UNIVERSITÀ DEGLI STUDI DI VERONA
FACOLTÀ DI MM.FF.NN
LAUREA SPECIALISTICA IN INFORMATICA

DISPENSA DEL CORSO DI
SISTEMI INFORMATIVI

Prof. Carlo Combi

Basi di dati e dimensioni temporali

Appunti a cura di

E. Peri
M. Devincenzi

Indice

Capitolo 1

Basi di dati temporali

Mentre come sappiamo nelle basi di dati tradizionali il tempo è utilizzato come dominio dei vari attributi (es. DATE, TIME, TIMESTAMP, INTERVAL etc.), nelle basi di dati temporali vengono introdotti dei costrutti specifici per definire dati che hanno un dominio nel tempo ovvero una dimensione temporale. Mentre nelle basi di dati tradizionali possiamo effettuare solo interrogazioni relative allo stato *attuale* al momento dell'interrogazione, in una base di dati temporale, in cui vengono salvate opportunamente tutte le modifiche effettuate nel tempo sui dati, possiamo richiedere il valore di un dato ad un certo punto della sua *storia*: in questo caso si parla di *realtà statica*. In una base di dati temporale, è inoltre possibile richiedere il numero di volte che un certo dato ha subito delle modifiche o effettuare un'interrogazione che ritorni l'evoluzione del dato in un certo intervallo di tempo: in questo caso il risultato dell'interrogazione descrive una *realtà dinamica*.

1.1 Tempo di transazione e tempo di validità

Le due dimensioni temporali fondamentali sono:

Tempo di transazione (TT): è l'intervallo di tempo in cui un'informazione è presente ed è corrente nella base di dati;

Tempo di validità (VT): è l'intervallo di tempo in cui un'informazione è vera nella realtà modellata.

Queste dimensioni sono ortogonali in quanto sono concettualmente indipendenti l'una dall'altra e ognuna ha proprietà specifiche. Il TT è append only ossia non è possibile modificare le informazioni presenti nella base di dati, ma solo aggiungerne di nuove; per questo motivo tutte le informazioni di inserimento, cancellazione e modifica devono essere gestite dal DBMS per avere una consistenza dell'informazione rispetto alla crescita temporale della

base di dati. Il VT invece è gestito dal sistema per quanto riguarda le interrogazioni e dall'utente per quanto riguarda gli inserimenti poichè è l'utente stesso che deve specificare la validità del dato. Le basi di dati che supportano il tempo di transazione sono dette basi di dati *rollback* (ovvero indicano che un'operazione sulla base di dati è annullabile). Le basi di dati che supportano il tempo di validità si dicono basi di dati *storiche* in quanto modellano l'evoluzione di un dato; basi di dati che supportano entrambe queste caratteristiche si dicono *bitemporali*. In generale l'aggettivo temporale è utilizzato per indicare una qualche forma di gestione del tempo.

1.1.1 Esempio

Il 10 agosto 2005, il medico prescrive ad un paziente una terapia a base di bipuvac dalle 10 alle 14. Le informazioni riguardo alla terapia sono inserite nella base di dati alle ore 9. A causa di un'inattesa reazione al farmaco da parte del paziente, l'infusione di bipuvac viene interrotta alle 11.15 e sostituita da una terapia basata su diazepam che inizia alle 11.25 e termina alle 14. La base di dati viene aggiornata dal medico alle ore 12. La Tabella 1.1 mostra lo stato della tabella prescrizioni al termine di tutti gli inserimenti. Dopo il primo inserimento la tabella prescrizioni conteneva solamente la prima riga in cui l'estremo destro del TT era caratterizzato dal simbolo speciale ∞. In questa tabella ogni tupla contiene i timestamp relativi al

Principio attivo	VT	TT
bipuvac	[10/08/05 10.00-10/08/05 14.00]	[10/08/05 09.00-10/08/05 12.00]
bipuvac	[10/08/05 10.00-10/08/05 11.15]	[10/08/05 12.00-∞)
diazepam	[10/08/05 11.15-10/08/05 14.00]	[10/08/05 12.00-∞)

Tabella 1.1: Prescrizioni

tempo di validità e al tempo di transazione. Entrambe queste dimensioni temporali sono rappresentate da intervalli in cui l'occorrenza del simbolo ∞ nell'estremo destro indica che l'intervallo include lo stato corrente (in letteratura tale simbolo è talvolta indicato con `NOW` oppure `Until Change`).

A questo punto ci domandiamo: possono esistere più tempi di validità per una stessa informazione in una tabella? L'approccio è quello di avere un unico VT per un'informazione. Nel caso si trovi necessario mettere diversi VT significa che probabilmente l'informazione descrive più eventi tra loro indipendenti. La soluzione è quindi quella di separare tali eventi ed assegnare a ciascuno il proprio VT.

Nella nostra trattazione assumiamo che il dominio di tempo valido sia quello ordinato linearmente. Questo esclude tempi di validità detti *branching* che sono in genere gestiti nelle basi di dati temporali tramite opportune estensioni e sono utilizzati ad esempio nello studio delle possibili evoluzioni del mercato finanziario.

1.2 Tempo dell'evento

Ci sono aspetti temporali che non possono essere colti con il VT ed il TT; consideriamo i seguenti scenari:

Scenario1 (evento on-time, aggiornamento ritardato): si vuole incrementare il salario di un medico a partire dalla data del 1 Ottobre 2006; l'informazione viene memorizzata nella base di dati il giorno 1 Novembre 2006;

Scenario2 (evento retroattivo, aggiornamento on-time): il 1 Novembre 2006 si memorizza nella base di dati il fatto che lo stipendio del medico viene aumentato a partire dal 1 Ottobre 2006.

Nel primo scenario l'aumento di stipendio è conosciuto nel momento stesso in cui diventa valido (1 Ottobre), mentre nel secondo scenario anche se il tempo in cui l'aumento di stipendio diventa valido è precedente al tempo della promozione, l'aumento di stipendio non è conosciuto finchè non avviene la promozione (1 Novembre). Le basi di dati bitemporali gestiscono gli aggiornamenti ritardati e retroattivi nello stesso modo quindi non è possibile distinguere i due scenari. Per risolvere questo problema si è proposta una nuova dimensione temporale: il *tempo dell'evento* (ET); si definisce il tempo dell'evento rispetto ad un fatto come il tempo in cui nel mondo reale è avvenuto l'evento che ha provocato il fatto stesso.

In base alla relazione tra ET e l'estremo destro dell'intervallo VT è possibile identificare gli eventi in tre classi:

Eventi on-time (VT=ET): per esempio, una terapia inizia immediatamente dopo l'analisi del medico;

Eventi retroattivi (VT<ET) per esempio, il direttore di un ospedale decide il 16/03/06 di aumentare il salario dei medici a partire dal 1/03/06;

Eventi proattivi (VT>ET): per esempio, un medico di famiglia decide il 23 luglio 2005 l'ospedalizzazione di un paziente per il 30 luglio 2005.

Un'altra classificazione può essere ottenuta considerando la relazione tra ET e TT:

Aggiornamento on-time (TT=ET): il tempo di transazione coincide con il tempo dell'evento. Questo avviene quando l'informazione è inserita nel momento in cui è generata;

Aggiornamento ritardato (TT<ET): l'informazione è inserita dopo essere stata generata;

Aggiornamento anticipato (TT>ET): l'informazione è inserita prima di essere stata generata. Tale nozione di aggiornamento anticipato è utile per modellare ipotetiche evoluzioni degli eventi.

Poiché le due precedenti classificazioni sono ortogonali, si possono avere tutte le possibili combinazioni delle due. In particolare lo scenario 1, visto in precedenza, corrisponde ad un evento on-time con aggiornamento ritardato mentre lo scenario 2 corrisponde ad un evento retroattivo con aggiornamento on-time. In alcuni casi, un singolo tempo dell'evento non è sufficente poichè vorremmo poter distinguere i tempi di occorrenza di due eventi che rispettivamente iniziano e terminano l'intervallo di validità dell'informazione, come mostrato nel seguente esempio.

1.2.1 Esempio

Il 10 agosto 2005 alle 8.00 il medico prescrive una terapia a base di bipuvac dalle 10.00 alle 14.00. La base di dati viene aggiornata alle ore 9.00. A causa di un'inattesa reazione da parte del paziente, alle 11.00 il medico decide di modificare la terapia. L'infusione di bipuvac viene quindi interrotta alle 11.15 e sostituita con una terapia a base di diazepam dalle 11.25 alle 14.00. Queste nuove informazioni sono inserite nella base di dati alle ore 12.00.

Soluzione 1 (non corretta)

La Tabella 1.2 mostra il primo tentativo di modellazione della situazione. La soluzione non è corretta in quanto la decisione di far terminare alle 11.15 la terapia di bipuvac non è stata presa alle 8.00, come invece indica l'attributo ET della seconda tupla. È da notare che la prima tupla non può essere utilizzata per spiegare la situazione, poichè essa è stata logicamente cancellata e non modella lo stato corrente della base di dati.

Principio attivo	VT	ET	TT
bipuvac	[10/08/05 10.00 - 10/08/05 14.00]	10/08/05 08.00	[10/08/05 09.00 - 10/08/05 12.00]
bipuvac	[10/08/05 10.00 - 10/08/05 11.15]	10/08/05 08.00	[10/08/05 12.00 - ∞)
diazepam	[10/08/05 11.25 - 10/08/05 14.00]	10/08/05 11.00	[10/08/05 12.00 - ∞)

Tabella 1.2: Prescrizioni

Soluzione 2 (non corretta)

Questa rappresentazione alternativa è comunque scorretta, infatti si deduce che il medico abbia prescritto il bipuvac alle ore 11.00 con validità dalle ore 10.00 alle ore 11.15, modellando così un evento retroattivo invece che proattivo.

Principio attivo	VT	ET	TT
bipuvac	[10/08/05 10.00 - 10/08/05 14.00]	10/08/05 08.00	[10/08/05 09.00 - 10/08/05 12.00]
bipuvac	[10/08/05 10.00 - 10/08/05 11.15]	10/08/05 11.00	[10/08/05 12.00 - ∞]
diazepam	[10/08/05 11.25 - 10/08/05 14.00]	10/08/05 11.00	[10/08/05 12.00 - ∞]

Tabella 1.3: Prescrizioni

Soluzione corretta

Utilizzando un tempo dell'evento di inizio e di fine è possibile ora modellare il tempo in cui è stata presa la decisione di iniziare o terminare la terapia a base di bipuvac. Infatti nella seconda tupla ET_i indica che la decisione di far iniziare la terapia alle 10.00 è stata presa alle 8.00 mentre ET_t indica il fatto che la decisione di farla terminare alle 11.15 è stata presa alle 11.00. In Tabella 1.4 viene mostrata la soluzione corretta.

Principio attivo	VT	ET_i	ET_t	TT
bipuvac	[10/08/05 10.00 - 10/08/05 14.00]	10/08/05 08.00	10/08/05 08.00	[10/08/05 09.00 - 10/08/05 12.00]
bipuvac	[10/08/05 10.00 - 10/08/05 11.15]	10/08/05 08.00	10/08/05 11.00	[10/08/05 12.00 - ∞]
diazepam	[10/08/05 11.25 - 10/08/05 14.00]	10/08/05 11.00	10/08/05 11.00	[10/08/05 12.00 - ∞]

Tabella 1.4: Prescrizioni

Ridefiniamo quindi il tempo dell'evento di un fatto come il tempo durante il quale si verificano, nel mondo reale, gli eventi che rispettivamente danno inizio e terminano il periodo di validità del fatto stesso. A differenza delle altre due dimensioni temporali, l'ET è rappresentato non da un periodo ma da due istanti: l'istante in cui si è verificato l'evento che ha determinato l'inizio del VT (ET_i) e l'istante in cui si è verificato l'evento che ne ha determinato la fine (ET_t). Ovviamente deve valere la relazione che $ET_i < ET_t$.

1.3 I sistemi informativi ed il tempo

Un sistema informativo è l'insieme dei flussi di informazioni di un'organizzazione e le risorse che lo gestiscono, come ad esempio le persone e i sistemi informatici. Da un certo punto di vista, vorremmo modellare il tempo al quale il sistema informativo viene a conoscenza di una certa informazione, allo stesso modo in cui si è modellato il tempo in cui l'informazione è memorizzata nella base di dati. Mentre il secondo aspetto è modellato dal tempo di transazione, il primo non è stato ancora modellato esplicitamente.

1.3.1 Esempio

A causa di un trauma avuto il 15 settembre 2005, Angela accusa un forte mal di testa a partire dal 1 ottobre. Il 7 ottobre, Angela viene visitata da un medico. Il 9 ottobre il medico le somministra una cura adeguata. Il giorno successivo, il medico inserisce le informazioni acquisite riguardo alla situazione medica di Angela nella base di dati. Il 15 ottobre, la paziente informa il medico che il suo mal di testa e' guarito il 14 ottobre; il medico inserisce queste nuove informazioni nella base di dati il giorno stesso. La situazione della base di dati relativa alla paziente dopo il primo inserimento è mostrata in Tabella 1.5 mentre quella relativa al secondo inserimento da parte del medico nella Tabella 1.6 Con il solo TT non è possibile dire che il

Sintomo	VT	ET_i	ET_t	TT
Mal di testa	[01/10/05 - ∞]	15/09/05	null	[10/10/05 - ∞]

Tabella 1.5: Prescrizioni

Sintomo	VT	ET_i	ET_t	TT
Mal di testa	[01/10/05 - ∞]	15/09/05	null	[10/10/05 - 15/10/05]
Mal di testa	[01/10/05 - 14/10/05]	15/09/05	09/10/05	[15/10/05 - ∞]

Tabella 1.6: Prescrizioni

medico era a conoscenza dell'informazione, ovvero non gli si può attribuire il merito della guarigione perchè la tupla è stata inserita il 10 Ottobre ma il sintomo è sparito il giorno prima. In generale il tempo al quale l'informazione diventa disponibile precede, ma non coincide necessariamente con il tempo al quale è memorizzata nella base di dati e quindi non si può utilizzare il tempo di transazione per modellarlo.

