FONDAMENTI DI BASI DI DATI

Soluzione degli esercizi

Antonio Albano, Giorgio Ghelli, Renzo Orsini

25 novembre 2022

INDICE

Capitolo 1

SISTEMI PER BASI DI DATI

Esercizio 1.1
Discutere le differenze tra un sistema per la gestione di basi di dati e in un sistema di archiviazione.

Soluzione

Sistemi di archiviazione	Sistemi per basi di dati	Benefici
In generale ogni applicazione ha propri archivi, con dati parzialmente duplicati.	I dati sono organizzati per poter essere usati da diverse applicazioni.	Integrazione dei dati.
Gli archivi sono modificati in tempi diversi e copie dello stesso dato possono risultare diverse.	Gli aggiornamenti vengono visti subito da tutte le applicazioni, a causa della condivisione dei dati.	Consistenza dei dati.
Ogni applicazione deve garantire l'integrità dei dati.	I DBMS offrono meccanismi per il controllo centralizzato dell'integrità dei dati.	Integrità dei dati.
Cambiamenti nella definizione dei record devono essere riportati in ogni applicazione.	Con la descrizione centralizzata dei dati solo le viste logiche devono essere modificate.	Indipendenza logica.
Le modalità di accesso ai dati dipendono dalla loro organizzazione fisica che, in generale, va programmata.	Si può accedere ai dati indipendentemente dalla loro organizzazione fisica, che viene scelta fra le organizzazioni previste dal DBMS.	Indipendenza fisica.

Continua sulla prossima pagina

Continua dalla pagina precedente

Sistemi di archiviazione	Sistemi per basi di dati	Benefici
I programmi usano nomi diversi per riferirsi agli stessi dati.	I programmi usano i nomi definiti nello schema.	Standardizzazione dei dati.
I dati sono accessibili solo da programmi.	I dati sono accessibili da programmi e interattivamente.	Facilità d'uso dei dati.
La sicurezza può essere assicurata solo scrivendo opportuni programmi.	I DBMS offrono meccanismi per garantire la sicurezza dei dati.	Sicurezza dei dati.
Ogni applicazione deve definire le proprie procedure per proteggere i dati da malfunzionamenti.	I DBMS offrono meccanismi per la protezione dei dati da malfunzionamenti.	Affidabilità dei dati.
I dati sono usati in modo esclusivo da ogni applicazione.	I dati sono accessibili in modo concorrente.	Condivisione dei dati.

Esercizio 1.2
Elencare alcune domande (fra cinque e dieci) da fare ad un produttore di sistemi per la gestione di dati al fine di stabilire se il sistema che propone può essere classificato come un sistema per la gestione basi di dati centralizzate.

Soluzione
Un insieme di domande per un sistema relazionale potrebbe essere:

1. *Schema logico*: il sistema gestisce insiemi di dati e associazioni tra di loro? Quale modello dei dati adotta? Il sistema gestisce e rende accessibile un catalogo delle definizioni dei dati (meta-dati)? Cosa si può definire in uno schema? Si può modificare uno schema con la base di dati già creata?
2. *Schema fisico*: il sistema permette di definire l'organizzazione fisica dei dati in maniera dichiarativa? Quali strutture di memorizzazione si possono definire? Le applicazioni sono indipendenti da queste scelte?
3. *Schemi esterni*: il sistema consente di definire "viste" dei dati, calcolate o memorizzate? Quali possibilità sono previste per modificare la struttura logica dei dati accessibili dalle viste?
4. *Controllo dei dati*: il sistema permette di dichiarare nello schema dei vincoli di integrità e cura il mantenimento di tali vincoli?
5. *Accesso ai dati*: il sistema prevede un linguaggio di interrogazione e di generazione di rapporti? Il sistema permette di accedere ai dati da un programma?

In quali linguaggi può essere scritto un tale programma?

6. *Sicurezza*: il sistema permette di definire utenti e classi di utenti e di stabilire i diritti di accesso ai diversi dati delle diverse classi di utenti, e controlla poi in maniera inviolabile che gli utenti non eseguano operazioni non ammesse?
7. *Affidabilità*: il sistema permette di definire transazioni, cioè sequenze di azioni da eseguire atomicamente? Il sistema garantisce l'atomicità delle transazioni indipendentemente dal tipo di guasto? Il sistema garantisce la persistenza degli effetti delle transazioni terminate normalmente? Il sistema permettere il recupero dei dati in caso di malfunzionamento dei dispositivi di memorizzazione?
8. *Controllo della concorrenza*: il sistema permette l'esecuzione concorrente di transazioni? Il sistema garantisce la serializzabilità dell'esecuzione di transazioni concorrenti?
9. *Prestazioni*: qual'è il numero massimo di insiemi di dati definibili, di attributi per ciascun insieme, di indici per ciascun insieme, di utenti, di gruppi di utenti, di connessioni contemporanee che il sistema può gestire? Quali limiti esistono sulla quantità di dati memorizzabili nella base di dati? Che tipo di ottimizzazione effettua il sistema sulle interrogazioni che gli vengono proposte?
10. *Strumenti*: il sistema mette a disposizione strumenti che permettano all'amministratore di valutare la bontà dell'organizzazione, quali, ad esempio, strumenti per verificare la frequenza di accesso ai dati e l'efficenza di esecuzione delle transazioni? Il sistema mette a disposizione strumenti per i programmatori, ad esempio per costruire agevolmente applicazioni con un'interfaccia grafica?

Esercizio 1.3
Discutere vantaggi e svantaggi di un sistema per la gestione di basi di dati.

Soluzione
Un sistema per la gestione di basi di dati offre la possibilità di integrare i dati all'interno di una organizzazione, permettendone una gestione centralizzata, l'uso da parte di applicazioni diverse in maniera concorrente, con meccanismi che ne semplificano la manutenzione e la modificabilità. Inoltre, offre meccanismi per mantenere i dati consistenti attraverso vincoli d'integrità, per renderli sicuri attraverso un sofisticato meccanismo di permessi d'accesso e per proteggerli da malfunzionamenti del sistema informatico.

Per quello che riguarda gli svantaggi, il loro uso ha un impatto sulla struttura organizzativa del settore informatico e sulle risorse necessarie; richiede programmatori esperti di applicazioni per basi di dati, oppure l'affidarsi ad aziende esterne per sviluppare le applicazioni. Inoltre, costituendo una risorsa centraliz-

zata, può diventare un collo di bottiglia, oltre a causare rischi di interruzione dei servizi.

Esercizio 1.4
Spiegare la differenza tra i seguenti termini: base di dati e sistema per la gestione di basi di dati.

Soluzione
Una base di dati è una collezione di dati strutturati ed organizzati secondo specifiche regole.

Un sistema di gestione di basi di dati è un sistema software che permette di amministrare ed operare su una o più basi di dati, rispettando le regole previste per esse.

Esercizio 1.5
Discutere i concetti di indipendenza logica e fisica, confrontandoli con i concetti di modulo e tipo di dato astratto dei linguaggi di programmazione.

Soluzione
Per *indipendenza logica* si intende che le applicazioni che usano una base di dati sono generalmente indipendenti dalle modifiche della struttura logica dei dati. Nei sistemi relazionali questo si realizza attraverso il livello di vista logico della base di dati. Ad esempio, definendo opportune *viste* o *tabelle derivate* le applicazioni possono continuare ad operare su dei dati di cui si è modificata la struttura logica usando le viste invece che direttamente i dati.

Per *indipendenza fisica* si intende che le applicazioni sono indipendenti dalla organizzazione fisica dei dati su dispositivi di memorizzazione. Ad esempio, è possibile aggiungere o rimuovere indici senza che l'applicazione debba essere modificata.

Entrambi i tipi di indipendenza hanno lo scopo di isolare il più possibile le applicazioni dalle modifiche dei dati su cui esse operano. In questo senso hanno molti aspetti in comune sia con il meccanismo dei moduli che quello dei tipi di dati astratti in alcuni linguaggi di programmazione ad alto livello.

Nei *linguaggi di programmazione* si isola l'implementazione concreta, fisica, di uno o più tipi di dato dal loro uso, definendo un'interfaccia, cioè una descrizione delle operazioni previste su questi dati che si limita ai tipi dei parametri attesi dalle operazioni e ai tipi dei risultati che producono. L'interfaccia rimane la stessa anche quando le implementazioni concrete cambiano, in modo che il resto dei programmi sia indipendente da questo tipo di cambiamento.

Esercizio 1.6
Discutere le differenze tra il modello dei dati di un sistema di archiviazione e un sistema per basi di dati.

Soluzione
Con riferimento ai dati persistenti, memorizzati dai due tipi di sistemi in archivi (*file*) e in basi di dati, le principali differenze sono:

- un sistema di archiviazione usa un modello dei dati basato su collezioni di record o di oggetti, ma non prevede associazioni fra le collezioni, come accade invece nei sistemi per basi di dati;
- il modello dei dati di un sistema di archiviazione non prevede operatori sui dati di tipo algebrico, come accade invece negli attuali sistemi per basi di dati.

Esercizio 1.7
Discutere i compiti del programmatore delle applicazioni e dell'amministratore della base di dati.

Soluzione
L'amministratore partecipa al progetto della base di dati, ne cura la definizione e la messa a punto e stabilisce le modalità di accesso degli utenti.

Il programmatore delle applicazioni sviluppa i programmi per rendere utilizzabili i servizi del sistema informatico da parte degli utenti.

Esercizio 1.8
Quali delle seguenti affermazioni è vera?

1. Sistema informatico è un sinonimo di sistema informativo.
2. Un linguaggio di interrogazione richiede la conoscenza di un linguaggio di programmazione.
3. Per usare correttamente una base di dati l'utente (persona o programma) deve conoscere l'organizzazione fisica dei dati.
4. L'organizzazione fisica di una base di dati va programmata dall'amministratore della base di dati.
5. Schema logico, schema fisico e schema esterno sono sinonimi.
6. Per soddisfare le esigenze degli utenti delle applicazioni non occorre un linguaggio di programmazione.
7. Le transazioni nei sistemi per basi di dati hanno le stesse proprietà dei programmi nei linguaggi con archivi.
8. Per realizzare un sistema informatico il personale tecnico realizza innanzitutto applicazioni che usano basi di dati.
9. Il programmatore delle applicazioni decide quali sono i dati accessibili agli utenti.

Soluzione
Le affermazioni sono tutte *False*.

Capitolo 2

I MODELLI DEI DATI

Esercizio 2.1
Discutere le differenze tra le nozioni di tipo oggetto e di classe.

Soluzione
Un *tipo oggetto* definisce la struttura degli elementi di quel tipo: cioè i tipi dei componenti dello stato e dei messaggi a cui l'oggetto può rispondere.

Una *classe di oggetti* invece è un insieme variabile nel tempo di oggetti dello stesso tipo. Quindi è un dato, con operatori per inserire e rimuovere elementi dall'insieme, ed a cui possono essere associati dei vincoli d'integrità.

Esercizio 2.2
Discutere le differenze tra le nozioni di tipo oggetto definito per ereditarietà e di sottoclasse.

Soluzione
La definizione di un tipo oggetto T_1 per *ereditarietà* da un altro tipo oggetto T_2 fa diventare T_1 sottotipo di T_2. Gli oggetti di tipo T_1 hanno tutti i componenti e i metodi degli oggetti di tipo T_2, eventualmente con dei tipi ridefiniti, e possono avere componenti o metodi aggiuntivi definiti nel sottotipo.

Una *sottoclasse* C_1 di una classe C_2 è una collezione, variabile nel tempo, contenente degli elementi di tipo T_1 presenti anche in C_2 (dove sono visti con tipo T_2, supertipo di T_1). Si dice che C_1 è in relazione di inclusione con C_2.

Esercizio 2.3
Discutere i vincoli che si possono descrivere graficamente con il modello ad oggetti.

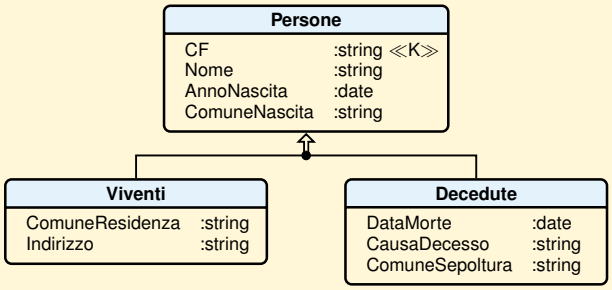
Soluzione
Si possono descrivere i vincoli d'integrità delle classi, i vincoli di totalità e di cardinalità delle associazioni, i vincoli di disgiunzione e di copertura fra sottoclassi.

Esercizio 2.4

Si definisca uno schema grafico per rappresentare con il modello a oggetti tre insiemi di entità: le persone e i sottoinsiemi delle persone viventi e delle persone decedute. Dare delle proprietà interessanti per gli elementi di questi insiemi.

Dire quali problemi si presenterebbero nel modellare questi insiemi se il modello non offrisse né il meccanismo dei tipi oggetti definiti per ereditarietà, né il meccanismo delle sottoclassi.

Soluzione



Non disponendo di meccanismi di ereditarietà e di sottoclassi si dovrebbero definire due classi distinte, Viventi e Deceduti, ognuna con tutti gli attributi di Persona oltre ai propri.

Oltre a complicare le operazioni sulla base di dati, questo avrebbe comportato i seguenti problemi:

- tutte le classi associate con Persone avrebbero dovuto avere anche due associazioni diverse con le classi Viventi e Deceduti;
- al momento del decesso di una persona si dovrebbe cancellare un elemento da una classe e inserirlo nell'altra. Questo comporterebbe anche la modifica di entrambe le associazioni per tutte le classi collegate alle persone, fatto che può provocare perdita di consistenza nel caso in cui si ometta qualche modifica.

Esercizio 2.5

Si vuole automatizzare il sistema di gestione degli animali in uno zoo.

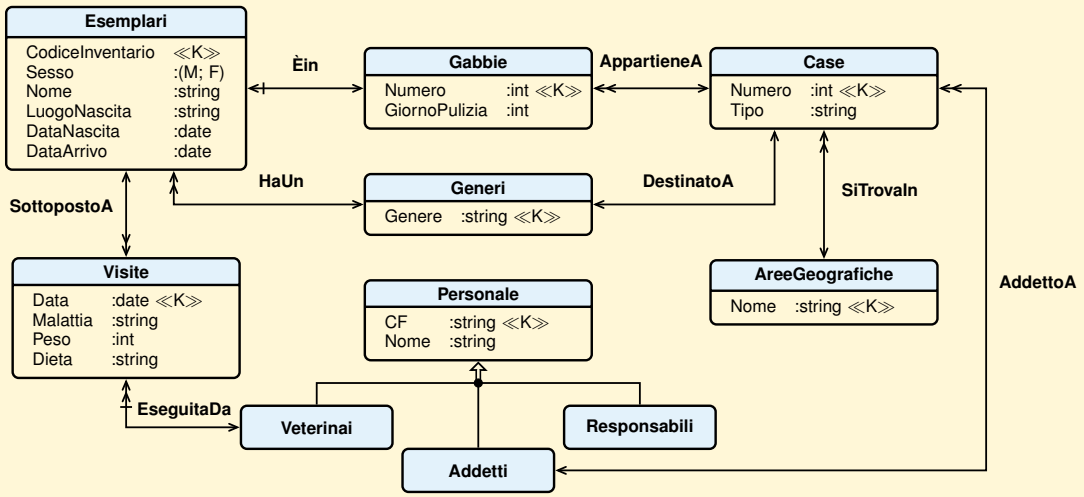
Di ogni *esemplare* di animale ospitato, identificato da un codice di inventario, interessano il *genere* (ad es., zebra), la data di arrivo nello zoo, il nome proprio, il sesso, il paese di provenienza e la data di nascita.

Lo zoo è diviso in *aree geografiche* di provenienza degli esemplari, con un nome, un responsabile e un insieme di *case*, ognuna destinata ad un solo genere di esemplari. Ogni casa è di un certo un tipo (ad es., recinto, tana, grotta, ecc.) e contiene un insieme di *gabbie* dove vive un solo esemplare. Ogni casa ha un addetto che pulisce ciascuna gabbia in un determinato giorno della settimana.

Gli animali sono sottoposti al loro arrivo, e poi periodicamente, a visite di un veterinario che controlla il peso degli esemplari, diagnostica un'eventuale malattia e prescrive il tipo di dieta da seguire.

Dare uno schema grafico della base di dati.

Soluzione



Esercizio 2.6

Una banca gestisce informazioni sui mutui dei propri clienti e le rate del piano di ammortamento.

Un cliente può avere più di un mutuo.

Ai clienti viene inviato periodicamente un resoconto sulle rate pagate del tipo mostrato in figura. Per le rate pagate dopo la data di scadenza è prevista una mora.

Si progetti la base di dati e successivamente si modifichi lo schema per trattare anche il fatto che i clienti fanno prima una richiesta di mutuo che poi *può essere concesso* con un rimborso a rate secondo un certo piano di ammortamento.

RESOCONTO MUTUO

Codice mutuo: 250 **Data**: 07/01/2018
Scadenza: 01/01/2020
Ammontare: 22 000,00
Codice cliente: 2000
Nome cliente: Mario Rossi
Indirizzo: Via Roma, 1 Pisa

Numero rata	Scadenza	Ammontare	Data Versamento
1	01/07/2018	6 000,00	29/06/2018
2	01/01/2019	6 000,00	30/12/2018
3	01/07/2019	6 100,00	29/06/2019
4	01/01/2020	6 100,00	30/12/2019

Soluzione
Il progetto concettuale iniziale è in Figura 2.1, mentro quello finale è in Figura 2.2.

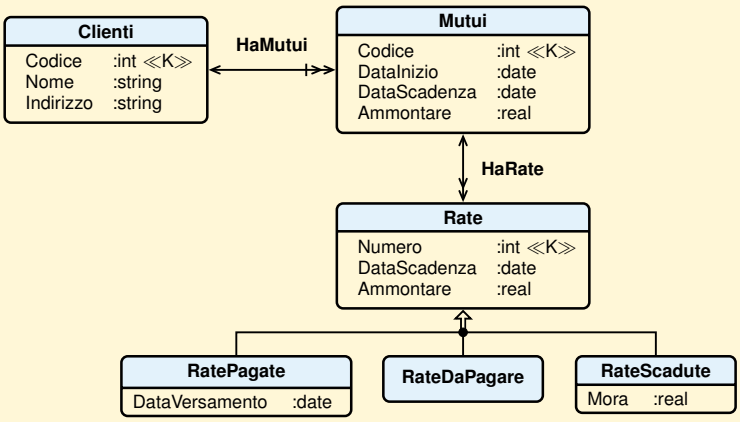


Figura 2.1: Gestione dei mutui: progetto concettuale iniziale

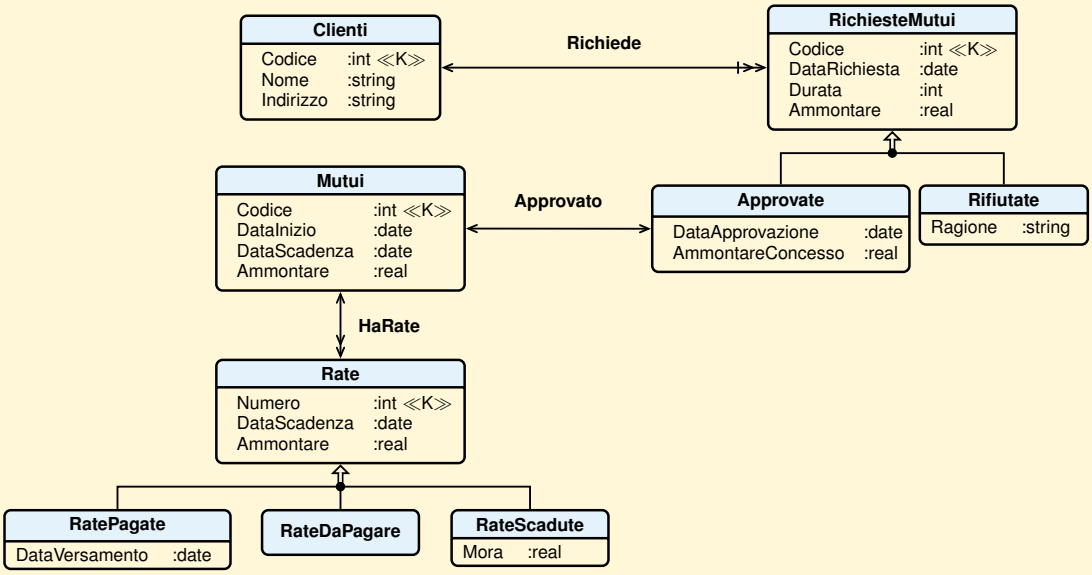


Figura 2.2: Gestione dei mutui: progetto concettuale finale

Esercizio 2.7

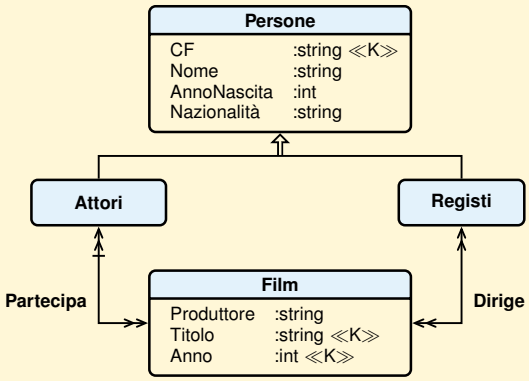
Si vogliono trattare informazioni su attori e registi di film.

Di un attore o un regista interessano il nome, che lo identifica, l'anno di nascita e la nazionalità. Un attore può essere anche un regista.

Di un film interessano il titolo, l'anno di produzione, gli attori, il regista e il produttore. Due film prodotti lo stesso anno hanno titolo diverso.

