando quelle tuple di STUDENTE e di ESAME che soddisfano il predicato.
Il Join si differenzia in vari sottotipi.

3.8.1 Join Naturale

equi-join di tutti gli attributi omonimi (si omette il predicato, si elimina la colonna ripetuta).

3.8.2 Semi Join

produce una tabella con Schema lo stesso schema di STUDENTE e come Istanze la proiezione su STUDENTE del Join tra STUDENTE ed ESAME. In parole povere restituisce una tabella (priva di nome) contenente le sole tuple di STUDENTE che rispettano la condizione di Join con ESAME.
Un'espressione equivalente è

$$\sqcap_{\text{Attr(STUDENTE)}} STUDENTE \bowtie_{\text{STUDENTE.Matr = ESAME.Matr}} ESAME.$$

3.8.3 Semi-Join Naturale

proietta su STUDENTE (Semi Join) il Join Naturale di studente ed ESAME.

3.9 Ottimizzazione delle Interrogazoni

Eisitono molte diverse soluzioni per uno stesso problema di interrogazione, e le soluzioni che producono lo stesso risultato sono considerate fra loro **equivalenti**. Alcune di queste espressioni. però riescono a essere più efficienti di altre.
Considerando questo, è naturale volere ottenere sempre delle interrogazioni che svolgono il loro compito più velocemente delle loro alternative. Per ottenere la forma algebrica ottimizzata di una interrogazione dobbiamo stabilire i criteri di ottimizzazione, e per questo scopo conviene utilizzare una rappresentazione della struttura dell'interrogazione **ad albero**.

Ogni espressione dell'algebra relazionale può essere rappresentata in modo grafico da un albero, che esplicita l'ordine di valutazione degli operatori. In questa struttura, ogni operatore corrisponde ad un nodo dell'albero, i cui argomenti sono i nodi figli. Gli operatori unari avranno un ramo in ingreso e un ramo in uscita, mentre quelli binari avranno due rami in ingresso ed uno in uscita.
Abbiamo quindi una serie di **trasformazioni di equivalenza**, la quale lettura è consigliata sulle slide messe a disposizione dal docente.

4 Calcolo Relazionale

Il calcolo relazionale si compone di una famiglia di linguaggi formali, come il Calcolo delle Tuple **TRC** e il Calcolo dei Domini **DRC**. TRC in particolare è diviso in due versioni, con tuple ristrette sul range e con tuple arbitrarie, il nostro caso di studio si concentrerà sul TRC a tuple arbitrarie.

4.0.1 TRC a Tuple Arbitrarie

TRC, come la maggio parte dei linguaggi relazionali, è un linguaggio di tipo dichiarativo, nel senso che si limta ad esprimere la descrizione del risultato del calcolo, ma non il metodo operativo tramite cui ottenerlo.

Definizione Formale di TRC

- Forma Standard: $\{t \mid p(t)\}$, dove $p(t)$ è una **formula** costruita tramite **atomi**.
- Un **atomo** può essere uno scalare o un'espressione del tipo
  espressione - comparatore - espressione. Un'espressione usa costanti e restrizioni sull'attributo A di una tupla t, tali restrizioni si indicano con t[A].
  Cosa importante è che un atomo deve essere una **formula ben formata** (vedi logica del prim'ordine).

Correttezza delle Formule Si considerano corrette solamente le formule la cui soluzione è indipendente dal dominio, ma dipende solamente dalle istanze del database.
Si definiscono formule **unsafe** le formule che danno un risultato infinito, come ad esempio $\{t|t \notin \Re\}$.

5 Datalog

Datalog è un linguaggio di *programmazione logica* basato su "formule", che prendono il nome di **regole**. Ogni regola è composta da una Testa (Head o LHS) ed un Corpo (Body o RHS) e si scrive come

$$P \ :- \ P_1, ..., P_n$$

dove ogni P rappresenta un **predicato** (o letterale) composto da un nome e degli argomenti, che possono essere variabili, costanti o "don't care" (qualsiasi elemento). Un esempio di regola può essere

$$persona(\ X\ ,\ \_\_\ ,\ M\ ).$$

L'**interpretazione** di una regola Datalog è vera se il corpo della regola è vero, ed è importante specificare che le variabili non esplicitamente quantificate all'interno della regola si intendono come quantificate *esistenzialmente*.
Per esemplificare i concetti e le regole del Datalog useremo la seguente Base di Dati.