In molte applicazioni gli inserimenti di dati sono raggruppati ed eseguiti in batch, spesso in base a form di report precedentemente compilati. In questi casi, il tempo di transazione non può assumere il valore del tempo durante il quale l'informazione è stata acquisita dal sistema informativo. Talvolta potrebbe succedere che l'ordine secondo cui le informazioni

sono conosciute dal sistema informativo differisce dall'ordine in cui esse sono memorizzate nella base di dati. In molti domini applicativi, incluso quello medico, le decisioni sono prese sulla base delle informazioni che sono disponibili in quel momento nel sistema informativo senza che siano necessariamente memorizzate nella base di dati. La Figura 1.1 mostra come i tempi di transazione e di disponibilità sono in relazione rispettivamente con la base di dati e il sistema informativo.

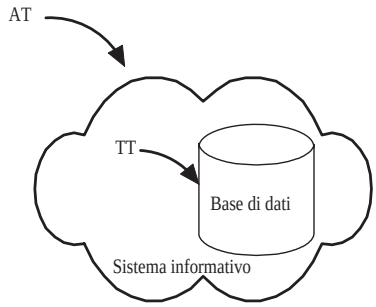

Figura 1.1: Tempo di disponibilità e di transazione e relazioni con base di dati e sistema informativo

Introduciamo quindi una nuova dimensione temporale, il *tempo di disponibilità* (AT) che rappresenta il tempo al quale un'informazione diventa disponibile al sistema informativo. Questa dimensione temporale può coincidere con il tempo di transazione quando l'acquisizione di un'informazione da parte del sistema informativo si traduce nell'immediata memorizzazione della stessa nella base di dati, ma ciò non sempre accade. Spesso il tempo in cui il fatto è noto al sistema informativo non coincide con il tempo in cui il fatto è presente nella base di dati. Ciò può accadere per diversi motivi:

- il personale che acquisisce l'informazione non coincide con quello che ha il compito di memorizzarla nella base di dati;
- il sistema di inserimento dei dati nell'archivio non è fisicamente accessibile da chi acquisisce l'informazione;
- non è possibile una memorizzazione immediata dei dati.

1.3.2 Soluzione modellata utilizzando il tempo di disponibilità

Il tempo di disponibilità permette di modellare in modo corretto il problema dell'esempio precedente; infatti il tempo di disponibilità della prima tupla, ovvero il tempo in cui il medico viene a conoscenza della presenza del mal di testa di Angela (7 ottobre), precede strettamente l'inizio dell'intervallo

di transazione (10 ottobre) mentre il tempo della base di dati e l'intervallo del tempo di transazione della seconda tupla coincidono (15 ottobre). La Tabella 1.7 mostra la situazione aggiornata al termine del secondo inserimento. Finora si è considerato l'AT per modellare il tempo rispetto al quale

Sintomo	VT	ET_i	ET_t	AT	TT
Mal di testa	[01/10/05 - ∞]	15/09/05	null	07/10/05	[10/10/05 - 15/10/05]
Mal di testa	[01/10/05 - 14/10/05]	15/09/05	09/10/05	15/10/05	[15/10/05 - ∞]

Tabella 1.7: Prescrizioni

l'informazione diventa disponibile al sistema informativo senza considerare la possibilità che il sistema informativo acquisisca informazioni non corrette. Nel seguito sarà mostrato come la nozione di AT possa essere generalizzata per tenere conto di questa possibilità.

1.3.3 Esempio

A causa di un errore d'inserimento, il trauma di Angela è stato memorizzato nella base di dati come avvenuto il 5 settembre 2005. Solamente il 20 ottobre l'errore viene notato. Il giorno successivo il medico inserisce il dato corretto nella base di dati. La situazione dopo l'ultimo inserimento è mostrata in Tabella 1.8. Una limitazione che si nota in questa rappresentazione è

Sintomo	VT	ET_i	ET_t	AT	TT
Mal di testa	[01/10/05 - ∞]	05/09/05	null	07/10/05	[10/10/05 - 15/10/05]
Mal di testa	[01/10/05 - 14/10/05]	05/09/05	09/10/05	15/10/05	[15/10/05 - 21/10/05]
Mal di testa	[01/10/05 - 14/10/05]	15/09/05	09/10/05	20/10/05	[21/10/05 - ∞]

Tabella 1.8: Prescrizioni

che l'informazione riguardante l'intervallo di tempo in cui ciascun fatto è conosciuto e ritenuto corretto non può essere ottenuta dalla corrispondente tupla. Ad esempio, per concludere che la seconda tupla era considerata corretta fino al 20 Ottobre, è necessario poter leggere anche la terza tupla.

Queste considerazioni portano quindi a modificare il tipo dell'AT da singolo istante a intervallo. Il tempo di disponibilità diventa quindi l'intervallo di tempo durante il quale un'informazione è conosciuta e considerata corretta dal sistema informativo. L'istante di partenza dell'AT è quello in cui il fatto diventa disponibile al sistema informativo mentre il suo istante finale è quello in cui il sistema informativo realizza che il fatto non è più corretto. Come nel caso del TT, il simbolo speciale ∞ posto nell'estremo destro dell'intervallo AT, sta ad indicare il fatto che l'informazione è ritenuta corretta allo stato attuale. In Tabella 1.9 vediamo la situazione precedente modellata considerando queste modifiche per il tempo di disponibilità.

Sintomo	VT	ET_i	ET_t	AT	TT
Mal di testa	[01/10/05 - ∞]	05/09/05	null	[07/10/05 - 15/10/05]	[10/10/05 - 15/10/05]
Mal di testa	[01/10/05 - 14/10/05]	05/09/05	09/10/05	[15/10/05 - 20/10/05]	[15/10/05 - 21/10/05]
Mal di testa	[01/10/05 - 14/10/05]	15/09/05	09/10/05	[20/10/05 - ∞]	[21/10/05 - ∞]

Tabella 1.9: Prescrizioni

Il tempo di disponibilità può essere visto come il tempo di transazione del sistema informativo. Se il sistema informativo esterno alla base di dati è considerato esso stesso una base di dati, allora il tempo di disponibilità di un'informazione è il tempo in cui il fatto è inserito o eliminato dal sistema informativo. Questo parallelismo tra tempo di disponibilità e tempo di transazione rende chiaro il motivo per cui il tempo di disponibilità debba essere rappresentanto mediante un intervallo. La Figura 1.2 chiarisce la relazione tra le varie dimensioni temporali, la realtà modellata e il sistema informativo.

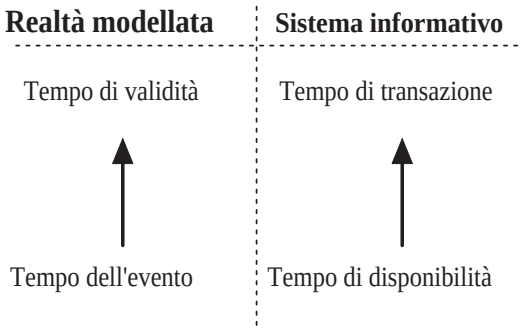

Figura 1.2: Relazioni tra i vari tempi

1.4 Riassunto e precisazioni

- Il tempo di transazione è append only;
- l'inserimento di una nuova informazione si modella mediante un'intervallo aperto a destra del TT ad indicare che tale informazione appartiene allo stato corrente della base di dati;
- la cancellazione logica di un'informazione non corretta è ottenuta sostituendo l'intervallo illimitato a destra con un intervallo limitato superiormente del corrispondente fatto;
- i tempi di validità e dell'evento, essendo relativi ai tempi reali rappre-

sentati, possono appartenere sia al presente che al futuro e possono essere modificati;

- il tempo di disponibilità è append only per sua natura poiché fatti precedentemente conosciuti e considerati corretti dal sistema informativo non possono essere cambiati;
- dal punto di vista della base di dati il tempo di validità potrebbe essere append only se non ci fossero errori nell'inserimento dei dati;
- poiché non è possibile escludere nessuna possibilità, gli stati precedenti del sistema informativo, riguardo il tempo di disponibilità, possono essere modificati aggiungendo nuove informazioni.

1.5 Alcuni problemi sugli aspetti temporali

1.5.1 Il problema delle chiavi

Quando sono presenti più attributi temporali, sorgono alcuni problemi riguardo la scelta della chiave primaria; alcune scelte che possono sembrare ovvie portano, in alcuni casi, ad avere informazione duplicata. Non basta quindi definire le chiavi primarie ma occorre gestire opportunamente i vincoli temporali.

Si consideri come esempio la relazione mostrata nella Tabella 1.10.

Sintomo	VT_s	VT_e	TT_s	TT_e
Mal di testa	1	8	10	∞
Mal di testa	1	8	28	∞
Mal di testa	1	8	40	∞

Tabella 1.10: Relazione *sintomo*

Come si può notare non è possibile considerare come chiave primaria solamente l'attributo "Sintomo", in quanto sono presenti più tuple con lo stesso valore. Se si considera il tempo di transazione e l'attributo "Sintomo" come chiave primaria, occorre garantire che la proprietà di chiave valga ad ogni istantanea della base di dati. Ad esempio se viene eseguita una selezione del tipo: "$TT_s \leq i \ AND \ TT_e > i$" per un $i = 30$, si avrebbe il risultato mostrato in Tabella 1.11: Le tuple ottenute rappresentano lo stesso evento: dal punto di vista del sistema è corretto, infatti è possibile inserire più eventi uguali in tempi di transazione diversi, ma questo non è corretto concettualmente, in quanto la stessa informazione risulta essere presente più volte.

Sintomo	VT_s	VT_e	TT_s	TT_e
Mal di testa	1	8	10	∞
Mal di testa	1	8	28	∞

Tabella 1.11: Risultato

Per evitare questo problema occorre considerare come chiave primaria sia tempo di transazione che il tempo di validità. In questa situazione è desiderabile memorizzare uno stesso sintomo in intervalli temporali diversi, ma si vogliono evitare situazioni in cui l'intersezione dei tempi dati dagli intervalli di inizio e fine del VT di tuple successive non siano disgiunti (Tabella 1.12 e Tabella 1.13).

Sintomo	VT_s	VT_e	TT_s	TT_e
Mal di testa	1	8	10	∞
Mal di testa	20	25	28	∞

Tabella 1.12: Caso corretto: $VT_s \bigcap VT_e = \varnothing$

Sintomo	VT_s	VT_e	TT_s	TT_e
Mal di testa	1	8	10	∞
Mal di testa	7	25	28	∞

Tabella 1.13: Caso scorretto: $VT_s \bigcap VT_e \neq \varnothing$

Infine si potrebbe considerare come chiave primaria sia un attributo di descrizione del fatto (come "Sintomo" nel nostro caso) che l'intervallo VT: in questo caso si risolve il problema di memorizzare uno stesso evento su intervalli di VT uguali, ma occorre gestire opportunamente il tempo di validità affinchè gli intervalli i tempi di inizio e di fine dei VT siano disgiunti.

1.5.2 Interrogazioni

Nella fase di interrogazione risulta molto utile gestire le dimensioni temporali tramite un linguaggio arricchito di clausole opportune. In caso contrario, una gestione diretta di tali informazioni da parte dell'utente può essere molto onerosa.

Esempio. Data la relazione R in Tabella 1.14, si vogliono determinare per ogni paziente, le coppie di sintomi uno successivo all'altro.

P_id	Sintomo	VT_s	VT_e
1	vomito	1	5
1	Mal di testa	7	10
1	febbre	15	20

Tabella 1.14: Relazione R

È possibile risolvere questa interrogazione mediante la seguente query dell'algebra relazionale:

• **Algebra relazionale**[1]**:**

$$Q \longleftarrow R \bowtie_{P\_id=P\_id1 \wedge VT_e<VT_{s1}} R1$$

$$Q - \pi(Q \bowtie_{P\_id=P\_id\star \wedge VT_s=VT_s\star \wedge VT_e=VT_e\star \wedge VT_{s1}>VT_{e1}\star \wedge sint=sint\star)} Q\star)$$

Con il join Q si sono trovate tutte le coppie successive, poi si sono eliminate le coppie che ne hanno un'altra in mezzo. Ad esempio con il primo join si sono ottenute le seguenti coppie: prima e seconda tupla, prima e terza, seconda e terza. La coppia costituita dalla prima e terza tupla deve essere eliminata.

• **L'equivalente query SQL è la seguente:**

```
SELECT P1.P_id, P1.sintomo, P2.sintomo
FROM   R AS P1, R AS P2
WHERE  P1.id = P2.id AND P2.VT_s = ANY (
      SELECT MIN (P3.val_s)
      FROM   R AS P3
      WHERE  P3.P_id = P1.P_id  AND
             P3.VT_s > P1.VT_e )
```

[1]R1 e Q$\star$ sono rispettivamente le relazioni R e Q in cui tutti gli attributi sono rinominati aggiungendo il suffisso 1 o $\star$.

Capitolo 2

Interrogazioni di basi di dati e aspetti temporali

Delle quattro dimensioni temporali considerate nel capitolo precedente, due, il tempo di validità ed il tempo di transazione, sono ampiamente utilizzate nel contesto delle basi di dati temporali commerciali. Le altre due, il tempo di disponibilità ed il tempo dell'evento, sono dimensioni temporali proposte solo recentemente all'attenzione della comunità scientifica.

