Basi di Dati

Luca Seggiani

10 aprile 2024

1 Introduzione alla base di dati relazionale

Una base di dati (in inglese "database") è una mole di dati organizzata in modo da favorirne la consultazione attraverso determinate interrogazioni ("query"). Formalmente, una base di datiè un archivio di dati contenente informazioni ben strutturate organizzate secondo un modello logico (nel caso di MySQL un modello relazionale).

L'elemento fondamentale di una base di dati fondata sul modello relazionale è la **Tabella**, formata da colonne (associate a determinati attributi), e righe (che rappresentano record dei suddetti attributi). Una colonna particolare, che deve differire per ogni record della tabella, viene detta chiave primaria e serve a distinguere tra di loro record (che potrebbero essere altrimente stati tra di loro identici!).

L'interrogazione della base di dati si effettua attraverso le cosiddette query, ovvero richieste compilate in linguaggio SQL (Structured Query Language), un linguaggio dichiarativo sviluppato appositamente per questo scopo. La query più semplice si basa su tre istruzioni: SELECT, FROM e WHERE. Prendiamo l'esempio in lingua naturale di un'interrogazione a una base di dati contenente dati su diverse persone (nello specifico, nome, cognome, età e codice fiscale come chiave.):

Riportare nomi e cognomi di tutte le persone con eta maggiore di 40 anni.

in questo caso gli attributi di interesse sono i nomi e cognomi (in quest'ordine) delle persone valide, e la condizione è un età maggiore di 40 anni. Di fronte a questa richiesta, la base di dati risponderà fornendo un insieme risultato (result set) contenente una tabella formata esattamente dalle informazioni richieste. In SQL potremmo definire tale query come segue:

```
SELECT Cognome, Nome
FROM Persona
WHERE Eta > 40
```

in questo caso, cognome e nome sono gli attributi di interesse che verrano inseriti nel nostro insieme risultato (attraverso un processo chiamato proiezione). La tabella persona sarà la fonte dei nostri dati, e età > 40 la nostra condizione.

Le condizioni in SQL supportano anche i classici operatori logici (AND, OR, ecc...). Un operatore degno di nota, applicabile ai soli dato di tipo numerico o su cui è comunque stabilita una relazione d'ordine, è il BETWEEN, che permette di selezionare (estremi inclusi) tutti i valori in un dato range.

Duplicati

2 righe risultano uguali se tutti gli attributi del record hanno valori identici. Non si possono avere righe uguali in tabelle SQL (la chiave distingue), ma è possibile avere duplicati tra gli attributi non chiave, con quindi risultanti problemi dati dalla proiezione di tali attributi su insiemi risultato privi di attributo chiave. Nel caso si rendesse necessario eliminare i duplicati da una data query, l'SQL offre il costrutto SELECT DISTINCT. Si nota che il costo per l'eliminazione dei duplicaty è computazionalmente $O(n^2)$, e quindi da evitare quando possibile.

Valori NULL

I valori NULL corrispondono solitamente a informazioni mancanti, non disponibili o comunque sconosciute. Qualsiasi condizione che coinvolge valori NULL è sempre falsa, compreso il confronto fra stessi NULL. È possibile utilizzare i valori NULL assegnandoli un valore semanticamente definito (e.g. una data non definita perchè non ancora determinata, ecc...). In questo caso, l'SQL offre i costrutti IS NOT NULL e IS NULL, che restituiscono rispettivamente falso e vero posti accanto ad un valore NULL. Si nota inoltre che la chiave di un record non può essere NULL.

Gestione delle date in SQL

Come per tutti i sistemi UNIX, le date in SQL vengono rappresentate come una distanza in secondi dalla mezzanotte del primo gennaio 1970 (il cosiddetto tempo di riferimento EPOCH, *EPOCH reference*).

Formato di data Esistono 2 tipi di dato data in SQL: DATE (formato YYYY-MM-DD) e TIMESTAMP (formato YYYY-MM-DD HH:MM:SS). La manipolazione di date è possibile attraverso la funzione DATE_FORMAT, che consente di cambiare il formato o effettuare manipolazioni d'utilità. Ad esempio potrò avere:

```
SELECT Matricola, DATE_FORMAT(DataLaurea, '%d|%m|%Y, %w')
```

```
FROM Studente
WHERE DataLaurea > 12-7-2004
```

che prende tutti gli studenti laureati dopo il 12 luglio 2004 e ne riporta il nome di matricola e la data formattata come DD-MM-YYYY + settimana. ancora, per includere ad esempio gli studenti laureati di mercoledì:

```
SELECT Matricola
FROM Studente
WHERE DATE_FORMAT(DataLaurea, '%w') = 3
```

dove il '%w' restituisce il giorno della settimana di DataLaurea. Qualsiasi ulteriore informazione sui formati delle date può essere trovata al link:
`https://www.w3schools.com/sql/func_mysql_date_format.asp`

Poniamoci adesso il problema di restituire i cognomi di tutti gli studenti laureati 5 anni fa, ovvero con un offset di 5 anni rispetto alla data attuale. Servirà chiaramente un qualche riferimento alla data di oggi, che l'SQL fornisce con la parola chiave CURRENT_DATE. Potrò quindi avere:

```
SELECT DISTINCT Cognome
FROM Studente
WHERE DataLaurea IS NOT NULL
  AND YEAR(DataLaurea) = YEAR(CURRENT_DATE) - 5
```

si noti che abbiamo usato l'operatore "-": l'SQL fornisce anche la funzione DATEDIFF(dataRecente, dataRemota), che restituisce il numero di giorni che separano due date. Nota bene: somme e sottrazioni di date non hanno senso: tutto quello che possiamo fare è sommare i singoli valori (anno, giorno, ecc...) di date diverse. Per qualsiasi altro caso occorre usare DATEDIFF(). Possiamo avere ad esempio:

```
SELECT Matricola, DATEDIFF('2005-07-15', DataIscrizione)
FROM Studente
WHERE DataIscrizione < '2005-07-15'
  AND DataLaurea > '2005-07-15'
```

che restituirà la matricola e giorni da quando si erano iscritti degli studenti ad oggi laureati e che non si erano ancora laureati il 15 Luglio 2005.
Inoltre, per sommare e sottrarre lassi di tempo a date possiamo usare le funzioni DATE_ADD e DATE_SUB. I suddetti lassi andranno espressi con la parola chiave INTERVAL:

```
INTERVAL NumeroIntero [YEAR|MONTH|DAY]
```

riportiamo ad esempio matricola e mese di iscrizione degli studenti che si sono laureati dopo cinque anni esatti dal giorno dell'iscrizione:

```
SELECT Matricola, MONTH(DataIscrizione)
```

```
FROM Studente
WHERE DataLaurea = DATE_ADD(DataIscrizione, INTERVAL 5 YEAR)
```

quando la somma è definita fra date e intervalli, possiamo usare anche solo l'operatore "+":

```
  AND DataLaurea = DataIscrizione + INTERVAL 5 YEAR
```

esistono poi diverse altre funzioni di utilità sulle date, che possono essere trovate nel sito precedentemente citato.

Operatori di aggregazione

Gli operatori di aggregazione permettono di fare determinati calcoli i cui operandi sono i valori assunti da un attributo in un insieme di record (ovvero conteggio, somma, minimo / massimo, media), e di collassarli in un unico attributo numerico. Gli operatori di aggregazione sono disponibili solamente nel SELECT, in quanto il WHERE non ha alcuna visibilità a livello globale della tabella su cui lavora, ma solo a livello di record.

Conteggio

Iniziamo col contare il numero di righe di una tabella o di un suo sottoinsieme. Definita la tabella di una semplice realtà medica, abbiamo:

```
SELECT COUNT(*) AS VisitePrimoMarzo
FROM Visita
WHERE Data = '2013-03-01'
```

per trovare il numero di visite effettuate in data 1 marzo 2013. La parola chiave AS serve solamente per rinominare la tabella generata dalla funzione COUNT(). Possiamo effettuare anche, anzichè il conteggio delle righe, il conteggio dei valori diversi assunti da un attributo. Ad esempio, se cercassimo il numero di pazienti visitati nel mese di marzo 2013, visto che lo stesso paziente potrà essere stato visitato più volte in un solo mese, dovremmo fare:

```
SELECT COUNT(DISTINCT Paziente) as PazientiMarzo
FROM Visita
WHERE MONTH(Data) = '03'
  AND YEAR(Data) = '2013'
```

Somma

Posso sommare i valori numerici di degli attributi di più record usando SUM(). Supponiamo di voler calcolare, data una tabella del reddito di più persone, il reddito totale di una sola famiglia:

```
SELECT SUM(Reddito) AS RedditoTotale
FROM Paziente
WHERE Cognome = 'Lepre'
```

Media

Allo stesso modo, potrò calcora il reddito medio:

```sql
SELECT AVG(Reddito) AS RedditoMedio
FROM Paziente
WHERE Cognome = 'Lepre'
```

Minimo / Massimo

Potrò inoltre calcolare valori massimi e minimi:

```sql
SELECT MIN(Reddito)
FROM Paziente

SELECT MAX(Reddito)
FROM Paziente
```

A questo punto, se volessimo cercare il reddito massimo, e il nome e cognome di chi lo detiene, incontreremmo un'ostacolo: non si può infatti ottenere un qualsiasi altro attributo da un insieme risultato ormai collassato ad un solo valore numerico. Non otterremmo nulla dal codice:

```sql
SELECT MAX(Reddito), Nome, Cognome
FROM Paziente
```

Riassumendo: non si possono affiancare agli operatori di aggregazione i nomi di attributi ormai collassati.

Query su più tabelle

In un database le informazioni sono spesso frammentate su più tabelle. Questo aiuta a evitare ridondanze, anomalie, ed a vere la possibilità di distribuire i dati. Ad esempio, immaginiamo il database di una clinica medica, che dovrà quindi immagazinare i dati di pazienti, dottori, visite mediche ecc... Potremo allora definire più tabelle separate per ciascuna di queste categorie di dati, ognuna con i propri specifici attributi. A questo punto, ogni tabella potrà contenere come attributi altre tabelle, o meglio record provienienti da altre tabelle in qualche modo "collassati" nel singolo attributo della tabella. Questo meccanismo si concrettizza immagazzinando nella tabella l'informazione minimale per ritrovare il record desiderato nella tabella d'appartenenza, ovvero riportandone soltanto la chiave (che sappiamo essere diversa per ogni record archviato). La successiva esplosione della chiave fino al record completo nella tabella d'appartenenza viene effettutato in SQL attraverso le operazioni di unione (join).