Dare uno schema grafico della base di dati.

Soluzione



Esercizio 2.8

Un'azienda vuole gestire le informazioni riguardanti gli impiegati, i dipartimenti e i progetti in corso.

Di un impiegato interessano il codice, assegnato dall'azienda, che l'identifica, il nome e cognome, l'anno di nascita, il sesso e i familiari a carico, dei quali interessano il nome, il sesso, la relazione di parentela e l'anno di nascita.

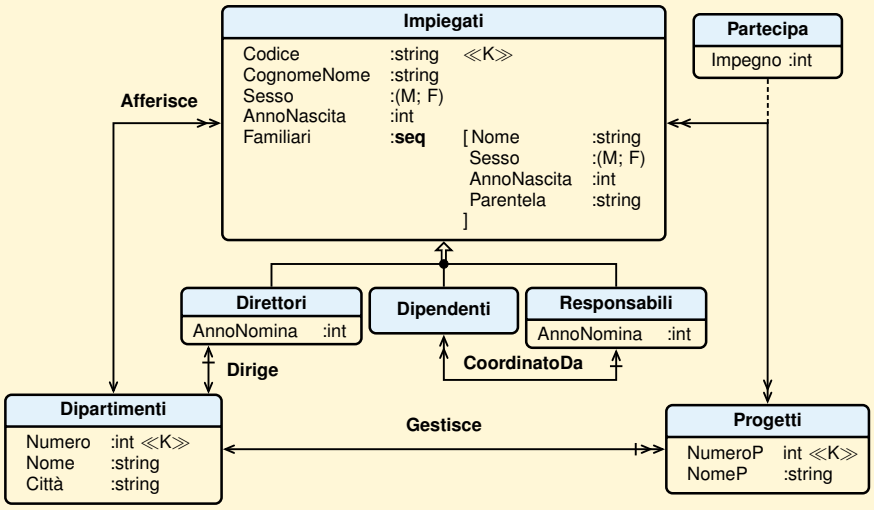
Di un dipartimento interessano il numero, che lo identifica, il nome, la città dove si trova.

Di un progetto interessano il numero, che lo identifica, e il codice. Un progetto è gestito da un solo dipartimento.

Gli impiegati afferiscono ad un dipartimento, che gestisce più progetti ed è diretto da un impiegato. Gli impiegati partecipano a più progetti, che si svolgono presso dipartimenti di città diverse, ad ognuno dei quali dedicano una percentuale del proprio tempo. Gli impiegati sono coordinati da un responsabile, che è un impiegato. Dei direttori e dei responsabili interessa l'anno di nomina.

Dare uno schema grafico della base di dati.

Soluzione


Impiegati
Codice :string ≪K≫
CognomeNome :string
Sesso :(M; F)
AnnoNascita :int
Familiari :seq [Nome :string
Sesso :(M; F)
AnnoNascita :int
Parentela :string
]
Partecipa
Impegno :int
Afferisce
Direttori
AnnoNomina :int
Dipendenti
Responsabili
AnnoNomina :int
Dirige
CoordinatoDa
Dipartimenti
Numero :int ≪K≫
Nome :string
Città :string
Gestisce
Progetti
NumeroP int ≪K≫
NomeP :string

Capitolo 3

LA PROGETTAZIONE DI BASI DI DATI

Esercizio 3.1
Condomìni

Si supponga di dover memorizzare in una base di dati le informazioni di interesse per un amministratore di condomìni. Di un condominio interessano l'indirizzo e il numero del conto corrente dove vengono fatti i versamenti per le spese sostenute. Un condominio si compone di un certo numero di appartamenti, dei quali interessano il numero dell'interno, il numero dei vani, la superficie.

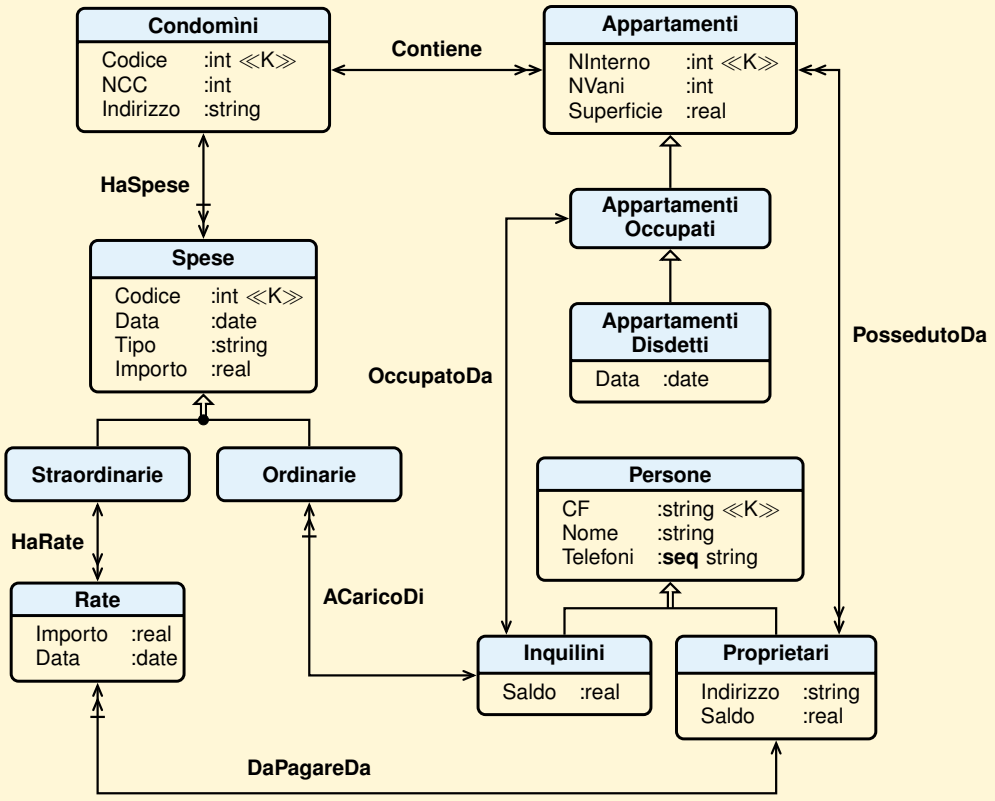
Gli appartamenti possono essere locati, e dell'inquilino interessano il nome, il codice fiscale, il telefono e il saldo, cioè la somma che l'inquilino deve all'amministrazione condominiale per le spese sostenute. Alcuni appartamenti locati possono essere stati disdetti, ed in questo caso interessa la data della disdetta.

Un appartamento può avere più proprietari, e un proprietario può possedere più appartamenti. Di ogni proprietario interessano il nome, il codice fiscale, l'indirizzo, il telefono e il saldo, cioè la somma che il proprietario deve all'amministrazione condominiale per le spese sostenute.

Le spese riguardano i condomìni e di esse interessano il codice di identificazione, la natura (luce, pulizia, ascensore ecc.), la data e l'importo. Fra le spese si distinguono quelle straordinarie, a carico dei proprietari, e quelle ordinarie, a carico degli inquilini. Le spese ordinarie vengono pagate in un'unica rata, mentre le spese straordinarie possono essere pagate in più rate e di ognuna di esse occorre ricordare la data e l'importo.

Progettare lo schema della base di dati e dare la specifica delle operazioni per l'immissione dei dati.

Soluzione



Riportiamo la lista delle operazioni di inserimento da prevedere, con una breve descrizione, e la specifica secondo lo schema dato nel libro di testo l'operazione di inserimento dei condomìni.

NuovaPersona
: Inserisce i dati di una persona

NuovoCondominio
: Inserisce un condominio e tutti i suoi appartamenti e i loro proprietari (che devono essere già inseriti come persone).

NuovoContrattoDiAffitto
: Inserisce un affittuario e lo collega all'appartamento affittato.

NuovaSpesaOrdinaria
: Inserisce i dati di una nuova spesa ordinaria e incrementa i saldi degli inquilini a cui si riferisce.

NuovaSpesaStraordinaria
: Inserisce i dati di una nuova spesa straordinaria, incluse le rate, e incrementa i relativi saldi dei proprietari.

Operazione	NuovoCondominio
Scopo	Immissione dei dati di un nuovo condominio
Argomenti	Codice :int, NCC :int, Indirizzo :string, AppartamentiPresenti :**seq** [NInterno :int; NVani :int; Superficie :real; CFProprietari :**seq** string];
Risultato	(OperazioneEseguita; Errore);
Errori	Numero vani minore di 1; Superficie errata; Codice già usato;
Usa	
Modifica	Condomìni, Appartamenti;
Prima	Codice $>$ 0; **not some** c **in** Condomìni **with** (c.Codice = Codice) **not some** a1 **in** AppartamentiPresenti **not some** a2 **in** AppartamentiPresenti **with** (a1.NInterno = a2.NInterno) **each** a **in** AppartamentiPresenti **each** cf **in** a1.CFProprietari **some** p **in** Persone **with** (p.CF = cf)
Poi	**some** c **in** Condomìni **with** (c.Codice = Codice); **each** a **in** AppartamentiPresenti **some** p **in** AppartieneA **with** (p.Condomìni.Codice = Codice) **and** (p.Appartamenti.NInterno = a.NInterno) **and some** d **in** PossedutoDa **with** (d.Proprietario.CF **in** a.CFProprietari)

Esercizio 3.2

Società Mega S.p.A.

Si vogliono gestire informazioni riguardanti gli impiegati, le loro competenze, i progetti a cui partecipano e i dipartimenti a cui appartengono.

Ogni impiegato ha una matricola che lo identifica, assegnata dalla società. Di ogni impiegato interessano il nome, la data di nascita e la data di assunzione. Se un impiegato è coniugato con un altro dipendente della stessa società, interessano la data del matrimonio e il coniuge. Ogni impiegato ha una qualifica (ad esempio, segretaria, impiegato, programmatore, analista, progettista ecc.). Dei laureati e delle segretarie interessano altre informazioni. Dei laureati interessa il tipo di laurea e delle segretarie le lingue straniere conosciute.

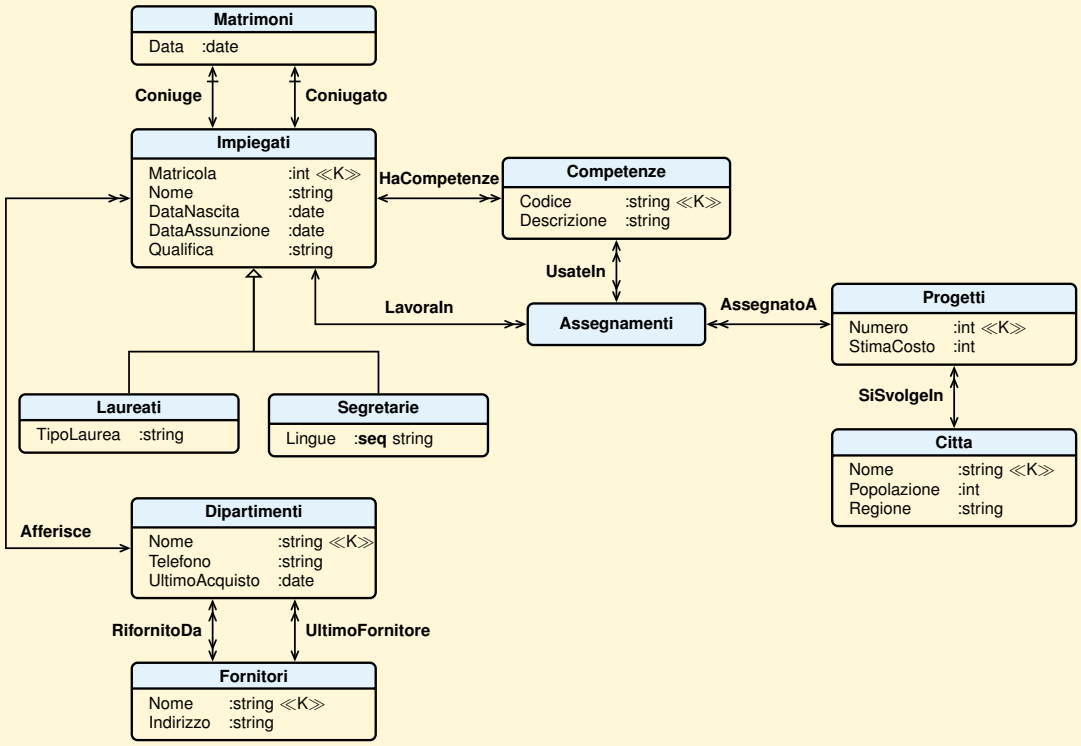
La società è organizzata in dipartimenti, caratterizzati da un nome e da un numero di telefono. Un impiegato afferisce ad un solo dipartimento. Ogni dipartimento si ap-

provvigiona presso vari fornitori e un fornitore può rifornire più dipartimenti. Di ogni fornitore interessano il nome e l'indirizzo. Interessano, inoltre, la data e il fornitore dell'ultimo acquisto fatto da un dipartimento.

La società lavora a diversi progetti, ciascuno dei quali è localizzato in una città. Più impiegati partecipano ad un progetto e un impiegato può partecipare a più progetti, ma tutti localizzati sulla stessa città. Di ogni città con un progetto in corso interessano la sua popolazione e la regione. Un impiegato può avere più competenze, ma usarne solo alcune per un particolare progetto. Un impiegato usa ogni sua competenza in almeno un progetto. Ad ogni competenza è assegnato un codice unico e una descrizione. I progetti in corso sono identificati da un numero ed interessa una stima del loro costo.

Progettare lo schema della base di datii.

Soluzione



È interessante elencare i vincoli non esprimibili graficamente nello schema:

1. Un'istanza della associazione UltimoFornitore deve essere anche istanza nella relazione RifornitoDa.
2. Un impiegato usa ogni sua competenza in almeno un progetto.
3. Tutti i progetti a cui partecipa un impiegato devono svolgersi nella stessa città.

Esercizio 3.3

Anagrafe

Si vogliono trattare informazioni sulle persone che vivono o sono decedute in un comune italiano.

Di una persona interessano: nome, cognome, codice fiscale, data di nascita (giorno, mese, anno), sesso (m, f) madre, e padre.

Una persona può essere vivente o deceduta. Di una persona vivente interessano: numero di cellulare, stato civile (celibe (nubile), coniugato(a), vedovo(a), separato(a), divorziato(a)), e i familiari conviventi. Le persone di un nucleo familiare condividono lo stesso indirizzo (via, numero, cap, località), telefono e comune di residenza.

Di una persona deceduta interessano: data del decesso, età al momento del decesso, comune del decesso, comune dove è stata seppellita.

Di un matrimonio interessano: data del matrimonio, le due persone che si sono unite in matrimonio e il comune dove è stato celebrato.

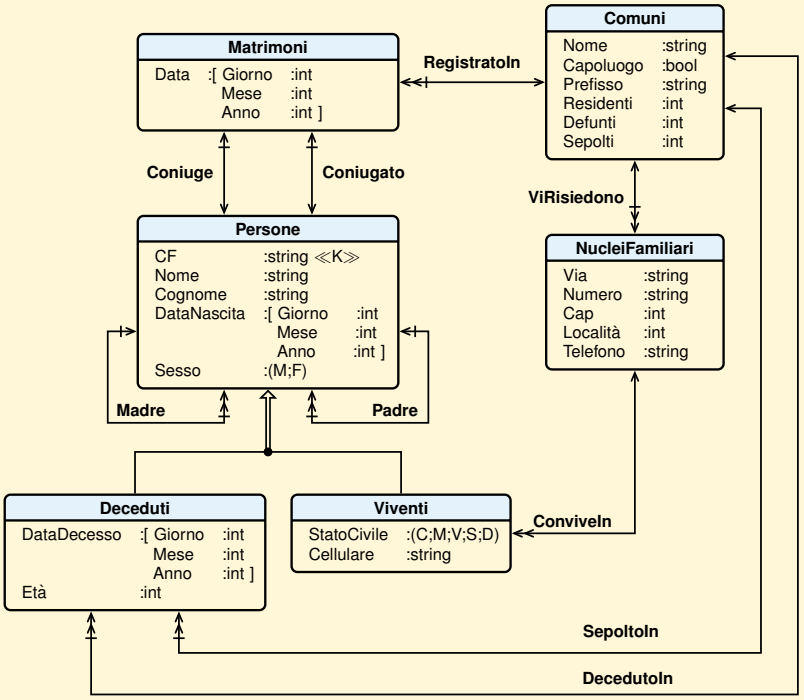
Di un comune interessano: nome, se capoluogo di provincia, prefisso telefonico, gli abitanti, il numero degli abitanti, il numero delle persone seppellite e decedute.

Vanno previste le seguenti operazioni; si usi questa lista per verificare che il modello della base di dati contenga tutti i dati necessari alla loro realizzazione.

a) creazione di una persona;
b) nascita di un figlio;
c) matrimonio;
d) antenati viventi di una persona;
e) nome e cognome dei genitori di una persona;
f) nome, cognome e relazione di parentela dei familiari conviventi di una persona;
g) figli viventi e figli conviventi di una persona;
h) cambio di residenza di una persona e dei suoi familiari conviventi;
i) una persona e i suoi conviventi vanno a vivere con un'altra persona;
j) una persona va a vivere da sola;
k) decesso di una persona.

Soluzione

Nota: i valori dello stato civile celibe (nubile), coniugato(a), vedovo(a), separato(a), divorziato(a) saranno abbreviati rispettivamente in C, M, V, S, D.



Esercizio 3.4

Ufficio della motorizzazione

Si vogliono modellare i seguenti fatti di interesse di un ipotetico Ufficio della motorizzazione.

Produttori di automobili

C'è un certo numero di produttori di automobili, ciascuno identificato da un nome (FIAT, FORD ecc.). I dati di nuovi produttori possono essere immessi in ogni momento, se il produttore ha l'autorizzazione ad iniziare l'attività commerciale.

L'autorizzazione non può essere ritirata e non più di cinque produttori possono essere in attività contemporaneamente. Un produttore è considerato attivo finché possiede automobili registrate come prodotte da lui e non ancora vendute; nel momento in cui un produttore non possiede auto, il suo permesso di operare può essere sospeso. I dati di un produttore vengono eliminati solo quando viene eliminata la storia di tutte le auto da lui prodotte.

Automobili

Un'automobile è caratterizzata da un modello, dall'anno di produzione, da un numero di serie assegnatogli dal produttore, unico fra le automobili da lui prodotte. I dati di un'automobile vengono immessi all'atto della sua registrazione presso l'Ufficio della Motorizzazione. Al momento della registrazione, all'automobile viene assegnato un numero, unico per ciascuna automobile e non modificabile, e la data di registrazione. Il produttore viene registrato come primo proprietario.

Un'automobile può essere registrata in qualsiasi giorno dell'anno in cui è stata costruita, ma può essere registrata solo entro il 31 gennaio se costruita l'anno precedente. Nel caso di distruzione, viene registrata la data di distruzione, e da questo momento l'automobile non può essere più trasferita. Infine, la storia di un'automobile va conservata per due anni dopo la sua distruzione.

Modelli di automobile

Ogni automobile è caratterizzata da un modello (Panda, Uno, Escort ecc.). Le automobili di ciascun modello sono prodotte dallo stesso produttore, il quale è libero di introdurre nuovi modelli sul mercato in qualsiasi momento. Il nome di ciascun modello è unico fra tutti i modelli registrati. Le automobili di uno stesso modello hanno lo stesso consumo di benzina. Un modello ha una potenza di almeno 6 cavalli e una cilindrata compresa fra 400 e 3.000 cc.

I dati su un modello vanno conservati finché esiste nella base di dati un'automobile di tale modello. Le automobili di un certo modello non possono essere registrate se tale modello non è ancora noto all'Ufficio della motorizzazione.

Rivenditori

I rivenditori sono preposti alla distribuzione di automobili nuove, o usate, ai privati. Di un rivenditore interessano il nome, l'indirizzo, il telefono e l'eventuale numero del fax. Nuovi rivenditori possono sorgere in ogni momento, ma la loro attività commerciale può iniziare solo se hanno ricevuto il permesso dagli uffici competenti. Un rivenditore può trattare automobili nuove di al più tre produttori diversi.

Ogni rivenditore è considerato operante finché possiede automobili; in caso contrario può richiedere la sospensione del permesso di operare. I dati di un rivenditore non operante vengono eliminati solo se questo non è stato proprietario di un'auto di cui si conserva la storia.

Privati

I privati sono persone proprietarie di una o più automobili già registrate. Di un privato interessano il nome, l'indirizzo e il telefono. I dati dei privati vengono immessi con l'acquisto della prima automobile, ed eliminati solo se essi non sono stati proprietari di un'automobile di cui si conserva la storia.

Trasferimenti di proprietà

In ogni momento un'automobile può essere posseduta: dal suo produttore (automobile invenduta), da un rivenditore, oppure da un gruppo di privati.

All'atto del trasferimento della proprietà di un'automobile vengono registrate le seguenti informazioni: un codice che identifica il trasferimento, la data di trasferimento, l'automobile trasferita, il vecchio e il nuovo proprietario.

