Riassunto di aspetti temporali

# Quanto è importante la dimensione temporale?

La dimensione temporale nella business intelligence è importante dato che permette di analizzare l’evoluzione del business al fine di ottimizzare i database e fare delle previsioni.

La gestione del tempo è molto rilevante in questo ambito, è inoltre molto complicata e ha bisogno di metodologie specifiche indipendenti dal campo di utilizzo.

Proprio per questo motivo esiste un’area di ricerca che si occupa di risolvere tutti i problemi legati al tempo:

* la prima soluzione è quella di aggiungere attributi temporali nel caso in cui i fatti o gli eventi siano puntuali o durativi, tuttavia non basta dato che SQL classico non supporta il tempo. Inoltre vengono fuori tantissime problematiche generali.
* L’attuale approccio è quello di utilizzare estensioni degli approcci classici.

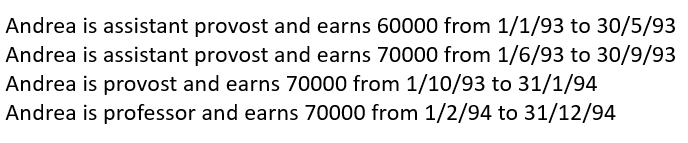
# Perchè gestire il tempo nei database non temporali è difficile?

Il tempo nei database non temporali è gestito in forma esplicita, di conseguenza il query answering permette la sola estrazione delle informazioni esplicitate nei dati.

Non esiste quindi un sistema di inferenza che permetta di estrarre informazioni implicite.

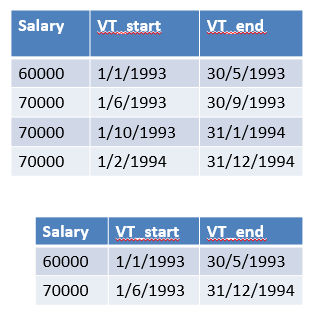
Un modo per gestire il tempo in questi database è quello di:

* inserire un attributo temporale per indicare i fatti puntuali, quelli che avvengono in un dato istante nel tempo;

inserire invece, due attributi per i fatti durativi, quelli validi solo nell’intervallo specificato.Esempio: storia del salario

Data la storia del salario e del titolo degli impiegati a destra, come si possono gestire questi dati?

Dal momento che grado e salario sono durativi, si possono aggiungere due attributi per indicarne inizio e fine.

Funziona? per verificarlo bisogna eseguire la seguente query:

Il risultato è quello che si può vedere nella tabella a destra, qui si ha

l’illusione di aver risolto il problema dal momento che le informazioni non vengono aggregate in base al salario.Se si vuole cercare qual è lo stipendio di Andrea, si utilizza la seguente query:

Il risultato che si vuole ottenere è quello nella tabella in basso a destra, SQL classico però non aggrega in modo implicito e quindi si ottiene la tabella in alto come risultato.

Il risultato voluto si può ottenere facendo aggregazioni in modo esplicito, quindi c’è bisogno di una query più complessa che le contiene.

## Problema della proiezione

Questo è un problema generale dal momento che bisogna fare una proiezione per togliere i dati che non interessano, quali sono?

Ebbene, queste tuple sono le cosiddette value-equivalent, in cui tutti gli attributi non temporali hanno gli stessi valori.

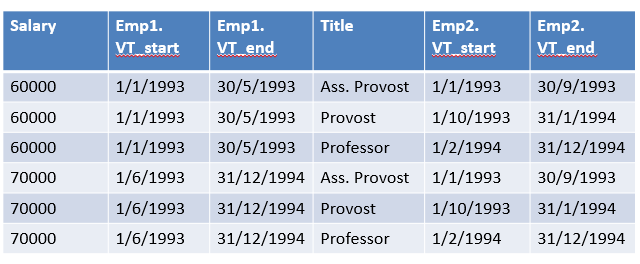
L’operazione da effettuare per unire tutte queste tuple è detta coalescing, un’operazione possibile anche con SQL classico ma in modo più complesso.

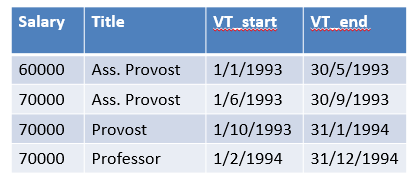
Come si risolve questo problema nei database non temporali? Si effettua il coalescing delle tuple sfruttando dei loop.

Per ottenere i risultati richiesti si cambia tecnologia, passando da SQL classico a TSQL2, il quale supporta il coalescing con normali query.

## Problema del join

Considerando il precedente esempio, come si comporta l’operazione di join?

Se si memorizzano salario e grado di un impiegato in due tabelle differenti, il join tra queste tabelle non dà un risultato con un significato, infatti si ottengono due coppie di tempi che tra loro non corrispondono.

Quello che si vuole ottenere è una tabella che contiene salario e grado degli impiegati rispetto al tempo (come si può vedere nella tabella in basso a destra), cosa possibile con SQL classico ma è oneroso da implementare dato che richiede i risultati passo per passo.

Il risultato ottenuto non è quindi consistente dal momento che i tempi non coincidono e alcune tuple non dovrebbero nemmeno esserci.

Per risolvere questo problema, c’è bisogno di controllare se c’è intersezione nei tempi tra ogni coppia di tuple e, nel caso, effettuare il coalescing, ciò però è esponenziale nel tempo. Si risolve tutto passando a TSQL2, il join sulle relazioni temporali involve nell’intersezione tra intervalli.

In generale, select e join non sono gli unici costrutti ridefiniti temporalmente, tutti i costrutti SQL sono stati ridefiniti a causa del tempo.

## Qual è il problema alla base?

Il problema è che nei database relazionali si assume che i valori degli attributi siano indipendenti tra loro, l’aggiunta della dimensione temporale aggiunge quindi una dipendenza che indica quando i valori degli attributi sono considerati veri.

Inoltre i database assumono che tutta la conoscenza che contengono sia esplicita, in questo caso l’aggiunta del tempo comporta delle assunzioni semantiche che non sono supportate dal modello, bisogna rendere in qualche modo esplicita la semantica del tempo.

Un base di dati non temporale è estensionale dato che i dati vengono tutti esplicitati, è invece detta intensionale quando è presente il tempo, quindi alcuni dati vengono espressi in forma implicita.

## Approcci ad-hoc

L’utilizzo di approcci ad-hoc per il trattamento di dati temporali è complesso e non vale il lavoro svolto, perché? Essenzialmente a causa della difficile gestione dei dati temporali e dall’enorme difficoltà dei problemi che possono venire fuori.

Inoltre i database relazionali sono espliciti dal momento che dicono tutte le informazioni, quelli temporali sono invece impliciti, quindi c’è bisogno di meccanismi di inferenza nel query answering per permettere certe proprietà non presenti nel database.

# Come possono essere i tempi?

I tempi nelle basi di dati temporali sono:

* il tempo di validità, indicante il periodo di tempo per cui un fatto è valido;
* il tempo di transazione, cioè il tempo di inserimento di una tupla nel database, esso non deve per forza coincidere col tempo di validità dal momento che si possono inserire tuple che avranno valore in futuro. Il tempo di transazione è importante dato che indica la disponibilità dei dati;
* il tempo user-defined, questo però non viene gestito dal momento che serve a modellare degli attributi come ad esempio la data di nascita, infatti non vengono generate dipendenze tra valori.

Un’applicazione utile del tempo di validità è l'esame del sangue, infatti l’intervallo preciso che denota permette di capire quali sono le condizioni del sangue del paziente in un dato momento.

Tempo di validità e di transazione sono tra loro ortogonali, cosa vuol dire ciò? Vuol dire che i due tempi devono essere trattati in modo indipendente l’un l’altro.

# Tipi di database

Dalle definizioni di tempo di validità e transazione, si possono definire quattro tipi di database:

* lo snapshot è quello che non gestisce i tempi;
* il rollback utilizza solo il tempo di transazione;
* l’historical gestisce solamente il tempo di validità;
* il temporal, li gestisce entrambi.

## Snapshot

Nel database snapshot non sono presenti nè attributi temporali, nè tantomeno la storicizzazione, ciò corrisponde a una classica tabella.

Il contenuto di questo database dà una panoramica di ciò che è vero in questo istante ma non quello di informazioni passate. Proprio per questo motivo le modifiche agiscono solo sulla versione corrente (che è anche l’unica).

Nello snapshot non sono possibili:

* cambiamenti proattivi, cioè l’inserimento o la modifica di dati che saranno validi in futuro;

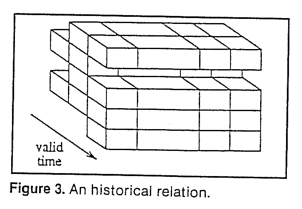
cambiamenti retroattivi, l’inserimento o la modifica di dati che erano validi in passato.

## Rollback

Il database rollback è un’evoluzione dello snapshot che permette la storicizzazione grazie all’aggiunta del tempo di transizione. Come nello snapshot, le modifica avvengono sull’ultima versione del database, generando un nuovo database con le informazioni aggiornate.

Grazie a ciò, è possibile effettuare delle query sulle informazioni passate. Su questo tipo di database non sono possibile cambiamenti proattivi e retroattivi nè tantomeno le query su ciò è vero nel presente, non si tiene traccia in alcun modo di ciò che vale nel mondo reale.

## Historical

Il database historical permette di gestire il tempo di validità e ciò lo rende molto simile alla visione multidimensionale dei datawarehouse.

L’utilizzo del tempo di validità permette di capire quando i fatti risultano validi nella realtà, inoltre permette anche cambiamenti proattivi e retroattivi.

A differenza del database rollback non sono possibili query sullo stato passato o futuro dei dati a causa della mancanza del tempo di validità.

## Temporal

Unendo le proprietà di un database rollback e un historical si ottiene un database temporal, il quale permette la gestione sia del tempo di validità, sia di quello di transizione.

L’utilizzo di entrambi i tempi permette di ottenere sia la storia dei dati nella realtà modellata, sia quella delle varie versioni precedenti.