Inner join

L'inner join trova il record nella tabella d'appartenenza che corrisponde alla chiave nella tabella su cui lavoriamo, e semplicemente lo affianca. Ad esempio, magari vogliamo indicare nome e cognome dei medici che hanno effettuato, nella nostra clinica, almeno una visita. Dovremmmo allora prendere la nostra tabella visita, che conterrà la chiave di un medico nella tabella medico, che

andremo quindi a sostituire con il record completo del medico corrispondente. In codice:

```
SELECT DISTINCT M.Nome, M.Cognome
FROM Visita V
  INNER JOIN
  Medico M ON V.MEDICO = M.Matricola
```

Riprendiamo la trattazione delle operazioni che permettono l'unione (join) di più tabelle.

Natural join

Il join naturale combina i record della prima tabella con i record della seconda tabella aventi valori uguali su tutti gli attributi omonimi. Ad esempio, in pseudocodice, ammettendo che la tabella visita abbia un attributo ominimo ad un'altro attributo sulla tabella medico:

```
  SELECT M.Nome, M.Cognome
  FROM Visita V
    NATURAL JOIN
    Medico M
```

Nota bene: **tutti** gli attributi omonimi dovranno avere valori identici, ed a quel punto il record della tabella di provenienza verrà unito alla tabella su cui lavoriamo una sola volta.

External join

Il join esterno combina tutti i record della prima tabella con tutti i record della seconda che soddisfano una condizione, mantenendo tutti i record in una delle tabelle, a differenza del join esterno che invece scarterebbe i record non appaiati con nessuno dei record della tabella madre. Il join esterno può essere pensato come una sorta di prodotto cartesiano fra tabelle. I record spaiati che verrano uniti avranno tutti valori NULL sugli attributi della tabella di provenienza (inesistenti). Poniamo ad esempio di voler indicare le visite effettuate da medici che non lavorano più presso la clinica. Supponiamo quindi che loro anagrafica sia stata quindi eliminata dal database.

```
SELECT V.*
FROM Visita V
  LEFT OUTER JOIN
  Medico M ON V.Medico = M.Matricola
WHERE M.Matricola IS NULL;
```

Così facendo potremo unire tutti i record di tutti i medici, anche quelli che non lavorano più nella mia clinica. Controllando a questo punto chi di questi medici ha elemento chiave (si controlla, per convenzione, proprio l'elemento chiave) NULL, otterremo le visite svolte dai suddetti. Notiamo infine la differenza fra join esterno destro e join esterno sinistro: il join esterno sinistro mantiene tutti i record della tabella di sinistra mettendo NULL a destra

per i record non appaiati. Il join esterno destro fa la cosa opposta, quindi mantenendo i record a destra e inserendo NULL a sinistra.

Query con join e condizioni sui record

Possiamo a questo punto combinare il FROM con instruzione di join allo WHERE con condizioni di selezione dei record d'interesse applicate alla tabella *dopo* il join. Ad esempio, se voglio ottenere matricola, cognome e specializzazione dei medici che hanno visitato almeno un paziente il giorno 1 Marzo 2013

```
SELECT DISTINCT M.Matricola, M.Cognome, M.Specializzazione
FROM Medico M
  INNER JOIN
  Visita V ON M.Matricola = V.Medico
WHERE V.Data = '2013-01-03'
```

Unioni multiple

Il join multiplo può essere effettuato su più di 2 tabelle. Diciamo che oltre al medico, la nostra tabella visita contiene un'attributo corrispondente al paziente visitato. Vogliamo allora ottenere nomi e cognomi di tutti i pazienti visitati da un certo medico. Notiamo che il nome del medico sta nella sottotabella corrispondente al medico, e il nome del paziente sta nella sottotabella corrispondente al paziente, ovvero dovremmo unire non una ma due tabelle alla nostra tabella madre. In codice:

```
SELECT DISTINCT P.Nome, P.Cognome
FROM Paziente P
  INNER JOIN
  Visita V ON P.CodFiscale = V.Paziente
  INNER JOIN
  Medico M ON V.Medico = M.Matricola
WHERE M.Nome = 'Rino'
  AND M.Cognome = 'Neri';
```

notiamo che in questo caso, per assicurare la sicurezza dell'operazione, nome e cognome del medico dovrebbero essere una cosiddetta *chiave candidata*, ovvero un insieme di attributi sempre diversi in tutti i record (come la chiave primaria ma senza lo status di chiave primaria).

Si noti inoltre di come gli alias (P V e M) nell'esempio precedente hanno il compito di evitare ambiguità su attributi con lo stesso nome ma appartenenti a tabelle diverse. Ulteriore esempio, supponiamo di voler indicare nome e cognome dei pazienti visitati nel mese di dicembre 2013, e il nome e cognome dei medici che li hanno visitati:

```
SELECT DISTINCT P.Nome AS NomePaziente, P.Cognome AS
    CognomePaziente
                M.Nome AS NomeMedico, M.Cognome AS
    CognomeMedico
```

```
FROM Paziente P
  INNER JOIN
  Visita V ON P.CodFiscale = V.Paziente
  INNER JOIN
  Medico M ON V.Medico = M.Matricola
WHERE YEAR(V.Data) = 2013
  AND MONTH(V.Data) = 12
```

Self join

Il self join combina le righe di una tabella con righe della stessa tabella. Più propriamente, non esiste una keyword SELF JOIN, ma possiamo sfruttare le possibilità dell'INNER JOIN. Diciamo ad esempio di volere il codice fiscale dei pazienti che sono stati visitati più di una volta da uno stesso medico della clinica, nel mese corrente:

```
SELECT DISTINCT V1.Paziente
FROM Visita V1
  INNER JOIN
  Visita v2 ON (
                V2.Medico = V1.Medico
                AND V2.Paziente = V1.Paziente
                AND V2.Data<> V1.Data
                )
WHERE MONTH(V1.Data) = MONTH(CURRENT_DATE)
  AND YEAR(V1.Data) = MONTH(CURRENT_DATE)
  AND MONTH(V2.Data) = MONTH(CURRENT_DATE)
  AND YEAR(V2.Date) = YEAR(CURRENT_DATE);
```

qui unisco alla mia tabella visita un record proveniente dalla stessa tabella quando medico, paziente corrispondono e la data differisce. A questo punto seleziono soltanto i record della nuova tabella creata che hanno date diverse fra le due visite. Si noti che per n visite, il paziente comparirà nel result set n - 1 volte (da cui il DISTINCT), ovvero il join accade in una sola direzione.

Common Table Expression

Le common table expression (CTE) sono result set dotati di identificatori che possono essere usati prima di una query per costruire risultati intermedi. Si scrivono, separate da virgole, prima della query che li usa, tramite la parola chiave WITH e separati da virgole. Sono parti di codice i cui risultati vengono stoccati e poi restituiti alle query che nli usano. Ad esempio:

```
WITH
  name1 AS (query1)
  name2 AS (query2)
  ...
  nameN AS (queryN)
query finale;
```

Diciamo per esempio, di volere indicare il numero di pazienti di Siena mai visitati da ortopedici. Avrò allora la CTE che trova tutti gli ortopedici:

```
WITH ortopedici AS
  (
  SELECT M.Matricola AS Medico
  FROM Medico M
  WHERE M.Specializzazione = 'Ortopedia'
  )
```

poi la CTE che trova tutti i pazienti visitati da tali ortpedici:

```
paz_visitati_ortopedici AS
  (
  SELECT V.Paziente AS CodFiscale
  FROM Visita V NATURAL JOIN Ortopedici O
  )
```

infine ottengo, usando le CTE precedenti, tutti i pazienti senesi:

```
SELECT COUNT(*)
FROM Paziente P
  NATURAL LEFT OUTER JOIN
  paz_visitati_ortopedici PVO
WHERE Citta = 'Siena'
  AND PVO.Matricola IS NULL
```

2 Subquery

Le subquery (query annidate) permettono di incapsulare query in altre query, in modo correlato o non correlato. Presentano un modo alternativo di risolvere i problemi su cui normalmente si userebbe il join: infatti per ogni problema che si può risolvere usando il join esiste sempre un corrispettivo che usa le subquery. Chiamiamo outer query la query più esterna, cioè quella che incapsula, e inner query la subquery incapsulata.

Subquery non correlate

Le subquery non correlate (*noncorrelated*) sono incapsulate nel WHERE, e permettono di ottenere un certo result set intermedio, che verrà poi usato dall'outer query per calcolare il result set finale. Il result set della subquery non correlata non dipende dall'outer query. Poniamo ad esempio di voler indicare nome, cognome e parcella degli ortopedici che hanno effettuato almeno una visita nell'anno 2013. Definiremo allora la subquery:

```
SELECT V.Medico
FROM Visita V
WHERE YEAR(V.Data) = 2013
```

che innesteremo poi in:

```
SELECT M.Nome, M.Cognome, M.Parcella
FROM Medico M
WHERE M.Specializzazione = 'Ortopedia'
  AND M.Matricola IN  (
                      SELECT V.Medico
                      FROM VIsita V
                      WHERE YEAR(V.Data) = 2013
                      );
```

La parola chiave IN permette di controllare la presenza dell'attributo M.Matricola nel result set della subquery. La subquery calcolerà quindi un result set composto da tutte le visite fatte nel 2013. L'outer query userà la subquery, confrontandola con tutti i medici specializzati in ortopedia la cui matricola trova un riscontro nel result set calcolato dalla subquery. A questo punto si restituiscono gli attributi richiesti. Il meccanismo è del tutto analogo a quello di un join naturale fra le tabelle "Medico" e "Visita". Notiamo che non è necessario alcuna istruzione DISTINCT, in quanto ogni record della tabella Medico viene considerato comunque una volta sola, e cercato nel result set delle visite.
Avremmo potuto porre la stessa query usando un comune join, ad esempio come:

```
SELECT M.Nome, M.Cognome, M.Parcella
FROM Medico M
  INNER JOIN Visita V ON V.Medico = M.Matricola
WHERE YEAR(V.Data) = 2013
  AND Medico.Specializzazione = 'Ortopedico';
```

Si può anche usare la negazione di quanto appena fatto, usando la parola chiave NOT IN. Ad esempio, se vogliamo indicare i cognomi dei pazienti che non appartengono anche a un medico, abbiamo la subquery:

```
SELECT M.Cognome
FROM Medico M
```

innestata in:

```
SELECT DISTINCT P.Cognome
FROM Paziente P
WHERE P.Cognome NOT IN  (
                        SELECT M.Cognome
                        FROM Medico M
                        );
```

Analogamente a prima, esisterà una versione che usa il join:

```
SELECT DISTINCT P.Cognome
FROM Paziente P
```

```
  LEFT OUTER JOIN Medico M ON
    M.Cognome = P.Cognome
WHERE M.Cognome IS NULL;
```