Vi sono norme che vincolano il trasferimento di un'automobile:

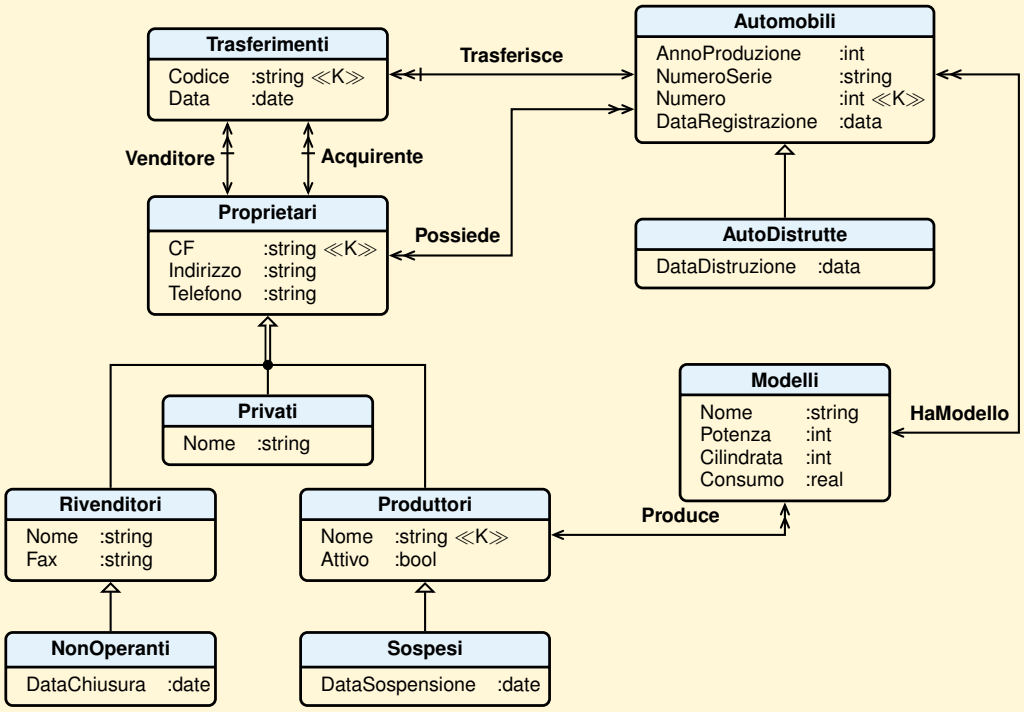
- un'automobile distrutta non può essere trasferita;
- un'automobile può essere venduta da un produttore solo ad un rivenditore, e un produttore non può acquistare automobili;
- un'automobile può essere venduta da un rivenditore solo a privati o gruppi di privati.

I dati su un trasferimento possono essere eliminati solo quando l'automobile cessa di essere di interesse per l'Ufficio della Motorizzazione.

Si usino le seguenti operazioni previste sul sistema per verificare che possono essere realizzate.

- Registrazione di una nuova auto.
- Distruzione di un'auto.
- Registrazione della vendita di un'auto.
- Eliminazione della storia delle auto distrutte da almeno due anni.
- Registrazione di un nuovo produttore in attesa del permesso di operare.
- Autorizzazione di un produttore ad operare.
- Sospensione delle attività di un produttore.
- Eliminazione di un produttore non operante.
- Registrazione di un nuovo modello.
- Registrazione di un nuovo rivenditore.
- Sospensione delle attività di un rivenditore.
- Eliminazione di un rivenditore non operante.

Soluzione



Esercizio 3.5

Organizzazione di una conferenza

Si vogliono trattare i dati riguardanti le *Working Conferences* dell'IFIP (*International Federation for Information Processing*).

L'IFIP è una organizzazione federativa che raggruppa 48 associazioni nazionali e accademie scientifiche che operano nell'area della tecnologia dell'informazione. Ogni membro ha un nome e il paese di appartenenza (ad es. *Associazione Italiana per l'Informatica ed il Calcolo Automatico*, Italia). Le attività dell'IFIP sono coordinate da 13 *Technical Committees (TC)*, a loro volta divisi in oltre 100 *Working Groups (WG)* ai quali appartengono oltre 3.500 esperti (professionisti ICT e ricercatori di tutto il mondo). Ogni *Technical Committee* copre un aspetto particolare dell'informatica e delle discipline correlate (ad es. *TC 1: Foundations of Computer Science*: *WG 1.1 Continuous Algorithms and Complexity*, *WG 1.2 Descriptional Complexity*, ecc.).

I *Working Groups* organizzano conferenze alle quali possono partecipare solo esperti membri dei *Working Groups* e *Technical Committees* che hanno ricevuto un invito.

Di un esperto interessano il codice, che lo identifica, il nome, la nazione, l'ente di appartenenza, l'email e il telefono.

Di una conferenza interessano la sigla, che la identifica, il titolo, l'anno, il luogo, le date di inizio e fine, la soglia minima e massima dei partecipanti ai lavori per garantirsi la copertura dei costi e per non superare le capacità ricettive delle strutture.

Delle sessioni di una conferenza interessano il titolo, la sala, la data e l'ora di inizio e fine.

La conferenza è organizzata da tre comitati di esperti:

1. il *Comitato Organizzatore* per curare gli aspetti finanziari, logistici, gli inviti e la pubblicità,
2. il *Comitato di Programma* per curare gli aspetti scientifici della conferenza;
3. il *Comitato dei Revisori*, nominato dal *Comitato di Programma*, per esaminare gli articoli sottoposti alla conferenza, e decidere quali articoli accettare, non superando il numero massimo prestabilito.

È previsto un *Chairman* per ogni comitato e un *General Chairman* per la conferenza. Tutti i comitati lavorano utilizzando dati comuni che vanno raccolti ed elaborati in modo consistente.

Procedure da automatizzare L'applicazione da realizzare ha lo scopo di agevolare le attività del *Comitato di Programma* e del *Comitato Organizzatore*, pensati come due settori aziendali che operano utilizzando dati in comune. I comitati devono svolgere le seguenti attività.

Comitato di Programma

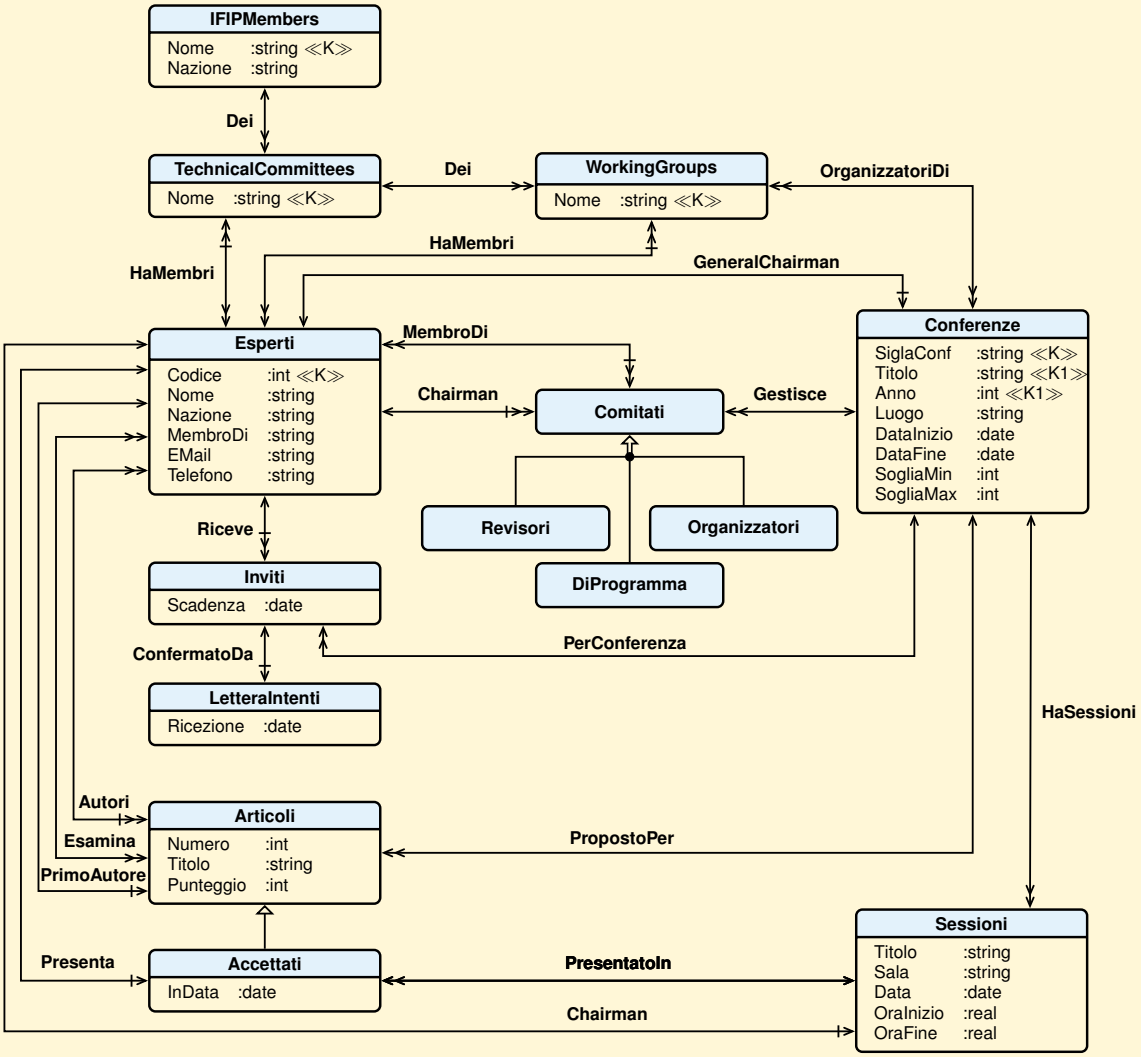
- Preparare la lista degli esperti a cui sollecitare la presentazione di un articolo.
- Registrare le lettere d'intenti, cioè le risposte date da coloro che intendono inviare un lavoro. Ogni esperto invia al più una lettera che verrà presa in considerazione solo se perviene entro la data prestabilita. I lavori con nessun autore fra gli esperti che hanno risposto con una lettera d'intenti avranno una priorità bassa, se il numero complessivo dei lavori da esaminare supera il massimo prestabilito.

Comitato Organizzatore

- Preparare la lista degli esperti da invitare alla conferenza. Nella lista devono essere inclusi: i membri di tutti i *Technical Committees* e *Working Groups* interessati; i membri del *Comitato dei Revisori* e tutti coloro che hanno proposto un articolo. Occorre evitare di mandare alla stessa persona più di un invito.
- Registrare le adesioni alla conferenza pervenute entro la data prefissata.
- Generare la lista finale dei partecipanti alla conferenza. Se le adesioni superano il numero massimo prestabilito, verrà data priorità, nell'ordine, ai membri dei *Technical Committees*, ai membri dei *Working Groups* e del *Comitato dei Revisori*, agli autori degli articoli ammessi alla conferenza, agli autori degli articoli rifiutati.
- Registrare gli articoli proposti per la conferenza e pervenuti entro una data prefissata.

– Distribuire i lavori fra i membri del *Comitato dei Revisori*. Ogni lavoro sarà revisionato da 3 a 5 revisori, a seconda del numero complessivo dei lavori pervenuti.
– Raccogliere i pareri dei revisori e selezionare il numero prefissato di lavori da presentare alla conferenza.
– Raggruppare i lavori selezionati in sessioni e scegliere un *Chairman* per ogni sessione fra i membri del *Comitato di Programma*.

Soluzione



Capitolo 4

IL MODELLO RELAZIONALE

Esercizio 4.1
Elencare le differenze tra la nozione di ennupla nei sistemi relazionali e la nozione di oggetto nei sistemi ad oggetti.

Soluzione
Le ennuple del modello relazionale hanno campi di tipo elementare, non hanno una nozione di identità, non hanno componenti procedurali come i metodi degli oggetti, non hanno nozioni di inclusione, eredità, incapsulazione. Dal punto di vista della modellazione, ne consegue, in particolare, che:

- nel modello relazionale non è possibile rappresentare le entità della realtà con un'unica ennupla se queste hanno una struttura complessa (ad esempio, se hanno attributi multivalore);
- nel modello relazionale le associazioni si rappresentano usando il meccanismo delle chiavi esterne;
- nel modello relazionale è necessario aggiungere un campo chiave per distinguere entità con gli stessi valori per tutti gli attributi.

Esercizio 4.2
Si traduca lo schema concettuale ottenuto dall'Esercizio 2.6 in uno schema relazionale.

Soluzione
In Figura 4.1 si ripete la soluzione dell'Esercizio 2.6. Lo schema relazionale è mostrato in Figura 4.2.

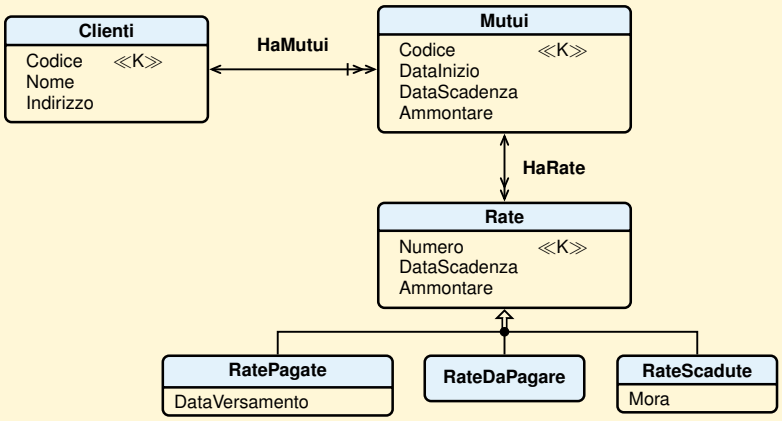


Figura 4.1: Schema concettuale

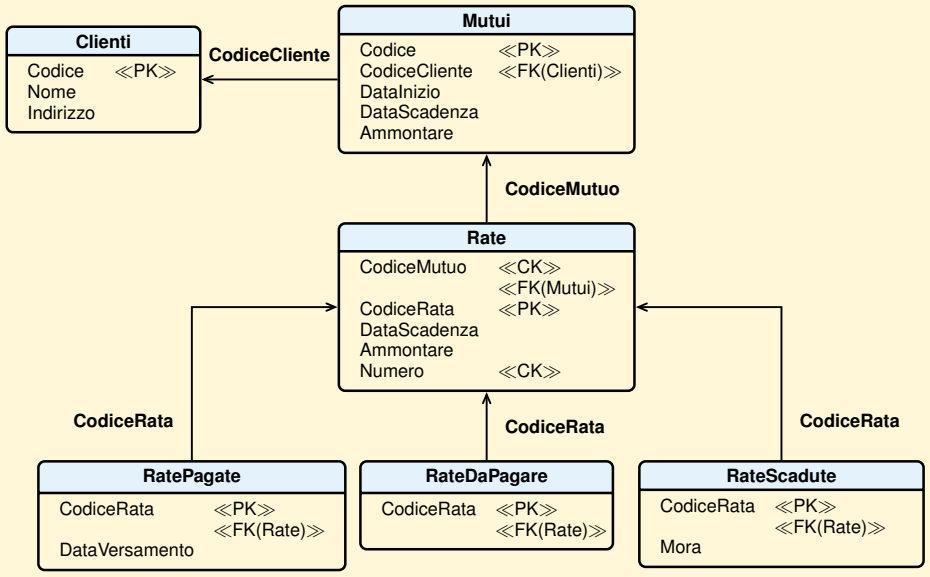


Figura 4.2: Schema relazionale

Esercizio 4.3
Si traduca lo schema concettuale ottenuto dall'Esercizio 2.8 in uno schema relazionale.

Soluzione

In Figura 4.3 si ripete la soluzione dell'Esercizio 2.8. Lo schema relazionale è mostrato in Figura 4.4.

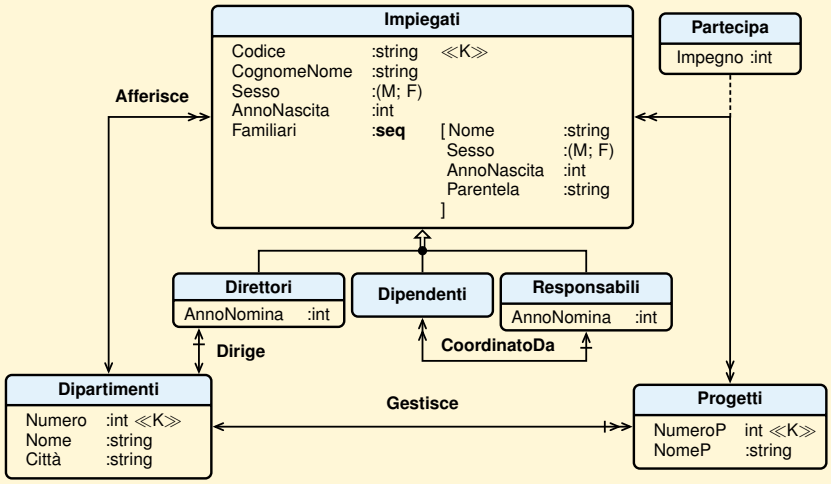


Figura 4.3: Schema concettuale

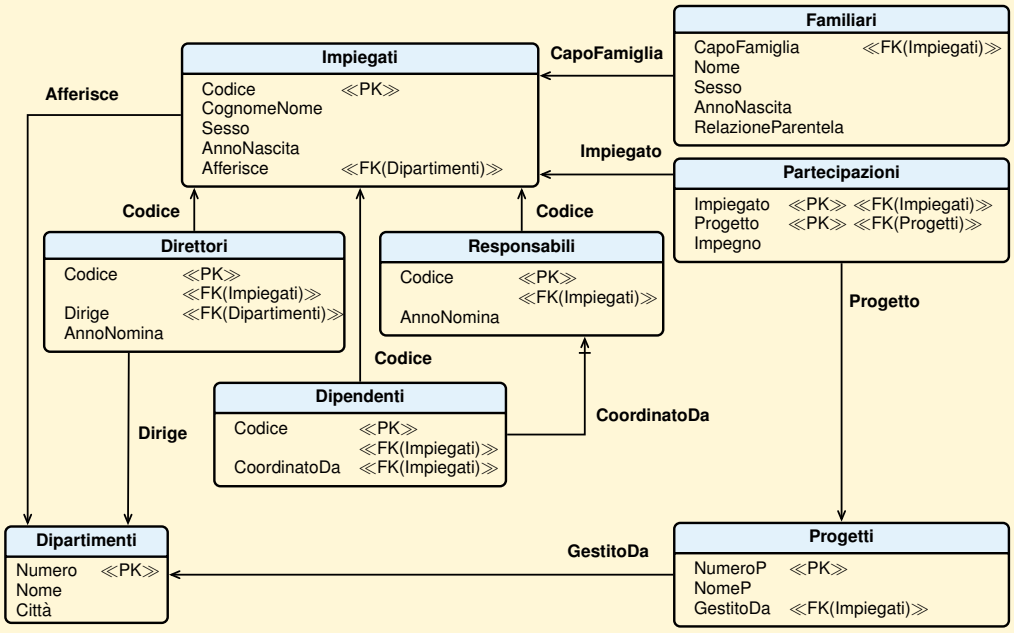


Figura 4.4: Schema relazionale

Esercizio 4.4
Dimostrare le seguenti proprietà:

a) $\sigma_{\phi \wedge \psi}(E) = \sigma_{\phi}(E) \cap \sigma_{\psi}(E)$;
b) $\sigma_{\phi}(\sigma_{\psi}(E)) = \sigma_{\phi \wedge \psi}(E)$;
c) $\sigma_{\phi}(E_1 \cup E_2) = \sigma_{\phi}(E_1) \cup \sigma_{\phi}(E_2)$;
d) $\pi_{Y}(\pi_{X}(E)) = \pi_{Y}(E)$, se $Y \subseteq X$;
e) $\pi_{X}(\sigma_{\phi}(E)) = \sigma_{\phi}(\pi_{X}(E))$, se ϕ usa solo attributi in X;
f) $\sigma_{\phi}(E_1 \underset{\phi_J}{\bowtie} E_2) = \sigma_{\phi}(E_1) \underset{\phi_J}{\bowtie} E_2$, se ϕ usa solo attributi di E_1;
g) $\sigma_{\phi}({}_{A}\gamma_{F}(E)) = {}_{A}\gamma_{F}(\sigma_{\phi}(E))$, se ϕ usa solo attributi in A.

Soluzione
Dimostriamo ciascuna proprietà $E_1 = E_2$ mostrando come, per ogni ennupla t, si ha $t \in E_1 \Leftrightarrow t \in E_2$.

Sfruttiamo le seguenti proprietà, che derivano immediatamente dalla definizione degli operatori relazionali; nell'ultima, usiamo $[A = v]$ per denotare un'ennupla con un solo attributo A con valore v.

$t \in \sigma_{\phi}(E) \Leftrightarrow t \in E \wedge \phi(t)$

$t \in \pi_{X}(E) \Leftrightarrow \exists t' \, (t' \in E \wedge t'[X] = t)$

$t \in (E_1 \underset{\phi}{\bowtie} E_2) \Leftrightarrow \exists t', t'' \, (t' \in E_1 \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi(t))$

$t \in {}_{X}\gamma_{f_1(B_1)\ \mathbf{AS}\ C_1, \ldots, f_n(B_n)\ \mathbf{AS}\ C_n}(E)$

$$\Leftrightarrow \exists t' \, (\, t' \in E \wedge t = t'[X] \circ [C_1 = f_1(\{s[B_1] \mid s \in E \wedge s[X] = t'[X]\})] \circ \ldots \circ [C_n = f_n(\{s[B_n] \mid s \in E \wedge s[X] = t'[X]\})])$$

a) $\sigma_{\phi \wedge \psi}(E) = \sigma_{\phi}(E) \cap \sigma_{\psi}(E)$.

$t \in \sigma_{\phi \wedge \psi}(E)$
$\Leftrightarrow t \in E \wedge \phi(t) \wedge \psi(t)$
$\Leftrightarrow (t \in E \wedge \phi(t)) \wedge (t \in E \wedge \psi(t))$
$\Leftrightarrow (t \in \sigma_{\phi}(E)) \wedge (t \in \sigma_{\psi}(E))$
$\Leftrightarrow t \in (\sigma_{\phi}(E) \cap \sigma_{\psi}(E))$

b) $\sigma_{\phi_1}(\sigma_{\phi_2}(E)) = \sigma_{\phi_1 \wedge \phi_2}(E)$.