Inoltre sono possibili, sia cambiamenti proattivi e retroattivi, sia questi sullo stato passato del database.Come si può vedere dalla foto, le tuple del database temporal sono stretchate lungo il tempo di transazione, quindi esistono differenti versioni del tempo di validità.

## Esempio: gestione del grado e del salario degli impiegati di un’università

### Inserimento dati

| Database | Come funziona |
| --- | --- |
| Snapshot | Memorizza le nuove informazioni. |
| Rollback | Memorizza le nuove informazioni aggiungendo la data di inserimento, questo approccio è corretto a meno di cambiamento pro e/o retroattivi. |
| Historical | Memorizza le nuove informazioni inserendo la data di inizio ma non quella di fine. |
| Temporal | Memorizza le nuove informazioni inserendo anche la data di inserimento e quella di inizio. |

### Aggiornamento dati

| Database | Come funziona |
| --- | --- |
| Snapshot | Cancella la vecchia tupla e inserisce quella nuova |
| Rollback | Aggiunge la tupla nuova (con la data di inserimento) mantenendo anche quella vecchia. |
| Historical | Aggiorna la vecchia query aggiungendo la data di fine e inserisce le informazioni nuove. |
| Temporal | Aggiorna la vecchia query aggiungendo la data di fine e inserisce le informazioni nuove (insieme alla data di inserimento). |

### Correzione dati

| Database | Come funziona |
| --- | --- |
| Snapshot | Cancella la tupla vecchia e inserisce quella nuova. |
| Rollback | aggiunge le informazioni nuove con data di inserimento, mantenendo quelle vecchie (ciò può portare a problemi). |
| Historical | Cancella la tupla errata e inserisce quella corretta. |
| Temporal | Viene creata una nuova versione contenente l’informazione corretta (senza quella errata). |

### Cambiamento retroattivo

| Database | Come funziona |
| --- | --- |
| Snapshot | Non si fa niente dato che non si può |
| Rollback | Si aggiunge l’informazione con la data di inserimento. |
| Historical | Si modifica la tupla in questione. |
| Temporal | Viene creata una nuova versione con l’informazione modificata. |

E’ importante sapere che, se una sorgente è storicizzata, al datawarehouse interessa la validità degli eventi, mentre alle dimension table serve anche il tempo di transazione.

Con le sorgenti non storicizzate invece, si approssima il tempo di validità a partire da quello di transazione, questa cosa è di per sé sbagliata ma non ci sono altre alternative.

Inoltre si assume che il tempo di inizio coincida con quello di validità.

# TSQL2

TSQL2 è un linguaggio per query relazionale conforme con gli attuali standard, inoltre è in grado di gestire sia tabelle temporali, sia quelle non temporali e presenta un algebra associata in grado di dimostrarne la validità e il funzionamento.

Attualmente TSQL2 rappresenta la soluzione che risolve in larga parte i problemi riguardanti i tempi di validità e transazione dei database relazionali.

TSQL2 punta a supportare:

* gli intervalli temporali, definiti come coppie di istanti;
* le granularità multiple, sia standard, sia user-defined;
* linguaggi e rappresentazioni sintattiche multiple, dall’alfabeto standard a quello cirillico a quello arabo, eccetera;
* i calendari multipli, dal gregoriano al cinese oppure il calendario accademico;
* l’indeterminatezza temporale, cioè la definizione di elementi temporali che non sono associati a un momento preciso del tempo. Ciò è correlato sia alla granularità temporale, sia al fatto che è impossibile misurare il tempo in modo preciso;
* tempo di validità e transazione.

## Obiettivi desiderati

* equivalenza semantica e identità devono essere sinonimi in assenza di informazioni implicite;
* rendere il tempo di validità un’unica dimensione dato che ogni fatto ne ha diversi associati;
* omogeneità dei timestamp su tuple e query;
* TSQL2 dovrebbe comportarsi come un’estensione di SQL92 compatibile verso l’alto;
* l’implementazione della prima forma normale;
* rendere il supporto temporale flessibile;
* essere implementabile attraverso un algebra.

## Come gestisce il tempo TSQL2?

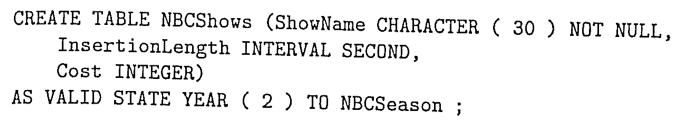
Per gestire le diverse epoche, TSQL2 le allinea tutte su una linea suddivisa in unità base dette chronon, tutti i timestamp verranno interpretati in base a essa.

TSQL2 supporta relazioni di differenti tipi:

* non temporali, i quali non gestiscono il tempo;
* relazioni di eventi, le quali gestiscono tempi di validità istantanei;
* relazioni di stato in cui il tempo è durativo su fatti o eventi;
* relazioni transazionali che utilizzano il tempo di transazione (che è durativo);
* relazioni tra eventi e transizioni;
* relazioni tra stati e transizioni.

Ogni tabella ha bisogno di specificare il tipo per essere creata, la tabella di default è quella non temporale per compatibilità con SQL.

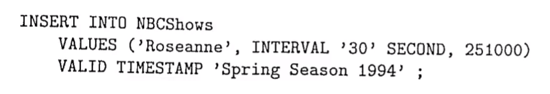
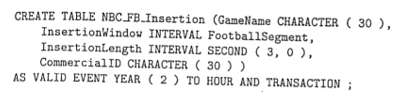
## Query in TSQL2

Le query in TSQL2 sono estensioni di quelle in SQL classico, infatti l’inserimento e tutte le altre operazioni sono praticamente identiche.

Considerando la query a destra:

* viene creata una tabella NBCShows;
* Questa tabella è formata da un nome non nullo di 30 caratteri al massimo, un tempo user-defined in secondi indicante la durata dell’inserzione e un costo espresso con un numero intero;
* AS VALID STATE indica che la relazione è di stato e quindi il tempo di validità viene gestito in modo durativo;
* YEAR(2) TO NBCSeason indica che la granularità scelta va dagli anni a quella di NBCSeason, quest’ultima è user-defined.

### Altri esempi di query

Ora si considera la query a destra, in questo caso si inserisce nella tabella NBCShows una tupla contenente Roseanne, un costo di 251000 e un intervallo di 30 secondi. Spring Season 1994 è un timestamp definito dall’utente.

L’ultima query a destra riguarda la creazione della tabella NBC\_FB\_Insertion:

* In questa tabella è presente un nome e un id formati entrambi da 30 caratteri e due intervalli, il primo di questi è user-defined e definito dall’utente FootballSegment, il secondo invece è in secondi;
* La tabella è VALID EVENT AND TRANSATION, quindi gestisce tempi di validità istantanei e di transizione;
* YEAR(2) TO HOUR indica che la granularità va dagli anni alle ore.

## Clausola From di TSQL2

La clausola from in TSQL2 è quella che avuto la maggior estensione, essa viene gestita tenendo conto del coalescing tra tuple.

Per spiegare bene il concetto, si considera la tabella a destra: qui si possono notare tuple value-equivalent da giustificare un coalescing, ciò si può fare:

* su intervalli non continui, unendo tutti gli intervalli tra loro continui e omettendo buchi, come visibile nella prima tupla;
* su intervalli continui, in questo caso i buchi non sono ammessi, per farlo si specifica con la keyword PERIOD. Come risultato si ottengono tante tuple quanti sono gli intervalli continui. Questo tipo di coalescing permette di ottenere intervalli continuativi e quindi la loro durata;
* in generale, il coalescing sulle tuple viene effettuato specificando la tabella e gli attributi, nel caso di “From Employee(Name,Salary), si collassano tutte le tuple value-equivalent per gli attributi Name e Salary. In base ai primi due punti, il coalescing collasserà le tuple value-equivalent nel modo corretto.

### Esempi di query con la clausola from

Considerando la query a destra, il coalescing viene effettuato per nome e salario senza considerare eventuali buchi. Dopodichè si cambia la granularità della tabella mettendola in giorni e si prendono tutte le tuple con un intervallo di almeno 25 giorni e con un salario maggiore di 100. Infine da tutte le tuple prese, si estrae il nome.

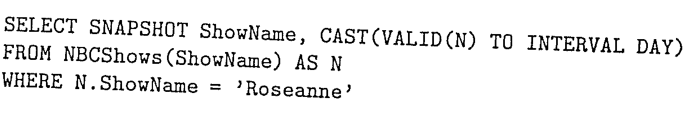
La seconda query si comporta più o meno allo stesso modo solo che qui è presente PERIOD, di conseguenza si tiene conto dei buchi tra gli intervalli.

## Esempio: relazione tra show e pubblicità

Nella query a destra è presente un cross join di una tabella con sè stessa in cui vengono selezionate solamente le query con lo stesso nome.

La clausola where messa in questo modo effettua anche le seguenti operazioni:

* recupero del tempo di transizione corrente dato che non viene specificato nella clausola;
* valid time selection implicita, cioè che le tuple di N e N2 devono essere valide allo stesso tempo ( VALID(N) = VALID(N2) );
* valid time projection implicita, in questo caso i tempi di validità delle tuple saranno l’intersezione di quelli di N e N2 ( VALID INTERSECT(N,N2) ).

E se si vuole sapere quanto dura “Roseanne”?

Per fare ciò si considera la query a destra, qui precisamente:

* si fa coalescing per ShowName;
* da questa tabella si prendono tutte le tuple aventi Roseanne come ShowName;
* da quest’ultime si ricava la durata in giorni (INTERVAL DAY) attraverso i tempi di validità;
* La keywork SNAPSHOT indica che la tabella in output non deve contenere intervalli di validità, anche se quella di input è una tabella temporale.