GENITORE

Genitore	Figlio
Carlo	Antonio
Carlo	Gianni
Anna	Antonio
Anna	Gianni
Gianni	Andrea
Antonio	Paola

PERSONA

Nome	Età	Sesso
Carlo	65	M
Antonio	40	M
Anna	60	F
Gianni	43	M
Andrea	22	M
Paola	20	F

Figure 3: Base di Dati di esempio per le regole Datalog.

5.1 Fatti e Goal

Per **Fatto** in Datalog si intende il letterale *ground*, per esempio

$$Genitore(\text{"}Carlo\text{"},\text{"}Antonio\text{"})$$

ed è in diretta corrispondenza con la **Tupla** di un database.
Un **Goal** corrisponde invece ad una Query.

$$?- \ Padre(\text{"}Carlo\text{"}, X)$$

ed è valutata cercando una regola che abbia per Testa il predicato, nell'esempio, "Padre" e unificando tutte le cariabili X che rendono la regola vera. Il risultato del Goal di esempio è

$$X = \{\text{"}Antonio\text{"},\text{"}Gianni\text{"}\}.$$

Esistono Goal nei quali non sono epresse variabili, questi restituiscono *True o Fase* . Esempi:

$$?- Padre(\text{"}Carlo\text{"},\text{"}Antonio\text{"}) \Rightarrow True$$

$$?- Padre(\text{"}Carlo\text{"},\text{"}Andrea\text{"}) \Rightarrow False.$$

5.2 Esempi di Regole

Di seguito alcuni esempi di regole Datalog con spiegazione:

$$Nonno(X, Z) : -Padre(X, Y), Genitore(Y, Z)$$

Questa Regola, sinteticamente, esprime in che modo un elemento X può essere considerato "Nonno" di un elemento Z. Infatti la Testa della regola è composta dal predicato "$Nonno$" che è vero, in base a quanto detto sulle regole, solamente quando il corpo è vero.
Il corpo della regola è composto dai predicati "$Padre$" e "$Genitore$", separati da una virola che va interpretata come un *Prodotto Logico (AND)*.Questa specifica regola si presta ad una lettura intuitiva, perchè un elemento X sia "Nonno" di un elemento Z è necessario che esista un elemento Y (dato che Y è quantificata esistenzialmente in modo implicito), del quale X è padre, tale che Y è genitore di Z.

$$Fratello(X, Y) : -Genitore(Z, X), Genitore(Z, Y), X \neq Y$$

Anche questa regola, come quella precedente, ha una lettura intuitiva. Esprime in che modo un elemento X è considerato "Fratello" di un elemento Y, cioè quando esiste un elemento Z (quantificato esistenzialmente nella regol in modo implicito), che è genitore di X e di Y, quando X e Y sono due elementi diversi (escudiamo quindi la possibilità

che un elemento sia "Fratello" di sè stesso).

$$Zio(X,Y) : -Persona(X,\_,"M"), Fratello(X,Z), Genitore(Z,Y)$$

In questa regola vediamo il primo esempio di "don't care", un simbolo speciale utilizzato per indicare un parametro di cui non ci interessa il valore. In questo caso il "don't care" è l'età della persona che possiamo considerare "Zio". Dovrebbe essere immediata a questo punto la lettura della suddetta regola. Consideriamo un elemento X "Zio" di un elemento Y quando: X è una persona di sesso maschile della quale non ci interessa l'età ed esiste un elemento Z, di cui X è fratello, tale che Z è genitore di Y.

