Dipartimento del Tesoro

SQL

1 Database

1.1 COS'È UN DATABASE

Con il termine database (base di dati) si indica l'insieme dei dati utilizzati in uno specifico sistema informativo, di tipo aziendale, scientifico, amministrativo o altro.

Un database è composto da due diversi tipi di informazione, appartenenti a distinti livelli di astrazione:

I dati, che rappresentano le entità del sistema da modellare. Le proprietà di tali entità sono descritte in termini di valori (di tipo numerico, alfanumerico ecc.). I dati, inoltre, sono raggruppati o classificati in categorie in base alla loro struttura comune (per esempio Libri, Autori);

Le strutture (metadati), che descrivono le caratteristiche comuni delle varie categorie di dati, quali i nomi e i tipi dei valori delle proprietà.

Per esempio, il database utilizzato in una biblioteca può contenere, tra gli altri, i dati di Figura 1.1.

La struttura di tali dati può, in uno specifico database, essere descritta nel modo seguente:

```
Libri:
Codice-Libro:    numerico
Titolo:          testo(50)
Collocazione:    testo(10)
```

Libri

CODICE_LIBRO	TITOLO	COLLOCAZIONE
1	LE RETI DI COMPUTER	1/C/2
2	IL MANUALE DI SQL	3/H/8
3	TECNICHE DI PROGRAMMAZIONE	3/P/2
4	C++ MANUALE DL RIFERIMENTO	1/H/9

Autori

CODICE_AUTORE	COGNOME	NOME
1	PAOLO	RETI
2	GIOVANNI	MARIOTTI
3	MARIO	BIANCHI

Figura 1.1 Esempio di dati contenuti in un database utilizzato in una biblioteca

```
Autori:
Codice, Autore:numerico
Cognome:   testo(30)
Nome:      testo(30)
```

Un database deve rappresentare i diversi aspetti della realtà, e in particolare, oltre ai dati veri e propri, anche le relazioni tra i dati, ovvero le connessioni logiche presenti tra le varie categorie. Per esempio, deve essere rappresentata l'associazione che lega ciascun autore ai propri libri e viceversa. Il database deve inoltre rispondere ai seguenti requisiti.

- I dati devono essere organizzati con ridondanza minima, ossia non essere inutilmente duplicati. Questa condizione deriva dalla esigenza di evitare non solo l'utilizzo non necessario di risorse di memorizzazione, ma anche e soprattutto l'onere di gestione di copie multiple;

inoltre, se le informazioni relative a una categoria di dati vengono duplicate, si corre il rischio che un aggiornamento effettuato su una delle copie e non riportato sulle altre abbia conseguenze negative sulla consistenza e sull'affidabilità dell'insieme dei dati.

- I dati devono essere utilizzabili contemporaneamente da più utenti. Questa esigenza deriva dal punto precedente: è da evitare la situazione in cui ogni utente (o categoria di utenti) lavora su una propria copia dei dati, ed è necessario che esista un'unica versione dei dati, cui tutti gli utenti possano accedere; ciò implica la necessità che ciascun tipo di utenza possa avere una specifica visione dei dati e specifici diritti di accesso ai dati stessi. Inoltre, sono necessarie delle tecniche che consentano di evitare che l'attività dei vari utenti generi conflitti per l'uso contemporaneo degli stessi dati.
- I dati devono essere permanenti. Ciò implica non solo l'utilizzo di memorie di massa, ma anche l'applicazione di tecniche che preservino l'insieme dei dati in caso di malfunzionamento di un qualsiasi componente del sistema.

1.2 DAGLI ARCHIVI AI DBMS

Il DataBase Management System (DBMS), o sistema per la gestione di basi di dati, è il componente del software di base che consente la gestione di uno o più database secondo i requisiti indicati nel paragrafo precedente.

I primi DBMS furono sviluppati alla fine degli anni Sessanta per i calcolatori di fascia alta (mainframe) di grandi organizzazioni. Precedentemente, l'approccio tradizionalmente applicato al problema dell'archiviazione dei dati era quello dell'utilizzo diretto delle strutture del file system (Figura 1.2). La caratteristica saliente che differenzia un sistema per la gestione di basi di dati rispetto all'approccio file system è la presenza di un componente specializzato in tale compito di gestione (Figura 1.3).

Nell'approccio file system le applicazioni Appl. 1, Appl. 2... accedono direttamente agli archivi dei dati. Nello sviluppo di tali applicazioni deve essere conosciuta la struttura interna degli archivi; inoltre sono a carico delle applicazioni stesse la rappresentazione delle relazioni tra i dati, le tecniche che permettono l'utilizzo contemporaneo degli archivi da parte di più applicazioni e che garantiscono la permanenza dei dati. Questi ultimi aspetti spesso vengono delegati a strati sottostanti di software non specializzato, quale il sistema operativo.

La Figura 1.3 mostra come nell'approccio DBMS le applicazioni rivolgano le proprie richieste di accesso al database (DB) al DBMS, il quale gestisce i dati, svincolando le applicazioni da questo onere. La transizione dall'approccio file system all'approccio DBMS deriva dall'evoluzione verso una visione modulare del software: un modulo - il DBMS – è specializzato nella gestione del database; tutti gli altri si rivolgono a questo per la gestione dei dati. Si ottiene così un duplice scopo:

- le funzionalità di gestione del database sono raggruppate in un unico insieme realizzato come componente autonomo, rendendo così più agevole lo sviluppo delle applicazioni;
- nessuna applicazione può effettuare operazioni scorrette sul database.

Sebbene i DBMS siano stati progettati originariamente per i grandi sistemi centralizzati, sono oggi disponibili su tutte le principali piattaforme e architetture, quali personal computer e workstation, sia stand-alone sia interconnessi in reti di diversa estensione, dalle quelle locali a Internet

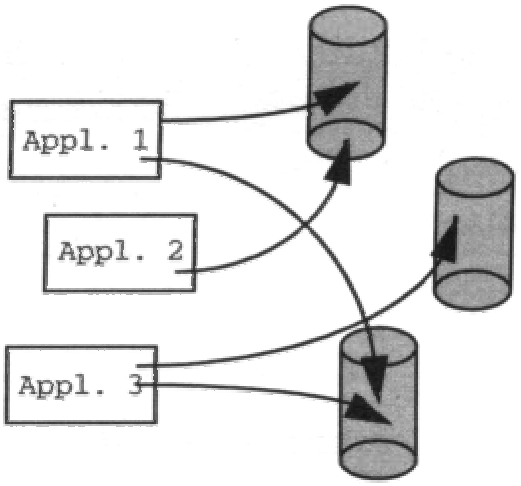

Figura 1.2 Approccio file system

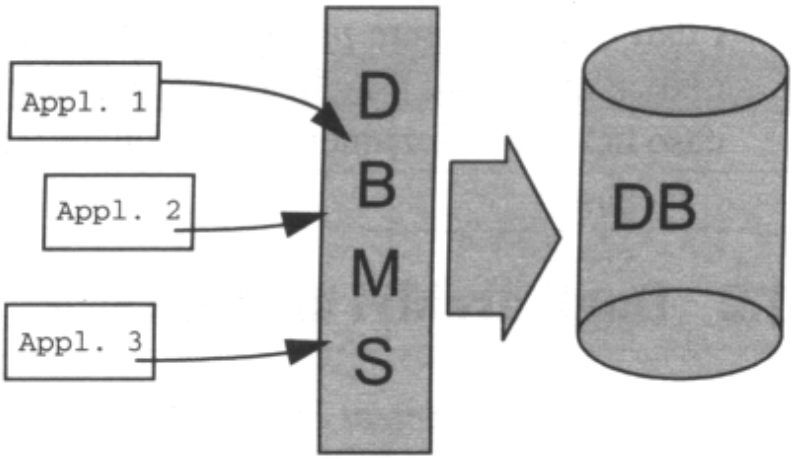

Figura 1.3 Approccio DBMS

1.3 VANTAGGI OFFERTI DAI DBMS

Rispetto alle soluzioni tradizionali al problema dell'archiviazione di dati basate sull'approccio file system, l'utilizzo di un DBMS comporta una serie di vantaggi che si traducono in una gestione dei dati più affidabile e coerente. In particolare ne derivano:

- indipendenza dei dati dall'applicazione;
- riservatezza nell'accesso ai dati;
- gestione dell'integrità fisica dei dati;
- gestione dell'integrità logica dei dati;
- sicurezza e ottimizzazione nell'uso dei dati.

Ciascuno di questi vantaggi viene approfondito nei prossimi paragrafi.

1.4 INDIPENDENZA DEI DATI DALL'APPLICAZIONE

In un ambiente tradizionale, che utilizza l'approccio di archiviazione su file system, i dati hanno senso e valore solo se interpretati dai programmi: la struttura dei dati e le applicazioni che gestiscono i dati stessi sono strettamente correlate. Le applicazioni devono conoscere il formato fisico dei dati (la struttura record), la loro localizzazione sui dispositivi di memorizzazione, le tecniche utilizzate per ottimizzare i tempi di ricerca (chiavi e strutture a indice).

Al contrario, in un ambiente che utilizza un DBMS i dati hanno una struttura che prescinde dalle applicazioni; queste, inoltre, non devono più fare riferimento alla struttura fisica dei dati. Ciò comporta notevoli vantaggi nella manutenzione del software del sistema informativo, poiché le modifiche alla struttura fisica dei dati non comportano la necessità di modificare le applicazioni preesistenti.

1.5 RISERVATEZZA NELL'ACCESSO AI DATI

Un sistema di gestione dati tradizionale basato su file system non consente visioni logiche diverse dello stesso insieme di dati. Ne consegue l'impossibilità di rendere disponibile a determinate categorie di utenza l'intero archivio e ad altre solo una parte, come nell'esempio di Figura 1.4. Verranno quindi create copie parziali dei dati, in base alle necessità delle specifiche categorie di utenza, e di conseguenza non verrà rispettato il requisito di minima ridondanza dei dati.

L'utilizzo di un DBMS consente invece visioni logiche multiple e contemporanee della stessa struttura fisica dei dati.

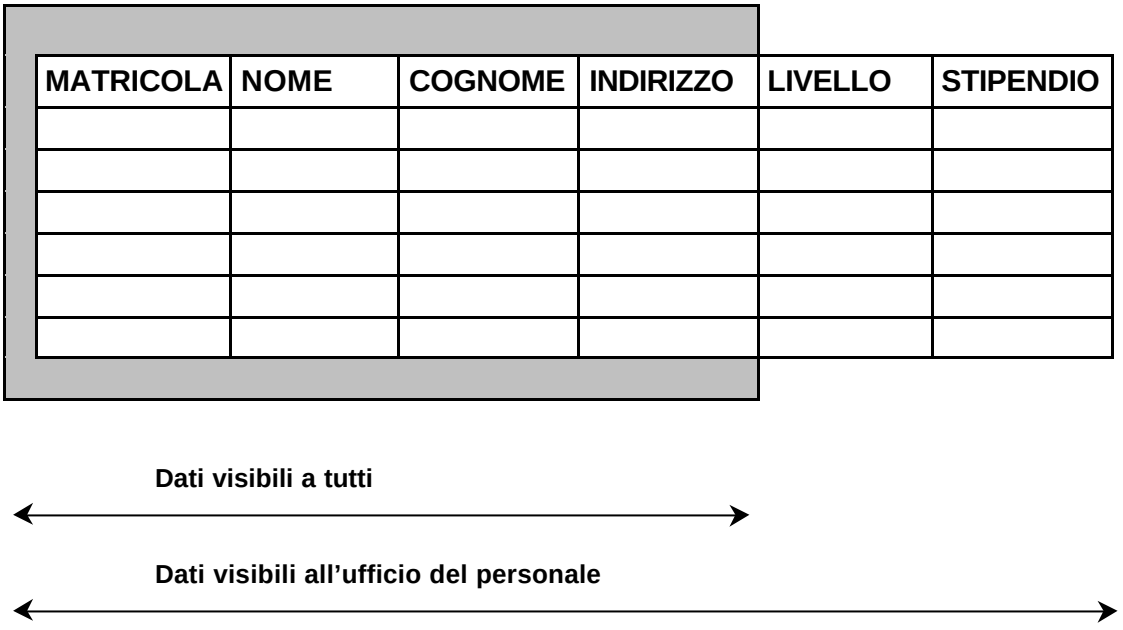

Figura 1.4 Visioni logiche diverse di una stessa struttura dati

1.6 GESTIONE DELL'INTEGRITÀ FISICA DEI DATI

Per soddisfare il requisito di persistenza dei dati è necessario garantire che il sistema di gestione dei dati mantenga nel tempo le informazioni registrate, e ne assicuri l'integrità in caso di caduta del sistema o di guasto ai dispositivi di memorizzazione.

Inoltre, è necessario che l'accesso contemporaneo da parte di più utenti al database sia effettuato in maniera controllata, per evitare stati di inconsistenza, che possono derivare da una scorretta sequenzializzazione delle singole azioni che ciascun utente effettua sull'insieme dei dati. A tale scopo i dati vengono parzialmente duplicati, contravvenendo in parte al principio di ridondanza minima; tuttavia questa operazione è completamente automatica e controllata dal DBMS, e non mette quindi a rischio la consistenza dei dati.

Uno dei compiti dell'amministratore del DBMS è appunto di impostare e configurare opportunamente il sistema al fine di garantire l'integrità fisica dei dati.

1.7 GESTIONE DELL'INTEGRITÀ LOGICA DEI DATI

Un aspetto sottovalutato nei primi DBMS, ma su cui attualmente viene posta molta attenzione, è quello della integrità logica dei dati. Essa consiste nella facoltà di stabilire dei vincoli in fase di definizione dei dati in modo che non sia possibile aggiungere dati o apportare modifiche ai dati stessi che non abbiano un senso nel contesto della realtà rappresentata. Si parla infatti in questo caso, oltre che di integrità logica, anche di integrità semantica o di significato.

Tali vincoli possono essere riferiti alle singole categorie di dati, per esempio per limitare dei valori a specifici intervalli o per definire le *n* possibili scelte di valori da immettere, o anche alle relazioni tra le categorie.

Un particolare caso di integrità logica su una relazione è quello della cosiddetta integrità referenziale (Figura 1.5). Essa viene utilizzata quando si vuole imporre che in una categoria di dati non vi possano essere elementi non associati, mediante la relazione in questione, a una seconda categoria, e pone conseguenti limitazioni sulle operazioni eseguibili sui dati. Per esempio, nella Figura 1.5 sono riportate le due categorie di dati, strettamente correlate tra loro, Ordini e Dettaglio Ordini. Definendo tra esse un vincolo di integrità referenziale il sistema impedirà la cancellazione di ordini con dettagli ancora esistenti e l'inserimento di dettagli non correlati a un ordine.

I controlli effettuati dal DBMS si baseranno sul fatto che ogni numero d'ordine N. Ordine in Ordini è correlato alle informazioni in Dettaglio Ordini aventi lo stesso N. Ordine.

1.8 SICUREZZA E OTTIMIZZAZIONE

Ulteriori vantaggi riscontrabili nell'uso di un DBMS derivano dalla centralizzazione della *gestione di utenti e risorse*.

Le funzioni di gestione degli utenti consentono all'amministratore del database di definire dei vincoli di accesso ai dati, ovvero di stabilire per ciascun utente i diritti di accesso (lettura, modifica ecc.) alle singole unità di informazione del database.

Il DBMS, inoltre, gestendo direttamente alcune risorse quali le periferiche e i task utente, consente di ottimizzarne l'utilizzo rispetto al caso in cui queste vengono controllate direttamente dal sistema operativo.

Ordini

N.ORDINE	DATA	CLIENTE

Dettaglio Ordini

N.Ordine	N.riga	Prodotto	Quantità

Figura 1.5 Esempio di categorie di dati con requisiti di integrità referenziale

1.9 MODELLI DEI DATI

Prima di trattare le differenti tipologie di DBMS, esaminiamo i passi che portano dalla realtà d'interesse alla realizzazione di un database, con riferimento ai modelli da utilizzare nelle varie fasi di questo cammino.

Un modello stabilisce le convenzioni per esprimere i diversi aspetti della realtà e costituisce un supporto alla sua rappresentazione. La rappresentazione generata seguendo le regole del modello viene definita *schema* (Figura 1.6).

Il processo di progettazione di un database si articola in tre fasi fondamentali, ciascuna delle quali si riferisce a un diverso livello di astrazione nella rappresentazione dei dati e delle relazioni tra essi. Le fasi riconosciute fondamentali nella progettazione di un database sono le seguenti: *progetto concettuale*, *progetto logico* e *progetto fisico*.

La scomposizione in fasi del processo di progettazione ha lo scopo di separare le diverse attività di risoluzione di problemi e di garantire la possibilità di modificare le soluzioni adottate ai livelli inferiori senza dover riprogettare quanto definito nei livelli superiori.

A ciascuna fase di progettazione corrispondono specifici modelli per la rappresentazione dei dati, ovvero tecniche per la rappresentazione degli aspetti rilevanti della realtà da modellare, definite da strumenti e vincoli specifici.

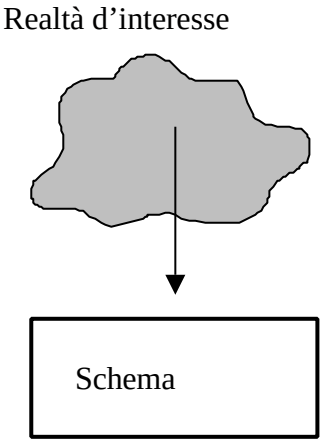

Figura 1.6 Realtà/modello/schema

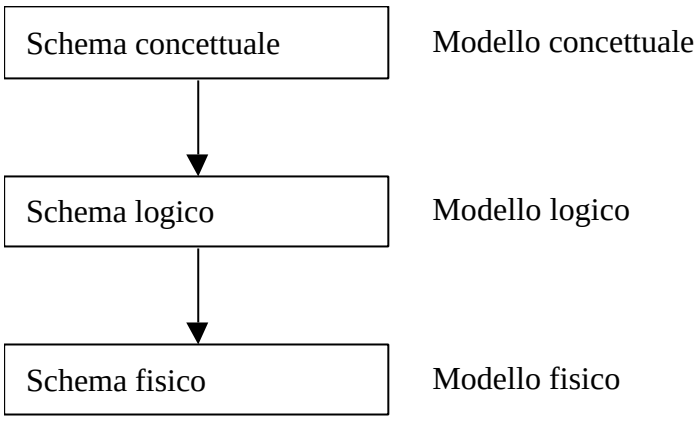

Figura 1.7 Fasi di progettazione di un database

1.10 PROGETTAZIONE CONCETTUALE

Obiettivo della fase di progettazione concettuale è la rappresentazione completa ed efficace della realtà interessante ai fini informativi, in maniera indipendente da qualsiasi specifico DBMS. Tale rappresentazione, detta *schema concettuale*, è la rappresentazione più astratta, ovvero più vicina alla logica umana, nella definizione di dati e relazioni.

I modelli dei dati usati nella progettazione concettuale vengono definiti *modelli semantici*; attualmente il più diffuso e noto tra essi è il modello *Entity-Relationship* (E-R, Entità-Relazione), elaborato originariamente da Peter Chen del Massachusetts Institute of Technology nel 1976, che introduce una rappresentazione grafica dello schema concettuale.

Il modello Entità-Relazione prevede come prima attività della progettazione concettuale l'individuazione di oggetti concreti o astratti rilevanti per il sistema informativo, e la loro classificazione in insiemi omogenei detti *entity set*. Graficamente un entity set, che tradurremo semplicemente con *entità*, viene rappresentato mediante un rettangolo.

Le proprietà caratteristiche di ciascuna entità vengono descritte mediante gli *attributi*. Per esempio, attributi dell'entità Veicolo possono essere: Targa, Cilindrata, Combustibile, Cavalli Fiscali,Velocità, Posti, Immatricolazione (Figura 1.8).

Ciascun attributo è caratterizzato da un nome e da un *dominio*, l'insieme dei valori che può assumere; per esempio, l'attributo Posti potrebbe assumere valori nel dominio { 1, 2, 3, 4, 5, 6, 7). Possiamo distinguere fra attributi elementari e attributi composti; questi ultimi sono costituiti da sottosequenze di valori; per esempio, l'attributo Indirizzo può essere composto dai sottoattributi Via e Numero Civico.