## E per considerare tutti gli show di NBC andati in onda per due anni in modo continuativo?

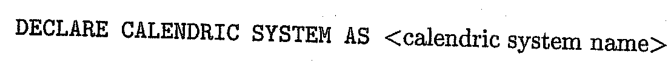
In casi come questi c’è bisogno della query a destra, infatti:

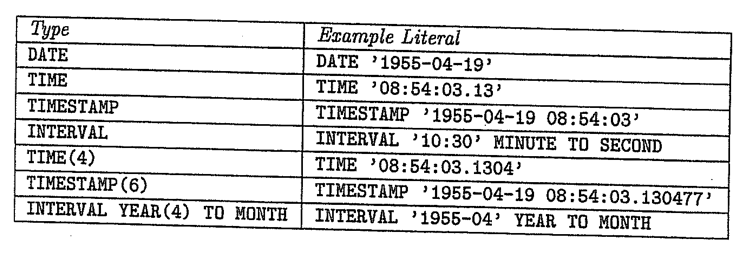
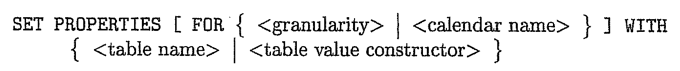
* si fa coalescing delle tuple value-equivalenti di NBCShows per ShowName tenendo conto dei buchi tra intervalli (terza riga);
* si converte in tempo di validità in un intervalli in anni ( CAST(...) );
* si prendono tutte le tuple con una durata di almeno due anni (WHERE …);
* di tutte le tuple si estraggono il nome e il giorno di inizio, quest’ultima si ottiene tramite una valid time projection a partire dal tempo di inizio validità delle tabella A (prima e seconda riga).

## Gestione dei calendari multipli

I calendari multipli permettono una gestione del tempo più flessibile, infatti utilizzare un solo calendario non basta, ne serve uno per ogni epoca per interpretare tutto al meglio.

TSQL2 permette di specificare questo sistema di calendari, associando a ogni epoca il calendario corretto in una data regione geografica.

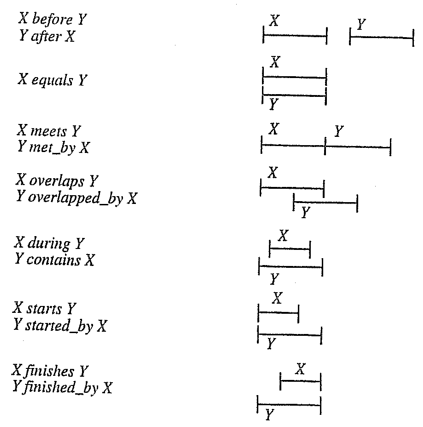
Tutto ciò è necessario dal momento che ogni epoca ha il suo calendario, ognuno con le proprie granularità, permettendo quindi di ottenere la semantica del tempo più completa possibile.

TSQL2 permette di dichiarare nuovi calendari utilizzando DECLARE, inoltre con WITH e SET è possibile sovrascrivere impostazioni e proprietà in modo da definirne di più specifici.

## Gestione dei periodi

La gestione dei periodi è una parte aggiuntiva di TSQL2 che viene gestita in maniera esplicita, il tutto attraverso delle apposite primitive e delle operazione con cui è possibile fare dei calcoli.

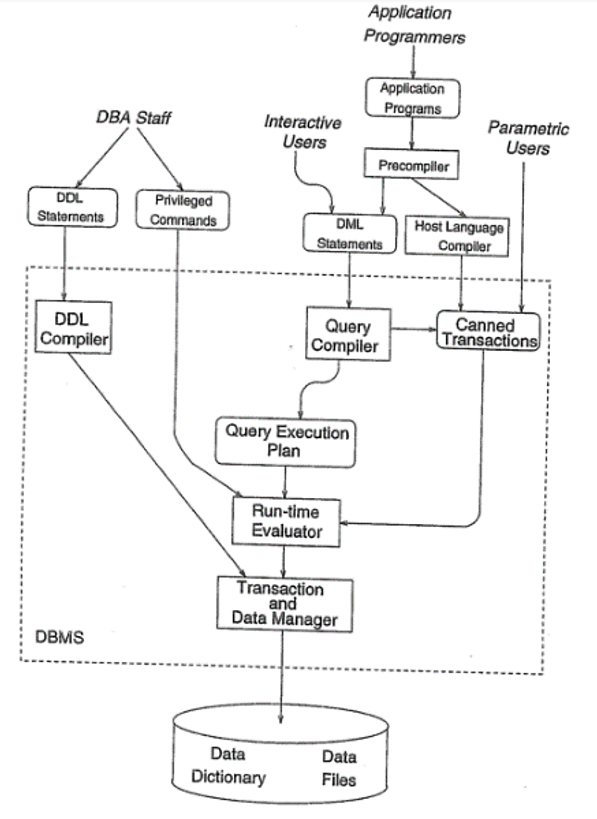
E’ inoltre possibile definire date e periodi in uno specifico fuso orario, questi vengono memorizzati insieme alla data stessa.

Per compatibilità con SQL classico, le date senza fuso orario vengono interpretate considerando il fuso orario locale.

Un’altra aggiunta è il tempo qualitativo, cioè quello definito in relazione ad altri elementi temporali. Queste relazioni avvengono tra coppie di intervalli, infatti esiste un’apposita algebra che permette di gestire ogni combinazione:

* precedenza: quando un intervallo precede l’altro;
* overlap: contenimento parziale tra intervalli;
* contenimento: un intervallo contenuto in un altro;
* incontro: un intervallo inizia quando il precedente termina.

## Architettura

L'architettura di TSQL2 permette la distinzione dei vari utenti e dell’accesso alle risorse, infatti ogni utente accede a un componente diverso in base alla sua esperienza.

Data una query scritta da un Interactive User, questa viene compilata da un query compiler, esso funziona in due step:

* un’analisi della query in maniera prima sintattica e poi semantica, come ad esempio il type checking;
* la compilazione della query algebrica corrispondente.

Al passo successivo, la query algebrica viene ottimizzata da un query optimizer, restituendo il cosiddetto piano di esecuzione, in essa è specificato l'algoritmo scelto per l’esecuzione.

Il piano di esecuzione viene poi dato in input a un run-time evaluator che esegue la query attraverso il transaction manager, recuperando i dati contenuti nei data files e nei data dictionary e garantendo la consistenza in caso di query multiple.

## TSQL2 come un’estensione di SQL

TSQL2 vuole essere una sorta di framework minimo di SQL92 per poter minimizzare il più possibile il lavoro.

Considerando la foto dell’architettura, nella parte bassa ci sono:

* data files, cioè i dati non temporali e i timestamp;
* data dictionary, utili per contenere le informazioni riguardanti i tipi temporali delle relazioni, le granularità o l’indeterminatezza temporale. Questi ultimi sono stati aggiunti per ovviare alla gestione del tempo in TSQL2.

### Estensione del DDL Compiler

Dal DDL Compiler è necessario estendere le primitive di creazione per garantire supporto alle tabelle temporali e le loro permutazioni, garantendo l’inizializzazione corretta degli attributi temporali aggiuntivi.

Le operazioni di ADD, REPLACE e DROP vengono anch’esse estese per garantire rispettivamente la modifica degli schemi e la rimozione di indica, tabelle e/o dati.

### Estensione del Query Compiler

Il query compiler è quello ha avuto la maggior estensione, infatti:

* la sintassi base di SQL viene estesa inserendo la semantica del tempo;
* vengono definiti gli algoritmi temporali, come ad esempio coalescing ed intersezione temporale;
* di tutte le operazioni vengono definite delle regole di ottimizzazione.

### Estensione del Run-time evaluator e del transaction manager

In queste parti i cambiamenti sono limitati, l’unica cosa è che il tempo di transazione deve essere gestito in modo automatico rispetto al commit di transazione. Ciò è dovuto al fatto che i dati non risultano disponibili fino al commit, quindi il tempo di transizione deve rimanere invariato.

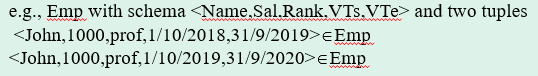
### Realizzazione effettiva

Ci sono due modi per effettuare queste strategie:

* Convertire le query temporali TSQL2 in SQL con una sintassi equivalente, ciò comporta a introdurre un nuovo strato di traduzione sopra il DBMS che permetta ciò;
* Costruire uno strato temporale sopra al DBMS che gestisce tutta la parte temporale, tutto il resto viene sempre gestito dal DBMS.

# Semantica del tempo nei database relazionali

Come si fa a definire la semantica del tempo? Nel contesto dei database, tutti gli approcci trattano il tempo allo stesso modo.

Per i database vale l’ipotesi del mondo chiuso, cioè che tutto ciò che è al suo interno viene considerato vero, per questo motivo non c’è bisogno di inferenza dato che i dati sono tutti espliciti.

Il tempo in questo caso ha cambiato qualcosa? Sì, infatti la sua introduzione rende meno questa regola, quindi viene dell’informazione implicita dato che questa semantica non è supportata nel modello relazionale.

L’idea generale è quella di compattare il più possibile l’informazione senza cambiarla,come si fa? Cosa significa aggiungere il tempo?

## Bitemporal Conceptual Data Model (BCDM)

BCDM è un modello di dati che gestisce tempo di validità e transazione (Bitemporal), esso non ha a che fare coi modelli concettuali bensì con la semantica.

Questo modello è il più generale ed è utilizzato non solo per la semantica di TSQL2 ma per la maggior parte degli approcci relazionali temporali.

Qual è il suo obiettivo? quello di definire una semantica per il tempo di validità e transizione senza considerare l’efficienza delle implementazioni pratiche.

BCDM utilizza come oggetti base i tempi, essi hanno le seguenti proprietà:

* linearità e totalmente ordinato: si sa quali informazioni vengono prima di altre e il modo in cui crescono;
* L’unità base è il chronon, ciò permette di gestire la crescita del tempo;
* Il dominio del tempo è inoltre finito, discreto e isomorfo su numeri naturali, ciò è dovuto al fatto che non ha senso andare troppo indietro nel tempo.

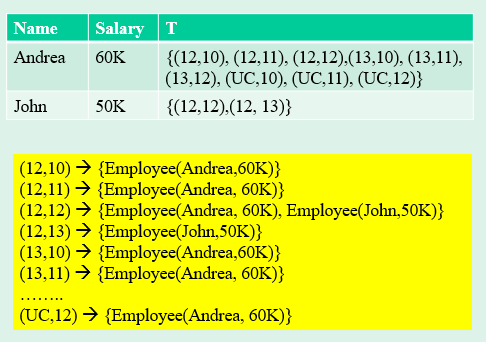
Dal punto di vista dei dati, essi hanno attributi aventi nomi e dominio, lo schema bitemporale raccoglie tutti i nomi e a questi aggiunge i tempi, stessa cosa vale per il dominio bitemporale.