$t \in \sigma_{\phi_1}(\sigma_{\phi_2}(E))$
$\Leftrightarrow t \in \sigma_{\phi_2}(E) \wedge \phi_1(t)$
$\Leftrightarrow (t \in E \wedge \phi_2(t)) \wedge \phi_1(t)$
$\Leftrightarrow t \in E \wedge (\phi_1(t) \wedge \phi_2(t))$
$\Leftrightarrow t \in \sigma_{\phi_1 \wedge \phi_2}(E)$

c) $\sigma_\phi(E_1 \cup E_2) = \sigma_\phi(E_1) \cup \sigma_\phi(E_2)$.

$t \in \sigma_\phi(E_1 \cup E_2)$
$\Leftrightarrow\ t \in (E_1 \cup E_2) \wedge \phi(t)$
$\Leftrightarrow\ ((t \in E_1) \vee (t \in E_2)) \wedge \phi(t)$
$\Leftrightarrow\ (t \in E_1 \wedge \phi(t)) \vee (t \in E_2 \wedge \phi(t))$
$\Leftrightarrow\ t \in \sigma_\phi(E_1) \vee t \in \sigma_\phi(E_2)$
$\Leftrightarrow\ t \in (\sigma_\phi(E_1) \cup \sigma_\phi(E_2))$

d) $\pi_Y(\pi_X(E)) = \pi_Y(E)$, se $Y \subseteq X$.

$t \in \pi_Y(\pi_X(E))$
$\Leftrightarrow\ \exists t'\ (t' \in \pi_X(E) \wedge t'[Y] = t)$
$\Leftrightarrow\ \exists t'', t'\ (t'' \in E \wedge t''[X] = t' \wedge t'[Y] = t)$
$\Leftrightarrow\ \exists t'', t'\ (t'' \in E \wedge t''[Y] = t \wedge t'[Y] = t)$
$\Leftrightarrow$ (dato che $Y \subseteq X$) $\exists t''\ (t'' \in E \wedge t''[Y] = t)$
$\Leftrightarrow\ t \in \pi_Y(E)$

e) $\pi_X(\sigma_\phi(E)) = \sigma_\phi(\pi_X(E))$, se ϕ usa solo attributi in X.

$t \in \pi_X(\sigma_\phi(E))$
$\Leftrightarrow\ \exists t'\ (t' \in \sigma_\phi(E) \wedge t'[X] = t)$
$\Leftrightarrow\ \exists t'\ (t' \in E \wedge \phi(t') \wedge t'[X] = t)$
$\Leftrightarrow$ (dato che ϕ usa solo attributi in X)
$\exists t'\ (t' \in E \wedge t'[X] = t) \wedge \phi(t)$
$\Leftrightarrow\ t \in \pi_X(E) \wedge \phi(t)$
$\Leftrightarrow\ t \in \sigma_\phi(\pi_X(E))$

f) $\sigma_\phi(E_1 \underset{\phi_J}{\bowtie} E_2) = \sigma_\phi(E_1) \underset{\phi_J}{\bowtie} E_2$, se ϕ usa solo attributi di E_1.

$t \in \sigma_\phi(E_1 \underset{\phi_J}{\bowtie} E_2)$
$\Leftrightarrow\ t \in (E_1 \underset{\phi_J}{\bowtie} E_2) \wedge \phi(t)$
$\Leftrightarrow\ \exists t', t''\ (t' \in E_1 \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi_J(t)) \wedge \phi(t)$
$\Leftrightarrow$ (dato che ϕ usa solo attributi di E_1)
$\exists t', t''\ (t' \in E_1 \wedge \phi(t') \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi_J(t))$
$\Leftrightarrow\ \exists t', t''\ (t' \in \sigma_\phi(E_1) \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi_J(t))$
$\Leftrightarrow\ t \in \sigma_\phi(E_1) \underset{\phi_J}{\bowtie} E_2$

g) Assumiamo, per semplicità, che F contenga una sola espressione $f(B)$ **AS** C, e dimostriamo: $\sigma_\phi({}_A\gamma_F(E)) = {}_A\gamma_F(\sigma_\phi(E))$, se ϕ usa solo attributi in A.

$t \in \sigma_\phi({}_A\gamma_F(E))$
$\Leftrightarrow\ t \in ({}_A\gamma_F(E)) \wedge \phi(t)$
$\Leftrightarrow\ \exists t'\ (t' \in E$
$\qquad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in E \wedge s[A] = t'[A]\})]) \wedge \phi(t)$
$\Leftrightarrow\ \exists t'\ (t' \in E \wedge \phi(t)$
$\qquad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in E \wedge s[A] = t'[A]\})])$
$\Leftrightarrow$ (Dato che ϕ usa solo attributi in A, e $t[A] = t'[A]$)

$$
\begin{aligned}
&\exists t' \, (t' \in E \wedge \phi(t') \\
&\qquad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in E \wedge \phi(s) \wedge s[A] = t'[A]\})]) \\
\Leftrightarrow \;& \exists t' \, (t' \in \sigma_{\phi}(E) \\
&\qquad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in \sigma_{\phi}(E) \wedge s[A] = t'[A]\})]) \\
\Leftrightarrow \;& t \in {}_{A}\gamma_{F}(\sigma_{\phi}(E))
\end{aligned}
$$

Esercizio 4.5
Si consideri lo schema relazionale

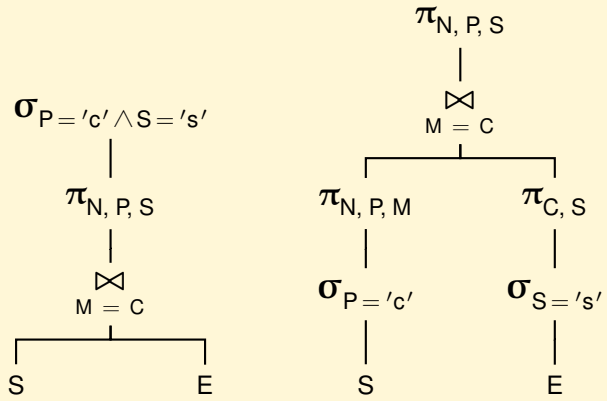
S(M, N, P, A)
E(C, S, V)

e l'espressione dell'algebra relazionale

$\sigma_{P='c' \wedge S='s'}(\pi_{N,P,S}(S \underset{M=C}{\bowtie} E))$.

Si dia una rappresentazione ad albero dell'interrogazione e si mostri come si modifica l'albero dopo la riscrittura algebrica.

Soluzione



(a) *Albero iniziale* **(b)** *Albero riscritto*

Esercizio 4.6
Si consideri lo schema relazionale

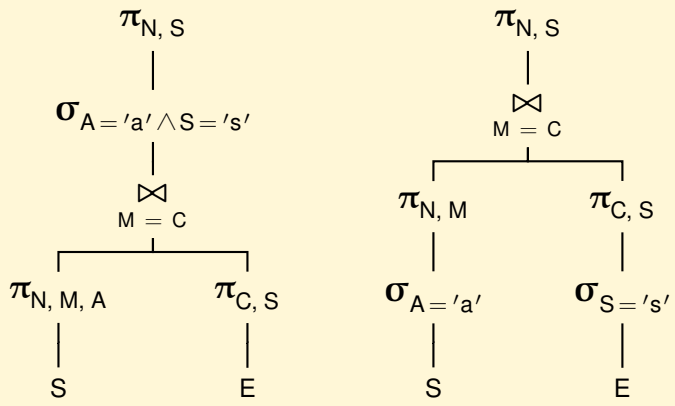
S(M, N, P, A)
E(C, S, V)

e l'espressione dell'algebra relazionale

$\pi_{N,S}(\sigma_{A='a' \wedge S='s'}(\pi_{N,M,A}(S) \underset{M=C}{\bowtie} \pi_{C,S}(E)))$.

Si dia una rappresentazione ad albero dell'interrogazione e si mostri come si modifica l'albero dopo la riscrittura algebrica.

Soluzione



(a) *Albero iniziale* **(b)** *Albero riscritto*

Capitolo 5

NORMALIZZAZIONE DI SCHEMI RELAZIONALI

Esercizio 5.1
Usando gli assiomi di Armstrong si dimostri che:

a) se $X \to YZ$, allora $X \to Y$ e $X \to Z$;
b) se $X \to Y$ e $WY \to Z$, allora $XW \to Z$;
c) se $X \to YZ$ e $Z \to BW$, allora $X \to YZB$.

Soluzione

a) Dimostriamo che $F \vdash X \to YZ$ implica $F \vdash X \to Y$.
$F \vdash YZ \to Y$ per riflessività, e la tesi segue per transitività.
$F \vdash X \to YZ \Rightarrow F \vdash X \to Z$ è analogo.
b) Dimostriamo che se $F \vdash X \to Y$ (**1**) e $F \vdash WY \to Z$ (**2**), allora $F \vdash XW \to Z$.
Per arricchimento, da (**1**) segue $F \vdash XW \to YW$ (**3**); la tesi segue per transitività da (**3**) e (**2**).
c) Dimostriamo che se $F \vdash X \to YZ$ (**1**) e $F \vdash Z \to BW$ (2), allora $F \vdash X \to YZB$.
Per arricchimento, da (**2**) segue $F \vdash YZ \to YBW$ (**3**); la tesi segue per transitività da (**1**) e (**3**).

Esercizio 5.2
Dimostrare che se uno schema $R\langle T, F\rangle$ ha solo due attributi A e B, e le possibili istanze di R non hanno i valori tutti uguali di A, o di B, allora è in BCNF.

Soluzione
Le uniche possibili DF (non banali) sono $A \to B$ and $B \to A$. Quindi, ci sono quattro possibili casi:

- Nessuna DF esiste in R: la chiave è AB e R è in *BCNF*.
- Esiste solo $A \to B$: la chiave è A e R è in *BCNF*.
- Esiste solo $B \to A$: la chiave è B e R è in *BCNF*.

– Esistono sia $A \to B$ e $B \to A$: A e B sono chiavi e R è in *BCNF*.

Esercizio 5.3
Sia $R(A, B, C, D)$ uno schema relazionale con dipendenze $F = \{A \to B, C \to B, D \to ABC, AC \to D\}$. Trovare un insieme di dipendenze G in forma canonica equivalente a F.

Soluzione
G è una *copertura canonica* di $F = \{A \to B, C \to B, D \to ABC, AC \to D\}$ se e solo se:

a) Ogni parte destra ha un *unico attributo*:
va riscritta $D \to ABC$; $G = \{A \to B, C \to B, D \to A, D \to B, D \to C, AC \to D\}$.
b) Le dipendenze non contengono *attributi estranei*:
$AC \to D$ non contiene attributi estranei perché $A^+ = \{B\}$ e $C^+ = \{B\}$.
c) Non esistono *dipendenze ridondanti*:
solo $D \to B$ è ridontante perché $D \to A$ e $A \to B$.

Lo schema in forma canonica è quindi:
$G = \{A \to B, C \to B, D \to A, D \to C, AC \to D\}$.

Esercizio 5.4
Sia $R\langle T, F\rangle$ uno schema relazionale. Dimostrare che se un attributo $A \in T$ non appare a destra di una dipendenza in F, allora A appartiene ad ogni chiave di R.

Soluzione
Consideriamo un qualunque Y tale che $A \notin Y$. Esaminando il funzionamento dell'algoritmo di chisura lenta, si osserva che A non può appartenere ad Y^+, per cui $A \notin Y$ implica che Y non può essere superchiave, per cui Y non può neppure essere una chiave.

Esercizio 5.5
Sia $R(A, B, C, D, E)$ uno schema di relazione con le dipendenze funzionali

$$F = \{AB \to CDE, AC \to BDE, B \to C, C \to B, C \to D, B \to E\}$$

Si richiede di:

a) portare F in forma canonica;
b) determinare le possibili chiavi;
c) mostrare che lo schema non è in terza forma normale;
d) portare lo schema in terza forma normale.

Soluzione

$$F = \{AB \to CDE, AC \to BDE, B \to C, C \to B, C \to D, B \to E\}$$

a) Una copertura canonica per F è data da:

$$G = \{B \to C, B \to D, B \to E, C \to B\}.$$

b) Ogni chiave deve contenere A, poiché A non è implicata da altri attributi. Quindi, dato che $AB^+ = AC^+ = \{A, B, C, D, E\}$, AB e AC sono chiavi. Calcolando AD^+, AE^+ e ADE^+, si verifica che non ci sono altre chiavi.

c) Lo schema non è in 3NF a causa delle dipendenze $C \to D$, $B \to E$, poiché D ed E non sono attributi primi e C e B non sono chiavi.

d) Applicando l'algoritmo di sintesi a G otteniamo la seguente decomposizione, che non è comunque la sola possibile: $\{BCDE, AB \text{ (o } AC)\}$. Si noti che AB (o AC) va aggiunto perché altrimenti lo schema risultante non conterrebbe alcuna relazione che contenga una chiave dello schema originale (e inoltre andrebbe perduto l'attributo A).

Esercizio 5.6
Mostrare (*a*) che se uno schema relazionale è in BCNF allora è anche in 3NF e (*b*) che se uno schema non è in 3NF allora non è neanche in BCNF.

Soluzione

(a) Per definizione, se uno schema relazionale è in BCNF allora per ogni $X \to A \in F^+$ non banale X è una superchiave, per cui lo schema è in 3NF.

(b) equivale ad (*a*) per contrapposizione.

Esercizio 5.7
Si consideri l'insieme di attributi $T = ABCDEGH$ e l'insieme di dipendenze funzionali $F = \{AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G\}$. Per ognuno dei seguenti insiemi di attributi X, (*a*) trovare una copertura della proiezione di F su X, e (*b*) dire qual è la forma normale più forte soddisfatta da X:

a) $X = ABC$;
b) $X = ABCD$;
c) $X = ABCEG$;
d) $X = ABCH$;
e) $X = ABCDEGH$.

Soluzione
Dobbiamo proiettare $F = \{AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G\}$ su

a) $X = ABC$;
b) $X = ABCD$;
c) $X = ABCEG$;
d) $X = ABCH$;
e) $X = ABCDEGH$.

Sia K l'insieme degli attributi che non appaiono a destra di nessuna dipendenza e sia U l'insieme degli attributi che non appaiono a sinistra di nessuna dipendenza; in questo caso $K = H$ e $U = GH$. Per proiettare le dipendenze su X calcoliamo Y^+ per ogni sottoinsieme stretto non vuoto di $X - U$, e, se $Y^+ \neq Y$, aggiungiamo $Y \to (X \cap Y^+ - Y)$ alle dipendenze della proiezione. Consideriamo per prima cosa i sottoinsiemi più piccoli e, ogni volta che scopriamo che $A \in Y^+$, ignoriamo tutti i soprainsiemi di YA, dato che A sarebbe estraneo nella dipendenza; indichiamo (*) sotto per indicare gli insiemi ignorati per questo motivo. In particolare, se troviamo un Y' tale che $Y'^+ = (X - K)$, allora non consideriamo più nessun soprainsieme di Y'.
Tra i singoletti, gli unici con chiusura non banale sono $B^+ = BD$ ed $E^+ = EG$.
Passando alle coppie abbiamo: $AB^+ = ABCDEG$, $AC^+ = ACBDEG$, $AD^+ = ADEG$, $AE^+ = AEG(**)$, $BC^+ = BCDAEG$, $BD^+ = (*)$, $BE^+ = BEDG(**)$, $CD^+ = CD(**)$, $CE^+ = CEG(**)$, $DE^+ = DEG(**)$. Le chiusure segnate con (**) non sono interessanti perché banali o perché il determinante può essere diviso in due sottoinsiemi Y' ed Y'' tali che $(Y' \cup Y'')^+ = Y'^+ \cup Y''^+$. Dato che AB^+, AC^+ e BC^+ contengono $X - K$, è inutile considerarne i soprainsiemi. Quindi, le sole terne da considerare sono ADE, BDE e CDE, utili nel caso e). $ADE^+ = (*)$, $BDE^+ = (*)$, $CDE^+ = CDEG(**)$. Possiamo ora calcolare una copertura per le proiezioni, considerando, per ciascun $Y \subset X$ con $(X \cap Y^+ - Y) \neq \emptyset$, la dipendenza $Y \to (X \cap Y^+ - Y)$. Consideriamo solo gli Y la cui chiusura sia "interessante" come sopra specificato, ovvero consideriamo solo: $B^+ = BD$, $E^+ = EG$, $AB^+ = ABCDEG$, $AC^+ = ACBDEG$, $AD^+ = ADEG$, $BC^+ = BCDAEG$.

a) $X = ABC$: $\{AB \to C, AC \to B, BC \to A\}$;
b) $X = ABCD$: $\{B \to D, AB \to CD, AC \to BD, BC \to AD\}$;
c) $X = ABCEG$: $\{E \to G, AB \to CEG, AC \to BEG, BC \to AEG\}$;
d) $X = ABCH$: $\{AB \to C, AC \to B, BC \to A\}$;
e) $X = ABCDEGH$: $\{B \to D, E \to G, AB \to CDEG, AC \to BDEG, BC \to ADEG, AD \to EG\}$.

Le coperture così ottenute non sono tutte canoniche. Le coperture canoniche corrispondenti sono:

a) $X = ABC$: $\{AB \to C, AC \to B, BC \to A\}$;
b) $X = ABCD$: $\{B \to D, AB \to C, AC \to B, BC \to A\}$;
c) $X = ABCEG$: $\{E \to G, AB \to C, AC \to B, BC \to A, BC \to E\}$;
d) $X = ABCH$: $\{AB \to C, AC \to B, BC \to A\}$;
e) $X = ABCDEGH$: $\{B \to D, E \to G, AB \to C, AC \to B, BC \to A, AD \to E\}$.

Osserviamo che, se $Y_A \to A$ è una dipendenza non banale in F^+, allora $BC \subseteq Y_A$, ed analogamente $Y_B \to B$ e $Y_C \to C$ implicano $AC \subseteq Y_B$ e $AB \subseteq Y_C$, per cui ogni chiave di uno $X \supseteq ABC$ deve includere AB oppure AC oppure BC. D'altra parte, la chiusura di AB, AC, e BC, contiene tutti gli attributi tranne H. Da questo possiamo dedurre l'insieme delle chiavi e degli attributi primi dei cinque schemi.

a) $X = ABC$: $\{AB \to C, AC \to B, BC \to A\}$;
chiavi: $\{AB, AC, BC\}$;
primi: $\{ABC\}$.
b) $X = ABCD$: $\{B \to D, AB \to C, AC \to B, BC \to A\}$;
chiavi: $\{AB, AC, BC\}$;
primi: $\{ABC\}$.
c) $X = ABCEG$: $\{E \to G, AB \to C, AC \to B, BC \to A, BC \to E\}$;
chiavi: $\{AB, AC, BC\}$;
primi: $\{ABC\}$.
d) $X = ABCH$: $\{AB \to C, AC \to B, BC \to A\}$;
chiavi: $\{ABH, ACH, BCH\}$;
primi: $\{ABCH\}$.
e) $X = ABCDEGH$: $\{B \to D, E \to G, AB \to C, AC \to B, BC \to A, AD \to E\}$;
chiavi: $\{ABH, ACH, BCH\}$;
primi: $\{ABCH\}$.

A questo punto è facile indicare per ogni schema la sua forma normale.

a) $X = ABC$: $\{AB \to C, AC \to B, BC \to A\}$;
chiavi: $\{AB, AC, BC\}$;
primi: $\{ABC\}$, BCNF e quindi 3NF.
b) $X = ABCD$: $\{B \to D, AB \to C, AC \to B, BC \to A\}$;
chiavi: $\{AB, AC, BC\}$;
primi: $\{ABC\}$, $B \to D$ viola le due forme normali.
c) $X = ABCEG$: $\{E \to G, AB \to C, AC \to B, BC \to A, BC \to E\}$;
chiavi: $\{AB, AC, BC\}$;
primi: $\{ABC\}$, $E \to G$ viola le due forme normali.
d) $X = ABCH$: $\{AB \to C, AC \to B, BC \to A\}$;
chiavi: $\{ABH, ACH, BCH\}$;
primi: $\{ABCH\}$, $AB \to C$ viola le due forme normali.
e) $X = ABCDEGH$: $\{B \to D, E \to G, AB \to C, AC \to B, BC \to A, AD \to E\}$;
chiavi: $\{ABH, ACH, BCH\}$;

primi: {ABCH}, $B \to D$ viola le due forme normali.

Esercizio 5.8
Si consideri la relazione con schema $R(A, B, C, D)$ e si supponga che l'unica istanza possibile di R sia:

A	B	C	D
a1	b1	c1	d1
a1	b1	c2	d2
a2	b1	c1	d3
a2	b1	c3	d4

Si trovi una copertura canonica delle dipendenze funzionali F soddisfatte da R.
Se lo schema non è in BCNF, si applichi l'algoritmo di decomposizione per trovare uno decomposizione in BCNF e dire se conserva le dipendenze.

Soluzione
Da un esame dei dati di $R(A, B, C, D)$ si trova che valgono le seguenti dipendenze funzionali dell'insieme F:

- Con determinante un singoletto:
  $F = \{A \to B, C \to B, D \to A, D \to B, D \to C\}$, con D una chiave.
- Con determinante due attributi non si considerano quelli con B, poiché B è costante, né con D che è una chiave, e si aggiungono ad F altre due dipendenze $AC \to B$ e $AC \to D$:
  $F = \{A \to B, C \to B, D \to A, D \to B, D \to C, AC \to B, AC \to D\}$

F è una *copertura canonica* se e solo se:

a) Ogni parte destra ha un *unico attributo*: condizione soddisfatta.
b) Le dipendenze non contengono *attributi estranei*: solo $AC \to B$ contiene l'attributo estraneo A perché $C^+ = \{B, C\}$ e si riscrive con $C \to B$, già in F.
c) Non esistono *dipendenze ridondanti*: $D \to B$ si elimina perché implicata da $D \to A$ e $A \to B$.