Il modello prevede la rappresentazione di *vincoli d'integrità* i quali descrivono le regole che soddisfano gli oggetti della realtà. Per esempio, nella Figura 1.8 è sottolineato l'attributo Targa a rappresentare il vincolo d'integrità per cui a ogni targa corrisponde uno e un solo veicolo.

Le dipendenze logiche o associazioni di interesse informativo tra i dati da rappresentare vengono espresse nello schema Entità-Relazione mediante relation set, che tradurremo semplicemente con *relazioni*, tra le corrispondenti entità. Graficamente una relazione viene rappresentata da un arco che collega i rettangoli che raffigurano le due entità correlate, interrotto da un ovale che contiene il nome della relazione.

Una relazione R tra due insiemi di entità E1 ed E2 viene classificata in base alla sua cardinalità.

Veicolo
Targa Cilindrata

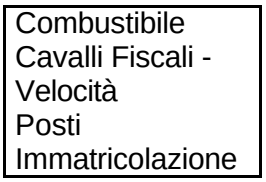

Figura 1.8 Esempio d'entità (entity set)

R ha cardinalità 1:1 (uno a uno) se a un elemento di E1 può corrispondere un solo elemento di E2 e viceversa. Un esempio di relazione 1:1 è la relazione tra nazioni e capitali di Figura 1.9.

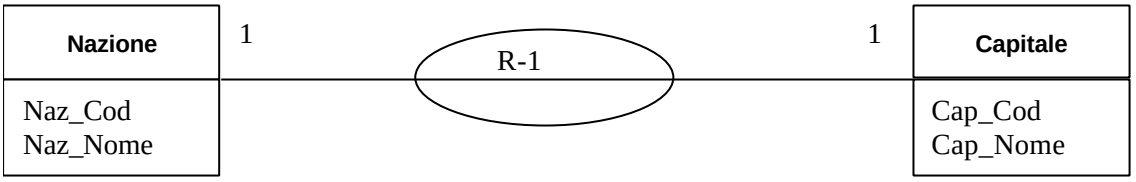

Figura 1.9 Rappresentazione di relazioni 1:1

R ha cardinalità 1:N (uno a molti) se a ogni elemento di E 1 possono corrispondere più elementi di E2, mentre a ogni elemento di E2 corrisponde al massimo un elemento di E1 . Un esempio di relazione 1:N è la relazione tra ordini e righe-ordine (Figura 1.10).

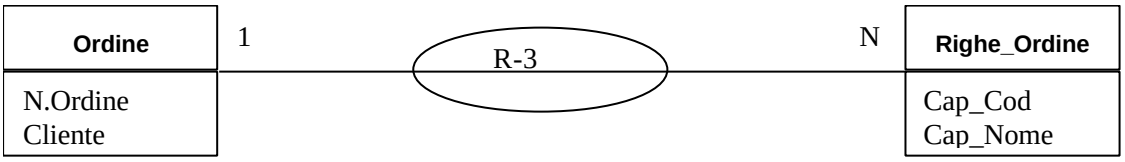

Figura 1.10 Rappresentazione di relazioni 1:N

R ha cardinalità N:M (molti a molti) se a ogni elemento di E 1 possono corrispondere più elementi di E2 e viceversa. Un esempio di relazione N:N è la relazione tra libri e autori (Figura 1.11).

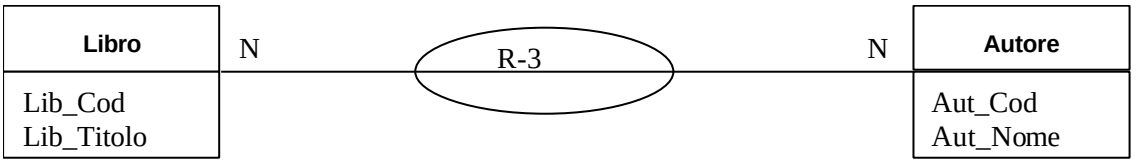

Figura 1.11 Rappresentazione di relazioni N:N

Il modello Entità-Relazione prevede che anche le relazioni possano avere degli attributi che ne specifichino le caratteristiche.

1.11 ESEMPIO DI SCHEMA CONCETTUALE

Nel seguente esempio viene illustrato lo schema concettuale del database di esempio Registro Automobilistico, che verrà utilizzato nei prossimi capitoli.

Il nostro database Registro Automobilistico, che fa ipoteticamente parte del sistema informativo di un ufficio di Motorizzazione, rappresenta la seguente situazione:

- di ciascun veicolo interessa registrare la targa, la cilindrata, i cavalli fiscali, la velocità. il numero di posti, la data di immatricolazione;
- i veicoli sono classificati in *categorie* (automobili, ciclomotori. camion, rimorchi ecc.);
- ciascun veicolo appartiene a uno specifico *modello*;
- tra i dati relativi ai veicoli vi è la codifica del tipo di combustibile utilizzato;

- di ciascun modello di veicolo è registrata la fabbrica di produzione e il numero delle versioni prodotte;
- ciascun veicolo può avere uno o più proprietari, che si succedono nel corso della durata del veicolo; di ciascun proprietario interessa registrare cognome, nome e indirizzo di residenza.

Nel progetto concettuale vengono individuate le seguenti entità:

Entità	Categoria
Categoria	Cod_Categoria Nome_Categoria
Veicolo	Targa Cilindrata Cavalli_Fiscali Velocità Posti Immatricolazione
Modello	Cod_Modello Nome_Modello Numero_Versioni
Fabbrica	Cod_Fabbrica Nome_Fabbrica
Proprietario	Cod_Proprietario Cognome Nome Indirizzo Provincia
Combustibile	Cod_Combustibile Descrizione_Combustibile

Tra tali insiemi di entità sussistono le seguenti relazioni:

- tra Categoria e Veicolo vi è una relazione di tipo 1:N, perché ciascuna categoria è comune a più veicoli, mentre un veicolo può appartenere a una sola categoria;
- Modello e Veicolo sono in relazione 1:N, perché ciascun modello è comune a più veicoli e ciascun veicolo è di un solo modello;
- Fabbrica e Modello sono in relazione 1:N perché una fabbrica può produrre più modelli, mentre un determinato modello viene prodotto da una sola fabbrica.
- Combustibile e Veicolo sono in relazione 1:N, perché un tipo di combustibile è comune a più veicoli, mentre ciascun veicolo utilizza al più un tipo di combustibile;
- Veicolo e Proprietario sono in relazione N:N, in quanto una persona può possedere più veicoli, e più persone possono succedersi nella proprietà di un veicolo nel corso del tempo; per rappresentare tale situazione la relazione ha come attributo la data di acquisto e la data di cessione del veicolo.

1.12 ASTRAZIONE

Del processo di definizione del modello concettuale (a volte indicato con il termine di data modeling) abbiamo dato una breve panoramica al solo fine di introdurre il database di esempio che utilizzeremo nei capitoli successivi e di tracciare un sentiero d'ingresso alla progettazione logica e quindi al modello relazionale che tratteremo nel prossimo capitolo. Questa breve nota risponde alle domande più ricorrenti da parte di chi affronta per la prima volta la definizione di uno schema concettuale di un sistema di una certa dimensione: "Successivamente alle fasi dello studio

preliminare e dell'analisi dei requisiti, che verrà fatta a stretto contatto con gli utenti, come si può realizzare nel miglior modo una rappresentazione della realtà d'interesse? In particolare, come si identificano gli oggetti nella realtà d'interesse?"

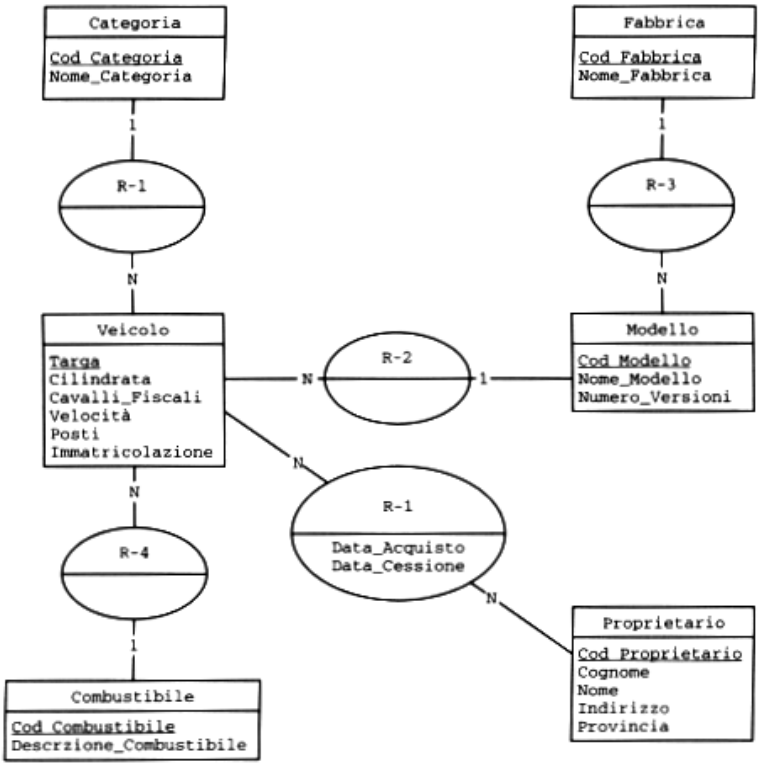

Figura 1.12 Schema concettuale Entità-Relazione del database Registro_automobilistico

Le domande in esame non hanno una risposta né semplice né immediata e neppure, naturalmente, univoca, ma possono essere fatte alcune interessanti riflessioni su come funzionano i meccanismi di astrazione che permettono di ottenere uno schema della realtà.

"L'astrazione è il processo logico per cui. prescindendo dai caratteri particolari e individuali degli oggetti, se ne considerano solo i caratteri generali, comuni a tutta una categoria" (Dizionario Garzanti). Un esempio di astrazione è il procedimento mentale con cui si giunge alla definizione del concetto di *autovettura*, indipendentemente dalle singole parti che la costituiscono, quali ruote, carrozzeria. vetri, volante ecc.; si considera l'insieme di tali componenti come un unico oggetto. Prendiamo in considerazione tre tipi di astrazione: la classificazione, l'aggregazione e la realizzazione.

Con la classificazione viene definita una classe a partire da un insieme di oggetti che possiedono proprietà comuni. Come abbiamo fatto nell'esempio precedente, applicando l'astrazione di classificazione all'insieme delle automobili esistenti nella realtà che percepiamo, si giunge al concetto di autovettura, cui si attribuiscono le proprietà comuni delle autovetture. La classificazione ci porta alla definizione di una classe, con cui si denota un insieme di oggetti della rappresentazione, che vengono detti istanze della classe; per esempio. "Fiat Brava" potrebbe essere un'istanza della classe Autovettura.

Un secondo meccanismo di astrazione è l'aggregazione, che definisce un concetto a partire da altri concetti che insieme lo vengono a costituire. Per esempio, si può giungere alla definizione del concetto di Proprietà mediante un'aggregazione, applicata ai concetti di Veicolo, Proprietario e Data di Acquisto. L'aggregazione consente di definire una nuova classe che è un aggregato di tali concetti.

Un terzo meccanismo di astrazione è la generalizzazione, che definisce un concetto a partire da altri concetti che siano stati definiti mediante l'astrazione di classificazione e aventi caratteristiche comuni. Per esempio, applicando l'astrazione di generalizzazione ai concetti di Docente, Studente, Borsista si può giungere al concetto di Persona. La generalizzazione definisce una nuova classe in cui l'insieme delle istanze contiene l'insieme delle istanze di ogni classe generalizzata. Nell'esempio precedente ciascuna delle classi Docente, Studente, Borsista è un sottoinsieme della classe Persona.

Alla individuazione di una classe si può giungere attraverso differenti processi di astrazione. Alla classe Veicoli possiamo arrivare attraverso l'astrazione di classificazione, individuando proprietà comuni a un insieme di oggetti (per esempio la proprietà di possedere ruote), ovvero le autovetture, i motocicli, i rimorchi e i furgoni presenti nella nostra realtà d'interesse. In alternativa, possiamo giungere alla definizione della classe Veicoli partendo da classi già individuate (Autovettura, Motociclo, Rimorchio e Furgone) e utilizzando l'astrazione di generalizzazione.

Il modello Entità-Relazione consente la rappresentazione di tutte e tre le astrazioni, anche se nel testo abbiamo introdotto solo le prime due: l'entità, che rappresenta un'astrazione di classificazione, e la relazione, che rappresenta un'astrazione di aggregazione. Non abbiamo visto le regole per rappresentare l'astrazione di generalizzazione poiché il modello logico relazionale, non ne permette la diretta rappresentazione. Il modello orientato agli oggetti consente invece di rappresentare tutte e tre le astrazioni.

1.13 PROGETTAZIONE LOGICA

La fase di progettazione logica del database ha lo scopo di tradurre lo schema concettuale espresso mediante un modello semantico in una rappresentazione mediante un modello logico dei dati. Tale rappresentazione viene definita schema logico del database.

Un modello logico dei dati è la tecnica di organizzazione e accesso ai dati utilizzata da specifiche categorie di DBMS. A differenza dello schema concettuale, lo schema logico dipende quindi strettamente dal tipo di DBMS utilizzato, e in particolare dal suo modello logico dei dati. I DBMS, in riferimento al modello logico dei dati su cui si basano, vengono distinti in gerarchici, reticolari e relazionali. Di questi ultimi ci occuperemo a partire dal prossimo capitolo, in cui definiremo lo schema logico del database Registro_Automobilistico.

Un ulteriore compito della progettazione logica è di individuare all'interno dello schema logico del database le parti che sono rilevanti per le singole applicazioni; queste parti vengono definite viste o anche schemi esterni o sottoschemi.

1.14 PROGETTAZIONE FISICA

Nella progettazione fisica viene stabilito come le strutture definite a livello logico devono essere organizzate negli archivi e nelle strutture del file system. Essa dipende quindi non solo dal tipo di DBMS utilizzato, ma anche dal sistema operativo e in ultima istanza dalla piattaforma hardware del sistema che ospita il DBMS. È pertanto il livello di progettazione in cui si può far uso del minor livello di astrazione, dovendo rispettare i vincoli tecnici imposti dal sistema ospite.

1.15 MODALITÀ D'USO DEI DBMS

Nell'utilizzo di un DBMS vengono tradizionalmente distinti tre tipi di linguaggi in base alle funzioni eseguite sui dati:

- DDL (Data Description Language), per la definizione dello schema logico del database;

- DML (Data Manipulation Language). per le operazioni di interrogazione e di aggiornamento dei dati, quali inserimento. modifica, cancellazione ecc. Può essere disponibile come linguaggio a se stante o come un insieme di istruzioni richiamabili da un linguaggio di programmazione esistente;
- DCL (Data Control Language), per operazioni di controllo dei dati, gestione degli utenti, assegnazione dei diritti di accesso, ottimizzazione del funzionamento del DBMS.

Si possono individuare tre classi di utenza di un DBMS.

- Utenti finali (end-user). Utilizzano generalmente i programmi applicativi sviluppati appositamente per automatizzare specifiche operazioni e vedono il database esclusivamente attraverso il filtro di tali applicazioni. La maggior parte dei DBMS dispongono, come parte integrante o come estensione esterna del DML, di linguaggi interattivi (query language) o anche di ambienti integrati per l'interrogazione e l'aggiornamento dei dati, che permettono anche a utenti non esperti di effettuare tali operazioni senza dover ricorrere alla realizzazione di appositi programmi applicativi.
- Programmatori. Realizzano i programmi applicativi destinati agli utenti finali, utilizzando il DML del DBMS.
- Amministratore del database o DBA (Data Base Administrator). È la persona o il gruppo di persone che, avendo in genere partecipato alla progettazione del database, mantiene in esercizio il DBMS. E' necessario pertanto che possieda le maggiori competenze circa il funzionamento del DBMS e la conoscenza globale riguardo alla struttura e al contenuto del database e al suo utilizzo da parte delle varie categorie di utenza. Si servirà in particolare del DDL per la definizione della struttura del database e del DCL per le operazioni di controllo dei dati. A volte, in particolare nelle grandi organizzazioni, si fa distinzione tra l'amministratore della base dati nel suo complesso e l'amministratore di uno o più DBMS. Il primo viene anche definito DA (Data Administrator) ed è una figura diversa rispetto al DBA in quanto lavora a un livello più alto, occupandosi dei dati come patrimonio del sistema informativo aziendale, indipendentemente dalla loro localizzazione all'interno di un DBMS.

1.16 DBMS NON RELAZIONALI

I primi DBMS commerciali, il capostipite dei quali è l'IMS sviluppato dalla IBM nel 1969, utilizzavano un modello dei dati che viene definito gerarchico, poiché lo schema del database era vincolato a essere costituito da strutture ad albero. Tale modello dei dati non era stato sottoposto ad alcuna forma di standardizzazione: quello dell'IMS era stato adottato come base comune da diverse case produttrici, ma le evoluzioni che ne seguirono erano anche molto diverse. Il modello gerarchico. Pur rappresentando bene alcune specifiche realtà, non si adatta a tutti i possibili contesti: esso consente infatti una efficace rappresentazione.delle relazioni 1:N, ma non delle relazioni N:N. Per accedere ai dati posti a un certo livello dell'albero si deve iniziare la ricerca dalla radice procedendo verso il basso e utilizzando per la navigazione i puntatori unidirezionali.

Intorno ai primi degli anni Settanta fu elaborato il modello reticolare, che segna il superamento della strutturazione necessariamente gerarchica dello schema logico. Questo modello nasce da una proposta per la standardizzazione della definizione e l'utilizzo dei database, elaborata nel 1971 dal gruppo di lavoro DBTG (Data Base Task Group) del comitato CODASYL. Sebbene molti sistemi (IDS/II della Honeywell, DMS-1100 dell'Univac, DBMS- 10 della Digital) abbiano adottato gli elementi fondamentali dello standard, problemi di tipo commerciale e di tipo tecnologico, quali la complessità del linguaggio di accesso ai dati e i requisiti hardware, hanno fatto sì che non sia stata sviluppata nessuna implementazione rispondente totalmente alle regole DBTG-CODASYL. Tuttavia la proposta ha avuto il ruolo fondamentale, se non di standardizzare totalmente un modello di dati, almeno di introdurre alcuni concetti basilari della teoria dei database. quali quelli di schema e sottoschema, di indipendenza dei dati dalle applicazioni, di DDL e DML. Tale modello è

un’estensione del precedente modello gerarchico, e permette di rappresentare tutte le connessioni possibili, anche quelle N:N, ancora una volta attraverso puntatori.

Negli stessi anni in cui veniva sviluppato il modello reticolare iniziò la definizione del modello relazionale; ci vollero tuttavia quasi altri dieci anni perché tale modello potesse essere utilizzato per l’implementazione di database commerciali.

Nell’ambito dei DBMS successivi al modello relazionale, particolare rilievo hanno i DBMS a oggetti (object oriented), che estendono ai database un modello dei dati sviluppato nell’ambito dei linguaggi di programmazione (Smalltalk, Simula, C++, Java), e i DBMS object relational.

2 Database relazionali

2.1 RDBMS

Storicamente i DBMS relazionali (RDBMS, Relational Database Management System) furono i primi a essere basati su un modello dei dati precedentemente formalizzato. Tale modello venne sviluppato nel 1970 da Edward Codd presso l'IBM San Josè Research Laboratory. L'approccio adottato è basato sulla teoria matematica delle relazioni tra insiemi. Esso prescinde da specifiche tecniche realizzative e da considerazioni relative alla facilità di realizzazione e all'efficienza delle prestazioni.

Le prime implementazioni - Oracle presso la Relational Software Inc., poi Oracle Corporation; System R e successivamente SQL/DS e DB2 presso la IBM; Ingres presso l'Università di Berkeley - risalgono agli anni compresi tra il 1977 e il 1983. L'indipendenza del modello da specifiche tecnologie informatiche, se da una parte ne ha frenato inizialmente l'applicazione, dall'altra ha costituito il suo punto di forza e ne ha permesso successivamente un'enorme diffusione su tutte le piattaforme e architetture.

L'uso di un modello matematico dei dati consente l'applicazione di linguaggi e metodologie formali per l'accesso ai dati. In particolare, le due metodologie su cui si basano i linguaggi per l'accesso ai dati di un database relazionale sono l'algebra relazionale e il calcolo relazionale. Nel capitolo successivo verrà introdotta la prima metodologia, dal momento che essa costituisce la base del linguaggio SQL.