In letteratura sono molti i linguaggi di interrogazione e di definizione temporali che gestiscono le dimensioni temporali. Ad esempio, esistono linguaggi che gestiscono solo il tempo di validità come il **TSQL**, l'**HSQL** e il **TempSQL**; solo il tempo di transazione come il **TOSQL** e altri che gestiscono sia il tempo di transazione che il tempo di validità come il **TSQL2**, il **TQuel** e l'**SQL3**. Allo stato dell'arte nessuno di tali linguaggi gestisce tutte e quattro le dimensioni temporali descritte nel capitolo precedente.

In questo capitolo sarà discussa la recente proposta per la gestione degli aspetti temporali relativa allo standard SQL3, in particolare verrà discussa la compatibilità del linguaggio nel passaggio da un sistema atemporale ad un sistema temporale. Verranno analizzati i costrutti principali relativi alla gestione delle dimensioni temporali, ma non verrà trattata la semantica formale, poichè esula dallo scopo di questa dispensa. Verranno infine argomentate, tramite un caso di studio (Sezione 2.2), alcune caratteristiche relative alla progettazione di un linguaggio di interrogazione per basi di dati con dimensioni temporali multiple.

2.1 SQL3

SQL3 è un linguaggio per basi di dati temporali a due dimensioni che estende il linguaggio SQL92. SQL/Temporal è la parte di SQL3 che raccoglie le estensioni temporali di SQL92 e in particolare aggiunge diversi costrutti per gestire le dimensioni temporali del tempo di validità e del tempo di transazione. Questa sezione mostra come sia possibile garantire la compatibilità con lo standard SQL92 e come riutilizzare su basi di dati temporali il codice prodotto per basi di dati atemporali .

2.1.1 Il problema

Come si è visto nel capitolo precedente, nelle basi di dati tradizionali è possibile modellare, utilizzando il linguaggio SQL92, varie dimensioni temporali gestendo opportunamente degli attributi specifici. Tuttavia la gestione di tali dimensioni temporali con l'SQL92 è molto complessa, infatti come si è visto, l'utente deve risolvere molti problemi, tra i quali la definizione della chiave primaria e le interrogazioni temporali senza l'uso di clausole opportune.
In questa sezione si discuterà una trattazione più generale di tali problemi e si vedrà come l'SQL/Temporal permetta di spostare la complessità della gestione temporale dal livello utente al livello DBMS.

Si suppone ad esempio di partire dalla relazione `impiegato` avente il seguente schema: `impiegato (num, nome, via, citta, nascita)`, e dalla relazione `salario` con lo schema: `salario (num, valore)`. Successivamente si vuole memorizzare, tramite SQL92, la storia delle informazioni in entrambe le tabelle aggiungendo una colonna, `When`, con tipo di dato `TIMESTAMP`. A questo punto occorre prestare particolare attenzione, non solo alla chiave primaria, ma anche alla chiave esterna: infatti essendo la colonna `salario.num` la chiave esterna per `impiegato.num`, significa che ad ogni istante temporale, il valore della prima colonna deve occorrere anche nella seconda colonna allo stesso istante di tempo. Questo non può essere espresso tramite il costrutto della chiave esterna di SQL, perchè esso non contiene l'informazione temporale.
Se poi si volesse tenere traccia dei cambiamenti e delle cancellazioni delle tuple, occorrerebbe aggiungere ulteriori due colonne, `TempoInserzione` e `TempoCancellazione`, entrambi con tipo di dato `TIMESTAMP`.

Le interrogazioni eseguite con SQL92 che coinvolgono gli aspetti temporali, potrebbero essere notevolmente complesse; parte di questa complessità, è dovuta alla difficoltà di specificare in SQL, il corretto valore della dimensione temporale nel risultato. Le operazioni di modifica inoltre, presentano ulteriori problemi, una cancellazione per esempio, deve essere implemen-

tata tramite una operazione di aggiornamento e una di inserimento. Tuttavia, non ci sono possibilità di impedire inserimenti errati dei valori di `TempoInserzione` e di `TempoCancellazione`, dovuti ad atti di distrazione o ad atti intenzionali, provocando così potenziali problemi di sicurezza.

La soluzione è quella di avere un DBMS che gestisca in modo automatico gli aspetti temporali, come ad esempio il calcolo del valore della dimensione temporale del risultato di una interrogazione e la gestione del tempo di transazione. Inoltre il linguaggio di interrogazione può fornire un ulteriore aiuto, fornendo parole chiave per gestire le query e le modifiche temporali.

L'SQL/Temporal implementa alcuni costrutti per agevolare la gestione degli aspetti temporali da parte dell'utente; ad esempio, il tempo di transazione può essere aggiunto nel seguente modo:
`ALTER TABLE impiegato ADD TRANSACTIONTIME.`
Il tempo di validità e i vincoli temporali possono essere specificati usando la parola riservata `VALIDTIME`:

```
CREATE TABLE impiegato
   (nome VARCHAR(12),
    num INTEGER VALIDTIME PRIMARY KEY,
    via VARCHAR(22), citta VARCHAR(10),
    nascita DATE)
AS VALIDTIME PERIOD(DATE)

CREATE TABLE salario
   (num INTEGER VALIDTIME PRIMARY KEY
      VALIDTIME REFERENCES impiegato(num),
    valore INTEGER)
AS VALIDTIME PERIOD(DATE)
```

Come si può notare dall'esempio sopra, per indicare che le tabelle gestiscono il tempo di validità viene utilizzato il costrutto
"`AS VALIDTIME PERIOD(DATE)`", mentre i vincoli temporali sono garantiti per ogni istante temporale attraverso "`VALIDTIME PRIMARY KEY`" e
"`VALIDTIME REFERENCES`".

Vengono ora fornite alcune interrogazioni, per mostrare come sia facile esprimere vincoli temporali usufruendo delle caratteristiche proprie di SQL/Temporal. Tali caratteristiche verranno descritte in dettaglio nella prossima Sezione 2.1.2, si consiglia pertanto di riguardare questi esempi dopo la lettura di tale Sezione.

Esempio 1. "Fornire la lista degli impiegati che non hanno salario"; la query si riferisce allo stato corrente della tabella `impiegato`. Per estrarre

questa informazione dalla tabella temporale `impiegato`, viene utilizzata la caratteristica di *upward compatibility*:

```
SELECT nome
FROM impiegato
WHERE num NOT IN
    (SELECT num FROM salario )
```

Il risultato è una relazione convenzionale con una colonna.

Esempio 1b. "Fornire la lista degli impiegati che non hanno avuto il salario e quando"; per questa query viene utilizzata la *semantica sequenced*:

```
VALIDTIME SELECT nome
FROM impiegato
WHERE num NOT IN
    (SELECT num FROM salario )
```

L'aggiunta della clausola "VALIDTIME" specifica che la query deve essere valutata per ogni istante di tempo. Il risultato è ancora una tabella temporale che supporta il tempo di validità.

Esempio 2. "Fornire il numero degli impiegati aventi una paga superiore a 5000Euro per ogni città"; questa semplice query utilizza ancora la proprietà *upward compatibilty*:

```
SELECT citta, COUNT(*)
FROM impiegato, salario
WHERE impiegato.num = salario.num AND
   valore > 5000
GROUP BY citta
```

Come nell'Esempio 1, si è interessati allo stato corrente della base di dati, ovvero al numero degli impiegati attuali per ciascuna città.

Esempio 2b. come nell'esempio precedente, ma ora si è interessati all'evoluzione temporale dei dati relativi agli stipendi.

```
VALIDTIME SELECT citta, COUNT(*)
FROM impiegato, salario
WHERE impiegato.num = salario.num AND
   valore > 5000
GROUP BY citta
```

Con questa modifica la query ritorna un conteggio che varia nel tempo.

Esempio 3. Effettuare la modifica: "Assegnare all'impiegato Pippo un salario di 6000*Euro* per l'anno 2005"

```
VALIDTIME PERIOD '[2005-01-01 - 2005-12-31]'
UPDATE salario
SET valore = 6000
WHERE num IN
          (SELECT num
           FROM impiegato
           WHERE nome = 'Pippo')
```

2.1.2 Passaggio da un sistema atemporale ad un sistema temporale

Il linguaggio SQL3 è stato realizzato in modo da soddisfare alcune caratteristiche che sono state raggruppate in quattro classi. Questi quattro livelli permettono una migrazione dal vecchio sistema atemporale ad un nuovo sistema temporale che sia efficiente, semplice e che permetta il riutilizzo del codice esistente.

Livello 1: Upward Compatibility

Tutti gli applicativi, che gestiscono o meno dimensioni temporali, costituiscono un grosso investimento per le aziende, pertanto è improponibile un linguaggio che gestisca implicitamente aspetti temporali e non permetta il riutilizzo del codice esistente.

Questa considerazione porta alla conclusione che lo standard SQL debba essere compatibile sintatticamente e semanticamente con il nuovo linguaggio temporale. Tale requisito è detto *upward compatibility*.

Rispetto al linguaggio in questione, implementare il requisito di upward compatibility significa gestire tabelle atemporali in modo sintatticamente e semanticamente equivalente alla gestione delle tabelle nel modello relazionale classico associato al linguaggio SQL.

Per enfatizzare la relazione tra dati atemporali, dati temporali e query, si farà riferimento ad una serie di figure in cui una relazione viene rappresentata da un rettangolo. Lo stato corrente di questa tabella è denotato dal

rettangolo più a destra rispetto alla linea del tempo (timeline).

In Figura 2.1 il rettangolo con bordo continuo denota una tabella atemporale mentre i rettangoli con contorno tratteggiato rappresentano gli stati precedenti della tabella. Secondo la semantica standard del modello dei dati relazionale, quando una tabella è modificata lo stato precedente è cancellato e non ve ne rimane traccia nella base di dati. Le interrogazioni, indicate dall'etichetta q, agiscono sullo stato corrente della tabella. Per il requisito di upward compatibility questa è la semantica delle operazioni sintatticamente compatibili con il linguaggio SQL eseguite su tabelle atemporali descritte tramie lo standard SQL/Temporal.

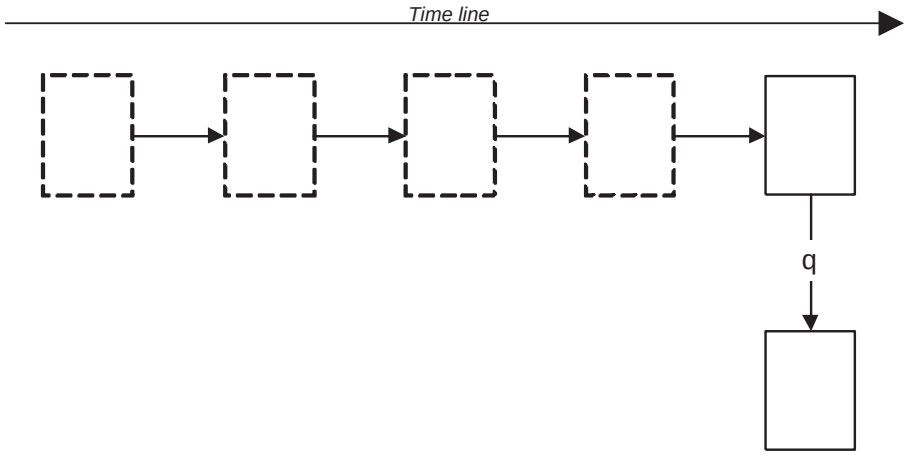

Figura 2.1: Livello 1. Interrogazione SQL su una tabella atemporale.

Esempio. In presenza di tabelle atemporali espresse in SQL3, si vuole aumentare lo stipendio del 10% all'impiegato Pippo. Lo stato precedente relativo all'impiegato Pippo viene sovrascritto dal nuovo stato. La query viene scritta in SQL3 esattamente come veniva scritta in SQL92.

Livello 2: Temporal Upward Compatibility

Il linguaggio SQL3 permette la gestione sia tabelle atemporali che temporali. Tuttavia è indesiderabile modificare il codice di applicazioni già esistenti che accedono a tabelle atemporali le quali sono state successivamente rimpiazzate con tabelle temporali. Per questo motivo, il requisito di *temporal upward compatibility* richiede che tutto il codice che agiva su una tabella atemporale, continui a produrre lo stesso risultato anche sulla medesima tabella a cui è stata aggiunta una dimensione temporale.
Le interrogazioni quindi devono agire sullo stato corrente della tabella e il

risultato è, per compatibilità, una relazione atemporale.

Per comprendere meglio la semantica legata a questo requisito si osservi la Figura 2.2. Una tabella temporale preserva tutti gli stati che nel tempo hanno subito una modifica fino ad arrivare allo stato corrente. Le interrogazioni (e tutte le operazioni di modifica) scritte in SQL92, devono agire solo sullo stato corrente della relazione.

Il requisito di temporal upward compatibility risulta fondamentale, per esempio, in tutti quei sistemi informativi in cui non si vuole mantenere traccia dell'evoluzione storica dei dati, ma non si vogliono modificare tutte le funzioni preesistenti.