Annidamento multiplo

Non c'è teoricamente limite al numero di subquery che si possono innestare l'una dentro l'altra. A volte può infatti tornare utile avere innesti multipli (e frontali!). Vediamo come ottenere il numero di tutti i pazienti di Siena mai visitati da pazienti. Innanzitutto si trovano i medici specializzati in ortopedia:

```
SELECT M.Matricola
FROM Medico M
WHERE M.Specializzazione = 'Ortopedia'
```

poi i pazienti visitati da suddetti medici:

```
SELECT V.Paziente
FROM Visita V
WHERE V.Medico IN (...)
```

e si rimuovono da una tabella dei soli pazienti senesi:

```
SELECT COUNT(*)
FROM Paziente P
WHERE P.Citta = 'Siena'
  AND P.CodFiscale NOT IN (
    SELECT V.Paziente
    FROM Visita V
    WHERE V.Medico IN (
      SELECT M.Matricola
      FROM Medico M
      WHERE M.Specializzazione = 'Ortopedia'
    )
  );
```

Subquery scalari

Una subquery scalare produce, invece che un insieme di record che poi verrà controllato con IN e NOT IN, un singolo record di un singolo attributo, come ad esempio farebbe la funzione COUNT(). A questo punto, il valore scalare prodotto può essere controllato con i vari operatori di confronto. Vogliamo ad esempio indicare il numero degli otorini aventi parcella più alta della media delle parcelle della loro specializzazione. Troviamo innanzitutto il la parcella media degli otorini (un valore scalare):

```
SELECT AVG(Medico.Parcella)
FROM Medico M
WHERE M.Specializzazione = 'Otorinolaringoiatria'
```

e la useremo in un'outer query:

```
SELECT COUNT(*)
FROM Medico M1
WHERE M1.Parcella >  (
                      SELECT AVG(Medico.Parcella)
                      FROM Medico M
          WHERE M.Specializzazione = 'Otorinolaringoiatria'
                      );
```

Un caso interessante: mettiamo di voler trovare le entrate complessive generate dai cardiologi della clinica negli ultimi 2 anni. Ciò risulta semplice con un join:

```
SELECT SUM(M.Parcella) AS IncassoTotale
FROM Visita V
```

```
INNER JOIN Medico M ON V.Medico = M.Matricola
WHERE V.Data > CURRENT_DATE - INTERVAL 12 YEAR
  AND M.Specializzazione = 'Cardiologia'
```

La versione con subquery è invece più complessa, richiedendo una subquery nel SELECT:

```
SELECT SUM((
  SELECT M.Parcella
  FROM Medico M
  WHERE M.Matricola = V.Medico
     AND M.Specializzazione = 'Cardiologia'
)) AS IncassoTotale
FROM Visita V
WHERE V.Data > CURRENT_DATE - INTERVAL 12 YEAR
```

Notiamo che la subquery è correlata, come vedremo nella prossima lezione.

3 Raggruppamento

Il raggruppamento (o aggregazione) divide in gruppi i record risultanti dalle clausole FROM e WHERE (detti record target), sulla base di un certo attributo particolare in comune, che resta quindi costante in un determinato gruppo. Poniamo di voler calcolare, nello schema clinica, per ogni specializzazione, la parcella media dei medici:

```
SELECT Specializzazione, AVG(Parcella) AS ParcellaMedia
FROM Medico
GROUP BY Specializzazione;
```

Dove il select di Specializzazione e AVG(Parcella) ha senso solo a seguito del GROUP BY, con Specializzazione proprio come attributo di raggruppamento (!).

Condizioni sui gruppi

Le condizioni sui gruppi sono controllate gruppo per gruppo (non record per record) e permettono di scartarne alcuni. Si applicano attraverso l'operatore HAVING. Poniamo di voler indicare le specializzazioni della clinica con più di due medici:

```
SELECT Specializzazione
FROM Medico
GROUP BY Specializzazione
HAVING COUNT(*) > 2
```

Si nota che il DISTINCT non è necessario: ogni specializzazione viene presa comunque una volta sola. Vediamo adesso un caso un attimo più complesso: ottenere le specializzazioni con la più alta parcella media. Dovremo allora

calcolare tutte le medie delle parcelle dei medici di ciascuna specializzazione, calcolare tra queste la più alta, e selezionare quindi le specializzazioni la cui media corrisponde a questa media massima.

Il massimo si troverà con:

```
SELECT MAX(D.MediaParcelle)
FROM (
  SELECT M2.Specializzazione, AVG(M2.Parcella) AS
   MediaParcella
  FROM Medico M2
  GROUP BY M2.Specializzazione
)
AS D;
```

e la query completa sarà:

```
SELECT M.Specializzazione
FROM Medico M
GROUP BY Specializzazione
HAVING AVG(Parcella) =
(
  SELECT MAX(D.MediaParcelle)
  FROM (
    SELECT M2.Specializzazione, AVG(M2.Parcella) AS
   MediaParcella
    FROM Medico M2
    GROUP BY M2.Specializzazione
  )
  AS D;
)
```

4 Subquery correlate

Abbiamo già visto le subquery non correlate (*non-correlated*): il result set di tali subquery viene calcolato una sola volta ed è indipendente dalle specifiche della query esterna (*outer query*). Nelle subquery correlate invece il result set dipende da ciascuna tupla della tupla esterna. L'ordine di esecuzione è: prima si esegue il WHERE della query esterna, e poi la subquery. Poniamo ad esempio di voler indicare matricola e parcella dei medici che hanno visitato per la prima volta almeno un paziente nel mese di ottobre 2013. Avremo bisogno di selezionare pazienti che non erano mai stati visitati da un determinato medico prima di ottobre:

```
SELECT DISTINCT V1.Medico
FROM Visita V1
WHERE YEAR(V1.Data) = 2013
  AND MONTH (V1.Data) = 10
  AND V1.Paziente NOT IN (
                            SELECT V2.Paziente
                            FROM Visita V2
```

```
                                WHERE V2.Medico = V1.Medico
                                  AND V2.Data < V1.Data
                              )
```

Notiamo la particolarità della subquery correlata: la relazione V1, appartenente alla query esterna, viene usata nella clausola WHERE della correlata. Ricordiamo che una subquery correlata viene eseguita nuovamente per ogni tupla, ed è quindi particolarmente inefficiente dal punto di vista della complessità. Notiamo inoltre che una subquery può essere usata anche nella clausola SELECT della query, per calcolare velocemnte un valore da inserire nel risultato. Ad esempio, vogliamo trovare tutti i pazienti di sesso maschile, indicandone nome e numero di visite effettuate. Quest'ultima informazione non sarà un dato da proiettare (nel WHERE), ma da calcolare nel SELECT.

```
SELECT Nome, (
  SELECT COUNT(*)
  FROM Visite V
  WHERE V.Paziente = P.CodFiscale
) AS NumeroVisite
FROM Paziente P
WHERE P.Sesso = 'M'
```

5 Introduzione alla teoria delle basi di dati

Che cos'è l'informatica?

- L'informatica è la scienza del trattamento razionale, spesso attraverso macchine automatiche, dell'informazione.

possiamo fare una distinzione fra:

- metodologica
- tecnologica

6 Sistema informativo

- Il sistema organizzativo è costituito da risorse e regole per lo svolgimento coordinato di attività di una certa organizzazione (azienda, ente, ecc...) Noi ci concentreremo sul
- Sistema informativo, ovvero la parte del sistema organizzativo che acquisice, conserva, elabora e produce informazioni d'interesse per l'organizzazione. Inoltre esegue e gestisce i processi informativi (che coinvolgono informazioni).

possiamo analizzare più nel dettaglio i tipo di attività svolte dal sistema informativo:

- Raccolta e acquisizione di informazioni
- Archiviazione e conservazione di suddette informazioni
- Elaborazione, trasformazione e ancora produzione di nuove informazioni sulla base di quelle ottenute
- Distribuzione e comunicazione delle informazioni così elaborate.

Occorre fare attenzione: il sistema informativo non è di per se in alcun modo legato all'informatica (esistono sistemi informativi, si pensi ai servizi anagrifici o alle banche, che non usano alcuna automatizzazione).
La parte del sistema informativo che usa la tecnologia informatica è il sistema informativo automatizzato, o semplicemente **sistema informatico**.
Ricapitolando, possiamo stabilire la seguente relazione:

Azienda → Sistema organizzativo → Sistema informativo → sistema informatico

7 Gestione delle informazioni

L'informazione può essere gestita secondo modalità diverse, e su supporti diversi. Ad esempio, abbiamo:

- Idee informali
- Linguaggio naturale
- Disegni, schemi, grafici
- Numeri e codici

su vari supporti:

- Mente umana
- Carta
- Dispositivi elettronici (e.g. hard disk, ecc...)

Definiamo ora la differenza fra informazioni e dati:

- **Informazione**
  Notizia, dato o elemento che consente di avere una conoscenza dei fatti, situazioni o modi di essere.

- **Dato**
  Ciò che è immediatamente presente alla conoscenza prima dell'elaborazione. In informatica, elementi di informazione costituiti da *simboli* che devono essere elaborati.

I dati sono spesso codifiche particolari di informazioni, che vanno quindi da essi estrapolate. Ad esempio, il codice fiscale, i cartelli stradali, ecc...

8 La base di dati

Il cuore di un sistema informativo automatizzato è la base di dati (database), cioè un insieme organizzato di dati rappresentanti informazioni di interesse. Nelle due accezioni (metodologica e tecnologica), possiamo dire:

- Metodologica: insieme organizzato di dati utilizzati come supporto per lo svolgimento di attività

- Tecnologica: insieme di dati gestito da un DBMS (Database Management System)

Le basi di dati hanno solitamente:

- Dimensioni molto maggiori della memoria centrale dei sistemi di calcolo utilizzati

- Tempo di vita indipendente dalle signole istanze dei programmi che li utilizzano (persistenza dei dati)

- Supporto per gestione di collezioni di dati condivise fra più dispositivi

- Capacità di garantire privacy, affidabilità, efficienza ed efficacia

Vediamo nel dettaglio dell'aspetto di condivisione:

- Ogni organizzazione è divisa in settori o almeno svolge disparate attività

- Ciascun settore potrebbe essere fornito di un sottosistema informativo, magari disgiunto a quello principale.

Questo può chiaramente portare a problemi di:

- **Ridondanza**: ripetizione dell'informazione
- **Incoerenza**: più versioni dell'informazione che non coincidono.

Nota: perchè non usare un semplice archivio di file invece che di un database?
Un archivio non fornisce alcuna gestione dell'interdipendenza fra informazioni, ed è quindi poco portato agli eventuali controlli sulla coerenza e la correttezza dell'informazione. Inoltre, in un comune filesystem abbiamo a disposizione solamente le operazioni rudimentali di scrittura/lettura, senza particolari controlli su concorrenza o funzionalità particolari.

Per ovviare a tutta questa serie di problemi, introduciamo:

- **Autorizzazione**: gestione dell'accesso a date risorse
- **Concorrenza**: gestione dell'accesso *simultaneo* (!) a date risorse
- **Affidabilità**: resistenza a malfunzionamenti hardware e software. Una tecnica fondamentale in questo campo è la corretta gestione delle **transazioni**
- **Efficienza**: gestione ottimale delle risorse in termini di memoria e tempo
- **Efficacia**: offerta di funzionalità articolate, potenti e flessibili.