Pertanto la forma canonica delle dipendenze di R è:
$F = \{A \to B, C \to B, D \to A, D \to C, AC \to D\}$
con *chiavi* {A, C} e {D}.
La dipendenza $A \to B$ viola la BCNF perché il determinante non è una chiave.
Applicando l'algoritmo di decomposizione lo schema di R si decompone in due relazioni $R1\langle\{A, B\}, \{A \to B\}\rangle$, $R2\langle\{A, C, D\}, \{D \to A, D \to C, AC \to D\}\rangle$, che sono in BCNF, ma con la *perdita della dipendenza* $C \to B$.

Un'*altra soluzione* si trova usando la dipendenza $\emptyset \to B$ per specificare che B ha il valore uguale in tutte le ennuple.

- Per i valori di B uguali in ogni ennupla:
  $F = \{\emptyset \to B\}$.
- Con determinante un singoletto:
  $F = \{\emptyset \to B, A \to B, C \to B, D \to A, D \to B, D \to C\}$, con D una chiave.
- Con determinante due attributi non si considerano quelli con B, poiché B è costante, né con D che è una chiave, e si aggiungono ad F altre due dipendenze $AC \to B$ e $AC \to D$:
  $F = \{\emptyset \to B, A \to B, C \to B, D \to A, D \to B, D \to C, AC \to B, AC \to D\}$

F è una *copertura canonica* se e solo se:

a) Ogni parte destra ha un *unico attributo*: condizione soddisfatta.
b) Le dipendenze non contengono *attributi estranei*:
   $AC \to B$ ha l'attributo estraneo A perché $C^+ = \{B, C\}$ e si riscrive con $C \to B$.
   La dipendenza $C \to B$ ha l'attributo estraneo C perché $\{\}^+ = \{B\}$ e si riscrive con $\emptyset \to B$, già in F. Stessa cosa accade per $A \to B$, $D \to B$ e $AC \to B$.
   $F = \{\emptyset \to B, D \to A, D \to C, AC \to D\}$
c) Non esistono *dipendenze ridondanti*: condizione soddisfatta.

Pertanto la forma canonica delle dipendenze di R è:
$F = \{\emptyset \to B, D \to A, D \to C, AC \to D\}$
con *chiavi* $\{A, C\}$ e $\{D\}$.

La dipendenza $\emptyset \to B$ viola la BCNF perché il determinante non è una chiave. Applicando l'algoritmo di decomposizione lo schema di R si decompone in due relazioni $R1\langle\{B\}, \{\emptyset \to B\}\rangle$, $R2\langle\{A, C, D\}, \{D \to A, D \to C, AC \to D\}\rangle$, che sono in BCNF, e le *dipendenze sono preservate*. Inoltre, il fatto che la relazione R1 ha solo l'attributo B implica che avrà solo una riga, contenente il valore costante presente nella colonna di R, e i dati sono conservati perché $R = R1 \times R2$.

Esercizio 5.9
Si consideri il seguente schema relazionale:

Impiegati(Nome, Livello, Stipendio)

per il quale valgono le seguenti dipendenze funzionali
Nome $\to$ Livello Stipendio
Livello $\to$ Stipendio

a) Lo schema è in 3NF o in BCNF?
b) Se lo schema non è in 3NF, si applichi l'algoritmo di sintesi per trovare una decomposizione che preservi dati e dipendenze.

c) Se lo schema non è in BCNF, si applichi l'algoritmo di decomposizione per trovare uno decomposizione in BCNF e dire se conserva le dipendenze.

Soluzione
Abbreviamo lo schema come segue: $I\langle\{N, L, S\}, \{N \to LS, L \to S\}\rangle$. Le dipendenze in forma canonica sono: $\{N \to L, L \to S\}$, *chiavi*: $\{N\}$.

a) Lo schema non è in 3NF né, quindi, in BCNF, per la dipendenza $L \to S$.
b) *Algoritmo di sintesi*: $R1\langle\{N, L\}, \{N \to L\}\rangle$; $R2\langle\{L, S\}, \{L \to S\}\rangle$. Non c'è bisogno di aggiungere schemi poiché NL è già una superchiave.
c) *Algoritmo di analisi*: poiché $L \to S$ viola la BCNF, si decompone I in due schemi $R1\langle\{L, S\}, \{L \to S\}\rangle$ e $R2\langle\{L, N\}, \{N \to L\}\rangle$, che sono in BCNF, e la decomposizione preserva le dipendenze.

Esercizio 5.10
Si supponga che per archiviare dati sull'inventario delle apparecchiature di un'azienda sia stata usata una tabella con la seguente struttura:

Inventario(NInventario, Modello, Descrizione, NSerie, Costo, Responsabile, Telefono)

NInventario	Modello	Descrizione	NSerie	Costo	Responsabile	Telefono
...	...	...	...	...	...	...
111	SUN3	Stazione Sun	ajk0785	25 000	Caio	576
112	PB180	Notebook Mac	a908m	6000	Tizio	587
113	SUN3	Stazione Sun	ajp8907	27 000	Tizio	587
...	...	...	...	...	...	...

Il numero di inventario identifica un'apparecchiatura. Un'apparecchiatura ha un costo, un modello e un numero di serie. Apparecchiature dello stesso modello possono avere costi differenti, perché acquistate in momenti diversi, ma hanno la stessa descrizione. Il numero di serie è una caratteristica dell'apparecchiatura e due diverse apparecchiature dello stesso modello hanno numero di serie diverso. Ogni apparecchiatura ha un responsabile, che può avere più apparecchiature, ma un unico numero di telefono. I responsabili sono identificati dal cognome.

Definire le dipendenze funzionali e dire se lo schema proposto presenta anomalie, giustificando la risposta.

Trasformare la rappresentazione in uno schema relazionale in 3NF.

Soluzione
Abbreviamo lo schema come segue: Inv(NInv, Mod, Descr, NSerie, Costo, Resp, Tel).

Dal testo ricaviamo le seguenti dipendenze:

1. NInv → Mod NSerie Costo
2. Mod → Descr
3. Mod NSerie → NInv
4. NInv → Resp
5. Resp → Tel

Le chiavi dello schema sono (NInv) e (Mod, NSerie). Lo schema presenta numerose anomalie, testimoniate dalle dipendenze (2), (4) e (5). Applicando l'algoritmo di sintesi, otteniamo il seguente schema, che rispetta sia la 3NF che la BCNF:

Apparecchiature(NInv, Mod, NSerie, Costo, Resp),
Modelli(Mod, Descr),
Responsabili(Resp, Tel).

Esercizio 5.11
Una palestra ospita diversi corsi appartenenti a diverse tipologie (aerobica, danza moderna, ...). Ogni corso ha una sigla, che lo identifica, un insegnante e alcuni allievi. Un insegnante offre in generale più corsi, anche con diverse tipologie, e anche un allievo può essere iscritto a più corsi. Di ogni insegnante interessano il nome (che lo identifica) e l'indirizzo. Di ogni allievo interessano il nome (che lo identifica) e il numero di telefono. Per ogni allievo interessa sapere, per ogni corso che frequenta, quanto ha già versato finora. La palestra gestisce attualmente i dati con un foglio elettronico con tante colonne quanti sono i fatti elementari da trattare.

Si chiede di:

a) Definire le dipendenze funzionali.
b) Dare una copertura canonica delle dipendenze in tale schema ed elencare le chiavi.
c) Applicare allo schema l'algoritmo di sintesi per portarlo in 3NF, e dire se lo schema così ottenuto è anche in BCNF.
d) Applicare allo schema l'algoritmo di decomposizione per portarlo in BCNF, e dire se tale decomposizione preserva le dipendenze.

Soluzione
Supponiamo che gli attributi dello schema siano i seguenti, di cui riportiamo un'abbreviazione:

SiglaCorso (SiglaC), Tipologia (Tipo), NomeInsegnante (NomeI), IndirizzoInsegnante (IndI), NomeAllievo (NomeA), TelefonoAllievo (TelA), VersatoFinora (Vers).

a) Dipendenze funzionali:
   - SiglaC → Tipo NomeI
   - NomeI → IndI
   - NomeA → TelA
   - NomeA SiglaC → Vers

   Non è chiaro se si assume anche "TelA → NomeA", ma la cosa non fa molta differenza (se non per ciò che riguarda l'insieme delle chiavi).

b) Le dipendenza sono già in forma minima. La sola chiave è la coppia (NomeA, SiglaC).

c) Applicando l'algoritmo di sintesi otteniamo lo schema:
   Corsi(SiglaC, Tipo, NomeI),
   Insegnanti(NomeI, indI),
   Allievi(NomeA, TelA),
   SoldiVersati(NomeA, SiglaC, Vers),
   che è anche in BCNF.

d) Considerando le dipendenze nell'ordine in cui sono elencate, otteniamo la seguente decomposizione:
   $R\langle\{SiglaC, Tipo, NomeI, IndI, NomeA, TelA, Vers\},$
   $\{SiglaC \to Tipo\ NomeI, NomeI \to IndI, NomeA \to TelA, NomeA\ SiglaC \to Vers\}\rangle \Rightarrow$

   $R1\langle\{SiglaC, Tipo, NomeI, IndI\}, \{SiglaC \to Tipo\ NomeI, NomeI \to IndI\}\rangle,$
   $R2\langle\{SiglaC, NomeA, TelA, Vers\}, \{NomeA \to TelA, NomeA\ SiglaC \to Vers\}\rangle \Rightarrow$

   $R11\langle\{NomeI, IndI\}, \{NomeI \to IndI\}\rangle,$
   $R12\langle\{NomeI, SiglaC\ Tipo\}, \{SiglaC \to Tipo\ NomeI\}\rangle,$
   $R2\langle\{SiglaC, NomeA, TelA, Vers\}, \{NomeA \to TelA, NomeA\ SiglaC \to Vers\}\rangle \Rightarrow$

   $R11\langle\{NomeI, IndI\}, \{NomeI \to IndI\}\rangle,$
   $R12\langle\{NomeI, SiglaC, Tipo\}, \{SiglaC \to Tipo\ NomeI\}\rangle,$
   $R21\langle\{NomeA, TelA\}, \{NomeA \to TelA\}\rangle,$
   $R22\langle\{NomeA, SiglaC, Vers\}, \{NomeA\ SiglaC \to Vers\}\rangle.$

   Questa decomposizione è uguale a quella ottenuta usando l'algorimo di sintesi, e conserva anche le dipendenze.

Esercizio 5.12
Quali di questi test ammettono algoritmi di complessità polinomiale:

a) Dato lo schema di relazione $R\langle T, F\rangle$, $A \in T$, A è primo?
b) Dati due insiemi di dipendenze F e G, $F \equiv G$?
c) Dato lo schema di relazione $R\langle T, F\rangle$, e $X \subseteq T$, X è una superchiave?
d) Dato lo schema di relazione $R\langle T, F\rangle$, e $X \subseteq T$, X è una chiave?

Soluzione

a) *Problema*: dato lo schema di relazione $R\langle T, F\rangle$, $A \in T$, A è primo?
Un algoritmo per valutare la primalità di un attributo A consiste nel generare tutti i sottoinsiemi di T che contengono A e verificare se uno di essi sia una chiave. Questo algoritmo ha complessità esponenziale rispetto al numero di attributi, poiché il numero di sottoinsiemi di un insieme di a elementi è 2^a, e verificare se un sottoinsieme di un insieme di a attributi sia una chiave, rispetto ad un insieme di p dipendenze, ha complessità polinomiale $O(ap)$. Non sono noti algoritmi polinomiali.

b) *Problema*: Dati due insiemi di dipendenze F e G, $F \equiv G$?
Il problema ammette un algoritmo polinomiale. È sufficiente verificare, per ogni $X \to Y \in F$ che si abbia $Y \subseteq X_G^+$, e che per ogni $X \to Y \in G$ si abbia che $Y \subseteq X_F^+$. Dato che la chiusura e l'inclusione possono essere valutate in tempo $O(ap)$, la complessità dell'algoritmo è quindi $O(ap^2)$.

c) *Problema*: Dato lo schema di relazione $R\langle T, F\rangle$, e $X \subseteq T$, X è una superchiave?
È sufficiente verificare se $T = X_F^+$, con complessità $O(ap)$.

d) *Problema*: Dato lo schema di relazione $R\langle T, F\rangle$, e $X \subseteq T$, X è una chiave?
È sufficiente verificare che si abbia $X_F^+ = T$ e che, per ogni $A \in X$, si abbia $(X - A)_F^+ \neq T$. La complessità è quindi $O(a^2p)$.

Esercizio 5.13
Dato lo schema $R\langle T, F\rangle$, discutere la complessità dei seguenti problemi:

a) Trovare una chiave di R.
b) Trovare tutte le chiavi di R.
c) $F - \{X \to Y\} \equiv F$.

Soluzione

a) Trovare una chiave di $R\langle T, F\rangle$: si può utilizzare il seguente algoritmo, dove `A[1..n]` enumera gli attributi in T:

```
input: A[1..n], F;
K := [A[1],...,A[n]];
for i in 1..n
    do if chiudi((K−A[i]),F) = T then K := K−A[i];
return(K)
```

È facile dimostrare le seguenti invarianti:

- sia K_i il valore di K dopo il ciclo i-esimo; per ogni i, si ha che $K_i \to T$;
- per ogni $j \leqslant i$, se l'attributo $A[j]$ appartiene a K_i, allora $A[j]$ non è estraneo in $K_i \to T$.

Ponendo $i = n$, abbiamo quindi che K_n è una superchiave senza attributi estranei, ovvero è una chiave.

b) Trovare tutte le chiavi di $R\langle T, F\rangle$: in generale, il numero di chiavi di uno schema può crescere in modo esponenziale con le dimensioni dello schema stesso, per cui nessun algoritmo che le enumeri può avere una complessità meno che esponenziale rispetto alle dimensioni dello schema.

c) $F - \{X \to Y\} \equiv F$: basta verificare se $X^+_{F-\{X\to Y\}} \supseteq Y$, con costo $O(ap)$.

Esercizio 5.14
Discutere la complessità dei seguenti test:

a) *Test 3NF*: dato lo schema di relazione $R\langle T, F\rangle$, R è in 3NF rispetto ad F?
b) *Test BCNF*: dato lo schema di relazione $R\langle T, F\rangle$, R è in BCNF rispetto ad F?
c) *Test BCNF di sottoschema*: dato lo schema di relazione $R\langle T, F\rangle$, dato $X \subseteq T$, X è in BCNF rispetto alla proiezione di F su X?
d) *Test copertura canonica*: dato lo schema di relazione $R\langle T, F\rangle$, F è in forma canonica?

Soluzione

a) *Test 3NF*. Un modo per determinare se uno schema $R\langle T, F\rangle$ sia in 3NF consiste nel portare lo schema in una forma canonica G e poi nel verificare se, per ogni dipendenza in $X \to A$ in G, se X non è una chiave, allora A è primo. Per portare F in forma canonica è sufficiente:

- dividere le dipendenze in modo che ogni membro destro sia composto di un solo attributo;
- eliminare, da ogni membro sinistro, gli attributi ridondanti;
- eliminare le dipendenze ridondanti.

Tutte queste operazioni si possono effettuare in tempo polinomiale.
Data poi una dipendenza $X \to A$ in G, verificare se X sia chiave si può fare in tempo polinomiale, poiché significa verificare se $X^+ = T$; tuttavia, quando X non è chiave, verificare se A sia primo usando l'algoritmo sopra descritto richiede un tempo esponenziale. Quindi questo algoritmo ha complessità esponenziale. Non sono noti algoritmi polinomiali.

b) *Test BCNF*. Questo problema ha complessità polinomiale; basta utilizzare lo stesso algoritmo visto al punto precedente per portare F in forma canonica G. Per ogni $X \to A$ in G bisogna poi verificare se X sia una chiave, e questa operazione è polinomiale.

   Proiezione delle dipendenze: per ogni $Y \subset X$ non vuoto si calcoli Y_F^+, e si generi in questo modo l'insieme $G = \{Y \to A \mid Y \subset X,\ A \in ((Y_F^+ - Y) \cap X)\}$. Questa operazione ha una complessità esponenziale rispetto alla dimensione di X. Si porti poi G in forma canonica, con un costo polinomiale rispetto alla dimensione di G (che, nel caso pessimo, è esponenziale rispetto alla dimensione di X).

   In pratica, si possono adottare gli accorgimenti descritti nell'esercizio 5.6 per ridurre la quantità di sottoinsiemi di X da considerare, ma questi non bastano a rendere la complessità di questo algoritmo meno che esponenziale. Non sono noti algoritmi polinomiali.

c) *Test BCNF di sottoschema*. Per risolvere questo problema si può operare come segue: prima si proietta F su X, operando come descritto sopra, e poi si verifica se $R\langle X, \Pi_X F\rangle$ è in BCNF. Questo algoritmo è esponenziale, dato che questo è il costo della fase di proiezione.

d) *Test copertura canonica*. L'algoritmo più semplice verifica le seguenti condizioni:

   - Tutti i membri destri sono formati da un solo attributo: costo $O(p)$
   - Per ogni $X \to A \in F$, per ogni $B \in X$, $A \notin (X - B)^+$: costo $p \times a \times O(ap) = O(a^2p^2)$
   - Per ogni $X \to A \in F$, $A \in X^+$, dove la chiusura è calcolata rispetto ad $F - \{X \to A\}$: costo $p \times O(ap) = O(ap^2)$

   Costo totale dell'algoritmo: $O(a^2p^2)$.

Esercizio 5.15
Si supponga che una dipendenza funzionale $X \to Y$ sia soddisfatta da un'istanza di relazione r. Sia $s \subseteq r$ (quindi s è una relazione con gli stessi attributi di r). s soddisfa $X \to Y$? Se sì, dire perché, altrimenti dare un controesempio.

Soluzione
Se un'istanza di relazione r soddisfa una dipendenza $X \to Y$, allora
$\forall t_1, t_2 \in r. t_1[X] = t_2[x] \to t_1[Y] = t_2[Y]$.
Da $s \subseteq r$ segue quindi che anche
$\forall t_1, t_2 \in s. t_1[X] = t_2[x] \to t_1[Y] = t_2[Y]$, per cui s soddisfa $X \to Y$.

Esercizio 5.16
Si supponga che una dipendenza funzionale $X \to Y$ sia soddisfatta da due istanze di relazione r ed s con gli stessi attributi.

$r \cap s$ soddisfa $X \to Y$? $r \cup s$ soddisfa $X \to Y$? Se sì, dire perché, altrimenti dare un controesempio.

Soluzione
Ragionando come nell'esercizio 5.14, si dimostra che $r \cap s$ soddisfa $X \to Y$. Invece $r \cup s$ potrebbe non soddisfare $X \to Y$. Si considerino le istanze di relazione $\{(A = 1, B = 1)\}$ e $\{(A = 1, B = 2)\}$, con schema $R(AB)$. Ciascuna delle due soddisfa $A \to B$, tuttavia la loro unione $\{(A = 1, B = 1), (A = 1, B = 2)\}$ non soddisfa la dipendenza.

Esercizio 5.17
Sia $R\langle T, F\rangle$ uno schema relazionale, con F una copertura canonica. Si dimostri che se lo schema ha una sola chiave ed è in 3NF allora è anche in BCNF. (Suggerimento: si inizi con lo scrivere la definizione di 3NF e di BCNF, e si dia un nome, ad esempio Y, all'insieme di attributi che forma la sola chiave di $R\langle T, F\rangle$. Si ragioni per assurdo, supponendo che $R\langle T, F\rangle$ sia in 3NF ma non in BCNF).

Soluzione
Sia Y la sola chiave di $R\langle T, F\rangle$, e assumiamo, per assurdo, che $R\langle T, F\rangle$ sia in 3NF ma non in BCNF. Ne segue che esiste una dipendenza non banale $X \to A$ tale che X non è superchiave ma $A \in Y$. Da $X \to A$ segue che $XY - A$ determina Y, per cui $XY - A$ è una superchiave, per cui contiene una chiave la quale non contiene A. Quindi $R\langle T, F\rangle$ ha almeno due chiavi, contraddicendo l'ipotesi.

Esercizio 5.18
Dimostrare il Teorema 5.9:
Uno schema $R\langle T, F\rangle$ *è in BCNF se e solo se per ogni dipendenza funzionale non banale* $X \to Y \in F$, X *è una superchiave.*

Soluzione
Definizione di BCNF: *Uno schema* $R\langle T, F\rangle$ *è in BCNF se e solo se per ogni dipendenza funzionale non banale* $X \to Y \in F^+$, X *è una superchiave.*

($\Rightarrow$) Assumendo che $R\langle T, F\rangle$ sia in BCNF, l'implicazione – per ogni dipendenza non banale $X \to Y \in F$, X è una superchiave – è ovvia dato che $F \subseteq F^+$.

($\Leftarrow$) Assumendo che in $R\langle T, F\rangle$ per ogni dipendenza non banale $X \to Y \in F$, X sia una superchiave, si mostra che per ogni $X \to Y \in F^+$ non banale X è una superchiave e quindi lo schema è in BCNF.

A tale scopo basta osservare che F^+ deriva da F tramite l'applicazione degli *assiomi di Armstrong*, e questi, partendo da uno schema che soddisfa la condizione sopra specificata, producono ancora solo dipendenze che o sono banali o hanno a sinistra una superchiave.

