5.2.4 Chiavi

Usando le nozioni di dipendenza funzionale e di chiusura di un insieme di dipendenze, si possono definire in modo formale le nozioni di *superchiave*, *chiave* e *attributo primo* di uno schema.

Definizione 5.8 Dato uno schema $R\langle T, F\rangle$, un insieme di attributi $W \subseteq T$ è una *superchiave* di R se $W \rightarrow T \in F^+$.

Definizione 5.9 Dato uno schema $R\langle T, F\rangle$, un insieme di attributi $W \subseteq T$ è una *chiave* di R se W è una superchiave e non esiste un sottoinsieme stretto di W che sia una superchiave di R.

Definizione 5.10 Dato uno schema $R\langle T, F\rangle$, un attributo $A \in T$ si dice *primo* se e solo se appartiene ad almeno una chiave, altrimenti si dice *non primo*.

Dato che in uno schema ci possono essere più chiavi, di solito ne viene scelta una (generalmente con il minimo numero di attributi), la *chiave primaria*, come identificatore delle ennuple delle istanze dello schema.

In generale, il problema di trovare tutte le chiavi di uno schema richiede un algoritmo di complessità esponenziale nel caso peggiore e il problema di sapere se un attributo è primo è NP-completo [LO78].

Come trovare tutte le chiavi

Vedremo più avanti come in alcuni casi occorra stabilire se un attributo di uno schema $R\langle T, F\rangle$ sia primo e, quindi, trovare tutte le chiavi di R.

Osserviamo per prima cosa che valgono le seguenti proprietà:

1. Se un attributo A di T non appare a destra di alcuna dipendenza in F, allora A appartiene ad ogni chiave di R (si veda l'Esercizio 3).
2. Se un attributo A di T appare a destra di qualche dipendenza in F, ma non appare a sinistra di alcuna dipendenza non banale, allora A non appartiene ad alcuna chiave.

Sia X l'insieme degli attributi che non appaiono a destra di alcuna dipendenza in F. Dalla proprietà (1), segue che se $X^+ = T$, allora X è una chiave di R ed è anche l'unica possibile.

Per esempio, sia $R\langle T, F\rangle$ con $T = \{A, B, C, D, E, G, H\}$ ed $F = \{BC \rightarrow AD, D \rightarrow E, CG \rightarrow B\}$. C, G e H non appaiono a destra delle dipendenze, quindi devono far parte di ogni chiave. Poiché $CGH^+ = T$, CGH è l'unica chiave di R.

Se invece $X^+ \neq T$, allora occorre aggiungere a X altri attributi. Per la proprietà (2), basta aggiungere quegli attributi W di T che appaiono a destra di qualche dipendenza e a sinistra di qualche altra, uno alla volta. Ad ogni passo occorre evitare di aggiungere attributi che siano già nella chiusura di X, poiché tali attributi sono chiaramente ridondanti, oppure attributi che producono un insieme X' che contiene una chiave trovata in precedenza. Poi si calcola la chiusura di ogni X', fino a che questa non coincide con T, il che garantisce che X' sia una chiave.

Per esempio, sia $R\langle T, F\rangle$ con $T = \{A, B, C, D, E, G\}$ ed $F = \{AB \rightarrow C, BC \rightarrow AD, D \rightarrow E, CG \rightarrow B\}$.

G non appare a destra delle dipendenza e $G^+ = \emptyset$. Aggiungendo un attributo di $W = \{A, B, C, D\}$ a G si trova che

$GA^+ = GA \neq T.$
$GB^+ = GB \neq T.$
$GC^+ = T.$ GC è una chiave di $R.$
$GD^+ = GDE \neq T.$

Si prova ad aggiungere a GA, GB e GD un altro attributo di W, considerando solo insiemi di attributi che non contengono la chiave GC, e si trova che

$GAB^+ = T.$ GAB è una chiave di $R.$
$GAD^+ = GADE \neq T.$
$GBD^+ = GBDE \neq T.$

Si prova infine ad aggiungere a GAD e GBD un altro attributo di W, ma non si trovano insiemi di attributi che non contengono le chiavi GC e GAB, e quindi si conclude che non ne esistono altre.

In generale, una soluzione si trova con il seguente algoritmo esponenziale che analizza tutti i "candidati", ovvero i sottinsiemi di T che potrebbero essere chiavi.

```
Algoritmo TROVA TUTTE LE CHIAVI
    input     R⟨T, F⟩
    output    Chiavi l'insieme di tutte le chiavi di R⟨T, F⟩
begin
    NoDes := T − ∪_{X→A∈F} A;
    SinDes := ∪_{X→A∈F} X ∩ ∪_{X→A∈F} A;
    Candidati := [NoDes::(SinDes)];
    Chiavi := [ ];
    while (Candidati non vuoto) do
        begin
            X::(Y) := first(Candidati);
            Candidati := rest(Candidati);
            if not some K in Chiavi with K ⊂ X
            then if X⁺ = T
                then Chiavi := Chiavi + X;
                else begin
                        A_1 ... A_n := Y − X⁺;
                        for i in 1..n do Candidati := Candidati
                                                  append [XA_i::(A_{i+1} ... A_n)]
                    end
        end
end
```

L'algoritmo memorizza in *Chiavi* le chiavi già trovate e nella lista *Candidati* i candidati ancora da analizzare.[3] Ogni candidato è un insieme di sottoinsiemi di T rappresentato in una forma compatta X::(Y), che denota tutti gli insiemi formati dagli attributi X uniti ad un qualsiasi sottoinsieme degli attributi Y. Ad esempio, AB::(CD) rappresenta $\{AB, ABC, ABD, ABCD\}$. Se $NoDes$ sono gli attributi che non appaiono a destra di nessuna dipendenza e $SinDes$ sono quelli che appaiono sia a sinistra che a destra, per le osservazioni precedenti tutte le chiavi appartengono all'insieme $NoDes$::$(SinDes)$, per cui inizialmente $Candidati = [NoDes$::$(SinDes)]$.