Vediamo un'attimo nel dettaglio l'aspetto delle transazioni:

Transazioni

Una transazione è un insieme di operazioni da considerare indivisibili (*"atomiche"*), corretto anche in presenza di concorrenza, e con effetti definitivi a fine esecuzione. Una transazione deve essere eseguita *per intero* o *per niente*. La corretta gestione della concorrenza deve invece assicurarsi che transazioni concorrenti vengano gestite correttamente (o in serie, e. g. transazioni bancarie, o in maniera mutualmente esclusiva, e. g. prenotazione biglietto aereo). La transazione dovrà poi essere permanente, ovvero la conclusione positiva di una transazione corrisponde ad un impegno (commit) a mantenere traccia della versione ormai aggiornata dei dati a seguito della stessa.

Descrizione dei dati

Nei programmi tradizionali che accedono a file, ogni programma contiene una descrizione della struttura del file stesso, con i conseguenti rischi di incoerenza fra informazioni e file stessa. Ecco perchè nei DBMS è opportuno dedicare una porzione della base di dati alla descrizione centralizzata dei dati. Introduciamo il concetto di **modello dei dati**: un insieme di costrutti utilizzati

per organizzare i dati di interesse e descriverne le varie dinamiche, fornendoci quindi una vista astratta dei suddetti. Occorre quindi definire la differenza fra:

- **Schema**: descrizione della struttura della dei dati, solitamente invariante nel tempo (attributi)
- **Istanza**: i valori attuali immagazzinati nella base di dati, che variano nel tempo (record).

introdiciamo l'elemento fondamentale della base di dati: la **tabella**. Una comune tabella di una base di dati contiene come intestazioni delle sue colonne gli attributi (schema) dei dati immagazinati, mentre le successive righe presentano vari record (istanze) dei dati definiti dalle colonne.

9 Modelli dei Dati

Modelli logici
Adottati nei DBMS esistenti per l'organizzazione dei dati, indipendenti dalle strutture fisiche. Esempi: relazionale, reticolare, gerarchico, a oggetti, basato su XML.

Modelli concettuali
Permettono di rappresentare i dati in modo indipendente da ogni sistema, cercando di descrivere i concetti del mondo reale. Sono utilizzati nelle fasi preliminari di progettazione.
Esempio: Entity-Relationship (ER).

Possiamo dire che l'utente si interfaccia con lo schema logico di un database, dettato dal modello logico adottato. A livello fisico accade quanto definito nello schema interno, ovvero la parte di effettiva implementazione del DBMS.

Schema logico
Lo schema logico è la descrizione della base di dati nel modello logico, quindi la struttura di tabelle, ecc... L'utente che sviluppa il database si interfaccia con lo schema logico, usufruendo delle sue astrazioni.

Schema interno
Lo schema interno è l'implementazione dello schema logico, ed è noto solo a chi implementa il DBMS.

10 Linguaggi per Basi di Dati

I DBMS dispongono di vari linguaggi e interfacce per la definizione di schemi, modifica e lettura dei dati, e formmulazione di query. Possiamo avere:

- Linguaggi testuali interattivi (SQL)
- Comandi (SQL) immersi in un linguaggio ospite (Java, C++, ...)
- Interfaccie grafiche (Access)

Si può fare l'ulteriore distinzione:

- **Data definition language** (DDL): per la definizione di schemi (logici e fisici)
- **Data manipulation language** (DML): per l'interrogazione e l'aggiornamento di istanze di basi di dati.

11 Architettura a tre livelli per DBMS

Possiamo complicare le cose introducendo, oltre allo schema interno e lo schema logico, un'ulteriore schema, lo schema esterno, attraverso cui permetteremo agli utenti di interfacciarsi con la base di dati a livello ancora più alto (astratto).
Lo schema esterno immplementa fondamentalmente viste parziali (magari solo alcune tabelle) di database. Sarà in ogni caso importante stabilire:

- Indipendenza fisica: il livello logico e quello esterno sono indipendenti da quello fisico, come nel paradigma dell'astrazione procedurale, la realizzazione dello schema fisico può variare senza che debbano essere attuate modifiche dello schema logico.
- Indipendenza logica: il livello esterno è poi indipendente da quello logico, aggiunte o modifiche alle viste non richiedono modifiche al livello logico, e viceversa le modifiche dello schema logico lasciano inalterato lo schema esterno.

12 Attori

Possiamo adesso stabilire quali sono gli "attori" che implementeranno, lavoreranno su ed interagiranno con il DBMS:

- Progettisti e realizzatori di DBMS
- Progettisti della base di dati e suoi amministratori
- Progettisti e programmatori di applicazioni
- Utenti:
  - Utenti finali: eseguono applicazioni predefinite:
  - Utenti casuali: eseguono operazioni noon previste a priori, usando linguaggi interattivi (SQL)

13 Modello relazionale

Vale la pena riportare i 3 modelli logici storici, tra cui figura il modello relazionale che andremo a studiare:

gerarchico, reticolare, relazionale

Più recentemente si è poi diffuso il paradigma ad oggetti, basato su XML, anche detto NoSQL (viene principalmente usato su database di dimensioni particolarmente grandi).

I modelli gerarchici e reticolari non sono molto utilizzati per un particolare difetto: la gestione di relazioni fra dati non viene gestita in maniera efficiente (si usa il meccanismo dei riferimenti, che sono però fin troppo dipendenti dalla struttura fisica adottata). Nel modello relazionale, invece, tutto è basato sui valori, completamente slegati alle specificità della struttura fisica. Gli stessi riferimenti, o più propriamente relazioni, sono basati su valori condivisi fra più tabelle.

Tabelle
La tabella è l'entità fondamentale di un database relazionale. Come già visto prima, una tabella contiene righe (record) e colonne (attributi), che rappresentano in un qualsiasi momento un istanza dello schema logico adottato.
Relazioni
Le relazioni del database relazionale si basano sul concetto matematico di relazione. Poniamo ad esempio due insiemi:

$$D_1 = (a, b)$$

$$D_2 = (x, y, z)$$

e il loro prodotto cartesiano:

$$D_1 \times D_2 = ((a, x), (a, y), (a, z), (b, x), ...)$$

una certa relazione R è:

$$R \subseteq D_1 \times D_2$$

sottoinsieme del loro prodotto cartesiano. Formalmente:
Dati n insiemi (anche non distinti) $D_1, D_2, ..., D_n$, e definito su di essi un prodotto cartesiano $D_1 \times D_2 \times ... \times D_n$, ovvero l'insieme di n-uple ordinate $(d_1, d_2, ..., d_n)$ con $d_1 \in D_1, d_2 \in D_2, ..., d_n \in D_n$, una relazione è un sottoinsieme del prodotto cartesiano fra gli insiemi.
Non esiste ordinamento fra le n-uple, e non esistono n-uple uguali. Gli insiemi $D_1, D_2, ..., D_n$ sono detti domini della relazione.

Struttura non posizionale
Dando un certo nome ai domini della relazione, possiamo passare da quella che è effettivamente una struttura posizionale (le n-uple sono ordinate), ad una struttura dove le posizioni dei domini (attributi) non sono più rilevanti.

Riferimenti fra relazioni
I riferimenti fra relazioni diverse sono rappresentati da valori dei domini che compaiono nelle n-uple.

Definizioni formali
Abbiamo adesso tutti gli strumenti necessari a definire formalmente il modello relazionale:

- **Schema di relazione**: simbolo R detto nome della relazione e un insieme di attributi $X = (A_1, ..., A_2)$ solitamente indicato con $R(X)$. A ciascun attributo X è associato un dominio. Gli attributi non possono essere relazioni.

- **Schema di base di dati**: un insieme di schemi di relazione con nomi diversi, solitamente indicato come $R = (R_1(X_1), ..., R_n(X_m))$.

N-uple
Una n-upla o tupla su insieme di attributi X è una funzione che associa a ciascun attributo $A \in X$ un elemento, o valore, nel dominio di A. Il simbolo $t[A]$ denota il valore della n-upla t sull'attributo A. Il simbolo $t[Y]$, con $Y \subseteq X$, denota il valore della n-upla t sull'insieme di attributi Y. N.B.: le n-uple non sono ordinate!

Definiamo allora le istanze di relazioni (e quindi di database):

- **Istanza di relazione**: su uno schema $R(X)$, un insieme r di n-uple su X

- **Istanza di base di dati**: su uno schema $R = (R_1(X_1), ..., R_n(X_m))$, un insieme di relazioni $r = (r_1, ..., r_2)$ dove ogni r_i è una relazione sullo schema $R_i(X_i)$

Graficamente, una n-upla è una riga della tabella, e un istanza l'insieme di tutte le sue righe.

Informazione incompleta
Il modello relazionale impone ai dati una struttura rigida, ma non è detto che i dati del mondo reale aderiscano sempre a questa struttura. Si introduce allora la parola chiave NULL, per evitare di usare valori particolari del dominio (0, ecc...), che potrebbero diventare significativi, o la cui gestione è comunque abbastanza complicata. Il valore di un certo attributo A, ovvero $t[A]$, potrà quindi appartenere al dominio D_A oppure essere il valore NULL. Si possono inoltre imporre determinate restrizioni sulla presenza dei valori nulli in una data relazione. Un valore NULL può modellizzare almeno 3 casi differenti:

- valore sconosciuto
- valore inesistente
- valore senza informazione

N.B.: Il DBMS non fa alcuna distinzione fra valori NULL!
La limitazione sui valori che possono o non possono essere NULL torna utile ad esempio nel caso della definizione di chiavi primarie di record, che devono essere fra di loro diverse ma comunque mai nulle.

14 Correttezza di basi di dati

Una base di dati può essere scorretta, ovvero immagazinare dati incompatibili con la realtà dei fatti. Ad esempio, sarebbe impensabile avere una base di dati contenente voti relativi ad esami universitari, e trovare in tale base di dati la valutazione "27 e lode". La correttezza dei dati può essere assicurata grazie al sistema dei vincoli:

15 Vincoli di integrità

I vincoli di integrità sono sostanzialmente funzioni booleane (predicati) che associano alla istanza completa di basi dati un valore vero o falso. Se il vincolo di una base di dati restituisce vero, significa che le sue proprietà sono soddisfatte. Possiamo distinguere i vincoli in:

- **Vincoli intra-relazionali**: i vincoli intra-relazionali sono definiti rispetto ad una singola relazione. Possiamo portare gli esempi:

  - Vincolo di n-upla: può essere valutata su ciascuna n-upla indipendentemente dalle altre
  - Vincolo di dominio: vincolo di n-upla che coinvolge un solo attributo.

- **Vincoli inter-relazionali**: i vincoli inter-relazionali sono definiti rispetto a più relazioni diverse fra loro.