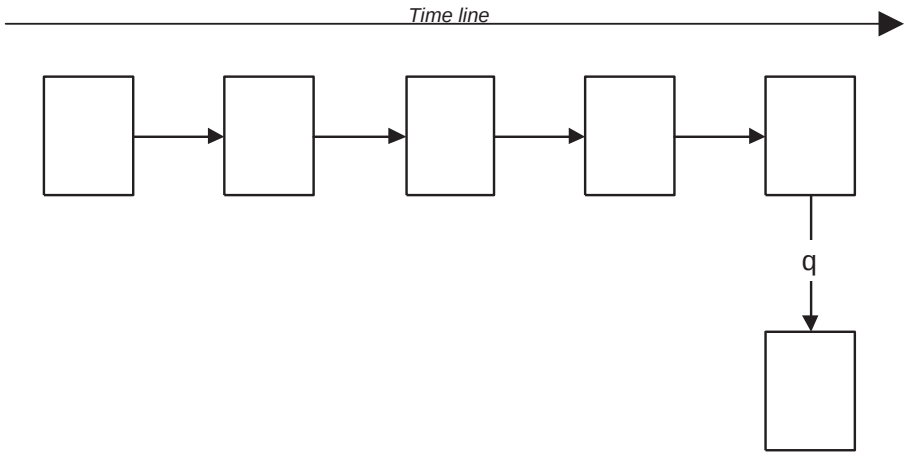

Figura 2.2: Livello 2. Interrogazione SQL su una tabella temporale.

Livello 3: Semantica Sequenced

Il requisito *upward compatibility* garantisce che il codice ereditato dalle precedenti applicazioni continui a funzionare con il nuovo sistema, e il requisito *temporal upward compatibility*, aggiunge la possibilità di tale codice di coesistere con le nuove applicazioni temporali.

Il terzo livello garantisce la modalità per cui l'interrogazione viene eseguita istante per istante su tutto l'asse temporale.
La Figura 2.3 mostra una query q' scritta in SQL/Temporal ed eseguita su ogni stato della tabella temporale. Il risultato è una tabella temporale e ad ogni risultato viene associato l'istante in cui è stata determinata.
Per esempio nel caso di una tabella che supporta il tempo di validità, il risultato è ancora una relazione con tempo di validità. L'utente è agevolato dal

fatto che può pensare alla query q' come ad una query q, scritta in SQL92 a cui si deve aggiungere la clausola per la semantica desiderata.

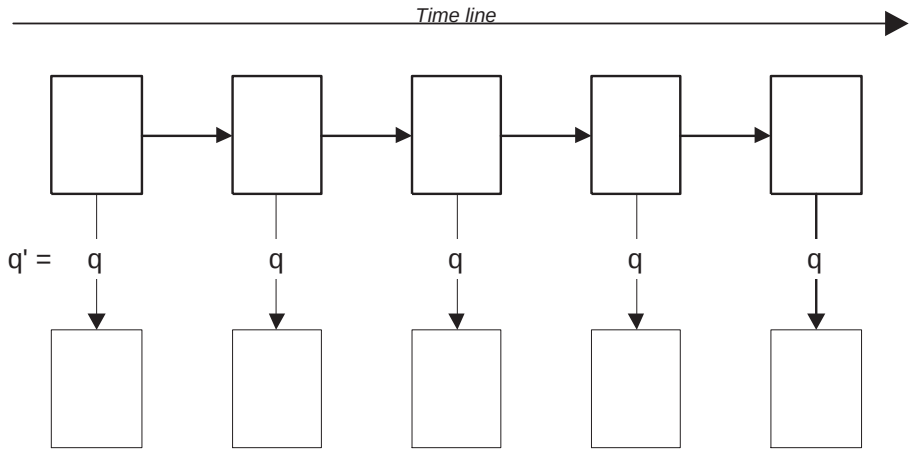

Figura 2.3: Livello 3. Interrogazione in SQL/Temporal su una tabella che supporta una dimensione temporale e ritorna una tabella che supporta la stessa dimensione.

In SQL3 questa semantica è indicata dall'utilizzo delle parole chiave `VALIDTIME`, nel caso del tempo di validità, e `TRANSACTIONTIME`, nel caso del tempo di transazione.

Come per la semantica legata ai requisiti di upward compatibility e temporal upward compatibility, anche la semantica sequenced può essere applicata ad ogni tipo di operazione SQL. La Figura 2.4 mostra come viene eseguita un'operazione di modifica u' con *semantica sequenced*: la modifica è applicata su ogni stato.

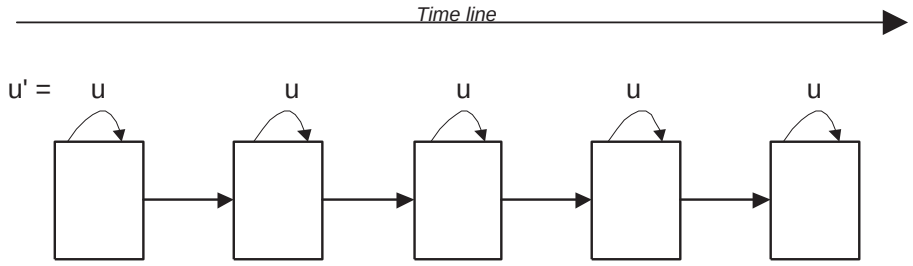

Figura 2.4: Livello 3. Modifica di una tabella temporale in SQL/Temporal.

Esempio. Un impiegato non lavora più nell'azienda e occorre cancellarlo dalla base di dati. Se viene utilizzata l'operazione `DELETE` di SQL3 (la

cui semantica è equivalente a quella di SQL92), il requisito di *upward compatibility* richiede di eliminare l'informazione solamente dallo stato corrente. Usando invece la clausola `VALIDTIME` assieme alla clausola `DELETE`, il dato viene eliminato da ogni stato della tabella.

Livello 4: Semantica Non-Sequenced

Applicando la semantica di tipo sequenced ogni informazione ricavata da un'interrogazione rispetto ad un istante della linea temporale si basa sulle informazioni della base di dati valide in quello specifico istante. Una semantica di questo tipo è molto limitata perchè esclude tutta una classe di interrogazioni che richiedono di valutare più stati per ottenere il risultato.

La semantica *non-sequenced* garantisce la possibilità di eseguire questo tipo di interrogazioni permettendo all'utente di accedere alla tabella temporale nella sua interezza. Tale semantica può essere applicata, anche in questo caso, ad ogni operazione SQL3.

La Figura 2.5 mostra come l'interrogazione q consideri, per determinare il risultato, tutti i possibili stati della tabella temporale.

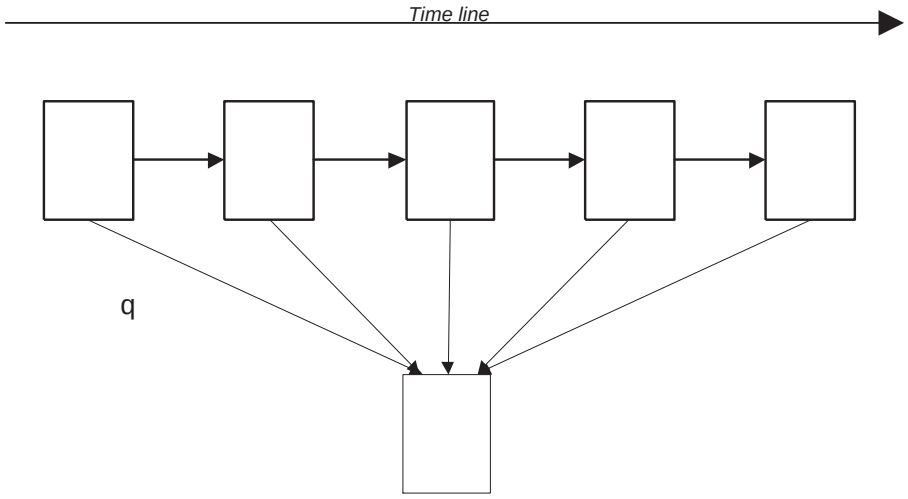

Figura 2.5: Livello 4. Interrogazione in SQL/Temporal con semantica non-sequenced con risultato atemporale.

Il risultato di un'interrogazione non-sequenced è in generale una tabella atemporale; infatti il sistema non sa come collegare la tupla risultante con gli altri stati della tabella usati per la valutazione. Se l'utente specifica tramite clausole opportune come calcolare la dimensione temporale da associare alle tuple risultanti, allora la tabella risultante risulta essere temporale

(vedi Figura 2.6).

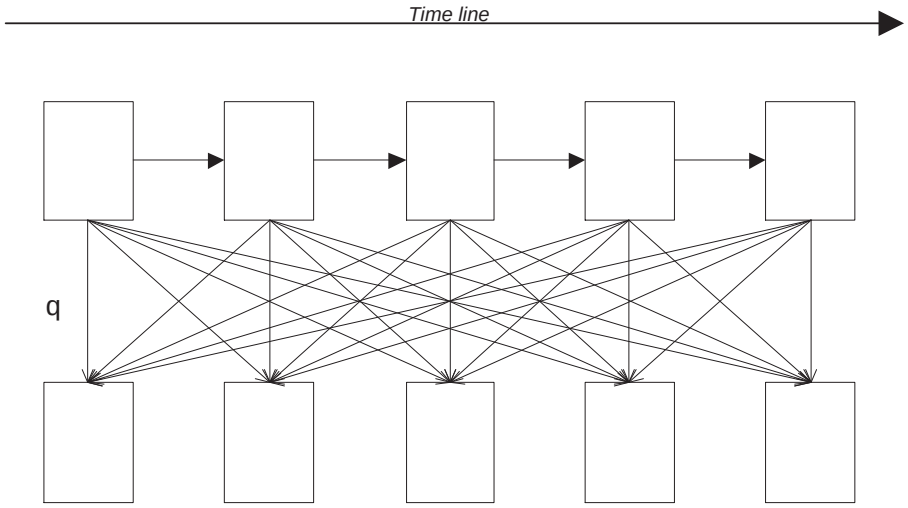

Figura 2.6: Livello 4. Interrogazione in SQL/Temporal con semantica non-sequenced con risultato temporale.

Ogni tipo di operazione può essere interpretata con semantica non-sequenced. Supponiamo, ad esempio, di voler modificare lo stato di una tabella con tempo di validità in modo che la modifica avvenga se è valida la condizione x sullo stato precedente. La Figura 2.7 mostra, schematicamente, come tale operazione è realizzata (u).

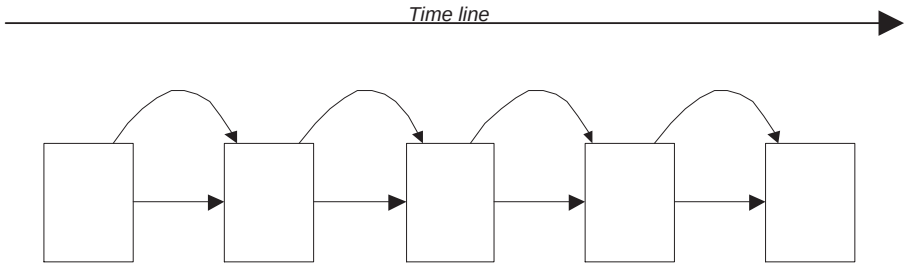

Figura 2.7: Livello 4. Valutazione di una modifica che considera lo stato precedente di una tabella.

Una semantica di tipo non-sequenced è indicata in SQL/Temporal con l'aggiunta delle parole chiave `NONSEQUENCED VALIDTIME` e `NONSEQUENCED`

`TRANSACTIONTIME`, se riferita rispettivamente al tempo di validità ed al tempo di transazione.

2.1.3 Sommario

In questa sezione, si sono formulati alcuni importanti requisiti che l'SQL/Temporal si propone di rispettare per garantire un corretto passaggio da basi di dati atemporali a basi di dati temporali.

L'*upward compatibility* garantisce che il codice ereditato da applicazioni atemporali non necessiti di modifiche quando migrato nel nuovo sistema temporale, mentre il *temporal upward compatibility* garantisce la coesistenza tra le nuove applicazioni temporali e le applicazioni già esistenti. La *semantica sequenced*, vista come estensione dei precedenti costrutti, garantisce che il nuovo linguaggio sia facile da utilizzare e da comprendere per gli utenti che hanno una formazione sui precedenti linguaggi.

L'SQL/Temporal suddivide queste proprietà in quattro livelli di funzionalità temporali:

Livello 1. Questo livello cattura la minima funzionalità che l'SQL/Temporal deve soddisfare per garantire l'upward compatibility con l'SQL3. In questo modo si ha un supporto per il codice ereditato da SQL3 (e di conseguenza da SQL92), ma in questo livello non ci sono ne relazioni ne query temporali. Praticamente l'utente dispone di un nuovo strumento con nuove potenzialità e che garantisce la compatibilità con il vecchio, ma le nuove caratteristiche non vengono utilizzate.

Livello 2. In questo livello si aggiunge la possibilità di avere relazioni che supportano dimensioni temporali. Non si utilizza la nuova sintassi nelle query e nelle operazioni, ma viene garantito che il codice che prima operava sull'ultimo stato della tabella atemporale, operi ancora sull'ultimo stato della medesima tabella a cui sono state aggiunte delle dimensioni temporali.

Livello 3. A questo livello vengono utilizzate le nuove potenzialità: le query e le varie operazioni utilizzano gli aspetti temporali per effettuare operazioni che coinvolgono i singoli stati delle relazioni. L'SQL/Temporal estende così l'SQL3 (e quindi l'SQL92) con una nuova semantica sequenced.

Livello 4. Infine a questo livello vengono pienamente utilizzate le funzionalità temporali del nuovo linguaggio. Il risultato di una query o di una operazione, non sarà legato ad un singolo stato come nella semantica sequenced, ma sarà legato a più stati della base di dati.

2.2 Caso di studio

In questa sezione verranno introdotte alcune clausole necessarie per interrogare una base di dati contenente le quattro dimensioni temporali descritte in precedenza. Nel dettaglio, saranno discusse clausole per interrogare gli stati passati, sia della base di dati che del sistema informativo (`AS OF, AS KNOWN`); l'aggiunta di qualificatori alla clausola `SELECT` per permettere di calcolare il valore della dimensione temporale sul risultato della query (`WITH`); e infine parole chiave per supportare la definizione di join temporale (`TJOIN`).