5.3 Negazione

Ovviamente per scrivere regole Datalog è possibile usare la negazione (anzi, essa aumenta il potere espressivo del linguaggio), essa però introduce dei problemi in quanto a correttezza. Ad esempio

$$q(x) : - \sim p(X)$$

$$p(0)$$

Non è safe! Perchè

$$? - p(0)$$

produce un risultato **infinito**.
Per usare correttamente l'operatore di negazione è necessario che la regola abbia 2 proprietà:

- La negazione deve essere **safe**. Tutte le variabili di un letterale negato devono comparire anche in un letterale positivo all'interno della regola.

  $$P(X) : - R(X), \sim P(X)$$

  Non è corretta.

- La negazione deve essere **stratificata**. Ossia non ci devono essere cicli di dipendenza tra letterali negati.

A fronte di questi accorgimenti, la negazione aumenta il potere espressivo di Datalog, superando anche quello dell'Algrbra Relazionale, per il fatto di poter scrivere Query **ricorsive**.

5.4 Query Ricorsive

Una Query ricorsiva si distingue dalle altre perchè contiene all'interno del corpo della regola anche il letterale di testa. Ad esempio:

$$Start : Antenato(X,Y) : -Genitore(X,Y)$$

$$Ricorsione : Antenato(X,Y) : -Antenato(X,Z), Genitore(Z,Y)$$

Il meccanismo di valutazione di una Query Datalog ricorsiva è molto simile a quello usato per il risolutore di un insieme di clausule (vedi corso di Logica e Algebra), valutando ad ogni passo la Query corrente e unendo il risultato alla Query del passo ricorsivo precedente, fino a quando

$$Antenato^{\text{n}} = Antenato^{\text{n-1}}$$

cioè si raggiunge il **punto fisso**.
La negazione descritta sopra può causare grandi problemi con questo tipo di Query. Ad esempio, la seguente Query

$$Antenato(X,Y) : -Genitore(X,Y)$$

$$Antenato(X,Y) : -Antenato(X,Z), Genitore(Z,Y)$$

$$Non-ant(X,Y) : -Persona(X,\_,\_), Persona(Y,\_,\_), \sim Antenato(X,Y)$$

se valutata nel modo che abbiamo descritto prima produce un risultato sbagliato, poichè al primo passo di iterazione, calcoliamo NON-ANT sottraendo il valore corrente della relazione ANTENATO (che è uguale a GENITORE) dal prodotto cartesiano delle persone. Al secondo passo di iterazione, aggiungiamo al valore precedente di NONANT il nuovo valore computato, che sarà **più piccolo**, etc. quindi NON-ANT non varia più.
Il risultato finale sarà scorretto, perché conterrà tutte le persone che non hanno tra loro un rapporto genitore-figlio.
Questo succede perchè non si è rispettata la proprietà di *stratificazione* della Query ricorsiva, la proprietà della negazione che abbiamo descritta come necessaria in precedenza.
Lo stesso programma Datalog riscritto stratificato produce un risultato corretto ed è anche di più facile lettura:
Strato 1:

$$Antenato(X,Y) : -Genitore(X,Y)$$

$$Antenato(X,Y) : -Antenato(X,Z), Genitore(Z,Y)$$

Strato 2:

$$Non-ant(X,Y) : -Persona(X,\_,\_), Persona(Y,\_,\_), \sim Antenato(X,Y).$$

La differenza sostanziale rispetto alla precedente è che nel primo strato calcoliamo iterativamente la relazione ANTENATO come prima, e procediamo col secondo strato solamente una volta raggiunto il punto fisso del primo strato.
Non tutti i programmi però sono stratificabili, è necessario che il grafo delle dipendenze non presenti cicli labellati da negazione.