16 Chiavi

Una chiave è un insieme di attributi che identifica univocamente le n-uple. Formalmente: una **superchiave** è un insieme K di attributi di un insieme r se r non contiene due n-uple distinte t_1 e t_2 tali che $t_1[K] = t_2[K]$. Una chiave K è tale per un insieme r se è una superchiave minimale in R (cioè non contiene un'altra superchiave). N.B.: non per forza una chiave è unica.

Qualsiasi relazione ammette sempre almeno una chiave, in quanto una relazione contiene n-uple tutte diverse fra loro, ergo tutto l'insieme degli attributi forma sempre una chiave.
L'esistenza di una chiave garantisce l'accessibilità a ciascun dato della base di dati, e permette di correlare fra loro dati in relazoni diverse. Le chiavi formano inoltre un vincolo di integrità, detto appunto vincolo di chiave.
Nel caso una relazione abbia più di una chiave, ne scegliamo una in particolare, la chiave primaria (*primary key*). Le altre chiavi verranno allora dette chiavi candidate.

Chiavi e valori nulli
La presenza di valori nulli nelle chiavi può rappresentare un grande problema: rende impossibile il riferimento da parte di altre relazioni e non permette l'identificazione univoca di ogni record. Va dunque evitato quando possibile.

Dipendenze Funzionali
I vincoli di chiave sono particolari tipo di vincoli, che fanno parte di una categoria può vasta: le **dipendenze funzionali**. Formalmente: dati due insiemi di attributi X e Y sulla relazione R, si dice che X determina Y, e si scrive $X \to Y$, se e solo se per ogni coppia di n-uple distinte t_1 e t_2, se $t_1[X] = t_2[X]$, allora $t_1[Y] = t_2[Y]$.

17 Integrità referenziali

Informazioni in relazioni diverse possono essere correlate attraverso valori comuni. Definiamo allora il concetto di chiave esterna (*foreign key*), ovvero

un'insieme di attributi di una relazione che corrispondono ad una chiave di un'altra relazione.

Vincolo di integrità referenziale
Un vincolo di integrità referenziale fra gli attributi X di una relazione R_1 (chiave esterna di R_1) e un'altra relazione R_2 impone ai valori su X di R_1 di comparire come valori della chiave primaria di R_2. N.B.: in questo caso l'ordine dei valori è significativo.

Integrità referenziale e valori nulli
In presenza di valori nulli i vincoli possono essere resi meno restrittivi: scegliamo da R_1 solamente i valori di X diversi da NULL.

18 Operazioni di aggiornamento

Le operazioni di aggiornamento di una relazione consistono in sostanza di:

- Operazioni di inserimento, può violare:
  - vincoli intra-relazionali
  - integrità referenziale
- Operazione di cancellazione, può violare:
  - integrità referenziale
- Operazione di modifica (cancellazione + inserimento), di norma non viola nulla.

La correttezza di queste operazioni è assicurata dal DMBS, che reagirà alle violazioni dei vincoli attraverso azioni compensative.

Azioni compensative Nel caso si provi ad eliminare una n-upla causando violazionzi del vincolo di chiave, il comportamento standard potrebbe essere quello del rifiuto dell operazione. Si potrebbero sennò adottare altre azioni compensative, come ad esempio:

- Eliminazione in cascata
- Introduzione di valori nulli

19 Linguaggi per basi di dati

I linguaggi per basi di dati appartengono a 2 categorie distinte:

- Operazioni sullo schema: **Data Definition Language (DDL)**;
- Operazioni sui dati: **Data Manipulation Lanaguage (DML)**
  - Interrogazione (query)
  - Aggiornamento (update).

Interrogazioni di basi di dati
Un operazione di interrogazione è un operazione di lettura che richiede l'accessso a una o più tabelle. Per specificare interrogazioni si possono seguire due formalismi:

- **Modo dichiarativo**: si specificano le proprietà del risultato (che cosa)
- **Modo procedurale**: si specificano le modalità di generazione del risultato (come).

L'**algebra relazionale** permette di specificare delle interrogazioni secondo il modello procedurale: cioè elencando i passi "primitivi" necessari alla generazione di una risposta. Il **calcolo relazionale** invece ci permette di definire in modo dichiarativo quello che sarà il risultato dell'interrogazione. Come vedremo, sarà il calcolo relazionale a definire la semantica del linguaggio relazionale, perchè permette di fornire un'implementazione slegata dai dettagli procedurali.

20 Algebra relazionale

Un'algebra è una struttura matematica dotata di un'insieme di dati e una serie di operatori che manipolano suddetti dati. Nell'algebra relazionale abbiamo che:

- **Dati**: relazioni
- **Operatori**:
  - su relazioni
  - che producono relazioni
  - che possono essere composti.

essi sono divisi fra:

- **Operatori su insiemi**:
  unione, intersezione, differenza
- **Operatori su relazioni**:
  ridenominazione, selezione, proiezione,
  join:
  - naturale, prodotto cartesiano, theta

Notazione dell'algebra relazionale
Specifichiamo adesso la notazione usata:

- $R, R_1, ...$ indicano nomi di relazionie
- $A, B, C, A_1...$ indicano nomi di attributo
- XY è un'abbreviazione di $X \cup Y$
- Una relazione con n-uple $t_1, t_2, ...$ è indicata con l'insieme $\{t_1, t_2, ...\}$
- $t_j[A_j]$ indica il valore della n-upla t_j sull'attributo A_j
- $t[X]$ indica l'n-upla ottenuta da considerando solo gli elementi di X.

Operatori su insiemi
Le relazioni sono insiemi di tuple, e non possono avere elementi duplicati (sarebbero altrimenti multiinsiemi). I risultati di operazioni fra relazioni sono a loro volta relazioni, ovvero insiemi di tuple. Gli operatori fra relazioni possono applcarsi solo e soltanto fra relazioni definite sullo stesso insieme di attributi X, e il risultato sarà a sua volta definito sullo stesso insieme di attributi X.

Unione
L'operatore di unione fra due relazioni sull'insieme di attributi X comporta la formazione di una nuova relazione che ha tutte le n-uple delle relazioni unite. Eventuali n-uple identiche fra le relazioni appariranno nel risultato una volta sola. Il suo simbolo è $\cup$.

Intersezione
L'operatore di intersezione fra due relazioni sull'insieme di attributi X restituisce una nuova relazione che contiene soltanto gli elementi appartenenti sia alla prima che alla seconda relazione. Il suo simbolo è $\cap$.

Ðifferenza
L'operatore di differenza fra due relazioni sull'insieme di attributi X restituisce una nuova relazione contenente tutte le n-uple che appartengono alla

prima relazione ma non alla seconda. L'operatore di differenza è l'unico fra gli operatori insiemistici a non essere commutativo. Il suo simbolo è il meno $(-)$, o più propriamente $\setminus$.

Notiamo che l'unione è l'unico operatore in grado di creare relazioni con un numero di elementi maggiore di quello degli operandi. Questa caratteristica tornerà poi utile nel calcolo relazionale.

Descriviamo adesso gli operatori non insiemistici (sulle relazioni):

Ridenominazione

L'operatore di ridenominazione è un'operatore monadico che modifica lo schema dell'operando, lasciando inalterata l'istanza. Si scrive, data una relazione R, come:

$$\rho_{B_1 B_2 \ldots} \leftarrow_{A_1 A_2 \ldots (R)}$$

I pedici a sinistra e a destra della freccia sono insiemi di attributi. Avremo che:

- L'attributo A_1 viene sostituito dall'attributo B_1
- L'attributo A_2 viene sostituito dall'attributo B_2
- ecc...

Selezione

L'operatore di selezione è un operatore monadico che produce un un risultato con lo stesso schema dell'operando, e un sottoinsieme di n-uple che rispettano una determinata condizione. Si scrive come:

$$\sigma_F(R)$$

sulla relazione R, dove F è un'espressione Booleana (predicato) ottenuta componendo con gli operatori logici AND, OR e NOT le condizioni atomiche:

- $A \star B$, dove A e B sono attributi di X con domini compatibili e $\star$ un operatore di confronto.
- $A \star k$, dove A è un attributo di X e k una costante con dominio compatibile con A e $\star$ sempre un operatore di confronto.

Notiamo che la condizione atomica è vera solo per valori non nulli in ogni attributo.

Selezione valori NULL

Per riferirsi ai valori NULL occorre usare le apposite condizioni: IS NULL e IS NOT NULL.

Proiezione

La proiezione è un'operatore monadico che produce un sottoinsieme degli attributi dell'operando contenente tutte le sue n-uple ristrette soltanto ad alcuni attributi. Si scrive, data una relazione $R(X)$ e un insieme di attributi $Y \subseteq X$, come:

$$\pi_Y(R)$$

Il risultato è una relazione su Y che contiene l'insieme delle n-uple di R ristrette ai soli attributi di Y. Notare che, di nuovo, non possono esserci righe ripetute. Ogni n-upla ripetuta nell'insieme risultato verrà inserita una volta sola.

Cardinalità di proiezioni

Una proiezione contiene al massimo tante n-uple quante ne contiene l'operando. Inolre, se l'insieme di attributi X è superchiave della relazione R, allora la proiezione $\pi_X(R)$ avrà tante n-uple quante ne ha l'operando.

Join

Non possiamo, usando gli operatori di selezione e proiezione, estrarre e combinare informazioni da più relazioni diverse fra loro, e non possiamo nemmeno combinare informazioni presenti in n-uple diverse di una stessa relazione. Avremo allora bisogno di un'ulteriore serie di operazioni, le cosiddette operazioni di join. I join ci permettono di unire sulla stessa riga più righe di relazioni diverse. Vediamo nel dettaglio:

Join naturale

Il join naturale è un'operatore con due operandi (generalizzabile), che produce come risultato l'unione degli attributi degli operandi, contente le n-uple costruite ciascuna a partire da un n-upla di ognuno degli operandi. Formalmente: Date due relazioni R_1X_1 e R_2X_2, il loro join naturale si scrive come:

$$R_1 \bowtie R_2$$

Il risultato è una relazione $R(X_1 \cup X_2)$ definita come:

$$R(X_1 \cup X_2) = R_1(X_1) \bowtie R_2(X_2) = \{t | \exists t_1 \in R_1, \quad t_2 \in R_2$$

$$\text{con} \quad t[X_1] = t_1, \quad t[X_2] = t_2\}$$

N.B.: è perfettamente plausibile voler fare il join naturale tra due relazioni senza attributi in comune, e in questo caso si ottiene il prodotto cartesiano fra le due.