2.2 IL MODELLO RELAZIONALE

La rappresentazione dei dati nel modello logico relazionale è basata sul concetto di relazione; questa va intesa in termini algebrici, e non va confusa con le relazioni tra i dati del modello concettuale. Il concetto di relazione algebrica è così definito:

- Dati gli insiemi finiti di valori D1, D2,..., Dn, una relazione K tra questi n insiemi è un insieme di n-uple <d1, d2, ..., dn>, dove d1 appartiene a D1, d2 appartiene a D2, ..dn appartiene a Dn;
- D1, D2, ... Dn sono detti domini, i loro nomi sono detti attributi, n è detto grado della relazione R.

Considerati i due domini

CODICE CATEGORIA={01, 02, 03, 04}

e

NOME CATEGORIA={Autovettura, Furgone, Motociclo, Rimorchio }

una possibile relazione tra essi è

CATEGORIE=<(01,Autovettura), (02,Rimorchio), (03,Motociclo), (04, Furgone)>

Per comodità di rappresentazione è conveniente descrivere una relazione algebrica in forma di tabella, come in Figura 2.1.

In base a questa rappresentazione, spesso i concetti di relazioni algebriche, di n-uple e di attributi vengono indicati con i termini più familiari di tabelle, di righe e di colonne.

Nel seguito del testo il termine "tabella" sostituirà quello di "relazione algebrica", mentre il termine "relazione" indicherà, come nel modello Entità-Relazione, un'associazione tra i dati. Dalla definizione di tabella come insieme derivano due conseguenze fondamentali:

- in una tabella non possono esistere due righe uguali;
- l'ordine tra le righe di una tabella non è significativo.

Dalla prima osservazione consegue che è possibile, e anche necessario, individuare in ciascuna tabella un insieme di attributi (colonne) in base ai quali

Targa	Cod_ Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_ Fiscali	Velocità	Posti	Immatricolazione
A123456X	004	01	1796	01	85	195	5	30/12/98
B256787Y	001	01	708	02	10	120	5	21/09/89
C76589AG	001	01	1106	01	54	130	5	13/08/98
C78905GT	009	01	1998	01	117	212	4	06/11/94
C845905Z	006	04		03			3	11/04/95
CFT346VB	007	01	1390	02	75	170	5	12/01/95
D238765W	002	03		01			2	12/08/97
DD4567XX	003	01	1581	01	17		5	05/06/97
DGH789JC	014	01	1590	02	114	170	5	05/10/95
DH79567H	011	01	1589	04	107	170	0	
ERG567NM	013	01	1598	04	107	175	5	18/12/97
F96154NH	010	01	1781	03	125	185	5	08/03/92
FGH673FV	012	01	899	01	39	140	5	09/08/98
XCH56GJK	005	01	1918	01	110	210	5	04/09/98
XF5789GY	008	01	1587	01	107	175	5	05/05/96

Figura 2.1 - Rappresentazione tabellare di una relazione

vengono identificate le singole righe, che rappresentano quindi una chiave di accesso univoca alle informazioni contenute nella tabella stessa. Questo insieme di colonne, che va definito in fase di creazione dello schema logico, è detto chiave primaria (primary key) della tabella. Come vedremo nel seguito, tale requisito, presente nel modello, come pure quello dell'assenza di righe uguali in una tabella, non è invece presente nel linguaggio SQL, la cui sintassi non vincola alla definizione di chiavi primarie, e che può restituire come risultato di un'interrogazione una tabella con due righe uguali; per quanto concerne il primo punto, tuttavia, è sempre raccomandabile, oltre che per correttezza di forma anche per ottimizzare le prestazioni del sistema, specificare nella creazione dello schema logico una chiave primaria per ciascuna tabella.

Per la creazione di uno schema logico relazionale, partendo da uno schema concettuale definito in base al modello Entità-Relazione, è necessario applicare le regole seguenti.

1. Le entità (entity set) dello schema concettuale divengono tabelle nello schema logico.
2. Le relazioni (relation set) dello schema concettuale vengono rappresentate nello schema logico facendo uso delle cosiddette chiavi esterne. Una chiave esterna (foreign key) di una tabella è un insieme di attributi che corrispondono a quelli che costituiscono la chiave primaria di un'altra tabella, e stabiliscono quindi un riferimento tra le righe delle due tabelle.

Nella rappresentazione di una relazione tra le tabelle T1 e T2 bisogna distinguere tra le cardinalità 1: 1, 1:N, N:N, individuate nella definizione dello schema concettuale.

2.2.1 Relazione 1:1

Agli attributi di T1 vanno aggiunti, come chiave esterna, gli attributi che costituiscono la chiave primaria di T2, o alternativamente a T2 vanno aggiunti, come chiave esterna, gli attributi che costituiscono la chiave primaria di T1. Le due soluzioni sono del tutto equivalenti. Per esempio, considerate le tabelle

Nazioni

Cod_Nazione	Nome_Nazione
001	Austria
002	Francia
003	Italia

Capitali

Cod_Capitale	Nome_Capitale
001	Parigi
002	Roma
003	Vienna

la relazione tra esse viene rappresentata estendendo la tabella Nazioni nel modo seguente:

Nazioni

Cod_Nazione	Nome_Nazione	Cod_Capitale
001	Austria	003
002	Francia	001
003	Italia	002

oppure in alternativa estendendo la tabella Capitali nel modo seguente:

Capitali

Cod_Capitale	Nome_Capitale	Cod_Nazione
001	Parigi	002
002	Roma	003
003	Vienna	001

2.2.2 Relazione 1:N

Agli attributi di T2 vanno aggiunti, come chiave esterna, gli attributi che costituiscono la chiave primaria di T1 (il viceversa in questo caso non è equivalente). Per esempio, considerate le tabelle

Ordini

N_Ordine	Data	Cliente
1	12/01/00	Bianchi S.p.A.
2	23/08/00	Rossi S.r.l.
3	05/09/00	Verdi S.n.c.

Righe_Ordine

N_Riga	Articolo	Quantità
1	Tavolo	2
2	Sedia	10
1	Armadio	1
2	Sedia	20

la relazione tra esse viene rappresentata estendendo la tabella Righe_Ordine nel modo seguente:

Righe_Ordine

N_Ordine	N_Riga	Articolo	Quantità
1	1	Tavolo	2
1	2	Sedia	10
2	1	Armadio	1
2	2	Sedia	20

2.2.3 Relazione N:N

In questo caso va definita una nuova tabella T3 che contiene, come chiavi esterne, le chiavi primarie sia di T1 sia di T2; è da notare come in questo caso la chiave primaria della tabella T3 possa essere costituita dalla totalità dei suoi attributi. Per esempio, considerate le tabelle

Libri

Codice_Libro	Titolo	Collocazione
1	LE RETI DI COMPUTER	1/C/2
2	IL MANUALE DI SQL	3/H/8
3	TECNICHE DI PROGRAMMAZIONE	3/P/2
4	C++ MANUALE DI RIFERIMENTO	4/H/9

Autori

Codice_Autore	Cognome	Nome
1	PAOLO	REDI
2	GIOVANNI	MARIOTTI
3	MARIO	BIANCHI

la relazione tra esse viene rappresentata nel modo seguente:

Libri-Autori

Codice_Libro	Codice_Autore
1	3
2	1
3	2
4	1

Come previsto dal modello Entità-Relazione, anche una relazione può possedere degli attributi. Gli eventuali attributi della relazione vengono inclusi come attributi della tabella in cui è rappresentata la relazione. ovvero quella che contiene le chiavi esterne.

Renderemo come base per la costruzione dello schema logico relazionale del database Registro_Automobilistico lo schema concettuale sviluppato nel capitolo precedente. I passi per tradurre lo schema concettuale nello schema logico corrispondente sono i seguenti.

1. Per ciascuna entità dello schema concettuale viene definita una tabella nello schema logico: Veicoli, Categorie, Combustibili, Modelli, Fabbriche e Proprietari (Figura2.2). I nomi delle entità nello schema concettuale sono espressi al singolare (Veicolo, Categoria ecc.); nello schema logico, seguendo una convenzione spesso adottata, sono resi plurali (veicoli, Categorie ecc.). Sarebbe stato possibile cambiare completamente i nomi o lasciare gli stessi; l'importante è avere come obiettivo la comprensibilità dello schema.
2. Per ciascuna tabella viene definito un insieme di attributi con funzione di una chiave primaria che ne identifichi univocamente le righe. Se tale insieme non è individuabile tra gli attributi della tabella, verrà aggiunto un nuovo attributo finalizzato a questo scopo. Nel caso in cui gli attributi della chiave primaria facciano parte delle informazioni presenti nella tabella, la chiave primaria è detta naturale; nel caso in cui venga definito un attributo ad hoc, la chiave primaria e detta artificiale:

Veicoli		**Modelli**	
Targa:	testo(10)	Cod_Modello:	testo(3)
Cilindrata:	numerico	Nome_Modello:	testo(30)
Cavalli_Fiscali:	numerico	Numero_Versioni:	numerico
Velocità:	numerico		
Posti:	numerico	**Fabbriche**	
Immatricolazione:	data	Cod_Fabbrica:	testo(3)
		Nome_Fabbrica:	testo(30)
Categorie			
Cod_Categoria:	testo(2)	**Proprietari**	
Nome_Categoria:	testo(30)	Cod_Proprietario:	testo(5)
		Cognome:	testo(30)
Combustibili		Nome:	testo(30)
Cod_Combustibile:	testo(2)	Indirizzo:	testo(30)
Descrizione_Combustibile:	testo(30)	Provincia:	testo(2)

Figura 2.2 Le tabelle corrispondenti agli insiemi di entità del database Registro_Automobilistico.

Tabella	Chiave primaria
Veicoli	Targa
Categorie	Cod_Categoria
Combustibili	Cod_Combustibile
Modelli	Cod_Modello
Fabbriche	Cod_Fabbrica
Proprietari	Cod_Proprietario

1. Vengono definite le chiavi esterne per la rappresentazione delle relazioni 1:N tra Categorie e Veicoli, tra Combustibili e veicoli, tra Modelli e Veicoli e tra Fabbriche e Modelli (Figura 2.3).
2. Viene definita la nuova tabella Proprietà per la rappresentazione della relazione N:N tra Veicoli e Proprietari (Figura 2.4). Questa contiene, come chiave primaria, le chiavi primarie di Veicoli e Proprietari, e gli attributi Data Acquisto e Data Cessione, che costituiscono gli attributi della relazione.

Veicoli		Modelli	
Targa	Testo(10)	Cod_Modello	Testo(3)
Cod_Categoria	Testo(2)	Nome_Modello	Testo(30)
Cilindrata	numerico	Cod_Fabbrica	Testo(3)
Cod_Combustibile	Testo(2)	Numero_versioni	numerico
Cavalli_Fiscali	numerico		
Velocità	numerico		
Posti	numerico		
Immatricolazione	data		
Cod_Modello	Testo(3)		

Figura. 2.3 Tabelle del database Registro_Automobilistico con le chiavi esterne per le relazioni 1 :N

Proprietà	
Targa: testo(10)	
Cod_Proprietario	testo(5)
Data_Acquisto	data
Data_Cessione	data

Tabella	Chiave primaria
Proprietà	Targa+Cod_Proprietario

Figura 2.4 Rappresentazione delle relazioni N:N nel database Registro_Automobilistico

In Figura 2.5 viene presentata una rappresentazione dello schema logico relazionale del database di esempio Registro_Automobilistico, e in Appendice A è riportato il contenuto dello stesso database, che verrà utilizzato negli esempi del testo.

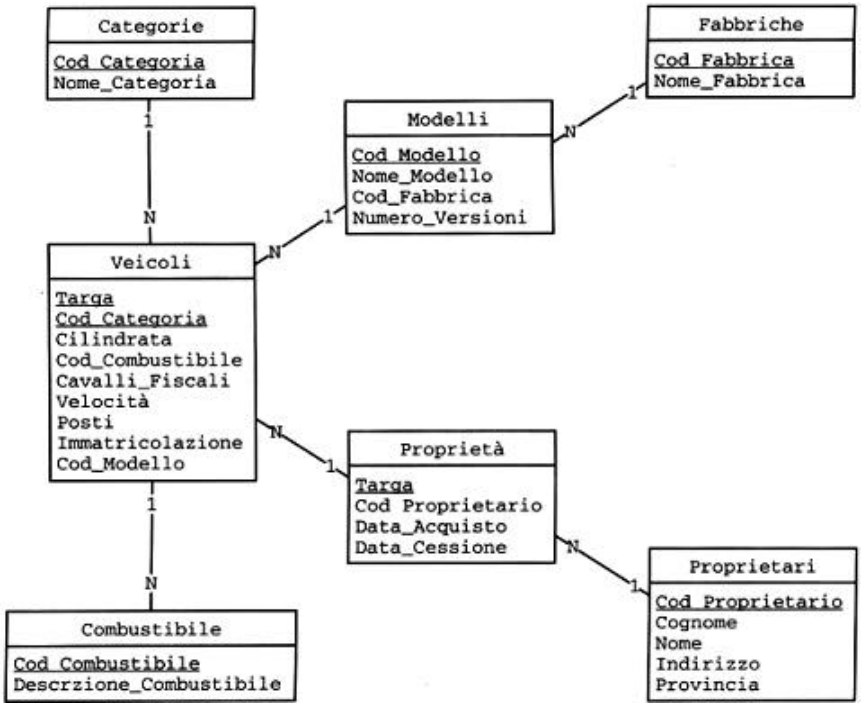

Figura 2.5 Rappresentazione dello schema logico relazionale di Registro_Automobilistico

2.3 NORMALIZZAZIONE

Una volta impostato uno schema logico relazionale è necessario effettuare una serie di verifiche sulla correttezza del procedimento svolto. Queste potranno portare a modificare la struttura dello schema stesso, al fine di renderlo corretto ed evitare il verificarsi, nella gestione dei dati, di errori difficilmente ovviabili a posteriori. Tale processo è detto normalizzazione dello schema relazionale ed è effettuabile mediante procedimenti di tipo algebrico, basati sul concetto di dipendenza funzionale; senza voler entrare in questa complessa componente della teoria relazionale, introdotta da Codd insieme al modello dei dati, ne esamineremo in maniera informale le regole principali.

2.4 PRIMA FORMA NORMALE

La prima forma normale stabilisce che in una tabella non possono esistere colonne definite per contenere una molteplicità di valori. Una tabella quindi non può contenere una struttura vettoriale o array, al contrario di quanto consentito in linguaggi di programmazione come il Pascal, il C o il Cobol, in alcuni DBMS prerelazionali e in quelli orientati agli oggetti.

Le tabelle che contengano una colonna non rispondente a questa condizione vanno trasformate, creando per ciascuna riga della tabella di partenza tante righe quanti sono i valori multipli presenti nella colonna considerata (Figura 2.6).

2.5 SECONDA FORMA NORMALE

La seconda forma normale riguarda le tabelle in cui la chiave primaria sia composta da più attributi e stabilisce che, in questo caso, tutte le colonne corrispondenti agli altri attributi dipendano dall'intera chiave primaria.

Nella prima parte della Figura 2.7 è mostrata una tabella che non risponde a questo requisito: infatti la chiave primaria è composta da Codice Città e Codice_via; la colonna Città dipende solo da Codice_Città; la colonna Via dipende invece da Codice_Città e Codice_Via; la colonna Città quindi non dipende da tutta la chiave. ma solo da una sua parte.

Per ricondurre una tabella con queste caratteristiche alla seconda forma normale è necessario introdurre una nuova tabella. Nella seconda parte della Figura 2.7 entrambe le tabelle, che rappresentano globalmente la stessa realtà della precedente, rispettano la seconda forma normale

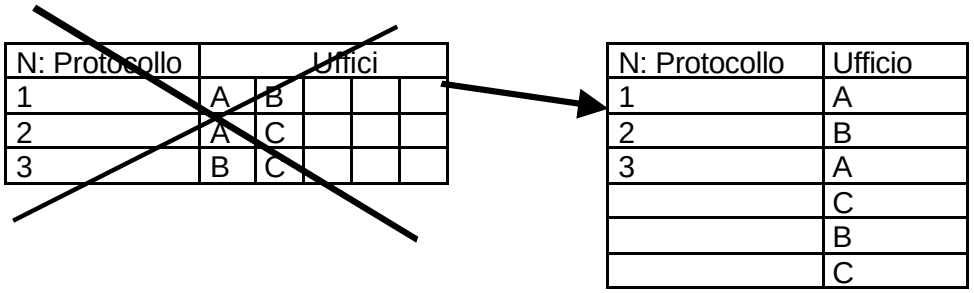

Figura 2.6 Trasformazione in prima forma normale

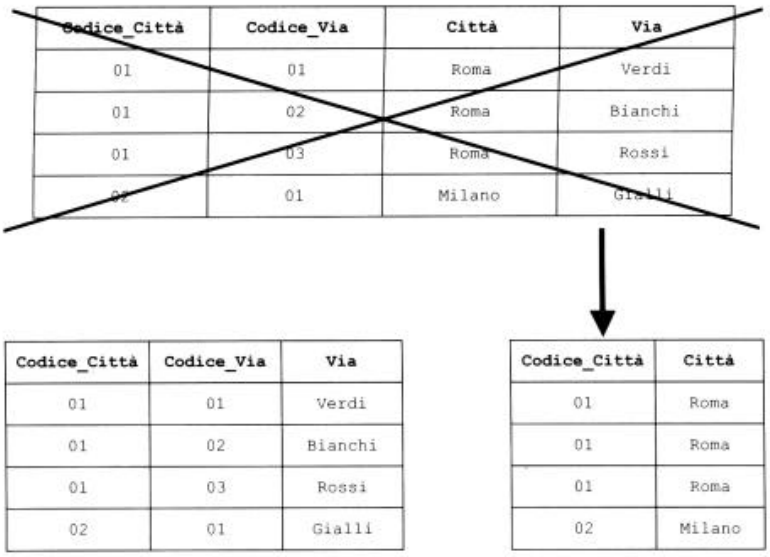

Figura 2.7 Trasformazione in seconda forma normale

2.6 TERZA FORMA NORMALE

La terza forma normale stabilisce che non esistano dipendenze tra le colonne di una tabella se non basate sulla chiave primaria.

Nella prima parte della Figura 2.8 sono mostrate due tabelle che non rispondono a questo requisito; infatti nella prima tabella la chiave primaria è composta da Protocollo; la colonna Urgenza non dipende direttamente dalla colonna Protocollo, bensì dalla chiave primaria della seconda tabella, ovvero dalla colonna Tipo.

Anche in questo caso la dipendenza va ricondotta alla opportuna tabella, eventualmente creandone una ad hoc. Nel nostro esempio le modifiche da apportare allo schema logico rappresentato sono le seguenti:

1. eliminare la colonna Urgenza dalla prima tabella;
2. aggiungere la colonna Urgenza alla seconda tabella, ottenendo così la terza tabella della Figura 2.8.

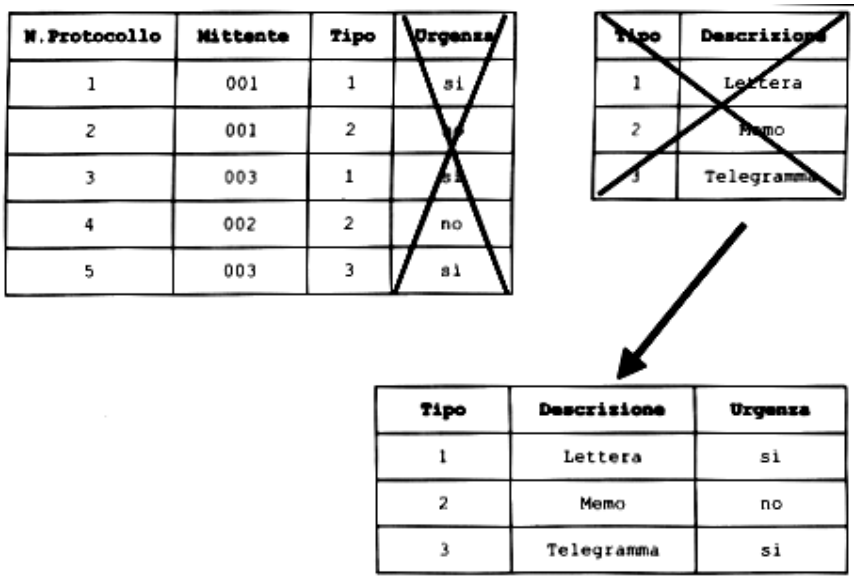

Figura 2.8 Trasformazione in terza forma normale

Ricapitolando, il corretto progetto di una base di dati relazionale dovrebbe partire dalla definizione dello schema concettuale derivato dall'esame della realtà d'interesse, per arrivare alla definizione di uno schema logico relazionale normalizzato.