Come esempio verrà considerata una base di dati clinica formata da due relazioni: *pat_sympt*, che descrive i sintomi di un paziente e *par_ther*, che descrive la terapia prescritta dal medico.
Le due tabelle contengono le seguenti relazioni:

`pat_sympt (Pid, sympton, VT, ET`$_i$`, ET`$_t$`, AT, TT)`

`pat_ther (Pid, therapy, VT, ET`$_i$`, ET`$_t$`, AT, TT)`

L'istanza della base di dati è descritta nella Tabella 2.1

P_id	sympton	VT	ET_i	ET_t	AT	TT
1	headache	[97Oct1, ∞]	97Sept5	null	[97Oct7, 97Oct15]	[97Oct10, 97Oct15]
2	vertigo	[97Aug8, 97Aug15]	97Aug7	97Aug12	[97Sept3, 97Oct17]	[97Oct15, 97Oct21]
2	vertigo	[97Aug10, 97Aug15]	97Aug7	97Aug12	[97Oct19, ∞]	[97Oct21, ∞]
1	headache	[97Oct1, 97Oct14]	97Sept5	97Oct9	[97Oct15, 97Oct20]	[97Oct15, 97Oct21]
1	headache	[97Oct1, 97Oct14]	97Sept15	97Sept9	[97Oct20, ∞]	[97Oct21, ∞]

P_id	therapy	VT	ET_i	ET_t	AT	TT
1	aspirin	[96Oct1, 96Oct20]	96Sept25	96Oct16	[97Oct7, ∞]	[97Oct10, ∞]
2	paracetamol	[97Aug11, 97Aug12]	97Aug11	97Aug12	[97Sept3, ∞]	[97Oct15, ∞]

Tabella 2.1: Istanza base di dati. Relazione pat_sympt e pat_ther

2.2.1 WHERE e WHEN

Per permettere il confronto tra diverse dimensioni temporali, è possibile optare tra due scelte: la prima è quella di inserire sia le condizioni tempo-

rali che che quelle non temporali nella clausola `where`; la seconda è quella di inserire le restrizioni puramente temporali in una specifica clausola `WHEN`. Le ragioni che portano all'utilizzo della nuova clausola sono dovute al fatto che le condizioni temporali possono essere molto complesse, e dal punto di vista dell'utente risulta più chiaro separare la parte temporale da quella non temporale; inoltre la gestione della query può essere ottimizzata attraverso strutture di indicizzazione opportune, diverse da quelle standard applicate invece nella clausola `WHERE`. Infine occorre decidere se le condizioni miste vanno collocate nella clausola `WHERE` oppure `WHEN`: entrambe le decisioni sono corrette, l'importante è che venga effettuata una scelta implementativa definitiva e univoca.

Negli esempi che seguono si è optata l'alternativa della clausola `WHEN`. Tale clausola è utilizzata nei linguaggi temporali TQuel e TSQL, mentre in TSQL2 e in SQL3 tutti i dati, temporali e non temporali, sono inseriti nella clausola `where`: questo per motivi di compatibilità con il linguaggio SQL92 e per rispettare il requisito di semplicità sintattica deciso dal comitato promotore del linguaggio.

Esempio 1. In accordo con la scelta fatta, viene ora formulata una query che ritorna tutti i sintomi del paziente con id=2, che appaiono non più tardi di tre giorni dopo il loro tempo di evento iniziale:

```
SELECT sympton
FROM pat_sympt S
WHERE S.P_id = 2
WHEN (BEGIN(VALID(S)) - INITIATING_ET(S)) <= 3
```

dove INITIATING_ET(·), VALID(·), e BEGIN(·) sono funzioni che ritornano rispettivamente il tempo dell'evento iniziale di una tupla, l'intervallo del tempo di validità di una tupla, e il punto di inizio di un intervallo dato.

2.2.2 AS OF e AS KNOWN

Come impostazione predefinita, un'interrogazione viene valutata rispetto allo stato corrente della base di dati, ovvero, sulle tuple come tempo di transazione un intervallo aperto a destra ([date, ∞]). Ovviamente, deve esistere la possibilità di valutare l'interrogazione sugli stati correnti e passati, sia rispetto alla base di dati che al sistema informativo; tale richiesta può essere gestita aggiungendo opportuni qualificatori alla clausola `FROM`.
In letteratura viene proposta la clausola `AS OF` per gestire le interrogazioni che riguardano gli stati passati della base di dati, ovvero legate al *tempo di transazione*, e la clausola `AS KNOWN` per interrogazioni sugli stati passati rispetto al sistema informativo, ovvero legate al *tempo di validità*.

Esempio 2. Di seguito vengono mostrate due interrogazioni che forniscono il contenuto della relazione pat_sympt alla data del *20 ottobre 1997* rispetto al *tempo di transazione* e al *tempo di validità*:

```
SELECT *
FROM pat_sympt
WHERE P_id = 1
AS OF 97Oct20

SELECT *
FROM pat_sympt
WHERE P_id = 1
AS KNOWN 97Oct20
```

Il risultato di queste interrogazioni è riportato nelle tabelle 2.2 e 2.3 rispettivamente. Nella prima tabella la tupla che ne risulta descrive l'informazione presente nella base di dati nell'istante richiesto; mentre nella seconda rappresenta l'informazione conosciuta dal sistema informativo nello stesso istante.
È interessante notare che l'utilizzo separato delle due clausole permette all'utente di eseguire interrogazioni che si focalizzano su aspetti differenti dei dati memorizzati. Infatti nell'esempio proposto, confrontando i risultati, è possibile concludere che alla stessa data, il sistema informativo e la base di dati sono in possesso di informazioni differenti e in particolare che il sistema informativo è venuto a conoscenza di un evento errato inserito nella base di dati.

P_id	sympton	VT	ET_i	ET_t	AT	TT
1	headache	[97Oct1, 97Oct14]	97Sept5	97Oct9	[97Oct15, 97Oct20]	[97Oct15, 97Oct21]

Tabella 2.2: Tupla fornita tramite la clausola AS OF

P_id	sympton	VT	ET_i	ET_t	AT	TT
1	headache	[97Oct1, 97Oct14]	97Sept15	97Oct9	[97Oct20, ∞]	[97Oct21, ∞]

Tabella 2.3: Tupla fornita tramite la clausola AS KNOWN

È ovviamente possibile utilizzare entrambe le clausole contemporaneamente, come nel seguente esempio.

Esempio 3. Si vogliono conoscere le informazioni è che sono a disposizione del dottore il giorno 18 Ottobre, rispetto a quanto memorizzato nella base di dati il giorno 20 Ottobre. Per soddisfare tale richiesta occorre combinare le clausole `AS OF` e `AS KNOWN`:

```
SELECT *
FROM pat_sympt
AS OF 97Oct20
AS KNOWN 97Oct18
```

Il risultato dell'interrogazione è mostrato nella tabella 2.4. Dal tempo di disponibilità è possibile concludere che, nonostante il dato relativo al mal di testa fosse presente il giorno 20, esso sia stato concettualmente cancellato il giorno 21, e che il medico si è accorto il giorno 20 che il dato memorizzato era scorretto.

P_id	sympton	VT	ET_i	ET_t	AT	TT
1	headache	[97Oct1, 97Oct14]	97Sept5	97Oct9	[97Oct15, 97Oct20]	[97Oct15, 97Oct21]

Tabella 2.4: Tupla fornita tramite la clausola AS OF e AS KNOWN

2.2.3 Il tipo di dato PERIOD e le relazioni di Allen

Il linguaggio *SQL92* gestisce tipi di dato temporali che rappresentano istanti[1] ed intervalli[2], ma non periodi. Un periodo è un insieme di istanti consecutivi compresi tra un istante iniziale ed uno finale. Esso può essere rappresentato utilizzando due attributi di tipo istante; uno conterrà l'istante iniziale del periodo e l'altro l'istante finale. Con questa soluzione, però, anche operazioni molto semplici, come il confronto tra periodi, richiedono diverse operazioni di confronto tra istanti. Per semplificare la gestione di tali informazioni, il linguaggio *SQL3* implementa il tipo di dato `PERIOD` e una serie di nuovi costrutti per operazioni di confronto.

In *SQL3* è possibile definire un periodo sul dominio di tipi di dato temporale o sul dominio numerico. I costrutti ammessi per il tipo di dato `PERIOD` sono:

- `PERIOD(DATE)`
- `PERIOD(TIME)` e `PERIOD(TIME WITH TIME ZONE)`

[1]Si definisce istante un punto dell'asse temporale.

[2]Si definisce intervallo una durata temporale di cui è nota la dimensione, ma non la collocazione rispetto all'asse temporale.

- `PERIOD(TIMESTAMP)` e `PERIOD(TIMESTAMP WITH TIME ZONE)`
- `PERIOD(INT)` e `PERIOD(INTEGER)`
- `PERIOD(SMALLINT)`
- `PERIOD(NUMERIC)`
- `PERIOD(DECIMAL)`

Esistono quattro tipi di periodi: *close-close*, *open-open*, *open-colse* e *close-open* [3]. Per indicare un estremo close si utilizzano le parentesi quadre (`[` e `]`) mentre per indicare un estremo open si utilizzano le parentesi tonde (`(` e `)`).

Ad esempio le seguenti costanti di tipo `PERIOD(DATE)` sono tutte sintatticamente corrette in linguaggio SQL3:

- `PERIOD '[1997-01-01 - 1997-12-31]'`
- `PERIOD '[1997-01-01 - 1997-12-31)'`
- `PERIOD '(1997-01-01 - 1997-12-31]'`
- `PERIOD '(1997-01-01 - 1997-12-31)'`
- `PERIOD '[1997-01-01,1997-12-31]'`

[3] Se un estremo è di tipo close l'istante è compreso nel periodo, altrimenti se è di tipo open non è compreso.

Le relazioni di Allen

Esistono 7 tipi diversi di relazioni tra due periodi: *before*, *meets*, *overlaps*, *equals*, *start*, *finishes* e *during*; se poi vengono considerate le loro inverse, l'insieme completo è composto da 13 relazioni (l'inversa della relazione uguale è se stessa). La semantica dei vari predicati è riportata in Figura 2.8.

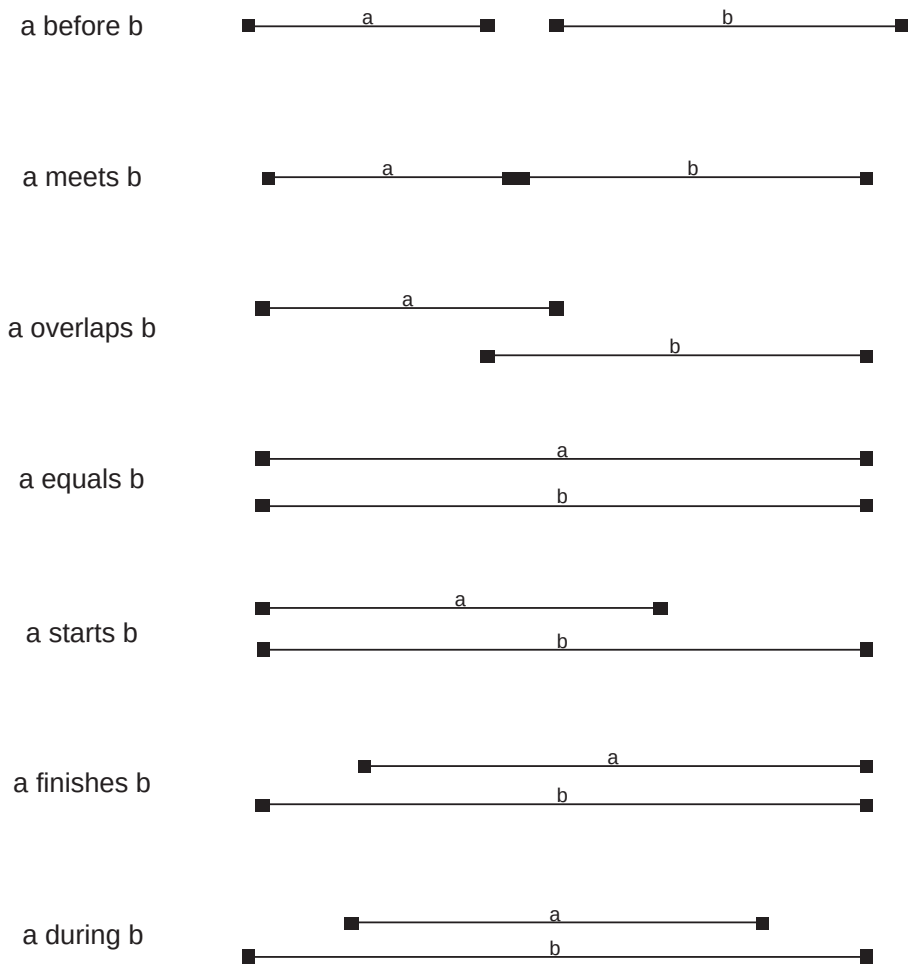

Figura 2.8: Relazioni tra periodi.

Il linguaggio SQL3 definisce le seguenti relazioni tra periodi: `OVERLAPS`, `PRECEDES`, `SUCCEEDS`, `MEETS`, `=`, `CONTAINS`. La loro semantica è definita in funzione delle relazioni di allen:

`p OVERLAPS q`, $p\ overlaps\ q \vee q\ overlaps\ p \vee p\ starts\ q \vee q\ starts\ p \vee p\ finishes\ q \vee q\ finishes\ p \vee p\ during\ q \vee q\ during\ p \vee p\ equals\ q$;

`p PRECEDES q`, $p\ before\ q$;

`p SUCCEEDS q`, $q\ before\ p$;

`p MEETS q`, $p\ meets\ q \vee\ q\ meets\ p$;

`p CONTAINS q`, $p\ during\ q\ \wedge\ p\ \neq\ q$;

`p = q`, $p\ equals\ q$.