Cardinalità del join

Il join di due relazioni R_1 e R_2 contiene un numero di n-uple compreso fra 0 e il prodotto di $|R_1|$ e $|R_2|$. Se il join coinvolge una chiave di R_2 allora il

numero di n-uple è compreso fra 0 e $|R_1|$. Se il join coinvolge una chiave di R_2 e un vincolo di integrità referenziale allora il numero di n-uple è uguale a $|R_1|$. Più formalmente:
Il join di $R_1(A, B)$ e $R_2(B, C)$ contiene un un numero di n-uple:

$$0 \leq |R_1 \bowtie R_2| \leq |R_1| \times |R_2|$$

Se B è una chiava di R_2 allora il numero di n-uple è

$$0 \leq |R_1 \bowtie R_2| \leq |R_1|$$

Se B è una chiave di R_2 ed esiste un vincolo di integrità referenziale fra B (in R_1) e R_2 allora il numero delle n-uple è:

$$|R_1 \bowtie R_2| = |R_1|$$

Una problematica del join è il fatto che le n-uple che non contribuiscono al risultato ("che non fanno join") restano tagliate fuori. Introduciamo allora altri tipi di operatori di join, i cosiddetti:

Join esterno
Il join esterno estende, con valori NULL, le n-uple che verrebbero altrimenti scartate dal join (interno). Ne esistono tre versioni:

- **Sinistro**: mantiene tutte le n-uple del primo operando ($\bowtie_{LEFT}$);
- **Destro**: mantiene tutte le n-uple del secondo operando ($\bowtie_{DESTRO}$);
- **Completo**: mantiene tutte le n-uple di entrambi gli operandi ($\bowtie_{FULL}$).

Join e proiezioni
Date due relazioni $R_1(X_1)$ e $R_2(X_2)$:

$$\pi_{X_1}(R_1 \bowtie R_2) \subseteq R_1$$

Date una relazione $R(X)$ con $X = X_1 \cup X_2$

$$(\pi_{X_1}(R) \bowtie \pi_{X_2}(R)) \supseteq R$$

Prodotto cartesiano
Come detto prima, date due relazioni $R_1(X_1)$ e $R_2(X_2)$, senza attributi in comune, cioè con $X_1 \cap X_2 = 0$, la definizione di join naturale e comunque senso e restituisce il prodotto cartesiano delle due relazioni:

$$R = R_1 \bowtie R_2 = R_1 \times R_2$$

la cardinalità del prodotto cartesiano è uguale al prodotto delle cardinalità delle due relazioni.

Theta join

Nella pratica, il prodotto cartesiano ha senso quasi solamente se seguito da una selezione $\sigma_F(R_1 \times R_2)$. Questa combinazione prende il nome di theta join (è un operatore derivato) ed è indicato come:

$$R_1 \bowtie_F R_2$$

dove F è un certo rpedicato. Spesso F è una congiunzione di atomi di confronto $A_1 \sigma A_2$ dove σ è un operatore di confronto ($<=$, $<$, $=$, ecc...) e A_1 e A_2 attributi di relazioni diverse. Quando l'operatore di confronto è l'uguaglianza parliamo di equi-join.

Tutti gli operatori visti finora, ovvero gli operatori insiemistici e quelli relazionali, bastano a realizzare qualsiasi possibile interrogazione. Ogni altro operatore sarà un' operatore derivato dei 5 operatori relazionali e i 3 poeratori insiemistici. Un particolare operatore derivato è:

21 Divisione

Dati due insiemi di attributi disgiunti X_1 e X_2, una relazione r sulla loro unione e una relazione r_2 su X_2, la divisione $r \div r_2$ è una relazione su X_1 che contiene le n-uple ottenute come "proiezione" di n-uple di r che si combinano con tutte le n-uple di r_2 per formare n-uple di r, in una sorta di prodotto cartesiano inverso. In simboli:

$$r \div r_2 = \{t_1[X_1] \quad | \quad \forall t_2 \in r_2 \quad \exists t \in r : t[X_1] = t_1, \quad t[X_2] = t_2\}$$

possiamo dimostrare che è un operatore derivato definendolo come composizione di operatori fondamentali, in questo modo:

$$r \div r_2 = \pi_{X_1}(r) - \pi_{X_1}((\pi_{X_1}(r) \times r_2) - r)$$

Ovvero:

- $\pi_{X_1}(r) \times r_2$ contiene tutte le n-uple di $\pi_{X_1}(r)$ "estese" con tutti i possibili valori di r_2.
- $(\pi_{X_1}(r) \times r_2) - r$ contiene le "estensioni" di $\pi_{X_1}(r)$ che non compaiono in r.
- $\pi_{X_1}((\pi_{X_1}(r) \times r_2) - r)$ contiene le n-uple di $\pi_{X_1}(r)$ per le quali un qualche completamento con r_2 non compare in r.
- Togliendo queste ultime n-uple a $\pi_{X_1}(r)$ otteniamo tutte le n-uple di $\pi_{X_1}(r)$ che si combinano con tutte le n-uple di r_2.

22 Chiusura transitiva

Poniamoci il problema di dover trovare, in un'opportuna tabella supervisione formata da matricole di impiegati e supervisori di tali impiegati, le matricole di tutti i supervisori di un dato impiegato (ammettendo che i supervisori siano impiegati e possano a loro volta avere supervisori). Tale richiesta sarebbe perfettamente valida, ma impossibile da esprimere attraverso gli operatori dell'algebra relazionale.
Nell'algebra relazionale non esiste la possibilità di esprimere interrogazioni che calcolino la chiusura transitiva di una relazione arbitraria. Tale operazione potrebbe infatti richiedere un numero infinito di di join (join illimitato).

23 Espressioni equivalenti

Due espressioni sono equivalenti se producono lo stesso risultato qualunque sia l'istanza fornitagli. In questo caso, sarà opportuno scegliere espressioni di costo minore, dove il loro "costo" è determinato dalle dimensioni delle istanze intermedie che la loro esecuzione genera. Vediamo alcune equivalenze fondamentali:

- **Atomizzazione delle selezioni**: una congiunzione di selezioni può essere sostituita da una sequenza di selezioni atomiche:

$$\sigma_{F_1 \wedge F_2}(E) = \sigma_{F_1}(\sigma F_2(E))$$

- **Idempotenza delle proiezoni**: una proiezione può essere trasformata in una sequenza di proiezioni:

$$\pi_X(E) = \pi_X(\pi_{XY}(E))$$

- **Push selections down**: se una condizione F coinvolge solo attributi dell'espressione E_2:

$$\sigma_F(E_1 \bowtie E_2) = E_1 \bowtie \sigma_F(E_2)$$

- **Push projections down**: se un'espressione E_1 ha attributi X_1, un'espressione E_2 ha attributi X_2, $Y_2 \subseteq X_2$ e gli attributi $X_2 - Y_2$ non sono coinvolti nel join ($X_1 \cap X_2 \subseteq Y_2$):

$$\pi_{X_1 Y_2}(E_1 \bowtie E_2) = E_1 \bowtie \pi_{Y_2}(E_2)$$

24 Ottimizzazione delle interrogazioni

Un modulo presente nel DBMS è il **query processor** (od ottimizzatore). L'ottimizzatore si occupa di scegliere la strategia realizzativa a partire dall'istruzione in linguaggio dichiarativo di alto livello, tenendo conto del costo di implementazioni diverse. Le fasi di eseecuzione di una query saranno:

- Analisi lessicale, sintattica e semantica della query in linguaggio di alto livello (SQL). Al termine di questa analisi, la query verrà tradotta in un'espressione dell'algebra relazionale.
- Ottimizzazione algebraica: a questo punto verranno calcolate una o più nuove espressioni algebriche equivalenti a quella di partenza, sfruttando le equivalenza sopra descritte.
- Ottimizzazoine basata sui costi: fra le alternative calcolate prima, viene selezionata la più efficiente, che viene poi utilizzata per effettuare l'interrogazione effettiva sulla base di dati.

Nella prima e l'ultima di queste fasi, il DMBS interagisce con un componente detto catalogo, che contiene informazioni sugli schemi contenuti nel database, la cardinalità delle loro istanze, ecc..

Profili delle relazioni

Tra le informazioni quantitative memorizzate nel catalogo troviamo:

- Cardinalità di ciascuna relazione;
- Dimensioni delle n-uple;
- Dimensioni dei valori;
- Numero di valori distinti degli attributi;
- Valore minimo e massimo degli attributi.

Queste informazioni vengono usate nella fase di ottimizzazione basata sui costi per stimare le dimensioni dei risultati intermedi di più espressioni algebriche alternative generate dall'ottimizzazione algebrica.

Ottimizzazione algebrica

In verità, il termine ottimizzazione non è completamente accurato: il processo utilizza infatti delle euristiche per trovare risultati migliori. Si basa sul concetto di equivalenza per trovare query che restituiscono lo stesso risultato generando dimensioni d'istanza intermedie minori. Ad esempio, un'ottimizzazione tipica è quella di eseguire selezioni e proiezioni il più presto possibile, ovvero le cosiddette "push selections down" e "push projections down".

25 Grafi

Un grafo $G = (V, E)$ consiste in un insieme V di vertici (o nodi) e un insieme E di coppie di vertici, detti archi. Ogni arco ovviamente connetter fra loro due vertici. Possiamo allora distinguere:

- **Grafi orientati**: detti anche grafi diretti, dove ogni arco è orientato è rappresenta relazioni ordinate fra oggetti;
- **Grafi non orientati**: detti anche grafi indiretti, dove ogni arco non è orientato e rappresenta relazioni simmetriche fra oggetti.

Sui grafi si possono definire **cammini** da un vertice x ad un vertice y. Formalmente, un cammino è:

$$(v_0, ..., v_k) \quad \text{di} \quad V, \quad v_0 = x, \quad v_k = y \quad | \quad 1 \leq i \leq k : (v_{i-1}, v_i) \in E$$

Un cammino che torna da dove parte, ovvero dove $(v_0, ..., v_k) : v_0 = v_k$, è detto ciclico. Si dice che un grafo diretto privo di cicli è **aciclico**.

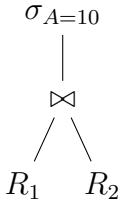
Alberi

Un grafo non orientato si dice connesso se esiste un cammino fra ogni coppia di vertici. Un'albero è un grafo non orientato nel quale due vertici qualsiasi sono connessi da uno e un solo cammino.

Un'interrogazione può essere rappresentata da un'albero, dove le foglie (i nodi finali) sono dati (relazioni, file, ecc...), e i nodi intermedi sono operatori (operatori algebrici, poi effettivi operatori di accesso ai dati). Ad esempio, l'albero corrispondente all'espressione:

$$\sigma_{A=10}(R_1 \bowtie R_2)$$

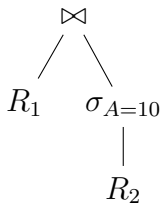
sarà:



La stessa interrogazione dopo il push down della selezione $\sigma_{A=10}$ sarà allora:

$$R_1 \bowtie \sigma A = 10(R_2)$$

con relativo albero:



Procedura euristica di ottimizzazione
La procedura di ottimizzazione consiste quindi nel:

- Decomporre le selezioni congiuntive in successive selezioni atomiche;
- Anticipare il più possibile le selezioni;
- In una sequenza di selezioni, anticipare le più selettive;
- Combinare prodotti cartesiani e selezioni per formare join;
- Anticipare il più possibile le proiezioni.