Esistono strumenti informatici detti di CASE (Computer Aided Software Engineering) che aiutano l'analista in questo processo; esempi di questa classe di strumenti sono Silverrun, ER-Win, PowerDesigner, ER-Studio; gli schemi rappresentati in questo testo sono stati prodotti utilizzando i moduli ERX (Entità Relationship eXpert) e RDM (Relational Data Modeler) di Silverrun.

Lo schema relazionale deve quindi essere espresso mediante i comandi del DDL dell'RDBMS utilizzato; tra i più diffusi sul mercato troviamo Sybase Adaptive Server e Anywhere, Microsoft SQL Server, Oracle, Informix, Ingres e DB/2 dell'IBM; tutti supportano il linguaggio SQL. Molti strumenti CASE permettono di creare e modificare automaticamente le strutture dati, generando l'SQL necessario, partendo dallo schema logico (direct engineering) e viceversa di creare o modificare lo schema logico partendo dalle strutture dati esistenti sul DBMS (riverse engineering).

In pratica, nella progettazione dello schema logico, a volte si compie per ragioni di efficienza un ulteriore passo. detto di denormalizzazione, in cui, in parziale contrasto con la teoria relazionale, si ammette una certa ridondanza dei dati in cambio di migliori prestazioni del sistema in termini di tempi di risposta.

2.7 EVOLUZIONE DELL'SQL

Successivamente alla definizione di Codd del modello relazionale vennero sviluppati diversi prototipi di linguaggi relazionali. ovvero di linguaggi che mettessero in atto nella loro sintassi i metodi definiti nel modello teorico.

Nel 1974 D.D. Chamberlain, anch'egli presso l'IBM San Josè Research Laboratory, elaborò il linguaggio SEQUEL (Structured English Query Language). Negli anni successivi venne definita la versione SEQUEL/2, ridenominata poi SQL, su cui la IBM basò il progetto di ricerca che partì) alla realizzazione del database relazionale System R, sperimentato sia presso la IBM stessa sia presso clienti che collaboravano al progetto. Nel corso dell'attività di test vennero apportate diverse modifiche al linguaggio, anche su suggerimento degli utenti del nuovo sistema; venne per esempio introdotto il predicato EXISTS per la verifica della presenza di dati rispondenti a specifiche condizioni. Il successo del System R condusse successivamente allo sviluppo dei prodotti commerciali SQL/DS (1981) e DB2 (1983).

Negli stessi anni ebbe inizio l'attività di standardizzazione del linguaggio, che lo vide passare da standard de facto a standard ufficiale: nel 1982 l'ANSI (American National Standard Institute) definì, adottando diverse proposte della IBM, un linguaggio per database relazionali denominato RDL (Relational Database Language).

Le caratteristiche dell'RDL vennero accolte dall'ISO (International Standard Organization) nella prima specifica dello standard SQL, pubblicata nel 1987 (ISO 9075:1987).

Dalla prima versione a oggi si sono succedute diverse revisioni da parte dell'ISO e dell'ANSI; alla revisione conosciuta come SQL89 è seguita nel 1992 SQL92, l'ultima versione universalmente accettata. denominata ISO 9075:1992, su cui essenzialmente si basa questo testo. Successivamente si è lavorato molto all'evoluzione del linguaggio, cui ci si riferisce spesso con il termine SQL3.

2.8 CARATTERISTICHE DEL LINGUAGGIO

Sebbene la sua denominazione, acronimo di Structured Query Language, che per ragioni storiche è stata mantenuta in tutte le versioni, faccia pensare esclusivamente a un linguaggio di interrogazione, l'SQL, è un linguaggio unificato che consente l'esecuzione di tutte le operazioni necessarie alla gestione e all'utilizzo di un database. I suoi costrutti infatti consentono di effettuare, oltre alla consultazione del database, anche la definizione e la gestione dello schema, la manipolazione dei dati e l'esecuzione di operazioni di amministrazione. Può, quindi essere utilizzato da tutte le categorie di utenza del database, dall'utente finale all'amministratore. L'SQL, tuttavia, non è un linguaggio di programmazione e non contiene pertanto i meccanismi di controllo tipici dei linguaggi procedurali quali IF ... THEN ... ELSE, WHILE, FOR ecc. Per poterlo utilizzare sono quindi state definite delle estensioni dei principali linguaggi di programmazione, che consentono di includere nei programmi comandi SQL; in questo caso l'uso dell'SQL viene definito embedded o incorporato, e il linguaggio di programmazione viene detto linguaggio ospite.

L'SQL può anche essere utilizzato direttamente dall'utente, tramite appositi ambienti per l'esecuzione che ne interpretano i comandi, che vengono forniti dal produttore del database relazionale utilizzato o da terze parti; in questo caso l'uso di SQL viene definito interattivo.

Ci sono poi i linguaggi detti RAD (Rapid Application Development, sviluppo rapido di applicazioni) che gestiscono in modo estremamente flessibile l'interfaccia verso il mondo SQL. Per esempio, con una sola istruzione permettono di mandare in esecuzione un'interrogazione SQL al database, catturare i dati restituiti e visualizzare il risultato in una lista a video dalla dimensione, posizione e caratteristiche volute.

La principale caratteristica dell'SQL per quanto concerne le modalità di accesso ai dati è di essere un linguaggio dichiarativo o non navigazionale. La maggior parte dei linguaggi di programmazione

che operano su archivi di dati sono di tipo procedurale o navigazionale: i dati costituiscono una sequenza ordinata, e il linguaggio permette di lavorare solo su singoli record: per operare quindi sui record che rispondono a specifiche condizioni è necessario posizionarsi tramite un file pointer su un record dell'archivio e verificare su tale singolo record la validità della condizione, quindi spostarsi sul record successivo. e così via.

In SQL, invece, non è necessario specificare al sistema alcun algoritmo procedurale da eseguire per individuare i record che rispondono a una condizione: è sufficiente dichiarare nel comando di selezione la condizione desiderata. Normalmente sI lavora quindi non su singole entità, ma su insiemi di entità che rispondono a determinate condizioni.

Per realizzare la compatibilità, nel caso di uso embedded, tra il linguaggio ospite (navigazionale) e l'SQL (non navigazionale), in quest'ultimo è stata introdotta la categoria sintattica dei cursori.

La critica che veniva rivolta ai primi database relazionali che facevano uso dell'SQL era di offrire scarse prestazioni nell'accesso ai dati, non essendo questo controllato, come avveniva con i linguaggi procedurali, direttamente dal programmatore, che può utilizzare informazioni sugli specifici dati per migliorare la velocità di ricerca.

I moderni sistemi relazionali sono quindi stati dotati di moduli di ottimizzazione che individuano in base alla richiesta il percorso di selezione dei dati che hanno maggiori probabilità di risultare il più efficiente.

2.9 IMPORTANZA DELLO STANDARD

Ci si potrebbe chiedere il perché dell'esigenza di uno standard come linguaggio d'interfaccia verso i DBMS. I motivi sono diversi e probabilmente evidenti. Come rileva C.J. Date in A Guide to the SQL Standard.

In primo luogo il tempo. Gli standard rimangono stabili per un arco temporale ragionevole, la loro variazione prevede un lungo periodo di studio, gestazione, formalizzazione e approvazione da parte di appositi comitati che sottostà al raggiungimento di un accordo tra numerosi soggetti. Di conseguenza, a loro volta le applicazioni basate sullo standard avranno assicurato un periodo di vita ragionevole.

In secondo luogo la plurivalenza. Gli utenti, gli sviluppatori di applicazioni e i DBA possono passare da un DBMS ad un altro senza la necessità di un lungo periodo di formazione e di apprendistato. Le applicazioni possono essere realizzate per un ambiente e poi fatte girare in un altro senza necessità di modifiche.

Infine la cooperazione. DBMS di produttori differenti possono lavorare insieme in un singolo ambiente distribuito. Infatti è sempre meno frequente incontrare grandi organizzazioni che abbiano adottato un solo DBMS, e sono invece sempre più diffuse realtà aziendali in cui più database, che costituiscono nel loro insieme il serbatoio del sistema informativo, devono lavorare in parallelo interagendo continuamente. Questa riflessione può naturalmente essere estesa a database di organizzazioni diverse, dal momento che l'evoluzione tecnologica delle reti di telecomunicazione sta portando a una sempre crescente interrelazione tra sistemi informativi autonomi.

Gli standard, in generale, portano inevitabilmente alcuni svantaggi; per esempio impongono delle strade obbligatorie, che non sempre sono le migliori per i singoli casi specifici. Inoltre, lo standard SQL non rispecchia appieno il modello relazionale e perciò non ne utilizza tutta la potenza. Come riporta Date nell'articolo "A critique of the SQL Database Language" inserito nella raccolta Relational Database: Selected Writings, malgrado al momento della sua definizione fossero da tempo noti i principi basilari per la progettazione di un linguaggio formale, l'SQL risulta solo in parte progettato secondo tali principi. Del resto è necessario ricordare che il linguaggio. nella sua stesura primaria - come spesso accade -, è nato sul campo e solo successivamente sottoposto a un processo di formalizzazione e standardizzazione.

Il fatto più rilevante è però che l'SQL standard è incompleto e non è provvisto di alcune funzionalità che nella pratica sono necessarie, per cui ogni implementazione commerciale contiene estensioni e variazioni che fanno sì che non esistano due implementazioni SQL identiche. Nel presente testo si tiene conto di questo fatto e. pur trattando l'SQL standard in tutta la sua area di competenza, si fa riferimento quando necessario ad alcuni suoi importanti dialetti.

2.10 FORMATO DEI COMANDI

Per descrivere il formato dei comandi verranno utilizzati i seguenti simboli. derivati dalla notazione sintattica BNF (Backus-Naur Form):

< > delimitano i nomi degli elementi del linguaggio; per esempio: <comando>, <espressione>, <identificatore>

[ ] denotano un elemento opzionale

... indicano che l'elemento precedente può essere ripetuto

{ } raggruppano più elementi di una definizione; per esempio, se sono seguite da ... indicano che la ripetizione è relativa a tutta la sequenza degli elementi contenuti nelle parentesi graffe

| indica che l'elemento che lo segue e quello che lo precede sono in alternativa tra loro.

Le parole chiave verranno scritte per maggiore chiarezza in caratteri maiuscoli.

Un comando SQL è normalmente costituito da una operazione seguita da una o più clausole che ne specificano nel dettaglio l'effetto. La definizione generale di comando risulta quindi:

<comando> ::= <operazione><clausola> [<clausola>]

Esempi:

SELECT Targa	*? operazione*
FROM Veicoli	*? clausola*
WHERE Velocità >150;	*? clausola*
UPDATE Veicoli	*? operazione*
SET Peso = 0	*? clausola*

3 Algebra relazionale

3.1 GLI OPERATORI DELL'ALGEBRA RELAZIONALE

L'algebra relazionale definisce gli operatori rilevanti nella selezione di dati rappresentati mediante il modello relazionale. La maggior parte dei comandi di interrogazione definiti dall'SQL costituisce la realizzazione di operazioni definite nell'algebra relazionale. Quest'ultima rappresenta pertanto il fondamento formale del linguaggio.

Gli operatori dell'algebra relazionale possono essere classificati in operatori di ***base e*** operatori derivati. 1 primi permettono di realizzare tutte le operazioni principali di selezione di dati all'interno di uno schema relazionale. i secondi, presenti in molti sistemi relazionali, sono derivabili dai primi mediante opportune operazioni algebriche. Gli operatori di base sono: proiezione, selezione, prodotto, ridenominazione, unione e differenza. Gli operatori derivati sono: giunzione, giunzione naturale, semigiunzione, giunzione esterna, intersezione e divisione. Molti sistemi relazionali forniscono dei costrutti aggiuntivi che realizzano operazioni che non fanno parte dell'algebra relazionale, e che in generale non sono derivabili dagli operatori di base. In particolare, sono spesso utilizzati il complemento, le funzioni di aggregazione e la chiusura transitiva.

Tutti gli operatori relazionali hanno la caratteristica comune di avere come argomenti delle relazioni (tabelle) e fornire come risultato altre relazioni (ancora tabelle).

Nel seguito useremo come sinonimi i termini "relazione" e "tabella"..

3.2 OPERATORI DI BASE

Proiezione. Data una tabella e un insieme di attributi, la proiezione restituisce una tabella con tutte le righe di quella di partenza, ma con le sole colonne identificate dagli attributi. Per esempio, considerata la tabella:

A1	A2	A3
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

e l'insieme di attributi { Al , A2 } , la proiezione dà come risultato:

A1	A2
a11	a12
a21	a22
a31	a32
a41	a42

Selezione o ***restrizione.*** Data una tabella e una condizione logica definita in base ai valori dei suoi attributi, la selezione restituisce una tabella con gli stessi attributi di quella di partenza, ma con le sole righe che soddisfano la condizione. Per esempio, data la tabella:

B1	B2	B3
B11	15	B13
b21	5	b23
b31	6	b33

b41	20	B43

e la condizione B2 > **10,** la selezione dà come risultato:

B1	**B2**	**B3**
B11	15	B13
b41	20	B43

Prodotto o congiunzione. Date due tabelle, il loro prodotto restituisce una tabella le cui righe sono ottenute concatenando ogni riga della prima con tutte le righe della seconda. Per esempio, date le due tabelle:

C1	C2	C3
c11	c12	c13
c21	c22	c23
c31	c32	c33

D1	**D2**
d11	d12
d21	d22

il loro prodotto è:

C1	**C2**	**C3**	**D1**	**D2**
c11	c12	c13	d11	d12
c11	c12	c13	d21	d22
c21	c22	c23	d11	d12
c21	c22	c23	d21	d22
c31	c32	c33	d11	d12
c31	c32	c33	d21	d22

Ridenominazione. Dati come argomenti una tabella e una sequenza di attributi che non appartengono a essa, la ridenominazione restituisce una tabella ottenuta dalla tabella di partenza cambiandone tutti gli attributi ordinatamente in quelli della sequenza di attributi data come argomento. Espresso in altri termini, la ridenominazione di una tabella in base a un insieme di nomi consente di ridenominarne ordinatamente le colonne assegnando loro tali nomi. Per esempio, data la tabella:

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

e l'insieme di attributi { A4 , A5 } , applicando la ridenominazione si ottiene:

A4	**A5**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

***Unione** di due tabelle.* Date due tabelle con gli stessi attributi, l'unione restituisce come risultato una tabella contenente tutte le righe delle due tabelle considerate. Per

esempio, date le due tabelle:

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

A1	**A2**	**A3**
e11	e12	e13
e21	e22	e23

la loro unione è:

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43
e11	e12	e13
e21	e22	e23

***Differenza** di due tabelle.* Anche in questo caso le tabelle devono avere la stessa struttura; il risultato è una tabella che contiene tutte le righe della prima escluse quelle contenute nella seconda. In altre parole, la tabella restituita è uguale alla prima tabella epurata dalle righe uguali cioè contenenti gli stessi valori a righe presenti nella seconda tabella. Per esempio, date le due tabelle:

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23

la differenza è:

A1	A2	A3
a31	a32	a33
a41	a42	a43

3.3 OPERATORI DERIVATI

Giunzione (join) di due tabelle. Date due tabelle A e B, e considerati due attributi distinti A1 di A e B1 di B, la giunzione restituisce una tabella contenente tutte le righe di A concatenate alle righe di B per cui i valori di A1 e B1 sono uguali, in cui sono eliminate le righe uguali in quanto ridondanti.

La giunzione può essere ottenuta mediante gli operatori di base definiti nel paragrafo precedente, applicando il seguente procedimento:

1. effettuare il *prodotto* tra le due tabelle A e B;
2. sulla tabella risultante eseguire la *selezione* delle righe in cui i valori degli attributi A1 e B1 sono uguali,

Per esempio, date le due tabelle:

A1	A2	A3
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

G1	G2
a11	g12
a31	g32

la loro giunzione è:

A1	A2	A3	G1	G2
a11	a12	a13	a11	g12
a31	a32	a33	a31	g32

La giunzione viene anche indicata con il termine *equi-giunzione (equi-join),* per distinguerla dal caso in cui la condizione di selezione del punto 2 non sia definita mediante l'operatore di uguaglianza, ma sia una condizione nella forma A1 op B1,

dove op è un qualsiasi operatore di confronto (> , < , <= ecc.).

Giunzione naturale (natural join) di due tabelle. Date due tabelle A e B, aventi un insieme di attributi A1 comuni, la giunzione naturale restituisce una tabella contenente tutte le righe di A concatenate alle righe di B per cui i valori di A1 in A e in B sono uguali, in cui sono eliminati i dati ridondanti, ovvero le righe uguali e uno dei due insiemi di colonne corrispondenti ad A1.

La giunzione naturale può essere ottenuta mediante gli operatori di base definiti nel paragrafo precedente, applicando il seguente procedimento:

1. effettuare la *ridenominazione* degli attributi di cui è composto A1 in A e in B, in modo da rendere distinti gli attributi uguali, ottenendo le tabelle A' e B' ;
2. effettuare il *prodotto* tra le due tabelle A' e B' ;
3. sulla tabella risultante dal punto 2 eseguire la *selezione* delle righe in cui i valori degli attributi ottenuti dalle ridenominazioni di A1 sono uguali;
4. sulla tabella risultante dal punto 3 effettuare una *proiezione* di tutti gli attributi di A, e di quelli di B esclusi quelli contenuti in A1.

Per esempio, date le due tabelle:

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

A1	**A4**
a11	a14
a31	a34

la loro giunzione naturale è:

A1	**A2**	**A3**	**A4**
a11	a12	a13	a14
a31	a32	a33	a34

Semi-giunzione (semi-join) di due tabelle. Date due tabelle A e B, aventi un insieme A1 di attributi comuni, la semi-giunzione restituisce una tabella contenente tutte le righe di A per cui i valori di A1 in A e in B sono uguali.

La semi-giunzione può essere ottenuta effettuando una proiezione degli attributi di A sulla giunzione naturale di A e B. Per esempio, date le due tabelle:

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

A1	**A4**
a11	a14
a31	a34

la loro semi-giunzione è:

A1	**A2**	**A3**
a11	a12	a13
a31	a32	a33

Giunzione esterna (external join) di due tabelle. Date due tabelle A e B, aventi un insieme A1 di attributi comuni, la giunzione esterna restituisce una tabella contenente:

1. tutte le righe di A concatenate alle righe di B per cui i valori di A1 in A e in B sono uguali, in cui sono eliminati i dati ridondanti, ovvero le righe uguali e uno dei due insiemi di colonne corrispondenti ad A1;
2. la concatenazione di tutte le righe di A con una riga di B contenente valori nulli;
3. la concatenazione di tutte le righe di B con una riga di A contenente valori nulli.

La giunzione esterna può essere ottenuta effettuando una unione della giunzione naturale di A e B e delle tabelle ottenute mediante il prodotto di ciascuna delle due tabelle con una riga dell'altra contenente valori nulli. Per esempio, date le due tabelle:

A1	**A2**	**A3**
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

A1	**A4**
a11	a14
a31	a34

la loro giunzione esterna è:

A1	**A2**	**A3**	**A4**
a11	a12	a13	a14
a31	a32	a33	a34
a11	a12	a13	null
a21	a22	a23	null
a31	a32	a33	null
a41	a42	a43	null
a11	null	null	a14
a31	null	null	a34

La giunzione esterna sopra definita è detta anche *giunzione esterna completa.* Esistono anche la *giunzione esterna destra e* la *giunzione esterna sinistra,* in cui mancano rispettivamente le righe definite nel punto *b e* nel punto c.