In SQL3 sono stati inoltre introdotti diversi costrutti che riguardano intervalli, istanti e periodi.

Costrutti per istanti

SQL3 implementa i seguenti nuovi costruttori per istanti:

`BEGIN(p)`, restituisce il primo istante appartenente al periodo `p`;

`END(p)`, restituisce l'istante successivo all'ultimo appartenente al periodo `p`;

`LAST(p)`, restituisce l'ultimo istante appartenente al periodo `p`;

`PRIOR(d)`, restituisce l'istante precedente all'istante `d`;

`NEXT(d)`, restituisce l'istante successivo all'istante `d`.

Costrutti per intervalli

L'unico nuovo costruttore per intervalli introdotto in SQL3 è `INTERVAL(p g)` che restituisce la durata dell'intervallo `p` espressa con granularità `g`.

Costrutti per periodi

SQL3 implementa i seguenti nuovi costruttori per periodi:

`PERIOD [a, b)`, restituisce il periodo con istante iniziale `a` ed istante finale `PRIOR(b)`. Le varianti close-close, open-close e open-open sono ammesse.

`p P_UNION q`, restituisce l'unione dei periodi `p` e `q`. Un'eccezione è sollevata se la loro intersezione è vuota.

`p P_EXCEPT q`, restituisce gli istanti contenuti in `p` che non sono contenuti in `q`. Un'eccezione è sollevata se `p` è contenuto in `q` o `q` è contenuto in `p`.

`p P_INTERSECT q`, restituisce l'intersezione tra il periodo `p` ed il periodo `q`. Un'eccezione è sollevata se l'intersezione è vuota.

`CAST(p AS g)`, cambia la granularità di `p` in `g`.

2.2.4 Temporal Join (TJOIN)

Ulteriori aspetti da considerare sorgono nel momento in cui più relazioni fanno parte di una interrogazione. Ad esempio, occorre interpretare in modo adeguato il prodotto cartesiano delle relazioni che occorrono nella clausola `FROM`; le semantiche possibili sono le seguenti:

- semantica `NON SEQUENCED`, in cui ogni tupla su un certo insieme di attributi viene accostata ad ogni altra tupla: è la semantica più generale.
- semantica `SEQUENCED`, in cui sono considerate solo le tuple delle relazioni coinvolte nella clausola `FROM` con valori delle dimensioni temporali sovrapposti; tale semantica ad esempio può essere considerata come impostazione predefinita rispetto al *valid time* o al *transaction time*.

Inoltre, il join temporale deve ovviamente permettere di definire delle condizioni temporali dopo la parola chiave `ON`: a tale scopo si intriduce un nuovo costrutto per il *temporal join* (`TJOIN`), che permette di esprimere ogni tipo di condizione.

Esempio 4. Sulla base della Tabella 2.1, il medico vuole determinare i sintomi dei pazienti per i quali ci sono delle intersezioni dei tempi di validità tra le tabelle dei sintomi e delle terapie.

La query può essere formulata come segue:

```
SELECT sympton, S.P_id
FROM   pat_sympt S, pat_ther T
WHEN   NOT(VALID(S) BEFORE VALID(T)) AND
        NOT(VALID(T) BEFORE VALID(S))
WHERE  S.P_id = T.P_id
```

Dove `BEFORE(·)` è una delle relazioni di Allen descritta nela Sezione 2.2.3. È da notare che se il sistema ha come impostazione predefinita una semantica *sequenced*, allora le condizioni nella clausola `WHEN` potrebbero essere omesse, in quanto è il sistema stesso a garantire tali condizioni. In questo caso l'utente chiede solamente di avere i pazienti che hanno un sintomo e una terapia, e deve in qualche modo dire al sistema di utilizzare la semantica adeguata; ad esempio potrebbe anteporre alla query la seguente clausola: `SEMANTIC SEQUENCED ON VALID`.

La clausola `WHEN` può essere sostituita dalla clausola `TJOIN`, come mostrato dalla seguente query:

```
SELECT sympton, S.P_id
FROM   pat_sympt S TJOIN pat_ther T ON NOT(VALID(S)
         BEFORE VALID(T)) AND NOT(VALID(T) BEFORE VALID(S))
WHERE  S.P_id = T.P_id
```

Esempio 5. Determinare i pazienti che hanno avuto come sintomo il mal di testa ed è finito e poi hanno preso il paracetamolo come terapia.

Poichè nella fase di join vengono coinvolti più stati, si tratta di una query *non sequenced*:

```
SELECT sympton, S.P_id
FROM   pat_sympt S, pat_ther T
WHERE  S.P_id = T.P_id AND sympton = "headache" AND
         therapy = "paracetamol"
WHEN   VALID(S) BEFORE VALID(T)
```

Esempio 6. Determinare i pazienti a cui è stato somministrato il paracetamolo in risposta al mal di testa.

È una query simile alla precedente, ma in questo caso viene usata la relazione `OVERLAPS`, per determinare quei pazienti che hanno iniziato a prendere il farmaco prima del termine del periodo in cui si è verificato il sintomo.

```
SELECT sympton, S.P_id
FROM   pat_sympt S, pat_ther T
WHERE  S.P_id = T.P_id AND sympton = "headache" AND
         therapy = "paracetamol"
WHEN   VALID(S) OVERLAPS VALID(T)
```

2.2.5 Dimensioni temporali sul risultato della query (WITH)

Per valutare correttamente una query che contiene diverse relazioni temporali l'interprete SQL deve considerare alcuni criteri di default per determinare il valore delle dimensioni temporali sulla relazione risultante. Ad esempio, il tempo di validità di ciascuna tupla appartenente al risultato può essere definito come l'intersezione del valore del tempo di validità delle corrispondenti tuple appartenenti alle relazioni che occorrono nella clausola `FROM`. Il tempo dell'evento iniziale e finale può invece essere definito come il massimo dei tempi dell'evento iniziale e finale delle corrispondenti tuple.

In base a questi criteri, il risultato dell'Esempio 4 è mostrato nella Tabella 2.5.

P_id	sympton	VT	ET_i	ET_t	AT	TT
2	vertigo	[97Aug11, 97Aug12]	97Aug11	97Aug12	[97Oct19, ∞]	[97Oct21, ∞]

Tabella 2.5: Sintomi del paziente intersecati con le terapie

Tuttavia, l'utente deve avere la possibilità di definire in modo esplicito come devono essere calcolati i valori delle dimensioni temporali sulla relazione ottenuta come risultato, assegnando così un significato personalizzato alla query e alla informazione ottenuta. Come esempio, viene di seguito riformulata la query dell'esempio 4 personalizzando i valori delle dimensioni temporali da visualizzare:

```
SELECT sympton, S.P_id WITH VALID(S) AS VALID,
        INITIATING_ET(S) AS INITIATING,
        TERMINATING_ET(S) AS TERMINATING
FROM   pat_sympt S TJOIN pat_ther T ON NOT(VALID(S)
        BEFORE VALID(T)) AND NOT(VALID(T) BEFORE VALID(S))
WHERE  S.P_id = T.P_id
```

In questo caso, l'utente assegna al tempo di validità e dell'evento del risultato, il valore che questi attributi hanno nella relazione *pat_sympt*. La relazione è mostrata nella Tabella 2.6.

P_id	sympton	VT	ET_i	ET_t	AT	TT
2	vertigo	[97Aug10, 97Aug15]	97Aug7	97Aug12	[97Oct19, ∞]	[97Oct21, ∞]

Tabella 2.6: Sintomi del paziente intersecati con le terapie (modificata)

L'informazione contenuta nelle Tabelle 2.5 e 2.6 si differenziano per una diversa interpretazione del tempo di validità. Nella prima, il tempo di validità rappresenta l'intervallo di tempo durante il quale il paziente ha presentato il sintomo descritto e contemporaneamente gli è stato somministrato il farmaco, mentre nella seconda, il tempo di validità indica l'intervallo durante il quale il paziente soffre del sintomo.

Appendice A

Tabelle temporali in SQL3

A.0.6 Tabelle con tempo di validità in SQL3

In questa sezione vengono introdotti alcuni nuovi costrutti di SQL/Temporal. La presentazione è divisa nei quattro livelli descritti nella sezione 2.1.2: upward compatibility, temporal upward compatibility, sequenced e non-sequenced.

Upward Compatibility

Garantire i requisiti del livello 1 obbliga ad implementare tutti i costrutti SQL in SQL/Temporal. Quindi non ci sono estensioni a questo livello.
Tale assunzione non garantisce di per se l'upward compatibility in quanto si deve fare in modo che ogni costrutto, compatibile con la sintassi del linguaggio SQL, eseguito su tabelle atemporali da un DBMS che gestisce SQL3, produca lo stesso risultato di un'esecuzione su DBMS che gestisce SQL.

Esempio. Si suppone che un'azienda, alla data del primo gennaio del 2005, decida di creare due relazioni atemporali per la gestione degli impiegati e dei loro salari.

```
CREATE TABLE employee (
    ename VARCHAR(12),
    eno INTEGER PRIMARY KEY,
    city VARCHAR(22),
    birthdate DATE
);
```

```
CREATE TABLE salary (
    eno INTEGER REFERENCES employee(eno),
    amount INTEGER
);
```

Le tabelle sono popolate con i seguenti valori:

```
INSERT INTO employee
    VALUES ('matteo', 5873, 'torino', '1961-03-21');

INSERT INTO employee
    VALUES ('enrico',6542, 'milano', '1963-07-04');

INSERT INTO salary
    VALUES (6542, 3200);

INSERT INTO salary
    VALUES (5873, 3300);
}
```

Il dipendente matteo riceve un aumento del 10%. Dato che la tabella non gestisce dimensioni temporali, l'informazione precedente è sovrascritta e non è più presente nella base di dati.

```
UPDATE salary s
    SET amount = 1.1 * amount
    WHERE s.eno = (
        SELECT e.eno
        FROM employee e
        WHERE e.ename = 'matteo'
    );
```

Lo stato della base di dati è riportato nelle Tabelle A.1 e A.2.

ename	eno	city	birthdate
matteo	5873	torino	1961-03-21
enrico	6542	milano	1963-07-04

Tabella A.1: Tabella `impiegato`.

numero	valore
6542	3200
5873	3630

Tabella A.2: Tabella `salario`.

Temporal Upward Compatibility

SQL/Temporal permette la gestione di tabelle con tempo di validità. È possibile ottenere una tabella con questa dimensione temporale in due modi:

- con il comando `CREATE TABLE` ed aggiungere la clausola `AS VALIDTIME PERIOD(DATE)`. Questa clausola crea una relazione e gli viene associata la dimensione temporale del tempo di validità con granularità di tipo `DATE`.
- modificare una tabella atemporale. Questa operazione è eseguita con il comando `ALTER TABLE <tb> ADD VALIDTIME PERIOD(TIME)`. Anche con questa sintassi è necessario specificare la granularità della dimensione temporale.

Una volta creata una tabella temporale è necessario garantire il requisito di temporal upward compatibility. Per questo motivo tutto il codice sintatticamente esprimibile in SQL ed eseguito su tabelle con tempo di validità, viene valutato sulle tuple valide allo stato corrente. Nel caso di un'interrogazione la tabella che risulta è atemporale.

Esempio. Continuando l'esempio introdotto nella sotto-sezione precedente, si supponga di eseguire il seguente codice SQL3 il primo marzo dell'anno 2005.

```
ALTER TABLE salario ADD VALIDTIME PERIOD(DATE)

ALTER TABLE impiegato ADD VALIDTIME PERIOD(DATE)
```

A questo punto le tabelle `salario` ed `impiegato` sono diventate tabelle temporali.

Le tuple presenti nella tabella devono ora assumere dei valori significativi rispetto alle dimensioni temporali aggiunte successivamente.

Per default viene assegnato il periodo `'[2005-03-01 9999-12- 31)'` ossia il periodo compreso tra l'istante della modifica della tabella e l'istante massimo rappresentabile rispetto alla granularità del periodo. In SQL3 l'istante massimo rappresentabile assume il significato della variabile `now` nel

linguaggio TSQL2 ed è inserito quando non è noto l'istante finale di un periodo.

I vincoli precedentemente espressi devono ora essere interpretati a seguito del cambiamento della tipologia della relazione. Anche in questo caso è applicata la semantica temporal upward compatibility.
Prendendo ad esempio il vincolo di chiave primaria espresso in fase di creazione della tabella `impiegato`, a seguito dell'aggiunta della dimensione temporale, il vincolo continua a rimanere valido, ma solo rispetto alla stato corrente. Questo significa che al momento non è possibile inserire nella tabella una tupla con `numero` nullo o il cui valore sia associato ad un'altra tupla valida allo stato corrente.

Supponiamo, in data 2-3-2005, di eseguire i seguenti inserimenti:

```
INSERTO INTO impiegato
    VALUES('marco' 3463, 'como', DATE '1970-03-09')

INSERT INTO salario
    VALUES('3463, 3400)
```

La situazione della base di dati è rappresentata nelle Tabelle A.3 e A.4.

nome	numero	citta	dataNascita	Valid
enrico	5873	milano	1961-03-21	[2005-03-01 - 9999-12-31)
matteo	6542	torino	1963-07-04	[2005-03-01 - 9999-12-31)
marco	3463	como	1970-03-09	[2005-03-02 - 9999-12-31)

Tabella A.3: Tabella `impiegato` in data 2-3-2005.

numero	valore	Valid
6542	3200	[2005-03-01 - 9999-12-31)
5873	3630	[2005-03-01 - 9999-12-31)
3463	3400	[2005-03-02 - 9999-12-31)

Tabella A.4: Tabella `salario` in data 2.3.2005.