26 Relazioni Derivate

Una **relazione di base** è una relazione il cui contenuto è autonomo. Una **relazione derivata** è una relazione il cui contenuto è funzione di altre relazioni. Si possono cosi avere rappresentazioni diverse per gli stessi dati, definite mediante interrogazioni, che possono rivolgersi a relazioni di base come ad altre relazioni derivate. Esistono due tipi di relazioni derivate:

- **Viste materializzate**, dove i risultati sono precalcolati;
- **Viste virtuali**, o più semplicemente viste. Sono le più comuni.

Viste materializzate
Sono relazioni derivate memorizzate nella base di dati. Il loro vantaggio è che sono precalcolate, e quindi immediatamente disponibili. I loro svantaggi sono che:

- Sono ridondanti;
- Appesantiscono gli aggiornamenti;
- Sono raramente supportate dai DBMS.

Viste virtuali
Relazioni derivate non memorizzate nella base di dati, e quindi ricalcolate ad ogni accesso. Sono supportate da tutti i DBMS.

Interrogazioni su viste
Le interrogazioni su viste sono eseguite sostituendo alla vista la sua definizione. Le viste possono semplificare la scrittura di interrogazioni, espressioni complesse, e sotto-espressioni ripetute. L'uso delle viste virtuali inoltre non influisce sull'efficienza delle interrogazioni. Ad esempio, supponiamo di avere le seguenti relazioni:

$$R_1(ABC), R_2(DEF), R_3(GH)$$

e di definire una vista R:

$$R = \sigma_{A>D}(R_1 \bowtie R_2)$$

Un'interrogazione potrà a questo punto essere definita:

- Senza vista:
$$\sigma_{B=G}(\sigma_{A>D}(R_1 \bowtie R_2) \bowtie R_3)$$

  item Con vista:
$$\sigma_{B=G}(R \bowtie R_3)$$

Viste e aggiornamenti
Aggiornare una vista significa modificare le relazioni di base in modo che la vista ricalcolata rispecchi l'aggiornamento. L'aggiornamento sulle relazioni di base dovrebbe essere associato univocamente a quello specificato sulla vista, ma questo non è sempre vero. Per questo motivo ben pochi aggiornamenti sono ammissibili sulle viste.

27 Calcolo relazionale

Il calcolo relazionale è una famiglia di linguaggi dichiaratvi basati sul calcolo dei predicati del primo ordine. Ne esistono diverse versioni:

- Calcolo relazionale sui domini, in breve calcolo dei domini;
- Calcolo su n-uple con dichiarazione di range, in breve calcolo delle tuple, usato in SQL.

Assunzioni
I simboli di predicato corrispondono alle relazioni presenti nelle basi di dati, più alcuni predicati standard come l'uguaglianza e la disuguaglianza. Non compaiono simboli di funzione. Nel calcolo relazionale vengono utilizzate prevalentemente formule aperte, cioè formule con variabili libere, il cui valore di verità dipende dai valori assegnati alle variabili libere. Il risultato di un'interrogazione (formula aperta) è costituito da tutte le tuple di valori che sostituiti alle variabili libere, la rendono vera. In coerenza con quanto fatto in algebra relazionale, useremo una notazione non posizionale (attributi con nome, etichettati).

Calcolo dei domini
Vediamo innanzitutto la sintassi: nel calcolo dei domini un'espressone ha forma:

$$\{A_1 : x_1, ..., A_k : x_k \mid f\}$$

dove:

- $A_1, ..., A_k$ sono attributi distinti (che possono anche non comparire nello schema della base di dati).
- $x_1, ..., x_K$ sono variabili (che assumiamo essere distinte, non è necessario).
- $A_1 : x_1, ..., A_k : x_k$ è la *target list* (lista degli obiettivi) e descrive il risultato.
- f è una formula costruita a partire da formule atomiche utilizzando eventualmente i connettivi booleani e i quantificatori logici $\forall x$, $\exists x$, con x variabile (logica del primo ordine).

Il risultato di un'espressione nel calcolo dei domini è una relazione su $A_1, ..., A_k$, contenente n-uple in $x_1, ..., x_2$ che rendono vera la formula f sull'istanza di basi di dati considerata. F rispetto all'istanza

Formule atomiche del calcolo dei domini
Il predicato f è costituito da formule atomiche. Una formula atomica è definita come una delle tre forme:

- **Schemi di relazione**

  $$R(A_1 : x_1..., A_p : x_p)$$

  dove $R(A_1, ..., A_P)$ è uno schema di relazione.

- **Operatori fra variabili**

$$x_i \; OP \; x_j$$

  dove x e y sono variabili e OP è un operatore di confronto (uguale, non uguale, maggiore, minore e varianti strette).

- **Operatori fra variabili e costanti**

$$x_i \; OP \; c, \quad c \; OP \; x_i$$

  dove c è una costante nel dominio A_i di x_i.

Definiamo allora alcune proprietà delle formule:

- Le formule atomiche sono formule.
- Se f è una formula, allora anche $\neg f$ è una formula.
- Se f_1 e f_2 sono formule, allora anche $f_1 \wedge f_2$ e $f_1 \vee f_2$ sono formule.
- Se f è una formula e x una variabile, allora anche $\exists x(f)$ e $\forall x(f)$ sono formule, dove $\exists$ and $\forall$ sono qualificatori (rispettivamente esistenziale e universale).
- Per convenienza, si raggruppano le variabili usate da quantificatori simili: ad esempio, $\exists x(\exists y(f))$ si può scrivere come $\exists x, y(f)$.

Valore di verità

L'inclusione di una data n-upla o meno da un'espressione è determinato dal valore di verità della sua formula. Il valore di verità di una formula è così definito (sulle formule atomiche):

- Una formula atomica $R(A_1 : x_1, ..., A_p : x_p)$ è vera sui valori $x_1, ..., x_p$ che costituiscono una n-upla valida di R.
- Una formula atomica $x \; \theta \; y$ o $x \; \theta \; c$ (con c costante) è vera sui valori x, y che soddisfano la condizione θ.
- Il valore di verità degli operatori logici (congiunzione $\wedge$, disgiunzione $\vee$, negazione $\not)$è definito nel modo usuale.
- Una formula della forma $\exists x(f)$ è vera se esiste almeno un elemnto del dominio di x che rende vera f. Analogamente, una formula della forma $\forall x(f)$ è vera se tutti gli elementi del dominio di x rendono f.

Interpretazione logica del calcolo dei domini
Un'espressione del calcolo sui domini $\{A_1 : x_1, ..., A_k : x_k | f\}$ può essere sostanzialmente interpretata come una formula logica del tipo:

$$\{x_1, ..., x_k | f(x_1, ..., x_l)\}$$

dove $x_1, ..., x_k$ sono variabili o costanti, e $f(x_1, ..., x_k)$ è un predicato (vero o falso) che determina l'appartenenza o meno della n-upla di variabili al risultato dell'espressione.

Vantaggi e svantaggi del calcolo dei domini
Il vantaggio principale del calcolo dei domini è la dichiaritività: si *dichiarano* le proprietà del risultato desiderato, senza specificare i passaggi necessari a calcolarli. I difetti stanno invece nella grande verbosità (non è permesso fare proiezioni su stadi intermedi, ergo siamo costretti a portarci dietro tutti gli attributi nell'enunciato della formula). Inoltre, il calcolo dei domini è dipendente dal dominio (ovvero non tutte le espressioni sintatticamente valide nel calcolo dei domini sono indipendenti dal dominio):

Indipendenza dal dominio
Si dice che un'espressione di un linguaggio d'interrogazione è indipendente dal dominio quando il suo risultato non varia al variare del dominio rispetto alla quale l'espressione su cui viene valutata, salvo ovviamente i valori presenti nell'istanza e nell'espressione. A questo punto un linguaggio si dice indipendente dal dominio se tutte le sue espressioni sono tali. L'algebra relazionale è indipendente dal dominio: i risultati vengono costruiti sulla base di operazioni svolte sulle relazioni presenti nella base di dati senza mai fare riferimento ai domini dei valori che compongono le n-uple delle istanze. In altre parole, tutti i valori compaiono in istanze effettive di relazioni della base di dati o nella formulazione dell'espressione. Il calcolo dei domini, al contrario, è dipendente dal dominio. Si può ad esempio scrivere, senza violare nessuna regola sintattica:

$$\{A : x, B : y | R(A : x) \wedge y = y\}$$

nel risultato, il valore x dell'attributo A è legato alla relazione R, mentre il valore y può essere qualsiasi valore del dominio dell'attributo B, in quanto $\forall B : y | y = y$. Al variare del dominio di B, il risultato dell'espressione cambia (l'espressione è dipendente dal dominio!). Fosse stato B infinito, l'espressione avrebbe restituito un insieme infinito. Un'altro esempio può essere:

$$\{A : x | \neg R(A : x)\}$$

Tale espressione ottiene esattamente il complemento dei valori x sull'attributo A contenuti nella relazione R, ovvero tutti i valori del dominio di A meno

quelli in R.
Per ovviare ai problemi di dipendenza dal dominio presentati dal calcolo dei domini è stato introdotto il calcolo delle tuple.

Calcolo delle tuple
Il calcolo delle tuple risolve i problemi di verbosità del calcolo dei domini, utilizzando variabili per denotare tuple anzichè singoli valori legati ad attributi. Ogni relazione coinvolta presenta quindi una variabile, che rappresenta ogni possibile tupla di quella relazione. Per accedere ai singoli attributi di una tupla occorre quindi associare una struttura ad ogni variabile. Le espressioni nel calcolo delle tuple hanno forma:

$$\{T|L|f\}$$

dove:

- T è una **target list** (obiettivi dell'interrogazione);
- L è una **range list**;
- f è una formula.

Target list
Stabiliamo che:

- x è una variabile;
- X è un insieme di attributi di una relazione;
- Y e Z sono sottoinsiemi di attributi di X di pari lunghezza.

A questo punto, possiamo dire che T è una lista di elementi del tipo:

- $Y : x.Z$, Denotiamo con Y solo gli attributi Z della variabile (tupla) x, che assumerà valori definiti in L (*range list*).
- $x.Z \equiv Z : x.Z$ vogliamo solo gli attributi Z della variabile x, che assumerà valori definiti in L, ma non li ridenominiamo.
- $x.* \equiv X : x.X$ Prendiamo tutti gli attributi della variabile x, che assumerà valori definiti in L.

Range list
La *range list* L è una lista che contiene, senza ripetizioni, tutte le variabili della *target list*, con associata la relazione da cui viene prelevata ogni variabile:

$$L \equiv x_1(R_1), ..., x_k(R_k)$$

dove x_i è una variabile e R_i è una relazione. L è una dichiarazione di range, ovvero specifica l'insieme dei valori che possono essere assegnati alle variabili. Non occorrono, come occorrevano nel calcolo dei domini, condizioni atomiche che vincolano una tupla ad appartenere ad una relazione $(A_1 : x_1, ..., A_n : x_n)$.