Riflessività: se $Y \subseteq X$, allora $X \to Y$.
Si derivano solo dipendenze banali.
Arricchimento: se $X \to Y$ e $W \subseteq T$, allora $XW \to YW$.
Se la dipendenza ottenuta è non banale, allora anche quella originale era non banale, per cui X e XW sono superchiavi.
Transitività: se $X \to Y$ e $Y \to Z$, allora $X \to Z$.
Se almeno una delle due dipendenze è non banale, allora almeno uno tra X ed Y è superchiave, e quindi, dato che X determina Y, X è superchiave.

Esercizio 5.19

Dimostrare il Teorema 5.11:
Uno schema $R\langle T, F\rangle$ *è in 3NF se e solo se, per ogni dipendenza funzionale* $X \to A_1, \ldots, A_n \in F$, *e per ogni* $i \in \{1 \ldots n\}$, $A_i \in X$ *oppure* X *è una superchiave oppure* A_i *è primo.*

Soluzione

Definizione di 3NF: *Uno schema* $R\langle T, F\rangle$ *è in 3FN se e solo se, per ogni dipendenza funzionale non banale* $X \to A \in F^+$, *allora* X *è una superchiave oppure* A *è primo.*

($\Rightarrow$) Assumendo che $R\langle T, F\rangle$ sia in 3NF, allora l'implicazione che soddisfa la condizione C (*per ogni dipendenza* $X \to A_1, \ldots, A_n \in F$, *e per ogni* $i \in \{1 \ldots n\}$, $A_i \in X$ *oppure* X *è una superchiave oppure* A_i *è primo*) è ovvia.

($\Leftarrow$) Assumendo che $R\langle T, F\rangle$ soddisfi la condizione C, allora si mostra che lo schema è in 3NF.

A tale scopo basta osservare che F^+ deriva da F tramite l'applicazione degli *assiomi di Armstrong*, e questi, partendo da uno schema che soddisfa C, producono ancora solo dipendenze che soddisfano C.

Riflessività: se $Y \subseteq X$, allora $X \to Y$.
Si derivano solo dipendenze banali che soddisfano $A_i \in X$.
Arricchimento: se $X \to A_1 \ldots A_n$ e $W \subseteq T$, allora $XW \to A_1 \ldots A_n W$.
Se valeva X superchiave, allora vale XW superchiave, per la nuova dipendenza. Altrimenti, ciascuno degli attributi A_i continua a soddisfare $A_i \in X$ o A_i *primo* come in $X \to A_1 \ldots A_n$, e tutti gli attributi in $A_i \in W$ soddisfano $A_i \in X$, dato che appartengono a XW.
Transitività: se $X \to Y$ e $Y \to Z$, allora $X \to Z$.
Se X è superchiave abbiamo finito. Se invece X non è superchiave, allora neppure Y è superchiave, e quindi, dato che $Y \to A_i$ soddisfa C, o A_i è *primo*, oppure A_i non è *primo* e $A_i \in Y$. In quest'ultimo caso, dato che $X \to Y$ soddisfa C, X non è superchiave, A_i non è *primo* e $A_i \in Y$, allora A_i deve soddisfare $A_i \in X$.

Capitolo 6

SQL
PER L'USO INTERATTIVO
DI BASI DI DATI

Esercizio 6.1
Dare un'espressione **SELECT** per stabilire se i valori di A in una relazione con schema R(A, B, C) siano tutti diversi. L'espressione deve essere diversa da **SELECT** A **FROM** R.

Soluzione
La seguente espressione restituisce true se i valori di A sono tutti diversi, false altrimenti.

```
SELECT    COUNT(*) = COUNT(DISTINCT A)
FROM      R;
```

Si noti che per stabilire se un insieme di attributi {B, C} determina l'attributo A, si controlla se la seguente interrogazione ritorna una tabella vuota:

```
SELECT    B, C, COUNT(DISTINCT A)
FROM      R
GROUP BY  B, C
HAVING    COUNT(DISTINCT A) > 1;
```

Esercizio 6.2
Si ricorda che il predicato **IN** è equivalente a =**ANY**. Spiegare perché il predicato **NOT IN** non è equivalente a <>**ANY** ma a <>**ALL**.

Soluzione
L'espressione elemento θ**ANY** Sottoselect verifica che elemento sia in relazione θ con *almeno* un elemento dell'insieme restituito da Sottoselect (ed è falsa se l'insieme è vuoto). Se θ è <>, quindi, l'espressione è vera se elemento è diverso da *almeno* un elemento di Sottoselect.

L'espressione elemento **NOT IN** Sottoselect, invece, significa che l'elemento *non* deve essere presente nell'insieme restituito da Sottoselect, e quindi deve essere diverso da *tutti* gli elementi di questo insieme, corrispondendo così all'espressione elemento <>**ALL** Sottoselect.

Ad esempio, le seguenti query equivalenti relative allo schema dell'esercizio 5.4 ritornano matricola e nome degli studenti che non hanno fatto esami:

```
SELECT    Matricola, Nome
FROM      Studenti
WHERE     Matricola NOT IN (SELECT MatricolaStudente
                            FROM   Esami));

SELECT    Matricola, Nome
FROM      Studenti
WHERE     Matricola <> ALL (SELECT MatricolaStudente
                            FROM   Esami));
```

Esercizio 6.3
È importante sapere quando una **SELECT** ritorna una tabella con righe diverse per evitare di usare inutilmente la clausola **DISTINCT**, che comporta un costo addizionale per l'esecuzione dell'interrogazione (perché?).

a) È vero che con una **SELECT** con una tabella nella parte **FROM**, e senza **GROUP BY**, non vi saranno righe duplicate nel risultato se gli attributi della parte **SELECT** sono una superchiave della tabella?
b) È vero che con una **SELECT** con più tabelle nella parte **FROM**, e senza **GROUP BY**, non vi saranno righe duplicate nel risultato se gli attributi della parte **SELECT** sono una superchiave di ogni tabella?
c) È vero che nessuna interrogazione con un **GROUP BY** può avere duplicati nel risultato?

Per ogni caso dare un esempio di interrogazione che ritorna duplicati se la condizione non è verificata.

Soluzione
L'uso della clausola **DISTINCT** richiede l'eliminazione dei duplicati di un insieme, operazione in generale costosa.

a) *Domanda*: È vero che con una **SELECT** con una tabella nella parte **FROM**, e senza **GROUP BY**, non vi saranno righe duplicate nel risultato se gli attributi della parte **SELECT** sono una superchiave della tabella?

Sì, è vero. Se gli attributi della **SELECT** provengono da un'unica tabella e sono una superchiave della stessa significa che contengono almeno una chiave candidata della tabella, quindi non vi possono essere due righe uguali nel risultato (altrimenti avrebbero lo stesso valore per la chiave candidata)

b) *Domanda*: È vero che con una **SELECT** con più tabelle nella parte **FROM**, e senza **GROUP BY**, non vi saranno righe duplicate nel risultato se gli attributi della parte **SELECT** sono una superchiave di ogni tabella?

Sì, è vero. Consideriamo ad esempio due tabelle R ed S. Il numero maggiore di ennuple che può essere restituito da una select che coinvolge le due tabelle è dato dalle ennuple del prodotto cartesiano delle due tabelle. Nel prodotto cartesiano ogni riga è costituita dalla combinazione di una ennupla di R con una ennupla di S. Anche se le singole componenti di R e di S sono ripetute, ogni combinazione è unica, dato che contiene superchiavi sia per R che per S.

c) *Domanda*: È vero che nessuna interrogazione con un **GROUP BY** può avere duplicati nel risultato?

Falso. Si consideri ad esempio la seguente query sulla relazione R con attributi K, chiave, ed S:

```
SELECT    COUNT(*)
FROM      R
GROUP BY  S
```

È possibile che il numero di ennuple di gruppi diversi sia uguale, e quindi in questo caso diverse righe del risultato saranno uguali.

Esercizio 6.4
Si consideri lo schema relazionale

Studenti(Matricola, Nome, Provincia, AnnoNascita)
Esami(Materia, MatricolaStudente, Voto, NumAppello, Anno)

Formulare in SQL le seguenti interrogazioni:

a) Trovare il nome degli studenti di Pisa che hanno superato l'esame di Programmazione con 30.
b) Trovare il nome degli studenti che hanno superato cinque esami.
c) Trovare per ogni studente di Pisa il numero degli esami superati, il voto massimo, minimo e medio.
d) Trovare per ogni materia il numero degli esami fatti al primo appello, il voto massimo, minimo e medio.

Soluzione
Di solito un'interrogazione può essere espressa in SQL in modi diversi. Solo in qualche caso si mostrano più soluzioni.

a)
```
SELECT    Nome
FROM      Studenti
          JOIN Esami ON Matricola = MatricolaStudente
WHERE     Provincia = 'Pisa'
AND       Materia = 'Programmazione' AND Voto = 30;
```

b)
```
SELECT    Nome
FROM      Studenti
          JOIN Esami ON Matricola = MatricolaStudente
GROUP BY  Nome
HAVING    COUNT(*) = 5;
```
oppure:
```
SELECT    Nome
FROM      Studenti
WHERE     5 = ( SELECT  COUNT(*)
                FROM    Esami
                WHERE   MatricolaStudente = Matricola);
```

Questa soluzione è meno efficiente della precedente perché la sottoselect viene calcolata per ogni record della tabella Studenti.

c)
```
SELECT    Matricola, Nome, COUNT(*) AS EsamiSuperati,
          MAX(Voto), MIN(Voto), AVG(Voto)
FROM      Studenti
          JOIN Esami ON Matricola = MatricolaStudente
WHERE     Provincia = 'Pisa'
GROUP BY  Matricola, Nome;
```

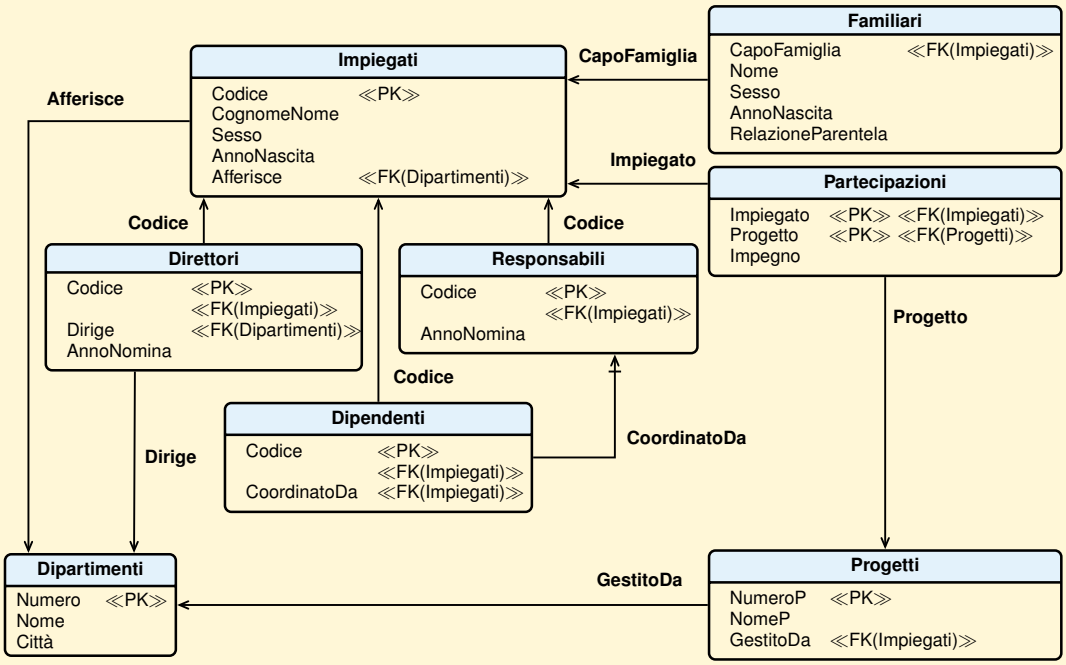
d)
```
SELECT    Materia, COUNT(*) AS NumeroEsami,
          MAX(Voto), MIN(Voto), AVG(Voto)
FROM      Esami
WHERE     NumAppello = 1
GROUP BY  Materia;
```

Esercizio 6.5
Usando la base di dati relazionale ottenuta dall'Esercizio 2.8, formulare in SQL le seguenti interrogazioni:

1. Trovare il nome e l'anno di nascita dei figli dell'impiegato con codice 350.
2. Trovare il nome e codice degli impiegati e il nome del dipartimento dove lavorano.
3. Trovare il nome degli impiegati, il nome e l'anno di nascita dei figli maschi a carico.
4. Trovare, per ogni progetto in corso a Pisa, il numero e il nome del progetto, il nome del dipartimento dove si svolge, il cognome del direttore del dipartimento.
5. Trovare il nome dei dipartimenti con almeno un impiegato con persone a carico.
6. Trovare il numero degli impiegati del dipartimento di Informatica.
7. Trovare, per ogni progetto al quale lavorano più di due impiegati, il nome, il numero e il numero degli impiegati che vi lavorano.
8. Trovare per ogni dipartimento il nome, il numero degli impiegati e la media del loro anno di nascita.
9. Trovare i nomi dei supervisori e dei loro dipendenti.
10. Trovare il nome degli impiegati e il nome del dipartimento in cui lavorano.
11. Trovare il nome degli impiegati senza familiari a carico.
12. Trovare i progetti cui partecipa il sig. Rossi come impiegato o come direttore del dipartimento che gestisce il progetto.
13. Trovare il nome degli impiegati che hanno i familiari a carico dello stesso sesso.
14. Trovare il nome degli impiegati che hanno tutti i familiari a carico dello stesso sesso dell'impiegato.
15. Trovare il nome degli impiegati che lavorano almeno a tutti i progetti dell'impiegato con codice 300.
16. Trovare il nome del dipartimento e il numero di impiegati nati dopo il 1950, per i dipartimenti con più di due impiegati.

Soluzione
Anche per questo esercizio solo in alcuni casi si mostrano più soluzioni.

Lo schema relazionale è mostrato in figura. Per poter provare queste ed altre interrogazioni nel sistema JRS sono disponibili nel file http://fondamentidibasididati.it/AziendaPerJRS.zip i comandi per la creazione della base di dati e l'inserimento di alcuni dati. Si ricordi che in JRS la giunzione si fa con l'uso di ",", quindi è necessario convertire tutte le operazioni che usano **JOIN** in operazioni con ",".



Tutte le query assumono la seguente definizione della base di dati:

```
CREATE TABLE Dipartimenti (
    Numero          INTEGER       NOT NULL,
    Nome            VARCHAR(16)   NOT NULL,
    Città           VARCHAR(16)   NOT NULL,
    PRIMARY KEY     (Numero),
    UNIQUE          (Nome, Città)    );

CREATE TABLE Impiegati (
    Codice          INTEGER       NOT NULL,
    CognomeNome     VARCHAR(20)   NOT NULL,
    Sesso           VARCHAR(1),
    AnnoNascita     INTEGER       NOT NULL,
    Afferisce       INTEGER       NOT NULL,
    PRIMARY KEY     (Codice),
    FOREIGN KEY     (Afferisce)
    REFERENCES      Dipartimenti  ON DELETE CASCADE );
```

```
CREATE TABLE Familiari (
    CapoFamiglia        INTEGER         NOT NULL,
    Nome                VARCHAR(16)     NOT NULL,
    Sesso               VARCHAR(1)      NOT NULL,
    AnnoNascita         INTEGER,
    RelazioneParentela  VARCHAR(8)      NOT NULL,
    PRIMARY KEY         (CapoFamiglia, Nome),
    FOREIGN KEY         (CapoFamiglia)
    REFERENCES          Impiegati       ON DELETE CASCADE );

CREATE TABLE Responsabili (
    Codice              INTEGER         NOT NULL,
    AnnoNomina          INTEGER         NOT NULL,
    PRIMARY KEY         (Codice),
    FOREIGN KEY         (Codice)
    REFERENCES          Impiegati       ON DELETE CASCADE   );

CREATE TABLE Dipendenti (
    Codice              INTEGER         NOT NULL,
    CoordinatoDa        INTEGER,
    PRIMARY KEY         (Codice),
    FOREIGN KEY         (Codice)
    REFERENCES          Impiegati       ON DELETE CASCADE,
    FOREIGN KEY         (CoordinatoDa)
    REFERENCES          Responsabili    ON DELETE SET NULL  );

CREATE TABLE Direttori (
    Codice              INTEGER         NOT NULL,
    AnnoNomina          INTEGER         NOT NULL,
    Dirige              INTEGER         NOT NULL,
    PRIMARY KEY         (Codice),
    FOREIGN KEY         (Codice)
    REFERENCES          Impiegati       ON DELETE CASCADE,
    FOREIGN KEY         (Dirige)
    REFERENCES          Dipartimenti    ON DELETE CASCADE   );
```

```
CREATE TABLE Progetti (
    NumeroP          INTEGER       NOT NULL,
    NomeP            VARCHAR(10)   NOT NULL,
    GestitoDa        INTEGER       NOT NULL,
    PRIMARY KEY      (NumeroP),
    FOREIGN KEY      (GestitoDa)
    REFERENCES       Dipartimenti  ON DELETE CASCADE   );

CREATE TABLE Partecipazioni (
    Impegno          INTEGER       NOT NULL,
    Progetto         INTEGER       NOT NULL,
    Impiegato        INTEGER       NOT NULL,
    PRIMARY KEY      (Impiegato, Progetto),
    FOREIGN KEY      (Impiegato)
    REFERENCES       Impiegati     ON DELETE CASCADE ,
    FOREIGN KEY      (Progetto)
    REFERENCES       Progetti      ON DELETE CASCADE  );

CREATE    VIEW  DatiDirettori
          (Codice, CognomeNome, Sesso, AnnoNascita, Afferisce,
          Dirige, AnnoNomina)
          AS
SELECT    i.Codice, i.CognomeNome, i.Sesso, i.AnnoNascita,
          i.Afferisce, d.Dirige, d.AnnoDiNomina
FROM      Impiegati i
          JOIN Direttori d ON i.Codice = d.Codice;

CREATE    VIEW  DatiResponsabili
          (Codice, CognomeNome, Sesso, AnnoNascita, Afferisce,
          AnnoNomina)
          AS
SELECT    i.Codice, i.CognomeNome, i.Sesso, i.AnnoNascita,
          i.Afferisce, r.AnnoDiNomina
FROM      Impiegati i
          JOIN Responsabili r ON i.Codice = r.Codice;
```

```
CREATE    VIEW DatiDipendenti
          (Codice, CognomeNome, Sesso, AnnoNascita, Afferisce,
          CoordinatoDa)
          AS
SELECT    i.Codice, i.CognomeNome, i.Sesso, i.AnnoNascita,
          i.Afferisce, d.CoordinatoDa
FROM      Impiegati i
          JOIN Dipendenti d ON .Codice = d.Codice;
```

Interrogazioni in SQL:

1. Trovare il nome e l'anno di nascita dei figli dell'impiegato con codice 350.

```
SELECT    Nome, AnnoNascita
FROM      Familiari
WHERE     CapoFamiglia = 350 AND RelazioneParentela = 'figlio';
```

2. Trovare il nome e codice degli impiegati e il nome del dipartimento dove lavorano.

```
SELECT    i.CognomeNome, i.Codice, d.Nome
FROM      Impiegati i
          JOIN Dipartimenti d ON i.Afferisce = d.Numero;
```

3. Trovare il nome degli impiegati, il nome e l'anno di nascita dei figli maschi a carico.

```
SELECT    i.CognomeNome, f.Nome, f.AnnoNascita
FROM      Impiegati i
          JOIN Familiari f ON f.CapoFamiglia = i.Codice
WHERE     f.RelazioneParentela = 'figlio' AND f.Sesso = 'm';
```

4. Trovare, per ogni progetto in corso a Pisa, il numero e il nome del progetto, il nome del dipartimento dove si svolge, il cognome del direttore del dipartimento.

```
SELECT    p.NumeroP, p.NomeP, d.Nome, dd.CognomeNome
FROM      Progetti p
          JOIN Dipartimenti d ON p.GestitoDa = d.Numero
          JOIN DatiDirettori dd ON dd.Dirige = d.Numero
WHERE     d.Citta = 'Pisa';
```

5. Trovare il nome dei dipartimenti con almeno un impiegato con persone a carico.

```
SELECT    DISTINCT d.Nome
FROM      Dipartimenti d
          JOIN Impiegati i ON i.Afferisce = d.Numero
          JOIN Familiari f ON f.CapoFamiglia = i.Codice;
```

oppure:

```
SELECT    d.Nome
FROM      Dipartimenti d
WHERE     EXISTS(SELECT *
                 FROM   Impiegati i
                 WHERE  i.Afferisce=d.Numero
                        AND EXISTS(SELECT *
                                   FROM   Familiari f
                                   WHERE  f.CapoFamiglia=i.Codice ));
```

6. Trovare il numero degli impiegati del dipartimento di Informatica.

```
SELECT    COUNT(*)
FROM      Dipartimenti d
          JOIN Impiegati i ON d.Numero = i.Afferisce
WHERE     d.Nome = 'Informatica';
```

7. Trovare, per ogni progetto al quale lavorano più di due impiegati, il nome, il numero e il numero degli impiegati che vi lavorano.

```
SELECT    p.NomeP, p.NumeroP, COUNT(*) AS NumeroImpiegati
FROM      Progetti p
          JOIN Partecipazioni pa ON p.NumeroP = pa.Progetto
          JOIN Impiegati i ON pa.Impiegato = i.Codice
GROUP BY  p.NumeroP, p.NomeP HAVING COUNT(*)>2;
```

8. Trovare per ogni dipartimento il nome, il numero degli impiegati e la media del loro anno di nascita.

```
SELECT    d.Nome, COUNT(*) AS NumeroImpiegati,
          AVG(i.AnnoNascita) AS MediaAnnoNascImpiegati
FROM      Dipartimenti d
          JOIN Impiegati i ON d.Numero=i.Afferisce
GROUP BY  d.Numero, d.Nome;
```

9. Trovare i nomi dei supervisori e dei loro dipendenti.

```
SELECT    dr.CognomeNome, ddi.CognomeNome
FROM      DatiResponsabili dr
          JOIN DatiDipendenti ddi ON ddi.CoordinatoDa = dr.Codice;
```

10. Trovare il nome degli impiegati e il nome del dipartimento in cui lavorano.

```
SELECT    i.CognomeNome, d.Nome
FROM      Impiegati i
          JOIN Dipartimenti d ON i.Afferisce = d.Numero;
```

11. Trovare il nome degli impiegati senza familiari a carico.

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     NOT EXISTS(SELECT *
                     FROM   Familiari f
                     WHERE  f.CapoFamiglia = i.Codice);
```

oppure:

```
SELECT    CognomeNome
FROM      Impiegati
  EXCEPT
SELECT    CognomeNome
FROM      Impiegati
          JOIN Familiari ON Codice = CapoFamiglia;
```

12. Trovare i progetti cui partecipa il sig. Rossi come impiegato o come direttore del dipartimento che gestisce il progetto.