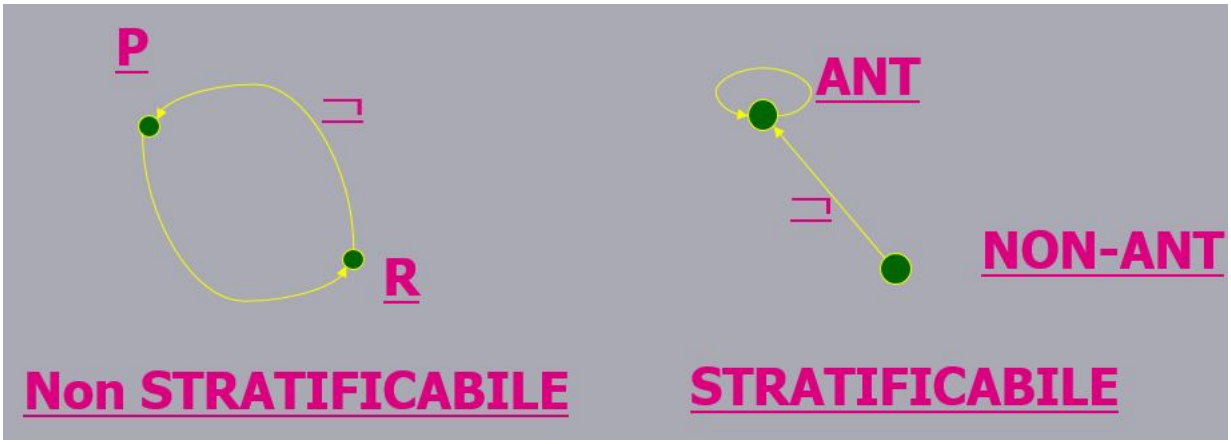

Figure 4: Esempi di grafi delle dipendenze di programmi stratificabili e non.

5.5 Corrispondenze con il modello relazionale standard

Possiamo stabilire una corrispondenza tra gli elementi del modello relazionale standard e la terminologia Datalog.

- Relazione ⇒ Attributo
- Attributo ⇒ Argomento
- Tupla ⇒ Fatto
- Query ⇒ Regole
- Interrogazione ⇒ Goal

6 Structured Query Language (SQL)

Il nome SQL indica un insieme di linguaggi: DDL (Data Definition Language) e DML (Data Management Language), utili a definire domini, tabelle, indici, autorizzazioni, viste, vincoli, procedure, trigger (DDL) e a scrivere Query, modificare la struttura di un database già formato, scrivere comandi transazionali.

6.1 Definizione degli Schemi

Per Schema si intende una collezione di oggetti: domini, tabelle, indici, asserzioni, viste, privilegi, con **nome e proprietario**.
Un esempio di Schema è

`create schema` [*NomeSchema*]

[[`authorization`] Autorizzazione]

{ *DefinizioneElementoSchema* }

6.2 Query SQL

6.2.1 Interrogazioni Nidificate

In termini semplici. è un blocco SQL "dentro" un blocco SQL. Il meccanismom è simile a quello della ricorsione. Esistono 3 modi per nidificare le Query SQL:

Nella clausola Where con i predicati ANY, ALL, o altri operatori di comparazione. L'operatore ANY rende vera l'espressione se esiste almeno una tupla che soddisfa la condizione.
L'operatore ALL rende vera l'espressione se tutte le tuple soddisfano la condizione.
Esempio di query con ANY ed ALL misti ad altri operatori:

```
select CodOrd
  from Ordine
where Importo > any
```

Query annidata

```
select Importo
  from Ordine
```

`select CodOrd`

`from Ordine`

`where Importo` $\geq$ **ALL**

Query annidata

`select Importo`

`from Ordine`

IN e NOT IN sono i corrispondendi di = ANY e $\neq$ ALL, sono i concetti intuitivi di appartenenza e non appartenenza ad un insieme.

Differenza C'è una effettiva differenza tra le quesy nidificate e le loro equivalenti non nidificate a volte. Guarda sulle slide e vedi l'esempio in cui la prima query, quella nidificata, estrae solo una volta il cliente, mentre la seconda che in teoria è equivalente mostra tante volte il cliente quanti sono gli ordini che ha fatto.
In teoria ci sono vari modi per scrivere una stessa query, e sono tutti qeuivalenti A MENO DEI DUPLICATI.