Considerate le due tabelle utilizzate come operandi nell'esempio precedente, le loro giunzioni esterne destra e sinistra sono rispettivamente le seguenti:

A1	**A2**	**A3**	**A4**
a11	a12	a13	a14
a31	a32	a33	a34
a11	a12	a13	null
a21	a22	a23	null
a31	a32	a33	null
a41	a42	a43	null

A1	**A2**	**A3**	**A4**

a11	a12	a13	a14
a31	a32	a33	a34
a11	null	null	a14
a31	null	null	a34

Intersezione di due tabelle. Date due tabelle con gli stessi attributi, l'intersezione restituisce come risultato una tabella contenente tutte le righe comuni alle due tabelle

considerate. Per esempio, date le due tabelle:

A1	A2	A3
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

A1	A2	A3
a11	a12	a13
a21	a22	a23
a51	a52	a53

la loro intersezione è:

A1	A2	A3
a11	a12	a13
a21	a22	a23

Divisione di una tabella per un'altra tabella. Date due tabelle A e B, dove B ha un sottoinsieme degli attributi di A, la divisione di A per B restituisce come risultato una tabella contenente le righe che, per gli attributi comuni, presentano gli stessi valori in A e in B e le colonne di A corrispondenti agli attributi di A che non sono attributi di B. Per esempio, date le due tabelle:

A1	A2	A3
a11	a12	a13
a21	a22	a23
a31	a32	a33
a41	a42	a43

A1
a11
a21
a51

la divisione della prima per la seconda dà come risultato:

A2	A3
a12	a13
a22	a23

3.4 ALTRI OPERATORI

Complemento di una tabella. Il complemento di una tabella A è una tabella con gli stessi attributi di A, le cui righe sono tutte le possibili righe costituite mediante i valori dei domini che non siano contenute in A. Poiché, come stabilito nella definizione di relazione, i domini sono insiemi finiti di valori, il complemento di una tabella ha un numero finito di righe. Per esempio, considerata la tabella:

dove A1 assume valori in {a11, a21, a31 } e A2 assume valori in {a12, a22}, il complemento della tabella è:

A1	A2
a11	a22
a21	a12
a31	a12
a31	a22

Aggregazione o raggruppamento delle righe di una tabella mediante una giunzione di gruppo. L'operatore di aggregazione permette di creare una nuova tabella che contiene dei valori calcolati in base a insiemi di valori contenuti nella tabella di partenza.

Essa prevede come parametri:

- la tabella di partenza;
- un insieme A dei suoi attributi, che definiscono il raggruppamento;
- una sequenza di espressioni di uguaglianza del tipo B=< espressione >, dove B identifica un attributo da inserire nella nuova tabella; uno degli elementi della sequenza, <espressione >, deve contenere una funzione di gruppo, ovvero una funzione che opera su più valori (per esempio la media, la somma, il valore massimo, il numero degli elementi distinti).

Il risultato dell'operazione è così determinato:

- si raggruppano le righe della tabella di partenza mettendo in un gruppo quelle per cui i valori di A sono uguali;
- si applica a ciascun gruppo la funzione di gruppo;
- si crea una nuova tabella con una riga per ciascun gruppo, e una colonna per ciascun elemento B=<espressione> della sequenza.

Per esempio, considerata la tabella:

A1	A2	A3
a11	a12	10
a21	A12	20
a31	A22	30
a41	A22	40

l'aggregazione degli elementi in base all'attributo A2 per il calcolo della somma dei valori di A3 da come risultato:

A2	**Somma(A3)**
a12	30
A22	70

Chiusura transitiva *di una tabella con due attributi.* Siano A e B i due attributi; il risultato dell'operazione di chiusura transitiva è così determinato: se in due righe della tabella i valori a1 di A e b1 di B per la prima riga e a2 di A e b2 di B per la seconda riga sono tali che b1 =a2**,** allora si aggiunge alla tabella la riga che contiene i valori a1 in A e b2 in B; si itera l'applicazione di questo procedimento finché vengono aggiunte righe alla tabella risultante. Per esempio, considerata la tabella:

A1	A2
a11	a12
a12	a13
a21	a22
a22	a23

la chiusura transitiva dà come risultato:

A1	A2
a11	a12
a12	a13
a11	a13
a21	a22
a22	a23
a21	a23

Nel risultato, alla tabella originaria è stata aggiunta la riga contenente a11 e a13, poiché sono presenti la riga contenente a11 e a12 a quella contenente a12 e a13. Inoltre, è stata aggiunta la riga contenente a21 e a23 perché sono presenti la riga contenente a21 e a22 e quella contenente a22 e a23.

4 Query

4.1 INTERROGAZIONI SUL DATABASE

Iniziamo il percorso di apprendimento da uno degli aspetti primari del linguaggio: l'interrogazione del database. I comandi d'interrogazione – query - consentono di effettuare operazioni di ricerca sui dati contenuti nelle tabelle del database, impostando le condizioni che tali dati devono soddisfare. Se si utilizza un interprete di comandi, il risultato dell'interrogazione verrà direttamente visualizzato sullo schermo. Tutte le interrogazioni sul database vengono effettuate utilizzando in maniera diretta o indiretta il comando di selezione specificato mediante l'operazione SELECT.

Le selezioni operano su tabelle e restituiscono come risultato una tabella. La sintassi del comando di selezione è la seguente:

<Comando-select> ::=

SELECT [ALL | DISTINCT] <lista-di-selezione>

<espressione-di-tabella>

[<clausola-di-ordinamento>]

dove <espressione-di-tabella> è così definita:

<espressione-di-tabella> ::=

<clausola-FROM>

[<clausola-WHERE>]

[<clausola-GROUP-BY>]

[<clausola-HAVING>]

Come risulta dalla sintassi, nel comando è obbligatorio specificare, oltre alla <lista di selezione>, la sola clausola FROM. Le specifiche d'interrogazione – ALL e DISTINCT - la cui funzione sarà spiegata più avanti - e le altre clausole sono opzionali (pertanto nella sintassi sono riportate tra parentesi quadra). ALL e DISTINCT sono tra loro alternative (nella sintassi sono quindi separate da |); in mancanza di entrambe viene assunta valida per default la specifica ALL.

La clausola FROM indica la tabella o le tabelle su cui eseguire la selezione:

<clausola-FROM> ::= FROM <nome-tabella> [{ ,<nome-tabella> } …]

Deve essere specificato il nome di almeno una tabella; successivamente, separati da virgole, possono essere indicati i nomi di altre tabelle.

Gli esempi e gli esercizi di questo capitolo si riferiscono a interrogazioni che operano su una tabella, perciò la clausola FROM sarà sempre seguita dal nome di una sola tabella. Nei paragrafi successivi verranno esaminate nel dettaglio le componenti del comando di selezione.

4.2 LA LISTA DI SELEZIONE

La forma più semplice del comando di selezione è quella in cui la lista di selezione è costituita da uno o più nomi di colonne della stessa tabella, ovvero:

<lista_di_selezione> ::= <nome_colonna> [{ ,<nome_colonna>) } …]

Il risultato del comando è una tabella che contiene le colonne indicate della tabella specificata nella clausola FROM. Questa forma del comando di selezione realizza direttamente l'operazione di

proiezione definita nell'algebra relazionale: data una tabella e un insieme di attributi (nomi delle colonne), la proiezione restituisce una tabella con tutte le righe di quella di partenza, ma con solo le colonne identificate dagli attributi. Osserviamo il seguente esempio:

```
SELECT Cod_Modello, Nome_Modello
FROM Modelli
```

Abbiamo considerato la tabella Modelli del database Registro_Automobilistico già introdotto, e la abbiamo utilizzata nella clausola FROM come <nome tabella>; abbiamo inoltre utilizzato le colonne Cod_Modello e Nome_Modello nella <lista_di_selezione>. Il risultato è:

Cod_Modello	Nome_Modello
001	PANDA
002	VESPA
003	BRAVA
004	MONDEO
005	V-10
006	DUCATO
007	CLIO
008	COROLLA
009	COUPE'
010	GOLF
011	MEGANE
012	SEICENTO
013	LAGUNA
014	CIVIC

Il comando restituisce i valori delle due colonne indicate relativi a tutte le righe della tabella. Nello stesso modo è possibile scrivere comandi SELECT che permettano di ottenere, data una tabella, i valori di qualsiasi colonna.

È da notare che l'ordine in cui viene chiesta la restituzione delle colonne è indipendente dall'ordine utilizzato al momento della costruzione della tabella. Qualora fosse stato necessario avremmo potuto richiedere le colonne in ordine opposto, semplicemente scambiandole nella lista di selezione:

```
SELECT Nome_Modello, Cod_Modello      --Esempio
FROM Modelli    -- di commento
```

I commenti all'interno di un comando SQL devono iniziare con una serie qualsiasi di due o più trattini (segni meno) consecutivi, seguiti da una sequenza qualsiasi di caratteri che termina con la prima indicazione di fine riga.

E' possibile ridenominare le colonne della tabella risultante nel modo seguente:

```
SELECT <nome_colonna> '<nome>' [ { ,<nome_colonna> '<nome>' } ...]
FROM <nome_tabella>
```

In questo modo <nome_colonna> viene ridenominato con il <nome> che la segue, racchiuso tra apici. Questa forma del comando di selezione realizza l'operazione di ridenominazione definita nell'algebra relazionale. Riprendiamo il comando SELECT precedente e modifichiamolo ridenominando le colonne:

```
SELECT Cod_Modello 'Codice', Nome_Modello 'Nome
```

```
FROM Modelli
```

Il comando restituisce gli stessi valori del precedente, con la differenza che i nomi delle colonne Cod_Modello e Nome_Modello appaiono nel risultato rispettivamente come Codice e Nome:

Codice	Nome
001	PANDA
002	VESPA
003	BRAVA
004	MONDEO
005	V-10
006	DUCATO
007	CLIO
008	COROLLA
009	COUPE'
010	GOLF
011	MEGANE
012	SEICENTO
013	LAGUNA
014	CIVIC

Quando il nome che sostituisce <nome_colonna> contiene il carattere apice, questo deve essere raddoppiato per distinguerlo dal carattere di fine stringa. Per esempio, utilizzando la tabella Veicoli, nel comando

```
SELECT Velocità ‘ Veicolo: velocità ‘ ‘
FROM Veicoli
```

dobbiamo aggiungere all’apice che accenta la a di velocità un ulteriore apice;

l’ultimo apice chiude la stringa. Se avessimo voluto ridenominare il nome della colonna con *Velocità dei Veicoli* avremmo scritto: ’ Velocità ' dei Veicoli ' .

Qualora sia necessario selezionare tutte le colonne di una tabella, la notazione fin qui adoperata può risultare prolissa; in questa situazione è possibile riferirsi a tutte le colonne tramite la seguente notazione abbreviata. in cui la lista di selezione è sostituita da *:

```
SELECT *
FORM <nome_tabella>
```

Il comando successivo restituisce quindi tutte le colonne della tabella Modelli:

```
SELECT *
FORM Modelli
```

Questa notazione estremamente sintetica è spesso utilizzata al fine di ottenere rapidamente una visione d’insieme di una tabella, per poi procedere a selezioni più ristrette.

Cod_Modello	Nome_Modello	Cod_Fabbrica	Numero_Versioni
001	PANDA	001	3
002	VESPA	004	4
003	BRAVA	001	2
004	MONDEO	003	3
005	V-10	005	2
006	DUCATO	001	5
007	CLIO	006	5
008	COROLLA	007	4
009	COUPE'	001	1
010	GOLF	008	4
011	MEGANE	006	2
012	SEICENTO	001	2
013	LAGUNA	006	2
014	CIVIC	009	3

Lo standard non ammette l'utilizzo misto in un'unica clausola di selezione delle notazioni * e <nome colonna>, sebbene alcuni DBMS lo permettano. Per esempio, se volessimo visualizzare tutte le colonne della tabella Veicoli seguite di nuovo dalla velocità moltiplicata per 1000, potremmo scrivere:

```
SELECT *, Velocità*1000   -- Non standard ANSI
FROM Veicoli
```

che non risulta conforme allo standard. L'uso di un'espressione. in questo caso Velocità * 1000 al posto di un nome colonna è invece possibile. e verrà trattato nel seguito.

Supponiamo ora di voler trovare tutti i modelli che compaiono nella tabella Veicoli:

```
SELECT Cod_Modello 'Modelli presenti'
FROM Veicoli
```

Il risultato è una tabella con tante righe quante sono le righe di Veicoli, alcune delle quali contengono lo stesso codice modello.

Modelli presenti
004
001
001
009
006
007
002
003
014
011
013
010
012
005
008

Per eliminare i valori duplicati esiste la specifica d'interrogazione DISTINCT, che troviamo nella sintassi generale del comando SELECT come specifica opzionale:

```
SELECT DISTINCT Cod_Modello 'Modelli presenti'
FROM Veicoli
```

Il risultato ottenuto è la prossima tabella, dove nessun valore di Cod_Modello viene mai ripetuto.

La clausola DISTINCT può apparire, all'interno di un comando SELECT, al più una volta. E' d'altra parte intuitivo che il comando

```
SELECT     DISTINCT Cod_Modello,    -- Non corretto
           DISTINCT Cod_Categoria
FROM Veicoli
```

non specifichi correttamente quali righe debbano essere restituite (una riga per ciascun valore distinto di Cod_Modello o una riga per ciascun valore distinto di Cod_Categoria?).

Modelli presenti
004
001
009
006
007
002
003
014
011
013
010
012
005
008

4.3 LA CLAUSOLA WHERE

La clausola WHERE permette di specificare delle condizioni nella selezione. La sua sintassi è la seguente:

<clausola_WHERE> ::= WHERE <condizione>

L'uso della clausola WHERE nel comando di selezione realizza l'operazione di *.selezione o restrizione* definita nell'algebra relazionale: data una tabella e una condizione logica definita sui suoi attributi, la selezione restituisce una tabella con gli stessi attributi di quella di partenza, ma con le sole righe che soddisfano la condizione.

La clausola WHERE non è l'unico modo per operare delle restrizioni sulle righe di una tabella, poiché, per esempio, anche la clausola DISTINCT che abbiamo già esaminato opera in tal senso; tuttavia è certamente il modo più flessibile e più frequentemente utilizzato.

Si noti che il risultato di una selezione che utilizzi la clausola WHERE su una ta-bella non vuota può anche essere un insieme di righe vuoto.

Le condizioni della clausola WHERE sono specificate mediante gli operatori di confronto, i connettori logici e gli operatori BETWEEN, IN, LIKE, IS NULL, che studieremo nei prossimi paragrafi.

4.3.1 Operatori di confronto

Le condizioni più frequentemente utilizzate sono definite in base agli operatori di confronto, che permettono di comparare due espressioni. Casi tipici sono quelli in cui i valori contenuti nelle colonne di una tabella vengono confrontati con delle co-stanti, con i valori di altre colonne della stessa riga della tabella o in generale con delle espressioni. Gli operatori di confronto sono elencati nella Tabella 5.1.

Tabella 5.1 Operatori di confronto

=	uguale	<	minore
<>	diverso	>=	maggiore o uguale
>	maggiore	<=	minore o uguale

Vediamo un primo esempio. in cui <condizione> nella sintassi della clausola WHERE corrisponde a Cilindrata>1000:

```
SELECT *
FROM Veicoli
WHERE Cilindrata>1000
```

Il comando dà come risultato:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
A123456X	004	01	1796	01	85	195	5	30/12/98
C76589AG	001	01	1106	01	54	130	5	13/08/98
C78905GT	009	01	1998	01	117	212	4	06/11/94
CFT346VB	007	01	1390	02	75	170	5	12/01/95
DD4567XX	003	01	1581	01	17		5	05/06/97
DGH789JC	014	01	1590	02	114	170	5	05/10/95
DH79567H	011	01	1589	04	107	170	0	
ERG567NM	013	01	1598	04	107	175	5	18/12/97
F96154NH	010	01	1781	03	125	185	5	08/03/92
XCH56GJK	005	01	1918	01	110	210	5	04/09/98
XF5789GY	008	01	1587	01	107	175	5	05/05/96

Eseguendo il comando

```
SELECT *
FROM Veicoli
WHERE Cilindrata>2000
```

Si ottiene, invece una tabella vuota.

Un terzo esempio mostra l'uso dell'operatore di uguaglianza:

```
SELECT *
FROM Veicoli
WHERE Targa = 'A123456X'
```

Il comando seleziona le righe in cui Targa è uguale ad A123456X.

Quest'ultimo valore è incluso tra apici poiché specifica una sequenza di caratteri, essendo la colonna con cui viene confrontato di tipo alfanumerico. Il risultato è:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
A123456X	004	01	1796	01	85	195	5	30/12/98

In questo caso specifico è particolarmente significativa la restituzione di una sola riga, poiché abbiamo utilizzato l'operatore di uguaglianza nel confronto con la colonna Targa che rappresenta la chiave primaria, univoca per definizione, della tabella Veicoli.

L'utilizzo della clausola WHERE nel comando di selezione esemplifica quanto abbiamo affermato nella teoria relazionale: nei linguaggi dichiarativi, non navigazionali, basati sul concetto di elaborazione di insiemi, è sufficiente specificare una condizione per la selezione dei dati, senza dover effettuare una scansione delle tabelle utilizzando i concetti di riga precedente e riga successiva.

Come esempio di confronto dei valori contenuti in due colonne di una tabella, selezioniamo tutti i veicoli per I quali la cilindrata è maggiore dei cavalli fiscali moltiplicati per 20:

```
SELECT *
FROM Veicoli
WHERE Cilindrata>Cavalli_Fiscali*20
```

Il risultato è:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
B256787Y	001	01	708	02	10	120	5	21/09/89
A123456X	004	01	1796	01	85	195	5	30/12/98
C76589AG	001	01	1106	01	54	130	5	13/08/98
DD4567XX	003	01	1581	01	17		5	05/06/97
FGH673FV	012	01	899	01	39	140	5	09/08/98

In generale, i valori confrontati dagli operatori di confronto possono essere anche costanti ed espressioni calcolate sui valori di più colonne. Le espressioni aritmetiche possono contenere gli usuali operatori + , - , * , / per effettuare addizioni, sottrazioni, moltiplicazioni e divisioni. Anche i valori alfanumerici possono essere utilizzati nelle espressioni; in questo caso verranno impiegati operatori e funzioni che operano su sequenze di caratteri, quale per esempio l'operatore di concatenazione. Esempi di espressioni contenenti valori alfanumerici verranno riportati nel seguito.

4.3.2 Connettori logici

È possibile specificare condizioni multiple combinando più condizioni tramite i connettori logici AND, OR e NOT.

La condizione ottenuta utilizzando l'operatore AND tra due condizioni è verificata quando entrambe le condizioni risultano verificate. Vediamone un'applicazione sulla tabella Veicoli:

```
SELECT *
FROM Veicoli
WHERE Cod_Combustibile='01' AND Cilindrata>1000
```

Il comando seleziona le righe in cui il valore di Cod_Combustibile è uguale a 01 e contemporaneamente il valore di Cilindrata a è maggiore di 1000:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
A123456X	004	01	1796	01	85	195	5	30/12/98
C76589AG	001	01	1106	01	54	130	5	13/08/98
DD4567XX	003	01	1581	01	17		5	05/06/97
XCH56GJK	005	01	1918	01	110	210	5	04/09/98
XF5789GY	008	01	1587	01	107	175	5	05/05/96
C78905GT	009	01	1998	01	117	212	4	06/11/94

La condizione ottenuta utilizzando l'operatore OR tra due condizioni è invece verificata quando risulta verificata almeno una delle condizioni, come nell'esempio che segue:

```
SELECT *
FROM Veicoli
WHERE Cod_Modello='001' OR Cod_Modello='004'
```

Il comando seleziona le righe in cui il valore di Cod_Modello è uguale a 001 e le righe in cui il valore di Cod_Modello è uguale a 004:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
B256787Y	001	01	708	02	10	120	5	21/09/89
A123456X	004	01	1796	01	85	195	5	30/12/98
C78905GT	009	01	1998	01	117	212	4	06/11/94

Infine, la condizione ottenuta applicando l'operatore NOT a una condizione è ve-rificata quando quest'ultima condizione non è verificata.

Il comando

```
SELECT *
FROM Veicoli
WHERE NOT Cod_Categoria='01'
```

seleziona le righe in cui il valore di Cod_Categoria non è uguale a 01. Il risultato è:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
C845905Z	006	04		03			3	11/04/95
D238765W	002	03		01			2	12/08/97

In questo caso lo stesso risultato poteva essere ottenuto scrivendo WHERE Cod Categoria<>'01' ; in altre circostanze le alternative non sono così immediate.

Tramite l'operatore NOT si realizza l'operazione di *complemento* di una tabella rispetto a una condizione, definita nel Capitolo 3, dedicato all'algebra relazionale. L'ordine di valutazione degli operatori prevede che vengano prima applicati gli operatori NOT, poi gli AND, infine gli OR. Questo significa per esempio che nella condizione multipla

NOT Cod_Categoria='01' AND Cilindrata>1500

l'operatore NOT è riferito solo alla condizione Cod_Categoria='01'. Nella condizione multipla

Cod_Categoria='03' OR Cod_Categoria='01' AND Cilindrata>1500

Viene prima eseguito l'AND tra Cod_Categoria='01' e Cilindrata>1500 e successivamente il risultato viene combinato in OR con Cod_Categoria='03'.

Per definire un ordine diverso di valutazione degli operatori nelle condizioni multiple possono essere usate le parentesi tonde:

NOT (Cod_Categoria='01' AND Cilindrata>1500)

In questo caso il NOT è riferito a tutta la condizione contenuta nella parentesi.

Nel seguente esempio:

(Cod_Categoria='01' OR Cod_Categoria='03') AND Cod_Combustibile='01'

viene eseguito prima l'OR tra le prime due condizioni, e successivamente il risultato viene combinato in AND con la terza condizione.

Le coppie di parentesi ( ) possono essere annidate una nell'altra, come nel seguente esempio, in cui vengono restituiti tutti veicoli esclusi quelli il cui codice combustibile è 01 o 02 con cilindrata minore di 1500:

NOT ((Cod_Combustibile='01' OR Cod_Combustibile='02') AND Cilindrata<1500)

4.3.3 BETWEEN

L'operatore BETWEEN verifica se un argomento è compreso in un intervallo di valori; esso utilizza la seguente sintassi:

<operando> BETWEEN <operando> AND <operando>;

Normalmente il primo operando è un nome di colonna. Nel comando successivo si ottengono i veicoli la cui cilindrata è compresa tra 1500 e 1800:

```
SELECT *
FROM Veicoli
WHERE Cilindrata BETWEEN 1500 AND 1800
```

La precedente è una notazione equivalente ma più sintetica e intuitiva di:

```
SELECT *
FROM Veicoli
WHERE Cilindrata>=1500 AND Cilindrata<=1800
```

I programmatori SQL, comunque, tendono a usare quest'ultima soluzione, sia per l'abitudine a lavorare con gli operatori di confronto, sia perché condizioni logiche complesse richiedono numerosi operatori e parentesi annidate, per cui può risultare più chiaro e immediato l'impiego di operatori omogenei. È da notare, inoltre, che l'uso dell'operatore BETWEEN richiede che gli operandi siano espressi nell'ordine corretto, mentre questo non vale per le condizioni espresse tramite operatori di confronto, che producono lo stesso risultato a prescindere dall'ordine in cui vengono utilizzate.

Il risultato del comando precedente è:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
A123456X	004	01	1796	01	85	195	5	30/12/98
DD4567XX	003	01	1581	01	17		5	05/06/97
DGH789JC	014	01	1590	02	114	170	5	05/10/95
DH79567H	011	01	1589	04	107	170	0	
ERG567NM	013	01	1598	04	107	175	5	18/12/97
F96154NH	010	01	1781	03	125	185	5	08/03/92
XF5789GY	008	01	1587	01	107	175	5	05/05/96

4.3.4 IN

L'operatore IN verifica se un operando è contenuto in una sequenza di valori.

Nell'esempio selezioniamo le righe della tabella Veicoli in cui Targa è uguale ad A123456X, a D238765W o a XCH56GJK:

```
SELECT *
FROM Veicoli
WHERE Targa IN ('A123456Xt,'D238765W','XCH56GJK')
```

Naturalmente avremmo anche potuto scrivere:

```
SELECT *
```

```
FROM Veicoli
WHERE Targa='A123456X' OR Targa='D238765W' OR Targa='XCH56GJK'
```

Il primo comando è più sintetico e chiaro, in particolare qualora il numero di valori sia elevato; IN si ricollega esplicitamente agli operatori insiemistici. Il comando produce quanto riportato nella tabella di pagina seguente.

Vedremo inoltre in seguito che l'utilizzo dell'operatore IN diviene essenziale qualora la lista di valori sui quali eseguire il confronto rappresenti il risultato di un'altra operazione di selezione.

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
A123456X	004	01	1796	01	85	195	5	30/12/98
D238765W	002	03		01			2	12/08/97
XCH56GJK	005	01	1918	01	110	210	5	04/09/98

4.3.5 LIKE

L'operatore LIKE verifica se una stringa di caratteri corrisponde a un determinato formato. Per definire il formato sono utilizzabili i seguenti caratteri jolly:

_ indica un singolo carattere qualsiasi;

% indica una sequenza di caratteri qualsiasi.

Nel comando

```
SELECT *
FROM Veicoli
WHERE Targa LIKE 'C__5___'
```

si richiedono le righe della tabella Veicoli dove la Targa è composta da otto ca-ratteri di cui il primo è C, il secondo e il terzo possono essere qualsiasi, il quarto deve essere 5 e gli ultimi quattro di nuovo qualsiasi. Ecco il risultato:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
C76589AG	001	01	1106	01	54	130	5	13/08/98
C845905Z	006	04		03			3	11/04/95

I caratteri maiuscoli vengono distinti dai corrispondenti minuscoli, a meno di non aver utilizzato delle opzioni particolari nella definizione dello schema logico; per-tanto il formato 'C__5___' risulta diverso da 'c__5___' .

Nell'esempio successivo la stringa in Targa deve iniziare con il carattere C se-guito da una sequenza di caratteri qualsiasi:

```
SELECT *
FROM Veicoli
WHERE Targa LIKE 'C%'
```

Il risultato è:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
C76589AG	001	01	1106	01	54	130	5	13/08/98

C78905GT	009	01	1998	01	117	212	4	06/11/94
C845905Z	006	04		03			3	11/04/95
CFT346VB	007	01	1390	02	75	170	5	12/01/95

Anche se il contenuto del database Registro_Automobilistico non le prevede per omogeneità, se fossero state presenti Targhe con il primo carattere uguale a C di lunghezza maggiore o minore a otto caratteri, come C0333398V, le righe corrispondenti sarebbero state restituite.

Come abbiamo visto i caratteri _ e % vengono utilizzati come caratteri speciali. Si possono tuttavia presentare situazioni in cui gli stessi caratteri _ e % facciano parte del contenuto stesso delle colonne e debbano essere utilizzati nei confronti. Si consideri il caso in cui tali caratteri siano contenuti nella colonna Nome_Modello della tabella Modelli e vogliamo selezionare tutte le righe di tale tabella che corrispondono a un formato che li contiene. In tal caso è necessario operare nel modo seguente; supponiamo che sia il carattere _ da ricercare:

- individuare un particolare carattere, detto carattere di *escape**,*** che non sia am-missibile nella colonna su cui si sta impostando la condizione LIKE;
- nella specifica del formato da confrontare far precedere _, quando tale carattere debba essere utilizzato come carattere di confronto e non come carattere speciale, dal carattere di escape;
- specificare il carattere di escape individuato facendolo precedere dalla parola chiave ESCAPE, subito dopo il formato da confrontare.

Per esempio, per trovare tutti i modelli il cui nome inizia con C_:

```
SELECT *
FROM Modelli
WHERE Nome-Modello LIKE 'C#_%' ESCAPE '#'
```

In questo caso abbiamo scelto # come carattere di escape; il carattere è utilizzato come carattere di confronto, essendo preceduto dal carattere di escape, mentre il carattere % mantiene il suo ruolo di carattere speciale. La procedura descritta può naturalmente essere applicata anche al carattere %.

4.3.6 Is NULL

L'operatore IS NULL verifica se il contenuto di un operando è NULL. Quest'ultimo è uno stato che indica che l'elemento della tabella nella specifica colonna non con-tiene alcun valore. (Si tenga presente che zero è un valore così come una stringa vuota.) Per esempio, il comando

```
SELECT *
FROM Veicoli
WHERE Cilindrata IS NULL
```

restituisce le linee in cui il valore della colonna Cilindrata è NULL, cioè non contiene alcun valore, come segue:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
C845905Z	006	04		03			3	11/04/95
D238765W	002	03		01			2	12/08/97

Gli ultimi operatori descritti sono presenti anche in forma negativa, ovvero nella forma:

NOT BETWEEN

NOT IN

NOT LIKE

NOT NULL

Nell’esempio

```
SELECT *
FROM Veicoli
WHERE Cilindrata NOT BETWEEN 1500 AND 1800
```

vengono perciò restituite le righe in cui la cilindrata non è compresa tra 1500 e 1800, o in altri termini è minore di 1500 o maggiore di 1800:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
B256787Y	001	01	708	02	10	120	5	21/09/89
C76589AG	001	01	1106	01	54	130	5	13/08/98
C78905GT	009	01	1998	01	117	212	4	06/11/94
C845905Z	006	04		03			3	11/04/95
CFT346VB	007	01	1390	02	75	170	5	12/01/95
D238765W	002	03		01			2	12/08/97
FGH673FV	012	01	899	01	39	140	5	09/08/98
XCH56GJK	005	01	1918	01	110	210	5	04/09/98

Analogamente, l'utilizzo dell'operatore NOT IN è utile per escludere un insieme

di valori che non qualificano un'operazione di selezione.

4.4 CALCOLO DI ESPRESSIONI

La lista di selezione di un comando SELECT può contenere non solo nomi di colonne, come negli esempi sinora visti, ma anche espressioni calcolate sui valori di una o più colonne. Per esempio:

```
SELECT Targa, Velocità * 1000 'Velocità' in m/h'
FROM Veicoli
```

Le colonne restituite riporteranno per ogni veicolo la targa e la velocità in metri all'ora, calcolata moltiplicando la velocità per 1000. Questa colonna viene denominata Velocità' in m/ h. Il risultato sarà quello della prima tabella di pagina seguente.

Se desideriamo ottenere l'elenco dei nomi dei modelli con accanto, tra parentesi, il loro codice, possiamo concatenare le colonne Nome_Modello e Cod_Modello con l'operatore | | .

```
SELECT Nome_Modello I l ' (' I l Cod_Modello I l ')' 'Elenco modelli'
FROM Modelli
```

Targa	Velocità in m/h
A123456X	195000
B256787Y	120000
C76589AG	130000
C78905GT	212000
C845905Z	
CFT346VB	170000
D238765W	
DD4567XX	
DGH789JC	170000
DH79567H	170000
ERG567NM	175000
F96154NH	185000
FGH673FV	140000
XCH56GJK	210000
XF5789GY	175000

Abbiamo denominato la nuova colonna Elenco Modelli . Il risultato è:

Elenco modelli
PANDA (001)
VESPA (002)
BRAVA (003)
MONDEO (004)
V-10(005)
DUCATO (006)
CLIO (007)
COROLLA(008)
COUPÈ (009)
GOLF (010)
MEGANE (011)
SEICENTO(012)
LAGUNA(013)
CIVIC (014)

Nelle espressioni possono essere usati tutti gli operatori e le funzioni definite per i tipi delle colonne considerate. Nella valutazione di un'espressione, se un operando è NULL, allora il risultato è NULL. Nel primo esempio del paragrafo la velocità in m/h è NULL, cioè non specificata, per i veicoli per cui la Velocità è NULL.

Le espressioni possono essere utilizzate all'interno di un comando SELECT nella lista di selezione e nella clausola WHERE, come abbiamo visto, dove possa essere inserito il riferimento a una colonna.

4.5 FUNZIONI DI GRUPPO

Le espressioni finora considerate operano sulle singole righe di una tabella, restituendo sempre una tabella. Esistono però delle funzioni che consentono di calcolare espressioni su insiemi di righe e che, a differenza delle precedenti, restituiscono un caso molto particolare di tabella, ovvero un singolo valore scalare. Le funzioni di-sponibili sono le seguenti:

MAX per la determinazione del valore massimo

MIN per la determinazione del valore minimo

SUM per il calcolo della somma di valori

AVG per il calcolo della media di valori

COUNT per il conteggio di valori

4.5.1 MAX e MIN

Le funzioni MAX e MIN calcolano rispettivamente il maggiore e il minore dei valori di una colonna. Per esempio:

SELECT MAX(Cilindrata) 'Cilindrata massima'

FROM Veicoli

La funzione MAX applicata alla colonna Cilindrata prende in considerazione i valori di questa colonna e ne restituisce il più alto:

Cilindrata Massima
1998

Le funzioni che operano su insiemi di righe evidenziano da una parte la potenza del linguaggio e dall'altra le differenze con i linguaggi procedurali. Per esempio, al fine di reperire il maggiore di una lista il programmatore C o Cobol avrebbe dovuto combinare l'utilizzo di cicli e confronti, scrivendo codice più complesso e certa-mente meno chiaro di quello SQL. Ovviamente le funzioni di gruppo possono operare su un sottoinsieme delle righe di una tabella, come nel seguente esempio:

```
SELECT MIN(Cilindrata) 'Cilindrata minima'
FROM Veicoli
WHERE Cod_Combustibile='01'
```

Il risultato è:

Cilindrata minima
899

In questo caso la funzione MIN è applicata ai soli valori della colonna Cilindrata relativi alle righe in cui Cod_Combustibile='01'.

4.5.2 SUM

La funzione SUM calcola la somma dei valori di una colonna. Tramite le specifiche ALL e DISTINCT è possibile indicare se devono essere sommati rispettivamente tutti i valori o solo i valori distinti. Il default è ALL.

Nell'esempio seguente vengono selezionate tutte le righe della tabella Modelli in cui Cod_Fabbrica è uguale a 001, e su queste viene calcolata la somma dei valori della colonna Numero_Versioni:

```
SELECT SUM(Numero_Versioni) 'Numero versioni'
FROM Modelli
WHERE Cod_Fabbrica='001'
```

Il risultato è:

Numero versioni
13

4.5.3 AVG

La funzione AVG calcola la media (average) dei valori non nulli di una colonna.

Tramite le specifiche ALL e DISTINCT è possibile indicare se devono essere considerati tutti i valori o solo i valori distinti. Il default è ALL. Con il comando

```
SELECT AVG(Cilindrata) 'Cilindrata media'
FROM Veicoli
```

viene calcolata la media della colonna Cilindrata delle righe di Veicoli, sommando le 13 cilindrate e dividendo il risultato per 13. Si noti che nel calcolo non sono considerati i valori nulli (NULL), mentre sarebbero considerati eventuali valori posti a zero. Il risultato è:

Cilindrata media
1503,15384615385

Indichiamo con *n* il numero dei valori non nulli di una colonna e con m il numero dei valori distinti. Se nessun valore nella colonna si ripete allora *n* è uguale a m. La clausola DISTINCT fa sì che AVG effettui la somma degli m valori distinti e divida

il risultato per m. Se *n* è uguale a m si ottiene lo stesso risultato specificando o no la clausola DISTINCT.

Nell'esempio successivo nel calcolo della media desideriamo prendere in consi-derazione solamente i valori differenti della colonna Cilindrata o, in altre parole, considerare una sola volta ciascun valore. A questo scopo facciamo precedere il nome della colonna su cui opera la funzione AVG dalla specifica DISTINCT.

```
SELECT AVG(DISTINCT Cilindrata) 'Cilindrata media'
FROM Veicoli
```

Nel caso specifico il risultato è uguale a quello ottenuto nell'esempio precedente, in quanto non vi sono valori uguali.

4.5.4 COUNT

La funzione COUNT conta le righe. Tramite le specifiche *, ALL e DISTINCT è possibile contare rispettivamente le righe selezionate, tutti i valori non nulli, i valori distinti. Il default è ALL. Il comando

```
SELECT COUNT(*) 'Numero Righe Veicoli'
FROM Veicoli
```

calcola il numero di righe della tabella Veicoli:

Numero Righe Veicoli
15

Se desideriamo ottenere il numero delle righe in cui Cavalli_Fiscali è diverso da NULL scriveremo

```
SELECT COUNT(ALL Cavalli_Fiscali) 'Numero Cavalli Fiscali'
FROM Veicoli
```

ottenendo come risultato:

Numero Cavalli Fiscali
13

Due dei veicoli sono stati esclusi dal conteggio perché non hanno specificato alcun valore per i cavalli fiscali. Eventuali veicoli con cavalli fiscali uguale a zero non verrebbero invece esclusi. La specifica ALL, come nella maggior parte dei comandi.di selezione, viene utilizzata come default ed è pertanto possibile ometterla. Vediamo ora l'uso della specifica DISTINCT nella funzione di gruppo COUNT. Il comando

```
SELECT COUNT(DISTINCT Cavalli_Fiscali) 'Numero Cavalli Fiscali Distinti'
FROM Veicoli
```

calcola il numero di righe della tabella Veicoli che hanno cavalli fiscali differenti, producendo come risultato:

Numero Cavalli Fiscali Distinti
11

Il risultato è 11 poiché esistono tre veicoli con lo stesso valore di cavalli fiscali.

È importante notare che, per quanto visto finora, non è possibile utilizzare in una stessa lista di selezione funzioni di gruppo e funzioni su singole righe; non è possi-bile, per esempio, ricercare la targa del veicolo con cilindrata più alta. In seguito verranno esaminate le tecniche per risolvere questo tipo di problema.

4.6 LA CLAUSOLA GROUP BY

La clausola GROUP BY aumenta ulteriormente le possibilità di interrogazione tra-mite le funzioni di gruppo. Si supponga di dover calcolare il numero totale di ver-sioni di modelli prodotti da ciascuna fabbrica. Per farlo dobbiamo reperire tutte le differenti fabbriche presenti nella tabella Modelli; successivamente per ogni fabbrica dobbiamo calcolare la somma delle versioni. Utilizzando gli strumenti del linguaggio esaminati finora i passi da fare sono i seguenti.

1. Selezioniamo tutti i codici distinti delle fabbriche presenti nella colonna Cod_Fabbrica della tabella Modelli:

```
SELECT
DISTINCT Cod_Fabbrica
FROM Modelli
```

Dobbiamo poi annotare su un foglio di carta i codici delle categorie restituiti dal comando:

Cod_Fabbrica
001
004
003
005
006
007
008
009

2. Applichiamo la funzione di gruppo SUM per il calcolo della somma dei valori della colonna Numero_Versioni di Modelli per ciascun codice fabbrica che abbiamo annotato.

2.1 Iniziamo con la fabbrica 001:

```
SELECT SUM(Numero-Versioni)'Numero versioni fabbrica 001'
FROM Modelli
WHERE Cod_Fabbrica = '001'
```

Il risultato è:

Numero versioni fabbrica 001
13

2.2 Proseguiamo con la fabbrica 004:

```
SELECT SUM(Numero-Versioni) 'Numero versioni fabbrica 004'
FROM Modelli
WHERE Cod_Fabbrica = '004'
```

Il risultato è:

Numero versioni fabbrica 004
4

Procederemo analogamente per tutte le altre fabbriche.

Il linguaggio SQL dà un'altra possibilità, che permette di evitare passi intermedi. La clausola GROUP BY viene in aiuto in casi di questo genere, poiché consente di calcolare espressioni su insiemi di righe. Il suo formato è:

<clausola_GROUP_BY> ::= GROUP BY <nome_colonna>

Il problema precedente viene così risolto molto più semplicemente.

```
SELECT Cod_Fabbrica, SUM(Numero_Versioni) 'Numero versioni per fabbrica'
FROM Veicoli
GROUP BY Cod_Fabbrica
```

La clausola GROUP BY applicata alla colonna Cod_Fabbrica realizza esattamente quello che facevamo manualmente nei passi 2.1, 2.2 ecc. Per ogni valore differente in Cod_Fabbrica reperisce e raggruppa tutte le righe; come specificato nella lista di selezione, vengono visualizzati i codici delle fabbriche Cod_Fabbrica e le somme delle versioni per le diverse fabbriche SUM (Numero Versioni):

Cod_Fabbrica	Numero versioni per fabbrica
001	13
004	3
003	4
005	2
006	9
007	4
008	4
009	3

Notiamo che in questo caso è possibile utilizzare in una stessa lista di selezione una funzione su singole righe (Cod_Fabbrica) e una funzione di gruppo (SUM), perché la prima coincide per tutti i valori coinvolti nella seconda, essendo questi così vincolati dalla clausola GROUP BY.

La clausola GROUP BY consente di realizzare l'operazione di aggregazione o raggruppamento definita nel Capitolo 3.

4.7 LA CLAUSOLA HAVING

La clausola HAVING può essere usata correttamente solo in combinazione con la clausola GROUP BY; essa consente di porre una condizione su una funzione di gruppo, come la clausola WHERE lo consente su singole righe.

Si supponga per esempio di voler calcolare la somma delle versioni prodotte da ciascuna fabbrica, come nel caso precedente, ma solo per le fabbriche con almeno 2 modelli:

```
SELECT Cod_Fabbrica, COUNT(*) 'Numero modelli', SUM(Numero_Versioni) 'Numero versioni'
FROM Modelli
GROUP BY Cod_Fabbrica
HAVING COUNT(*) >= 2
```

La clausola GROUP BY indica che per ogni valore differente in Cod_Fabbrica siano raggruppate tutte le righe con tale valore; la clausola HAVING indica che sia contato per ogni gruppo il numero di righe reperito e controlla che questo sia maggiore o uguale a 2. Dei gruppi che hanno verificato tale condizione viene visualizzato il codice fabbrica, il numero dei modelli e la somma del numero delle versioni per quella fabbrica:

Cod_Fabbrica	Numero modelli	Numero versioni
001	5	13
006	3	9

Nella clausola HAVING possono essere utilizzati gli operatori di confronto, i connettori logici e gli operatori BETWEEN, IN, LIKE. Affinché una clausola HAVING sia corretta deve essere espressa su una funzione di gruppo. Si noti che una stessa operazione di selezione può utilizzare sia la clausola WHERE sia la clausola GROUP BY. È necessario avere ben chiaro in questo caso che la WHERE opera una restrizione a priori, ossia sulle righe alle quali applicare la funzione di gruppo; la clausola HAVING esegue invece una selezione a posteriori, sulle righe risultanti dalla esecuzione della selezione. Il seguente esempio illustra meglio questo concetto.

Utilizzando la tabella Modelli, vogliamo calcolare la somma totale delle versioni prodotte da fabbriche che producono almeno un modello in 4 versioni. Una possibile soluzione è la seguente:

```
SELECT Cod_Fabbrica, SUM(Numero_Versioni) ‘Numero versioni per fabbrica'
FROM Modelli
GROUP BY Cod_Fabbrica
HAVING MAX(Numero_Versioni) >= 4
```

Il risultato del comando è il seguente:

Cod_Fabbrica	Numero versioni per fabbrica
001	13
004	4
006	9
007	4
008	4

In questo caso la clausola HAVING è basata sul fatto che la condizione che almeno un modello abbia un numero di versioni maggiore o uguale a 4 corrisponde a controllare se tale condizione è vera per il modello con il numero di versioni più alto.

La query richiede quindi solo le fabbriche per cui esiste un numero di versione maggiore o uguale a 4, ma il calcolo della somma viene eseguito su tutti i *modelli* di tali fabbriche. Ben diverso sarebbe stato esprimere la seguente operazione di selezione, apparentemente simile, ma che calcola invece la somma per *tutte le fabbriche* escludendo però dal calcolo i modelli con numero di versioni inferiore a 4:

```
SELECT Cod_Fabbrica, SUM(Numero_Versioni) 'Numero versioni per fabbrica'
FROM Modelli
WHERE Numero_Versioni >= 4
GROUP BY Cod_Fabbrica
```

Il risultato in questo caso è il seguente:

Cod_Fabbrica	Numero versioni per fabbrica
001	5
004	4
006	5
007	4
008	4

4.8 LA CLAUSOLA DI ORDINAMENTO

La clausola di ordinamento serve a dare un ordinamento al risultato di una selezione.

Se tale clausola non viene specificata l'ordine delle righe di un comando SELECT non è definito, e lo stesso comando può produrre ordinamenti diversi se eseguito in contesti diversi. Ciò rivela come la teoria relazionale derivi dalla teoria degli insiemi: un insieme non è, per definizione, ordinato.

L'ordinamento può essere effettuato in base a una o più colonne e per ciascuna colonna può essere crescente o decrescente. La clausola ha il formato:

<clausola-di-ordinamento> ::= ORDER BY <ordine> [{,<ordine> } ...]

dove

<ordine>::= <nome_colonna l <numero_colonna> [ASC | DESC]

Ricordiamo che il simbolo | nella sintassi indica un'alternativa, pertanto la colonna su cui effettuare l'ordinamento può essere indicata come <nome colonna> o come <numero colonna>. In ogni tabella le colonne sono numerate da sinistra verso destra nell'ordine in cui sono state definite, per cui per esempio la colonna 3 Veicoli è Cod_Categoria. Questo modo di fare riferimento alle colonne è comunque sconsigliato, soprattutto per motivi di leggibilità.

La specifiche ASC e DESC, anch'esse alternative tra loro, indicano se l'ordi-namento deve essere crescente, nel caso di ASC, o decrescente, nel caso di DESC; il valore di default è ASC.

Il comando seguente restituisce le righe della tabella Veicoli in cui la cilindrata risulta maggiore di 1500. La clausola ORDER BY fa sì che il risultato venga ordinato primariamente in modo decrescente in base ai cavalli fiscali, e secondariamente in modo crescente per targa:

```
SELECT *
FROM Veicoli
WHERE Cilindrata > 1500
ORDER BY Cavalli_Fiscali DESC, Targa
```

Il risultato del comando è il seguente:

Targa	Cod_Modello	Cod_Categoria	Cilindrata	Cod_Combustibile	Cavalli_Fiscali	Velocità	Posti	Immaturi colazione
F96154NH	010	01	1781	03	125	185	5	08/03/92
C78905GT	009	01	1998	01	117	212	4	06/11/94
DGH789JC	014	01	1590	02	114	170	5	05/10/95
XCH56GJK	005	01	1918	01	110	210	5	04/09/98
DH79567H	011	01	1589	04	107	170	0	
ERG567NM	013	01	1598	04	107	175	5	18/12/97
XF5789GY	008	01	1587	01	107	175	5	05/05/96
A123456X	004	01	1796	01	85	195	5	30/12/98
DD4567XX	003	01	1581	01	17		5	05/06/97

5 Join

5.1 JOIN SU DUE TABELLE

Nel capitolo precedente abbiamo visto le tecniche che consentono la selezione dei dati contenuti in un'unica tabella; facendo riferimento all'algebra relazionale, abbiamo realizzato le operazioni di

proiezione, selezione e ridenominazione; abbiamo inoltre esaminato la realizzazione delle operazioni di complemento e aggregazione.

In questo capitolo vedremo come effettuare la ricerca di informazioni mettendo in relazione i dati presenti in tabelle diverse; utilizzeremo a questo scopo la tecnica delle *join* (congiunzioni), che realizza ulteriori operazioni definite dall'algebra relazionale. Consideriamo due tabelle che abbiano colonne contenenti dati in comune. Se due tabelle soddisfano questa condizione possono essere correlate tramite una operazione di join. Spesso, come nei nostri esempi, i nomi stessi delle due colonne sono uguali, ma ciò non è strettamente necessario; le due colonne devono in ogni caso avere lo stesso significato, ovvero rappresentare lo stesso insieme di entità. In termini algebrici, diremo che i valori delle colonne delle due tabelle devono appartenere allo stesso dominio.

Questa caratteristica rappresenta una delle maggiori potenzialità dei sistemi relazionali:

i collegamenti tra i dati vengono implementati dinamicamente al momento dell'esecuzione delle query basandosi sul contenuto di colonne omogenee senza dover utilizzare dei puntatori diretti. Ciò rende sempre possibile collegare tra di loro informazioni contenute in tabelle diverse ma accomunate da un identico valore in una particolare colonna. Grazie a questa caratteristica i modelli dei dati di un sistema relazionale possono crescere al crescere delle esigenze del sistema che rappresentano, senza richiedere, la maggior parte delle volte, interventi di riprogettazione.

Un caso molto comune e significativo di colonne comuni a due tabelle è quello in cui l'appartenenza allo stesso dominio deriva dalla presenza di relazioni tra le due entità rappresentate dalle due tabelle, e quindi dalla presenza di chiavi esterne. Facendo riferimento al database Registro Automobilistico, rispondono a questa situazione le seguenti coppie di colonne:

- Cod_Categoria della tabella Categorie e Cod_Categoria della tabella Veicoli;
- Cod_Modello della tabella Modelli e Cod_Modello della tabella Veicoli; Cod_Combustibile della tabella Combustibili e Cod_Combustibili e della tabella Veicoli
- Cod_Fabbrica della tabella Fabbriche e Cod_Fabbrica della tabella Modelli;
- Targa della tabella Veicoli e Targa della tabella Proprietà;
- Cod_Proprietario della tabella Proprietari e cod_Proprietario della tabella Proprietà.

5.2 EQUI-JOIN

La forma più utile e immediata della operazione di join è la *equi-join* tra due tabelle.

Questa operazione realizza l'operazione di *giunzione naturale* definita nell'algebra relazionale: restituisce infatti una terza tabella le cui righe sono tutte e sole quelle ottenute dalle righe delle due tabelle di partenza in cui i valori delle colonne in comune sono uguali. L'operazione di equi-join è implementata in SQL come una forma particolare del comando di selezione:

- nella clausola FROM vanno indicate le due tabelle correlate su cui va effettuata la join;
- nella clausola WHERE va espresso il collegamento tra le due tabelle, mediante un'apposita condizione detta *condizione di join;*

Nella clausola WHERE e nella <lista di selezione> è possibile indicare i nomi delle colonne qualificandoli mediante il nome della tabella cui appartengono, utilizzato come prefisso nella forma. Join 89 nome_tabella.nome_colonna

Ciò spesso si rivela necessario per evitare ambiguità poiché, come abbiamo detto, in molti casi i nomi delle colonne contenenti dati comuni coincidono nelle due tabelle.

Utilizziamo ora la equi-join per visualizzare per ciascun veicolo la descrizione della relativa categoria; in questo caso l'operazione di ricerca coinvolge le due tabelle Veicoli e Categorie,

poiché l'informazione relativa al nome della categoria, Nome Categoria, non è presente nella tabella Veicoli. Le due tabelle sono correlate tra loro: ricordiamo infatti che nello schema concettuale esiste tra esse una relazione 1:N. (Si veda lo schema logico di Figura 6.1, relativo alla relazione tra le due tabelle, e l'esemplificazione della condizione di join in Figura 6.2.)

Il comando SELECT che realizza l'equi-join è il seguente.

```
SELECT Targa, Categorie.Cod_Categoria, Nome_Categoria
FROM Veicoli, Categorie
WHERE Veicoli.Cod_Categoria = Categorie.Cod_Categoria
```

Abbiamo realizzato l'operazione procedendo come segue:

- nella clausola FROM abbiamo indicato la tabella Veicoli e la tabella Categorie che dobbiamo correlare;
- nella clausola WHERE abbiamo espresso il collegamento con la condizione di join Veicoli.Cod_Categoria=Categorie.Cod_Categoria (Figura 6.2);
- nella clausola WHERE e nella <lista di selezione> abbiamo fatto precedere i nomi delle colonne Cod_Categoria di Veicoli e Cod_Categoria di Categorie dal nome della tabella di appartenenza seguito da un punto, eliminando in questo modo qualsiasi ambiguità. Si noti che le colonne relative alla targa del veicolo e al nome della categoria non sono state prefissate con il nome della tabella di appartenenza poiché, comparendo, rispettivamente solo, nella tabella Veicoli e Categorie e non in entrambe, non generano alcuna ambiguità.

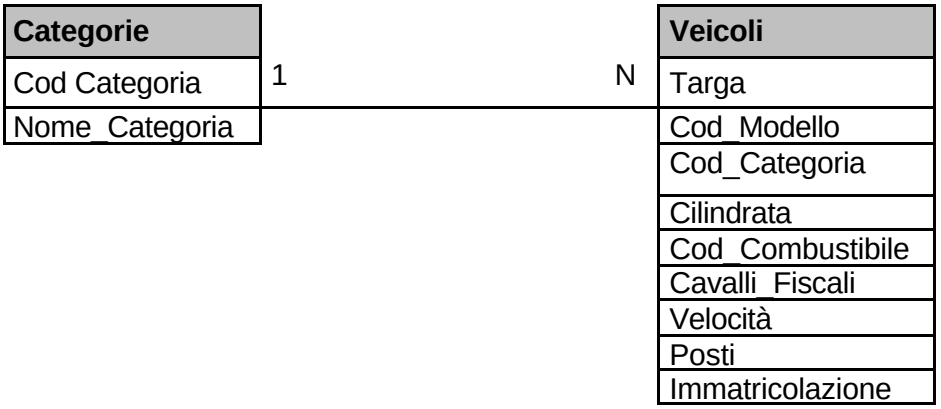

Figura 6.1 Relazione tra categorie e veicoli.

Targa	Cod_Categoria
A123456X	01
B256787Y	01
C76589AG	01
C78905GT	01
C845905Z	04
CFT346VB	01
D238765W	03
DD4567XX	01
DGH789JC	01
DH79567H	01
ERG567NM	01
F96154NH	01
FGH673FV	01
XCH56GJK	01
XF5789GY	01

Cod_Categoria	Nome_Categoria
01	Autovettura
02	Rimorchio
03	Motociclo
04	Furgone

Figura 6.2 Join tra due tabelle

Il risultato è:

Targa	Cod_Categoria	Nome_Categoria
A123456X	01	Autovettura
B256787Y	01	Autovettura
C76589AG	01	Autovettura
C78905GT	01	Autovettura
C845905Z	04	Furgone
CFT346VB	01	Autovettura
D238765W	03	Motociclo
DD4567XX	01	Autovettura
DGH789JC	01	Autovettura
DH79567H	01	Autovettura
ERG567NM	01	Autovettura
F96154NH	01	Autovettura
FGH673FV	01	Autovettura
XCH56GJK	01	Autovettura
XF5789GY	01	Autovettura

Nella clausola WHERE possono essere aggiunte altre condizioni combinandole in AND con la condizione di join per restringere l'insieme delle righe restituite, o combinandole in OR con la condizione di join per estendere l'insieme delle righe restituite.

Possono inoltre essere applicate tutte le clausole (GROUP BY, HAVING, ORDER BY) viste nel capitolo precedente.

Restringiamo l'esempio appena visto ai soli veicoli con cilindrata maggiore di 1600 e ordiniamo il risultato per valori crescenti della targa:

```
SELECT Targa, Categorie.Cod_Categoria, Nome_Categoria
FROM Veicoli, Categorie
WHERE Veicoli.Cod_Categoria = Categorie.Cod_Categoria
```

```
AND Cilindrata > 1600
ORDER BY Targa
```

Il risultato è:

Targa	Cod_Categoria	Nome_Categoria
A123456X	01	Autovettura
C78905GT	01	Autovettura
F96154NH	01	Autovettura
XCH56GJK	01	Autovettura

Ricerchiamo infine tutti i veicoli della categoria la cui descrizione è Autovettura, visualizzando tutte le colonne presenti nella tabella Categorie e nella tabella Veicoli.

Per evitare di specificare i nomi di tutte le colonne di una determinata tabella è definita la seguente notazione abbreviata:

<nome_tabella>.*

che utilizza il carattere speciale *, che avevamo già utilizzato a proposito della selezione su una sola tabella, estendendone l'uso ai casi in cui le tabelle utilizzate siano più di una. Il comando, dunque, sarà:

```
SELECT Categorie.*, Veicoli.*
FROM Categorie, Veicoli
WHERE Veicoli.Cod_Categoria = Categorie.Cod_Categoria
AND Nome_Categoria= 'Autovettura'
```

La disponibilità di operazioni per l'esecuzione di join tra tabelle è uno degli aspetti qualificanti del linguaggio SQL. Basti pensare alla complessità che la scrittura di algoritmi di questo tipo, risolti in SQL in poche righe di codice, presenterebbe in un linguaggio di programmazione tradizionale.

5.3 INNER-JOIN

Esaminiamo la *inner-join,* di cui la equi-join è un caso particolare. Per la inner-join la condizione di join non deve necessariamente essere una condizione di uguaglianza.

Cerchiamo per esempio tutti i veicoli la cui categoria ha descrizione Autovettura o camion:

```
SELECT Veicoli.*
FROM Veicoli, Categorie
WHERE Veicoli.Cod_Categoria = 'Categorie.Cod_Categoria
AND Nome_Categoria IN ('Autovettura', 'Furgone')
```

Nella condizione di join è possibile utilizzare, oltre all'operatore IN, anche gli operatori di confronto e gli operatori BETWEEN e LIKE esaminati nel capitolo precedente.

L'esempio successivo mette in evidenza quanto detto relativamente alla possibilità di effettuare join tra colonne espresse sullo stesso dominio, seppure utilizzate per rappresentare informazioni non esplicitamente correlate.

La cilindrata media nella tabella Modelli e la cilindrata nella tabella Veicoli non sono legate da relazioni rappresentate da chiavi esterne. Tali dati sono utilizzati nella seguente query, che ricerca tutti i modelli dalla cilindrata media superiore alla cilindrata del veicolo con targa C76589AG.

```
SELECT Modelli.*
FROM Modelli, Veicoli
```

```
WHERE Cilindrata_Media > Cilindrata
AND Targa = 'C76589AG'
```

Negli esempi visti finora abbiamo utilizzato la forma tradizionale di scrittura di una join mediante l'uso del comando SELECT. Nelle ultime revisioni allo standard sono stati introdotti, per specificare il tipo di join, dei costrutti *ad hoc* che però spesso non sono supportati da DBMS commerciali.

Per realizzare la inner-join, e quindi anche, come caso particolare, per la equi-join, è stato introdotto il comando INNER JOIN, il cui formato è:

<comando_INNER_JOIN> ::=

<riferimento_a_tabella> [NATURAL] INNER JOIN

<riferimento_a_tabella>

[ON <condizione_di_join>

l USING <lista_colonne_di_join>]

Le clausole ON e USING sono alternative e servono entrambe a specificare la condizione di join. Utilizzando la prima clausola la condizione deve essere espressa esplicitamente e può contenere operatori diversi da quello di uguaglianza; la seconda è una forma abbreviata che esprime una equi-join effettuata in base alla lista di colonne che seguono la parola chiave USING. L'opzione NATURAL è un'ulteriore abbreviazione che indica una equi-join effettuata in base a tutte le colonne che nelle due tabelle sono identificate dallo stesso nome.

Riprendiamo l'esempio in cui volevamo ottenere per ciascun veicolo la descrizione della relativa categoria:

```
SELECT *
FROM Veicoli, Categorie
WHERE Veicoli. Cod_Categoria = Categorie..Cod_Categoria
```

Utilizzando il comando INNER JOIN anziché il comando SELECT, possiamo scrivere:

Categorie NATURAL INNER JOIN Veicoli

oppure

Categorie INNER JOIN Veicoli

USING Cod_Categoria

oppure

Categorie INNER JOIN Veicoli

ON Categorie.Cod_Categoria = Veicoli.Cod_Categoria

La categoria sintattica <riferimento_a_tabella> include sia nomi di tabelle, come abbiamo visto negli esempi, sia comandi SELECT e ulteriori comandi di join.

Come per il comando INNER JOIN, i comandi OUTERJOIN e CROSS JOIN

che verranno descritti nei paragrafi successivi sono definiti dallo standard, ma non sempre sono presenti nelle attuali versioni dei database commerciali.

5.4 OUTER-JOIN

Quando vengono correlate mediante una join due tabelle con colonne contenenti dati in comune, è possibile che un valore presente in una delle due colonne non compaia nell'altra.

Considerando la tabella Categoria, effettuando la equi-join del primo esempio, in cui per ciascun veicolo si ricercava la corrispondente categoria, nessuna riga della tabella risultante ha Cod_Categoria uguale a 0 2, poiché per tale categoria non è presente alcun veicolo. In alcuni casi è rilevante non perdere nella join la nozione di mancata corrispondenza.