```
SELECT    p.NomeP, i.Codice, i.CognomeNome
FROM      Progetti p
          JOIN Partecipazioni pa ON p.NumeroP = pa.Progetto
          JOIN Impiegati i ON pa.Impiegato = i.Codice
          JOIN Dipartimenti d ON p.GestitoDa = d.Numero
          JOIN DatiDirettori dd ON dd.Dirige = d.Numero
WHERE     (i.CognomeNome = 'Rossi Pablo' OR
          dd.CognomeNome = 'Rossi Pablo');
```

oppure:

```
SELECT    p.NomeP, i.Codice AS c, i.CognomeNome AS n
FROM      Progetti p
          JOIN Partecipazioni pa ON p.NumeroP = pa.Progetto
          JOIN Impiegati i ON pa.Impiegato = i.Codice
WHERE     i.CognomeNome = 'Rossi Pablo'
  UNION
SELECT    p.NomeP, dd.Codice AS c, dd.CognomeNome AS n
FROM      Progetti p
          JOIN Dipartimenti d ON p.GestitoDa = d.Numero
          JOIN DatiDirettori dd ON dd.Dirige = d.Numero
WHERE     dd.CognomeNome = 'Rossi Pablo';
```

13. Trovare il nome degli impiegati che hanno i familiari a carico dello stesso sesso. Se esistessero le forme di quantificazione universale ed esistenziale descritte nel testo la soluzione si potrebbe scrivere così:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     ((FOR ALL f IN Familiari WHERE f.CapoFamiglia = i.Codice:
              f.Sesso = 'm') OR
          (FOR ALL f IN Familiari WHERE f.CapoFamiglia = i.Codice:
              f.Sesso = 'f')) AND
          FOR SOME d in Familiari WHERE d.CapoFamiglia = i.Codice :
              TRUE;
```

applicando la doppia negazione ad ogni quantificatore universale si ottiene:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     ((NOT FOR SOME f IN Familiari WHERE f.CapoFamiglia = i.Codice:
                  NOT (f.Sesso = 'm')) OR
          (NOT FOR SOME f IN Familiari WHERE f.CapoFamiglia = i.Codice:
                NOT (f.Sesso = 'f'))) AND
          FOR SOME d in Familiari WHERE d.CapoFamiglia = i.Codice :
               TRUE;
```

che in SQL diventa:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     ((NOT EXISTS (SELECT *
                        FROM   Familiari f
                        WHERE  f.CapoFamiglia = i.Codice
                        AND NOT (f.Sesso = 'm'))) OR
          (NOT EXISTS (SELECT *
                       FROM   Familiari f
                       WHERE  f.CapoFamiglia = i.Codice
                       AND NOT (f.Sesso = 'f')))) AND
          EXISTS (     SELECT *
                       FROM   Familiari f
                       WHERE  f.CapoFamiglia = i.Codice );
```

L'interrogazione si può anche scrivere senza sottoselect, utilizzando opportunamente le giunzioni e le funzioni di aggregazione:

```
SELECT    i.CognomeNome
FROM      Impiegati i
          JOIN Familiari f ON f.CapoFamiglia = i.Codice
GROUP BY  i.Codice, i.CognomeNome
HAVING    COUNT(DISTINCT f.Sesso) = 1;
```

14. Trovare il nome degli impiegati che hanno tutti i familiari a carico dello stesso sesso dell'impiegato.
La soluzione con il quantificatore universale è:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     (FOR ALL f IN Familiari WHERE f.CapoFamiglia = i.Codice:
              f.Sesso = i.Sesso)
          AND (FOR SOME f IN Familiari WHERE f.CapoFamiglia = i.Codice:
              TRUE );
```

con la doppia negazione si ottiene:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     (NOT FOR SOME f IN Familiari
               WHERE f.CapoFamiglia = i.Codice:
                     NOT (f.Sesso = i.Sesso))
          AND (FOR SOME f IN Familiari
                    WHERE f.CapoFamiglia = i.Codice:
                          TRUE );
```

che in SQL diventa:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     NOT EXISTS(SELECT *
                     FROM   Familiari f
                     WHERE  f.CapoFamiglia = i.Codice
                            AND NOT (f.Sesso = i.Sesso))
          AND EXISTS(SELECT *
                     FROM   Familiari f
                     WHERE  f.CapoFamiglia = i.Codice;
```

La query può anche essere scritta usando la giunzione e le funzioni di aggregazione:

```
SELECT    i.CognomeNome
FROM      Impiegati i
          JOIN Familiari f ON f.CapoFamiglia = i.Codice
WHERE     EXISTS (SELECT *
                  FROM   Familiari f
                  WHERE  f.CapoFamiglia = i.Codice
                         AND (f.Sesso = i.Sesso))
GROUP BY  i.Codice, i.CognomeNome
HAVING    COUNT (DISTINCT f.Sesso) = 1;
```

15. Trovare il nome degli impiegati che lavorano almeno a tutti i progetti dell'impiegato con codice 300.
La soluzione con il quantificatore universale è:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     FOR ALL x IN Partecipazioni WHERE x.Impiegato = 300:
              (FOR SOME y IN Partecipazioni
                   WHERE y.Impiegato = i.Codice:
                         x.Progetto = y.Progetto);
```

con la doppia negazione si ottiene:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     NOT FOR SOME x IN Partecipazioni
                WHERE x.Impiegato = 300:
            (NOT FOR SOME y IN Partecipazioni
                WHERE y.Impiegato = i.Codice:
                    x.Progetto = y.Progetto);
```

che in SQL diventa:

```
SELECT    i.CognomeNome
FROM      Impiegati i
WHERE     NOT EXISTS
            (SELECT *
             FROM   Partecipazioni x
             WHERE  x.Impiegato = 300
                    AND NOT EXISTS
                        (SELECT *
                         FROM   Partecipazioni y
                         WHERE  y.Impiegato = i.Codice
                                AND x.Progetto = y.Progetto));
```

16. Trovare il nome del dipartimento e il numero di impiegati nati dopo il 1950, per i dipartimenti con più di due impiegati.

```
CREATE    VIEW DipConAlmenoDueImp AS
SELECT    Numero, Nome
FROM      Impiegati
          JOIN Dipartimenti ON Afferisce = Numero
GROUP BY  Numero, Nome
HAVING    COUNT(*) > 2;

SELECT    Nome, COUNT(*) AS NImp
FROM      Impiegati
          JOIN DipConAlmenoDueImp ON Afferisce = Numero
WHERE     AnnoNascita > 1950
GROUP BY  Numero, Nome;
```

Il risultato non contiene i dipartimenti con più di due impiegati nati prima del 1950, che si trovano con l'interrogazione:

```
SELECT    *
FROM      DipConAlmenoDueImp
WHERE     FOR ALL x IN Impiegati WHERE x.Afferisce = Numero:
              x.AnnoNascita <= 1950;
```

con la doppia negazione si ottiene:

```
SELECT    *
FROM      DipConAlmenoDueImp
WHERE     NOT FOR SOME x IN Impiegati
              WHERE x.Afferisce = Numero:
                  NOT (x.AnnoNascita <= 1950);
```

che in SQL diventa:

```
SELECT    Nome, 0 AS NImp
FROM      DipConAlmenoDueImp
WHERE     NOT EXISTS(SELECT *
                     FROM   Impiegati x
                     WHERE  x.Afferisce = Numero
                     AND    x.AnnoNascita > 1950);
```

Quindi la soluzione completa è la seguente:

```
SELECT    Nome, COUNT(*) AS NImp
FROM      Impiegati
          JOIN DipConAlmenoDueImp ON Afferisce = Numero
WHERE     AnnoNascita > 1950
GROUP BY  Numero, Nome
  UNION
SELECT    Nome, 0 AS NImp
FROM      DipConAlmenoDueImp
WHERE     NOT EXISTS(SELECT *
                     FROM   Impiegati x
                     WHERE  x.Afferisce = Numero
                     AND    x.AnnoNascita > 1950);
```

Capitolo 7

SQL PER DEFINIRE E AMMINISTRARE BASI DI DATI

Esercizio 7.1
Si definisca uno schema relazionale con i comandi **CREATE TABLE** per trattare le informazioni e i vincoli d'integrità sui dipendenti di un'azienda, con attributi CodiceFiscale, CognomeNome, AnnoAssunzione e Stipendio, e sui loro familiari a carico, con attributi CognomeNome, AnnoNascita e RelazioneDiParentela.

Soluzione

```
CREATE   TABLE Impiegati (
         CodiceFiscale      CHAR(16)    NOT NULL,
         CognomeNome        CHAR(30)    NOT NULL,
         AnnoAssunzione     INTEGER     NOT NULL,
         Stipendio          INTEGER     NOT NULL,
         PRIMARY KEY        pk_Impiegati (CodiceFiscale)
         CHECK              (AnnoAssunzione > 1990)
                            AND (AnnoAssunzione < 2021));
CREATE   TABLE Familiari (
         CFCapoFamiglia     CHAR(16)    NOT NULL,
         CognomeNome        CHAR(30)    NOT NULL,
         AnnoNascita        INTEGER     NOT NULL,
         Parentela          CHAR(30)    NOT NULL,
         PRIMARY KEY        pk_Impiegati (CFCapoFamiglia, CognomeNome)
         FOREIGN KEY        fk_FamiliareImpiegato (CapoFamiglia)
                            REFERENCES Impiegati
                            ON DELETE CASCADE
         CHECK              (AnnoNascita > 1890)
                            AND (AnnoNascita < 2021));
```

Esercizio 7.2
Si supponga che sia stato definito uno schema relazionale con le istruzioni:

```
CREATE TABLE R (K CHAR(8) NOT NULL, A CHAR(8), B CHAR(8) )
PRIMARY KEY(K)
GRANT SELECT ON R TO caio
```

e siano stati immessi dei dati in R. Usando i comandi:

```
DROP TABLE Nome;
CREATE TABLE Nome (Attributo Tipo, ...) AS ExprSQL'
CREATE VIEW Nome (Attributo, ...) AS ExprSQL;
GRANT SELECT ON Nome TO Utente;
```

mostrare come si possa modificare lo schema in modo che i dati di R vengano memorizzati esclusivamente nelle tabelle:

```
R1( K CHAR(8) NOT NULL, A CHAR(8))
R2( K CHAR(8) NOT NULL, B CHAR(8))
```

e l'utente "caio" possa continuare a lavorare sulla base di dati come se esistesse la tabella:

R(K **INTEGER**(8) **NOT NULL**, A **INTEGER**(8), B **INTEGER**(8)).

Soluzione

```
CREATE TABLE R1 (K CHAR (8) NOT NULL ,
                 A CHAR (8),
                 PRIMARY KEY(K))
    AS SELECT    K, A
       FROM      R;

CREATE TABLE R3 (K CHAR (8) NOT NULL ,
                 B CHAR (8),
                 PRIMARY KEY(K))
    AS SELECT    K, B
       FROM      R;

DELETE TABLE R;

CREATE VIEW R
    AS SELECT  R1.K, A, B
       FROM    R1
               JOIN R2 ON R1.K = R2.K;
GRANT SELECT ON R TO Caio;
```

Esercizio 7.3
Definire lo schema per la base di dati relazionale ottenuta dall'Esercizio 4.3.

Soluzione

```
CREATE TABLE Dipartimenti (
    Numero              INTEGER         NOT NULL,
    Nome                VARCHAR (16)    NOT NULL,
    Città               VARCHAR (16)    NOT NULL,
    PRIMARY KEY         (Numero),
    UNIQUE              (Nome, Città)    );


CREATE TABLE Impiegati (
    Codice              INTEGER         NOT NULL,
    CognomeNome         VARCHAR (20)    NOT NULL,
    AnnoNascita         INTEGER         NOT NULL,
    Sesso               VARCHAR (1),
    Afferisce           INTEGER         NOT NULL,
    PRIMARY KEY         (Codice),
    FOREIGN KEY         (Afferisce)
    REFERENCES          Dipartimenti    ON DELETE CASCADE );


CREATE TABLE Familiari (
    CapoFamiglia        INTEGER         NOT NULL,
    Nome                VARCHAR(16)     NOT NULL,
    AnnoNascita         INTEGER,
    RelazioneParentela  VARCHAR(8)      NOT NULL,
    Sesso               VARCHAR(1)      NOT NULL,
    PRIMARY KEY         (CapoFamiglia, Nome),
    FOREIGN KEY         (CapoFamiglia)
    REFERENCES          Impiegati       ON DELETE CASCADE );


CREATE TABLE Responsabili (
    Codice              INTEGER         NOT NULL,
    AnnoNomina          INTEGER         NOT NULL,
    PRIMARY KEY         (Codice),
    FOREIGN KEY         (Codice)
    REFERENCES          Impiegati       ON DELETE CASCADE   );
```

```
CREATE TABLE Dipendenti (
    Codice              INTEGER         NOT NULL,
    CoordinatoDa        INTEGER,
    PRIMARY KEY         (Codice),
    FOREIGN KEY         (Codice)
    REFERENCES          Impiegati       ON DELETE CASCADE,
    FOREIGN KEY         (CoordinatoDa)
    REFERENCES          Responsabili    ON DELETE SET NULL  );

CREATE TABLE Direttori (
    Codice              INTEGER         NOT NULL,
    AnnoNomina          INTEGER         NOT NULL,
    Dirige              INTEGER         NOT NULL,
    PRIMARY KEY         (Codice),
    FOREIGN KEY         (Codice)
    REFERENCES          Impiegati       ON DELETE CASCADE,
    FOREIGN KEY         (Dirige)
    REFERENCES          Dipartimenti    ON DELETE CASCADE   );

CREATE TABLE Progetti (
    NumeroP             INTEGER         NOT NULL,
    NomeP               VARCHAR (10)    NOT NULL,
    GestitoDa           INTEGER         NOT NULL,
    PRIMARY KEY         (NumeroP),
    FOREIGN KEY         (GestitoDa)
    REFERENCES          Dipartimenti    ON DELETE CASCADE   );

CREATE TABLE Partecipazioni (
    Impegno             INTEGER         NOT NULL,
    Progetto            INTEGER         NOT NULL,
    Impiegato           INTEGER         NOT NULL,
    PRIMARY KEY         (Impiegato, Progetto) ,
    FOREIGN KEY         (Impiegato)
    REFERENCES          Impiegati       ON DELETE CASCADE ,
    FOREIGN KEY         (Progetto)
    REFERENCES          Progetti        ON DELETE CASCADE  );
```

```
CREATE    VIEW DatiDirettori
          (Codice, CognomeNome, Sesso, AnnoNascita, Afferisce,
          Dirige, AnnoNomina)
          AS
SELECT    i.Codice, i.CognomeNome, i.Sesso, i.AnnoNascita,
          i.Afferisce, d.Dirige, d.AnnoDiNomina
FROM      Impiegati i
          JOIN Direttori d ON i.Codice = d.Codice;

CREATE    VIEW DatiResponsabili
          (Codice, CognomeNome, Sesso, AnnoNascita, Afferisce,
          AnnoNomina)
          AS
SELECT    i.Codice, i.CognomeNome, i.Sesso, i.AnnoNascita,
          i.Afferisce, r.AnnoDiNomina
FROM      Impiegati i
          JOIN Responsabili r ON i.Codice = r.Codice;

CREATE    VIEW DatiDipendenti
          (Codice, CognomeNome, Sesso, AnnoNascita, Afferisce,
          CoordinatoDa)
          AS
SELECT    i.Codice, i.CognomeNome, i.Sesso, i.AnnoNascita,
          i.Afferisce, d.CoordinatoDa
FROM      Impiegati i
          JOIN Dipendenti d ON .Codice = d.Codice;
```

Esercizio 7.4
Si mostri come trattare il vincolo di chiave esterna con i *trigger*.

Soluzione
Supponiamo di aver definito le seguenti due tabelle.

```
CREATE TABLE R (
    K1              INTEGER     NOT NULL,
    A               INTEGER     NOT NULL,
    PRIMARY KEY  (K1) );
CREATE TABLE S (
    K2              INTEGER     NOT NULL,
    FK1             INTEGER     NOT NULL,
    PRIMARY KEY  (K2) );
```

Rafforziamo il vincolo di integrità della chiave esterna K2 in S rispetto alla relazione R con dei trigger. Per semplicità supponiamo che il vincolo sia di tipo **ON DELETE NO ACTION**.

```
CREATE TRIGGER InserisciInS
BEFORE INSERT ON S
BEFORE UPDATE OF FK1 ON S
DECLARE
    Esistente INTEGER;
BEGIN
    SELECT    COUNT(*) INTO Esistente
    FROM      R
    WHERE     K1 = :new.FK1;
    IF  Esistente = 0
    THEN
        RAISE.APPLICATION_ERROR('Nessun elemento corrispondente in R');
    END IF
END;

CREATE TRIGGER CancellaInR
BEFORE DELETE ON R
DECLARE
    Esistente INTEGER;
BEGIN
    SELECT    COUNT(*) INTO Esistente
    FROM      S
    WHERE     FK1 = :old.K1;
    IF  Esistente > 0
    THEN
        RAISE.APPLICATION_ERROR('Elementi esistenti in S');
    END IF
END;
```

Esercizio 7.5
Si ricorda che date due sottoclassi C_1 e C_2 di una classe C, diciamo che:

a) C_1 e C_2 soddisfano il vincolo di copertura di C se $C_1 \cup C_2 = C$;
b) C_1 e C_2 soddisfano il vincolo di disgiunzione se non hanno nessun elemento in comune.

In generale si possono quindi avere quattro tipi di situazioni:

a) copertura disgiunta o partizione (vincolo di copertura e di disgiunzione);
b) copertura non disgiunta (solo vincolo di copertura);

c) sottoinsiemi disgiunti (solo vincolo di disgiunzione);
d) sottoinsiemi non disgiunti (nessun vincolo).

Si supponga di rappresentare C_1, C_2 e C con tre relazioni RC_1, RC_2 ed RC, con RC_1 ed RC_2 che contengono gli attributi propri di C_1 e C_2 e una chiave esterna per RC. Si supponga di poter dichiarare nello schema solo il vincolo di chiave primaria e di chiave esterna. Si mostri se è possibile rappresentare i vincoli dei quattro tipi di sottoclassi con il comando **CREATE TABLE**.

Soluzione
Attraverso il vincolo di integrità referenziale dato dalla chiave esterna si può garantire solo il vincolo (d) delle sottoclassi, sottinsiemi non disgiunti. Infatti, non si può garantire il vincolo di copertura, perché è sempre possibile inserire un elemento in RC senza necessariamente inserirlo in RC_1 o RC_2, indipendentemente dalle chiavi primarie ed esterne definite nelle tre tabelle. Analogamente, è impossibile garantire il vincolo di disgiunzione, perché si possono inserire ennuple con due chiavi uguali nelle relazioni RC_1, RC_2.

Esercizio 7.6
Si risolva l'esercizio precedente usando i *trigger*.

Soluzione
Vincolo di Disgiunzione, Se è presente il vincolo di disgiunzione, le operazioni consentite sono le seguenti:

a) Inserire una ennupla in RC.
b) Inserire una ennupla in RC1 se esiste in RC, ma non esiste in RC2.
c) Inserire una ennupla in RC2 se esiste in RC, ma non esiste in RC1.

La condizione di esistenza in RC per le operazioni (b) e (c) viene controllata con il vincolo di chiave esterna, mentre è necessario definire dei trigger che, prima di inserire in una sottoclasse, controllano che l'elemento non sia già presente nell'altra. Supponiamo che entrambe le relazioni RC1 e RC2 abbiamo K come chiave primaria e chiave esterna per RC.

```
CREATE TRIGGER InserisciInRC1
BEFORE INSERT ON RC1
DECLARE
        Esistente INTEGER;
BEGIN
        SELECT    COUNT(*) INTO Esistente
        FROM      RC2
        WHERE     K = :new.K;
```

```
    IF  Esistente > 0
    THEN
        RAISE.APPLICATION_ERROR('Elemento già presente in RC2');
    END IF
END;
```

Analogamente per RC2.

Vincolo di Copertura. Se è presente il vincolo di disgiunzione, le operazioni di inserzione consentite sono le seguenti:

a) Inserire una ennupla in RC1 e contestualmente in RC.
b) Inserire una ennupla in RC2 e contestualmente in RC.

Oltre a questo, è necessario proibire l'inserzione di un elemento di RC.

Le operazioni (a) e (b) richiedono la scrittura di due procedure memorizzate che prendono come parametri tutti i dati e si occupano di effettuare due inserzioni.

Il trigger da scrivere quindi è quello che impedisce l'inserimento di un elemento nella tabella RC.