Variabili Nelle Query nidificate le variabili possono essere usate per "passare" i valori da una query alla sua sottoquery. Una variabile rappresenta una tupla, posso quindi definire una variabile O in una query e riusarla nella sua sottoquery come O.attributo. Questi strumenti sono molto utili per il push della selezione rispetto al Join
Una cosa IMPORTANTISSIMA da tenere a mente quando si decide di usare le variabili in una query con nidificazioni è quella di **rispettare le regole di visibilità** delle variabili. Come nel C (più o meno), le variabili possono essere utilizzate nella query in cui si definiscono oppure nelle sottoquery ricorsive nidificate direttamente "sotto" di essa, non in altre query ricorsive in un altro livello.

Costruttori di Tupla (Nome, Cognome) "costruisce" virtualmente una tupla formata dai parametri Nome e Cognome, che posso poi usare per confrontarla ad esempio con i risultati di una sottoquery che ha la forma

`select Nome, Cognome`

Equivalenza del potere espressivo Ci sono delle classi di equivalenza tra operatori:
IN, =ANY, EXIST
NOT IN, $\neq$ ANY, NOT EXIST
"comparatore" ANY, Theta Join (non Equi-Join)
"comparatore" ALL, query che abbiano raggruppamento od estrazione di minimo o massimo.

Modifiche Le query nidificate ricorsive semplificano molto anche le modifiche dei dati nel database.

6.2.2 Viste

Le viste offrono la "visione" di Tabelle Virtuali con schemi diversi da quello del database originale. Sono classificabili in Viste semplici e complesse, quelle semplici hanno solo operazioni di selezione e proiezione su una sola tabella.
Possono anche essere usate per realizzare Query, come strumento di modularizzazione, scomponendo una Query in "pezzi" (senza eccedere, ovviemente). Sono anche talvolta **necessarie** per l'espressione di alcune Query, in particolare quelle che combinano nidificazioni di operatori aggregati diversi. Una volta creata una Vista posso usarla a tutti gli effetti come una Tabella del Database nella scrittura di una Query.

Le seguanti Quey estraggono il cliente che ha generato il maggior fatturato dal database "Ordini", sono realizzate nel modo "più classico" e la sua forma equivalente con l'utilizzo delle viste. Query "Classica":

```
select CodCli
from Ordine
groupby CodCli
having sum(Importo) ≥ ALL (
    select sum(Importo)
    from Ordine
    groupby CodCli )
```

Query con View, creazione della Vista:

```
create view CliFatt(CodCli, FattTotale) as
select CodCli, sum(Importo)
from Ordine
groupby CodCli )
```

Query con View, scrittura della Query finale:

```
select CodCli
from Ordine
where FattTotale = [
select max(Fatttotale)
from CliFatt ]
```

Ovviamente le viste possono anche essere usate per scrivere Query di modifica del database. La maggiore "differenza" con le Query "normali" è che le operazioni di modifica sulle Viste (una volta create), sono possibili solamente sulle Viste Semplici, mentre non sono possibili su quelle complesse. Un esempio di Query complessa è una qualsiasi Query che usa anche solamente un Join, e il problema fondamentale è che con l'utilizzo del Join (o tutti gli operatori simili) si va a calcolare una tabella con divesre tuple ottenute dalla combinazione delle tuple delle tabelle operando sulla base della condizione. Quindi, volendo aggiornare un parametro che è presnte nella Vista si crea un **ambiguità**, dato che viene ad essere ambigua la "posizione" del dato da modificare, siccome può essere presente in due tabelle diverse.

6.2.3 Ricorsione con Viste

6.2.4 Controllo dell'Accesso

L'obiettivo è quello di eseguire un controllo selettivo su chi ha la possibilità di accedere a determinati dati, in modo tale da proteggerli da accessi non autorizzati. Il controllo di accesso e modifica si implementa attraverso il meccanismo delle **autorizzazioni**.

Copia dalle slide.

Grant e Revoke. Ciascuno, quando ha la grant option, può passare ad un altro utente solamente i privilegi che ha (ricevuto), non quell che non ha.