Una tupla è quindi un’assegnazione di un valore a ogni attributo.

I domini del tempo sono validità e transazione, nulla vieta di renderli uguali, tuttavia quest’ultimo non può avere tempi nel futuro, quindi come si gestiscono casi del genere? Si utilizza un elemento segnaposto detto UC.

## Esempio: salario degli impiegati

Andrea ha un salario di 60K valido nei tempo 10, 11 e 12, quindi, se si è a tempo 12, sono state inserite delle tuple in cui Andrea è impiegato ai tempi di validità e transazione dati, come visibile in figura.

Tutte le tuple in cui Andrea è impiegato è ha un stipendio di 60K sono value-equivalent, esse vengono scritte una sola volta nel database, aggiornando adeguatamente i tempi.

Si fa così perchè non avrebbe senso scrivere più volte la stessa informazione, inoltre si garantisce l’unicità della rappresentazione.

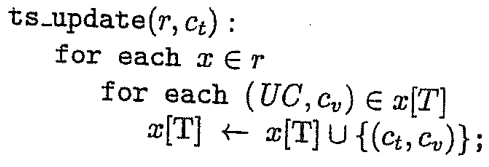
Le query in BCDM hanno la stessa parte dati mentre in quella temporale ci sono tutti i tempi per cui è valida.

BCDM rappresenta tutto evidenziando i fatti e collegandoli ai tempi, esiste anche una rappresentazione più astratta che fa tutto il contrario, rappresentando meglio un database temporale.

Quest’ultima rappresentazione cosa permette di concludere? Che un database relazionale temporale è un insieme di database relazionali, uno per ogni tempo

SI può quindi considerare BCDM come un’estensione dei database standard.

## Cosa significa UC?

UC è un valore cha fa da prestanome a tempi futuri o in generale sconosciuti, l’idea è quella di non togliere la tupla ma di aggiungere tempi di transizione ogni volta.

Nel codice a destra, Ct indica un momento del tempo di transazione mentre r è la tabella:

* Quel che si fa è prendere tutte le tuple x in r a tempo Ct;
* di esse si considerano quelle con UC come tempo di transazione;
* si aggiunge un tempo (Ct,Cv) alla tupla;
* infine si aggiunge (UC,Cv).

## Cancellazione di dati

La cancellazione di una tupla in BCDM avviene togliendo i tempi di transazione per una data tupla. Se una tupla non presenta UC, allora non si fa nulla.

## Inserimento di dati

Dal momento che BDCM non prevede tuple value-equivalent, l’inserimento può avvenire in vari modi

* se l’informazione non è presente, si aggiunge una nuova tupla;
* se invece è presente, si controlla se la tupla ha UC, in tal caso si aggiorna l’informazione;
* se invece non c’è, non si fa nulla.

## Aggiornamento di dati

L’aggiornamento dei dati in BCDM modifica le tuple sui tempi di transazione, precisamente effettua prima una delete e poi una insert coi valori aggiornati.

# Implementazione di BCDM: TSQL2

BCDM ha una visione molto chiara ma inefficiente per le seguenti ragioni:

* le query non sono in prima forma normale, di conseguenza una tupla può avere un numero di chronon arbitrario, quindi la dimensione delle tuple è variabile;
* bisogna gestire gli UC in ogni istante, un’operazione non realizzabile.

L’obiettivo è ottenere un’implementazione efficiente che garantisca unicità nella rappresentazione, come fare?

Si associa un valore a ogni attributo temporale nelle tuple bitemporali.

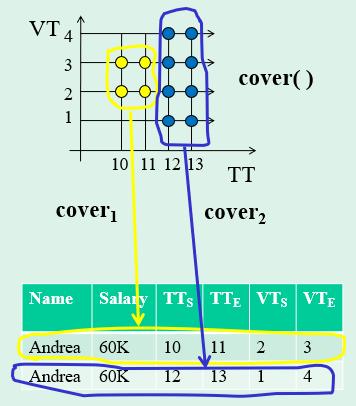
TSQL2 è un’implementazione value-equivalent più compatta e che permette di capire i vari temporal chronon, ciò permette di ottenere la semantica e da qui BCDM.

Un esempio di implementazione TSQL2 implica l’utilizzo di quattro attributi temporali (due per ciascun tempo), in essi si utilizzano timestamp o UC come valori.

Inoltre,TSQL2 permette tuple value-equivalent!

## Come si può ottenere uno schema BCDM da TSQL2 e viceversa?

Esistono due metodi per fare ciò:

* Snap to conceptual: permette di convertire la rappresentazione da TSQL2 a BCDM, come fa? data una tabella r, si prende ogni tupla z e da esse si estraggono i temporal chronon con la funzione bi\_chr(). Dopodichè si crea una tupla x avente la stessa parte non temporale e l’unione di tutti i temporal chronon come parte temporale.

In poche parole, tutte le tuple z vengono collassate nella tupla x unendo tutti i tempi.

* Conceptual to snap: Questa funzione fa esattamente il contrario di quella precedente. Data una tabella r’ e una funzione di cover (utile per convertire i temporal chronon in un intervallo), per ogni tupla x in r’ creo tante tuple z value-equivalent quanti sono gli intervalli prodotti dalla funzione di cover, assegnando i tempi giusti agli attributi temporali corretti.

Si può notare come nella foto a destra che le informazioni vengono memorizzate in più tuple value-equivalent, questo perchè il coalescing può essere fatto solo sui rettangoli. Ciò indica l’importanza delle tuple value-equivalent e perchè non è possibile farne a meno. Al costo dell’unicità della rappresentazione, si è riusciti a ottenere un’implementazione efficiente.

## Inserimento

L’inserimento prende in input una tabella, una funzione di cover e una tupla per inserire quest’ultima, come fa a farlo? Utilizzando la funzione di cover sul tempo di validità, si generano degli intervalli, per ognuno di essi si generano tante tuple value-equivalent.

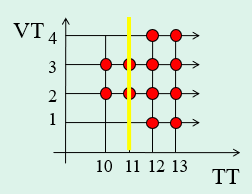
## Cancellazione

La cancellazione va a cercare nella tabella r tutte le tuple uguali a quella data, di queste si considerano solamente quelle con UC come tempo di transizione finale, sostituendo quest’ultimo col tempo ct dato in input.

## E’ davvero un problema l’assenza dell’unicità della rappresentazione?

L’assenza dell’unicità della rappresentazione è un grosso problema dal momento che si hanno modi differenti per indicare la stessa cosa.

Infatti, considerando il grafico a destra, si possono ottenere differenti tuple in base a come viene effettuato il coalescing.

Esiste una soluzione?

Sì, si può infatti dire che due rappresentazioni sono snapshot equivalent se sono identiche in ogni chronon bitemporale.

## Operatori di Time slice

Il primo operatore di time slice riguarda il tempo di transizione, esso funziona come segue:

* in base a una tabella r su cui agisce un tempo di transazione t1, si restituisce una tabella le cui tuple hanno t1 come tempo di transizione, facendolo poi sparire;
* Precisamente, per ogni tupla, si restituiscono delle tuple z tali che deve esistere una tupla x in r con la stessa parte dati e contiene solo chronon nella forma (t1,t2) (in cui t1 è il tempo di transizione e t2 quello di validità).
* Una cosa molto importante è che l’insieme dei tempi di validità della tupla z non deve essere vuoto.



Il secondo operatore di time slice riguarda invece il tempo di validità e funziona praticamente allo stesso modo dell’operazione precedente, solo che viene tagliato il tempo di validità.

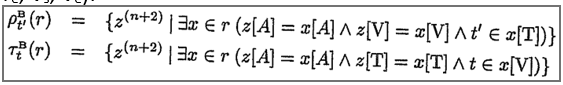


Cosa si ottiene se si applicano queste operazioni in modo annidato? Ebbene, si ottengono tutte le tuple che possiedono (t1,t2) come chronon temporale, tutte queste non presentano la parte temporale e valgono in un punto specifico del tempo.

Cosa si può concludere? Si conclude che due tabelle r e s sono snapshot equivalent se, applicando a entrambe gli operatori di time slice, si ottiene lo stesso insieme di tuple.

Questi due operatori possono essere definiti in TSQL2 considerando i quattro attributi temporali, precisamente:

* il time slice di transazione permette di ottenere un insieme di tuple z con n attributi value e due temporali di validità. L’algoritmo va a controllare se esiste una tupla x avente la stessa parte value di z, gli stessi tempo di validità e t’ come tempo di transazione.
* il time slice di validità effettua la stessa operazione ma sul tempo di validità.



## Snapshot equivalence in TSQL2

* Date due tabelle r e s concettuali e con lo stesso schema, r e s sono snapshot equivalent se sono uguali;
* r e s sono snapshot equivalent se, data una relazione concettuale rt, C2S(rt, coverR)=r e C2S(rt,coverS)=s dove C2S indica la funzione conceptual to snap;
* se r e s sono snapshot equivalent, allora S2C(r)=S2C(s) dove S2C indica la funzione snap to conceptual.

# Algebra e semantica delle query BCDM

Per implementare l’algebra delle query BCDM, si vogliono preservare le seguenti proprietà:

* preservare la snapshot equivalence in ogni operatore, ottenendo un oggetto snapshot equivalent da due input che lo sono;
* essere un’estensione consistente dell’algebra non temporale per garantire interoperabilità, l’unico modo attuale per farlo è estendere in modo consistente l’algebra di Codd (quella tradizionale). Questo obiettivo si può raggiungere attraverso le dimostrazioni di riducibilità e di equivalenza.

L’obiettivo è quello di garantire coerenza con la semantica dei dati temporali, inoltre si vuole fare in modo che l’algebra BCDM si comporti come quella di Codd, infatti gli operatori:

* coi dati non temporali devono comportarsi come nell’algebra di Codd;
* coi dati temporali di rispettare una semantica temporale.

Questo è un importante punto di vista pratico per garantire l’interoperabilità.

## Selezione non temporale

La selezione non temporale non considera i dati temporali, questa operazione è quindi già definita nell’algebra di Codd, precisamente si prendono tutte le tuple z nella tabella r che rispettano il predicato P dato in input. Bisognerebbe inoltre definire nuovi operatori di selezione temporale, essendo predicati nuovi però, non è necessario dimostrare riducibilità ed equivalenza.

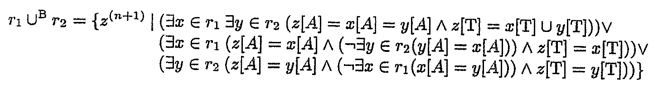


## Unione

L’unione tra due tabelle r1 e r2 e crea una terza tabella che comprende tutte le tuple z tali che esistono due tuple x e y (rispettivamente in r1 e r2) con la stessa parte dati e l’unione delle parti temporali.

Ci sono poi due casi in cui le tuple sono presenti in una tabella ma non nell’altra, qui la tupla z sarà uguale alla tupla esistente (sia nella parte dati, sia in quella temporale).

In questa operazione si dà per scontato che non ci possono essere più di due tuple value-equivalent, averne di più significherebbe andare contro la definizione di BCDM.

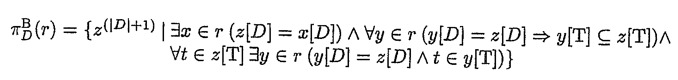


## Proiezione

La proiezione può generare un numero arbitrario di tuple value-equivalent, la situazione quidi si fa complicata.

Considerando D come l’insieme di attributi da considerare, la proiezione genera un insieme di tuple z con |D|+1 attributi, qui devono vale le seguenti proprietà:

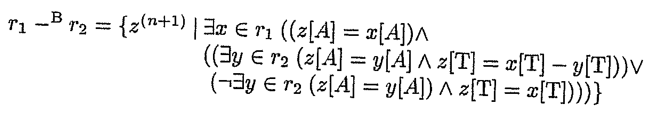
* deve esistere una tupla x con la stessa parte dati di z;
* per ogni tupla y con la stessa parte dati, la parte temporale di y deve essere contenuto in quella di z;
* per ogni tempo t nella parte temporale di z, esiste una tupla y con la stessa parte temporale e t come parte temporale.



## Differenza

L’operazione di differenza tra due tabelle restituisce un insieme di tuple z che soddisfano i seguenti requisiti:

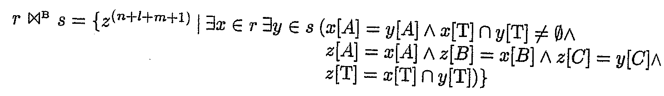
* le tuple z devono avere la stessa parte dati di una tupla x esistente in r1;
* le tuple z devono avere la stessa parte dati di una tupla y esistente in r2, inoltre la parte temporale di z deve essere la differenza di quelle di x e y;
* se esiste una tupla y con la stessa parte dati di z, z e x non possono avere la stessa parte temporale.

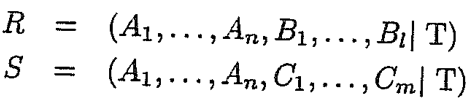


## Join naturale

Considerando due tabelle con n attributi A comuni e rispettivamente l e m attributi B e C esclusivi, il join naturale dà in output l’unione delle tabelle, l’intersezione dei tempi e le tuple con gli stessi valori negli attributi comuni.

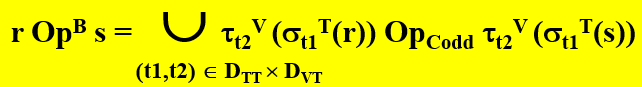
Precisamente, l’insieme delle tuple z avrà n+l+m+1 attributi e deve rispettare le seguenti condizioni:

* prendendo una tupla x in r e una y in s, x e y devono avere la stessa parte dati, inoltre l’intersezione delle due parti temporali non deve essere nulla;
* La parte A e B di z sono uguali alla parte A e B di x, lo stesso discorso vale anche per la tupla y e l’insieme C.
* la parte temporale di z è l’intersezione di quelle di x e y.



## Riflessioni sulla semantica funzionale

Esiste un’interpretazione geometrica da cui deriva l’implementazione vista in precedenza, qual è? Ebbene, questa interpretazione dice che ogni operatore bitemporale è l’unione di tutte le operazioni (fatte col rispettivo operatore di Codd) applicate su ogni chronon.



Dal momento che il database bitemporale può essere considerato con un’insieme di database uno per ogni tempo, per ogni coppia (t1,t2) si prende ogni tempo e si vanno a vedere i dati che interessano.

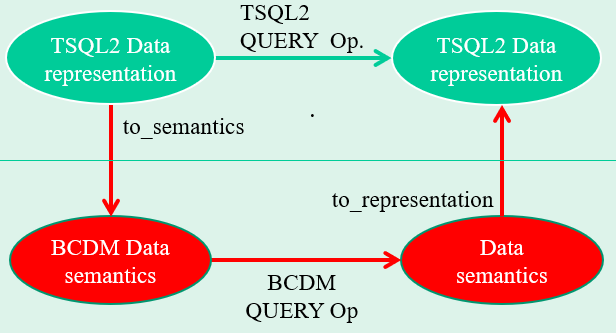
Quel che si ottiene è una coppia di tabelle non temporali, quindi si procede utilizzano la medesima operazione ma utilizzando Codd, mettendo insieme tutti i tempi.

L'interpretazione geometrica permette di concludere che tutti gli operatori temporali sono coerenti con la semantica del tempo, ciò vale per via della replicazione dell’operatore di Codd su ogni tempo.

# Algebra e semantica delle query TSQL2

L’algebra di TSQL2 deve essere efficiente oltre che rispettare la semantica, di conseguenza non si lavora su ogni chronon bitemporale bensì sulla rappresentazione in TSQL2.

Per rispettare la semantica BCDM, occorre ottenere un risultato snapshot equivalent, come si fa?

* Si convertono i dati da TSQL2 a BCDM;
* Si eseguono le query in BCDM coi dati convertiti;
* Si mappa il risultato delle query in TSQL2.

## Selezione

Come già visto, la selezione è un’operazione che non tiene conto della parte temporale, quindi rimane uguale alla controparte in BCDM.



## Unione

Questa versione dell’unione non ha bisogno del coalescing, infatti quest’ultimo solitamente non si fa per via del suo costo, oltre al fatto che l’unione è l’operazione più frequente.

In questi casi non è quindi un problema indicare la stessa informazione più volte.

L’unione restituisce quindi un insieme di tuple z avente n attributi valore più quattro temporali, secondo la seguente condizione:

* esistono una tupla x e y rispettivamente in r1 e r2 uguali alla tupla z.



## Proiezione

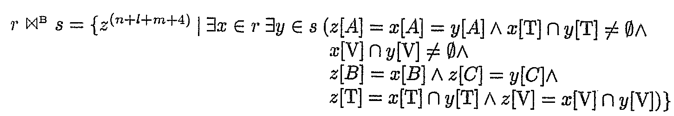
Anche nella proiezioni non si fa coalescing per lo stesso motivo dell’unione. Qui l’insieme di tuple z ottenuto su un insieme D di attributi è ottenuto:

* controllando se esiste una tupla x in r con la stessa parte dati D;
* z e x devono inoltre avere gli stessi tempi di validità.



## Join naturale

Il join naturale è praticamente uguale alla sua controparte in BCDM, la differenza è che l’intersezione è quella geometrica invece che su un insieme di punti.



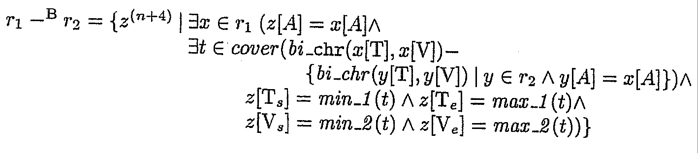
## Differenza

La differenza in TSQL2 contiene tutta la complessità non espressa nelle precedenti operazioni. Date due tuple x e y rispettivamente nelle tabelle r1 e r2, l’idea è quella di effettuare la differenza dei tempi, funziona? Non facendola per ogni coppia di tuple, bisogna trovare un altro modo.

Quindi come si procede? Data una tupla x in r1, si effettua la differenza tra essa e ogni tupla y nella tabella r2. Il risultato è un insieme di tuple z ottenuto come segue:

* si verifica se esiste una tupla x con la stessa parte dati di z;
* si prende la parte temporale di x e si estraggono i chronon con bi\_chr;
* dopodichè per y in r2 con la stessa parte dati di z, si estraggono i chronon da essa;
* si effettua la differenza e infini si riconverte tutto con una funzione di cover;
* Da ciò che è stato ottenuto in precedenza, si estrae t;
* I vari attributi temporali della tupla vengono ricavati con delle funzioni di massimo e minimo.

Questa operazione risulta la più inefficiente, tuttavia è accettabile dal momento che è quella meno utilizzata tra tutte.



## Preservare la snapshot equivalence

Che proprietà hanno tutti gli operatori temporali citati prima? Tutti preservano la snapshot equivalence e sono un’estensione degli operatori dell’algebra di Codd.

Come si può dimostrare ciò?

Date due tabelle, bisogna verificare per ogni operatore le proprietà di:

* riducibilità: Da una tabella temporale, si vuole dimostrare che il risultato ottenuto è una tabella temporale snapshot equivalent. Se a quest’ultima si applica una slice, si ottiene una tabella risultato.

Allo stesso modo se si effettua prima lo slice e poi l’operatore di Codd si ottiene lo stesso risultato.

* Equivalenza: essenzialmente è il contrario della riduzione, tutto il sistema continua a funzionare se a una tabella non temporale si aggiungono i tempi. Fare questa operazione prima o dopo non conta, i risultati ottenuti sono differenti ma snapshot equivalent.

## Snapshot equivalence

Dato un operatore a, esso preserva la snapshot equivalence tra r e r’ se per tutti i parametri X, vale che:



Questa proprietà deve valere per ogni operatore dell’algebra.

### Dimostrazione

Date delle tabelle r, r’, s, s’ per ipotesi snapshot equivalent a due a due, si vuole dimostrare la tesi che r join s è snapshot equivalent con r’ join s’.

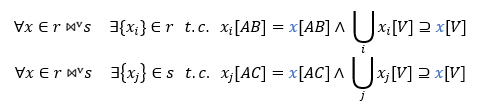
Considerando la tupla x in r join s, la sua presenza in questa tabella indica che esistono tuple in r e s che rispettano le condizioni dell’operatore.

Di conseguenza, esistono due tuple xi e xj rispettivamente in r e s value equivalent su un insieme A di attributi.

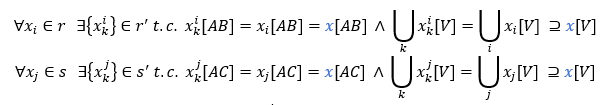
Considerando le tuple x e xi, esse sono snapshot equivalent per un insieme di attributi A e un insieme B, inoltre i tempi di validità di x sono contenuti in quelli di xi.

Dato che è stata definita una tupla z avente come tempo di validità l’intersezione di quelle di x e y, non è quindi detto che ogni tempo di validità di xi sia contenuto in xi, quindi l’unione di tutti i tempi di xi può presentare elementi non inclusi nei tempi di x.

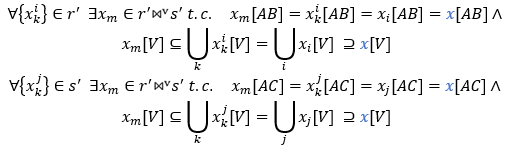
Lo stesso discorso vale anche tra x e xj per gli insiemi di attributi A e C.



Ora si trasportano i vincoli di xi e xj sulle tuple di r’ e s’ che, essendo snapshot equivalent con r e s, deve valere che:



Inoltre, l’esistenza di xik e xjk implica che:



Ciò dimostra che ogni tupla xm in r’ join s’ ha la stessa parte dati e lo stesso tempo di validità di x, di conseguenza è stato dimostrato che r join s è contenuto in r’ join s’, non è difficile dimostrare anche l’opposto.



## Dimostrazione di riducibilità

Date due tabelle temporali r e s e un operatore di slice p bitemporale, si vuole dimostrare che p(r join s) e p(r) join p(s) sono tra loro snapshot-equivalent.

Dato x’’ in p(r join s), occorre far riferimento alla definizione di slice: esiste x’ in r join s tale che la parte dati e il tempo di validità sono uguali mentre il tempo di transazione t contenuto in quello di x’.



Il passo successivo sta nel capire se x’’ è in r join s, ma cosa c’è in queste due tabelle? Precisamente si trovano delle tuple x1 e x2 (appartenenti rispettivamente a r e s) avente la stessa parte dati di x’, contengono il tempo di transazione t e la loro validità copre x’.



Con una tupla con queste caratteristiche, cosa c’è nello slice? essenzialmente c’è una tupla uguale ma senza il tempo di transazione, ciò vale sia per r, sia per s. Di conseguenza il join contiene le tuple di r e s aventi lo stesso tempo di validità.Si può dimostrare tutto ciò anche in un altro modo: per definizione di p, esiste un tupla x1’ in p(r) tale che x’ e x1’ hanno la stessa parte dati e lo stesso tempo di validità, la stessa cosa vale per p(s).



x1’ e x2’ risultano quindi essere tuple uguali a x1 e x2 in cui è stato applicato lo slice temporale.

Cosa sarà presente allora in p(r) join p(s)? Ebbene, risultaterà esserci una tupla x’’12 avente la stessa parte dati di x1’ e x2’ e come tempo di validità l’intersezione di quelli delle due tuple.

### Dimostrazione alternativa

Dal momento che gli operatori temporali sono rappresentabili come un’unione come visto in precedenza, la proprietà di riducibilità è garantita per definizione dell’operatore dato che l’operatore temporale su tabelle non temporali funziona come quello di Codd.

## Dimostrazione di equivalenza

L’equivalenza dimostra il contrario di quello che dimostra la riducibilità.

Data una tabella r, se si applica a essa un operatore tau che aggiunge il tempo e subito dopo si applica un operatore op temporale, si ottiene una tabella temporale. Un risultato snapshot equivalent si ottiene applicando a r prima l’operatore di Codd corrispondente e poi si aggiunge il tempo con tau.

## Altre operazioni che preservano la snapshot equivalence

### Coalescing

L’operazione di coalescing tra due tuple x e x’ deve rispettare la semantica per essere snapshot equivalent. Essa può essere realizzata solo se l’area di copertura risulta essere un rettangolo. Senza questa condizione, non si può rappresentare l’informazione con un’unica tupla in TSQL2.

### Eliminazione delle ripetizioni

L’eliminazione di ripetizioni risulta utile in TSQL2 in presenza di duplicazioni a causa di due o più tuple value equivalent, i cui rettangoli si intersecano tra loro. Eliminare ripetizioni permette di rappresentare la stessa area utilizzando tre rettangoli adiacenti, come una sorta di standardizzazione. Ciò permette di mette in evidenza il tempo che interessa preservando la snapshot equivalence, l’area occupata infatti non cambia.

## Valutazione di algebre relazionali temporali alternative

TSQL2 non è l’unico approccio per gestire il tempo nei database, esistono infatti tanti approcci alternativi, con quali approcci si valutano?

* Vedere se il tempo di validità sono associati alle tuple o agli attributi, quindi come modellare la storia delle combinazioni di attributi oppure quella degli attributi stessi. Una rappresentazione è omogenea quando l’unione dei tempi attributo per attributo è sempre la stessa, di conseguenza gli attributi sono definiti per gli stessi tempi. L’omogeneità è importante dato che permette di garantire una semantica consistente e senza ambiguità. Una tupla non omogenea può infatti avere attributi non associati a dei tempi;
* come vengono rappresentati i tempi i generale, di solito si utilizzano intervalli, insiemi di chronos o insiemi di intervalli ( quest’ultimo viola la prima forma normale ma tutto rimane comunque gestibile dato che mantiene l’unicità di rappresentazione)
* Vedere se l’algebra mantiene gli operatori di Codd e se li estende. Qui la risposta è scontata e difficilmente si tornerà indietro da questa scelta. Ciò comunque non vuol dire che non esistano operatori specifici per gestire il tempo che non derivano da Codd.

## Criteri di valutazione

### Omogeneità

L’omogeneità è importante dato che garantisce una semantica chiara sul contenuto del database. Quando la rappresentazione non è omogenea, non si capisce bene il contenuto, infatti nella foto a destra non si da quale sia il salario dopo il 31 giugno (indeterminatezza temporale).

### Estensione consistente dell’algebra di Codd

La consistenza dell’algebra vale se equivalenza e riducibilità vengono dimostrate, ciò è fondamentale per garantire l’interoperabilità degli approcci, tuttavia non è scontato dal momento che esistono periodi temporali non gestibili.

### Supporto alla periodicità dei dati

La periodicità dei dati indica come si ripetono nel tempo, ci sono due modi per rappresentarli:

* Utilizzare il linguaggio delle query, ciò però occupa troppo tempo e spazio;
* Utilizzare anche il linguaggio dei dati per limitare le ripetizioni, tutta l’informazione è quindi implicita dato che si compattano tutte le ripetizioni.

In quest’ultimo caso entrano in gioco manipolatori simbolici per il query answering, è comunque necessaria la dimostrazione per la snapshot equivalence.

Anche in questo caso deve valere la snapshot equivalence con dimostrazione di riducibilità ed equivalenza.

### Legalità delle tuple e delle relazioni

Ogni insieme di valori legali per gli attributi può costituire una tupla legale, ciò significa che, aggiungendo il tempo, tante cose date per scontate potrebbero non esserlo più.

Lo stesso discorso vale anche per le relazioni, infatti non è detto che un insieme di tuple legali costituisca una relazione legale.

### L’algebra è davvero un algebra?

Per essere tale, in un’algebra deve vale la proprietà di chiusura sull’operatore, ciò non è banale dato che si rischia di violare la definizione stessa.

Con chiusura si intende che, data la rappresentazione di un linguaggio, ogni operatore che prende un input in quel formato genera un output espresso allo stesso modo.

Nel caso di BCDM e TSQL2 gli operatori visti sono tutti chiusi, ciò non è tuttavia scontato dal momento che potrebbero servire operatori per cui non vale la proprietà di chiusura.

### Supporto a timestamp multidimensionali

In alcuni domini può essere utile associare più tempi di validità alle tuple, come gestirli?

Non cambia molto rispetto al caso unidimensionale né nella rappresentazione, nè tantomeno nella gestione, infatti ciò si può ottenere utilizzando tabelle distinte.

In questi casi il tempo di validità non è visto come un singolo tempo bensì come un vettore di tempi.

### Riducibilità a Codd

La riducibilità è una proprietà importante dal momento che garantisce interoperabilità coi database non temporali.

In ogni caso non è sempre una proprietà desiderabile, un esempio riguarda la gestione dei fatti teli e atelici.

### Supporto ad algebre equivalenti

Questa proprietà è importante dal momento che è alla base delle ottimizzazioni delle query. Anche qui l’aggiunta dei tempi rende tante cose poco scontate.

### Supporto a ogni classe di relazioni

Questa proprietà indica il supporto ai database non temporali, snapshot, historical e bitemporal, tuttavia è più simile a un approccio che a un criterio.

### Trattamento ortogonale dei tempi

Nel caso generale tutto funziona in maniera ortogonale, ma nello specifico come si può rappresentare il “now”? C’è tutto un problema semantico non banale che i vari approcci affrontano a modo loro, alcuni ad esempio lo unificano con l’UC ma non è la stessa cosa.

### Unicità di rappresentazione

L’unicità di rappresentazione indica l’assenza di tuple value equivalent, questa proprietà è desiderabile ma esistono approcci come TSQL2 che non la supportano per efficienza e altri motivi.

# Approfondimento sulla semantica temporale

Tutti gli approcci considerati sono basati su un’assunzione: un database temporale è un’insieme di database normali, uno per ogni tempo.

Questa assunzione è detta “snapshot semantic” o “point based semantic” dal momento che tutto viene valutato punto per punto.

Tutto ciò è bello dal punto di vista dei database dato che permette di ottenere proprietà come riducibilità ed equivalenza, tuttavia non è sempre la scelta più corretta: secondo Aristotele, l’ontologia del mondo distingue i fatti in:

* telici: fatti aventi un obiettivo, ad esempio “John costruì una casa”;
* atelici: fatti senza obiettivo, come per esempio “John dormì”.

Questa distinzione è alla base della cultura odierna, a livello cognitivo è parte dell’acquisizione verbale dei bambini, nell’informatica invece? secondo Steedman e Moens, un sistema per essere user-friendly ha bisogno di una rappresentazione dei dati molto vicina alla visione umana.

Senza questa distinzione, ci sarebbero errori sulla rappresentazione.

## Distinzione nella linguistica

Nella linguistica si classificano le frasi in base all’azionalità, distinguendo le frasi sia nel linguaggio, sia nel significato. Precisamente si distinguono:

* le attività, cioè le azioni senza obiettivo, un esempio è “John dormì”;
* risultati, le azioni durativeche culminano con un obiettivo, “John costruì una casa” ne è un esempio;
* obiettivi: azioni istantanee con un obiettivo, ad esempio “John raggiunse la vetta di una montagna”;
* stati: azioni durative ma senza dinamicità, infatti vengono utilizzati per definire delle proprietà (“John ha la febbre”).

Le distinzioni sopra sono dei raffinamenti, tuttavia possono esserci anche eccezioni.

Nei quattro casi definiti sopra, attività e stati sono fatti atelici dato che non presentano obiettivi, tutti gli altri invece sono fatti telici.

## Semantica

Solo con una certa informazione, non si riesce a capire la semantica di una frase, infatti aggiungendo un progressivo la frase cambia. Anche le classi stesse si comportano in un modo semanticamente omogeneo: dato un intervallo indicante la verità per una frase f, f è uno stato se è vera in ogni sottointervallo, è invece un risultato o un obiettivo se in ogni sottointervallo è falsa.

Per fare degli esempi:

* “John è professore dal 1 al 31 Gennaio” indica che John è professore ad esempio il 15 Gennaio!;
* “John costruì la casa dal 1 al 31 Gennaio” non indica ad esempio che John ha costruito la casa il 20 Gennaio, in quel periodo la casa non è ancora finita!

I fatti atelici hanno le seguenti proprietà:

* L’ereditarietà verso il basso, cioè che una proprietà vale su intervallo se vale anche in ogni suo sottointervallo;
* L’ereditarietà verso l’alto invece è duale alla precedente: se un proprietà vale in un sottointervallo, allora vale anche per tutto l’intervallo.

Queste proprietà però non valgono per i fatti telici, il linguaggio però è flessibile, infatti esiste un modo per convertire una frase telica in atelica e viceversa.

Un fatto atelico per definizione è un fatto per cui vale sia l’eredità verso l’alto, sia quella verso il basso, ciò invece non vale per i fatti telici.

## Revisione di BCDM e TSQL2 tenendo conto di questa visione

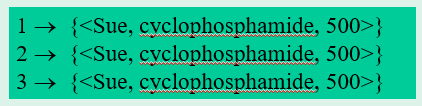
La semantica di BCDM è point-based e ciò permette l’estensione consistente di un database relazionale in uno temporale.

TSQL2, dato che è un’implementazione di BCDM ad alto livello, permette di esprimere le cose in maniera più compatta rispettando la semantica.

Considerando la foto a destra, in termini di ereditarietà, di che si parla? Per le tuple 10, 11 e 12 vale sempre l’ereditarietà verso il basso in TSQL2, quella verso l’alto è garantita dal coalescing.

Tutti i database temporali si basano su BCDM, quindi c’è un supporto naturale all’ereditarietà e di conseguenza ai fatti atelici, per quelli telici invece?

## Supporto ai fatti telici nei database temporali

Per fare un esempio, si considera che Sue ha l’amministrazione di 500 milligrammi di un farmaco per il cancro da 1 a 3, ebbene l’unico modo per modellarlo in BCDM è quello nella foto a destra, tuttavia si va a forzare l’ereditarietà verso il basso (risulta che Sue prende 500 mg di farmaco ogni giorno).

Allo stesso modo con “Sue prende 500 mg da 4 a 5” si va a forzare anche l’ereditarietà verso l’alto, dicendo che Sue prende da 1 a 4 un totale di 1000 mg di farmaco.

Tutto ciò è sbagliato!La semantica point-based non è quindi espressiva per esprimere i fatti telici, come fare?

Ciò che ha ingannato i progettisti di database è quindi la semantica stessa, infatti due tuple con lo stesso significato non rendono per come sono rappresentate.

Per permettere ciò si aggiungono semantiche basate su intervalli, ciò permette di mantenere i fatte atelici e di essere più espressivi per quelli telici, mantenendo le loro proprietà.

Gli intervalli in questo caso vengono trattati come oggetti atomici e non scomponibili in insiemi di punti, di conseguenza non supporta l’ereditarietà.

Per ovviare a questo problema, si gestiscono i fatti atelici con una semantica a punti, per quelli telici invece si utilizza la semantica a intervalli citata prima.

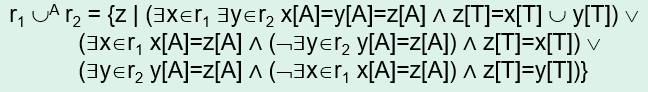
## Cosa si aggiunge a livello di complessità?

Dal punto di vista dell’utente, si ha un incarico in più per la gestione dei fatti telici. Tutte queste distinzioni sono tutte a disposizione e quindi non ci sono scuse per non utilizzarle, avvicinando i database alla visione del mondo.

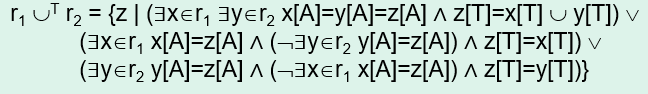
La complessità aggiunta è quindi solo apparente dal momento che l’utente ha poche cose in più da fare al livello dei dati.

## Semantica delle query

L’impatto della dicotomia telica/atelica nei database ha introdotto un approccio interval-based per la gestione dei fatti telici, questo lavora insieme a una semantica point-based utile per i fatti atelici.

La semantica delle query ha bisogno di flessibilità dal momento che il linguaggio naturale è flessibile:

* L’unione atelica è già definita in BCDM, qui effettua l’unione dei tempi delle tuple value-equivalent, permettendo il coalesce automatico.
* L’unione telica invece utilizza l’unione insiemistica tra concetti atomici e non, in questo caso non si può forzare l’ereditarietà verso l’alto se due o più tuple sono disgiunte.

Questo approccio permette polimorfismo dato che l’operatore insiemistico viene utilizzato allo stesso modo su cose differenti.

Alcuni operatori possono essere o solo atelici o solo telici, la durata ad esempio è telica.

## Come si fanno le conversioni?

Le conversioni per convertire un intervallo in un insieme di punti e viceversa si fa attraverso appositi operatori, ciò causa però perdita di informazione.

L’operatore ToInterval ricava il numero minimo di intervalli a partire da un insieme di punti, coprendoli tutti, ToPoint invece estrae ogni punto dell'intervallo dato in input.

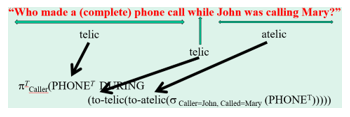
Un discorso simile vale anche per la conversione di un fatto da telico ad atelico e viceversa, precisamente:

* ToTelic converte un fatto atelico in telico sfruttando la funzione ToInterval, infatti la parte temporale rimane tale e viene convertita in modo adeguato;
* ToAtelic converte un fatto telico in atelico allo stesso modo di ToTelic ma utilzzando la funzione ToPoint per convertire la parte temporale.

Tutte queste conversione permettono ogni giochino possibile, quindi per utilizzare operatori telici su oggetti atelici e viceversa, si fanno delle conversioni.

## Esempio: durata delle chiamate telefoniche

Considerando le chiamate telefoniche, queste sono oggetti telici, quindi si utilizza l’apposita semantiche per gestirli.

Considerando la foto a destra, quando John stava chiamando Mary? Quando chiamate complete stanno completamente all’interno? chi le ha fatte?

Il fatto che John stia chiamando Mary è vero in ogni punto, rendendo il fatto atelico, tuttavia quel “mentre” indica una durata, un’operatore telico. Come si procede?

Si convertono le chiamate in oggetti atelici, si fa il coalescing e si converte tutto in oggetti telici per calcolare le durate, da qui poi si estraggono le telefonate con una selezione telica.

Tutto questo non aggiunge complessità dal momento che tutto è già presente nel linguaggio naturale.

A livello di implementazione c’è poco da fare dato che l’algebra è rimasta la stessa, è solo più polimorfa.

Inoltre, le tabelle teliche vengono trattate come tabelle non temporali.

Questo è uno “sporco trucco” dal momento che TSQL2 fornisce ereditarietà per i fatti nelle tabelle temporali, cosa invece che non vale per quelle non temporali.

# Gestione del tempo nell’intelligenza artificiale e nella conoscenza

Il tempo in questi ambiti è gestito in maniera differente rispetto ai dati, qui infatti si parla anche di informazioni implicite, cosa non presente nei dati.

Considerando la conoscenza base, essa contiene informazioni vere e, tramite l’inferenza, è possibile estrarre altrettante informazioni vere.

La conoscenza rende quindi la gestione del tempo più complessa a causa della presenza di conoscenza implicita, cosa che per sua natura non vuole essere imposta su un istante temporale.

Prendendo come esempio le linee guida mediche, si può definire un ordine temporale delle azioni definendo quale viene prima, ciò però non ha un tempo preciso dal momento che la linea guida deve essere generale per l’applicazione su ogni tipo di pazienti.

## Come si gestisce l’ontologia temporale?

La gestione tempo avviene definendo come entità base punti o intervalli temporali, essi sono due modelli semantici con potenze espressive differenti, ognuno coi propri vantaggi e svantaggi.

Per quanto riguarda la struttura del tempo, esso può essere:

* bounded quando limitato sia all’inizio, sia alla fine, altrimenti è unbounded;
* discreto/continuo quando rappresentato per punti discreti o in maniera continua.