```
CREATE TRIGGER InserisciInRC1
BEFORE INSERT ON RC1
BEGIN
    RAISE.APPLICATION_ERROR('Inserire elementi solo in RC1 e RC2');
END;
```

Capitolo 8

SQL PER PROGRAMMARE LE APPLICAZIONI

Esercizio 8.1
Si consideri il seguente frammento di codice SQLJ:

```
#sql [contesto]
        SELECT   Ammontare INTO  :ammontare
        FROM     Ordini
        WHERE    CodiceAgente = :numAgente
```

L'elaborazione del codice procede in tre fasi: (a) precompilazione dei frammenti SQL in Java, (b) compilazione del programma Java risultante, (c) esecuzione. Durante l'esecuzione, il controllo si alterna tra macchina astratta Java e il DBMS. Esemplifichiamo ora alcuni motivi per cui tale codice potrebbe essere scorretto. Immaginando che ogni errore sia scoperto prima possibile, specificare chi dei quattro attori (precompilatore, compilatore, macchina astratta Java e DBMS) segnala ciascun errore.

a) *Sintassi SQL*: il programmatore potrebbe scrivere WEHRE anziché **WHERE**.
b) *Nomi*: il programmatore potrebbe avere sbagliato a scrivere il nome della relazione, oppure quello dell'attributo Ammontare, oppure quello della variabile :ammontare.
c) *Tipi*: il tipo di Ammontare e quello di :ammontare potrebbero essere incompatibili.
d) *Variazione dello schema*: quando il programma viene eseguito la relazione Ordini potrebbe essere stata cancellata, oppure il tipo dell'attributo Ammontare potrebbe essere cambiato.
e) *Univocità*: il valore di :NumAgente potrebbe non essere associato ad alcun agente, oppure essere associato a più agenti.

Soluzione

a) *Sintassi SQL*: il programmatore potrebbe scrivere WEHRE anziché **WHERE**.

Il pre-processore SQLJ analizza l'uso della corretta sintassi SQL prima di generare il codice Java, quindi può identificare questo problema.

b) *Nomi*: il programmatore potrebbe avere sbagliato a scrivere il nome della relazione, oppure quello dell'attributo Ammontare, oppure quello della variabile :ammontare.

   Per il nome della relazione o dell'attributo il pre-processore SQLJ può connettersi ad un database di esempio, collezionando quindi i nomi corretti delle tabelle e dei loro attributi, conseguenza il pre-processore è in grado di identificare questo tipo di errore. Per la varibile :ammontare il preprocessore non ha alcuna conoscenza dei nomi delle variabili Java, e quindi l'errore sarà segnalato successivamente dal compilatore Java.

c) *Tipi*: il tipo di Ammontare e quello di :ammontare potrebbero essere incompatibili.

   Risposta: il pre-processore SQLJ non sa nulla dei tipi di Java, quindi procederà alla generazione del codice Java quando avrà verificato la correttezza sintattica del codice ospite. Il compilatore Java si accorgerà del problema in fase di type-checking.

d) *Variazione dello schema*: quando il programma viene eseguito la relazione Ordini potrebbe essere stata cancellata, oppure il tipo dell'attributo Ammontare potrebbe essere cambiato.

   Nè il pre-processore SQLJ, nè il compilatore Java possono avere informazioni sullo stato del database a runtime. Solo il DBMS potrà sapere se la relazione Ordini è ancora presente o meno in fase di esecuzione della query.

e) *Univocità*: il valore di :NumAgente potrebbe non essere associato ad alcun agente, oppure essere associato a più agenti.

   Anche in questo caso nè il pre-processore SQLJ nè il compilatore Java possono avere informazioni sul comportamento del database a runtime. Dal punto di vista del DBMS è lecito ritornare nessun risultato o più di un risultato, quindi solo il runtime di Java potrà accorgersi del problema.

Esercizio 8.2
Si consideri la versione API del frammento di codice dell'esempio precedente.

```
PreparedStatement pstmt =
    con.prepareStatement(
        "SELECT Ammontare
          FROM Ordini WHERE CodiceAgente = ?");
pstmt.setString(1, codAgente);
risultato = pstmt.executeQuery();
int ammontare = risultato.next().getInt(0);
```

In questo caso l'elaborazione di tale frammento procede come segue: compilazione del programma Java, esecuzione da parte della macchina astratta Java che interagisce con il DBMS. Anche in questo caso si cerchi di individuare in quale momento verrebbe scoperto ciascuno degli errori sopra elencati.

Soluzione

a) *Sintassi SQL*: il programmatore potrebbe scrivere WEHRE anziché **WHERE**.

La query SQL è rappresentata da una stringa di caratteri. Le librerie JDBC in generale non fanno il parsing di questa stringa e la inviano direttamente al DBMS, il quale segnalerà l'errore al momento dell'esecuzione della query.

b) *Nomi*: il programmatore potrebbe avere sbagliato a scrivere il nome della relazione, oppure quello dell'attributo Ammontare, oppure quello della variabile :ammontare.

L'API non conosce la struttura del database, quindi anche in questo caso sarà il DBMS a segnalare l'errore.

c) *Tipi*: il tipo di Ammontare e quello di :ammontare potrebbero essere incompatibili.

L'API non può conoscere il tipo corretto di Ammontare e dal suo punto di vista l'esecuzione della query è lecita. Il DBMS riceve una query corretta e produce il risultato atteso. Ci sarà però un problema di conversione di tipo durante l'esecuzione da parte della macchina virtuale Java, quandoproverà a convertire il risultato della query ad intero, per esempio tramite il metodo getInt().

d) *Variazione dello schema*: quando il programma viene eseguito la relazione Ordini potrebbe essere stata cancellata, oppure il tipo dell'attributo Ammontare potrebbe essere cambiato.

Solo il DBMS potrà sapere se la relazione Ordini è ancora presente o meno in fase di esecuzione della query.

e) *Univocità*: il valore di :NumAgente potrebbe non essere associato ad alcun agente, oppure essere associato a più agenti.

Dal punto di vista del DBMS è lecito ritornare nessun risultato o più di un risultato, quindi solo il runtime di Java potrà accorgersi del problema durante l'esecuzione del programma.

Esercizio 8.3
Specificare vantaggi e svantaggi della programmazione di applicazioni usando un linguaggio integrato anziché un'API.

Soluzione
L'uso di un linguaggio integrato invece di API ha i seguenti vantaggi:

a Permette di scoprire durante la compilazione errori, sia sintattici che semantici, nella scrittura delle query, e di segnalarli in maniera comprensibile al programmatore.
b Semplifica la scrittura e la manutenzione del programma, con scambi di valori fra i dati del programma e quelli del Database in maniera automatica, senza bisogno di aggiungere codice di conversione.

I principali svantaggi sono la scarsa diffusione e la difficile portabilità dei programmi scritti con questi linguaggi, disponibili solo per alcuni DBMS e in generali diversi fra i vari DBMS.

Esercizio 8.4
Si considerino le seguenti applicazioni in ambito bancario. Indicare il livello di isolamento più opportuno per ciascuna di esse, spiegando la risposta.

a) Per ogni cliente della banca, contare le operazioni effettuate negli ultimi 500 giorni, ed aggiungere il nome del cliente ad un elenco se le operazioni sono più di trecento. L'elenco servirà a scopi di marketing.
b) Effettuare un trasferimento fondi, sottraendo un ammontare da un conto per aggiungerlo ad un altro.
c) Gestire un prelievo allo sportello come segue: il cassiere legge sul terminale il saldo corrente del cliente; se questo supera la cifra richiesta dal cliente, il cassiere comunica al sistema la cifra ed effettua il pagamento (specificare quali di queste operazioni sarebbero racchiuse nella transazione).

Soluzione

a) Per ogni cliente della banca, contare le operazioni effettuate negli ultimi 500 giorni, ed aggiungere il nome del cliente ad un elenco se le operazioni sono più di trecento. L'elenco servirà a scopi di marketing.

   Visto che la transazione viene usata a fini di marketing, è plausibile che qualche sporadico errore nei risultati non rappresenterà un grosso problema. Per questo motivo è sufficiente un livello di isolamento **READ UNCOMMITTED**.

b) Effettuare un trasferimento fondi, sottraendo un ammontare da un conto per aggiungerlo ad un altro.
   La transazione dovrebbe avere il livello di isolamento **REPEATABLE READ**: infatti due transazioni di questo tipo, se avvengono contemporaneamente sullo

stesso conto corrente, potrebbero interferire nell'aggiornamento di uno dei due saldi con un livello inferiore di isolamento.

c) Gestire un prelievo allo sportello come segue: il cassiere legge sul terminale il saldo corrente del cliente; se questo supera la cifra richiesta dal cliente, il cassiere comunica al sistema la cifra ed effettua il pagamento (specificare quali di queste operazioni sarebbero racchiuse nella transazione).

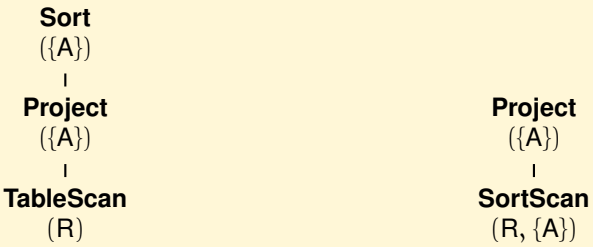
   In questo caso è necessario fare due transazioni, invece di una, perché altrimenti in caso di gestione delle transazioni con il blocco dei dati, i dati verrebbero bloccati per troppo tempo. e ciò per il tempo richiesto per l'operatore umano per effettuare o meno il prelievo. La prima transazione farà solamente la lettura del saldo, e dato che questa deve fornire un dato corretto, sarà necessario il livello di isolamento **READ COMMITTED**. La seconda, di aggiornamento, dovrà rileggere il valore corrente del saldo e fare l'operazione di modifica se permangono le condizioni corrette, e quindi essere fatta con il livello di isolamento **REPEATABLE READ**, per garantire la consistenza dell'aggiornamento.

Capitolo 9

REALIZZAZIONE DEI DBMS

Esercizio 9.1
Dire quali delle seguenti affermazioni è vera o falsa e giustificate la risposta:

a) I seguenti piani di accesso per l'interrogazione
SELECT A **FROM** R **ORDER BY** A
non sono equivalenti:



b) Operatore logico e operatore fisico sono sinonimi.
c) Ad ogni interrogazione SQL corrispondono più alberi logici.
d) Ad ogni albero logico corrispondono più alberi fisici.
e) La gestione della concorrenza con il blocco dei dati non crea condizioni di stallo.
f) Con il metodo disfare–rifare, nessun dato modificato da una T può essere riportato nella BD prima che il corrispondente record del giornale sia scritto nella memoria permanente.
g) Con il metodo rifare, tutte le modifiche di una T devono essere riportate nella BD prima che il record di commit sia scritto nel giornale.

Soluzione

a) *Falso*. Il primo piano esegue l'interrogazione con un algoritmo che recupera prima i record di R, poi li proietta su A e infine li ordina. Il secondo piano usa un algoritmo che prima ordina R poi recupera i record ordinati e infine li proietta su A, producendo lo stesso risultato.

b) *Falso*. Un operatore logico è un operatore dell'algebra relazionale, mentre un operatore fisico è un algoritmo per realizzare un operatore logico sfruttando le strutture di memorizzazione disponibili nella macchina fisica.
c) *Vero*. L'albero logico iniziale può essere trasformato in modi diversi applicando le regole di equivalenza dell'algebra relazionale.
d) *Vero*. Un operatore logico può essere realizzato con algoritmi diversi.
e) *Falso*. La tecnica del blocco dei dati può creare situazioni di stallo.
f) *Vero*. Il metodo disfare–rifare segue le regole per *disfare* e per *rifare*: prima di modificare la BD scrive nel giornale la vecchia versione e la nuova versione dei dati modificati.
g) *Falso*. Il metodo rifare non prevede che le modifiche delle transazioni siano riportate nella BD prima che il record di commit sia scritto nel giornale.

Esercizio 9.2
Si considerino due relazioni unarie $R(A)$ e $S(B)$ con i seguenti record:
$R = \{(A := 7), (A := 2), (A := 8), (A := 3), (A := 1), (A := 3), (A := 6)\}$
$S = \{(B := 4), (B := 2), (B := 1), (B := 3), (B := 2), (B := 7), (B := 3)\}$

Mostrare (a) il contenuto di un indice sull'attributo A di R e (b) il risultato prodotto dalla giunzione $R \underset{A=B}{\bowtie} S$ usando il NestedLoop.

Soluzione

IdxA

A	TID
1	5
2	2
3	4
3	6
6	7
7	1

Risultato

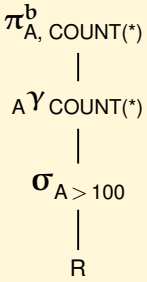
A	B
7	7
2	2
2	2
3	3
3	3
1	1
3	3
3	3

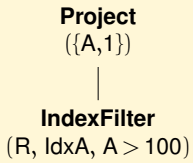
Esercizio 9.3
Si consideri la relazione R(A, B, C), con chiave primaria A, e l'interrogazione:

SELECT A, **COUNT**(∗)
FROM R
WHERE A > 100
GROUP BY A;

Dare l'albero logico iniziale dell'interrogazione e un possibile piano di accesso.
Come cambia la soluzione se nella **SELECT** ci fosse **DISTINCT** A, **COUNT**(∗)?

Soluzione



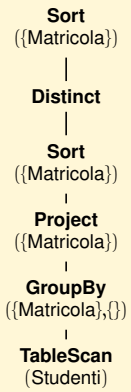


Non occorre raggruppare perché A è chiave. La soluzione non cambia se nella **SELECT** ci fosse **DISTINCT**.

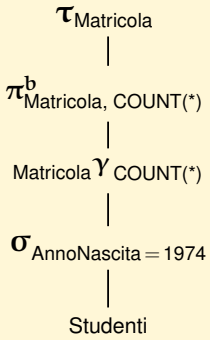
Esercizio 9.4
Si consideri la relazione Studenti(Matricola, Nome, AnnoNascita), ordinata sulla chiave primaria Matricola, e l'interrogazione:

```
SELECT    DISTINCT Matricola, COUNT(*)
FROM      Studenti
WHERE     AnnoNascita = 1974
GROUP BY  Matricola
ORDER BY  Matricola;
```

Dare l'albero logico iniziale dell'interrogazione e si dica se il seguente piano d'accesso produce il risultato cercato. Se non va bene, lo si modifichi in tre modi:
a) aggiungendo prima solo le parti mancanti (operatori e parametri),
b) semplificando poi il piano eliminando operatori inutili e
c) modificando infine il piano supponendo che esista un indice su AnnoNascita:

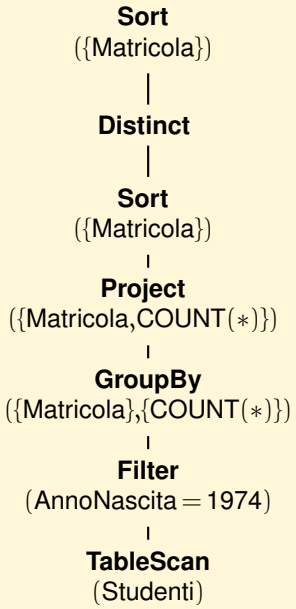


Soluzione
Albero logico iniziale dell'interrogazione:

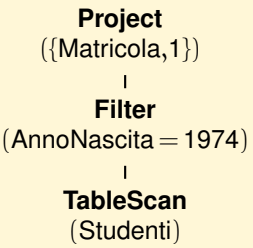


Il piano di accesso non produce il risultato dell'interrogazione.

a) Completiamo il piano di accesso aggiungendo le parti mancanti, che sono il filtro su AnnoNascita e la funzione di aggregazione COUNT(*):



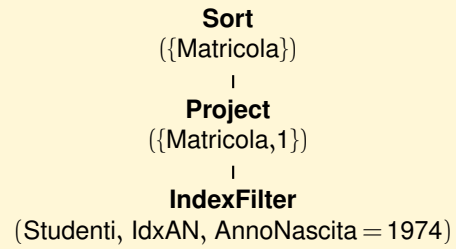
b) Adesso eliminiamo gli operatori superflui:



Partendo dal basso si eliminano:

- **GroupBy**: l'attributo di raggruppamento è chiave e quindi non serve raggruppare.
- **Project**:: l'operatore **GroupBy** restituisce record con campi Matricola, COUNT(*) e quindi il **Project** è inutile,
- **Sort**: la tabella è memorizzata ordinata sulla chiave Matricola e quindi non occorre ordinare,
- **Distinct**: tra gli attributi dei record che arrivano all'operatore **Distinct** è compresa una chiave, dunque non possono esistere record uguali,
- **Sort**: non serviva prima, non serve adesso.

c) Considerando l'indice su AnnoNascita il piano diventa:



Esercizio 9.5
Si consideri il seguente schema relazionale:

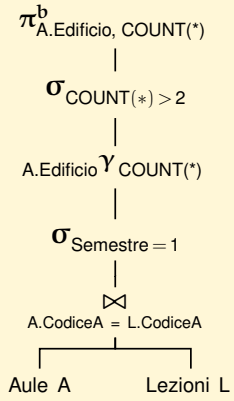
Aule(CodiceA, Edificio, Capienza)
Lezioni(CodiceA, CodiceC, Ora, GiornoSett, Semestre)
Corsi(CodiceC, NomeC, Docente)

e l'interrogazione:

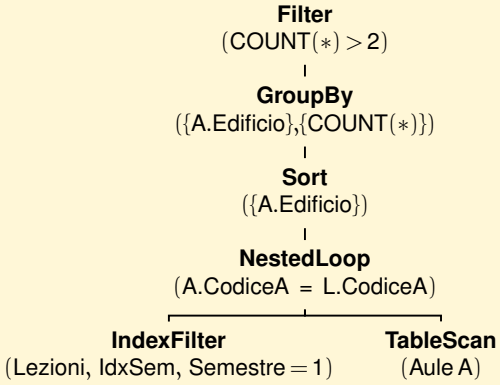
```
SELECT      A.Edificio, COUNT(*)
FROM        Aule A, Lezioni L
WHERE       A.CodiceA = L.CodiceA AND Semestre = 1
GROUP BY    A.Edificio
HAVING      COUNT(*) > 2;
```

Si dia l'albero logico iniziale dell'interrogazione e un piano di accesso che utilizzi indici:
a) nel caso di giunzione con l'operatore NestedLoop e
b) nel caso di giunzione con l'operatore IndexNestedLoop.

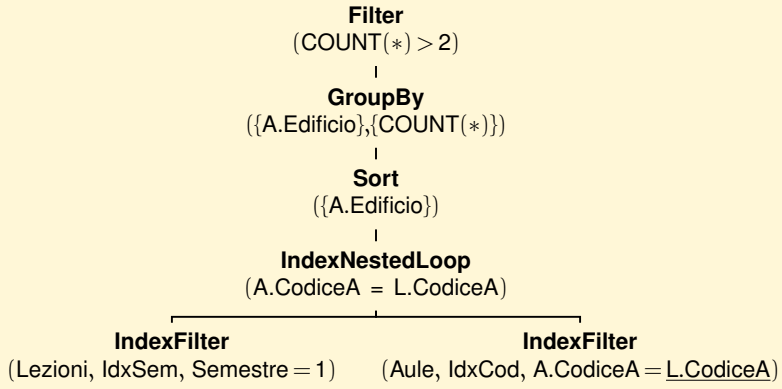
Soluzione
Albero logico iniziale dell'interrogazione:



a) *Piano di accesso con* **NestedLoop**:



b) *Piano di accesso con* **IndexNestedLoop**:



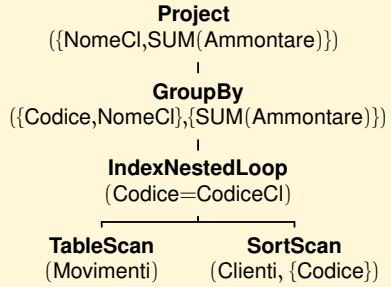
Esercizio 9.6
Si consideri la base di dati:

Clienti(Codice, NomeCl, AnnoNascita), con chiave primaria Codice
Movimenti(CodiceCl, Ammontare, Tipo), con chiave esterna CodiceCl

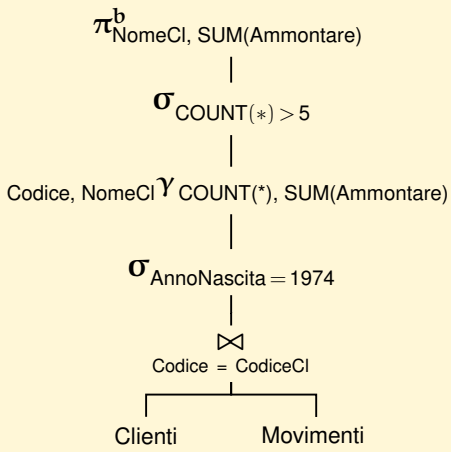
e l'interrogazione:

```
SELECT      NomeCl, SUM(Ammontare)
FROM        Clienti, Movimenti
WHERE       Codice = CodiceCl AND AnnoNascita = 1974
GROUP BY    Codice, NomeCl
HAVING      COUNT(∗) > 5 ;
```

Dare l'albero logico iniziale dell'interrogazione e si dica
a) se il seguente piano d'accesso è corretto,
b) se produce il risultato cercato.
Se non va bene, lo si modifichi aggiungendo le parti mancanti (operatori e parametri).



Soluzione
Albero logico iniziale dell'interrogazione:



a) il piano non è corretto perché mancano i filtri per AnnoNascita = 1974 e per **HAVING**, il **GROUP BY** richiede i record ordinati su Codice, NomeCl, il **SortScan** deve essere un **IndexFilter** per la presenza dell'**IndexNestedLoop**, e quindi
b) non produce il risultato cercato. Un piano corretto è: