

BASI DI DATI

Riassunto del libro “Basi di dati – Modelli e linguaggi di interrogazione”, terza ed., Atzeni, ceri, Paraboschi, Torione e riassunto slide del corso

Jacopo De Angelis

SOMMARIO

Programma del corso	7
1 Introduzione	8
1.1 Sistemi informativi, informazioni e dati	8
1.2 Basi di dati e sistemi di gestione di basi di dati	8
1.3 Modelli di dati.....	8
1.3.1 Schemi e istanze	9
1.3.2 Livelli di astrazione nei DBMS.....	9
1.3.3 Indipendenza dei dati	10
1.4 Linguaggi e utenti delle basi di dati	10
1.4.1 Linguaggi per basi di dati	10
1.4.2 Utenti e progettisti	10
1.5 Vantaggi e svantaggi dei DBMS	10
2 Il modello relazionale	12
2.1 Il modello relazionale: strutture.....	12
2.1.1 Modelli logici nei sistemi di base di dati.....	12
2.1.2 Relazioni e tabelle.....	12
2.1.3 Relazioni con attributi.....	12
2.1.4 Relazioni e basi di dati	13
2.1.5 Informazione incompleta e valori nulli.....	13
2.2 Vincolo di integrità	14
2.2.1 Chiavi	14
2.2.2 Chiavi e valori nulli.....	14
2.2.3 Vincoli di integrità referenziale	14
3 Algebra e calcolo relazionale	15
3.1 Algebra relazionale	15
3.1.1 Unione, intersezione, differenza	15
3.1.2 Ridenominazione	15
3.1.3 Selezione.....	15
3.1.4 Proiezione	16
3.1.5 Join.....	16
3.1.5.1 <i>Join naturale</i> (\bowtie).....	16
3.1.5.2 <i>Join completi e incompleti</i>	17
3.1.5.3 <i>Join esterni</i>	17
3.1.5.4 <i>Theta-join ed equi-join</i>	17

3.1.6	Interrogazioni in algebra relazionale	17
3.1.7	Algebra con valori nulli	18
3.1.8	Viste	18
3.2	Calcolo relazionale	18
3.2.1	Calcolo relazionale su domini	18
3.2.2	pregi e difetti del calcolo su domini	19
3.2.3	Calcolo su tuple con dichiarazioni di range	19
3.3	Datalog	20
4	SQL: concetti base	21
4.1	Definizione dei dati in SQL	21
4.1.1	I domini elementari	21
4.1.2	Definizione di schema	22
4.1.3	Definizione delle tabelle	22
4.1.4	Definizione dei domini	22
4.1.5	Specifiche di valori di default	23
4.1.6	Vincoli intrarelazionali	23
4.1.6.1	<i>Not null</i>	23
4.1.6.2	<i>Unique</i>	23
4.1.6.3	<i>Primary key</i>	24
4.1.7	Vincoli interrelazionali	24
4.1.8	Modifica degli schemi	25
4.1.8.1	<i>Alter</i>	25
4.1.8.2	<i>Drop</i>	25
4.1.9	Cataloghi relazionali	26
4.2	Interrogazioni in SQL	26
4.2.1	Dichiaratività di SQL	26
4.2.2	Interrogazioni semplici	26
4.2.2.1	<i>Clausola select</i>	26
4.2.2.2	<i>Clausola from</i>	27
4.2.2.3	<i>Clausola where</i>	27
4.2.2.4	<i>Gestione dei valori nulli</i>	27
4.2.2.5	<i>Interpretazione formale delle interrogazioni SQL</i>	27
4.2.2.6	<i>Duplicati</i>	27
4.2.2.7	<i>Join interni ed esterni</i>	27
4.2.2.8	<i>Ordinamento</i>	28
4.2.3	Operatori aggregati	28
4.2.4	Interrogazioni con raggruppamento	29

4.2.4.1	<i>Predicati sui gruppi</i>	29
4.2.5	Interrogazioni di tipo insiemistico	30
4.2.6	Interrogazioni nidificate	30
4.2.6.1	<i>Interrogazioni nidificate complesse</i>	30
4.3	Modifica dei dati in SQL.....	30
4.3.1	Inserimento	30
4.3.2	Cancellazione.....	30
4.3.3	Modifica.....	31
5	SQL: caratteristiche evolute.....	32
5.1	Caratteristiche evolute di definizione dei dati	32
5.1.1	Vincoli di integrità generici.....	32
5.1.2	Asserzioni.....	32
5.2	Funzioni scalari	33
5.2.1	Famiglie di funzioni.....	33
5.2.2	Funzioni condizionali	33
5.2.2.1	<i>Coalesce</i>	33
5.2.2.2	<i>Nullif</i>	33
5.2.2.3	<i>Case</i>	33
5.3	Controllo dell'accesso.....	34
5.3.1	Risorse e privilegi.....	34
5.3.2	Comandi per concedere e revocare privilegi.....	34
5.4	Transazioni.....	35
5.4.1	Atomicità	35
5.4.2	Consistenza	36
5.4.3	Isolamento	36
5.4.4	Persistenza.....	36
6	SQL per le applicazioni.....	37
6.1	procedure	37
6.2	Trigger.....	37
6.3	SQL embedded	38
6.3.1	Cursori.....	39
6.3.2	SQL dinamico	40
6.3.2.1	<i>Esecuzione immediata</i>	40
6.3.2.2	<i>Fase di preparazione</i>	41
6.3.2.3	<i>Fase di esecuzione</i>	41
6.3.2.4	<i>Cursori con SQL dinamico</i>	41
6.4	Call Level Interface (CLI)	41

6.4.1	ODBC e soluzioni proprietarie Microsoft.....	42
6.4.1.1	<i>ODBC</i>	42
6.4.1.2	<i>OLE DB</i>	43
6.4.1.3	<i>ADO.NET</i>	43
6.4.2	Java Database Connectivity	43
7	Metodologie e modelli per il progetto	45
7.1	Introduzione alla progettazione	45
7.1.1	Il ciclo di vita dei sistemi informativi	45
7.1.2	Metodologie di progettazioni e basi di dati	45
7.2	Il modello entità-relazione (E-R).....	46
7.2.1	I costrutti principali del modello.....	46
7.2.1.1	<i>Entità</i>	46
7.2.1.2	<i>Relazioni (o associazioni)</i>	47
7.2.1.3	<i>Attributi</i>	47
7.2.1.4	<i>Costruzione di schemi con i costruttori di base</i>	48
7.2.2	Altri costrutti del modello	48
7.2.2.1	<i>Cardinalità delle relazioni</i>	48
7.2.2.2	<i>Cardinalità degli attributi</i>	49
7.2.2.3	<i>Identificatori delle entità</i>	49
7.2.2.4	<i>Generalizzazioni</i>	49
7.2.3	Panoramica finale sul modello E-R	51
7.3	Documentazioni di schemi E-R	53
7.3.1	Regole aziendali.....	53
7.3.2	Tecniche di documentazione	54
7.4	Modellazione dei dati in UML.....	54
7.4.1	Rappresentazione di dati con i diagrammi delle classi.....	55
7.4.1.1	<i>Classi</i>	55
7.4.1.2	<i>Associazioni</i>	55
7.4.1.3	<i>Molteplicità</i>	55
7.4.1.4	<i>Identificatori</i>	56
7.4.1.5	<i>Generalizzazioni</i>	56
7.4.1.6	<i>Note</i>	56
8	La progettazione concettuale.....	57
8.1	La raccolta e l'analisi dei requisiti.....	57
8.2	Rappresentazione concettuale di dati.....	59
8.2.1	Criteri generali di rappresentazione	59
8.2.2	Pattern di progetto	59

8.3	Strategie di progetto	62
8.3.1	Strategia top-down.....	62
8.3.2	Strategia bottom-up	63
8.3.3	Strategia inside-out	63
8.3.4	Strategia mista.....	64
8.4	Qualità di uno schema concettuale	64
8.4.1	Correttezza	64
8.4.2	Completezza	64
8.4.3	Leggibilità.....	64
8.4.4	Minimalità.....	64
8.5	Una metodologia generale	65
8.6	Un esempio di progettazione concettuale	66
9	La progettazione logica.....	68
9.1	Fasi della progettazione logica	68
9.2	Analisi delle prestazioni su schemi E-R.....	68
9.3	Ristrutturazione di schemi E-R	70
9.3.1	Analisi delle ridondanze.....	70
9.3.2	Eliminazione delle generalizzazioni	72
9.3.3	Partizionamento/accorpamento di concetti	73
9.3.3.1	<i>Partizionamento di entità</i>	73
9.3.3.2	<i>Eliminazione di attributi multivалore.....</i>	74
9.3.3.3	<i>Accorpamento di entità.....</i>	74
9.3.3.4	<i>Altri tipi di partizionamento/accorpamento</i>	74
9.3.4	Scelta degli identificatori principali	74
9.4	Traduzione verso il modello relazionale.....	75
9.4.1	Entità e associazioni molti a molti.....	76
9.4.2	Associazioni uno a molti	76
9.4.3	Entità con identificatore esterno.....	77
9.4.4	Associazioni uno a uno	77
9.4.5	Tabelle riassuntive	78
9.4.6	Documentazioni di schemi logici	79
9.5	Un esempio di progettazione logica	80
9.5.1	Fase di ristrutturazione.....	81
9.5.1.1	<i>Analisi delle ridondanze</i>	81
9.5.1.2	<i>Eliminazione delle gerarchie</i>	81
9.5.1.3	<i>Partizionamento/accorpamento di concetti</i>	82
9.5.1.4	<i>Scelta degli identificatori principali</i>	82

9.5.2	Traduzione verso il relazionale	83
-------	---------------------------------------	----

PROGRAMMA DEL CORSO

- 1) Introduzione (1° parziale)
 - a. Concetti generali
 - b. Concetto di informazione e dato
 - c. Introduzione a basi di dati e DBMS, modello dei dati, concetto di schema e istanza
 - d. Indipendenza logica e fisica dei dati e tipologia di linguaggi per basi di dati.
- 2) Il modello relazionale (1° parziale)
 - a. relazioni e tabelle
 - b. schemi ed istanze
 - c. informazione incompleta e valori nulli
 - d. chiavi
 - e. vincoli di integrità.
- 3) Linguaggi di interrogazione (2° parziale)
 - a. Algebra Relazionale
 - b. Select, Project, Natural Join, Prodotto Cartesiano, Theta-Join
- 4) SQL (2° parziale)
 - a. Definizione dei dati in SQL
 - b. Definizione di interrogazioni in SQL: interrogazioni semplici, con operatori insiemistici, nidificate e con raggruppamento
 - c. Operazioni di inserimento, modifica e cancellazione
 - d. Definizione di viste.
- 5) Il modello Entità-Relazione (E-R) esteso con generalizzazioni (1° parziale)
 - a. Metodologie di progettazione di basi di dati
 - b. La progettazione concettuale.
- 6) La progettazione logica (2° parziale)
 - a. Ristrutturazione e ottimizzazione di schemi E-R: eliminazione delle gerarchie, degli attributi composti e multivaleore
 - b. Traduzione da schemi E-R a schemi relazionali

1 INTRODUZIONE

1.1 SISTEMI INFORMATIVI, INFORMAZIONI E DATI

Ogni organizzazione è dotata di un sistema informativo, che organizza e gestisce le informazioni necessarie per perseguire gli scopi dell'organizzazione stessa.

L'esistenza del sistema informativo è in parte indipendente dalla sua automatizzazione.

Le informazioni vengono rappresentate per mezzo di dati, che hanno bisogno di essere interpretati per fornire informazioni.

Ricordiamo che i dati non hanno alcun significato fino a quando non vengono interpretati e correlati opportunamente, a quel punto forniscono informazioni.

Introdotto il concetto di dato, possiamo passare a quello di base di dati. Secondo l'accezione più generale, una base di dati è una collezione di dati utilizzati per rappresentare le informazioni di interesse per un sistema informativo.

1.2 BASI DI DATI E SISTEMI DI GESTIONE DI BASI DI DATI

L'approccio "convenzionale" alla gestione dei dati sfrutta la presenza di archivi o file per memorizzare e ricercare dati, ma fornisce solo semplici meccanismi di accesso e condivisione.

Un sistema di gestione di basi di dati (Data Base Management System, DBMS) è un sistema software in grado di gestire collezioni di dati che siano grandi, condivise e persistenti, assicurando la loro affidabilità e privatezza. Un DBMS deve essere efficiente ed efficace. Una base di dati è una collezione di dati gestita da un DBMS.

Le basi di dati sono:

- **Grandi:** possono assumere dimensioni di varia entità, quindi devono presupporre l'utilizzo di una memoria secondaria
- **Condivise:** diversi applicazioni e utenti devono poter accedere a dati comuni. È importante notare che in questo modo si riduce la ridondanza dei dati, poiché si evitano le ripetizioni, e si riduce anche la possibilità di inconsistenze
- **Persistenti:** la loro vita non è limitata alla singola esecuzione
- **Affidabili:** capacità del sistema di conservare intatto il contenuto della base di dati
- **Privatezza:** ciascun utente viene abilitato a svolgere determinate azioni sui dati in base a meccanismi di autorizzazione
- **Efficienti:** sono capaci di svolgere le operazioni utilizzando un insieme di risorse che sia accettabile per gli utenti
- **Efficaci:** sono capaci di rendere produttive le attività dei loro utenti

1.3 MODELLI DI DATI

Un modello di dati è un insieme di concetti utilizzati per organizzare i dati di interesse e descriverne la struttura in modo che essa risulti comprensibile a un elaboratore. Ogni modello di dati fornisce meccanismi di strutturazione che permettono di definire nuovi tipi sulla base di tipi predefiniti e costruttori di tipo.

Il modello relazionale dei dati, attualmente il più diffuso, permette di definire tipi per mezzo del costruttore relazione, che consente di organizzare i dati in insiemi di record a struttura fissa.

Una relazione viene spesso rappresentata per mezzo di una tabella le cui righe rappresentano i singoli record e le colonne i campi del record. Qua di seguito un esempio di base di dati relazionale.

Docenza	
Corso	NomeDocente
Basi di dati	Rossi
Reti	Neri
Linguaggi	Verdi

Manifesto		
CdL	Materia	Anno
Inflinf	Basi di dati	2
Inflinf	Reti	3
Inflinf	Linguaggi	2
IngEl	Basi di dati	3
IngEl	Reti	3

Oltre al modello relazionale sono stati definiti altri quattro tipi di modelli:

- **Modello gerarchico:** basato sull'uso di strutture ad albero
- **Modello reticolare:** basato sull'uso di grafi
- **Modello a oggetti:** estende alle basi di dati il paradigma di programmazione a oggetti
- **Modello XML:** rivisitazione del modello gerarchico, i dati vengono rappresentati assieme alla loro descrizione e non devono sottostare rigidamente a un'unica struttura logica. È autodescrittivo e semi strutturato.

Questi modelli vengono detti “logici”, questo per sottolineare il fatto che le strutture utilizzate da questi modelli, pur essendo astratte, riflettono una particolare organizzazione.

Più recentemente sono stati introdotti i modelli detti “concettuali”, utilizzati per descrivere i dati in maniera completamente indipendente dalla scelta del modello logico.

1.3.1 Schemi e istanze

Nelle basi di dati esiste una parte detta “schema della base di dati”, costituita dalle caratteristiche dei dati, e una parte detta “istanza” o “stato”, costituita dai valori effettivi.

Lo schema di una relazione è costituito dalla sua intestazione, cioè dal nome della relazione seguito dai nomi dei suoi attributi, ad esempio “docenza(Corso, NomeDocente)”.

Viceversa, le righe della tabella variano nel tempo. Quindi le istanze non sono fisse ma sono soggette a variazioni.

Si dice che lo schema è la componente intensionale della base di dati e l'istanza la componente estensionale.

1.3.2 Livelli di astrazione nei DBMS

Esiste una proposta di **architettura standardizzata per DBMS articolata su tre livelli**, detti rispettivamente esterno, logico e interno:

- **Schema logico:** descrizione dell'intera base di dati per mezzo del modello logico adottato dal DBMS
- **Schema interno:** costituisce la rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione
- **Schema esterno:** costituisce la descrizione di una porzione della base di dati di interesse per mezzo del modello logico.

1.3.3 Indipendenza dei dati

L'architettura così descritta garantisce l'**indipendenza dei dati**, la principale proprietà dei DBMS, ovvero ciò che permette a utenti e programmi applicativi che utilizzano una base di dati di interagire a un elevato livello di astrazione, che prescinde dai dettagli realizzativi.

Vi sono **due possibilità di indipendenza**:

- **Indipendenza fisica**: consente di interagire con il DBMS in modo indipendente dalla struttura fisica dei dati. In base a questa proprietà è possibile modificare le strutture fisiche senza influire sulle descrizioni
- **Indipendenza logica**: consente di interagire con il livello esterno della base di dati in modo indipendente dal livello logico. Per esempio, è possibile aggiungere uno schema esterno in base alle esigenze di un nuovo utente oppure modificare uno schema esterno senza dover modificare lo schema logico e, quindi, l'organizzazione dei dati.

Gli accessi alla base di dati avvengono solo attraverso il livello esterno (che può coincidere con quello logico).

1.4 LINGUAGGI E UTENTI DELLE BASI DI DATI

1.4.1 Linguaggi per basi di dati

I linguaggi per le basi di dati si distinguono in due categorie:

- **Linguaggi di definizione dei dati (DDL)**: utilizzati per definire gli schemi logici, esterni e fisici e le autorizzazioni per l'accesso
- **Linguaggi di manipolazione dei dati (DML)**: utilizzati per l'interrogazione e l'aggiornamento delle istanze di basi di dati

1.4.2 Utenti e progettisti

Varie categorie di persone possono interagire con una base di dati:

- **Amministratore della base di dati (DBA)**: è la persona responsabile della progettazione, controllo e amministrazione.
- **Progettisti e programmatore di applicazioni**: definiscono e realizzano i programmi che accedono alla base di dati.
- **Utenti**: utilizzano la base di dati per le proprie attività. Si dividono in due categorie:
 - **Utenti finali**: utilizzano transazioni, cioè programmi che realizzano attività predefinite e di frequenza elevata
 - **Utenti casuali**: formulano interrogazioni di vario tipo.

1.5 VANTAGGI E SVANTAGGI DEI DBMS

Vantaggi:

- Permettono di considerare i dati come una risorsa comune di un'organizzazione
- La base di dati fornisce un modello unificato e preciso della parte del mondo reale di interesse per l'organizzazione
- È possibile un controllo centralizzato dei dati
- Evita ridondanze e inconsistenze
- I dati sono indipendenti e ciò favorisce lo sviluppo di applicazioni più flessibili e facilmente modificabili

Svantaggi:

- Sono costosi, complessi e abbastanza diversi da molti altri strumenti informatici
- Forniscono, in forma integrata, una serie di servizi, che sono associati ad un costo

2 IL MODELLO RELAZIONALE

2.1 IL MODELLO RELAZIONALE: STRUTTURE

2.1.1 Modelli logici nei sistemi di base di dati

Il modello relazionale si basa su due concetti: relazione e tabella. In questo caso “relazione” deriva dalla sua accezione matematica.

Il modello relazionale risponde al requisito dell’indipendenza dei dati, che prevede una distinzione tra il livello fisico e il livello logico: gli utenti che accedono ai dati e i programmatore che sviluppano le applicazioni fanno riferimento solo al livello logico, i dati descritti al livello logico sono poi realizzati per mezzo di opportune strutture fisiche, ma per accedere ai dati non è necessario conoscere le strutture fisiche stesse.

Precisiamo che il termine “relazione” viene utilizzato in tre accezioni che, nei dettagli, differiscono in modo importante:

- Relazione matematica
- Relazione del modello relazionale
- Relazione (tradotto da relationship), costrutto del modello concettuale entità-relazione

2.1.2 Relazioni e tabelle

La differenza tra una rappresentazione tabellare di un prodotto cartesiano e di una relazione è che la relazione è un sottoinsieme di tutti i prodotti cartesiani, ovvero solo i prodotti che ci interessano. Per fare un esempio:

$A = \{1, 2, 4\}$, $D = \{a, b\}$ e vogliamo la relazione dove si prendono solo le coppie aventi 1 e 4.

Prodotto cartesiano	
1	a
1	b
2	a
2	b
4	a
4	b

Relazione	
1	a
1	b
4	a
4	b

2.1.3 Relazioni con attributi

Associando ad ogni campo un nome, detto attributo, si può descrivere il ruolo del dominio dei dati, rendendo così un dato un’informazione.

Ad esempio, vediamo ora due tabelle con gli stessi dati ma dove, nel primo caso, sono stati etichettati i campi.

SquadraDiCasa	SquadraOspitata	RetiCasa	RetiOspitata
Juventus	Lazio	3	1
Lazio	Milan	2	0
Juventus	Roma	1	2
Roma	Milan	0	1

Juventus	Lazio	3	1
Lazio	Milan	2	0
Juventus	Roma	1	2
Roma	Milan	0	1

I dati sono gli stessi, nel primo caso i dati assumono un significato che li rende interpretabili.

Ora introduciamo una notazione che utilizzeremo in seguito: Se t è una tupla X su $A \in X$, allora $t[a]$ (o $t.a$), indica il valore di t su A .

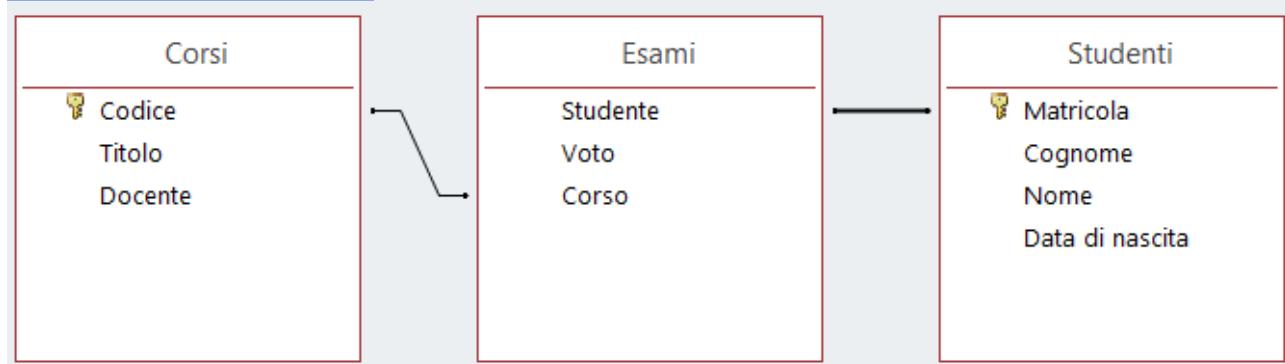
Ad esempio, se t è la prima tupla della relazione indicata sopra, possiamo dire $t[\text{SquadraOspitata}] = \text{Lazio}$.

Si può usare anche una notazione per denotare un insieme di attributi, ad esempio $t[\{\text{SquadraOspitata}, \text{RetiOspitata}\}] = \text{Lazio}, 1$.

2.1.4 Relazioni e basi di dati

Studenti			
Matricola	Cognome	Nome	Data di nascita
276545	Rossi	Maria	25/11/1981
485745	Neri	Anna	23/04/1982
200768	Verdi	Fabio	12/02/1982
587614	Rossi	Luca	10/10/1981
937653	Bruni	Mario	1/12/1981

Esami			Corsi		
Studente	Voto	Corso	Codice	Titolo	Docente
276545	28	01	01	Analisi	Giani
937653	25	01	03	Chimica	Melli
937653	25	01	04	Chimica	Belli
200768	24	04			



Rispetto a un modello basato su puntatori e record, il modello relazionale, basato su valori, presenta diversi vantaggi:

- Richiede di rappresentare solo ciò che è rilevante dal punto di vista dell'applicazione
- La rappresentazione logica dei dati non fa alcun riferimento a quella fisica, che può cambiare nel tempo: il modello relazionale permette quindi di ottenere l'indipendenza fisica dei dati.
- Essendo tutta l'informazione contenuta nei valori, è relativamente semplice trasferire i dati da un contesto a un altro.

2.1.5 Informazione incompleta e valori nulli

Il modello relazionale impone un certo grado di rigidità, in quanto le informazioni devono essere rappresentate per mezzo di tuple di dati omogenee.

Vale la pena notare che non sarebbe corretto utilizzare un valore del dominio per rappresentare l'assenza di informazione. Per rappresentare in modo semplice, ma al tempo stesso comodo, la non disponibilità di valori, si usa un valore speciale: null.

2.2 VINCOLO DI INTEGRITÀ

Riprendiamo la tabella di prima ma introduciamo l'attributo "lode".

Esami			
Studente	Voto	Lode	Corso
276545	28		01
937653	25	Lode	01
937653	25		01
200768	24		04

Possiamo vedere che è impossibile che il valore alla seconda tupla sia corretto in quanto "25 e lode" non è un voto valido.

Allo scopo di evitare queste situazioni è stato introdotto il concetto di "vincolo di integrità", ovvero una proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette.

Ogni vincolo può essere visto come un predicato che associa ad ogni istanza il valore vero o falso a seconda se l'istanza sia lecita o no.

Ci sono due categorie di vincoli:

- **Vincolo intrarelazionale:** il suo soddisfacimento è definito rispetto a singole relazioni della base di dati
 - Vincolo di tupla: è un vincolo che può essere valutato su ciascuna tupla
 - Vincolo su valori / di dominio: un vincolo definito con riferimento ai singoli valori (ad esempio il voto deve essere compreso tra 18 e 30)
- **Vincolo interrelazionale:** coinvolge più relazioni, ad esempio che un numero di matricola possa comparire in Esami se e solo se compare anche in Studenti

2.2.1 Chiavi

Una chiave è un attributo che identifica univocamente una tupla, ad esempio le matricole per gli studenti.

2.2.2 Chiavi e valori nulli

Dobbiamo porre dei limiti alla presenza di valori nulli nelle chiavi delle relazioni. Su una delle chiavi (detta la chiave primaria) si vieta la presenza di valori nulli. Un modo per evidenziare la chiave primaria è sottotitolare il nome dell'attributo.

2.2.3 Vincoli di integrità referenziale

Un vincolo di integrità referenziale fra un insieme di attributi X di una relazione R₁ e un'altra relazione R₂ è soddisfatto se i valori su X di ciascuna tupla dell'istanza di R₁ compaiono come valori della chiave (primaria) dell'istanza di R₂.

In pratica: un dato presente in una tabella e con un riferimento che lo lega ad un'altra tabella deve essere presente da entrambe le parti. Riprendendo le tabelle di prima

Studenti			
Matricola	Cognome	Nome	Data di nascita
485745	Neri	Anna	23/04/1982
200768	Verdi	Fabio	12/02/1982
587614	Rossi	Luca	10/10/1981
937653	Bruni	Mario	1/12/1981

Esami		
Studente	Voto	Corso
276545	28	01
937653	25	01
937653	25	01
200768	24	04

Possiamo vedere che lo studente con matricola 276545 presente nella tabella Esami non esiste nella tabella Studenti. Essendo le tue tabelle legate dalla relazione tra Studenti.Matricola e Esami.Studenti, ed essendo Matricola la chiave primaria della tabella Studenti, la mancanza del valore di matricola prima citato è una violazione al vincolo di integrità referenziale.

3 ALGEBRA E CALCOLO RELAZIONALE

3.1 ALGEBRA RELAZIONALE

3.1.1 Unione, intersezione, differenza

Dobbiamo prima di tutto prestare attenzione al fatto che una relazione non è genericamente un insieme di tuple ma un insieme di tuple omogenee, ovvero definite sugli stessi attributi. Per tanto consideriamo ammissibili, nell'algebra relazionale, solo applicazioni di unione, intersezione e differenza a coppie di operandi omogenei.

LAUREATI			DIRIGENTI		
Matricola	Cognome	Età	Matricola	Cognome	Età
7274	Rossi	37	9297	Neri	56
7432	Neri	39	7432	Neri	39
9824	Verdi	38	9824	Verdi	38

LAUREATI \cup DIRIGENTI		
Matricola	Cognome	Età
7274	Rossi	37
7432	Neri	39
9824	Verdi	38
9297	Neri	56

LAUREATI \cap DIRIGENTI		
Matricola	Cognome	Età
7432	Neri	39
9824	Verdi	38

LAUREATI – DIRIGENTI		
Matricola	Cognome	Età
7274	Rossi	37

- L'unione di due relazioni r_1 e r_2 definite sullo stesso insieme di attributi X è indicata con $r_1 \cup r_2$ ed è una relazione ancora su X contenente le tuple che appartengono a r_1 o a r_2 oppure ad entrambe
- L'intersezione di r_1 e r_2 è indicata con $r_1 \cap r_2$ ed è la relazione su X contenente le tuple che appartengono a r_1 e a r_2
- La differenza di r_1 e r_2 è indicata con $r_1 - r_2$ ed è una relazione su X contenente le tuple che appartengono a r_1 e non a r_2

3.1.2 Ridenominazione

Per risolvere i problemi di omogeneità tra insiemi dove i dati possono esprimere le stesse informazioni possiamo usare la ridenominazione. L'operatore è detto di ridenominazione perché cambia il nome degli attributi lasciando inalterato il contenuto delle relazioni.

Prendiamo ad esempio le tabelle Paternità e Maternità e immaginiamo di volerle unire. Ci si pone subito davanti il problema per il quale i termini Padre e Madre, per quanto equivalenti dal punto di vista della genitorialità, non sono omogenei. Possiamo quindi applicare due ridenominazioni ai due attributi rendendoli Genitore e poi eseguire l'unione.

Paternità		$P_{\text{Genitore} \leftarrow \text{Padre}} \text{ Paternità}$		$\text{Paternità} \cup \text{Maternità}$	
Padre	Figlio	Genitore	Figlio	Genitore	Genitore
Adamò	Caino	Adamò	Caino	Adamò	Adamò
Adamò	Abele	Adamò	Abele	Adamò	Abele
Abramo	Isacco	Abramo	Isacco	Abramo	Isacco
Abramo	Ismaele	Abramo	Ismaele	Abramo	Ismaele

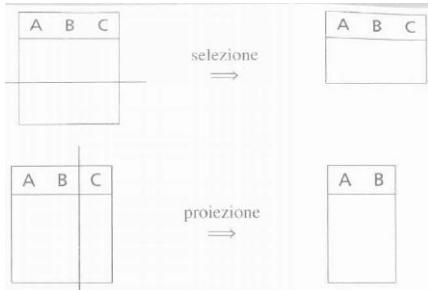
Maternità		$P_{\text{Genitore} \leftarrow \text{Madre}} \text{ Maternità}$		$\text{Paternità} \cup \text{Maternità}$	
Madre	Figlio	Genitore	Figlio	Genitore	Genitore
Eva	Caino	Eva	Caino	Adamò	Adamò
Eva	Set	Eva	Set	Adamò	Adamò
Sara	Isacco	Sara	Isacco	Abramo	Isacco
Agar	Ismaele	Agar	Ismaele	Abramo	Ismaele

3.1.3 Selezione

Ci sono tre operatori che ci permettono di manipolare le relazioni:

- Selezione (σ)

- Proiezione (Π)
- Join



Selezione e proiezione svolgono due compiti complementari:

- Selezione produce un sottoinsieme di tuple con tutti gli attributi
- Proiezione produce un sottoinsieme di attributi su tutte le tuple

Esempio di selezione:

Cittadini			
Cognome	Nome	CittàDiNascita	Residenza
Rossi	Mario	Roma	Milano
Neri	Luca	Roma	Roma
Verdi	Nico	Firenze	Firenze
Rossi	Marco	Napoli	Firenze

$\sigma_{\text{CittàDiNascita} = \text{Residenza}}(\text{Cittadini})$			
Cognome	Nome	CittàDiNascita	Residenza
Neri	Luca	Roma	Roma
Verdi	Nico	Firenze	Firenze

3.1.4 Proiezione

Esempio di proiezione:

Impiegati			
Cognome	Nome	Reparto	Capo
Rossi	Mario	Vendite	Gatti
Neri	Luca	Vendite	Gatti
Verdi	Nico	Personale	Lupi
Rossi	Marco	Personale	Lupi

$\Pi_{\text{Reparto}, \text{Capo}}(\text{Impiegati})$	
Reparto	Capo
Vendite	Gatti
Personale	Lupi

Bisogna notare una cosa: nella proiezione le tuple uguali “collassano” in una sola tupla, evitando ripetizioni.

3.1.5 Join

È l'operatore che permette di correlare dati contenuti in relazioni diverse.

Esistono due versioni dell'operatore, riconducibili l'una all'altra:

- Join naturale
- Theta-join

3.1.5.1 Join naturale (\bowtie^1)

È un operatore che correla dati in relazioni diverse sulla base di valori uguali in attributi con lo stesso nome.

R1		R2		R1 \bowtie R2		
Impiegato	Reparto	Reparto	Capo	Impiegato	Reparto	Capo
Rossi	Vendite	Produzione	Mori	Rossi	Vendite	Bruni
Neri	Produzione	Vendite	Bruni	Neri	Produzione	Mori
Bianchi	Produzione			Bianchi	Produzione	Mori

¹ 22C8 + alt + x

3.1.5.2 Join completi e incompleti

- Se il join di r_1 e r_2 è completo, allora contiene almeno un numero di tuple pari al massimo tra la cardinalità di r_1 e r_2 ; questo vuol dire che tutti i campi di join erano presenti in entrambe le tabelle
- Se il join di r_1 e r_2 è incompleto, allora contiene un numero di tuple inferiore al massimo tra la cardinalità di r_1 e r_2 ; questo vuol dire che non tutti i campi di join erano presenti in entrambe le tabelle

Il caso estremo di join incompleto è il join vuoto.

3.1.5.3 Join esterni

r_1	Impiegato	Reparto		r_2	Reparto	Capo
	Rossi	vendite			produzione	Mori
	Neri	produzione			acquisti	Bruni
	Bianchi	produzione				
$r_1 \bowtie_{LEFT} r_2$	Impiegato	Reparto	Capo			
	Rossi	vendite	NULL			
	Neri	produzione	Mori			
	Bianchi	produzione	Mori			
$r_1 \bowtie_{RIGHT} r_2$	Impiegato	Reparto	Capo			
	Neri	produzione	Mori			
	Bianchi	produzione	Mori			
	NULL	acquisti	Bruni			
$r_1 \bowtie_{FULL} r_2$	Impiegato	Reparto	Capo			
	Rossi	vendite	NULL			
	Neri	produzione	Mori			
	Bianchi	produzione	Mori			
	NULL	acquisti	Bruni			

È una versione del join che prevede che tutte le tuple diano un contributo al risultato, eventualmente estese con valori nulli. Esistono tre varianti:

- **Join esterno sinistro:** estende solo le tuple del primo operando
- **Join esterno destro:** estende solo le tuple del secondo operando
- **Join completo:** le estende tutte

3.1.5.4 Theta-join ed equi-join

Un prodotto cartesiano ha spesso poca utilità dal punto di vista informativo, infatti da questo solitamente viene eseguita una selezione successiva.

Il theta join esegue una selezione legata ad un join, ovvero

$$r_1 \bowtie_F r_2 = \sigma_F (r_1 \bowtie r_2)$$

3.1.6 Interrogazioni in algebra relazionale

In algebra relazionale, le interrogazioni su uno schema di base di dati R vengono formulate con espressioni i cui atomi sono nomi di relazioni in R.

Ci sono più tipi di interrogazioni, da quelle semplici come "Impiegati(Matricola, Nome, Età, Stipendio)" a quelle più elaborate che sfruttano selezione, proiezione e join, come " $\Pi_{Capo}(Supervisione \bowtie_{Impiegato = Matricola \sigma_{Stipendio > 40}} (Impiegati))$ ", ovvero trovare le matricole dei capi degli impiegati che guadagnano più di 40 mila euro.

Nell'algebra relazionale esistono delle trasformazioni di equivalenza, cioè operazioni che sostituiscono un'espressione con un'altra a essa equivalente.

3.1.7 Algebra con valori nulli

Fino ad ora abbiamo presupposto l'assenza di valori nulli. In questo caso la valutazione può assumere il valore vero, falso oppure unkown (sconosciuto), rappresentato con il simbolo U. L'estensione delle tabelle di verità con questo nuovo valore sono:

Not	
F	V
U	U
V	F

And	V	U	F
V	V	U	F
U	U	U	F
F	F	F	F

Or	V	U	F
V	V	V	V
U	V	U	U
F	V	U	F

Il significato di queste tavole di verità è utile solo nel caso di interrogazioni complesse.

La soluzione per superare questi inconvenienti è trattare i valori nulli da un punto di vista sintattico, rinunciando alla speculazione sul loro valore di verità.

Si introducono quindi due nuove forme di condizioni atomiche di selezioni:

- A IS NULL: assume valore vero su una tupla t se il valore di t su A è nullo e falso se esso è specificato
- A IS NOT NULL: assume valore vero su una tupla t se il valore di t su A è specificato e falso se il valore è nullo.

3.1.8 Viste

Nel modello relazionale, la tecnica per avere rappresentazioni differenti degli stessi dati è quella delle relazioni derivate, relazioni il cui contenuto è funzione del contenuto di altre relazioni.

In una base di dati possono esistere relazioni di base, il cui contenuto è autonomo, e relazioni derivate, il cui contenuto è funzione di quello di altre relazioni. In linea di principio, possono esistere due tipi di relazioni derivate:

- **Viste materializzate**: relazioni derivate effettivamente memorizzate nella base di dati
- **Relazioni virtuali** (chiamate anche semplicemente viste): relazioni definite per mezzo di funzioni, non memorizzate nella base di dati ma utilizzabili nelle interrogazioni come se lo fossero

Le viste materializzate occupano spazio in memoria, quindi sono utili quando gli aggiornamenti sono rari rispetto alle interrogazioni o se il loro calcolo è complesso.

Le viste vengono definite nei sistemi relazionali per mezzo di espressioni del linguaggio di interrogazione.

3.2 CALCOLO RELAZIONALE

Con il termine calcolo relazionale si fa riferimento a una famiglia di linguaggi di interrogazione, basati sul calcolo dei predicati del primo ordine, che hanno la caratteristica di essere dichiarativi, cioè di specificare le proprietà del risultato delle interrogazioni.

Sono tante le versioni del calcolo relazionale, quindi qua descriveremo solo:

- Il calcolo relazionale su domini
- Il calcolo su tuple con dichiarazioni di range

3.2.1 Calcolo relazionale su domini

Le espressioni del calcolo relazionale su domini hanno la forma:

$$\{A_1 : x_1, \dots, A_k : x_k \mid f\}$$

Dove:

- A_1, \dots, A_k sono attributi
- x_1, \dots, x_k sono variabili
- f è una formula che segue le seguenti regole:
 - $R(A_1 : x_1, \dots, A_p : x_p)$, dove $R(A_1 \dots A_p)$ è uno schema di relazione e x_1, \dots, x_p sono variabili
 - $x\theta y$ o $x\theta c$, dove x e y sono variabili, c è una costante e θ è un operatore di confronto

la lista di coppie $A_k : x_k$ viene chiamata target list in quanto definisce la struttura del risultato, che è costituito dalla relazione su A_1, \dots, A_k che contiene le tuple i cui valori sostituiti a x_1, \dots, x_k rendono vera la formula.

Ora vediamo la trasformazione da algebra relazionale a calcolo relazionale:

$$\sigma_{\text{Stipendio} > 40}(\text{Impiegati})$$



$$\{\text{Matricola} : m, \text{Nome} : n, \text{Età} : e, \text{Stipendio} : s \mid \text{Impiegati}(\text{Matricola} : m, \text{Nome} : n, \text{Età} : e, \text{Stipendio} : s) \wedge s > 40\}$$

Per vederne una più complicata:

$$\pi_{\text{Capo}}(\text{Supervisione} \bowtie_{\text{Impiegato} = \text{Matricola}} \sigma_{\text{Stipendio} > 40}(\text{Impiegati}))$$



$$\{\text{Capo} : c \mid \text{Impiegati}(\text{Matricola} : m, \text{Nome} : n, \text{Età} : e, \text{Stipendio} : s) \wedge \text{Supervisione}(\text{Impiegato} : m, \text{Capo} : c) \wedge s > 40\}$$

3.2.2 pregi e difetti del calcolo su domini

in primo luogo, notiamo che il calcolo ammette espressioni che hanno poco senso dal punto di vista pratico.

Un linguaggio di interrogazione è indipendente dal dominio se il suo risultato, su ciascuna istanza di base di dati, non varia al variare del dominio rispetto al quale l'espressione è valutata.

Il requisito dell'indipendenza dal dominio è chiaramente fondamentale per i linguaggi reali, perché nella maggior parte dei casi le espressioni dipendenti dal dominio non hanno utilità pratica e possono produrre risultati di grandi dimensioni.

3.2.3 Calcolo su tuple con dichiarazioni di range

Le espressioni del calcolo su tuple con dichiarazioni di range hanno la forma:

$$\{T \mid \lambda \mid f\}$$

Dove:

- T è la target list, con elementi del tipo $Y : x.Z$, con x variabile e Y e Z sequenze di attributi di pari lunghezza
- λ è la range list, elenca le variabili libere dalla formula f con i relativi range
- F è una formula con:
 - Atomi del tipo $x.A\theta y$ o $x_1.A_1\theta x_2.A_2$, dove x e y sono variabili, c è una costante e θ è un operatore di confronto
 - Connettivi come nel calcolo su domini
 - Quantificatori di esistenza

Per fare un esempio:

$$\Pi_{\text{Capo}}(\text{Supervisione} \bowtie_{\text{Impiegato} = \text{Matricola}} \sigma_{\text{Stipendio} > 40} (\text{Impiegati}))$$

↓

$$\{s.\text{Capo} \mid i(\text{Impiegati}), s(\text{Supervisione}) \mid i.\text{Matricola} = s.\text{Impiegato} \wedge i.\text{stipendio} > 40\}$$

Su queste interrogazioni possiamo anche applicare unione, intersezione e differenza.

SQL è basato sul calcolo di tuple con dichiarazioni di range.

3.3 DATALOG

L'idea fondamentale su cui si basa il linguaggio Datalog è quella di adattare alle basi di dati il linguaggio di programmazione logica Prolog².

Per chi conosce Prolog, possiamo dire che in Datalog non sono previsti simboli di funzione f come in f(i1, i2, o).

In Datalog ci sono due tipi di predicati:

- Predicati estensionali: corrispondono alle relazioni nella base di dati
- Predicati intensionali: sono specificati (ma non materializzati) per mezzo di regole logiche. Concettualmente, definiscono le viste.

Le regole datalog hanno la forma: testa ← corpo, in cui:

- La testa è un predicato atomico
- Il corpo è una lista di condizioni

Sono imposte le seguenti condizioni:

- I predicati estensionali possono comparire solo nel corpo delle regole
- Se una variabile compare nella testa di una regola, allora deve comparire anche nel corpo della stessa regola
- Se una variabile compare in un atomo di confronto, allora deve comparire anche in un atomo nel corpo della stessa regola

Le interrogazioni in Datalog sono specificate semplicemente per mezzo di atomi R(A1 : a1, ..., Ap : ap) preceduti, qualche volta, da ?, che producono come risultato le tuple della relazione R che possono essere ottenute sostituendo correttamente le variabili.

Ad esempio:

$$\Pi_{\text{Capo}}(\text{Supervisione} \bowtie_{\text{Impiegato} = \text{Matricola}} \sigma_{\text{Stipendio} > 40} (\text{Impiegati}))$$

↓

$$\text{CapiDeiRicchi}(\text{Capo} : c) \leftarrow \text{Impiegati}(\text{Matricola} : m, \text{Nome} : n, \text{Età} : e, \text{Stipendio} : s), \text{Supervisione}(\text{Impiegato} : m, \text{Capo} : c), s > 40$$

In Prolog la , ha funge da congiunzione \wedge .

² Se proprio sei interessato a Prolog (perché?), vai a prendere il riassunto di Linguaggi di Programmazione. Proprio sicuro, eh?

4 SQL: CONCETTI BASE

4.1 DEFINIZIONE DEI DATI IN SQL

- () permettono di isolare un termine della sintassi
- [] indicano che il termine all'interno è opzionale, ossia può non comparire o comparire una sola volta
- { } indicano che il termine racchiuso può non comparire o comparire un numero arbitrario di volte
- | indica che deve essere scelto uno tra i termini separati dalle barre, un elenco di termini può essere racchiuso tra < >

4.1.1 I domini elementari

A partire da questi domini elementari si possono definire i domini da associare agli attributi dello schema.

- Caratteri: il dominio character permette di rappresentare singoli caratteri oppure stringhe. La lunghezza può essere fissa o variabile, se variabile si indica la lunghezza massima. La famiglia di caratteri (latino, cirillico, greco, ecc.) è una e una sola.
 - Sintassi: character [varying] [(Lunghezza)] [character set NomeFamigliaCaratteri]

SQL accetta anche le forme abbreviate `char` e `varchar`.

 - Esempio: character varying (1000) character set Greek
- Tipi numerici esatti: questa famiglia contiene i domini che permettono di rappresentare interi o con una parte decimale di lunghezza prefissata. Vengono messi a disposizione quattro tipi:
 - numeric [(Precisione [, scala])]
 - decimal [(Precisione [, scala])]
 - integer
 - smallint
 - bigint

Scala indica quante cifre decimali mostrare dopo la virgola. La differenza tra `numeric` e `decimal` consiste nel fatto che la precisione per `numeric` rappresenta un valore esatto, mentre per il dominio `decimal` costituisce un requisito minimo. Nel caso non venga scritta si assume precisione 0.
- Tipi numerici approssimati: SQL fornisce tre tipi:
 - float [(precisione)]
 - real
 - double precision

Tutti questi domini permettono di descrivere numeri approssimati mediante una rappresentazione in virgola mobile, in cui a ciascun numero corrisponde una coppia di valori: mantissa ed esponente. La mantissa è un valore frazionario, mentre l'esponente è un numero intero.

- Istanti temporali:
 - date
 - time [(Precisione)] [with time zone]
 - timestamp [(Precisione)] [with time zone]

Il dominio `date` ammette i campi `year`, `month` e `day`, il dominio `time` ammette i campi `hour`, `minute` e `second`, `timestamp` ammette tutti i campi.

L'opzione `with time zone` permette di accedere a due campi, `timezone_hour`, `timezone_minute`, che rappresentano la differenza di fuso orario tra l'ora locale e l'ora universale.

- Intervalli temporali:

- interval PrimaUnitàDiTempo [(precisione)] [to UltimaUnitàDiTempo [(precisione)]]

PrimaUnitàDiTempo e UltimaUnitàDiTempo definiscono le unità di misura che devono essere usate, dalla più precisa alla meno precisa, è così possibile indicare interval year to month per indicare che la durata dell'intervallo di tempo deve essere indicata in anni e mesi.

Dopo ogni unità è possibile specificare un valore indicante la precisione.

- Boolean
- BLOB e CLOB: permettono di rappresentare oggetti di grandi dimensioni, costituiti da una sequenza arbitraria di valori binari (blob) o di caratteri (clob). Per entrambi i domini il sistema memorizza il valore ma non permette che il valore venga utilizzato come criterio di selezione per le interrogazioni.

4.1.2 Definizione di schema

SQL consente la definizione di uno schema di base di dati come collezione di oggetti.

```
Create schema [nome schema] [ [ authorization ] Autorizzazione ]
{DefElementoSchema}
```

Autorizzazione rappresenta il nome del proprietario dello schema, se omesso si assume che il proprietario sia l'utente che ha lanciato il comando.

Il nome può essere omesso, nel caso si assume che sia il nome del proprietario.

Dopo il comando di create schema compaiono le definizioni dei suoi componenti. Vediamo ora alcune definizioni.

4.1.3 Definizione delle tabelle

Una tabella in SQL è costituita da una collezione ordinata di attributi e da un insieme (eventualmente vuoto) di vincoli.

La sintassi è:

```
create table NomeTabella
( NomeAttributo Dominio [Valore di default] [Vincoli]
  {, NomeAttributo Dominio [Valore di default] [Vincoli] }
  AltriVincoli
)
```

Per fare un esempio:

```
create table Dipartimento
(
    Nome      varchar(20) primary key,
    Indirizzo varchar(50),
    Città     varchar(20)
)
```

4.1.4 Definizione dei domini

I domini elementari possono essere sfruttati per definire domini da parte dell'utente. La sintassi è:

```
create domain NomeDominio as TipoDiDato
    [ValoreDiDefault]
    [Vincolo]
```

La dichiarazione di nuovi domini permette di associare un insieme di vincoli a un nome di dominio.

4.1.5 Specifica di valori di default

La specifica vista prima ValoreDiDefault assume null come valore se non è specificato altrimenti.

La sintassi per la specifica è:

```
default < GenericoValore | user | null >
```

- GenericoValore rappresenta un valore compatibile col dominio
- User impone come valore l'identificativo dell'utente che esegue il comando di aggiornamento

Per fare un esempio: NumeroFigli smallint default 0

4.1.6 Vincoli intrarelazionali

I vincoli sono proprietà che devono essere verificate da ogni istanza della base di dati.

Il costrutto più potente per specificare vincoli generici, ovvero sia intrarelazionali che interrelazionali, è il costrutto di check, che richiede però di formulare delle interrogazioni sulla base di dati.

I più semplici vincoli di tipo intrarelazionali³ sono i vincoli:

- not null
- unique
- primary key

4.1.6.1 Not null

SQL non permette di distinguere tra le diverse interpretazioni del valore nullo. Il vincolo not null indica che il valore nullo non è ammesso come valore dell'attributo; in tal caso l'attributo deve sempre essere specificato a meno che non sia stato associato un valore di default differente da null.

4.1.6.2 Unique

Un vincolo unique si applica a un attributo o a un insieme di attributi di una tabella e impone che i valori dell'attributo sia una (super) chiave, cioè righe differenti della tabella hanno valori differenti ad eccezione del caso null.

Nel caso non sia un singolo attributo a necessitare di questo vincolo, si può porre in calce come elenco, per fare un esempio:

```
Nome      varchar(20) not null,
Indirizzo  varchar(50) not null,
unique (Nome, Indirizzo)
```

in questo caso si impone che due righe non possano avere uguale la coppia di attributi Nome e Indirizzo, altrimenti:

```
Nome      varchar(20) not null unique,
```

³ il suo soddisfacimento è definito rispetto a singole relazioni della base di dati

Indirizzo varchar(50) not null unique

Impone il vincolo ai singoli attributi.

4.1.6.3 Primary key

Di norma è necessario specificare per ogni relazione la chiave primaria. SQL permette di specificarlo una sola volta per ogni tabella. Può essere applicato anche ad insiemi di attributi come per `unique`. I valori con vincolo `primary key` non possono essere uguali a `null`.

4.1.7 Vincoli interrelazionali

I vincoli interrelazionali più diffusi e significativi sono i vincoli di integrità referenziale, per la loro definizione si usa l'apposito vincolo di `foreign key`, ovvero chiave esterna.

Questo vincolo crea un legame tra i valori di un attributo della tabella su cui è definito e i valori di un'altra tabella. Il vincolo impone che ogni elemento della prima tabella abbia un suo omologo nella seconda e, soprattutto, che nella prima tabella l'attributo abbia vincolo `unique`.

Se c'è un solo attributo coinvolto si può usare il costrutto `references`, con il quale si specificano la tabella esterna e l'attributo.

Se è un insieme di attributi si usa `foreign key`, posto al termine della definizione degli attributi. Il costrutto elenca gli attributi della tabella coinvolti nel legame, cui segue la definizione dei corrispondenti attributi della tabella esterna mediante il costrutto `references`.

Per fare due esempi:

```
create table Impiegato
{
    Matricola character(6) primary key,
    Nome      varchar(20) not null,
    Cognome   varchar(20) not null,
    Dipart     varchar(15)
        References Dipartimento(NomeDip),
        unique (Cognome, Nome)
}
```

Mentre per il secondo caso possiamo porre in calce:

```
foreign key (Nome, Cognome)
    references Anagrafica(Nome, Cognome)
```

Le operazioni sulla tabella esterna che possono introdurre delle violazioni sono le modifiche del valore dell'attributo riferito e la cancellazione delle righe.

In particolare, per le operazioni di modifica, è possibile reagire in uno dei seguenti modi:

- **Cascade:** il nuovo valore dell'attributo della tabella esterna viene riportato su tutte le corrispondenti righe della tabella interna
- **Set null:** all'attributo referente viene assegnato il valore nullo al posto del valore modificato nella tabella esterna

- `Set default`: all'attributo referente viene assegnato il valore di default al posto del valore modificato nella tabella esterna
- `No action`: l'azione di modifica non viene consentita, senza che il sistema provi a riparare la violazione

Usando la politica cascade si assume che le righe della tabella interna siano strettamente legate alle corrispondenti righe della tabella esterna, per cui se si apporta una modifica alla tabella esterna deve modificare anche quella interna.

4.1.8 Modifica degli schemi

SQL fornisce primitive per la manipolazione degli schemi delle basi di dati.

I comandi sono:

- `Alter`
- `Drop`

4.1.8.1 *Alter*

Il comando alter permette di modificare domini e schemi di tabelle.

```
Alter domain NomeDominio <set default ValoreDefault |
    Drop default |
    Add constraint DefVincolo |
    Drop constraint NomeVincolo >
```

```
Alter table NomeTabella <
    Alter column NomeAttributo <set default NuovoDefault |
        Drop default > |
        Add constraint DefVincolo |
        Drop constraint NomeVincolo |
        Add column DefAttributo |
        Drop column NomeAttributo >
```

Tramite `alter domain` e `alter table` è possibile aggiungere e rimuovere vincoli e modificare i valori di default.

Notare che i nuovi vincoli devono essere già soddisfatti dai dati presenti.

4.1.8.2 *Drop*

Mentre il comando alter effettua delle modifiche, drop permette di rimuovere di componenti.

```
Drop < schema | domain | table | view | assertion > NomeElemento
    [restrict | cascade]
```

L'opzione `restrict` specifica che il comando non deve essere eseguito in presenza di oggetti non vuoti: uno schema non è rimosso se contiene tabelle o altri oggetti. `Restrict` è l'opzione di default.

`Cascade`, invece, rimuove tutti gli oggetti specificati.

4.1.9 Cataloghi relazionali

Tutti i DBMS relazionali gestiscono il proprio dizionario dei dati (ovvero la descrizione delle tabelle presenti nella base di dati) mediante una struttura relazionale, cioè tramite tabelle.

Ci sono due tipi di tabelle:

- Quelle che contengono i dati
- Quelle che contengono i metadati, ovvero dati che descrivono i dati

Questa caratteristica delle implementazioni dei sistemi relazionali viene detta riflessività. Quasi sempre una base di dati gestisce il catalogo mediante strutture analoghe a quelle usate per conservare l'istanza.

4.2 INTERROGAZIONI IN SQL

4.2.1 Dichiaratività di SQL

SQL esprime le interrogazioni in modo dichiarativo, ovvero si specifica l'obiettivo dell'interrogazione e non il modo in cui ottenerlo, a differenza del modello relazionale.

Il query optimizer legge la query SQL e la traduce nella corrispettiva query in algebra relazionale.

Esistono molti modi diversi per esprimere la stessa interrogazione in SQL: il programmatore dovrà effettuare una scelta basandosi non sull'efficienza, bensì su caratteristiche come la leggibilità e la modificabilità dell'interrogazione.

4.2.2 Interrogazioni semplici

```
select ListaAttributi
from ListaTabelle
[where Condizione]
```

Figura 1 - Query base

Le operazioni di interrogazione in SQL vengono specificate per mezzo di una struttura base composta da `select`, `from` ed eventualmente altre parole chiave.

L'interrogazione SQL seleziona, tra le righe che appartengono al prodotto cartesiano delle tabelle elencate nella clausola `from`, quelle che soddisfano le condizioni espresse nell'argomento `where`. Il risultato è una tabella con una riga per ogni riga prodotta dalla clausola `from` e filtrata da `where`, le cui colonne si ottengono dalla valutazione delle espressioni `AttrEspr` che appaiono nella clausola `select`.

Ogni colonna del risultato viene eventualmente ridenominata con l'alias. Anche le tabelle possono essere ridenominate.

```
select AttrEspr [[as] Alias] {, AttrEspr
[[as] Alias]}
```

```
from Tabella [[as] Alias] {, Tabella [[as]
Alias]}
```

```
[where Condizione]
```

Figura 2 - Query dettagliata

4.2.2.1 Clauses select

Specifica gli elementi dello schema della tabella risultato. Come argomento può anche comparire il carattere speciale `*` che rappresenta la selezione di tutti gli attributi. Per scegliere uno specifico attributo, nel caso ci siano più tabelle, si usa la notazione `NomeTabella.Attributo`.

4.2.2.2 *Clausola from*

Quando si desidera formulare un'interrogazione che coinvolge righe appartenenti a più di una tabella, si pone come argomento della clausola from l'insieme di tabelle. Sul prodotto cartesiano delle tabelle elencate verranno applicate le condizioni contenute nella clausola where.

4.2.2.3 *Clausola where*

La clausola where, opzionale, ammette come argomento un'espressione booleana costruita combinando predicati semplici con gli operatori and, or e not.

4.2.2.4 *Gestione dei valori nulli*

Per selezionare i termini con valori nulli SQL fornisce "is null", la cui sintassi è semplicemente "is [not] null".

4.2.2.5 *Interpretazione formale delle interrogazioni SQL*

```
Select T1.Attributo11, ..., Th.Attributohm
From Tabella1T1, ..., TabellanTn → π T1.Attributo11, ..., Th.Attributohm (σCondizione(Tabella1⋈ ... ⋈ Tabellan))
Where condizione
```

4.2.2.6 *Duplicati*

In SQL si possono avere in una tabella più righe uguali. Il rimuovere i duplicati, però, è un lavoro costoso e spesso non necessario, si è quindi deciso in SQL di permettere la presenza di duplicati all'interno delle tabelle, lasciando a chi scrive l'interrogazione il computo di specificare esplicitamente quando l'operazione è necessaria.

L'eliminazione dei duplicati è specificata con la parola chiave distinct, da porre immediatamente dopo la parola select. La sintassi prevede che si possa specificare anche all al posto di distinct, anche se è in realtà l'opzione di default.

4.2.2.7 *Join interni ed esterni*

```
from Tabella [[as] Alias] {[TipoJoin]} join
Tabella [[as] Alias] on CondizioneDiJoin}
```

```
Select I.Nome, Cognome, D.Città
```

```
From Impiegato I join Dipartimento D on
Dipart = D.Nome
```

Una sintassi alternativa per i join permette di distinguere tra le condizioni che compaiono nell'interrogazione, quelle che rappresentano condizioni di join e quelle che rappresentano condizioni di selezione sulle righe.

Il parametro TipoJoin specifica qual è il tipo di join da usare (inner, che è di default, right outer, left outer, full outer, dove outer è opzionale).

La distinzione tra i tre join è:

- Left: fornisce come risultato il join interno esteso con le righe della tabella che compare a sinistra per le quali non esiste una corrispondente riga nella tabella di destra
- Right: fornisce come risultato il join interno esteso con le righe della tabella che compare a destra per le quali non esiste una corrispondente riga nella tabella di sinistra
- Full: restituisce l'insieme delle precedenti

Per fare un esempio:

Guidatore	Nome	Cognome	NroPatente
Mario	Rossi	VR 2030020Y	
Carlo	Bianchi	PZ 1012436B	
Marco	Neri	AP 4544442R	

Automobile	Targa	Marca	Modello	NroPatente
AB 574 WW	Fiat	Punto	VR 2030020Y	
AA 652 FF	Fiat	Brava	VR 2030020Y	
BJ 747 XX	Lancia	Delta	PZ 1012436B	
BB 421 JJ	Fiat	Uno	AP 4544442R	

Select Nome, Cognome, G.NroPatente, Targa, Marca, Modello

From Guidatore G [TipoJoin] join Automobile A on (G.NroPatente=A.NroPatente)

Left	Nome	Cognome	G.NroPatente	Targa	Marca	Modello
Mario	Rossi	VR2030020Y	AB 574 WW	Fiat	Punto	
Mario	Rossi	VR2030020Y	AA 652 FF	Fiat	Brava	
Carlo	Bianchi	PZ 1012436B	BJ 747 XX	Lancia	Delta	
Marco	Neri	AP 4544442R	NULL	NULL	NULL	

Full	Nome	Cognome	G.NroPatente	Targa	Marca	Modello
Mario	Rossi	VR2030020Y	AB 574 WW	Fiat	Punto	
Mario	Rossi	VR2030020Y	AA 652 FF	Fiat	Brava	
Carlo	Bianchi	PZ 1012436B	BJ 747 XX	Lancia	Delta	
Marco	Neri	AP 4544442R	NULL	NULL	NULL	
NULL	NULL	NULL	BB 421 JJ	Fiat	Uno	

Right	Nome	Cognome	G.NroPatente	Targa	Marca	Modello
Mario	Rossi	VR2030020Y	AB 574 WW	Fiat	Punto	
Mario	Rossi	VR2030020Y	AA 652 FF	Fiat	Brava	
Carlo	Bianchi	PZ 1012436B	BJ 747 XX	Lancia	Delta	
NULL	NULL	NULL	BB 421 JJ	Fiat	Uno	

4.2.2.8 Ordinamento

order by AttrDiOrdinamento [asc | desc] {, AttrDiOrdinamento [asc | desc]}

SQL permette di specificare un ordinamento delle righe del risultato di un'interrogazione tramite la clausola "order by", con la quale si chiude l'interrogazione.

4.2.3 Operatori aggregati

Sono cinque gli operatori aggregati:

- Count: conta gli elementi dell'attributo selezionato
- Sum: restituisce la somma dei valori posseduti dall'espressione

- Max: restituisce il massimo⁴
- Min: restituisce il minimo
- Avg: restituisce la media dei valori (ovvero sum/count)

Count ($< * | [distinct | all] \text{ListaAttributi} >$)

$< \text{sum} | \text{max} | \text{min} | \text{avg} > ([\text{distinct} | \text{all}] \text{AttrEspr})$

4.2.4 Interrogazioni con raggruppamento

```
select ListaAttributi
from ListaTabelle
group by Attributo
```

Molto spesso serve applicare l'operatore aggregato separatamente a sottoinsiemi di righe. Per poter utilizzare in questo modo l'operatore aggregato, SQL mette a disposizione la clausola "group by", che permette di specificare come dividere le tabelle in sottoinsiemi.

ad esempio:

Impiegato	Nome	Cognome	Dipart	Ufficio	Stipendio
Mario	Rossi		Amministrazione	10	45
Carlo	Bianchi		Produzione	20	36
Giovanni	Verdi		Amministrazione	20	40
Franco	Neri		Distribuzione	16	45
Carlo	Rossi		Direzione	14	80
Lorenzo	Gialli		Direzione	7	73
Paola	Rosati		Amministrazione	75	40
Marco	Franco		Produzione	20	46

```
select Dipart, sum(Stipendio)
from Impiegato
group by Dipart
```

Dipart	sum(Stipendio)
Amministrazione	125
Produzione	82
Distribuzione	45
Direzione	153

4.2.4.1 Predicati sui gruppi

```
select ListaAttributi
from ListaTabelle
group by Attributo
having Condizione
```

il corrispettivo della clausola `where` per i gruppi è "having". Questa clausola descrive le condizioni che si devono applicare al termine dell'esecuzione di un'interrogazione che fa uso della clausola `group by`.

Ad esempio, mettendo "having sum(Stipendio) > 100" nella query precedente avremmo avuto solo due righe, Amministrazione e Direzione.

⁴ GAC

4.2.5 Interrogazioni di tipo insiemistico

SQL mette a disposizione anche degli operatori insiemistici, simili a quelli dell'algebra relazionale. Gli operatori sono:

- Union
- Intersect
- Except (chiamato anche minus)

```
Query { < union | intersect | except > [all] query }
```

4.2.6 Interrogazioni nidificate

Esempio: estrarre i dipartimenti in cui non lavorano persone di cognome "Rossi"

Per estendere i classici operatori logici, SQL introduce le interrogazioni nidificate, dove viene usata un'interrogazione all'interno della clausola where.

```
Select Nome
```

```
From Dipartimento
```

```
Where Nome <> all (select Dipart
                      From Impiegato
                      Where Cognome = 'Rossi')
```

La soluzione offerta consiste nell'estendere, con le parole chiave all o any i normali operatori di confronto:

- Any: specifica che la riga soddisfa la condizione se risulta vero il confronto tra il valore dell'attributo per la riga e almeno uno degli elementi restituiti dall'interrogazione
- All: la riga soddisfa la condizione se e solo

se tutti gli elementi restituiti dall'interrogazione nidificata rendono vero il confronto

4.2.6.1 Interrogazioni nidificate complesse

Un'interpretazione molto semplice e intuitiva delle interrogazioni nidificate consiste nell'assumere che l'interrogazione nidificata venga eseguita prima di analizzare le righe dell'interrogazione esterna e che questo venga memorizzata separatamente.

4.3 MODIFICA DEI DATI IN SQL

I comandi che permettono di modificare la base di dati sono insert, delete e update.

4.3.1 Inserimento

```
Insert into      NomeTabella
              [ListaAttributi] <values
              (ListaValori) |
SelectSQL >
```

SelectSQL indica che si può nidificare un'altra selezione e i suoi valori saranno inseriti in NomeTabella.

4.3.2 Cancellazione

```
Delete from NomeTabella
[where Condizione]
```

4.3.3 Modifica

```
Update NomeTabella  
Set Attributo = < Espressione | SelectSQL | null | default >  
{, Attributo = < Espressione | SelectSQL | null | default > }  
[where Condizione]
```

5 SQL: CARATTERISTICHE EVOLUTE

5.1 CARATTERISTICHE EVOLUTE DI DEFINIZIONE DEI DATI

5.1.1 Vincoli di integrità generici

Check <Condizione> SQL permette di specificare un certo insieme di vincoli sugli attributi e sulle tabelle. Per specificare ulteriori vincoli è stata introdotta la clausola check, con la sintassi qua accanto.

La condizione deve essere sempre verificata affinchè la base di dati sia corretta.

```
create table Impiegato
(Matricola character(6)
    check (Matricola is not null and
           1 = (select count(*)
                 from Impiegato I
                 where Matricola=I.Matricola)),
Cognome character(20) check (Cognome is not null),
Nome      character(20) check (Nome is not null and
                                2 > (select count(*)
                                      from Impiegato I
                                      where Nome = I.Nome
                                        and Cognome = I.Cognome)),
Dipart     character(15) check (Dipart in
                                (select NomeDip
                                  from Dipartimento))
```

5.1.2 Asserzioni

Create assertion NomeAsserzione
check (Condizione)

associati a nessun attributo o tabella in particolare ma direttamente all'intero schema.

Grazie alla clausola check è possibile definire anche un ulteriore componente dello schema di una base di dati, le asserzioni. Rappresentano dei vincoli che non sono

Ogni vincolo di integrità può essere:

- Immediato: viene controllato dopo ogni modifica
- Differito: viene controllato solo al termine dell'esecuzione di una serie di operazioni.

Il controllo differito viene tipicamente introdotto per gestire situazioni in cui non è possibile costruire una situazione consistente con una singola modifica della base di dati.

Quando un vincolo immediato non è soddisfatto l'operazione di modifica che ha causato la violazione è stata appena eseguita e il sistema può disfarla, questo modo di procedere è chiamato rollback parziale.

Le asserzioni possono essere cancellate tramite drop.

5.2 FUNZIONI SCALARI

SQL mette a disposizione diverse funzioni scalari che possono essere usate all'interno delle espressioni del linguaggio.

5.2.1 Famiglie di funzioni

Postgres, un DBMS open-source particolarmente interessante descritto in una delle appendici, offre svariate funzioni:

- Funzioni temporali
 - current_date, current_time, current_timestamp: restituiscono il valore dell'orologio del sistema.
 - Extract: isola una qualsiasi componente di un dominio temporale (year, month, ecc.)
 - Age: restituisce l'intervallo di differenza tra una data e l'istante corrente
- Funzioni di manipolazione di stringhe
 - Si applicano a espressioni che rappresentano stringhe di caratteri e permettono di trasformare il loro contenuto
 - Char_length
 - Lower, upper
 - Substring
- Funzioni di conversione dominio
 - Cast: permette di convertire un valore in un dominio nella sua rappresentazione in un altro dominio (esempio: cast (Data as char(10)) converte un valore del dominio date nella sua rappresentazione testuale)
- Funzioni condizionali
- Funzioni per la formattazione dell'output
- Funzioni matematiche
- Funzioni di accesso ai servizi del sistema operativo

5.2.2 Funzioni condizionali

5.2.2.1 Coalesce

`coalesce(NomeTabella, nomeSostituto)`

La funzione coalesce ammette come argomento una sequenza di espressioni e restituisce il primo valore non nullo.

5.2.2.2 Nullif

`nullif(NomeTabella, nomeSostituto)`

La funzione nullif richiede come argomento una espressione e un valore costante. Se l'espressione è pari al valore costante, la

funzione restituisce il valore nullo, altrimenti restituisce il valore dell'espressione.

5.2.2.3 Case

```
case Espressione
    when Valore then EsprRisultato
    {when Valore then EsprRisultato}
    [else EsprRisultato]
end
```

La funzione case permette di specificare strutture condizionali, il cui risultato dipende dalla valutazione del contenuto delle tabelle. La sintassi ammette due varianti.

```

case when Condizione then Espressione
      { when Condizione then Espressione}
      [else Espressione]
end

```

la prima forma restituisce risultati diversi a seconda del valore di una specifica espressione.

La seconda forma del costrutto case, invece, ammette la valutazione di predicati SQL generici.

5.3 CONTROLLO DELL'ACCESSO

SQL prevede che ogni utente sia identificato in modo univoco dal sistema.

5.3.1 Risorse e privilegi

Il sistema basa il controllo di accesso su un concetto di privilegio. Gli utenti possiedono dei privilegi di accesso alle risorse del sistema.

Ogni privilegio è caratterizzato dai seguenti parametri:

- La risorsa cui si riferisce
- L'utente che concede il privilegio
- L'utente che riceve il privilegio
- L'azione che viene permessa sulla risorsa
- Se il privilegio può essere trasmetto o meno ad altri utenti

Esiste inoltre un utente predefinito, _system, che rappresenta il database administrator, il quale possiede tutti i privilegi su tutte le risorse.

I privilegi sono:

- Insert
- Update
- Delete
- Select
- References: permette che venga fatto un riferimento a una risorsa nell'ambito della definizione dello schema di una tabella. Con questo privilegio l'utente può definire un vincolo di foreign key
- Usage: permette che venga usata la risorsa

5.3.2 Comandi per concedere e revocare privilegi

Grant Privilegi on Risorsa to
Utenti [with grant option]

il comando permette di concedere i privilegi sulla risorsa agli utenti.

La clausola “with grant option” specifica se deve essere concesso all’utente anche la possibilità di propagare il privilegio ad altri utenti- è possibile utilizzare al posto dei privilegi la parola chiave “all privileges”, che specifica tutti i privilegi possibili.

Revoke Privilegi on Risorsa from Utenti
[restrict | cascade]

Il comando revoke fa l'inverso. L'opzione restrict è il valore di default e specifica che il comando non deve essere eseguito

qualora la revoca dei privilegi all’utente comporti qualche altra revoca di privilegi, l’opposto si ottiene invece con l’opzione cascade.

Create role NomeRuolo

Si possono anche creare ruoli con il comando create role. Il ruolo si comporta come una sorda di contenitore di privilegi, che vengono attribuiti tramite il comando di grant. Per fruire dei privilegi è necessario che l'utente invochi un esplicito comando set role.

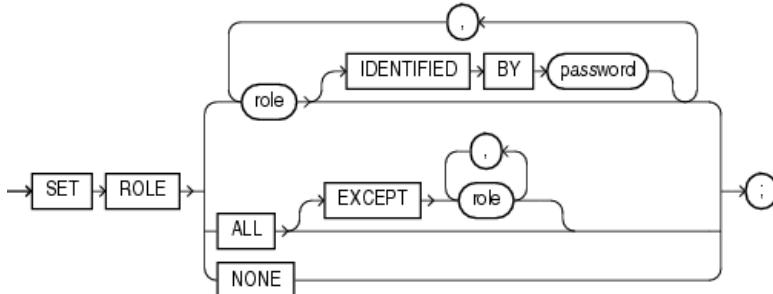


Figura 3 - set role

5.4 TRANSAZIONI

Una transazione identifica una unità elementare di lavoro svolta da un'applicazione, cui si vogliono associare particolari caratteristiche di correttezza, robustezza e isolamento. Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni viene detto "sistema transazionale".

```

Start transaction;
update ContoCorrente
    set Ammontare = Ammontare +10
    where NumConto = 12202;
update ContoCorrente
    set Ammontare = Ammontare -10
    where NumContro = 42177;
commit work;
  
```

commenti elencati precedentemente.

Tutto il codice che viene eseguito all'interno di una transazione gode di proprietà particolari, le cosiddette proprietà "acide"⁵ delle transazioni:

- Atomicità
- Consistenza
- Isolamento
- Persistenza

5.4.1 Atomicità

L'atomicità rappresenta il fatto che una transazione è un'unità indivisibile di esecuzione: o vengono resi visibili tutti gli effetti di una transazione, oppure la transazione non deve avere alcun effetto sulla base di dati.

L'inizio di una transazione è rappresentato dal comando "start transaction". Il termine della transazione è rappresentato da due istruzioni particolari:

- Commit work
- Rollback work

L'effetto di rollback è quello di annullare gli effetti del lavoro svolto dalla transazione, indipendentemente dalla sua complessità.

Viene detta ben formata una transazione iniziata da start transaction nel corso della cui esecuzione viene invocato uno solo dei due

⁵ "ACID properties" che sta per "Atomicity, Consistency, Isolation, Durability". Che traduzione del cazzo.

Nel caso la transazione non vada a compimento, il sistema deve essere in grado di ricostruire la situazione precedente (undo). Viceversa, dopo l'esecuzione del commit, il sistema deve assicurare che la transazione lasci la base di dati nel suo stato finale, ciò può comportare di dover rifare il lavoro svolto (redo).

Quando viene eseguito il comando rollback work, la situazione è simile a un “suicidio” autonomamente deciso nell’ambito della transazione. Viceversa, il sistema può decidere che la transazione non può essere portata a corretto compimento e “uccidere” la transazione.

5.4.2 Consistenza

La consistenza richiede che l'esecuzione della transazione non violi i vincoli di integrità definiti sulla base di dati.

5.4.3 Isolamento

L'isolamento richiede che l'esecuzione di una transazione sia indipendente dalla contemporanea esecuzione di altre transazioni.

L'isolamento si pone come obiettivo anche di rendere l'esito di ciascuna transazione indipendente da tutte le altre, si vuole cioè impedire che l'esecuzione di un rollback di una transazione causi l'esecuzione del rollback di altre transazioni, eventualmente generando una reazione a catena.

5.4.4 Persistenza

La persistenza richiede che l'effetto di una transazione che ha eseguito il commit correttamente non venga più perso.

In pratica, una base di dati deve garantire che nessun dato venga perso per nessun motivo.

6 SQL PER LE APPLICAZIONI

6.1 PROCEDURE

```

Procedure CambiaCittaATutti (:NuovaCitta
varchar(20), :VecchiaCitta
varchar(20))

Begin
    Update Dipartimento
    Set Città = :NuovaCitta
    Where Città = :VecchiaCitta
    Update Impiegato
    Set Città = :NuovaCitta
    Where Città = :VecchiaCitta
End;

```

Figura 4 - procedura SQL che aggiorna il nome della città di un dipartimento

```

Procedure CambiaCittaADip (:NuovaCitta varchar(20),
:VecchiaCitta varchar(20))

If not exists (select *
    From Dipartimento
    Where Nome = :NomeDip)
    Insert into ErroriDip values(:NomeDip)

Else
    Update Dipartimento
    Set Città = :NuovaCitta
    Where Nome = :NomeDip;
end if;

End;

```

6.2 TRIGGER

I trigger, detti anche “regole attive”, rappresentano una funzionalità particolarmente significativa dei moderni sistemi relazionali.

Seguono il paradigma Evento-Condizione-Azione (ECA): ogni trigger si attiva quando occorre uno specifico evento all’interno della base di dati, se è soddisfatta una data condizione, anche il trigger esegue un’azione stabilita.

sono dette anche stored procedures per il fatto che normalmente vengono memorizzate all’interno della base di dati come parti dello schema. Alle procedure può essere assegnato un nome.

La procedura può essere invocata avendo cura di associare un valore ai parametri.

L’uso di queste funzioni è fuori dallo standard e rende non portabile il codice SQL generato.

Una delle estensioni normalmente fornite dagli attuali sistemi relazionali è la struttura di controllo if-then-else

```

Create trigger ImpiegatiSenzaDip
After insert into Impiegati
For each row
When (new.Dipart is null)
Update Impiegati
    Set Dipart = 'NuoviArrivati'
    Where Matr = new.Matr

```

questo trigger viene attivato tutte le volte che il sistema rileva l'inserimento di tuple all'interno della tabella Impiegati.

Il principale problema che si presenta a chi vuole sfruttare il potenziale dei trigger è dato dalla complessità del comportamento di un sistema composto da molteplici trigger. Infatti, è possibile che si creino catene di attivazione difficili da analizzare.

6.3 SQL EMBEDDED

L'integrazione del linguaggio SQL con i normali linguaggi di programmazione di alto livello presenta alcuni ostacoli. SQL è un linguaggio molto ricco, con una propria sintassi, per questo sono state proposte due soluzioni per consentirne l'uso all'interno di un normale linguaggio di programmazione:

- Incastonamento (SQL embedded)
- CLI (Call Level Interface)

L'incastonamento prevede di introdurre direttamente nel programma sorgente scritto nel linguaggio di alto livello le istruzioni SQL, distinguendole dalle normali istruzioni tramite un opportuno separatore.

Lo standard SQL prevede che il codice SQL sia preceduto dalla stringa "exec sql" e temini con ";".

```

1) #include <stdlib.h>
2) Main()
3) {
4)     Exec sql begin declare section;
5)     char *NomeDip = "Manutenzione";
6)     char *CittaDip = "Pisa";
7)     int NumeroDip = 20;
8)     exec sql end declare section;

9)     exec sql connect to utente@librodb;
10)    if (sqlca.sqlcode != 0) {
11)        printf("Connessione al DV non riuscita\n");
12)    else {
13)        exec sql insert into Dipartimento
14)            values(:NomeDip, :CittaDip, :NumeroDip);
15)        exec sql disconnect all;
16)    }
17) }
```

Figura 5 - Un programma C con SQL Embedded

Questo tipo di sistema viene gestito da un preprocessore che riconosce le istruzioni SQL e predisporrà l'insieme opportuno di strutture ausiliarie richieste per la loro esecuzione.

Si può osservare come le dichiarazioni di variabili del programma C siano racchiuse tra "begin declare section" e "end declare section", questo consente di utilizzare le variabili del programma come parametri

per i comandi SQL. Il meccanismo che viene offerto per l'uso delle variabili è molto agevole e richiede solamente di far precedere il nome della variabile dal carattere di due punti.

Un importante problema che caratterizza l'integrazione tra SQL e i normali linguaggi di programmazione è il cosiddetto problema del conflitto d'impedenza (impedance mismatch). I linguaggi di programmazione accedono agli elementi di una tabella scandendone le righe una a una, utilizzando quello che viene detto un approccio tuple-oriented. Al contrario, SQL è un linguaggio di tipo set-oriented, che opera su intere tabelle, non su singole righe.

Questo problema ammette due soluzioni:

- Utilizzo dei cursori
- Costruttori di dati e la gestione naturale di un insieme di righe

6.3.1 Cursori

Un cursore è uno strumento che permette a un programma di accedere alle righe di una tabella una alla volta, il cursore viene definito su una generica interrogazione.

```
Declare NomeCursore [scroll] cursor for SelectSQL
[for <read only | update [of Attributo {, Attributo} ] > ]
```

l'opzione scroll specifica se si vuole permettere al programma di muoversi liberamente sul risultato dell'interrogazione.

L'opzione finale for update specifica se il cursore deve essere utilizzato nell'ambito di un comando di modifica, permettendo di specificare eventualmente gli attributi che saranno oggetto del comando di update.

Open NomeCursore	Il comando open ha come argomento un cursore. Al momento dell'esecuzione del comando open, viene seguita l'interrogazione associata al cursore e il risultato diventa accessibile tramite l'istruzione fetch.
------------------	---

```
Fetch [Posizione from]
NomeCurosre into ListaDiFetch
```

il comando fetch prende una riga dal cursore e la ripone nelle variabili del programma che compaiono in ListaDiFetch. In ListaDiFetch vi sarà una variabile per ogni elemento della target list dell'interrogazione.

Il parametro posizione permette di specificare quale riga dovrà essere oggetto dell'operazione di fetch; il parametro può assumere i valori:

- Next: riga successiva alla corrente
- Prior: riga precedente alla corrente
- First: prima riga del risultato
- Last: ultima riga del risultato
- Absolute EspressioneIntera: la riga che compare in posizione i-esima nel cursore, se i è il risultato della valutazione dell'espressione
- Relative EspressioneIntera: come assoluto, solo che viene preso come punto di riferimento la posizione della riga corrente

Queste opzioni sono utilizzabili a condizione che sia stata specificata al momento della definizione del cursore l'opzione scroll, altrimenti l'unico parametro possibile è next.

```
Update NomeTabella
```

```
Set Attributo = <Espressione | null |default >
{, Attributo = <Espressione | null |default >}
Where current of NomeCursore
```

l'unica differenza qua
accanto è la presenza
nella clausola where
del predicato
current of
NomeCursore che
identifica la riga

corrente (affinchè venga aggiornata o rimossa).

```
close NomeCursore
```

Il comando close chiude il cursore, ovvero comunica al sistema che il risultato dell'interrogazione non serve più.

```
Declare CursoreImpiegati scroll cursor for
```

```
Select Cognome, Nome, Stipendio
From Impiegato
```

```
Where Stipendio > 40 and Stipendio < 100
```

Qua accanto una semplice dichiarazione di cursore.

6.3.2 SQL dinamico

In diverse situazioni sorge la necessità di permettere all'applicazione di definire al momento dell'esecuzione le interrogazioni da effettuare sulla base di dati.

Se le interrogazioni hanno una struttura predefinita e ciò che varia solamente il valore dei parametri usati nell'interrogazione, allora diventa possibile costruire una applicazione che gestisce le tipologie di interrogazione richieste dall'utente sfruttando il meccanismo di passaggio dei parametri dall'ambiente del programma al sistema di gestione di basi di dati.

In altri casi però l'utente ha bisogno di effettuare delle interrogazioni caratterizzate da estrema variabilità, con la necessità di definire al momento dell'esecuzione del programma non solo i valori dei parametri, ma anche la forma delle interrogazioni, l'insieme delle tabelle e la struttura del risultato.

I meccanismi di sql visiti fino ad ora non valgono per questo caso d'uso, per questo si ricorre a delle soluzioni con sql embedded che permettono un uso dinamico di sql.

L'uso di sql dinamico modifica la modalità di interazione con il sistema. Nel caso di sql statico, i comandi vengono gestiti da un preprocessore che analizza la struttura del comando. Sql dinamico offre due metodi di interazione:

- esecuzione diretta dell'interrogazione
- gestione in due fasi:
 - analisi
 - esecuzione

6.3.2.1 Esecuzione immediata

```
Execute immediate IstruzioneSQL
```

Mediante il comando di execute immediate si richiede l'esecuzione di un'istruzione SQL, specificata direttamente in un parametro di tipo stringa di

caratteri dell'ambiente del programma.

Questo modo può essere usato solo per comandi che non richiedono parametri né in ingresso né in uscita.

Quando un comando viene seguito più volte, o quando il programma deve gestire un o scambio di parametri di ingresso p do uscito con l'istruzione, diventa necessario distinguere le due fasi di preparazione e di esecuzione.

6.3.2.2 Fase di preparazione

```
Prepare NomeComando from Istruzione SQL
```

```
Prepare :comando from "select Città from Dipartimento where Nome = ?"
```

Il comando prepare analizza un'istruzione SQL e le traduce nel linguaggio procedurale. In questo caso l'istruzione può contenere dei parametri in ingresso, rappresentati dal carattere di punto interrogativo.

tradotta non serve più, è possibile rilasciare la memoria occupata dalla traduzione dell'istruzione utilizzando il comando di deallocate prepare.

6.3.2.3 Fase di esecuzione

```
Execute NomeComando [into ListaTarget] [using ListaParametri]
```

```
Execute :comando into :città using :dipartimento
```

per eseguire un comando preallocato si usa il comando execute.

La lista dei target contiene l'elenco dei parametri in cui deve essere scritto il risultato dell'esecuzione del comando. La lista dei parametri specifica quali sono i valori da usare.

6.3.2.4 Cursori con SQL dinamico

```
Prepare :comando from :istruzioneSQL
declare Cursore cursor for :comando
```

```
Open Cursore using :nome1
```

L'uso dei cursori con SQL dinamico è molto simile all'uso dei cursori che viene fatto da SQL statico. Le uniche due differenze sono:

- si associa al cursore l'identificativo dell'interrogazione invece che l'interrogazione stessa

• i comandi d'uso del cursore ammettono la specifica delle clausole into e using

6.4 CALL LEVEL INTERFACE (CLI)

Una soluzione alternativa consiste nel mettere direttamente a disposizione del programmatore un insieme di funzioni che permettano di interagire con il DBMS. Questa famiglia di soluzioni va sotto il nome di Call Level interface (CLI), in quanto il programmatore dispone direttamente di una libreria di funzioni per realizzare il dialogo con la base di dati. In confronto a embedded, la CLI è più flessibile, meno integrato con il linguaggio di programmazione, ma ha l'inconveniente di dover gestire esplicitamente aspetti che in SQL embedded vengono risolti esplicitamente dal preprocessore.

Diversi sistemi offrono delle proprie CLI. Il modo generale d'uso di questi strumenti è il seguente:

- Si utilizza un servizio della LCI per creare una connessione con il DBMS
- Si invia sulla connessione un comando SQL che rappresenta la richiesta

ODBC	Interfaccia standard che permette di accedere a basi di dati in qualunque contesto, realizzando interoperabilità con diverse combinazioni di DBMS-Sist.Op.-Reti
OLE DB	Soluzione proprietaria Microsoft, basata sul modello COM, che permette ad applicazioni Windows di accedere a sorgenti dati generiche (non solo DBMS)
ADO	Soluzione proprietaria Microsoft che permette di sfruttare i servizi OLE DB, utilizzando un'interfaccia record-oriented
ADO.NET	Soluzione proprietaria Microsoft che adatta ADO alla piattaforma .NET; offre un'interfaccia set-oriented e introduce i <i>DataAdapter</i>
JDBC	Soluzione per l'accesso ai dati in Java sviluppata da Sun Microsystems; offre in quel contesto un servizio simile a ODBC

- Si riceve come risposta del comando una struttura relazionale in un opportuno formato. La CLI dispone di un certo insieme di primitive che permettono di analizzare e descrivere la struttura del risultato del comando
- Al termine della sessione di lavoro, si chiude la connessione e si rilasciano le strutture dati utilizzate per la gestione del dialogo

6.4.1 ODBC e soluzioni proprietarie Microsoft

6.4.1.1 ODBC

Open DataBase Connectivity è un'interfaccia applicativa proposta originariamente dalla Microsoft e diventata in seguito uno standard.

ODBC è un SQL ristretto, caratterizzata da un insieme limitato di istruzioni, definito nell'ambito del comitato SQL Access Group (SAG).

nell'architettura ODBC il collegamento da un'applicazione e un server richiede l'uso di un driver, una libreria che viene collegata dinamicamente alle applicazioni e viene da esse invocata. Il driver maschera le differenze di interazione legate non solo al DBMS, ma anche al sistema operativo e al protocollo di rete utilizzato.

Pertanto, per garantire la compatibilità rispetto allo standard ODBC, ciascun venditore deve garantire dei driver che prevedano l'uso del DBMS nell'ambito di una specifica rete e con uno specifico sistema operativo.

l'accesso a una base di dati tramite ODBC richiede la cooperazione di quattro componenti di sistema:

- l'applicazione richiama le funzioni SQL per eseguire interrogazioni e per acquisirne i risultati. La scelta del protocollo di comunicazione, del server DBMS e del sistema operativo del nodo dove è installato il DBMS sono tutti trasparenti per l'applicazione, essendo mascherati dal driver
- il driver manager è responsabile di carica i driver a richiesta dell'applicazione. Questo software, fornito direttamente dalla Microsoft per il mondo Windows, garantisce anche alcune funzioni aggiuntive
- i driver sono responsabili di eseguire funzioni ODBC, pertanto sono in grado di eseguire interrogazioni in SQL, traducendole in modo da adattarsi alla sintassi e alla semantica degli specifici prodotti cui viene fatto accesso
- la fonte di informazione è il sistema che esegue le funzioni trasmesse dal client.

I codici di errore sono standardizzati. Le interrogazioni possono essere specificate in modo statico o dinamico.

Data la sua complessità, ODBC non è adatto a essere utilizzato direttamente dal programmatore per lo sviluppo di applicazioni. Per questo Microsoft ha sviluppato delle soluzioni proprietarie, che semplificano la realizzazione di applicazioni che fanno accesso ai servizi di una base di dati.

6.4.1.2 OLE DB

Object Linking and Embedding for DataBase è un'interfaccia generale, che permette di accedere a sorgenti dati di tipo generico: non solo sistemi relazionali, ma anche una vasta tipologia di archivi di dati, come per esempio caselle di posta elettronica.

6.4.1.3 ADO

ActiveX Data Object costituisce un'interfaccia di alto livello ai servizi offerti d OLE DB. Il modello di ADO si basa sulla definizione di quattro concetti fondamentali:

- la connessione
- il comando
- la tupla
- l'insieme di tuple

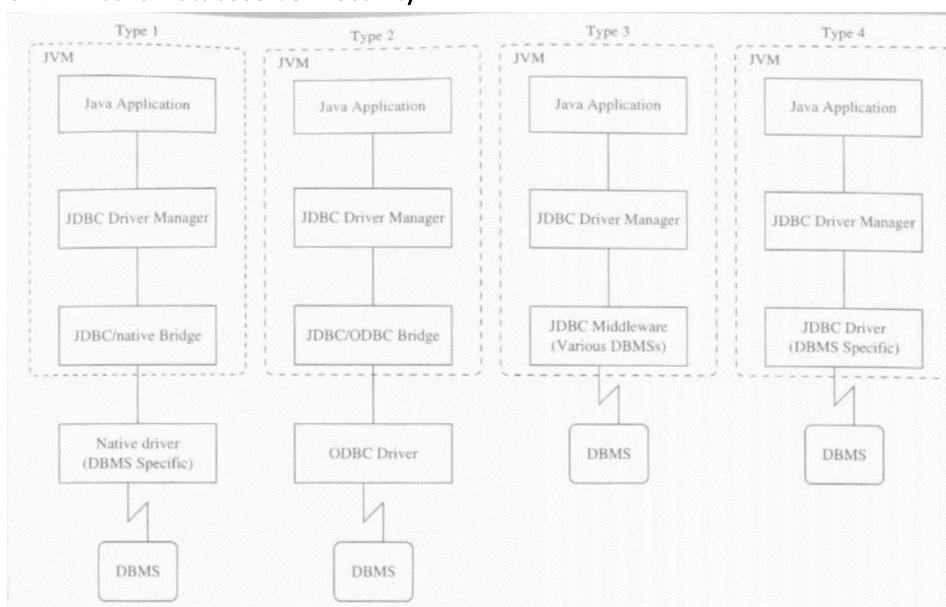
la connessione rappresenta il canale di comunicazione che deve essere stabilito per interagire con una sorgente dati, fornendo la locazione della sorgente dati nel sistema e normalmente informazioni di login. Una componente importante della connessione è la collezione di errori. Il comando rappresenta la stringa di caratteri che contiene l'istruzione SQL che si vuole fare seguire sulla sorgente dati. La tupla descrive la singola riga di una tabella e offre strumenti per riconoscere la struttura dei dati che compongono la tupla.

6.4.1.3 ADO.NET

Rappresenta l'evoluzione della soluzione ADO, progettata per operare all'interno della piattaforma .NET di Microsoft. Non è una semplice evoluzione, è il frutto di un'attività di riprogettazione che trae spunto dalle caratteristiche di .NET.

La differenza di fondo tra ADO.NET e ADO è il diverso approccio per la gestione del dialogo con la base di dati. In ADO l'oggetto centrale è il Record Set, il quale viene gestito tramite connessioni dirette con la base di dati. A differenza di ADO, l'accesso interno agli elementi dei dataset e datatable avviene utilizzando i normali meccanismi di accesso a collezioni di oggetti.

6.4.2 Java Database Connectivity



Tra i vari moduli Java è presente il modulo Java DataBase Connectivity (JDBC), il quale permette a programmi java di accedere in modo uniforme a basi di dati relazionali, in modo simile a quanto offerto dal modulo ODBC.

L'architettura prevede uno strato, costituito dal driver manager, il quale isola l'applicazione dal

componente responsabile di implementare il servizio. Vi sono molteplici architetture:

- Driver nativo: questa soluzione prevede la realizzazione di funzioni Java che convertono le richieste JDBC in richieste a un driver nativo della macchina.
- Ponte (bridge) JDBC/ODBC: traduce le richieste JDBC in richieste ODBC

Queste due soluzioni non sono realmente portabili, perché richiedono l'utilizzo di componenti nativi, cioè specifici dell'ambiente. Vi sono due soluzioni che si appoggiano solamente su Java:

- Middleware-server: questa soluzione prevede l'uso di un server scritto in Java responsabile di tradurre le richieste provenienti dal driver manager nel formato riconosciuto dal particolare DBMS che si intende utilizzare
- Driver Java: questa soluzione prevede l'uso di un driver specifico per il particolare sistema relazionale

JDBC rappresenta una soluzione molto interessante per la realizzazione di applicazioni portabili che facciano accesso a basi di dati. Per utilizzare i servizi di JDBC normalmente si seguono i passi seguenti:

- 1) Si carica il driver richiesto
- 2) Si crea una connessione con la base di dati
- 3) Si compone il comando SQL e lo si invia alla base di dati
- 4) Si gestisce il risultato del comando SQL

```
1) Import java.sql.*;
2) Public class PrimoJdbc {
3)     Public static void main(Stirng[] arg) {
4)         Connection conn = null;
5)         Try {
6)             // Caricamento del driver
7)             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8)             // Apertura della connessione
9)             Conn = DreiverManager.getConnectgio(
10)                 "jdbc:odbc:Corsi");
11)         }
12)         Catch (Exception e) {System.exit(1); }
13)         Try {
14)             // Esecuzione dell'interrogazione
15)             Statement interrogazione =
16)                 conn.createStatement();
17)             ResultSet risultato =
18)                 Interrogazione.executeQuery(
19)                     "select * from Corsi");
20)             While (risultato.next()) {
21)                 String nomeCorso =
22)                     Risultato.getString("NomeCorso");
23)                 System.out.println(nomeCorso);
24)             }
25)         }
26)         Catch (Exception e) {System.exit(1); }
27)     }
28) }
```

Tabella 1 - Esempio di utilizzo di driver di tipo 2

7 METODOLOGIE E MODELLI PER IL PROGETTO

7.1 INTRODUZIONE ALLA PROGETTAZIONE

7.1.1 Il ciclo di vita dei sistemi informativi



Il ciclo di vita di un sistema informativo comprende, generalmente, le seguenti attività:

- **Studio di fattibilità:** costi delle varie alternative e priorità di realizzazione delle componenti
- **Raccolta e analisi dei requisiti:** individuazione e studio delle proprietà e funzionalità che si vogliono avere
- **Progettazione:**
 - Progettazione dei dati: si individua la struttura e l'organizzazione dei dati
 - Progettazione delle applicazioni: si definiscono le caratteristiche dei programmi applicativi
- **Implementazione:** realizzazione del sistema informativo
- **Validazione e collaudo:** verifica del corretto funzionamento e la qualità del sistema informativo
- **Funzionamento:** il sistema informativo diventa operativo. A meno che non ci siano malfunzionamenti o revisioni, questa attività richiede solo gestione e manutenzione

Il processo non è quasi mai strettamente sequenziale.

7.1.2 Metodologie di progettazioni e basi di dati

Una metodologia di progettazione consiste in:

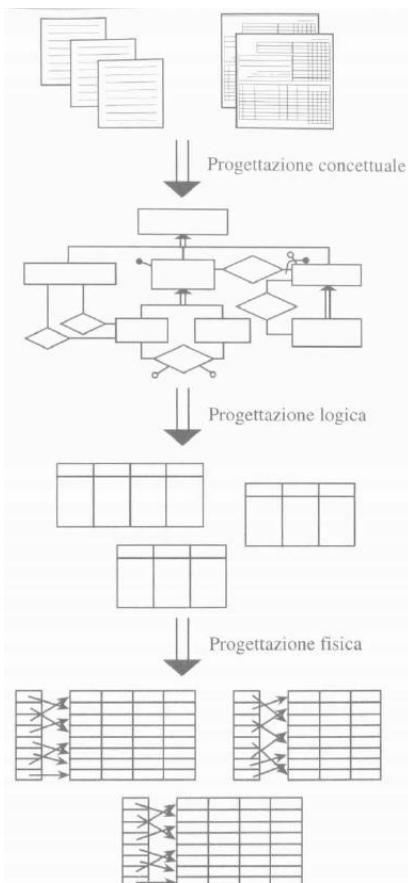
- Decomposizione dell'intera attività di progetto in passi successivi indipendenti tra loro
- Strategie da seguire nei vari passi e criteri per la scelta di alternative
- Modelli di riferimento per descrivere i dati di ingresso e uscita delle varie fasi

Le proprietà di una metodologia sono principalmente:

- Generalità rispetto alle applicazioni e ai sistemi in gioco
- Qualità del prodotto in termini di correttezza, completezza ed efficienza
- Facilità d'uso delle strategie e dei modelli

Nell'ambito delle basi di dati si è consolidata una metodologia in tre fasi:

- **Progettazione concettuale:** il suo scopo è rappresentare le specifiche informali della realtà di interesse in termini di una descrizione formale e completa, ma indipendente dai criteri di rappresentazione utilizzati. Il prodotto di questa fase viene chiamato "schema concettuale"
- **Progettazione logica:** traduzione dello schema concettuale in termini del modello di rappresentazione dei dati adottato dal sistema di gestione di base di dati a disposizione. Il prodotto di questa fase viene denominato schema logico e fa riferimento al modello logico dei dati.



criteri di organizzazione dei dati.

7.2.1 I costrutti principali del modello

Ci sono tre costrutti principali:

- Entità
- Relazioni
- Attributi

7.2.1.1 Entità



Rappresentano classi di oggetti che hanno proprietà comuni ed esistenza "autonoma".

Una occorrenza⁶ di un'entità è un oggetto della classe che l'entità rappresenta.

Esempio:

Figura 6 - Esempi di entità

Entità	Occorrenza
CITTÀ	Città di Roma, Città di Milano...
IMPIEGATO	Impiegato Rossi, Impiegato Verdi...

Notare che un'occorrenza di un'entità non è un valore che identifica un oggetto (ad esempio Roma o Milano) ma è l'oggetto stesso (la vera e propria città di Roma). Una conseguenza di ciò è che l'occorrenza ha un'esistenza indipendente dalla proprietà ad essa associate. Questa è una delle differenze principali

⁶ In genere si usa il termine "istanza" invece di occorrenza.

rispetto al modello relazionale dove non possiamo rappresentare un oggetto senza conoscere alcune sue proprietà.

In uno schema, ogni entità ha un nome che la identifica univocamente e viene rappresentata mediante un rettangolo con il nome dell'entità all'interno.

7.2.1.2 Relazioni (o associazioni)

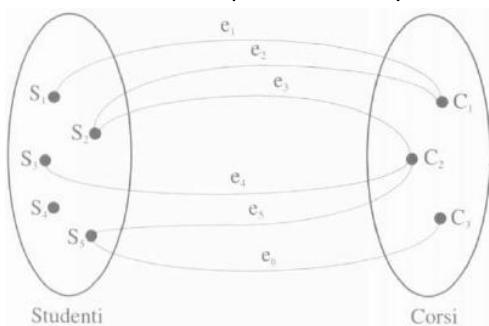


Figura 7 - Esempi di occorrenza di relazione

Rappresentano legami logici, significativi per l'applicazione di interesse, tra due o più entità.

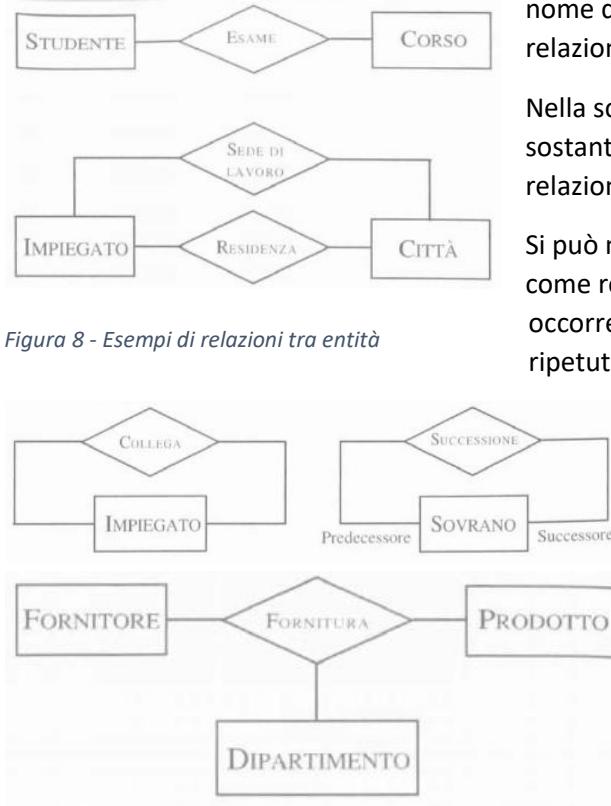
RESIDENZA è un esempio di entità che può sussistere tra le entità CITTÀ e IMPIEGATO.

Una occorrenza di relazione è una ennupla costituita da occorrenze di entità, una per ciascuna delle entità coinvolte.

Ogni relazione ha un nome che la identifica univocamente e viene rappresentata graficamente mediante un rombo, con il nome della relazione all'interno, e da linee che connettono la relazione con ciascuna delle sue componenti.

Nella scelta dei nomi delle relazioni è preferibile usare sostantivi e non verbi per non assegnare un "verso" alle relazioni.

Si può notare dalla figura 2 che le relazioni sono rappresentabili come relazioni matematiche, questo significa che tra le occorrenze di una relazione non ci possono essere ennuple ripetute.



7.2.1.3 Attributi

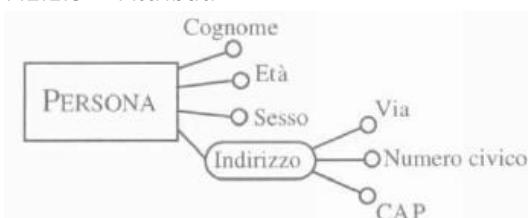


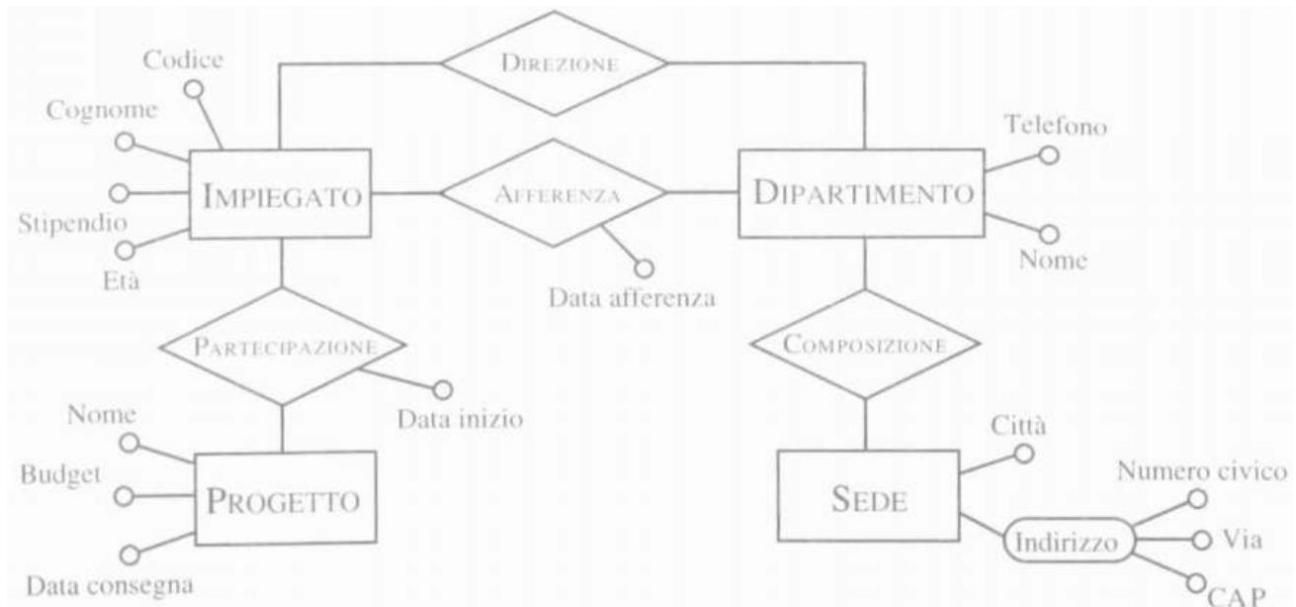
Figura 9 - Un esempio di attributo composto

Descrivono le proprietà elementari di entità o relazioni che sono di interesse ai fini dell'applicazione. Un attributo associa a ciascuna occorrenza di entità (o di relazione) un valore appartenente a un insieme, detto dominio, che contiene i valori ammissibili per l'attributo.

Attributi che presentano affinità nel loro significato o uso. L'insieme di

attributi viene detto "attributo composto".

7.2.1.4 Costruzione di schemi con i costruttori di base



Partendo dall'entità sede:

- Una sede dell'azienda è dislocata in una certa città e ha un certo indirizzo
- Ogni sede è organizzata in dipartimenti e ogni dipartimento ha un nome e un numero di telefono
- A questi dipartimenti afferiscono impiegati in una certa data e ci sono impiegati che dirigono tali dipartimenti
- Ogni impiegato ha cognome, stipendio, età e un codice per identificarli
- Gli impiegati lavorano su progetti a partire da una certa data
- Ogni progetto ha un nome, un budget e una data di consegna

7.2.2 Altri costrutti del modello

I rimanenti costrutti sono:

- Cardinalità delle relazioni e degli attributi
- Identificatori delle entità
- Generalizzazioni

I primi due costituiscono vincoli di integrità su costrutti già visti.

7.2.2.1 Cardinalità delle relazioni

Vengono specificate per ciascuna partecipazione di entità a una relazione e descrivono il numero minimo e massimo di occorrenze di relazione a cui una occorrenza dell'entità può partecipare.



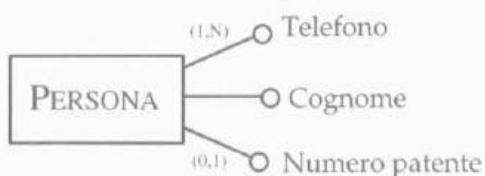
L'esempio qua sopra mostra che un impiegato può partecipare massimo a cinque incarichi mentre un incarico può avere tra 0 e 50 impiegati.

In linea di massima è possibile assegnare un qualunque intero non negativo a una cardinalità di una relazione con l'unico vincolo che la cardinalità minima deve essere minore o uguale a quella massima.

In realtà, generalmente, si usano solo tre cardinalità: 0, 1, N:

- Cardinalità minima, 0 o 1:
 - 0: partecipazione opzionale
 - 1: partecipazione obbligatoria
- Cardinalità massima, 1 o molti (N):
 - 1: partecipazione vista come una funzione, parziale se la cardinalità minima vale 0
 - N: numero arbitrario di occorrenze

7.2.2.2 Cardinalità degli attributi



Descrivono il numero minimo e massimo di valori dell'attributo associati a ogni occorrenza di entità o relazione. La cardinalità (1, 1) viene omessa.

7.2.2.3 Identificatori delle entità



Figura 10 - Esempio di identificatore

Vengono specificati per ciascuna entità di uno schema e descrivono i concetti che permettono di identificare in maniera univoca le occorrenze delle entità.

Alcune volte però gli attributi di un'entità non sono sufficienti a identificare univocamente le sue occorrenze, ad esempio due studenti potrebbero avere due matricole uguali ma essere iscritti a due università differenti.

In questo caso l'identificatore deve comprendere sia l'attributo matricola sia la relazione tra lo studente e l'università.

Attenzione, l'identificazione univoca tramite relazione solo se l'entità coinvolta nella relazione ha cardinalità (1,1). Nei casi



Figura 11 - Esempio di identificatore esterno

in cui si legano attributi e relazioni si parla di "identificatore esterno".

Quindi:

- Un identificatore può coinvolgere uno o più attributi, ognuno dei quali deve avere cardinalità (1,1)
- Un'identificazione esterna può coinvolgere una o più entità, ognuna delle quali deve essere membro di una relazione alla quale l'entità da identificare partecipa con cardinalità (1,1)
- Un'identificazione esterna può coinvolgere un'entità che è a sua volta identificata esternamente, purché non vengano generati cicli di identificazioni esterne
- Ogni entità deve avere almeno un identificatore (interno o esterno) ma ne può avere anche più di uno, in questo caso gli attributi e le entità in alcune identificazioni (tranne una) possono essere opzionali

7.2.2.4 Generalizzazioni

Rappresentano legami logici tra un'entità E, detta entità genitore, e una o più entità dette entità figlie di cui E è più generale nel senso che le comprende come casi particolari.

Ad esempio, PERSONA è una generalizzazione di UOMO, DONNA.

Valgono le seguenti proprietà:

- Ogni occorrenza di un'entità figlia è anche una occorrenza dell'entità genitore

- Ogni proprietà dell'entità genitore è anche una proprietà delle entità figlie. Questa proprietà delle generalizzazioni è nota come "ereditarietà".

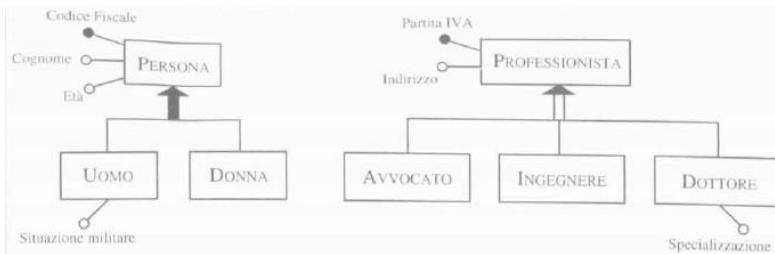
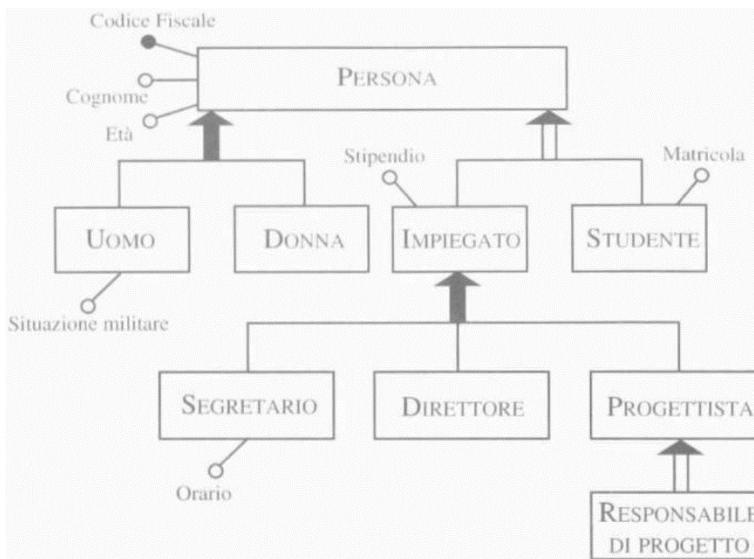


Figura 12 - Esempi di generalizzazione, esclusiva/totale ed esclusiva/parziale

- occorrenza di almeno una delle entità figlie, altrimenti è parziale
- Generalizzazione esclusiva:** ogni occorrenza dell'entità genitore è al più un'occorrenza di una delle entità figlie, altrimenti è sovrapposta



Le generalizzazioni vengono rappresentate graficamente mediante delle frecce che congiungono le entità figlie con l'entità genitore. Notare che le proprietà eritate non vanno rappresentate esplicitamente.

- Generalizzazione totale:** ogni occorrenza dell'entità genitore è una

Notare come le generalizzazioni sovrapposte possano diventare esclusive tramite l'utilizzo delle intersezioni, ad esempio STUDENTE o LAVORATORE genera una generalizzazione sovrapposta, l'inclusione di STUDENTELAVORATORE crea una generalizzazione esclusiva.

Ci possono essere gerarchie di generalizzazioni e ci sono generalizzazioni con una sola entità figlia, in questo caso si parla di sottoinsieme.

7.2.3 Panoramica finale sul modello E-R

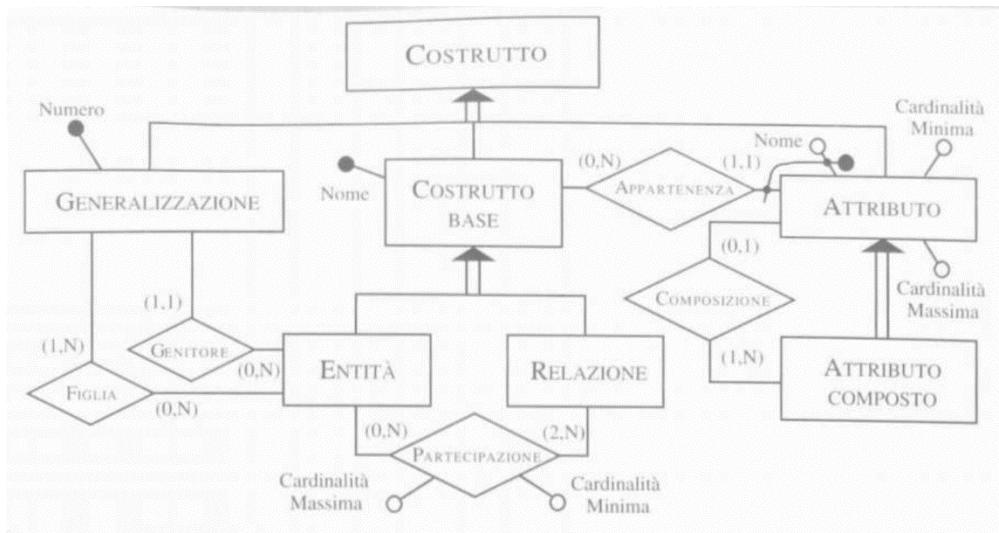


Figura 13 – Descrizione del modello E-R tramite modello E-R

Gli schemi E-R possono essere utilizzati a scopo documentativo. Possono essere utilizzati per descrivere i dati di un sistema informativo già esistente e, nel caso di sistema costituito da diversi sottosistemi, c'è il vantaggio di poter rappresentare le varie componenti con un linguaggio astratto. Infine, possono essere usati per comprendere, in caso di modifica dei requisiti di un'applicazione, su quali porzioni del sistema si deve operare e in cosa consistono le modifiche da effettuare.

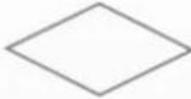
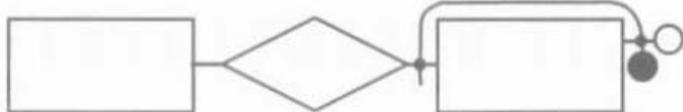
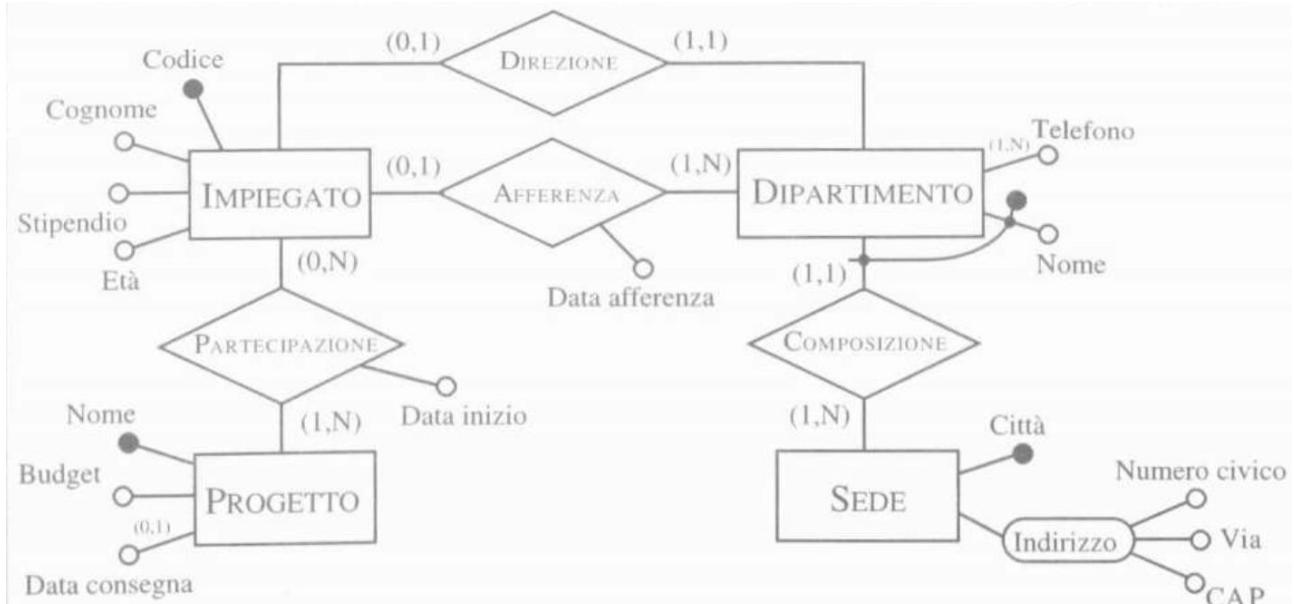
Costrutti	Rappresentazione grafica
Entità	
Relazione	
Attributo semplice	
Attributo composto	
Cardinalità di relazione	
Cardinalità di attributo	
Identificatore interno	 
Identificatore esterno	
Generalizzazione	
Sottoinsieme	

Figura 14 - Rappresentazioni grafiche del modello E-R

7.3 DOCUMENTAZIONI DI SCHEMI E-R

Uno schema E-R non è quasi mai sufficiente da solo a rappresentare nel dettaglio tutti gli aspetti di un'applicazione.



prendiamo come esempio questo schema, sarebbe difficile inserire altri attributi per IMPIEGATO senza ridurre la leggibilità dello schema. PROGETTO si riferisce ai progetti interni o esterni?

In più ci sono alcune informazioni che col modello E-R sono impossibili da rappresentare, come ad esempio immaginiamo che un impiegato possa essere direttore solo di un dipartimento al quale afferisce.

In conclusione, risulta indispensabile corredare ogni schema E-R con una documentazione di supporto per facilitare l'interpretazione dello schema stesso e a descrivere proprietà dei dati rappresentati che non possono essere espressi direttamente dai costrutti del modello.

7.3.1 Regole aziendali

Una regola aziendale può essere:

- La descrizione di un concetto rilevante per l'applicazione, ovvero la definizione precisa di un'entità, di un attributo o di una relazione del modello E-R
- Un vincolo di integrità sui dati dell'applicazione, sia esso la documentazione di un vincolo espresso con qualche costrutto del modello E-R o la descrizione di un vincolo non esprimibile direttamente con i costrutti del modello
- Una derivazione, ovvero un concetto ottenuto attraverso inferenza o calcolo aritmetico da altri concetti dello schema

Per le regole del primo tipo è impossibile definire una sintassi precisa e si fa ricorso al linguaggio naturale.

Le regole che descrivono vincoli di integrità e derivazioni sono più adatte a definizioni formali. In particolare, le regole che descrivono vincoli di integrità possono essere espresse sotto forma di asserzioni, ovvero affermazioni che devono sempre essere verificate. Si esprimono nella forma:

<concetto> deve/non deve <espressione su concetti>

Ad esempio "il direttore di un dipartimento deve afferire a tale dipartimento".

Consideriamo ora le derivazioni. Una possibile struttura è:

<concetto> si ottiene <operazione su concetti>

Ad esempio “il numero degli impiegati di un dipartimento si ottiene contando gli impiegati che vi afferiscono”.

7.3.2 Tecniche di documentazione

La documentazione dei vari concetti rappresentati in uno schema, ovvero le regole aziendali di tipo descrittivo, può essere prodotta facendo uso di un dizionario dei dati. Esso è composto da due tabelle:

- La tabella delle entità (Entità | Descrizione | Attributi | Identificatore)
- La tabella delle relazioni (Relazione | Descrizione | Entità coinvolte | Attributi)

Per quanto riguarda le regole aziendali si può far ricorso ancora ad una tabella, nella quale vengono elencate le varie regole dividendole per tipologia:

- Regole di vincolo
- Regole di derivazione

Entità	Descrizione	Attributi	Identificatore
Impiegato	Impiegato che lavora nell'azienda.	Codice, Cognome, Stipendio, Età	Codice
Progetto	Progetti aziendali sui quali lavorano gli impiegati.	Nome, Budget, Data consegna	Nome
Dipartimento	Dipartimenti delle sedi dell'azienda.	Telefono, Nome	Nome, Sede
Sede	Sede dell'azienda in una certa città.	Città, Indirizzo (Numero, Via e CAP)	Città

Relazione	Descrizione	Entità Coinvolte	Attributi
Direzione	Associa un dipartimento al suo direttore.	Impiegato (0,1), Dipartimento (1,1)	
Afferenza	Associa un impiegato al suo dipartimento.	Impiegato (0,1), Dipartimento (1,N)	Data afferenza
Partecipazione	Associa agli impiegati i progetti sui quali lavorano.	Impiegato (0,N), Progetto (1,N)	Data inizio
Composizione	Associa una sede ai dipartimenti di cui è composta.	Dipartimento (1,1), Sede (1,N)	

Regole di vincolo
(RV1) Il direttore di un dipartimento deve afferire a tale dipartimento.
(RV2) Un impiegato non deve avere uno stipendio maggiore del direttore del dipartimento al quale afferisce.
(RV3) Un dipartimento con sede a Roma deve essere diretto da un impiegato con più di dieci anni di anzianità.
(RV4) Un impiegato che non afferisce a nessun dipartimento non deve partecipare a nessun progetto.
Regole di derivazione
(RD1) Il budget di un progetto si ottiene moltiplicando per 3 la somma degli stipendi degli impiegati che vi partecipano.

7.4 MODELLAZIONE DEI DATI IN UML

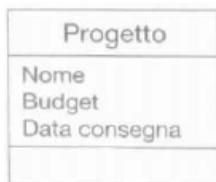
A causa del suo successo, UML viene talvolta utilizzato, in alternativa al modello E-R, per la rappresentazione concettuale di una base di dati. In particolare, vengono utilizzati i diagrammi delle classi che descrivono le classi di oggetti di interesse per l'applicazione e le relazioni che intercorrono tra di esse.

UML può rappresentare aspetti che il modello E-R non può evidenziare. Al contrario, alcuni costrutti del modello E-R non sono rilevanti nella modellazione UML, quindi possono essere rappresentati solo tramite accordi personali tra i progettisti.

7.4.1 Rappresentazione di dati con i diagrammi delle classi

Ci soffermiamo sui costrutti che si possono usare per descrivere, a livello concettuale, una base di dati.

7.4.1.1 Classi



Sono le componenti principali dei diagrammi delle classi e corrispondono alle entità del modello E-R.

A differenza del modello E-R, per una classe è possibile specificare nel riquadro in basso anche le operazioni eseguibili sull'entità. In UML si possono associare gli attributi ai rispettivi domini (interi, reali, stringhe, ecc.)

7.4.1.2 Associazioni



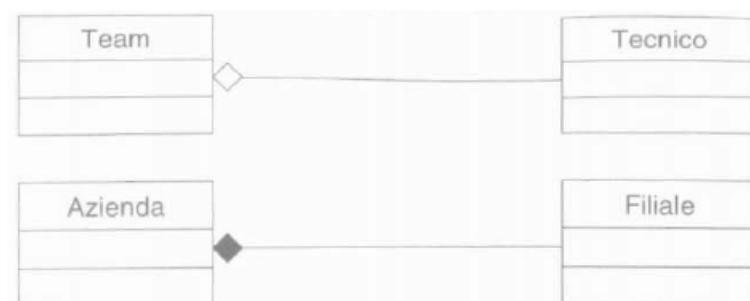
Corrispondono alle relazioni del modello E-R.

Il nome delle relazioni viene posto sulla linea.

Non è possibile assegnare attributi alle relazioni, per questo si fa uso delle cosiddette "classi di associazione" che descrivono proprietà di un'associazione da descrivere. Semplicemente la relazione è descritta da una classe intermedia come nell'esempio della fornitura qua accanto.

Per le associazioni è possibile specificare una serie di proprietà, ad esempio la navigabilità o la possibilità di specificare associazioni che sono aggregazioni di concetti, cioè associazioni che definiscono una relazione tra un concetto composito e uno o più concetti che ne costituiscono una sua parte.

Figura 15 - Esempio di classe di associazione



7.4.1.3 Molteplicità

È possibile indicare la cardinalità di partecipazione delle classi di associazioni. Le convenzioni adottate nei due formalismi sono differenti:



- Le cardinalità minima e massima vengono separate da .. e non da ,
- La cardinalità molti (N) viene rappresentata tramite *
- La cardinalità di default è 1 a meno che non si parli di aggregazioni, dove per la classe aggregante è *
- La cardinalità di una classe è scritta non accanto alla stessa ma accanto all'altra classe coinvolta nella relazione

7.4.1.4 Identificatori

In UML non esiste una notazione per esprimere identificatori di classi. Siccome però gli identificatori sono indispensabili nel modello relazionale si è arrivati ad una soluzione, ovvero usare un costrutto chiamato vincolo utente.

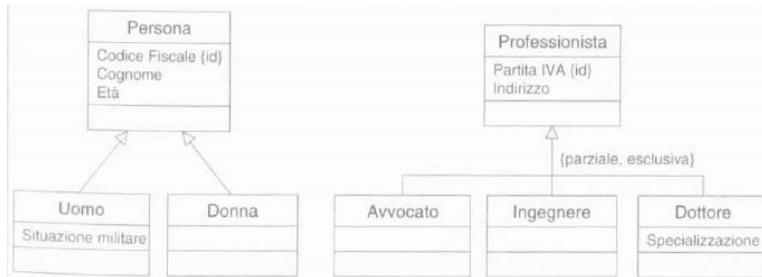


In UML si possono definire vincoli di integrità su associazioni e su attributi specificandoli tra parentesi graffe vicino all'elemento oggetto del vincolo. È possibile definire liberamente vincoli propri (detti vincoli utente).

Semplicemente accanto gli attributi necessari per l'identificazione si scrive `{id}`. con questa notazione è possibile specificare solo un identificatore per classe.

Per quel che riguarda le identificazioni esterne, dato che sintatticamente non è possibile usare un vincolo, è pratica ricorrere ad uno stereotipo, descrivibile tramite un nome racchiuso tra `<>`.

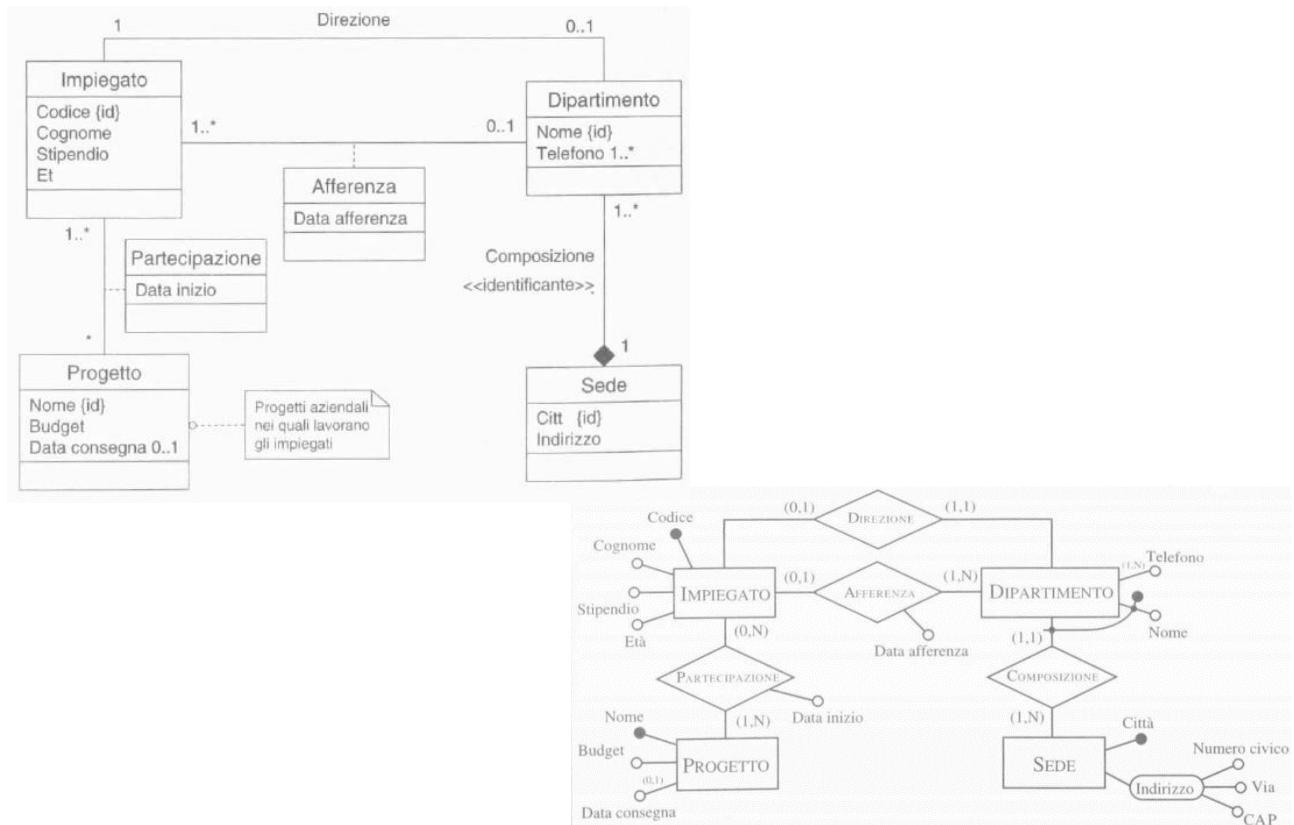
7.4.1.5 Generalizzazioni



Le generalizzazioni sono simili a quelle rappresentate nel modello E-R. eventuali proprietà delle generalizzazioni possono essere rappresentate con vincoli, usando la medesima sintassi descritta per gli attributi.

7.4.1.6 Note

Sono semplici commenti testuali. Vengono riportate sul diagramma stesso in un rettangolo con l'angolo superiore destro ripiegato. È possibile legare una nota ad un elemento tramite una linea tratteggiata.



8 LA PROGETTAZIONE CONCETTUALE

8.1 LA RACCOLTA E L'ANALISI DEI REQUISITI

Il reperimento e l'analisi dei requisiti di un'applicazione sono attività difficilmente standardizzabili perché dipendono molto dall'applicazione con cui si ha a che fare.

Per raccolta dei requisiti si intende la completa individuazione dei problemi che l'applicazione dovrà risolvere e le caratteristiche che tale applicazione dovrà avere. Per caratteristiche del sistema si intendono sia gli aspetti statici (i dati) che gli aspetti dinamici (le operazioni sui dati).

L'analisi dei requisiti consiste nel chiarimento e nell'organizzazione delle specifiche dei requisiti.

Si tratta di attività fortemente interconnesse, in molti casi è l'attività stessa di analisi dei requisiti che suggerisce successive attività di raccolta.

Le principali fondi di informazione sono:

- **Utenti dell'applicazione:** informazioni acquisite tramite interviste o documentazioni scritte
- **Documentazione esistente:** moduli, regolamenti interni, procedure aziendali, normative.
- **Realizzazioni preesistenti:** applicazioni che si devono rimpiazzare o che devono interagire con il software da realizzare

Come criterio generale da seguire possiamo dire che nel corso delle interviste è opportuno effettuare con l'utente verifiche di comprensione e consistenza delle informazioni che si stanno raccogliendo. Questo può essere fatto tramite esempi o tramite richieste di definizioni e classificazioni precise.

<i>Società di formazione</i>	
1	Si vuole realizzare una base di dati per una società che eroga corsi,
2	Di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei
3	Docenti. Per i partecipanti (circa 5000), identificati da un codice, si
4	Vuole memorizzare il codice fiscale, il cognome, l'età, il sesso, il luogo
5	Di nascita, il nome dei loro attuali datori di lavoro, i posti dove han-
6	No lavorato in precedenza insieme al periodo, l'indirizzo e il numero
7	Di telefono, i corsi che hanno frequentato (i corsi sono in tutto circa
8	200) e il giudizio finale. Rappresentiamo anche i seminari che stanno
9	Attualmente frequentando e, per ogni giorno, i luoghi e le ore dove
10	Sono tenute le lezioni. I corsi hanno un codice, un titolo e possono
11	Avere varie edizioni con date di inizio e fine e numero di partecipanti.
12	Se gli studenti sono liberi professionisti, vogliamo conoscere l'area
13	Di interesse e, se lo possiedono, il titolo. Per quelli che lavorano al-
14	Le dipendenze di altri, vogliamo conoscere invece il loro livello e la
15	Posizione ricoperta. Per gli insegnanti (circa 300), rappresentiamo il
16	Cognome, l'età, il posto dove sono nati, il nome del corso che inse-
17	Gnano, quelli che hanno insegnato nel passato e quelli che possono
18	Insegnare. Rappresentiamo anche tutti i loro recapiti telefonici. I do-
19	Centi possono essere dipendenti interni della società o collaboratori
20	Esterini.

Supponiamo di voler progettare una società di formazione e di aver raccolto, sulla base di alcune interviste, le specifiche dei dati espresse in linguaggio naturale:

È facile rendersi conto che tale testo presenta un certo numero di ambiguità e imprecisioni. Per esempio, si utilizzano i termini partecipante e studente per indicare lo stesso concetto, uguale per docente/professore e corso/seminario.

Poniamo ora alcune

regole generali per ottenere una specifica dei requisiti più precisa e senza ambiguità:

- Scegliere il corretto livello di astrazione: è bene evitare termini troppo generici o troppo specifici che rendono chiaro un concetto.

- Standardizzare la struttura delle frasi: è preferibile utilizzare sempre lo stesso stile sintattico. Per esempio “per <dato> rappresentiamo <insieme di proprietà>”.
- Evitare frasi contorte: le definizioni devono essere semplici e chiare, ad esempio “lavoratori dipendenti” è da preferire a “quelli che lavorano alle dipendenze di altri”.
- Individuare sinonimi/omonimi e unificare i termini
- Rendere esplicito il riferimento tra termini: ad esempio alle righe 6 e 7 non è chiaro se i termini indirizzo e numero di telefono siano relativi ai partecipanti o ai loro datori di lavoro
- Costruire un glossario dei termini

Termino	Descrizione	Sinonimi	Collegamenti
Partecipante	Partecipante ai corsi. Può essere un dipendente o un professionista.	Studente	Corso, Datore
Docente	Docente dei corsi. Possono essere collaboratori esterni.	Insegnante	Corso
Corso	Corsi offerti. Possono avere varie edizioni.	Seminario	Docente, Partecipante
Datore	Datori di lavoro attuali e passati dei partecipanti ai corsi.	Posto	Partecipante

Il primo passo è quindi eliminare le modifiche da apportare al testo. È utile, in questa fase, decomporre il testo in gruppi di frasi omogenee, relative cioè agli stessi concetti.

Frasi di carattere generale
Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti
Frasi relative ai partecipanti
Per i partecipanti (circa 5000), identificati da un codice, rappresentiamo il codice fiscale, il cognome, l'età, il sesso, la città di nascita, i nomi dei loro attuali datori di lavoro e di quelli precedenti (insieme alle date di inizio e fine rapporto), le edizioni dei corsi che stanno attualmente frequentando e quelli che hanno frequentato in passato, con la relativa votazione finale in decimi.
Frasi relative ai datori di lavoro
Relativamente ai datori di lavoro presenti e passati dei partecipanti, rappresentiamo il nome, l'indirizzo e il numero di telefono
Frasi relative ai corsi
Per i corsi (circa 200), rappresentiamo il titolo e il codice, le varie edizioni con date di inizio e fine e, per ogni edizione, rappresentiamo il numero di partecipanti e il giorno della settimana, le aule e le ore dono si sono tenute le lezioni
Frasi relative a tipi specifici di partecipanti
Per i partecipanti che sono liberi professionisti, rappresentiamo l'area di interesse e, se lo possiedono, il titolo professionale. Per i partecipanti che sono dipendenti, rappresentiamo invece il loro livello e la posizione ricoperta.

Frasi relative ai docenti

Per i docenti (circa 300), rappresentiamo il cognome, l'età, la città di nascita, tutti i numeri di telefono, il titolo del corso che insegnano, di quelli che hanno insegnato in passato e di quelli che possono insegnare. I docenti possono essere dipendenti interni della società di formazione o collaboratori esterni.

Naturalmente, accanto alle specifiche sui dati, vanno raccolte le specifiche sulle operazioni da effettuare su questi dati. Dobbiamo usare la stessa terminologia usata per i dati e informarci anche sulla frequenza con la quale le varie operazioni vengono eseguite. Per la nostra applicazione, le operazioni potrebbero essere:

- 1) Inserisci un nuovo partecipante indicando tutti i suoi dati (circa 40 volte al giorno)
- 2) Assegna un partecipante a una edizione di corso (circa 50 volte al giorno)
- 3) Inserisci un nuovo docente indicando tutti i suoi dati e i corsi che può insegnare (2 volte al giorno)
- 4) Assegna un docente abilitato a una edizione di un corso (15 volte al giorno)
- 5) Stampa tutte le informazioni sulle edizioni passate di un corso con titolo, orario lezioni e numero partecipanti (10 volte al giorno)
- 6) Stampa tutti i corsi offerti, con informazioni sui docenti che possono insegnarli (20 volte al giorno)
- 7) Per ogni docente, trova i partecipanti a tutti i corsi da lui/lei insegnati (5 volte a settimana)
- 8) Effettua una statistica su tutti i partecipanti a un corso con tutte le informazioni su di essi, sull'edizione alla quale hanno partecipato e sulla rispettiva votazione (10 volte al mese)

8.2 RAPPRESENTAZIONE CONCETTUALE DI DATI

Cerchiamo di stabilire alcune buone pratiche per una corretta rappresentazione concettuale dei dati.

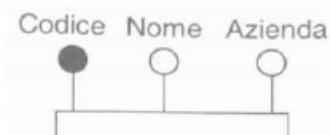
8.2.1 Criteri generali di rappresentazione

Spesso non esiste una rappresentazione univoca di un insieme di specifiche. Comunque, quando ci si trova davanti a diverse possibilità, è utile avere delle indicazioni sulle scelte più opportune. Nel caso della progettazione concettuale conviene seguire le regole concettuali del modello E-R:

- Se un concetto ha proprietà significative e/o descrive classi di oggetti con esistenza autonoma, è opportuno rappresentarlo con una entità: ad esempio è naturale rappresentare il concetto di docente con un'entità, in quanto possiede diverse proprietà e la sua esistenza è indipendente dagli altri concetti
- Se un concetto ha una struttura semplice e non possiede proprietà rilevanti associate, è opportuno rappresentarlo con un attributo di un altro concetto a cui si riferisce: per esempio il concetto di città è certamente un attributo e non un'entità nel nostro esempio
- Se sono state individuate due (o più) entità e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato da una relazione: ad esempio il concetto di partecipazione a un corso è certamente rappresentabile da una relazione tra partecipanti e corsi.
- Se uno o più concetti risultano essere casi particolari di un altro, è opportuno rappresentarli facendo uso di una generalizzazione: ad esempio professionista e dipendente sono casi particolari di partecipante.

8.2.2 Pattern di progetto

Cominciamo da un caso semplice: quello in cui si individua nelle specifiche un concetto autonomo con proprietà associate, le chiare caratteristiche di un'entità nel modello E-R.



In questo caso l'azienda è un attributo ma, così, non siamo rappresentando il concetto di azienda. Dobbiamo quindi rendere l'azienda un'entità a sé stante. Da qui l'immagine sottostante.

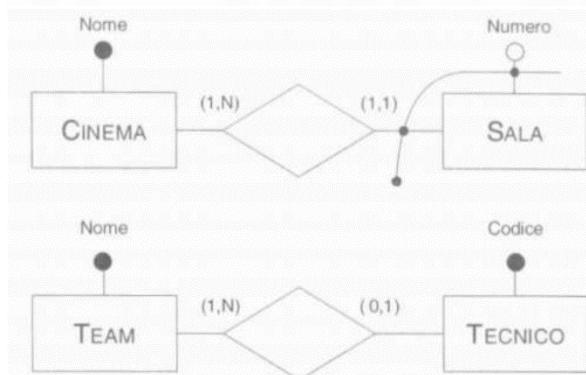


Figura 17 - Relazione "parte di"

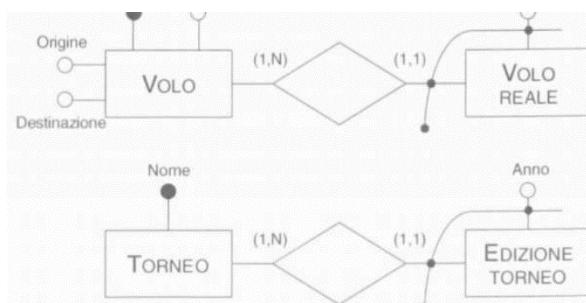
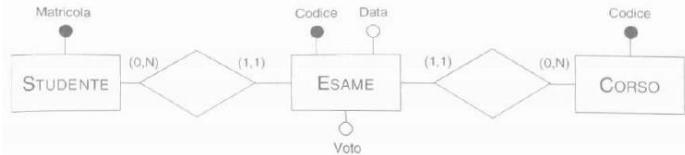


Figura 18 - Relazione "istanza di"

esse stesse entità come nell'esempio qua sotto. Bisogna tenere a mente che ogni tipo di cambiamento alla struttura presentata richiederà successivamente un adattamento del codice.



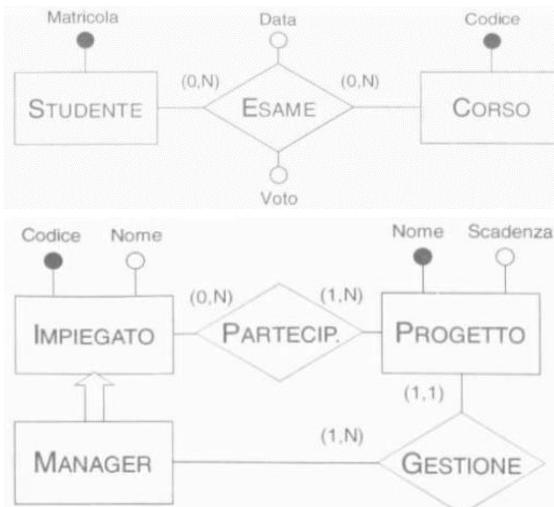


Figura 19 - Generalizzazione

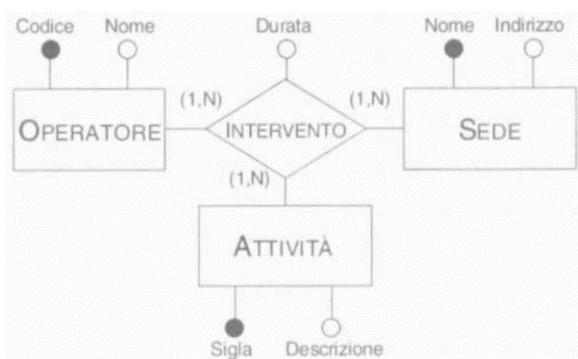
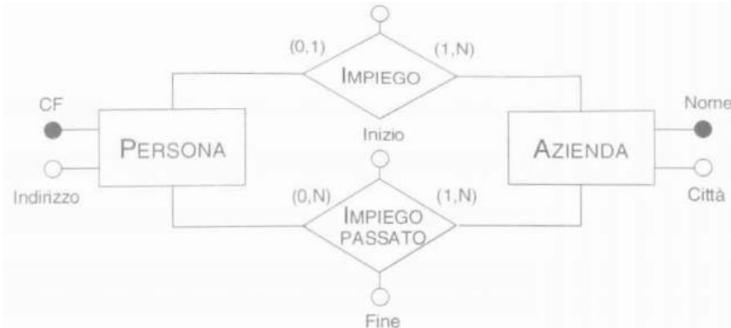
passiamo ora ad alcuni pattern che coinvolgono le generalizzazioni. Nell'esempio qua sotto possiamo pensare che un manager sia un caso particolare di impiegato. Possiamo anche assumere che un manager possa gestire solo un progetto al quale partecipa. Questo implica che ogni coppia manager-progetto che compare tra le occorrenze della relazione gestione deve comparire anche tra quelle della relazione partecipazione. Questo vincolo non può essere espresso direttamente nello schema e quindi andrà aggiunta una regola alla documentazione.

Un altro caso tipico di generalizzazione è quello usato per storizzare un'entità, ovvero creare uno storico delle sue occorrenze. Si può effettuare tramite due metodi:

- Storicizzazione di entità



- Storicizzazione di relazione

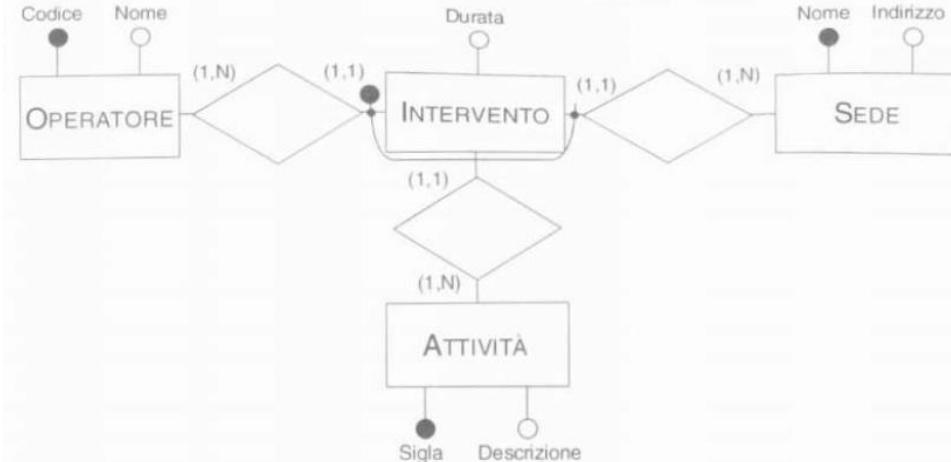


Ricordiamo che ogni relazione, volendo, può essere reificata⁷ e resa un'entità a sé stante. Ad esempio, l'intervento qua sotto viene reificato e da una relazione ternaria si passa a tre relazioni binarie.

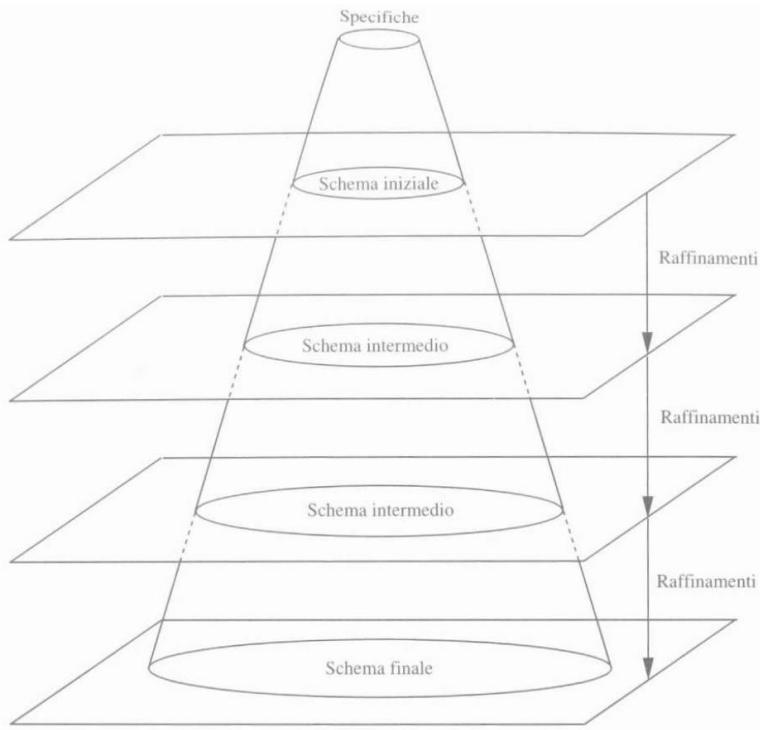
⁷ Resa entità superiore

8.3 STRATEGIE DI PROGETTO

Lo sviluppo di uno concettuale a partire dalle sue specifiche può essere considerato un processo di ingegnerizzazione e, per questo, sono applicabili le strategie di progetto utilizzate anche in altre discipline.



8.3.1 Strategia top-down



Lo schema concettuale viene prodotto mediante una serie di raffinamenti successivi a partire da uno schema iniziale che descrive tutte le specifiche con pochi concetti molto astratti. Lo schema viene poi raffinato mediante opportune trasformazioni che aumentano il dettaglio dei vari concetti presenti.

Con questa strategia quindi tutti gli aspetti presenti nello schema finale sono presenti, in linea di principio, a ogni livello di raffinamento.

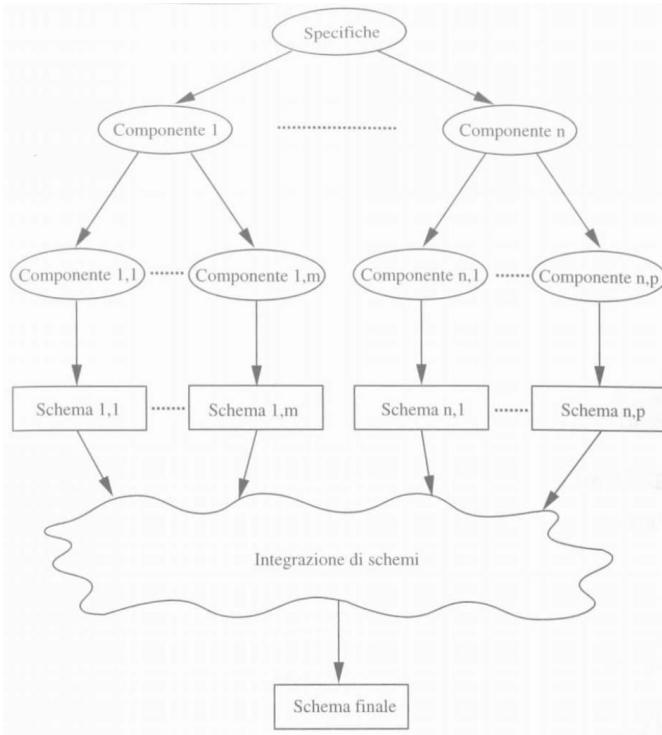
Le trasformazioni apportate vengono denominate “primitive di trasformazione top-down”.

Il vantaggio di questa strategia è che il progettista può descrivere inizialmente

tutte le specifiche trascurandone i dettagli per poi entrare nel merito poco alla volta. Questo è possibile solo quando, sin dall'inizio, si possiede una visione globale e astratta di tutte le componenti del sistema.

Esempio: dall'avere solo l'entità al definire i suoi attributi

8.3.2 Strategia bottom-up



Le specifiche iniziali sono suddivise in componenti via via sempre più piccole, fino a quando queste componenti descrivono un frammento elementare della realtà di interesse. A questo punto, le varie componenti vengono rappresentate da semplici schemi concettuali che possono consistere anche in singoli concetti. I vari schemi vengono poi fusi fino a giungere allo schema finale con la loro completa integrazione.

A differenza della strategia top-down, con questa strategia i vari concetti presenti nello schema finale vengono via via introdotti durante le varie fasi.

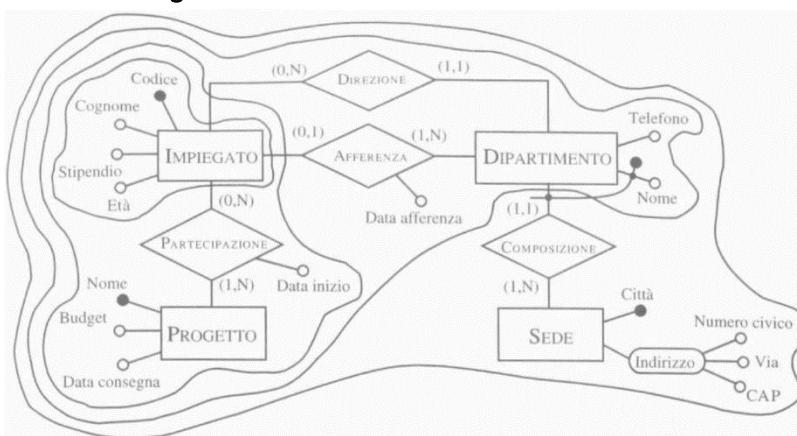
Anche in questo caso le varie trasformazioni sono dette "primitive di trasformazione bottom-up".

Il vantaggio di questa strategia è che si adatta a una decomposizione del problema in componenti più semplici, facilmente individuabili, il cui

progetto può essere affrontato da progettisti differenti. È una strategia che si presta bene a lavori svolti in collaborazione o suddivisi all'interno di un gruppo. Lo svantaggio è che richiede delle operazioni di integrazione di schemi concettuali diversi che, nel caso di schemi complessi, presentano quasi sempre grosse difficoltà.

esempio: introduzione di una nuova entità o di una relazione dall'analisi delle specifiche; per esempio nell'applicazione relativa alla società di formazione, questo può accadere quando tra le specifiche individuiamo l'entità partecipante oppure quando individuiamo la relazione abilitazione tra le entità docente e corso.

8.3.3 Strategia inside-out



Questa strategia può essere vista come un caso particolare di bottom-up. Si individuano inizialmente solo alcuni concetti importanti e poi si procede espandendo, ovvero si rappresentano prima i concetti in relazione con i concetti iniziali, per poi muoversi verso quelli più lontani attraverso una "navigazione" tra le specifiche.

Nell'esempio qua accanto si è partiti da impiegato, per poi passare all'idea di progetto, di dipartimento e infine di sede.

Questa strategia ha il vantaggio di non richiedere passi di integrazione. D'altro canto, è necessario, di volta in volta, esaminare tutte le specifiche per individuare concetti non ancora rappresentati e descrivere i nuovi concetti nel dettaglio.

8.3.4 Strategia mista

La strategia mista cerca di combinare i vantaggi della strategia top-down con quelli della strategia bottom-up. Il progettista suddivide i requisiti in componenti separate, come nella strategia bottom-up, ma allo stesso tempo definisce uno schema scheletro contenente, a livello astratto, i concetti principali dell'applicazione.

La strategia mista è probabilmente la più flessibile tra le strategie viste perché si adatta bene a esigenze contrapposte: suddividere un problema complesso in sotto problemi e procedere per raffinamenti successivi. In parte comprende anche la strategia inside-out poiché è naturale che durante il progetto ci si espanda per rappresentare nuove caratteristiche.

La maggior parte delle volte la strategia mista è l'unica possibile.

8.4 QUALITÀ DI UNO SCHEMA CONCETTUALE

8.4.1 Correttezza

Uno schema concettuale è corretto quando utilizza propriamente i costrutti messi a disposizione dal modello concettuale di riferimento.

Gli errori possono essere:

- **Sintattici:** uso non ammesso di costrutti, ad esempio una generalizzazione tra relazioni
- **Semantici:** uso di costrutti che non rispetta la loro definizione ad esempio una relazione al posto di una generalizzazione

La correttezza si può verificare per ispezione.

8.4.2 Completezza

Uno schema concettuale è completo quando rappresenta tutti i dati di interesse e quando tutte le operazioni possono essere eseguite a partire dai concetti descritti nello schema.

Si può verificare controllando che tutte le specifiche sui dati siano rappresentate in qualche modo.

8.4.3 Leggibilità

Uno schema concettuale è leggibile quando rappresenta i requisiti in maniera naturale e facilmente comprensibile.

Lo schema deve essere auto esplicativo, ad esempio mediante una scelta opportuna dei nomi e tramite scelte estetiche.

Alcuni suggerimenti sono:

- Disporre i concetti su una griglia scegliendo come elementi centrali con più legami (relazioni) con altri
- Tracciare solo linee perpendicolari e cercare di minimizzare le intersezioni
- Disporre le entità che sono genitori di generalizzazioni sopra le relative entità figlie

La leggibilità si può verificare facendo delle prove di comprensione con gli utenti.

8.4.4 Minimalità

Uno schema è minima quando tutte le specifiche sui dati sono rappresentate una sola volta nello schema. Uno schema non è minima se vi sono ridondanze, un esempio è quando in uno schema E-R ci sono cicli di relazioni e/o generalizzazioni.

A differenza delle altre proprietà, quindi, non sempre una ridondanza è indesiderata, anche se in questo caso bisogna documentarla.

La minimalità di uno schema si può verificare per ispezione.

8.5 UNA METODOLOGIA GENERALE

Per quel che riguarda le strategie di progetto viste va precisato che, in pratica, non accade quasi mai che un progetto proceda sempre in maniera top-down o bottom-up. Nelle situazioni reali capita di modificare lo schema in via di costruzione sia con trasformazioni che raffinano un concetto precedente (top-down), sia con trasformazioni che aggiungono un concetto non presente (bottom-up).

Presentiamo ora una metodologia per la progettazione concettuale con il modello E-R:

1) Analisi dei requisiti

- a. Costruire un glossario dei termini
- b. Analizzare i requisiti ed eliminare le ambiguità presenti
- c. Raggruppare i requisiti in insiemi omogenei

2) Passo base

- a. Individuare i concetti più rilevanti e rappresentarli in uno schema scheletro

3) Passo di decomposizione (da effettuare se appropriato o necessario)

- a. Effettuare una decomposizione dei requisiti con riferimento ai concetti presenti nello schema scheletro

4) Passo iterativo: da ripetere, per tutti i sottoschemi (se presenti), finché ogni specifica è rappresentata

- a. Raffinare i concetti presenti sulla base delle loro specifiche
- b. Aggiungere nuovi concetti allo schema per descrivere specifiche non ancora descritte

5) Passo di integrazione (da effettuare se è stato eseguito il passo 3)

- a. Integrare i vari sottoschemi in uno schema generale facendo riferimento allo schema scheletro

6) Analisi di qualità

- a. Verificare la correttezza dello schema ed eventualmente ristrutturare lo schema
- b. Verificare la completezza dello schema ed eventualmente ristrutturare lo schema
- c. Verificare la minimalità, documentare le ridondanze ed eventualmente ristrutturare lo schema
- d. Verificare la leggibilità dello schema ed eventualmente ristrutturare lo schema

Notare che se 3 e 5 non vengono eseguiti e il passo 4 vede eseguito solo il punto a, allora abbiamo una strategia top-down.

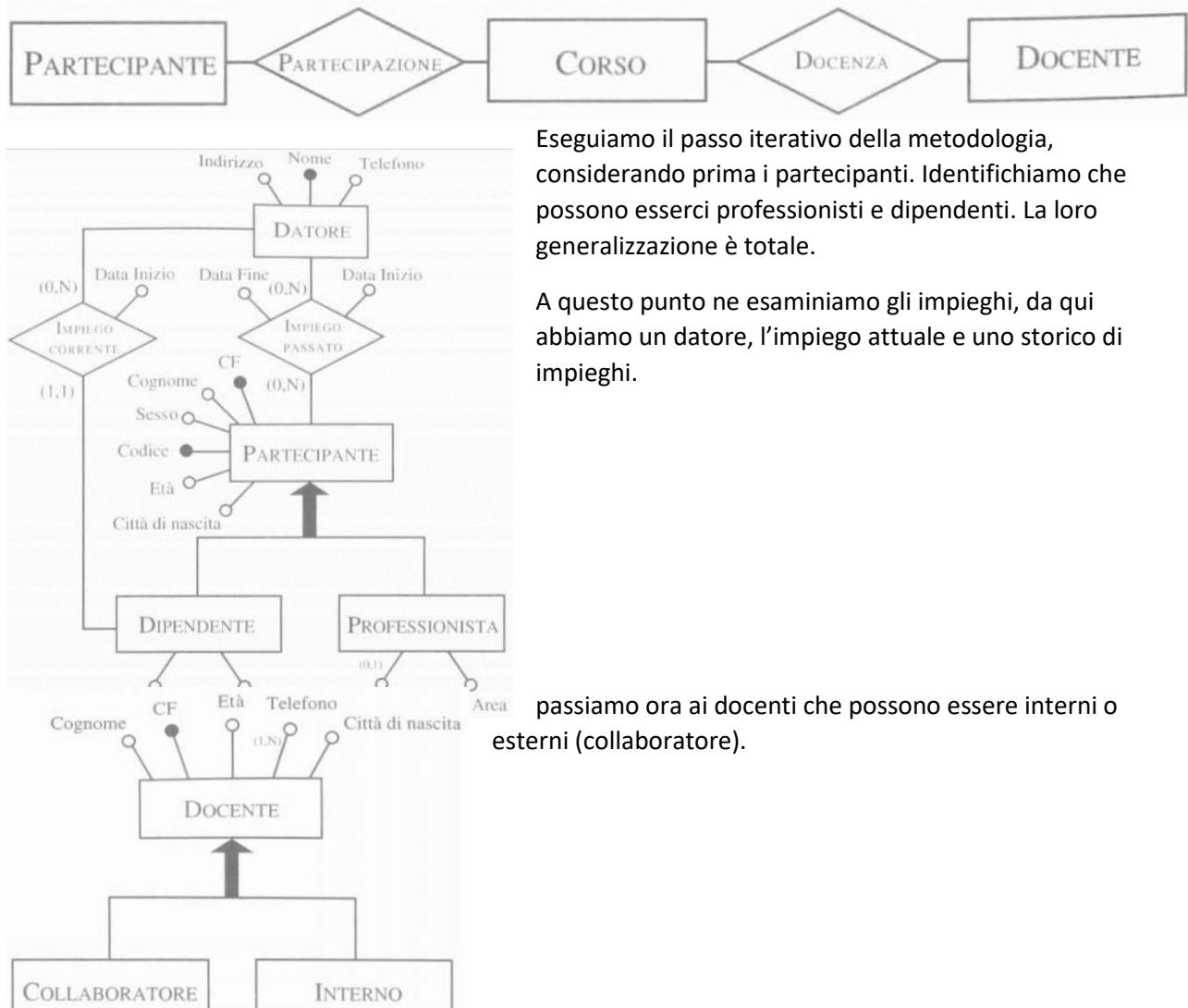
Al contrario, se 2 non viene effettuato e nel 4 vengono aggiunti solo nuovi concetti, allora abbiamo una strategia bottom-up.

Accanto a tutto questo si dovrebbe sempre tenere una documentazione degli schemi aggiornata.

Va sottolineato che il passo 6, l'analisi di qualità, è un passo importante nel momento di verifica dello stato corrente del progetto nel quale è spesso necessario dover effettuare delle ristrutturazioni per rimediare a "errori" fatti nelle fasi precedenti.

8.6 UN ESEMPIO DI PROGETTAZIONE CONCETTUALE

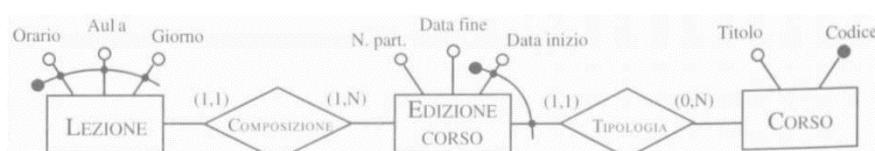
abbiamo già analizzato i requisiti (capitolo 8.1) e questo è lo scheletro del nostro progetto.



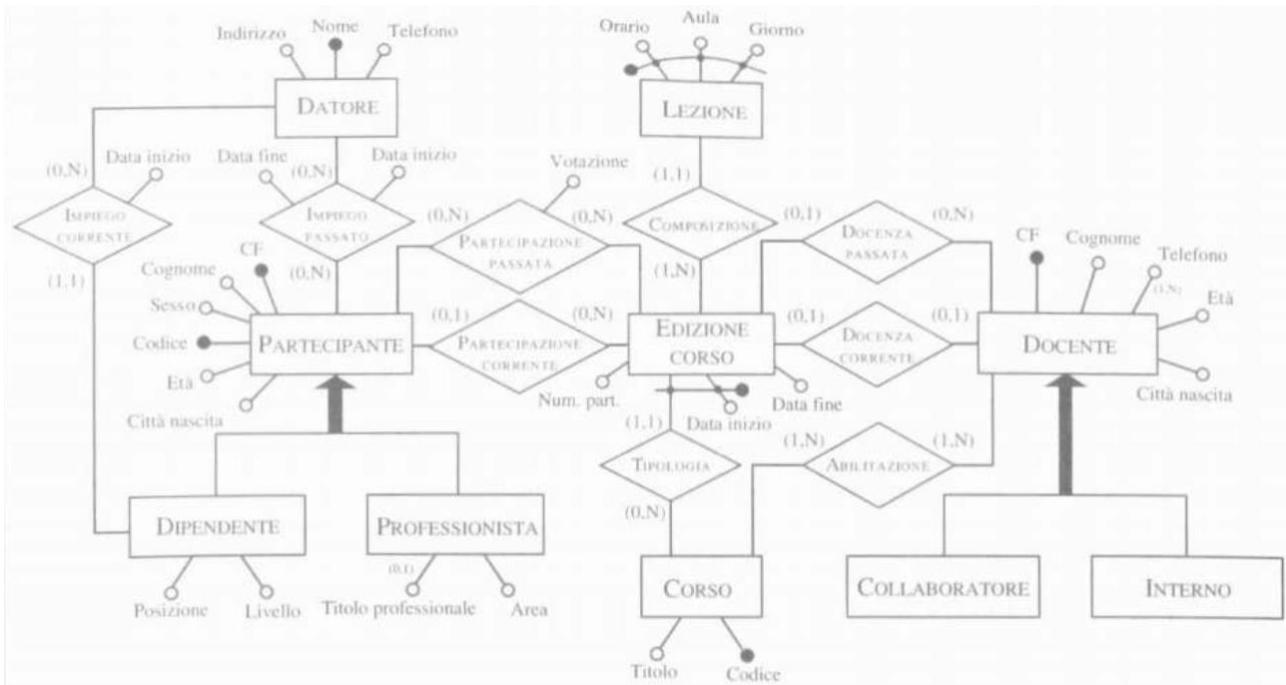
Eseguiamo il passo iterativo della metodologia, considerando prima i partecipanti. Identifichiamo che possono esserci professionisti e dipendenti. La loro generalizzazione è totale.

A questo punto ne esaminiamo gli impieghi, da qui abbiamo un datore, l'impiego attuale e uno storico di impieghi.

passiamo ora ai docenti che possono essere interni o esterni (collaboratore).



Passando all'entità corso vanno distinti due concetti: il concetto astratto di corso e la sua edizione. Saranno quindi due entità distinte. Poi analizziamo le lezioni.



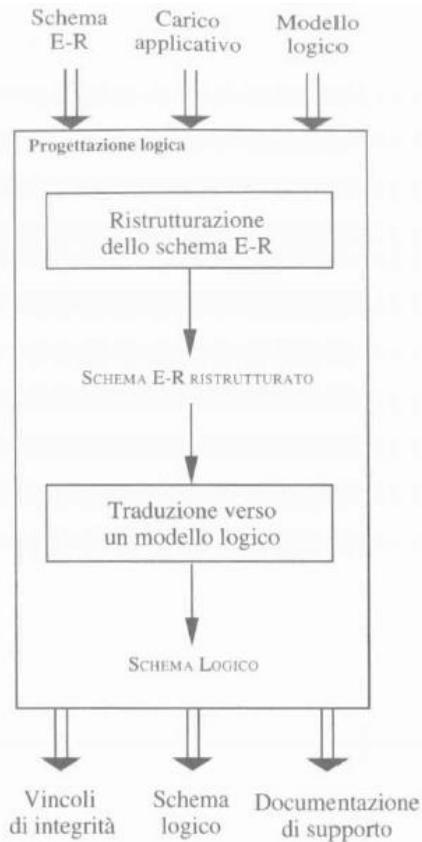
Lo schema finale si ottiene per integrazione degli schemi ottenuti fino a questo punto.

A questo punto restano da verificare le proprietà dello schema così ottenuto, ad esempio la completezza si verifica ripercorrendo tutti i requisiti sui dati e sulle operazioni controllando che tutti i dati siano rappresentati e che tutte le operazioni possano essere eseguite mediante una navigazione sullo schema.

Per quanto riguarda la minimalità, si può osservare che esiste una ridondanza: l'attributo numero di partecipanti dell'entità edizione di corso può essere derivato, per una certa edizione, contando il numero di istante dell'entità partecipante che sono legate a quell'edizione.

9 LA PROGETTAZIONE LOGICA

9.1 FASI DELLA PROGETTAZIONE LOGICA



Le attività principali della progettazione logica sono la riorganizzazione dello schema concettuale e la traduzione in un modello logico.

È utile di solito articolare la progettazione logica in due fasi:

- 1) Ristrutturazione dello schema entità-relazione: è una fase indipendente dal modello logico scelto e si basa su criteri di ottimizzazione dello schema
- 2) Traduzione verso il modello logico: fa riferimento a uno specifico modello logico e può includere un'ulteriore ottimizzazione

Lo schema E-R e il carico applicativo, ovvero le dimensioni dei dati e le caratteristiche delle applicazioni, sono l'input della prima fase, mentre lo schema E-R ristrutturato e il modello logico lo sono della seconda.

Lo schema logico finale, i vincoli di integrità e la relativa documentazione sono i prodotti finali della progettazione logica.

9.2 ANALISI DELLE PRESTAZIONI SU SCHEMI E-R

È possibile effettuare studi di massima dei due parametri che generalmente regolano le prestazioni dei sistemi software:

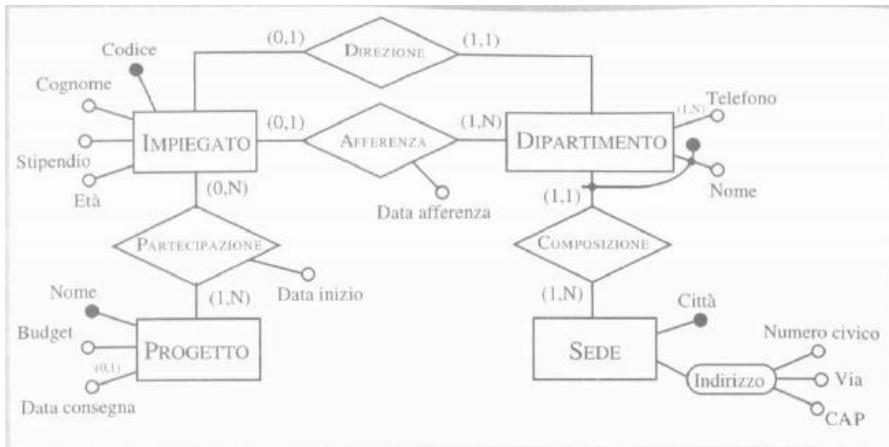
- Costo di un'operazione: viene valutato in termini di numero di occorrenze di entità e associazioni che mediamente vanno visitate per rispondere a una operazione sulla base di dati
- Occupazione di memoria: viene valutato in termini dello spazio di memoria (byte) necessario per memorizzare i dati descritti dallo schema

Per studiare questi parametri abbiamo bisogno di conoscere:

- Volume dei dati
 - Numero di occorrenze di ogni entità e associazione
 - Dimensioni di ciascun attributo
- Caratteristiche delle operazioni
 - Tipo dell'operazione
 - Frequenza (numero medio di esecuzioni in un certo intervallo di tempo)
 - Dati coinvolti

Per fare un esempio pratico vediamo il seguente schema.

Operazioni:



impiegati del dipartimento

- 1) Assegna un impiegato a un progetto
- 2) Trova i dati di un impiegato, del dipartimento nel quale lavora e dei progetti ai quali partecipa
- 3) Trova i dati di tutti gli impiegati di un certo dipartimento
- 4) Per ogni sede, trova i suoi dipartimenti con il cognome del direttore e l'elenco degli

Le operazioni sulle basi di dati seguono la cosiddetta regola “ottanta-venti”: l’ottanta per cento del carico è generato dal venti per cento delle operazioni.

Tavola dei volumi

Concetto	Tipo	Volume
Sede	E	10
Dipartimento	E	80
Impiegato	E	2000
Progetto	E	500
Composizione	R	80
Afferenza	R	1900
Direzione	R	80
Partecipazione	R	6000

Tavola delle operazioni

Operazione	Tipo	Frequenza
Op. 1	I	50 al giorno
Op. 2	I	100 al giorno
Op. 3	I	10 al giorno
Op. 4	B	2 a settimana

il volume dei dati e le caratteristiche generali delle operazioni possono essere descritte facendo uso di tabelle.

Nella tavola dei volumi vengono riportati tutti i concetti dello schema con il volume previsto a regime.

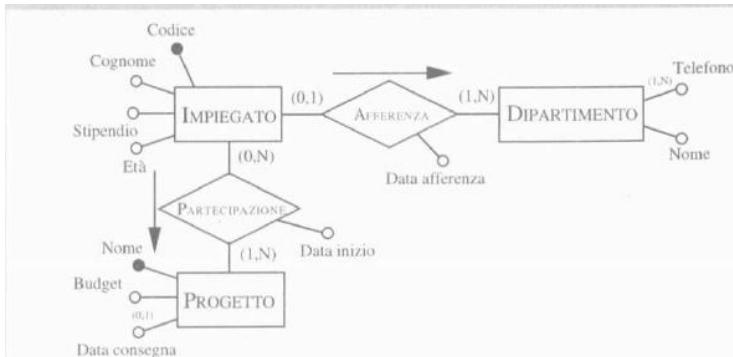
Nella tavola delle operazioni riportiamo, per ogni operazione, la frequenza prevista e un simbolo che indica se l’operazione è

interattiva (I) o batch (B).

Nella tavola dei volumi, il numero di occorrenze delle associazioni dipende da:

- Il numero di occorrenze delle entità coinvolte
- Numero medio di partecipazioni di un’occorrenza di entità alle occorrenze di associazioni

Per esempio, il numero di occorrenze dell’associazione Composizione è pari al numero dei dipartimenti, perché le cardinalità ci dicono che un dipartimento appartiene a una sola sede. Il numero di occorrenze dell’associazione Afferenza è invece poco meno del numero degli impiegati, perché dalle cardinalità si evince che ci sono impiegati che non afferiscono a nessun dipartimento. Infine, assumendo che un impiegato partecipa in media a tre progetti, abbiamo $2000 \times 3 = 6000$ occorrenze per l’associazione Partecipazione (e, quindi, $6000 / 500 = 12$ impiegati in media su ogni progetto).



Per ogni operazione possiamo descrivere graficamente i dati coinvolti con uno schema di operazione che consiste nel frammento dello schema E-R interessato, sul quale viene disegnato il “cammino logico”.

Successivamente, per conoscere i dati dei progetti ai quali lavora, dobbiamo

Figura 20 - Schema di operazione

Tavola degli accessi			
Concetto	Costrutto	Accessi	Tipo
Impiegato	Entità	1	L
Afferenza	Relazione	1	L
Dipartimento	Entità	1	L
Partecipazione	Relazione	3	L
Progetto	Entità	3	L

Figura 21 - Tavola degli accessi

9.3 RISTRUTTURAZIONE DI SCHEMI E-R



La fase di ristrutturazione di uno schema E-R si può suddividere in una serie di passi da effettuare in sequenza:

- 1) **Analisi delle ridondanze:** si decide se eliminare o mantenere eventuali ridondanze presenti nello schema
- 2) **Eliminazione delle generalizzazioni:** tutte le generalizzazioni presenti nello schema vengono analizzate e sostituite da altri costrutti
- 3) **Partizionamento/accorpamento di entità e associazioni:** si decide se è opportuno partizionare concetti dello schema in più concetti o viceversa
- 4) **Scelta degli identificatori principali:** si seleziona un identificatore per quelle entità che ne hanno più di uno

9.3.1 Analisi delle ridondanze

In uno schema E-R ci possono essere varie forme di ridondanza. Le più comuni sono:

- Attributi derivabili da altri della stessa entità. Ad esempio, nell'entità Fattura l'importo lordo è derivabile tramite somma di importo netto e IVA
- Attributi derivabili da attributi di altre entità. Ad esempio, l'importo totale di Acquisto è derivabile tramite composizione del prezzo per l'arietà della relazione
- Attributi derivabili da operazioni di conteggio di occorrenze. Ad esempio, Numero di abitanti di Città è derivabile dal numero di occorrenze di persone tramite l'associazione Residenza
- Associazioni derivabili dalla composizione di altre associazioni in presenza di cicli. Ad esempio, la relazione Docenza è derivabile dalle associazioni frequenza e insegnamento

La presenza di un dato derivato presenta un vantaggio e alcuni svantaggi:

- **Vantaggio:** è una riduzione degli accessi necessari per calcolare il dato derivato
- **Svantaggi:** maggiore occupazione della memoria, necessità di effettuare operazioni aggiuntive per mantenere il dato derivato aggiornato.

accedere a tre occorrenze dell'associazione Partecipazione e, attraverso queste, a tre occorrenze dell'entità Progetto. Tutto questo può essere riassunto in una tavola degli accessi. L sta per accesso in lettura, S per accesso in scrittura. Questa distinzione fatta perché le operazioni di scrittura sono più onerose di quelle in lettura.

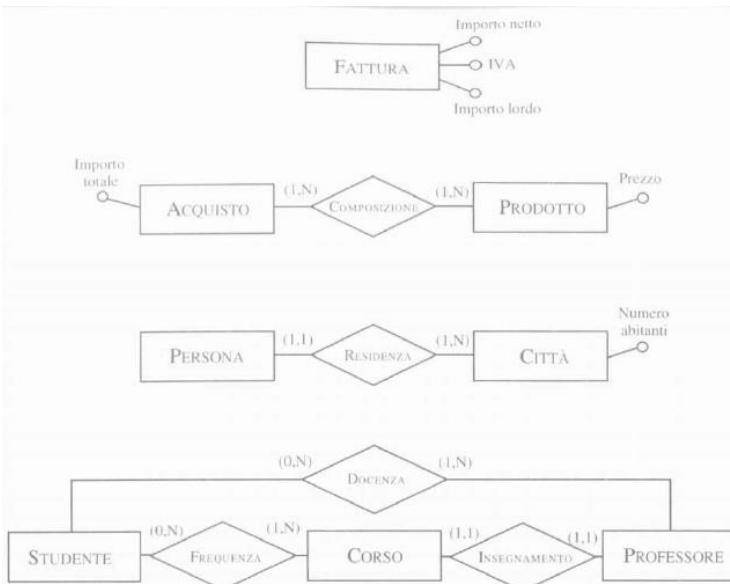


Figura 22 - Esempio di schema con ridondanze

Tavola dei volumi		
Concetto	Tipo	Volume
Città	E	200
Persona	E	1 000 000
Residenza	R	1 000 000

Tavola delle operazioni		
Operazione	Tipo	Frequenza
Op. 1	I	500 al giorno
Op. 2	I	2 al giorno

scrittura all'entità persona, all'associazione residenza e un accesso in lettura e uno in scrittura a città, il tutto ripetuto per 500 volte, quindi 1500 accessi in scrittura e 500 in lettura.

- Operazione 2: ha un costo trascurabile

Supponendo che un accesso in scrittura abbia un costo doppio rispetto a un accesso in lettura, abbiamo un totale di 3500 accessi al giorno in caso di presenza di un dato ridondante.

Passiamo ora ai calcoli in **assenza** di ridondanza:

- Operazione 1: accesso in scrittura a persona e a residenza per un totale di 1000 accessi in scrittura al giorno
- Operazione 2: accesso in lettura a città e 5000 accessi in lettura all'associazione residenza per calcolare il numero di persone, per un totale di 10000 accessi al giorno. Considerando doppi gli accessi in scrittura, abbiamo 12000 accessi al giorno in caso di dato ridondante assente. Quindi circa 8500 accessi in più contro un risparmio di un solo kilobyte.

Tavole degli accessi in presenza di ridondanza			
Operazione 1			
Concetto	Costr.	Acc.	Tipo
Persona	E	1	S
Residenza	R	1	S
Città	E	1	L
Città	E	1	S

Operazione 2			
Concetto	Costr.	Acc.	Tipo
Città	E	1	L
Residenza	R	5000	L

Tavole degli accessi in assenza di ridondanza			
Operazione 1			
Concetto	Costr.	Acc.	Tipo
Persona	E	1	S
Residenza	R	1	S

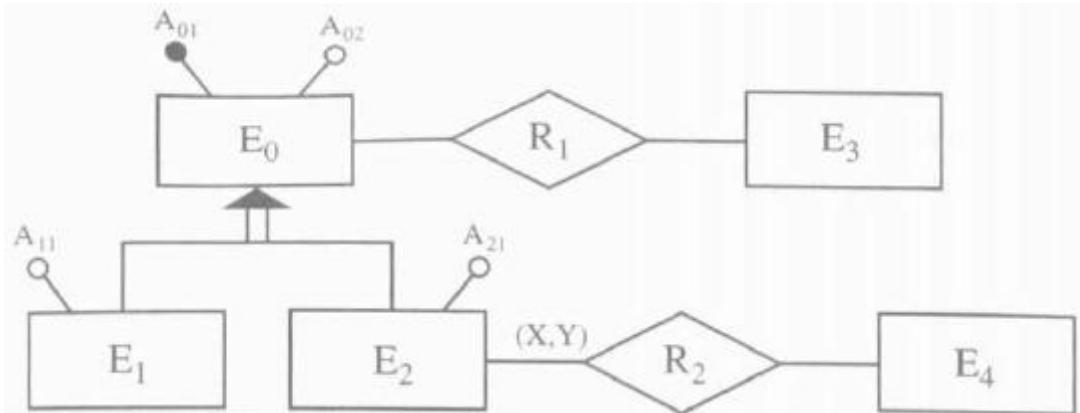
Operazione 2			
Concetto	Costr.	Acc.	Tipo
Città	E	1	L
Residenza	R	5000	L

Possiamo concludere che il dato ridondante possa essere mantenuto.

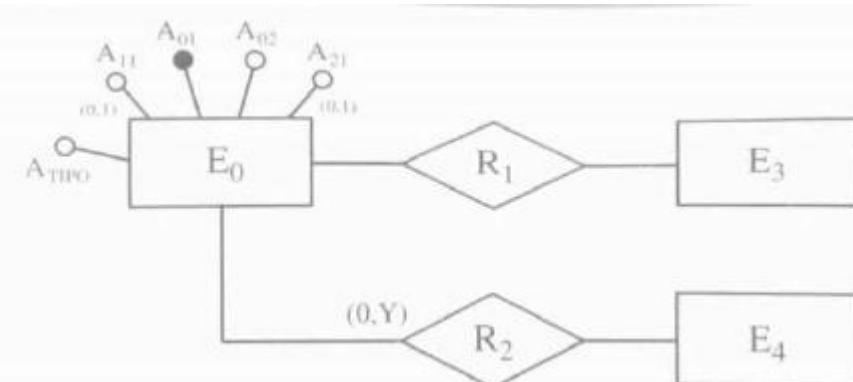
9.3.2 Eliminazione delle generalizzazioni

Risulta spesso necessario trasformare le generalizzazioni in entità e associazioni.

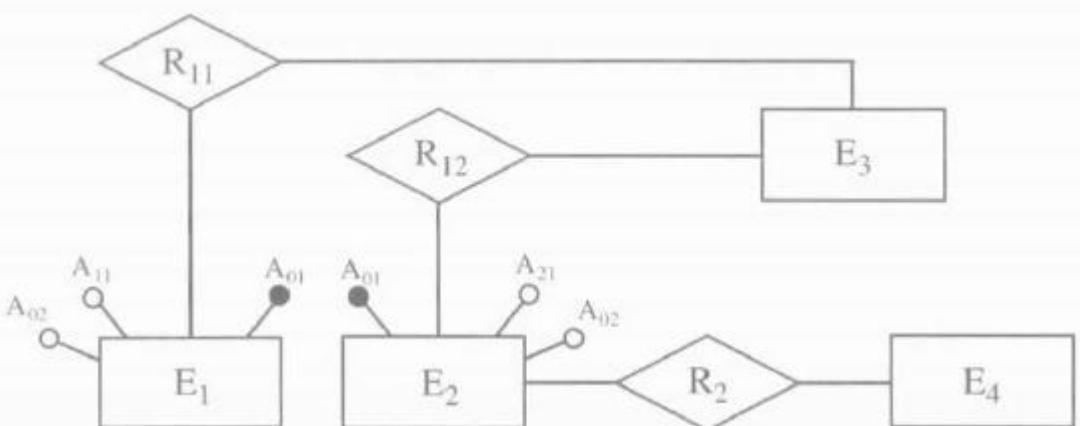
Per rappresentare una generalizzazione mediante entità e associazioni abbiamo tre alternative che verranno mostrate usando il seguente schema come esempio.



- 1) **Accorpamento delle figlie della generalizzazione nel genitore:** le entità E₁ ed E₂ vengono eliminate e le loro proprietà vengono aggiunte all'entità genitore E₀, alla quale viene aggiunto un ulteriore attributo per indicare il tipo di occorrenza di E₀

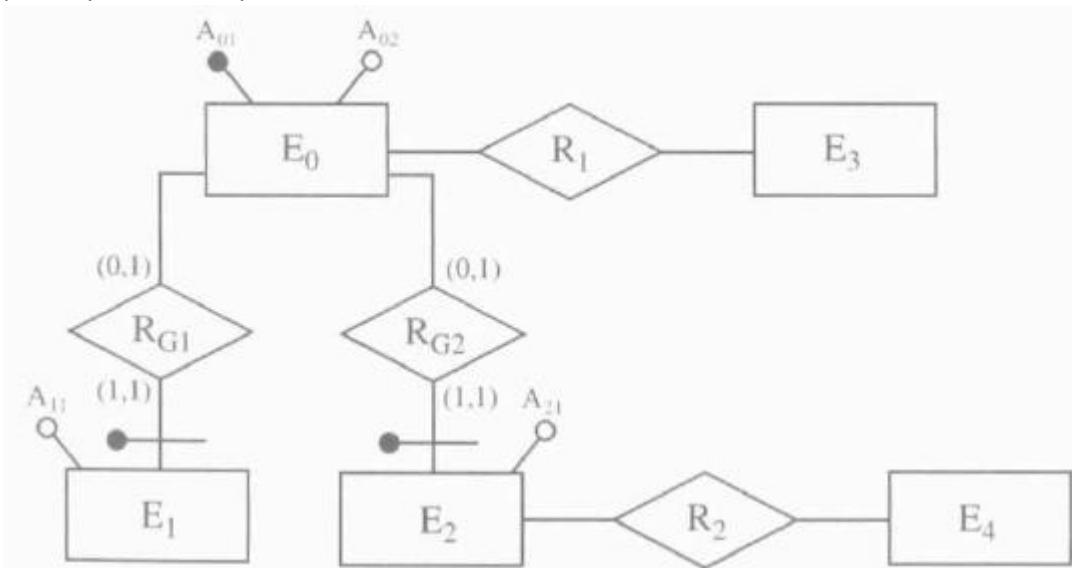


- 2) **Accorpamento del genitore della generalizzazione nelle figlie:** l'entità E₀ viene eliminata e, per ereditarietà, i suoi attributi e il suo identificatore vengono aggiunti alle figlie E₁ ed E₂



- 3) **Sostituzione della generalizzazione con associazioni:** la generalizzazione si trasforma in due associazioni uno a uno che legano rispettivamente l'entità genitore con le entità figlie E₁ ed E₂. Non ci sono trasferimenti di attributi o associazioni e le entità E₁ ed E₂ sono identificate esternamente ad E₀. Nello schema vanno aggiunti però dei vincoli, ovvero che ogni occorrenza di E₀ non può

partecipare contemporaneamente a R_{G1} e R_{G2}



La scelta tra le varie alternative può essere fatta sempre tramite analisi di vantaggi e svantaggi. È possibile stabilire delle regole di carattere generale:

- 1) L'alternativa 1 è conveniente quando le operazioni non fanno molta distinzione tra le occorrenze e tra gli attributi di E0, E1 ed E2. Vi è aumento della memoria ma riduzione degli accessi
- 2) L'alternativa 2 è possibile solo se la generalizzazione è totale, altrimenti le occorrenze di E0 ma non di E1 o E2 non sarebbero rappresentate. Vi è risparmio di memoria e riduzione di accessi
- 3) L'alternativa 3 è conveniente quando la generalizzazione non è totale e ci sono operazioni che si riferiscono solo a occorrenze di E1, E2 o E0 e dunque fanno molta distinzione tra entità figlia e genitore. Vi è risparmio di memoria rispetto all'alternativa 1 ma c'è un incremento di accessi per mantenere la consistenza delle occorrenze

Bisogna chiarire che la riduzione delle generalizzazioni non può appoggiarsi solo sul conteggio di istanze e accessi, infatti secondo questi criteri la scelta 3 non converrebbe mai. Questa scelta ha però il vantaggio di creare generiche entità con pochi attributi.

Le alternative viste non sono le uniche, ma è possibile effettuare ristrutturazioni che sono combinazioni delle tre trasformazioni viste.

Per esempio, per quanto riguarda le generalizzazioni su più livelli, si può procedere analogamente analizzando una generalizzazione alla volta a partire dal fondo dell'intera gerarchia.

9.3.3 Partizionamento/accorpamento di concetti

Entità e associazioni possono essere partizionati o accorpati per garantire una maggiore efficienza delle operazioni in base al seguente principio: gli accessi si riducono separando attributi di uno stesso concetto che vengono acceduti da operazioni diverse e aggregando attributi di concetti diversi che vengono acceduti dalle medesime operazioni.

9.3.3.1 Partizionamento di entità

Un esempio di partizionamento è quello della figura seguente. Questa ripartizione è conveniente se le operazioni che coinvolgono frequentemente l'entità originaria richiedono, per un impiegato, o solo informazioni di carattere anagrafico o solo informazioni relative alla retribuzione.

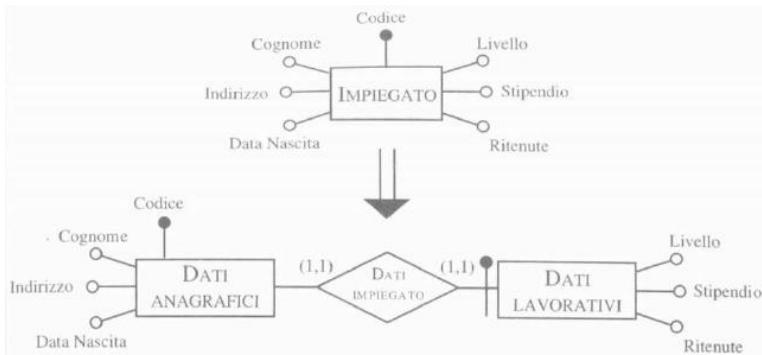
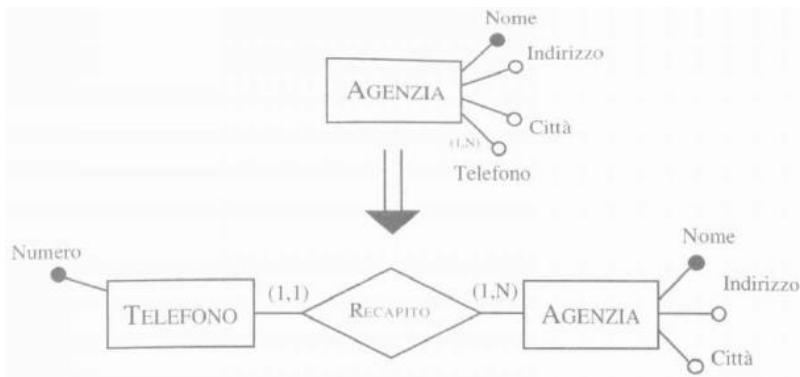


Figura 23 - Partizionamento verticale

duplicare tutte le associazioni a cui l'entità originaria partecipa.

9.3.3.2 Eliminazione di attributi multivalore

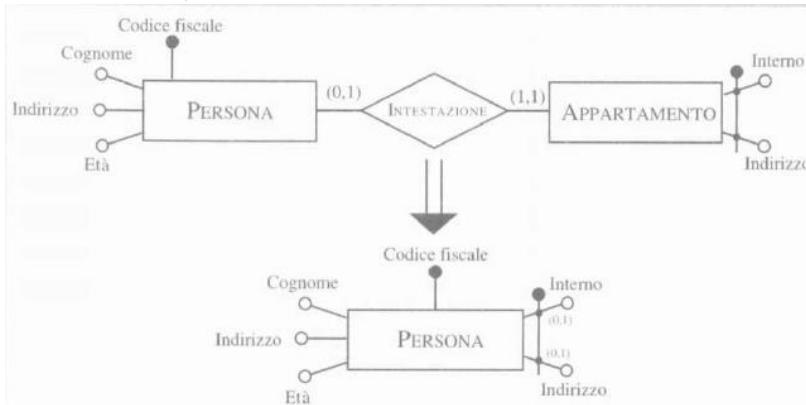


Un partizionamento di questo tipo è detto “decomposizione verticale”. È possibile una “decomposizione orizzontale” nella quale la suddivisione avviene sulle occorrenze dell’entità, ovvero si divide l’entità nei suoi sottogruppi in entità distinte. In questo caso le nuove entità hanno gli stessi attributi di quella di partenza.

I partizionamenti orizzontali hanno un effetto collaterale: quello di dover

Questa ristrutturazione si rende necessaria perché. Come per le generalizzazioni, il modello relazionale non permette di rappresentare in maniera diretta questo tipo di attributo.

9.3.3.3 Accorpamento di entità



è l’operazione inversa del partizionamento. Questa ristrutturazione può essere suggerita dal fatto che le operazioni più frequenti sull’entità Persona richiedono sempre i dati relativi all’appartamento che occupa e vogliamo quindi risparmiare gli accessi.

9.3.3.4 Altri tipi di partizionamento/accorpamento

Fino ad ora abbiamo visto partizionamento e accorpamento di entità ma lo stesso discorso si può estendere alle associazioni.

9.3.4 Scelta degli identificatori principali

I criteri di decisione sono:

- Gli attributi con valori nulli non possono costituire identificatori principali
- Un identificatore composto da uno o da pochi attributi è preferibile a uno con molti attributi
- Un identificatore interno è preferibile ad un identificatore esterno

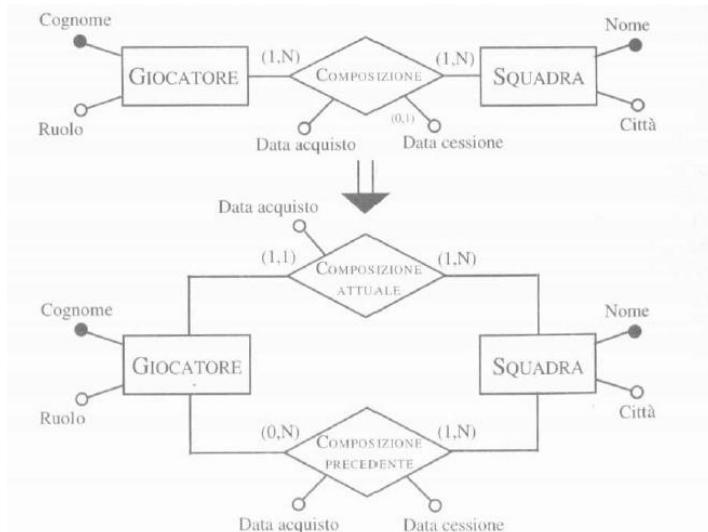


Figura 24 - Esempio di partizionamento di associazione

- Un identificatore utilizzato da molte operazioni per accedere alle occorrenze di un'entità è da preferire rispetto agli altri

Se non vi è un attributo candidabile che soddisfa questi requisiti si può pensare di introdurre un nuovo attributo che conterrà valori speciali (detti codici) generati appositamente per identificare l'occorrenza.

9.4 TRADUZIONE VERSO IL MODELLO RELAZIONALE

La seconda fase della progettazione logica corrisponde a una traduzione tra modelli di dati diversi: a partire da uno schema E-R ristrutturato si costruisce uno schema logico equivalente, in grado cioè di rappresentare le medesime informazioni. Si fa riferimento a una versione semplificata del modello E-R che non contiene generalizzazioni e attributi multivaleure e nella quale ogni entità ha un solo identificatore.

9.4.1 Entità e associazioni molti a molti



- Per l'associazione, una relazione con lo stesso nome avente per attributi gli attributi dell'associazione e gli identificatori delle entità coinvolte, tali identificatori formano la chiave della relazione

IMPIEGATO(Matricola, Cognome, Stipendio)
PROGETTO(Codice, Nome, Budget)

PARTECIPAZIONE(Matricola, Codice, DataInizio)

Figura 25 - Schema relazionale

PARTECIPAZIONE(Impiegato, Progetto, DataInizio)

Figura 26 - Ridenominazione

la traduzione dello schema qua accanto prevede:

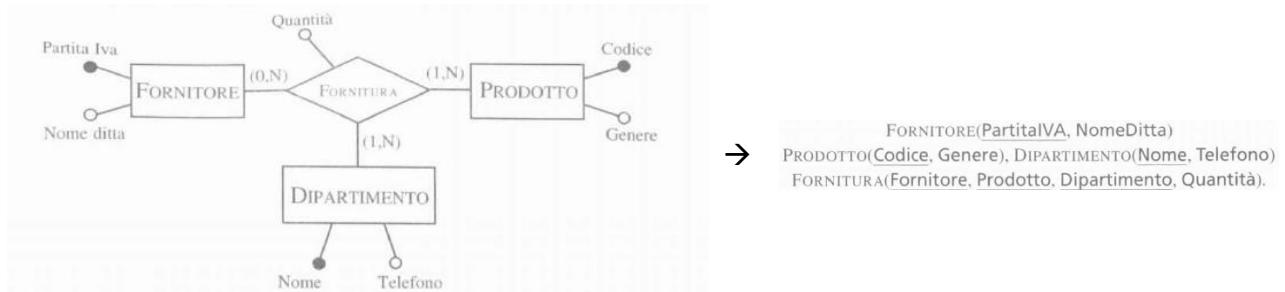
- Per ogni entità, una relazione con lo stesso nome avente per attributi i medesimi attributi dell'entità e per chiave il suo identificatore

Lo schema relazionale che i ottiene è evidenziato qua accanto.

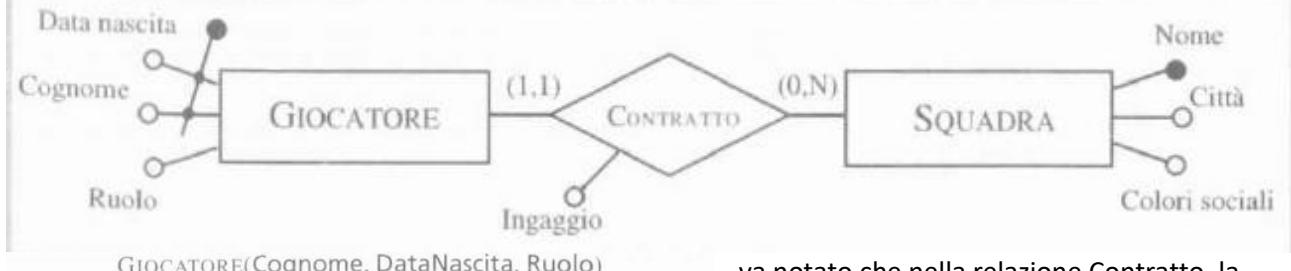
Esistono due vincoli di integrità referenziale tra gli attributi Matricola e Codice di Partecipazione e gli omonimi attributi di Impiegato e Progetto.

Per rendere più comprensibile lo schema si passa attraverso varie ridenominazioni.

In caso di associazioni ternarie si procede allo stesso modo:



9.4.2 Associazioni uno a molti



un contratto con una sola squadra.

A questo punto le relazioni Giocatore e contratto hanno la stessa chiave ed è allora possibile fonderle in un'unica relazione (perché esiste una corrispondenza biunivoca tra le rispettive occorrenze).

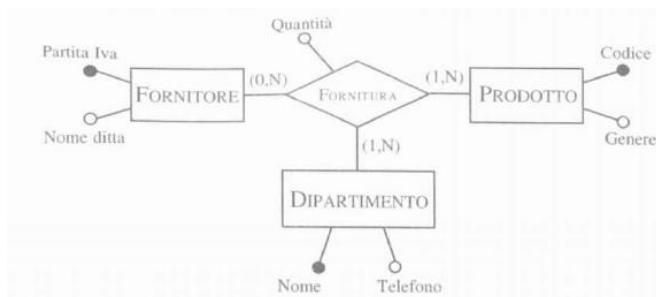
GIOCATORE(Cognome, DataNascita, Ruolo, NomeSquadra, Ingaggio)
SQUADRA(Nome, Città, ColoriSociali)

NomeSquadra della relazione Giocatore e l'attributo Nome della relazione Squadra.

In questo caso la cardinalità era 1, se fosse stata 0..1 entrambe le alternative viste sono valide.

va notato che nella relazione Contratto, la chiave è costituita solo dall'identificatore di Giocatore perché la cardinalità dell'associazione ci dice che ogni giocatore ha

in questo schema esiste ovviamente il vincolo d'integrità referenziale tra l'attributo

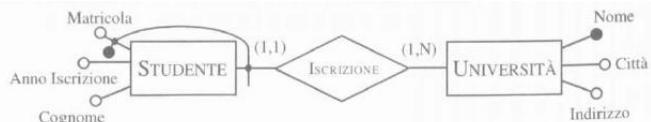


Nel caso in cui una delle entità partecipi con cardinalità massima pari ad uno, le cose non cambiano molto rispetto a quanto detto per le associazioni binarie. L'entità che partecipa all'associazione con cardinalità massima pari a uno, viene infatti tradotta in una relazione che contiene anche gli identificatori delle altre entità coinvolte nell'associazione. Ad

esempio, riprendendo lo schema qua accanto, se l'entità Prodotto partecipasse con cardinalità (1, 1) (e quindi per ogni prodotto esiste un solo fornitore che lo fornisce e un solo dipartimento al quale viene fornito), allora lo schema diventerebbe come segue.

FORNITORE(PartitaIva, NomeDitta) **DIPARTIMENTO(Nome, Telefono)**
PRODOTTO(Codice, Genere, Fornitore, Dipartimento, Quantità).

9.4.3 Entità con identificatore esterno



Le entità con identificatori esterni danno luogo a relazioni con chiavi che includono gli identificatori delle entità "identificanti".

Rappresentando l'identificatore NomeUniversità in Studente con vincolo di integrità, si può rappresentare direttamente l'associazione tra le due università.

STUDENTE(Matricola, NomeUniversità, Cognome, Annolscrizione)
UNIVERSITÀ(Nome, Città, indirizzo)

9.4.4 Associazioni uno a uno

In genere ci sono diverse possibilità di traduzione.

per l'esempio di figura 27 abbiamo due possibilità, elencate qua sotto.

nel primo caso abbiamo un vincolo tra DipartimentoDiretto e Nome, nel secondo tra Direttore e Codice.



È possibile quindi rappresentare l'associazione in una qualunque delle relazioni che rappresentano le due entità.

Dobbiamo però tenere a mente che stiamo ragionando con uno schema E-R rielaborato, abbiamo quindi deciso volontariamente di tenere separate le due entità.

Figura 27 - Esempio di partecipazione obbligatoria delle entità

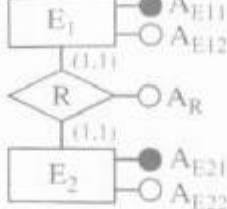
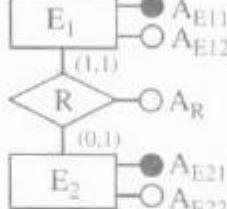
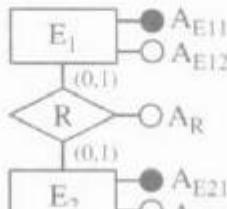
DIRETTORE(Codice, Cognome, Stipendio, DipartimentoDiretto, InizioDirezionamento)
DIPARTIMENTO(Nome, Telefono, Sede, Direttore, InizioDirezionamento)

DIRETTORE(Codice, Cognome, Stipendio)
DIPARTIMENTO(Nome, Telefono, Sede, Direttore, InizioDirezionamento)

Nel caso di un'associazione dove entrambe le entità sono opzionali, è preferibile mantenere l'associazione separata.

9.4.5 Tabelle riassuntive

Tipologia	Concetto iniziale	Risultati possibili
Associazione binaria molti a molti		$E_1(A_{E11}, A_{E12})$ $E_2(A_{E21}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$
Associazione ternaria molti a molti		$E_1(A_{E11}, A_{E12})$ $E_2(A_{E21}, A_{E22})$ $E_3(A_{E31}, A_{E32})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, \underline{A_{E31}}, A_R)$
Associazione uno a molti con partecipazione obbligatoria		$E_1(A_{E11}, A_{E12}, A_{E21}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$
Associazione uno a molti con partecipazione opzionale		$E_1(A_{E11}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$ Oppure: $E_1(A_{E11}, A_{E12}, A_{E21}, A_R^*)$ $E_2(\underline{A_{E21}}, A_{E22})$
Associazione con identificatore esterno		$E_1(A_{E12}, \underline{A_{E21}}, A_{E11}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

Tipologia	Concetto iniziale	Risultati possibili
Associazione uno a uno con partecipazione obbligatoria per entrambe le entità		$E_1(A_{E11}, A_{E12}, A_{E21}, A_R)$ $E_2(A_{E21}, A_{E22})$ Oppure: $E_2(A_{E21}, A_{E22}, A_{E11}, A_R)$ $E_1(A_{E11}, A_{E12})$
Associazione uno a uno con partecipazione opzionale per una entità		$E_1(A_{E11}, A_{E12}, A_{E21}, A_R)$ $E_2(A_{E21}, A_{E22})$
Associazione uno a uno con partecipazione opzionale per entrambe le entità		$E_1(A_{E11}, A_{E12})$ $E_2(A_{E21}, A_{E22}, A_{E11}, A_R^*)$ Oppure: $E_1(A_{E11}, A_{E12}, A_{E21}, A_R^*)$ $E_2(A_{E21}, A_{E22})$ Oppure: $E_1(A_{E11}, A_{E12})$ $E_2(A_{E21}, A_{E22})$ $R(A_{E11}, A_{E21}, A_R)$

9.4.6 Documentazioni di schemi logici

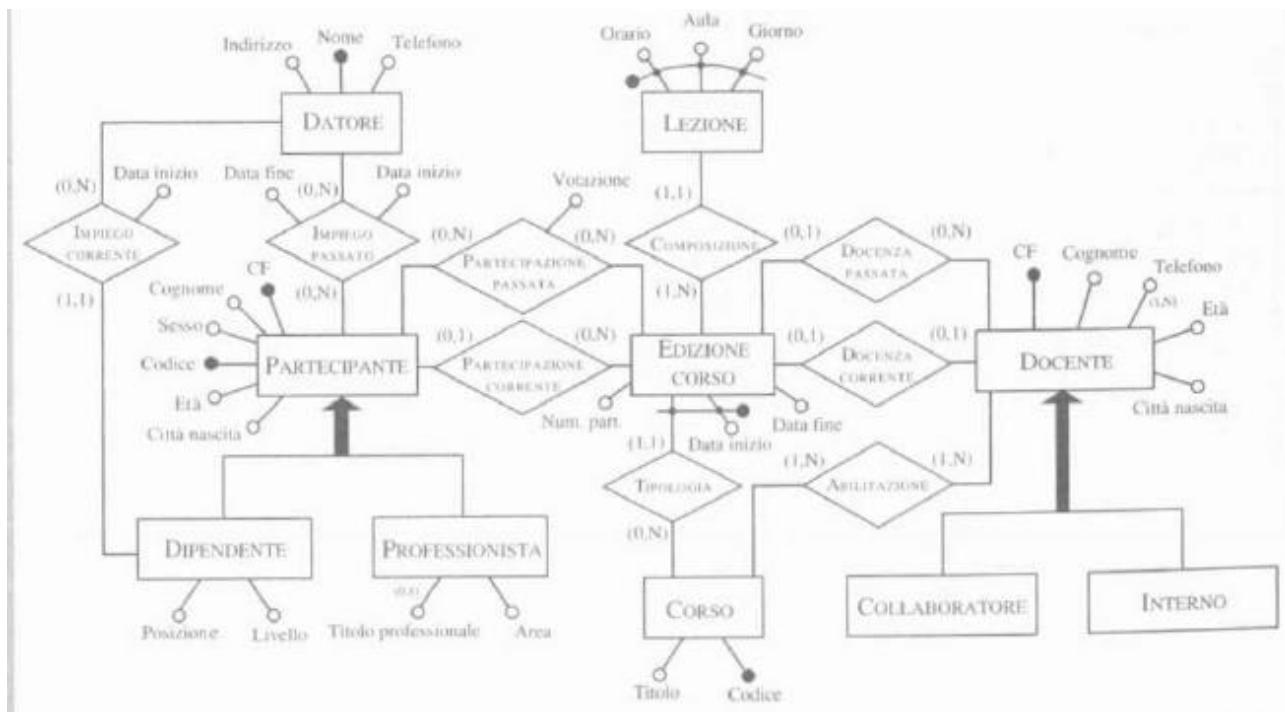
buona parte della documentazione dello schema concettuale in ingresso alla fase di progettazione logica può essere ereditata dallo schema logico ottenuto come risultato di questa fase.

A questa documentazione ne va aggiunta un'altra per descrivere i vincoli di integrità referenziale.

Si può adottare un formalismo grafico per permettere di rappresentare sia le relazioni che i vincoli.