Anche in questo caso l’approccio migliore dipende dal caso d’uso, nell’intelligenza artificiale ad esempio il tempo è infinito mentre nei database non lo è.

Ultima cosa ma non meno importante è l’ordinamento del tempo, come può essere?

* con un ordinamento lineare, c’è un’unica linea temporale;
* con i branching, il tempo è ramificato, quindi ci sono più linee temporali possibili, utile in IA per rappresentare più possibili futuri;
* circolare, cioè il tempo rappresentato come un anello.

## Approcci di gestione del tempo

L’obiettivo generale è quello di modellare sia il mondo (attraverso azioni ed eventi), sia la sua evoluzione nel tempo, qui moti strumenti di intelligenza artificiale fanno spesso capo alla logica e possono essere di conseguenza convertiti.

Esistono due approcci per fare ciò:

* situation calculus: si analizzano i casi come una successione di situazioni, precisamente si analizzano gli effetti causati dalle azioni;
* event calculus: essenzialmente la stessa cosa ma utilizzando gli eventi.

## Situation calculus

L’idea del situation calculus è quella di aggiungere un argomento per indicare il tempo, la cosiddetta situazione. Ciò permette di ottenere le proposizioni fluent, cioè delle funzioni che danno valori di verità alle situazioni, utili per descrivere i fatti.

Per affrontare un problema temporale, bisogna quindi descriverne:

* le causa con le proposizioni fluent, in questo caso una situazione s è seguita da una situazione in cui il propositional fluent a è vero, il tutto descritto con il fluent F(a,x);
* le azioni, modellati come risultati di una causa attraverso i situation fluent, dando in output un nuovo fatto, descrivendone l’effetto.

## Action and Change

Dopo circa una decade, i ricercatori hanno capito la difficoltà del situation calculus dal momento che risultava troppo cognitivo e i comportamenti ottenuti non erano quelli aspettati (tenendo conto del tempo).

Per tenere conto del tempo bisogna specificare troppe cose, precisamente:

* troppe precondizioni per tirare fuori tutte le inferenze (qualification problem);
* troppe conseguenze delle azioni per avere inferenza completa (ramification problem);
* specificare cosa rimane persistente nonostante le azioni, cosa che non si può fare con la logica standard (frame problem).

## Quindi come si procede?

Quello che si può fare è ragionare andando a fare assunzioni per definire la persistenza e le normalità, le logiche standard però non hanno questi meccanismi e quindi si utilizzano quelle non monotone.

Con le logiche non monotone, a partire da fatti, le cose aggiunte potrebbero non aggiungere nuove soluzioni oppure potrebbero non inferirne altre.

Dal punto di vista tecnico c’è però un problema: il ragionamento non monotono è inefficiente per un calcolatori, infatti occorre dimostrare prima ciò che non è ordinario ( e lo si fa utilizzando tutta la conoscenza).

Da ciò però ci sono state delle soluzioni che hanno permesso l’utilizzo efficiente della logica.

## Logiche temporali

Le logiche temporali permettono lo studio diverse modalità della verità in base alla gestione di più mondi possibili, ognuno associato a un tempo. Ogni mondo è in relazione con gli altri tramite delle relazioni di precedenza temporale.

Le logiche temporali forniscono diversi operatori temporali per descrivere e ragionare come i valori di verità asseriscono nel tempo.

## Propositional Linear Temporal Logic (PLTL)

PLTL è una logica temporale che utilizza un tempo lineare, essa definisce quattro operatori temporali:

* F p indica che p è vero nel mondo s attuale se esiste un mondo tra i successivi s in cui p è vero;
* G p indica che p è vero nel mondo s attuale se p è vero in ogni mondo successivo a s;
* in X p, p è vero nel mondo s attuale se nel mondo successivo a s, p è vero;
* p U q: p è vero nel mondo s attuale e in quelli successivi in cui q è vero.

Tutti questi operatori semplificano la gestione delle varie casistiche, se così non fosse, tutto sarebbe molto complesso da gestire in una logica del primo ordine.

# Gestione dei vincoli temporali

La gestione dei vincoli temporali deve essere il più possibile generale in modo da coprire più casi possibili, tutto ciò al costo dell’efficienza.

Negli approcci general purpose, si cerca di rendere tutto il più generale possibile, ciò però porta a essere computazionalmente inefficienti dato che la generalità porta ad avere grandi quantità di problemi.

Rinunciando alla generalità, si ottengono gli approcci specializzati, i quali sono molto più efficienti dei primi ma permettono di gestire solo dei casi specifici. L’idea è quella di costruire soluzioni efficient su parti ben definite in modo modulare, dopodichè si realizzano knowledge servers i quali, insieme ad altri sistemi, gestiscono i ragionamenti.

Al base di questi sistemi c’è il concetto della gestione dei vincoli temporali, definiti di solito come parte del problema della gestione del tempo, separata da tutto il resto.

Prima di definire i vincoli temporali, bisogna distinguere tra:

* semantica intesa, quella per cui l’utente pensa sia supportata dal reasoner/database;
* semantica supportata, quella realmente supportata da questi.

Senza meccanismi di inferenza, è inutile provare a rappresentare i vincoli temporali, ciò infatti va in contrasto con l’intuizione dell’utente. Un reasoner è quindi necessario e deve offrire:

* trattabilità, cioè che deve dare risposte in tempo polinomiale;
* correttezza delle inferenze;
* completezza, tutti i vincoli impliciti devono essere esplicitati dall’algoritmo di propagazione dei vincoli, facendo tutte le inferenze possibili. Perdere la completezza significa non garantire correttezza, infatti quest’ultima proprietà viene meno quando non è possibile fare tutte le inferenze, anche se tutte corrette.

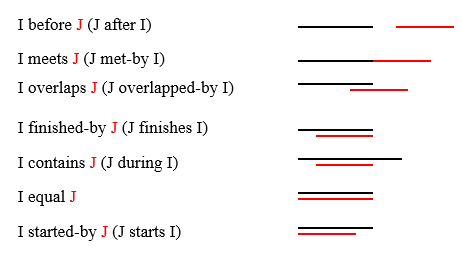
## Come possono essere i vincoli?

I vincoli possono essere punti, intervalli o insiemi di questi, tutto dipende dal livello di dettaglio che si vuole ottenere.

I vincoli temporali possono essere:

* qualitativi: indicano un posizionamento relativo delle entità, quindi non hanno un un tempo fisso associato;
* quantitativi: vincoli in cui entra in gioco la metrica del tempo, essi indicano date, durate e ritardi;
* periodici: vincoli tra oggetti ripetuti nel tempo.

Per quanto riguarda i linguaggi per i vincoli qualitativi, questi possono essere:

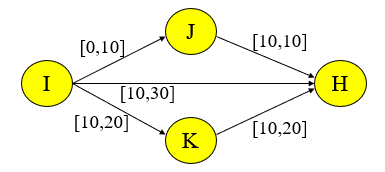
* algebre di punti: in questo linguaggio occorre capire quale punto viene prima (<), dopo (>) o durante (=) un altro.

Questo linguaggio è il più semplice e presenta incertezza dato che un punto può essere a destra o a sinistra. Le relazioni composte tra punti sono ogni possibile combinazione delle relazioni base.

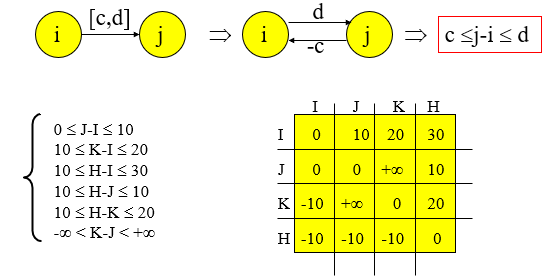
* algebre di punti continui: è come il linguaggio precedente con la differenza che i punti non possono essere sia prima, sia dopo, quindi non c’è l’operatore composto <>;
* algebre di intervalli: i vincoli base riguardano ogni possibile posizione tra due intervalli, in tutto sono 13 e ciò equivale a 2^13 possibili combinazioni per le relazioni composte.

## Temporal reasoning

Il ragionamento temporale si basa sulla chiusura transitiva: considerando la foto a destra, se si suppone un vincolo da I a J e da J a K, allora per transitività esiste un vincolo da I a K.

E nei casi come quello della foto a destra?

Essenzialmente si etichettano i vincoli calcolando la distanza minima e massima. In questo caso il nuovo vincolo I→ H è dato da I→ J e J→ H, quindi se la distanza minima è minore si sostituisce il vincolo.

Con questa regola di inferenza, si può sfruttare il fatto che i vincoli hanno uno specifico linguaggio, ciò permette di scrivere tutto come un insieme di disuguaglianze lineare e gestire tutto con l’algoritmo giusto.

Considerando la foto a destra, si utilizza l’algoritmo di Floyd-Warshall per il calcolo dei cammini minimi, qui la consistenza è garantita dall’assenza di cicli negativi.

In caso di vincoli qualitativi, l’algoritmo di Floyd-Warshall funziona con opportuni cambiamenti:

* con la normale algebra dei punti, si deve aggiungere un ciclo for per gestire le disuguaglianze, la complessità diventa quindi O(n^4);
* con l’algebra dei punti continua, la complessità non cambia.

L’unico caso in cui l’algoritmo di Floyd-Warshall non funziona in tempo ragionevole è il caso dell’algebra degli intervalli, l’algoritmo infatti presenta un’esplosione quadratica dovuta al grande numero di combinazioni.

Per quanto riguarda la gestione dei vincoli quantitativi, l’approccio STP è quello che permette una soluzione in un tempo ragionevole, TCSP è un meccanismo più ricco ma richiede un tempo esponenziale, esistono tuttavia delle ottimizzazioni.

E’ inoltre possibile avere vincoli sia qualitativi, sia quantitativi insieme, in tal caso esistono approcci ibridi che permettono di combinare tra loro diversi sistemi, oppure si può effettuare un mapping su STP di un sottoinsieme di vincoli qualitativi.

Ultimi ma non meno importanti, esistono linguaggi per gestire la propagazione di vincoli qualitativi su durate, oppure approcci più omogenei basati sulla programmazione lineare.