Se a questo punto si volesse valutare una qualsiasi interrogazione con codice compatibile con lo standard SQL, tutte le tuple di entrambe le tabelle verrebbero considerate, in quanto contengono l'istante corrente.

Semantica Sequenced

La semantica *sequenced* permette di eseguire un'operazione SQL su di una certa dimensione temporale istante per istante. Ad esempio, il vincolo di

chiave primaria con semantica sequenced verifica che non esistano tuple con valori nulli sugli attributi della chiave e che, istante per istante, non esistano due righe con lo stesso valore sugli attributi espressi nel vincolo.

Una interrogazione con semantica sequenced permette di ricavare dei dati validi in un certo istante utilizzando l'informazione della tabella valida nello stesso istante. Il risultato è ovviamente una tabella temporale, in quanto il risultato ottenuto deve essere riferito al momento in cui è stato calcolato.

Per utilizzare una semantica di tipo sequenced in SQL3 sul tempo di validità, è necessario utilizzare la parola chiave `VALIDTIME`. Questa semantica può essere applicata, oltre che ad operazioni di modifica ed interrogazione, anche ad asserzioni, vincoli e viste.

Esempio. Ritornando all'esempio della sezione precedente di suppone di eseguire il seguente codice nel linguaggio SQL3:

```
VALIDTIME SELECT
    nome, valore
FROM
    salario AS s, impiegto AS e
WHERE
    s.numero = e.numero
```

L'interrogazione associa al nome di ogni dipendente il suo salario ed il periodo in cui questo è stato percepito. Il risultato è riportato nella Tabella A.5.

nome	valore	Valid
enrico	3200	[2005-02-01 - 9999-12-31)
matteo	3630	[2005-02-01 - 9999-12-31)
marco	3400	[2005-03-02 - 9999-12-31)

Tabella A.5: Esempio interrogazione con semantica sequenced in SQL3.

Quando un'interrogazione è eseguita con semantica sequenced anche le interrogazioni annidate hanno la stessa semantica. Osserviamo il seguente codice:

```
VALIDTIME SELECT
    nome
FROM
    impiegato AS e1, salario AS s1
```

```
WHERE
    e1.numero = s1.numero AND
    NOT EXISTS (
        SELECT nome
        FROM impiegato AS e2, salario AS s2
        WHERE e2.numero = s2.numero AND
            s2.valore > s1.valore AND
            e1.citta <> e2.citta
    );
```

Questa interrogazione chiede di elencare i dipendenti che non hanno colleghi di altre città con uno stipendio maggiore, su tutto l'arco temporale. Il risultato è riportato nella tabella A.6.

nome	Valid
enrico	[2005-02-01 - 2005-02-02)
marco	[2005-02-01 - 9999-12-31)

Tabella A.6: Esempio interrogazione annidata con semantica sequenced in SQL3.

Come precedentemente riportato, la semantica sequenced non è applicata solo alle interrogazioni. I seguenti esempi riportano alcuni casi di tale semantica applicata ad operazioni di modifica e a vincoli.

```
VALIDTIME DELETE FROM impiegato
    WHERE numero = 5873

VALIDTIME DELETE FROM salario
    WHERE numero = 5873
```

Le operazioni di cancellazione hanno eliminato il dipendente `5873` istante per istante. A seguito delle modifiche, la situazione della base di dati è riportata nelle Tabelle A.7 e A.8.

nome	numero	citta	dataNascita	Valid
enrico	6542	milano	1963-07-04	[2005-02-01 - 9999-12-31)
marco	3463	como	1970-03-09	[2005-03-02 - 9999-12-31)

Tabella A.7: Situazione della base di dati dopo cancellazione con semantica sequenced: tabella `impiegato`.

Il codice seguente mostra l'utilizzo del time slice in un'interrogazione con semantica sequenced.

numero	valore	Valid
6542	3200	[2005-03-01 - 9999-12-31)
3463	3400	[2005-03-02 - 9999-12-31)

Tabella A.8: Situazione della base di dati dopo cancellazione con semantica sequenced: tabella `salario`.

```
VALIDTIME PERIOD '[2005-01-01 - 2005-07-01)'
SELECT
    nome
FROM
    impiegato;
```

Il risultato dell'interrogazione è riportato nella Tabella A.9.

nome	Valid
enrico	[2005-03-01 - 2005-07-01)
marco	[2005-03-02 - 2005-07-01)

Tabella A.9: Esempio di interrogazione con semantica sequenced e time slice.

Semantica Non-Sequenced

La semantica di tipo *non-sequenced* è molto più complessa di quella sequenced, ma è indispensabile per alcune tipologie di operazioni temporali.

La complessità deriva dal fatto che l'utente ha la possibilità di manipolare contemporaneamente tutta l'informazione temporale presente nella base di dati senza usufruire di strumenti che possano facilitare alcune operazioni, come invece accade eseguendo operazioni con semantica sequenced.

Il linguaggio SQL3 fornisce diversi operatori di confronto per la gestione di tipi di dato temporale come istanti, intervalli e periodi.

Per indicare l'utilizzo di questa semantica, nel caso della dimensione temporale del tempo di validità, si utilizzano le parole chiave `NONSEQUENCED VALIDTIME`.

Un'interrogazione con semantica non-sequenced restituisce una tabella atemporale, tranne nel caso in cui l'utente specifichi come calcolare il periodo da associare alla dimensione temporale.

Esempio. Si suppone di riprendere l'esempio trattato nella sezione precedente: il primo aprile del 2005 il dipendente 3463 ottiene un incremento del salario pari al 5%. La situazione aggiornata è riportata nella Tabella A.10.

numero	valore	Valid
6542	3200	[2005-02-01 - 9999-12-31)
3463	3400	[2005-02-02 - 2005-04-01)
3463	3570	[2005-04-01 - 9999-12-31)

Tabella A.10: Situazione della base di dati dopo l'aumento di stipendio del dipendente 3463.

Per determinare quali dipendenti hanno avuto un incremento, è necessario utilizzare un'interrogazione con semantica non-sequenced, visto che devono essere considerate informazioni appartenenti a stati diversi. Il codice relativo è riportato di seguito.

```
NONSEQUENCED VALIDTIME SELECT
    ename
FROM
    impiegato AS e, salario AS s1, salario AS s2
WHERE
    e.numero = s1.numero AND
    e.numero = s2.numero AND
    s1.valore > s2.valore AND
    VALIDTIME(s1) MEETS VALIDTIME(s2);
```

Nell'esempio precedente è stato utilizzato un operatore di confronto tra periodi, (vedere Sezione 2.2.3), ed ha la funzione di reperire il periodo di validità di una relazione temporale (`VALIDTIME()`). Il risultato dell'esecuzione del codice è riportato nella Tabella A.11.

nome
marco

Tabella A.11: Esempio di interrogazione con semantica non-sequenced.

Per ottenere come risultato una tabella temporale è necessario specificare come calcolare il periodo di validità da associare ad ogni tupla risultante. Il seguente codice mostra un esempio di questo tipo di operazione.

```
NONSEQUENCED VALIDTIME valid
SELECT
    nome,
```

```
    VALIDTIME(s2) AS valid
FROM
    impiegato AS e, salario AS s1, salario AS s2
WHERE
    e.numero = s1.numero AND
    e.numero = s2.numero AND
    s1.valore > s2.valore AND
    VALIDTIME(s1) MEETS VALIDTIME(s2);
```

Il risultato dell'interrogazione è riportato nella Tabella A.12.

nome	Valid
marco	[2005-04-01 - 9999-12-31)

Tabella A.12: Esempio di interrogazione con semantica non-sequenced con risultato temporale.

A.0.7 Tabelle con tempo di transazione e bitemporali in SQL3

La dimensione temporale del tempo di transazione è semanticamente molto diversa da quella del tempo di validità. Non è possibile infatti che questa dimensione sia gestita esplicitamente dall'utente, visto che la semantica ad essa associata è di tipo *append-only*, ossia non è possibile modificare informazioni presenti nella base di dati, ma solo aggiungerne di nuove. Per questo motivo tutte le operazioni di inserimento, cancellazione e modifica devono essere opportunamente gestite dal DBMS per avere la sicurezza della consistenza delle informazioni.

Anche nel caso del tempo di transazione si sono implementati i requisiti di upward compatibility, temporal upward compatibility, di semantica sequenced e di semantica non-sequenced. Essendo questa dimensione temporale ortogonale rispetto al tempo di validità, è possibile scegliere una semantica diversa da applicare ad ogni dimensione temporale.

Per la gestione del tempo di transazione si utilizza una sintassi molto simile a quella relativa alla gestione del tempo di validità. L'unica differenza riguarda la parola chiave che è `TRANSACTIONTIME` nel caso di semantica sequenced e `NONSEQUENCED TRANSACTIONTIME` nel caso di semantica non-sequenced.

Altra differenza fondamentale riguarda le interrogazioni con semantica non-sequenced. Nel caso del tempo di validità è possibile specificare come calcolare il periodo di validità da associare ad ogni tupla se si vuole ottenere

una relazione temporale. Questo non è possibile con il tempo di transazione perchè, tale dimensione temporale è gestita totalmente dal sistema.

Esempio. Si consideri l'istanza delle relazioni *impiegato* e *salario* della Tabella A.13.

nome	numero	via	citta	dataNascita	VT
pietro	6542	pascoli	verona	1963-08-22	[2005-02-01 - 9999-12-31)
pietro	6542	pascoli	verona	1963-08-22	[2006-01-01 - 9999-12-31)
giovanni	3463	carducci	milano	1979-04-13	[2005-02-02 - 9999-12-31)

numero	valore	VT
6542	3200	[2005-02-01 - 2005-06-01]
6542	3360	[2005-06-01 - 2005-07-01]
6542	3360	[2006-01-01 - 9999-12-31)
3463	3400	[2005-02-02 - 2005-04-01]
3463	3570	[2005-04-01 - 9999-12-31)

Tabella A.13: Istanza base di dati: relazione impiegato e salario

Si desidera aggiungere alla relazione *impiegato* la dimensione temporale del tempo di transazione. Si assuma che la data corrente sia il 01-07-2005.

```
ALTER TABLE impiegato ADD TRANSACTIONTIME;
```

Essendo la tabella *impiegato* già una tabella temporale che supporta il tempo di validità, essa diventa una relazione bitemporale come mostrato nella Tabella A.14.

nome	numero	via	citta	dataNascita	VT	TT
pietro	6542	pascoli	verona	1963-08-22	[2005-02-01 - 2005-07-01]	[2005-07-01 - 9999-12-31)
pietro	6542	pascoli	verona	1963-08-22	[2006-01-01 - 9999-12-31]	[2005-07-01 - 9999-12-31)
giovanni	3463	carducci	milano	1979-04-13	[2005-02-02 - 9999-12-31]	[2005-07-01 - 9999-12-31)

Tabella A.14: Istanza base di dati. Relazione impiegato e salario

Appendice B

Tipi di dato temporale in SQL92

In questa appendice sono descritti i tipi di dato temporali forniti dallo standard SQL92.

Attualmente lo standard SQL92 è da considerarsi il linguaggio più utilizzato per basi di dati relazionali commerciali. I tipi di dato temporale forniti da tale linguaggio sono implementati, a meno di piccole modifiche di sintassi, da quasi tutti i sistemi commerciali di basi di dati.

Il linguaggio SQL92 non prevedere la possibilità di definire periodi (intesi come un insieme di istanti consecutivi), di utilizzare e definire sistemi di calendari, di definire tipi di dato indeterminato. L'unico calendario fornito si basa sul calendario gregoriano mentre i tipi di dato temporale implementati sono `DATE` [B.1], `TIME` [B.2], `TIMESTAMP` [B.3] e `INTERVAL` [B.5] che sono descritti di seguito.

B.1 Il tipo `DATE`

Il tipo di dato SQL92 `DATE` permette di memorizzare l'anno, il mese e il giorno di un istante.

Il range dell'anno di un tipo di dato `DATE` è delimitato dai valori 1 A.D e 9999 A.D.

Il range del mese di un tipo di dato `DATE` è delimitato dai valori 1 e 12. Il riferimento è al calendario gregoriano secondo il quale al valore 1 si associa il mese di gennaio, al valore 2 il mese di febbraio e così fino al valore 12 relativo al mese di dicembre.

Il range del giorno di un tipo di dato `DATE` non è fisso, come per i campi precedentemente descritti, ma varia in funzione del mese a cui è riferito. L'estremo inferiore del range è sempre il valore 1, mentre l'estremo superiore

può assumere i valori 28, 29, 30 o 31. Ad esempio, il range del giorno riferito al mese di gennaio è [1,31][1].

Per definire una costante `DATE` si utilizza la seguente sintassi:

```
DATE 'yyyy-mm-gg'
```

dove:

- `yyyy` specifica il valore dell'anno;
- `mm` specifica il valore del mese;
- `gg` specifica il valore del giorno.

Il 31 marzo del 1976 è rappresentato dalla costante `DATE '1976-03-31'`.

La variabile `CURRENT_DATE` dello standard SQL92 si riferisce alla data corrente.

B.2 Il tipo `TIME`

Il tipo di dato SQL92 `TIME` permette di memorizzare l'ora, i minuti, i secondi ed una opzionale frazione di secondo di un istante in funzione dello standard UTC [2]. Il tipo di default non prevede frazioni di secondo ed è identificato dalla stringa `TIME`, mentre per poter memorizzare anche frazioni di secondo si utilizza la sintassi `TIME(n)`, dove n è il numero di digit della frazione di secondo. Il tipo `TIME` coincide con il tipo di dato `TIME(0)`.