Gli insiemi in X::(Y) sono analizzati a partire da X. Se X è chiave, allora tutti gli altri insiemi in X::(Y) sono scartati. Altrimenti, poiché gli elementi di X^+ non possono apparire in una chiave che contiene X, dato $Y - X^+ = \{A_1 \ldots A_n\}$, si

3. Su una lista $L = [x_1, \ldots, x_n]$ si usano i seguenti operatori: first(L) che ritorna il primo elemento di una lista non vuota L; rest(L) che ritorna la lista non vuota L priva del primo elemento; $L + x_i$ che ritorna una lista i cui primi elementi sono quelli di L e l'ultimo x_i; L_1 append L_2 che ritorna una lista i cui primi elementi sono quelli di L_1 ed i successivi quelli di L_2.

mettono in $Candidati$ i nuovi candidati $XA_1::(A_2, \ldots, A_n)$, $XA_2::(A_3, \ldots, A_n)$, $\ldots$, XA_n::(), i quali coprono $(X::(A_1 \ldots A_n)) - \{X\}$.

Il test $X^+ = T$ assicura che X è superchiave. Per essere certi che X sia anche chiave, si controlla che X non contenga chiavi già trovate in precedenza. Le chiavi trovate in seguito invece non potranno mai essere contenute strettamente in X perché tutti i candidati analizzati dopo X avranno lunghezza maggiore o uguale. Questa invariante è assicurata mantenendo *Candidati* ordinata per lunghezze crescenti, inserendo i nuovi candidati sempre in coda alla lista, con l'operatore append.

Esemplifichiamo l'applicazione dell'algoritmo su $R\langle T, F\rangle$ con $T = \{A, B, C, D, E\}$ ed $F = \{AC \to B, C \to D, D \to E, ABD \to C, B \to E\}$, mostrando l'effetto su $Chiavi$ e $Candidati$ di ciascuna esecuzione del ciclo while, con le variabili inizializzate a

$$NoDes = A;\ Chiavi = [\,];\ SinDes = BCD;\ Candidati = [A::(BCD)]$$

1. $X::(Y)$:= first($Candidati$) $= A::(BCD)$;
   $Candidati$:= rest($Candidati$) = [ ]
   $X^+ = A^+ = A \Rightarrow A$ non chiave
   $Y - X^+ = BCD - A = BCD$ e quindi
   $Candidati := [\,] + AB::(CD) + AC::(D) + AD::()$
2. $X::(Y)$:= first($Candidati$) $= AB::(CD)$
   $Candidati$:= rest($Candidati$) = $[AC::(D), AD::()]$
   $X^+ = AB^+ = ABE \Rightarrow AB$ non chiave
   $Y - X^+ = CD - ABE = CD$ e quindi
   $Candidati := [AC::(D), AD::()] + ABC::(D) + ABD::()$
3. $X::(Y)$:= first($Candidati$) $= AC::(D)$
   $Candidati$:= rest($Candidati$) = $[AD::(), ABC::(D), ABD::()]$
   $X^+ = AC^+ = ACBDE \Rightarrow AC$ chiave
   $Chiavi := [AC]$;
4. $X::(Y)$:= first($Candidati$) $= AD::()$
   $Candidati$:= rest($Candidati$) = $[ABC::(D), ABD::()]$
   $X^+ = AD^+ = ADE \Rightarrow AD$ non chiave
   $Y = ()$, per cui non si genera nessun nuovo candidato.
5. $X::(Y)$:= first($Candidati$) $= ABC::(D)$
   $Candidati$:= rest($Candidati$) = $[ABD::()][\,]$
   esiste $AC \in Chiavi$ con $AC \subset ABC$, per cui $ABC::(D)$ è scartato
6. $X::(Y)$:= first($Candidati$) $= ABD::()$
   $Candidati$:= rest($Candidati$) = [ ]
   $X^+ = ABD^+ = ABDEC \Rightarrow ABD$ chiave
   $Chiavi := [AC, ABD]$;

   $Candidati$ è vuoto, per cui l'algoritmo si ferma e restituisce le due chiavi trovate.

5.2.5 Copertura di un insieme di dipendenze

Per operare su insiemi di dipendenze, è comodo portarli in una forma "minima". Per arrivare ad una definizione formale di minimalità, si introduce per prima cosa il concetto di *copertura*.

Definizione 5.11 Due insiemi di dipendenze F e G sugli attributi T di una relazione R sono *equivalenti*, $(F \equiv G)$, se e solo se $F^+ = G^+$. Se $F \equiv G$ allora F è una *copertura* di G e viceversa.