Formule atomiche del calcolo delle tuple
Possiamo definire, come avevamo fatto per il calcolo dei domini, 2 formule atomiche (manca la condizione di vincolo d'appartenenza alla relazione):

- **Operatori fra variabili**

  $$x_i.A_1 \; OP \; x_j.A_j$$

  dove x e y sono variabili, A_1 e A_j attributi di tali variabili, e OP è un operatore di confronto (uguale, non uguale, maggiore, minore e varianti strette).

- **Operatori fra variabili e costanti**

  $$x_i.A_i \; OP \; c, \quad c \; OP \; x_i.A_i$$

  dove c è una costante nel dominio dell'attributo A_i di x_i.

Possiamo quindi definire alcune proprietà delle formule, quasi del tutto analoghe a quelle definite per il calcolo dei domini:

- Le formule atomiche sono formule.
- Se f è una formula, allora anche $\neg f$ è una formula.
- Se f_1 e f_2 sono formule, allora anche $f_1 \wedge f_2$ e $f_1 \vee f_2$ sono formule.
- Se f è una formula e x una variabile che indica una n-upla sulla relazione R, allora anche $\exists x R(f)$ e $\forall x R(f)$ sono formule, dove $\exists$ and $\forall$ sono qualificatori (rispettivamente esistenziale e universale). Notiamo che anche i quantificatori contengono adesso delle dichiarazioni di range(*range list*). La notazione $\exists(R)(f)$ significa "esiste nella relazione R una n-upla x che soddisfa la formula f".

Vantaggi e svantaggi del calcolo delle tuple Il vantaggio principale del calcolo delle tuple è la minore verbosità rispetto al calcolo dei domini. Svantaggiosa è invece l'effettiva impossibilità di esprimere alcune interrogazioni, in particolare l'unione:

$$R_1(AB) \cup R_2(AB)$$

Questo perchè ogni variabile nel risultato ha un solo range, mentre l'unione comporta la derivazione di n-uple da più relazioni distinte. Per questo motivo linguaggi quali l'SQL, definiscono un operatore esplicito di unione (l'UNION). Intersezione e differenza sono invece esprimibili.

Calcolo e algebra
Calcolo e algebra sono sostanzialmente equivalenti, ovvero ogni espressione dell'algebra (indipendente dal domino!) ha un equivalente nel calcolo e viceversa. Esistono però alcune interrogazioni non esprimibili:

- **Calcolo dei valori derivati:** abbiamo la possbilita di estrarre valori, mai di generarne di nuovi.
- **Ricorsività:** le interrogazioni non hanno la possibilità di esprimere ricorsività, come avevamo visto nell'esempio della chiusura transitiva (riguardante riferimenti potenzialmente infiniti fra relazioni).

28 Modellazione e progettazione concettuale

La definizione di schemi adeguati per le basi di dati richiede metodologie precise per la modellazione accurata della realtà che ci interessa. Bisogna tenere a mente che:

- Non conviene concetrarsi subito sui dettagli;
- Conviene invece stabilire subito interdipendenze fra relazioni;
- Il modello relazionale sarà rigido una volta ultimato.

In generale si può dire che esiste sempre (e spesso è unico) uno schema che modella accuratamente e nel modo più semplice possibile una certa realtà di interesse. La progettazione di basi di dati è solo una fase delloo sviluppo di un sistema informativo (vedere a riguardo le prime lezioni di teoria). Bisogna quindi tenere conto di:

- **Ciclo di vita** *(lifecycle)* del sistema informativo, ovvero l'insieme delle attivita svolte da analisti, progettisti e utenti nello sviluppo e nell'uso del sistema informativo. Questa attività è iterativa, e quindi ciclica.

I passi ciclo di vita dovranno essere ben definiti attraverso linguaggi e modelli prestabiliti. Per le basi di dati, in particolare, conviene adottare modelli di facile utilizzo, che consentano la decomposizione delle attività in fasi (e/o livelli) distinti, e di utilizzare strategie e criteri di scelta nei vari passaggi.

Modello a cascata
Il modello a cascata (*waterfall model*) le fasi sono sequenzialmente ordinate, etichettate, e non ripetibili. In ordine, esse sono le seguenti:

1. **Studio di fattibilità**: definizione di costi e priorità della produzione;
2. **Raccolta e analisi dei requisiti**: studio delle proprietà del sistema che andranno implementate;
3. **Progettazione**: progettazione di strutture dati e funzioni;
4. **Realizzazione**: implementazione effettiva del codice;
5. **Validazione e collaudo**: sperimentazione del prodotto;
6. **Funzionamento**: il sistema diventa opertaivo in produzione (*shipping*).

Vediamo alcune fasi nel dettaglio.

29 Raccolta e analisi dei requisti

Questa fase può essere, a sua volta, divisa in due sotto-fasi:

- **Acquisizione dei requisiti**: il reperimento dei requisiti è un'attività non standardizzata. Esistono più modalità:
  - Direttamente dagli utenti:
    - Interviste, focus group, recensioni, ecc...
    - Documentazioni apposite;
  - Attraverso documentazioni preesistenti:
    - Normative (legislazioni, regolamenti di settore);
    - Regolamenti interni, procedure aziendali;
    - Realizzazioni preesistenti.

  Esistono linguaggi per definire requisiti (*UML*).
- **Analisi dei requisiti**: si analizzano i requisiti raccolti, spesso nella prospettiva di successive acquisizioni.

Interazione con gli utenti
Le problematiche dell'interazione con gli utenti possono essere:

- Utenti diversi danno risposte diverse;
- Utenti a livello più alto hanno spesso una visione più ampia ma meno dettagliata;
- Spesso l'acquisizione di requisiti avviene per raffinazione.

Conviene quindi:

- Effettuare spesso verifiche di comprensione e coerenza;
- Verificare anche attraverso esempi (sopratutto nei casi limite);
- Richiedere definizioni e classificazioni chiare e specifiche;
- Separare gli aspetti essenziali da quelli marginali (*ranking*).

Interazione con gli utenti tramite documentazione
E' opportuno seguire alcune linee guide generali, assicurando:

- Standardizzazione della struttura delle frasi;
- Separazione delle frasi riguardanti dati da quelle riguardanti funzioni;
- Organizzazione di termini e concetti:
  - Unificazione di termini (eliminazione di sinonimi);
  - Esplicitazione del riferimento fra termini;
- Organizzazione delle frasi per concetti.

30 Progettazione

La progettazione è una fase del ciclo di vita. Per un sistema software la progettazione si divide effettivamente in:

- Progettazione dei dati;
- Progettazione delle applicazioni.

Progettazione per astrazione
Come in tutte le applicazioni informatiche, abbiamo visto che è necessario progettare per livelli successivi di astrazione:

- **Livello concettuale**: esprime i requisiti di un sistema in una descrizione adatta all'analisi da punti di vista esterni
- **Livello logico**: evidenzia l'organizzazione dei dati dal punto di vista del loro contenuto informativo, descrivendo la struttura dei record e le loro interdipendenze.
- **Livello fisico**: si concentra sulla base di dati vista come un insieme di blocchi fisici sul disco, e riguarda quindi l'allocazione dei dati e le modalità di memorizzazione.

Con riferimento a quanto detto sugli schemi logici avremo quindi che la progettazione si divide in:

- **Progettazione concettuale**, parte dai requisiti individuati della base e produce uno schema concettuale;
- **Progettazione logica**, parte dallo schema concettuale e produce uno schema logico;
- **Progettazione fisica**, parta dallo schema logico e produce lo schema fisico finale.

Come lo era stato il modello a cascata, la progettazione per astrazione è composta da fasi sequenzialmente ordinate che vanno eseguite strettamente in ordine.

Modello dei dati

Il modello dei dati è l'insieme dei costrutti utilizzati per organizzare i dati di interesse e definirne la dinamica. Componente fondamentale del modello sono i meccanismi di strutturazione (costruttori di tipi). Come per i linguaggi di programmazione comuni, esistono meccanismi che permettono di definire nuovi tipi. Ogni modello dei dati prevede alcuni costruttori: ad esempio il modello relazionale prevede un costruttore relazionale che permette di definire insiemi di record omogenei. Per riassumere in breve, in ogni base di dati si ha:

- Lo **schema**, invariante nel tempo, che ne descrive la struttura. Si notino inoltre le **intestazioni** delle tabelle (previste nel modello relazionale).
- L'**istanza**, i valori effettivi assunti dalla base di dati in un dato momento. Nel modello relazionale rappresenta il **corpo** di ciascuna tabella.

Modello concettuale Entity-Relationship

Il modello concettuale che utilizzeremo sarà quello **Entity-Relationship (ER)**. Si noti che la parola *relationship*, sebbene abbia lo stesso significato in lingua inglese, non si riferisce al concetto matematico di relazione che sta alla base del modello relazionale. Per questo motivo useremo sempre il termine inglese per descrivere le relazioni del modello entity-relationship, in modo da distinguerle dalle relazioni del modello relazionale. Il modello entity-relationship viene sviluppato da P.P. Chen nel 1976, ed è oggi una delle metodologie più affermate nel campo della progettazione dei sistemi informatici, anche se in un'accezione leggermente diversa da quella in cui era stato concepito inizialmmente.
I suoi costrutti base sono:

- Entità
- Relationship
- Attributi

Entità

Un'entità è una classe di oggetti dell'applicazione d'interesse con proprietà comuni e esistenza autonoma. Un'occorrenza (o istanza) di entità è un elemento della classe (un'elemento, non i dati ad esso legati!). Ogni entità deve avere un nome che la identifica univocamente nello schema. Graficamente è rappresentato da una scatola.

Relationship

Una relationship è un legame logico fra due o più entita, rilevante nell'applicazione d'interesse. Può essere chiamata anche relazione (vedi sopra), correlazione o associazione. Ogni relationship, come per le entità, ha un nome che la identifica univocamente nello schema. Graficamente è rappresentata da una losanga.
Vediamo allora di definire il concetto di occorrenza di relationship, più complesso di quello di occorrenza di entità:

- Un'occorrenza di **relationship binaria** è una coppia di occorrenza di entità, una per ciascuna entità coinvolta.
- Una occorrenza di una **relationship n-aria** è una n-upla di occorenze di entità, una per ciascuna delle n entità coinvolte.

Nell'ambito di una relationship non ci possono essere occorrenze (né coppie né n-uple) ripetute.

Attributo

L'attributo è una proprietà elementare di un'entità o di una relationship, che ci interessa ai fini dell'applicazione d'interesse. Associa a ogni occorrenza di entità o relationship un valore appartenente ad un dominio, il cosiddetto dominio dell'attributo.

Attributo composito

Gli attributi commpositi raggruppano attributi di una medesima entità o relatioship che presentano affinità nel loro significato o uso (e.g. giorno, mese, anno si compone in data, via, numero civico, CAP in indirizzo, ecc...).