Ad esempio, per memorizzare anche i millisecondi è necessario definire un attributo di tipo `TIME(3)`.

Il range dell'ora di un tipo di dato `TIME` è delimitato dai valori 0 e 23.

Il range dei minuti di un tipo di dato `TIME` è delimitato dai valori 0 e 59.

Il range dei secondi di un tipo di dato `TIME` è delimitato dai valori 0 e 61[3].

Per definire una costante `TIME` si utilizza la seguente sintassi:

```
TIME 'hh:mm:ss[.f][4]'
```

dove:

- `hh` specifica il valore dell'ora;
- `mm` specifica il valore dei minuti;

[1] Con la notazione `[a,b]` si intende l'insieme di istanti consecutivi compresi tra gli istanti `a` e `b` inclusi.

[2] Coordinated Universal Time.

[3] È necessario considerare anche il leap second.

[4] [] denota un valore opzionale.

- `ss` specifica il valore dei secondi;
- `f` specifica il valore opzionale della frazione di secondo la cui lunghezza è variabile. Il valore è non negativo ed il suo massimo dipende dall'implementazione.

Esempio: `TIME '14:09:15.123'`.

La variabile `CURRENT_TIME` dello standard SQL92 si riferisce all'ora corrente.

B.3 Il tipo `TIMESTAMP`

Il tipo di dato SQL92 `TIMESTAMP` permette di memorizzare l'informazione contenuta nel tipo `DATE` e `TIME` contemporaneamente.

Come per il tipo `TIME` è possible memorizzare anche frazioni di secondo, fornendo il numero di digit al momento della definizione, secondo la sintassi `TIMESTAMP(n)`. Il valore di default è 6, quindi è possible memorizzare frazioni di secondo fino ai microsecondi. Il tipo `TIMESTAMP` coincide con il tipo di dato `TIMESTAMP(6)`.

Per definire una costante `TIMESTAMP` si utilizza la seguente sintassi:

```
TIMESTAMP 'yyyy-mm-gg hh:nn:ss[.f]'
```

dove:

- `yyyy` specifica il valore dell'anno;
- `mm` specifica il valore del mese;
- `gg` specifica il valore del giorno;
- `hh` specifica il valore dell'ora;
- `nn` specifica il valore dei minuti;
- `ss` specifica il valore dei secondi;
- `f` specifica il valore opzionale della frazione di secondo la cui lunghezza è variabile. Il valore è non negativo ed il suo massimo dipende dall'implementazione.

Esempio: `TIMESTAMP '1976-03-31 14:09:15.123'`.

La variabile `CURRENT_TIMESTAMP` dello standard SQL92 si riferisce all'ora e alla data corrente.

B.4 Il tipo TIMESTAMP WITH TIME ZONE e TIME WITH TIME ZONE

Il parametro Time Zone descrive la distanza, espressa in ore e minuti, dal meridiano di Greenwich. Il range di questo valore è compreso tra -12:59 e +13:00.

Il Time Zone è memorizzato con in tipi SQL92 `TIME` e `TIMESTAMP` implicitamente in funzione delle impostazione del DBMS. Per esprimere esplicitamente il Time Zone si utilizzano i tipi `TIMESTAMP WITH TIME ZONE` e `TIME WITH TIME ZONE`.

Per definire una costante `TIMESTAMP WITH TIME ZONE` si utilizza la seguente sintassi:

```
TIMESTAMP 'yyyy-mm-gg hh:nn:ss[.f] - HH:MM'
```

dove:

- `yyyy` specifica il valore dell'anno;
- `mm` specifica il valore del mese;
- `gg` specifica il valore del giorno;
- `hh` specifica il valore dell'ora;
- `nn` specifica il valore dei minuti;
- `ss` specifica il valore dei secondi;
- `f` specifica il valore opzionale della frazione di secondo la cui lunghezza è variabile. Il valore è non negativo ed il suo massimo dipende dall'implementazione.
- `HH` specifica il valore dell'ora della Time Zone;
- `MM` specifica il valore dei minuti della Time Zone.

Esempio: `TIMESTAMP '1976-03-31 14:09:15.123 - 07:00'`.

Per definire una costante `TIME WITH TIME ZONE` si utilizza la seguente sintassi:

```
TIME 'hh:mm:ss[.f] - HH:MM'
```

dove:

- `hh` specifica il valore dell'ora;
- `mm` specifica il valore dei minuti;

- `ss` specifica il valore dei secondi;
- `f` specifica il valore opzionale della frazione di secondo la cui lunghezza è variabile. Il valore è non negativo ed il suo massimo dipende dall'implementazione.
- `HH` specifica il valore dell'ora della Time Zone;
- `MM` specifica il valore dei minuti della Time Zone.

Esempio: `TIME '14:09:15.123 - 11:15'`.

B.5 Il tipo `INTERVAL`

Il tipo `INTERVAL` esprime una porzione continua non ancorata del tempo; in altre parole una durata temporale non riferita ad alcun istante dell'asse temporale.

In SQL92 esistono due tipi diversi di `INTERVAL`: *Year-Month Interval* e *Day-Time Interval*.

B.5.1 Year-Month Interval

L'intervallo *Year-Month* permette di memorizzare intervalli di granularità: anni (`INTERVAL YEAR`), mesi (`INTERVAL MONTH`) o anni e mesi (`INTERVAL YEAR TO MONTH`).

Come per i tipi `TIME` e `TIMESTAMP`, è possibile specificare anche il range dell'intervallo. Le definizioni alternative possibili sono:

- `INTERVAL YEAR(p)`
- `INTERVAL MONTH(p)`
- `INTERVAL YEAR(p) TO MONTH`

dove `p` esprime il numero di digit del tipo indicato. Il valore di default è 2 così `INTERVAL YEAR` o `INTERVAL YEAR(2)` identifica un tipo intervallo di granularità anno e con range pari a 100 anni.

Alcuni esempi di costanti sono:

```
INTERVAL '7' YEAR
INTERVAL '3' MONTH
INTERVAL '7-3' YEAR TO MONTH
```

Nel caso di un intervallo di tipo `INTERVAL YEAR TO MONTH` il valore assegnato ai mesi deve essere compreso tra 1 e 12.

Un intervallo può avere anche un segno per indicare durate positive o negative rispetto allo scorrere del tempo. Per indicare il verso è possibile porre il segno prima della stringa tra gli apici, riportarlo tra gli apici oppure utilizzare entrambe le soluzioni. In quest'ultimo caso valgono le classiche regole algebriche di moltiplicazione dei segni. Di seguito sono elencati alcuni esempi:

```
INTERVAL '+7-3' YEAR TO MONTH =
INTERVAL +'7-3' YEAR TO MONTH =
INTERVAL +'+7-3' YEAR TO MONTH =
INTERVAL -'-7-3' YEAR TO MONTH =
INTERVAL '7-3' YEAR TO MONTH

INTERVAL '-7-3' YEAR TO MONTH =
INTERVAL -'7-3' YEAR TO MONTH =
INTERVAL -'+7-3' YEAR TO MONTH =
INTERVAL +'-7-3' YEAR TO MONTH
```

B.5.2 Day-Time Interval

L'intervallo *Day-Time* è un tipo di dato più complesso rispetto al precedente perchè permette di contenere alpiù quattro campi (giorni, ore, minuti, secondi) più un campo opzionale per indicare una frazione di secondo. Nella definizione del tipo tutti i campi compresi tra quello di testa e quello di coda sono inclusi; così il tipo `INTERVAL DAY TO SECOND` contiene tutti i quattro campi, mentre il tipo `INTERVAL DAY TO HOUR` ne contiene solo due ed il tipo `INTERVAL DAY TO DAY` o `INTERVAL DAY` solo uno.

Come descritto per gli intervalli Year-Month, anche per l'intervallo Day-Time è possibile specificare un range. Anche in questo caso il range si riferisce al campo di testa, così l'intervallo `INTERVAL DAY(4) TO SECOND` ha un range di 9999 giorni. Il valore di default è sempre 2.

La specifica del tipo si complica se si intende memorizzare anche frazioni di secondo. Per fare ciò dovremo definire un tipo di dato che ha come campo di coda `SECOND` al quale associare una precisione $p > 0$, per indicare il numero di digits della frazione di secondo. Un esempio è `INTERVAL DAY TO SECOND(3)`, che può contenere la costante `INTERVAL '8 23:45:12.123' DAY TO SECOND`. Nel caso che l'intervallo abbia come unico campo `SECOND`, è necessario assegnare una doppia precisione. Esempio: `INTERVAL SECOND(5, 3)`.

Per quanto riguarda il segno, tutte le considerazioni fatte per l'intervallo Day-Month valgono anche in questo caso.

B.6 Costruttori temporali

B.6.1 Costruttori per istanti

SQL92 fornisce sette tipi di costruttori che coinvolgono i datetimes (`DATE`, `TIME`, `TIMESTAMP`, `TIME WITH TIME ZONE` e `TIMESTAMP WITH TIME ZONE`).

- `DATE '1996-02-24' + INTERVAL '7' DAY` Questa espressione permette di costruire un istante di tipo `DATE` pari alla data spostata in avanti nel tempo di sette giorni. Questo tipo di costruttore può coinvolgere solo intervalli la cui granularità è contenuta nel tipo associato all'istante. Se cio non accade è generato un errore di tipo.
- `INTERVAL '7' DAY + DATE '1996-02-24'` Questa espressione mostra la commutatività dell'operatore.
- `DATE '1996-02-24' - INTERVAL '7' DAY` Questa espressione permette uno spostamento all'indietro nel tempo di sette giorni.
- `TIMESTAMP '1996-02-24 12:34:58' AT LOCAL` Questa espressione associa al timestamp un time zone pari a quello del DBMS.
- `TIMESTAMP '1996-02-24 12:34:58' AT TIME ZONE INTERVAL '-7:00' HOUR TO MINUTE` Questa espressione associa al timestamp un time zone pari alla costante di tipo intervallo.
- `CURRENT_DATE`, `CURRENT_TIME` e `CURRENT_TIMESTAMP` ritornano rispettivamente la data, l'ora ed la data ed ora corrente.
- `CAST('1996-02-24' AS DATE)` La funzione `CAST` permette di convertire un elemento di un qualche tipo in un altro tipo specificato. Nell'esempio si converte una stringa in una costante `DATE`. É possibile convertire il tipo `CHARACTER` in un datetimes, il tipo `TIME` in `TIMESTAMP`, associando alla data la data corrente, il tipo `TIMESTAMP` in un qualsiasi datetimes ed infine il tipo `DATE` in `TIMESTAMP`, ponendo a zero ore, minuti e secondi.
- `EXTRACT(TIMEZONE_HOUR FROM TIME '12:34:59 - 07:00')` La funzione `EXTRACT` permette di estrarre un particolare valore specificato in datetimes o intervalli come il giorno, l'ora, i minuti, i secondi ecc.. Il risultato è un valore intero.

B.6.2 Costruttori per intervalli

Alcuni esempi di costruttori per intervalli forniti dal linguaggio SQL92 sono riportati di seguito:

- `INTERVAL '3' DAY + INTERVAL '4' DAY` Questa espressione è valutata in `INTERVAL '7' DAY`. È possibile sommare o sottrarre intervalli anche con precisioni diverse. Il risultato sarà una costante con la minima precisione necessaria per non perdere informazioni.

- `(DATE '1996-02-24' - DATE '1996-01-01') DAY` Questa espressione restituisce l'intervallo compreso tra le due date con la granularità indicata.

- `INTERVAL '3' DAY * 4` Questa espressione restituisce un intervallo pari a `INTERVAL '12' DAY`. Se l'intervallo è composto da campi multipli, prima il valore è convertito nella granularità minore, poi è moltiplicato ed infine riconvertito nella granularità iniziale. È possibile eseguire anche una divisione la cui semantica non cambia.

- `- INTERVAL '3' DAY` Questa espressione restituisce l'intervallo `INTERVAL '-3' DAY`.

- `CAST(INTERVAL '3' DAY AS INTEGER)` La funzione `CAST` applicata ad un intervallo con campo singolo restituisce il valore indicato con tipo intero.

B.7 Il tipo periodo

Un periodo è un intervallo temporale ancorato, ossia un intervallo di cui è nota la locazione rispetto all'asse temporale. Il linguaggio SQL92 non implementa questo tipo di dato temporale, ma definisce un costrutto (`BETWEEN`) ed un operatore (`OVERLAPS`) che agiscono rispettivamente su due istanti, considerati come estremi di un periodo, e su due periodi.

L'operatore `p1 OVERLAPS p2` verifica se due periodi si sovrappongono. I periodi possono essere definiti utilizzando una delle due seguenti sintassi:

- `(t, int)` dove `t` indica un istante e `int` un intervallo. `t` sarà l'istante iniziale del periodo se `int` è positivo, altrimenti sarà l'istante finale.

- `(ts, te)` dove `ts` indica l'istante iniziale del periodo e `te` quello finale.

Il costrutto `t BETWEEN ts AND te` è equivalente a `t <= te AND t >= tb` oppure a `t` $\in$ `(ts, te)`.

Bibliografia

[1] A. Montanari C. Combi. Querying data with multiple temporal dimensions. *CAISE-02*, 2002.

[2] A. Montanari C. Combi. Data models with multiple temporal dimensions: Completing the picture. *Interlaken 6/6/2001*, 2001.

[3] R. Snodgrass C. Jensen. *Information Systems*. 1996.

[4] F. Moretto. Un linguaggio di interrogazione relazionale esteso per basi di dati temporali multidimensionali. Master's thesis, Università degli studi di Verona, 2002.

[5] C. S. Jensen A. Steiner R. Snodgrass, M.Böhlen. *Transitioning Temporal Support in TSQL2 to SQL3*. 1996.