Una join che preservi questa informazione è detta *outer-join.* La outer-join può mantenere i valori per cui non esiste corrispondenza per una, per l'altra o per entrambe le tabelle; i tre casi vengono rispettivamente definiti *outer-join sinistra, outer-join destra, outer-join completa.*

Prima delle ultime revisioni del linguaggio non esisteva un metodo standard per esprimere l'outer-join, sebbene tutti i principali DBMS avessero implementato uno specifico costrutto. Attualmente lo standard prevede di esprimere la outer-join nel modo seguente:

<riferimento_a_tabella> [NATURAL] {LEFT l RIGHT I FULL} [OUTER] JOIN <riferimento_a_tabella>

[ON <condizione_di_join>

l USING <lista_colonne_di_join>]

Nella outer-join sinistra vengono preservati tutti i valori della tabella che viene specificata come primo operando; nella outer-join destra vengono preservati tutti i valori della tabella che viene specificata come secondo operando. L'opzione NATURAL e le clausole ON e USING hanno lo stesso significato descritto nel comando INNER JOIN.

Per ottenere tutti i codici delle categorie, compresi quelli per cui non esiste alcun veicolo, sarà necessario scrivere

Categorie LEFT OUTER JOIN Veicoli

ON Categorie.Cod_Categoria Veicoli.Cod_Categoria

oppure

Categorie LEFT JOIN Veicoli

ON Categorie.Cod_Categoria Veicoli.Cod_Veicolo

oppure

Veicoli RIGHT JOIN Categorie

ON Categorie.Cod_Categoria = Veicoli.Cod_Categoria

Il risultato è riportato nella Figura 6.4.

Se nella tabella Veicoli fossero presenti veicoli con categoria non definita, in maniera del tutto analoga sarebbe possibile includerli nel risultato della join.

Per ottenere invece tutti i casi di mancata corrispondenza, sia della tabella Categorie sia della tabella Veicoli, scriveremo

Categorie FULL OUTER JOIN Veicoli

ON Categorie.Cod_Categoria = Veicoli.Cod_Categoria

oppure

Categorie FULL JOIN Veicoli

ON Categorie.Cod_Categoria = Veicoli.Cod_Categoria

oppure

Veicoli FULL JOIN Categorie

ON Categorie.Cod_Categoria = Veicoli.Cod_Categoria

5.5 CROSS-JOIN

L'operazione di ***prodotto*** tra due tabelle, definita nell'algebra relazionale. si ottiene mediante una operazione di join espressa in maniera tradizionale in cui non venga specificata la condizione di join. Se le due tabelle sono composte rispettivamente da n e da m righe, la nuova tabella conterrà n x m righe, ottenute accodando ciascuna riga della prima tabella con ciascuna riga della seconda, come in

```
SELECT Categorie.*, Fabbriche.*
FROM Categorie, Fabbriche.
```

Lo stesso risultato si ottiene utilizzando il comando INNER JOIN senza specificare né l'opzione NATURAL, né le clausole ON o USING.

Questo caso è previsto esplicitamente dalle nuove revisioni allo standard; è stato infatti definito il comando CROSS JOIN con la seguente sintassi:

<riferimento_a_tabella> CROSS JOIN

<riferimento_a_tabella>

Il seguente comando genera lo stesso risultato dell'esempio precedente:

Categorie CROSS JOIN Fabbriche

5.6 JOIN SU PIÙ DI DUE TABELLE

Talvolta un'interrogazione può dover coinvolgere più di due tabelle. Supponiamo per esempio di voler reperire il nome della fabbrica del modello con targa A123456X. Come possiamo dedurre dall'esame dello schema logico riportato in Figura 6.6, nell'operazione sono coinvolte le tre tabelle Veicoli, Modelli e Fabbriche

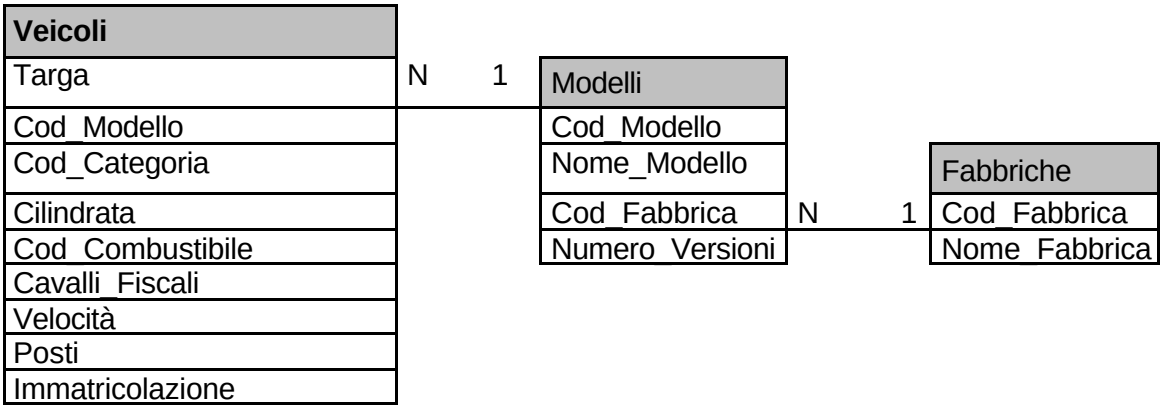

Figura 6.6 Relazioni tra le tabelle Veicoli, Modelli e Fabbriche

Utilizzando le nozioni finora apprese avremmo dovuto procedere nel modo seguente.

1. Ricavare nella tabella Modelli il codice della fabbrica Cod_Fabbrica corrispondente al codice modello Cod Modello del veicolo con Targa A123456X. Dobbiamo dunque effettuare una equi-join tra le tabelle Veicoli e Modelli:

```
SELECT Cod_Fabbrica
FROM Veicoli, Modelli
WHERE Targa = 'A123456X'
AND Veicoli.Cod_Modello = Modelli.Cod_Modello
```

e annotare il risultato:

Cod_Fabbrica
003

2. Per il codice Cod_Fabbrica così ottenuto trovare, nella tabella Fabbriche, il relativo nome:

```
SELECT Nome_Fabbrica
FROM Fabbriche
WHERE Cod_Fabbrica = '003'
```

Il risultato è:

Nome_Fabbrica
Ford

Per abbreviare questo procedimento il linguaggio permette di effettuare la join tra più di due tabelle. Le regole da rispettare sono le seguenti:

- nella clausola FROM vanno elencate tutte le tabelle coinvolte;
- la condizione di join deve contenere le informazioni necessarie a correlare tutte le tabelle coinvolte; se le tabelle sono n, sono necessarie almeno n-1 condizioni (Figura 6.7).

Nella lista di selezione non devono necessariamente essere incluse colonne di tutte le tabelle coinvolte; in altri termini alcune delle tabelle della clausola FROM possono avere l'unico ruolo di essere utilizzate nella condizione di join e in altre condizioni della clausola WHERE. Con questa tecnica l'esempio precedente può essere tradotto in un unico comando SELECT:

```
SELECT Nome_Fabbrica
FROM Veicoli, Modelli, Fabbriche
WHERE Veicoli.Cod_Modello = Modelli.Cod_Modello
AND Modelli.Cod_Fabbrica : Fabbriche.Cod_Fabbrica
AND Targa = 'A123456X'
```

La clausola WHERE esprime il legame mostrato in Figura 6.7, vincolando inoltre Targa a A123456X.

La join di tre tabelle è rilevante perché viene spesso utilizzata nel caso di schemi che traducono relazioni N:N; per esempio, in

```
SELECT Cognome, Nome
FROM Veicoli, Proprietà, Proprietari
WHERE Veicoli.Targa = Proprietà.Targa
AND Proprietà.Cod_Proprietario = Proprietari.
Cod_Proprietario
AND Targa 'A123456X'
```

due tabelle, Veicoli e Proprietari, rappresentano gli insiemi di entità e la terza, Proprietà, la relazione N:N tra esse. Il linguaggio SQL permette join tra un numero arbitrario di tabelle, anche se ragioni dovute alla necessità di ottenere dei tempi di risposta accettabili inducono normalmente a porsi delle limitazioni.

5.7 SELF-JOIN

Un caso molto particolare di join è quello che utilizza una singola tabella, correlandola con se stessa; si ha in questo caso una *self-join.* Una possibile situazione di questo genere è quella in cui si vogliono elencare tutte le coppie di veicoli che hanno lo stesso modello. La tecnica utilizzata in tale caso è la seguente: si suppone. anziché di avere una sola tabella dei veicoli, di averne due copie assolutamente identiche, diciamo V1 e V2; in questo caso la join può essere facilmente ricondotta a una equi-join che fa corrispondere a tutti i veicoli della prima tabella quelli della seconda con lo stesso modello.

Per mettere in pratica quanto detto è necessario utilizzare gli *alias,* ovvero la ridenominazione di tabelle; questa consente di definire, nella clausola FROM, dei nomi alternativi per le tabelle, che possono essere utilizzati sia nella lista di selezione sia nelle clausole successive (WHERE, ORDER BY ecc.). La sintassi è la seguente:

<nome_tabella> <nome_alias>

Per ridenominare in un comando SELECT la tabella Veicoli con l'alias V1 scriveremo nella clausola FROM:

```
FROM Veicoli V1
```

In prima approssimazione la soluzione al problema proposto è la seguente:

```
SELECT V1.Targa, V1.Cod_Modello, V2.Targa, V2.Cod_Modello
FROM Veicoli V1, Veicoli V2
WHERE V1.Cod_Modello = V2.Cod_Modello
```

Il risultato ottenuto può apparire in prima analisi scorretto, producendo una riga anche per quei veicoli che compaiono con un unico modello. Ricordando però di aver lavorato su due copie identiche della tabella dei veicoli, il risultato è corretto.

Infatti per esempio il modello 004 presente in un veicolo della prima tabella è presente nello stesso veicolo della seconda tabella. Inoltre le coppie di veicoli che hanno realmente lo stesso modello compaiono due volte, poiché se è vero che il modello del veicolo X è uguale a quello del veicolo Y, è anche vero il viceversa. Occorre quindi apportare delle correzioni al fine di eliminare le coppie in cui entrambi gli elementi sono uguali e quelle duplicate, ovvero quelle in cui gli elementi sono uguali a quelli di un'altra coppia ma in ordine diverso. Questo si ottiene ponendo la condizione V1.Targa > V2.Targa, poiché in questo modo i due codici saranno certamente diversi e il primo sarà maggiore del secondo, escludendo così necessariamente la seconda coppia con gli stessi elementi. Scriveremo pertanto:

```
SELECT V1.Targa, V1.Cod_Modello, V2.Targa,
V1.Cod_Modello
FROM Veicoli V1, Veicoli V2
WHERE V1.Cod_Modello = V2.Cod_Modello AND
V1.Targa > V2.Targa
```

Il risultato è:

V1.Targa	V1.Cod_Modello	V2.Targa	V2.Cod_Modello
C765589AG	001	B256787Y	001

Sebbene la ridenominazione di tabelle sia indispensabile solo nelle self-join. È possibile utilizzare tale tecnica anche nelle operazioni di join viste in precedenza.

conseguendo il risultato di abbreviare la scrittura delle operazioni SELECT in cui i nomi delle tabelle siano lunghi e di ottenere un codice più leggibile.

6 Operatori su insiemi

6.1 INTRODUZIONE

Riferendoci all'algebra relazionale, abbiamo visto finora come vengono realizzate in SQL le operazioni di proiezione, selezione, prodotto, ridenominazione, giunzione. Di seguito riassumiamo le tecniche principali rispettivamente utilizzate.

Proiezione tramite la specifica di colonne nella lista di selezione nel comando SELECT

Selezione tramite la clausola WHERE nel comando SELECT

Prodotto tramite la cross-join

Ridenominazione tramite ridenominazione di colonne nella lista di selezione

Giunzione tramite la equi-join

Verranno nel seguito descritti tre operatori che realizzano altre operazioni ovvero *l'unione,* la *differenza e l'intersezione;* tali operatori, derivati direttamente dalla teoria degli insiemi, lavorano sulle tabelle trattandole come insiemi di righe.

La maggior parte degli attuali DBMS implementano in maniera conforme allo standard solo l'operatore di unione; spesso non implementano, o almeno non in maniera standard, gli operatori di differenza e intersezione: tali operatori sono stati introdotti solo nella revisione allo standard del 1992.

6.2 UNION

L'operatore UNION, che realizza l'operazione di *unione* definita nell'algebra relazionale, utilizza come operandi due tabelle risultanti da comandi SELECT e restituisce come risultati una terza tabella che contiene tutte le righe della prima tabella e tutte quelle della seconda.

Il formato del comando è

<comando-select> UNION [ALL]

<comando-select>

Le tabelle utilizzate come operandi possono essere restituite da comandi SELECT

su singole tabelle o da join; i due operandi devono comunque avere la stessa struttura.

Per elencare tutti i codici dei modelli dei veicoli con cilindrata minore di 1400:

Cod.Modello
001
007
012

e i codici dei modelli con codice fabbrica uguale a 001 scriveremo:

```
SELECT Cod_Modello
FROM Veicoli
WHERE Cilindrata<1400
```

```
UNION
SELECT Cod_Modello
FROM Modelli
WHERE Cod_Fabbrica='001'
```

e otterremo:

Cod.Modello
001
003
006
007
009
012

Se non viene specificata alcuna opzione, l'operatore UNION funziona come un operatore su insiemi "puro": infatti restituisce un'unica istanza per ciascuna riga, eliminando le eventuali righe duplicate. Qualora le ripetizioni fossero rilevanti, le righe duplicate possono essere incluse utilizzando l'opzione ALL. Il comando

```
SELECT Cod_Modello
FROM Veicoli
WHERE Cilindrata<1400
UNION ALL
SELECT Cod_Modello
FROM Modelli
WHERE Cod_Fabbrica='001'
```

restituisce la seguente tabella:

Cod.Modello
001
007
012
001
003
006
009
012

Si noti che in questo caso la situazione è opposta a quella che abbiamo già visto nella sintassi del comando SELECT con le specifiche [ALL | DISTINCT]: in quel caso il default è ALL. mentre in questo caso il default è un non specificato DISTINCT.

L'operatore UNION è molto utile quando sia necessario trattare come unitario il risultato di due selezioni su tabelle distinte ma con la stessa struttura. Sebbene la teoria relazionale preveda la rappresentazione di un tipo di entità mediante una singola tabella, situazioni di questo genere compaiono in realtà abbastanza spesso per considerazioni dovute a motivi di ottimizzazione. Per esempio, si considerino i movimenti contabili di un'azienda di grandi dimensioni, che possono risultare in una tabella contenente centinaia di migliaia o milioni di righe. Per effettuare ricerche in

tempi accettabili, sarà frequente trovare una struttura di questo tipo suddivisa in più tabelle in base a specifici criteri, per esempio con la limitazione di ogni tabella ai movimenti contabili di un singolo anno.

6.3 EXCEPT

L'operatore EXCEPT utilizza come operandi due tabelle risultanti da comandi SELECT e restituisce come risultato una terza tabella che contiene tutte le righe della prima tabella che non si trovano nella seconda tabella. Tale operazione realizza l'operazione di ***differenza*** di due tabelle dell'algebra relazionale. Le due tabelle utilizzate come operandi devono avere la stessa struttura. Il formato del comando è

<comando-select> EXCEPT[ALL]

<comando-select>

Per elencare tutti i codici dei modelli dei veicoli con cilindrata inferiore a 1400:

Cod.Modello
001
007
012

esclusi i codici dei modelli con codice fabbrica uguale a 001:

Cod.Modello
001
003
006
009
012

scriveremo:

```
SELECT Cod_Modello
FROM Veicoli
WHERE Cilindrata<1400
EXCEPT
SELECT Cod_Modello
FROM Modelli
WHERE Cod_Fabbrica='001'
```

e otterremo:

Cod.Modello
007

Anche l'operatore EXCEPT, come UNION, se non viene specificata alcuna opzione, funziona come un operatore su insiemi puro. L'opzione ALL ha in questo caso il seguente significato: si

supponga di avere due tabelle T1 e T2 che abbiano una riga in comune, e che tale riga compaia n volte nella tabella T1 e m volte nella tabella T2; nel risultato di

```
T1 EXCEPT ALL T2
```

la riga comune apparirà n-m volte.

6.4 INTERSECT

L'operatore INTERSECT utilizza come operandi due tabelle risultanti da comandi SELECT e restituisce come risultato una terza tabella che contiene tutte le righe che compaiono sia nella prima tabella sia nella seconda. Tale operazione realizza l'operazione di *intersezione* di due tabelle dell'algebra relazionale. Le due tabelle utilizzate come operandi devono avere la stessa struttura. Il formato del comando è <comando-select> INTERSECT [ALL]

<comando-select>

Per eseguire l'intersezione della tabella che contiene i codici dei modelli dei veicoli con cilindrata inferiore a 1400:

Cod.Modello
001
007
012

con quella contenente i codici dei modelli con codice fabbrica uguale a 001:

Cod_Modello
001
003
006
009
012

scriveremo:

```
SELECT Cod_Modello
FROM Veicoli
WHERE Cilindrata<1400
INTERSECT
SELECT Cod_Modello
FROM Modelli
WHERE Cod_Fabbrica='001'
```

e otterremo:

Anche l'operatore INTERSECT, come UNION ed EXCEPT, se non viene specificata alcuna opzione, funziona come un operatore su insiemi puro. L'opzione ALL ha anche in questo caso l'effetto di non eliminare le righe duplicate.

7 Linguaggio per la manipolazione dei dati

I comandi di interrogazione e di manipolazione dei dati costituiscono la parte del linguaggio SQL che realizza le funzioni di DML(Data Manipulation Language). tutti i comandi fin qui analizzati erano finalizzati allo svolgimento di operazioni di ricerca, dalle più semplici alle più complesse su dati già esistenti e tabelle già definite.

I comandi che vedremo consentono di inserire nuovi dati in una tabella precedentemente definita, di modificare i dati contenuti in una nuova tabella e di effettuarne la cancellazione.

7.1 ISTRUZIONE INSERT

L'istruzione INSERT serve per aggiungere nuove righe in una tabella o in una vista basata su tabelle. L'istruzione è:

```
INSERT INTO {nome_tabella | nome_vista}
[(lista_colonne)]
{ VALUES (lista_valori) | subquery}
```

- È necessario mettere il nome della tabella od il nome della vista in cui fare l'inserimento;
- *lista_colonne* è la lista delle colonne in cui dovranno essere inseriti i nuovi valori, è importante l'ordine; se non viene messa tale lista, i valori inseriti vengono messi nelle colonne nell'ordine in cui queste ultime sono state definite durante la creazione della tabella;
- i valori inseriti possono essere elencati esplicitamente nella *lista_valori* o attraverso una subquery.

Esempio

```
INSERT INTO anagrafica
VALUES ('Fontana', 'Giorgio', 'XYZ826K', null, null, 'taichi')
```

In questo modo si inserisce nella tabella anagrafica una nuova riga attribuendo ai campi, nell'ordine in cui sono stati definiti, i valori elencati nella clausola VALUE.

7.2 ISTRUZIONE UPDATE

L'istruzione UPDATE serve per cambiare valori esistenti in una tabella o in una vista.

```
UPDATE {nome_tabella | nome_vista}
SET
   nome_colonna = {espressione | subquery}
   [, nome_colonna = {espressione | subquery}]
   [, ...]
[WHERE condizione]
```

- È necessario mettere il nome della tabella od il nome della vista in cui aggiornare i dati;
- *nome_colonna* è il nome della colonna a cui si associa un nuovo valore ottenuto come risultato di un'espressione o di una subquery posta a destra del segno di uguaglianza;

- è possibile mettere condizioni sulle tuple da aggiornare usando la clausola WHERE.

Esempio.

```
UPDATE anagrafica
SET
  età=25,
  sport='pattinaggio'
WHERE cognome='Feltri' and nome='Daniela'
```

Tale istruzione consente di modificare informazioni già presenti nella tabella anagrafica; in particolare, le tuple che soddisfano la condizione vengono aggiornate ponendo 25 come età e pattinaggio come sport.

7.3 ISTRUZIONE DELETE

L'istruzione DELETE serve per cancellare righe da una tabella o da una vista.

```
DELETE FROM {nome_tabella | nome_vista}
WHERE condizione]
```

- È necessario mettere il nome di una tabella od il nome di una vista;
- si usa la clausola WHERE se si vogliono cancellare solo righe che soddisfano certe condizioni.

Esempio

```
DELETE FROM anagrafica
WHERE cognome='Feltri'
```

Con tale istruzione si eliminano dalla tabella anagrafica tutte le tuple che hanno 'Feltri' come valore del campo cognome.