9.5 UN ESEMPIO DI PROGETTAZIONE LOGICA



Operazioni:

- 1) Inserisci un nuovo partecipante indicando tutti i suoi dati
- 2) Assegna un partecipante a un'edizione di corso
- 3) Inserisci un nuovo docente indicando tutti i suoi dati e i corsi che può insegnare
- 4) Assegna un docente abilitato a un'edizione di un corso
- 5) Stampa tutte le informazioni sulle edizioni passate di un corso con titolo, orari delle lezioni e numero di partecipanti
- 6) Stampa tutti i corsi offerti, con informazioni sui docenti che possono insegnarli
- 7) Per ogni docente, trova i partecipanti a tutti i corsi da lui insegnati
- 8) Effettua una statistica su tutti i partecipanti a un corso con tutte le informazioni su di essi, sulla edizione alla quale hanno partecipato e la rispettiva votazione

9.5.1 Fase di ristrutturazione

Tavola dei volumi

Concetto	Tipo	Volume
Lezione	E	8000
Edizione corso	E	1000
Corso	E	200
Docente	E	300
Collaboratore	E	250
Interno	E	50
Partecipante	E	5000
Dipendente	E	4000
Professionista	E	1000
Datore	E	8000
Part. passata	R	10 000
Part. corrente	R	500
Composizione	R	8000
Tipologia	R	1000
Doc. passata	R	900
Doc. corrente	R	100
Abilitazione	R	500
Impiego corrente	R	4000
Impiego passato	R	1000

Tavola delle operazioni

Operazione	Tipo	Frequenza
Op. 1	I	40/giorno
Op. 2	I	50/giorno
Op. 3	I	2/giorno
Op. 4	I	15/giorno
Op. 5	I	10/giorno
Op. 6	I	20/giorno
Op. 7	I	5/sett.
Op. 8	B	10/mese

9.5.1.1 Analisi delle ridondanze

C'è un solo dato ridondante: l'attributo

Numero di partecipanti in Edizione Corso che può essere derivato dalle associazioni

Partecipazione Corrente e Partecipazione Passata.

Questo dato richiede $4 \times 1000 = 4000$ byte.

Le operazioni coinvolte da questo dato sono la 2, la 5 e la 8.

Dato ridondante presente: 490 accessi giornalieri

Dato ridondante assente: 440 accessi giornalieri

Quindi, in caso di ridondanza ci sono svantaggi sia di memoria che di efficienza.

Tavole degli accessi in presenza di ridondanza

Operazione 2

Concetto	Costr.	Acc.	Tipo
Partecipante	E	1	L
Par. corrente	R	1	S
Ediz. corso	E	1	L
Ediz. corso	E	1	S

Tavole degli accessi in assenza di ridondanza

Operazione 2

Concetto	Costr.	Acc.	Tipo
Partecipante	E	1	L
Par. corrente	R	1	S

Operazione 5

Concetto	Costr.	Acc.	Tipo
Ed. corso	E	1	L
Tipologia	R	1	L
Corso	E	1	L
Composiz.	R	8	L
Lezione	E	8	L

Operazione 5

Concetto	Costr.	Acc.	Tipo
Ediz. corso	E	1	L
Tipologia	R	1	L
Corso	E	1	L
Composiz.	R	8	L
Lezione	E	8	L
Par. corrente	R	10	L

9.5.1.2 Eliminazione delle gerarchie

Nello schema sono presenti due gerarchie: quella relativa ai docenti e quella relativa ai partecipanti.

Per i docenti si può notare che le operazioni non fanno distinzione tra collaboratori interni ed esterni, quindi possiamo accorpare le entità figlie della generalizzazione nel genitore, aggiungendo un attributo Tipo all'entità docente.

Per quanto riguarda i partecipanti idem, però possiamo osservare dallo schema che i professionisti e i dipendenti hanno degli attributi che li distinguono, risulta quindi preferibile lasciare le entità e aggiungere due associazioni uno a uno tra queste entità e l'entità partecipante, in questo modo si evitano valori nulli.

9.5.1.3 *Partizionamento/accorpamento di concetti*

La prima riguarda l'entità EDIZIONE DI CORSO: si può osservare che l'operazione 5 riguarda solo una frazione delle edizioni, quelle passate, e che le associazioni DOCENZA PASSATA e PARTECIPAZIONE PASSATA fanno riferimento solo a queste edizioni di corso. Si potrebbe quindi pensare, per rendere più efficiente l'operazione suddetta, di decomporre orizzontalmente l'entità in maniera da distinguere le edizioni correnti da quelle passate. L'inconveniente di questa scelta però è che le associazioni COMPOSIZIONE e TIPOLOGIA andrebbero duplicate; inoltre, le operazioni 7 e 8, che non fanno grosse distinzioni tra le edizioni correnti e quelle passate, risulterebbero più costose perché richiedono la visita di due entità distinte. Decidiamo quindi di non partizionare tale entità.

Due altre possibili ristrutturazioni che si può pensare di effettuare, proprio in conseguenza a quanto detto sulle edizioni dei corsi, sono l'accorpamento delle associazioni **DOCENZA PASSATA e DOCENZA CORRENTE e delle associazioni analoghe PARTECIPAZIONE PASSATA e PARTECIPAZIONE CORRENTE**. Si tratta infatti, in entrambi i casi, di due concetti simili (l'unica differenza è di carattere temporale) tra i quali alcune operazioni non fanno differenza (la 7 e la 8). **Il loro accorpamento produrrebbe un altro beneficio: non sarebbe necessario trasferire occorrenze da un'associazione a un'altra quando un'edizione di corso termina.** Per le partecipazioni ai corsi, un inconveniente è la presenza dell'attributo Votazione che non si applica alle partecipazioni correnti e quindi provocherebbe la presenza di valori nulli. **Del resto, la tavola dei volumi ci dice che il numero medio di occorrenze dell'entità PARTECIPAZIONE CORRENTE è 500 e quindi, supponendo di aver bisogno di 4 byte per memorizzare la votazione, lo spreco di memoria sarebbe di soli due kilobyte.** Decidiamo quindi di accoppare le due coppie di relazioni come descritto in Figura 9.33. Va aggiunto il vincolo non esprimibile dallo schema che un docente non può insegnare più di una edizione di corso nello stesso periodo e, analogamente, il vincolo che un partecipante non può seguire più di un corso nello stesso periodo.

Infine, bisogna eliminare l'attributo multivalore Telefono associato all'entità DOCENTE. Per far questo, introduciamo una nuova entità TELEFONO legata da una associazione uno a molti con l'entità DOCENTE, che viene privata del relativo attributo.

9.5.1.4 *Scelta degli identificatori principali*

Solo l'entità PARTECIPANTE presenta due identificatori: il codice fiscale e il codice interno. Tra i due è certamente preferibile scegliere il secondo. Infatti, un codice fiscale richiede 16 byte di memoria mentre un codice interno, che serve a distinguere al più 5000 occorrenze (vedi tavola dei volumi), richiede non più di 2 byte.

L'entità Corso identificata dall'attributo Data inizio e dall'entità CORSO. Ne risulta un identificatore piuttosto pesante che, in una rappresentazione relazionale, deve essere usato per rappresentare due associazioni (PARTECIPAZIONE e DOCENZA) con molte occorrenze. Si può osservare però che ogni corso ha un codice e che, in media, il numero di edizioni di un corso è pari a cinque. Questo significa che è sufficiente aggiungere un intero di una cifra al codice di un corso per avere un identificatore delle edizioni dei corsi.

9.5.2 Traduzione verso il relazionale

EDIZIONECORSO(Codice, DataInizio, DataFine, Corso, Docente)

LEZIONE(Ora, Aula, Giorno, EdizioneCorso)

DOCENTE(CF, Cognome, Età, CittàNascita, Tipo)

TELEFONO(Numero, Docente), CORSO(Codice, Nome)

ABILITAZIONE(Corso, Docente)

PARTECIPANTE(Codice, CF, Cognome, Età, CittàNascita, Sesso)

PARTECIPAZIONE(Partecipante, EdizioneCorso, Votazione*)

DATORE(Nome, Telefono, Indirizzo)

IMPIEGOPASSATO(Partecipante, Datore, DataInizio, DataFine)

PROFESSIONISTA(Partecipante, Area, Titolo*)

DIPENDENTE(Partecipante, Livello, Posizione, Datore, DataInizio)

10 MySQL

10.1 INTRODUZIONE A MySQL

MySQL: Il DBMS su cui implementeremo ed eseguiremo le basi di dati progettate

MySQL Workbench:

- Definizione (e modellazione grafica) di database, tabelle, viste e relazioni
- Manipolazione dei dati
- Gestione dei ruoli e dei privilegi degli utenti
- Amministrazione e monitoring del DBMS

MySQL Command Line client: Client da riga di comando che consente di eseguire tutti gli statement SQL riconosciuti da MySQL per la definizione, manipolazione e amministrazione di basi di dati

10.2 DBMS

Definizione: Un DBMS è un sistema software che facilita il processo di definizione, costruzione e manipolazione di una base di dati, garantendone la persistenza e consentendo l'accesso concorrente ai dati in essa contenuti da parte di utenti e applicazioni.

10.2.1 Tipologie di DBMS

10.2.1.1 DBMS gerarchico (*hierarchical model*)

- Sviluppato in IBM durante gli anni 60'. Implementato per la prima volta nel 1968 da IBM Information Management System (IMS), il DBMS dei mainframe IBM.
- Rappresentazione dei dati ad albero (segmento radice e segmenti figli), come in un file system
- Rappresenta con efficacia strutture gerarchiche (ovvero con sole relazioni 1-N)
- Non supporta relazioni M-N, che possono essere rappresentate solamente ridondando i dati

10.2.1.2 DBMS reticolare (*network model*)

- Introdotto dal CODASYL nel 1969, utilizzato dal linguaggio COBOL
- Supera il modello gerarchico attraverso il concetto di reticolo, che permette la rappresentazione di relazioni M-N
- Il database assume la forma di un grafo: ogni record è un nodo che può essere associato ad altri nodi tramite puntatori
- Poco diffuso: superato dal modello relazionale

10.2.1.3 DBMS relazionale (*relational model*)

- BMS relazionale (*relational model*) - RDBMS: '70
- Struttura basata su relazioni (tabelle) caratterizzate da attributi aventi un dominio specifico (tipo di dato)
- Dati organizzati in tuple, insieme non ordinati dei valori degli attributi
- L'insieme di tuple di una relazione non è ordinato
- Operazioni definite dall'algebra relazionale
- Possibilità di definire vincoli:
 - Vincoli intra-relazionali: vincoli di dominio e vincoli di tupla
 - Vincoli di chiave
 - Vincoli inter-relazionali (chiavi esterne)
- È il modello più diffuso (Oracle, IBM DB2, MySQL, MS SQL Server, Access, ...)

10.2.1.4 DBMS orientato agli oggetti (*object database*)

- Colmano il gap tra i RDBMS e linguaggi di programmazione ad oggetti
 - Gli oggetti utilizzati dall'applicazione possono essere salvati direttamente su memoria secondaria
 - Implementazione nativa del concetto di ereditarietà
- Poco diffusi: esistono anche ibridi object-relational, come PostgreSQL

10.2.1.5 DBMS NoSQL

- abbandonano il modello relazionale
- Concetto generico che racchiude vari modelli concepiti per applicazioni specifiche
- Mirano a elevate performance per task specifici
- Driver principali: big data, social network, analytics, semantic web
- Maggiori dettagli nel corso di Basi di Dati Complementi
- Esempi:
 - abbandonano il modello relazionale
 - Concetto generico che racchiude vari modelli concepiti per applicazioni specifiche
 - Mirano a elevate performance per task specifici
 - Driver principali: big data, social network, analytics, semantic web

10.2.2 Caratteristiche di un DBMS

Mantenimento della correttezza dei dati

Rispetto alla struttura

Nel tempo (persistenza)

Facilitare l'accesso ai dati da parte di utenti e applicazioni

Gestione degli accessi concorrenti ai dati

Controllo delle transazioni: Transaction Processing

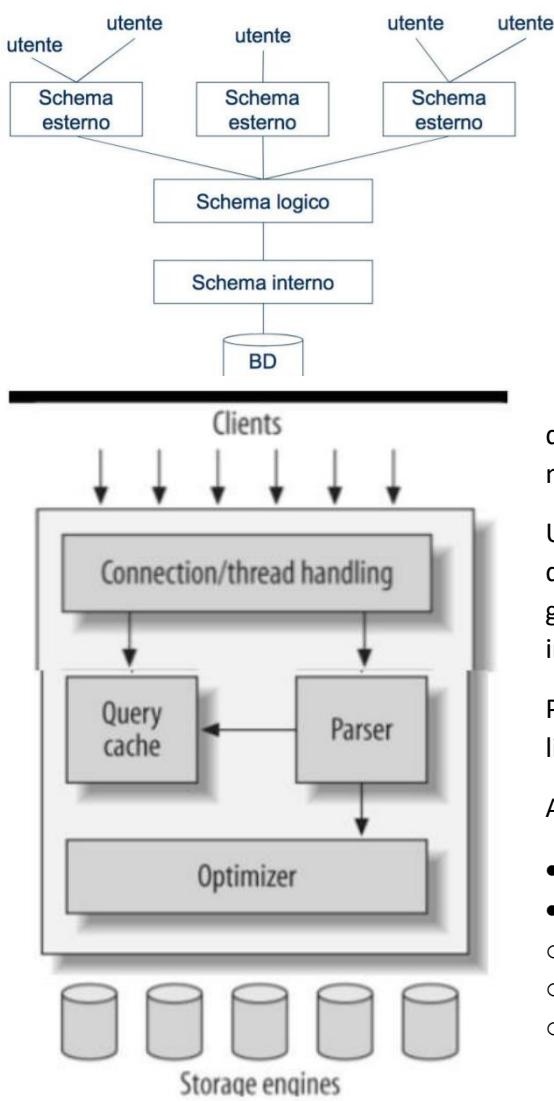
10.2.3 Proprietà ACID

Tutto il codice che viene eseguito all'interno di una transazione gode di proprietà particolari, le cosiddette proprietà "acide"⁸ delle transazioni:

- Atomicità: ogni transazione è completa
- Consistenza: i dati sono validi secondo regole definite
- Isolamento: le transazioni non interferiscono con le altre
- Persistenza: i commit non dovrebbero essere persi

⁸ "ACID properties" che sta per "Atomicity, Consistency, Isolation, Durability". Che traduzione del cazzo.

10.3 ARCHITETTURA DI UN DBMS



Un DBMS è un sistema software che facilita il processo di definizione, costruzione e manipolazione di una base di dati, garantendone la persistenza e consentendo l'accesso ai dati in essa contenuti da parte di utenti e applicazioni.

- Schema esterno: porzione di schema logico visibile all'utente/applicazione
- Schema logico: descrizione della struttura dei dati secondo il modello adottato dal DBMS (tabelle nel caso di un RDBMS)
- Schema interno: rappresentazione degli elementi dello schema logico tramite strutture fisiche di memorizzazione (dipende dallo specifico DBMS)

Un DBMS è un sistema software che facilita il processo di definizione, costruzione e manipolazione di una base di dati, garantendone la persistenza e consentendo l'accesso ai dati in essa contenuti da parte di utenti e applicazioni.

Parte server (il DBMS) e parte client (connettori per i linguaggi di programmazione, driver JDBC/ODBC)

Alcune criticità:

- Efficienza: ottimizzazione delle richieste
- Meccanismi per garantire:
 - l'affidabilità e la persistenza (fault tolerance)
 - il controllo di concorrenza
 - il controllo degli accessi

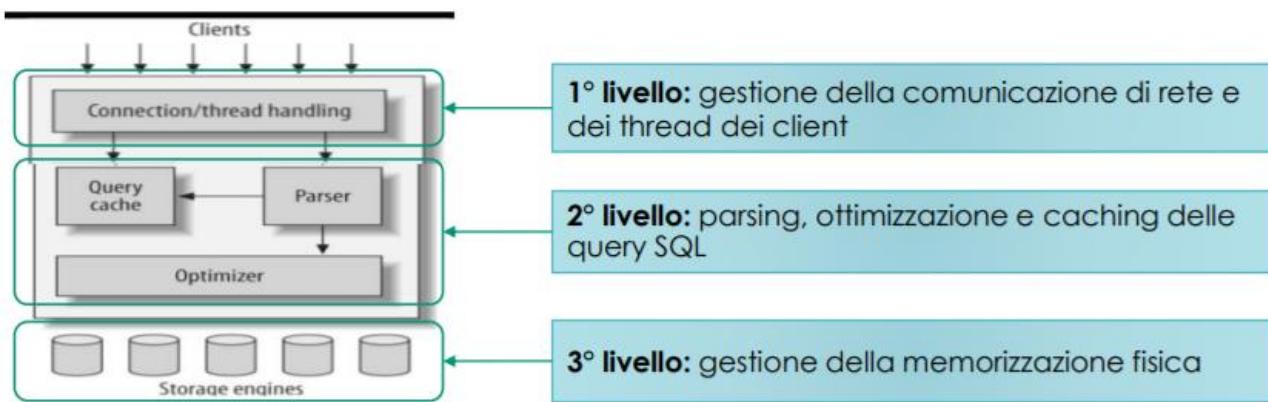
10.4 Cos'È MySQL

È un RDBMS open source di proprietà di Oracle. Due versioni:

- Versione Community: distribuita con licenza GPL
- Versione Enterprise: distribuita con una licenza commerciale proprietaria, aggiunge servizi di supporto e alcuni strumenti di gestione e amministrazione non open-source

Supporta Linux, Windows e MacOSX. Supporta standard SQL ANSI. Dispone di connettori e driver ODBC, JDBC, e si interfaccia con i principali linguaggi di programmazione e Supporta ricerche full-text.

Supporta transazioni sia locali sia distribuite (XA Transactions). ACID-compliant: garantisce affidabilità e persistenza.



Un singolo processo server rimane in ascolto su una socket (di default sulla porta 3306) e attiva un thread per ogni connessione in ingresso.

È possibile ospitare più server MySQL sullo stesso host (su porte differenti).

Ogni server gestisce uno o più database e fornisce meccanismi di autenticazione e autorizzazione degli utenti.

10.4.1 MySQL engines

MySQL viene distribuito con 8 storage engine. Possono essere aggiunti, rimossi, attivati e disattivati a runtime. Il comando SHOW ENGINES elenca gli engine supportati e disponibili.

Ognuno ha caratteristiche differenti, la scelta dipende dalle esigenze dell'applicazione. L'engine è definito a livello di tabella.

10.4.1.1 InnoDB

- È l'engine di default a partire dalla versione 5.5
- Transazionale, DML ACID-compliant, lock a livello di tupla, supporto ai vincoli di integrità referenziale
- Consente letture e scritture concorrenti
- Supporto alle ricerche full-text solo dalla versione 5.6.4
- Tablespace composti da uno o più file: ognuno contiene una o più tabelle

10.4.1.2 MyISAM

- Era l'engine di default fino alla versione 5.1
- Non transazionale, non ACID-compliant, lock a livello di tabella, non supporta vincoli di integrità
- Elevate prestazioni in lettura, scritture concorrenti non consentite
- Supporto alle ricerche full-text
- Ogni tabella ha un proprio file di schema e un file di dati

10.4.1.3 Memory

- Mantiene i dati in memoria principale: non è persistente!
- Elevate prestazioni in lettura e scrittura
- Utile per implementare buffer, cache, storage temporanei, in-memory processing

10.4.1.4 CSV

- Memorizza i dati in file di testo in formato CSV
- Non supporta gli indici

10.4.1.5 Archive

- Consente la memorizzazione compressa di grandi quantità di dati per scopi di archiviazione
- Non supporta gli indici

10.4.1.6 Merge

- Consente di definire una tabella come unione di tabelle MyISAM identiche
- Di default permette accessi read-only. Inserimenti possibili solo nella prima o nell'ultima tabella del set

10.4.1.7 Federated

Consente accedere a tabelle memorizzate fisicamente in un DBMS remoto come se fossero in locale

10.4.2 Strumenti

Command Line Client: Client a riga di comando per eseguire statement DDL, DML e DCL

Varie utility di amministrazione da riga di comando

- mysqladmin: operazioni di amministrazione del server MySQL
- mysqlcheck: verifica dell'integrità e riparazione dei file di dati delle tabelle
- mysqldump: backup di database

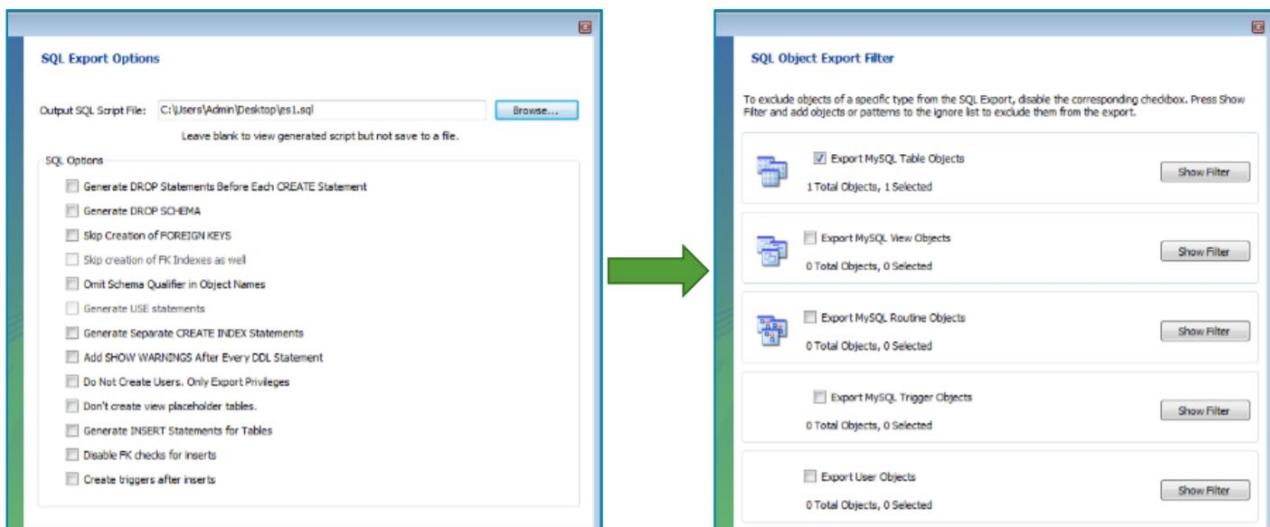
MySQL Workbench

- Modellazione fisica e gestione di database MySQL
- Amministrazione di server MySQL

10.5 DATA MODELING – FORWARD ENGINEERING

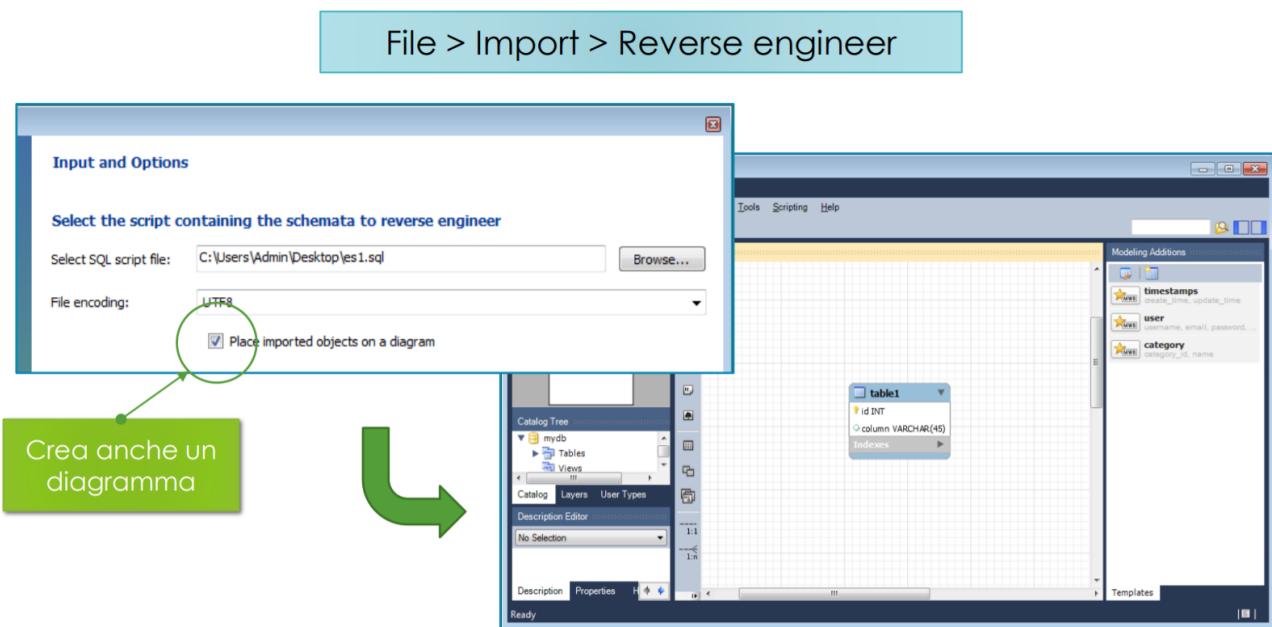
Consente di generare uno script SQL per la costruzione dei database modellati

File > Export > Forward engineer

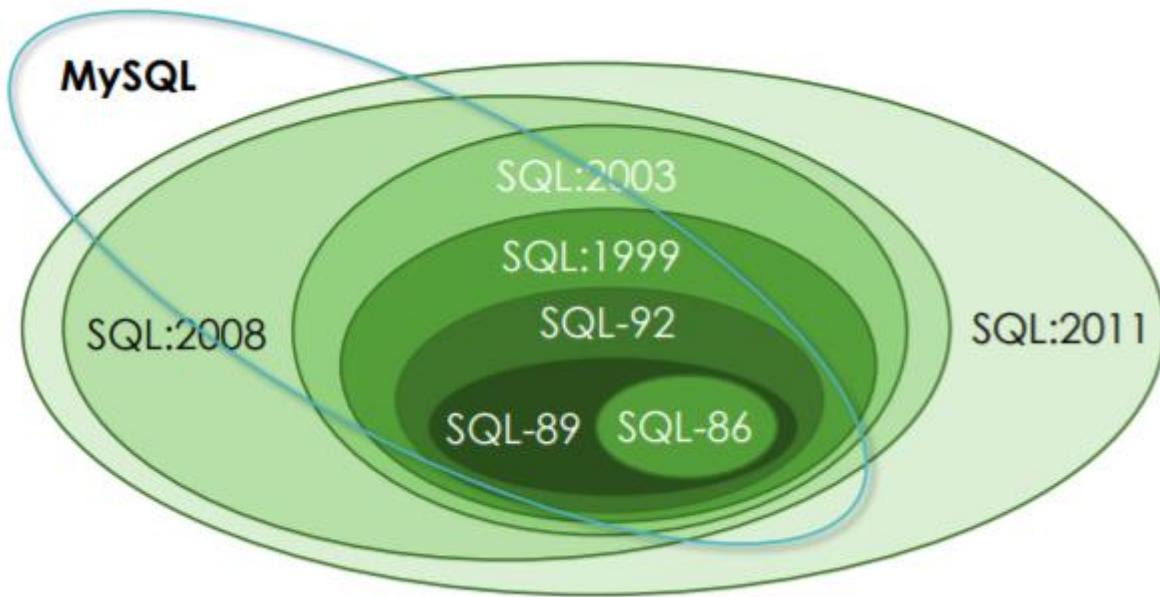


10.6 DATA MODELING – REVERSE ENGINEERING

Consente di generare un modello a partire da uno script SQL



10.7 SQL



MySQL implementa parte degli standard SQL, fino a SQL:2008. MySQL estende SQL aggiungendo alcune funzionalità.

DDL – Data Definition Language

- Definizione e modifica dello schema del DB (db, tabelle, colonne, viste, ...)
- Operazioni CREATE, ALTER, DROP

```
mysql> create table studente (matricola int, nome varchar(100));
mysql> drop table esame;
```

DML – Data Manipulation Language

- Interrogazione e modifica dei dati
- Operazioni **CRUD**: Create, Read, Update, Delete

```
mysql> select * from studente;
mysql> update studente set name = "Mario";
```

DCL – Data Control Language

- Controllo del DBMS e dei database

```
mysql> use univ;
mysql> show databases;
```

10.7.1 DCL

- Elenco dei database
 - SHOW DATABASES;
 - SHOW SCHEMAS;
- Selezione di un database
 - USE univ;
- Elenco delle tabelle
 - SHOW TABLES FROM univ;
 - SHOW TABLES;
- Ispezione della struttura di una tabella
 - DESCRIBE univ.studente;
 - DESCRIBE studente;
 - SHOW COLUMNS FROM studente;
 - SHOW COLUMNS FORM studente FORM univ;
- Indici definiti su una tabella
 - SHOW INDEX FROM univ.studente;
 - SHOW INDEX FORM studente FROM univ;
 - SHOW INDEX FROM studente;

10.7.2 DDL

Statement CREATE, ALTER e DROP per database e tabelle

- Creazione di un database
 - CREATE DATABASE univ;
 - CREATE SCHEMA univ;
 - CREATE DATABASE IF NOT EXISTS univ;
- Eliminazione di un database
 - DROP DATABASE univ;
 - DROP SCHEMA univ;
- Specificare il character set e la collation

- CREATE SCHEMA IF NOT EXISTS univ CHARSET='UTF8';
- CREATE SCHEMA IF NOT EXISTS univ CHARSET='utf8' COLLATE='utf8_general_ci';

10.7.3 Tipi di dati principali

10.7.3.1 Stringa

Lunghezza fissa: CHAR(N) dove N = 0-255 caratteri

Lunghezza variabile: VARCHAR(N) dove N = 0-65.536 caratteri), TEXT, MEDIUMTEXT, LONGTEXT

10.7.3.2 Interi (signed e unsigned)

- TINYINT (8 bit)
- SMALLINT (16 bit)
- MEDIUMINT (24 bit)
- INT (32 bit)
- BIGINT (64 bit)

10.7.3.3 Tipi decimali (signed e unsigned)

- Virgola mobile:
 - FLOAT (32 bit)
 - DOUBLE (64 bit)
- Virgola fissa: DECIMAL(M, D)

10.7.3.4 Tipi temporali

- DATE
- DATETIME
- TIMESTAMP(data minima 1/1/1970, data massima 19/01/2038)
- YEAR

10.7.3.5 Tipi binari

- BIT (1-64 bit)
- BINARY(N) (0-255 byte)
- VARBINARY(N) (0-65.536 byte)
- BLOB
- LONGBLOB

10.7.3.6 Enumerazioni

ENUM("VALORE1", "VALORE2", "VALORE3", ...)

10.7.3.7 Booleani

Non sono supportati, benché lo standard SQL definisca il tipo BOOLEAN. MySQL converte BOOLEAN in TINYINT(1)

10.7.4 Auto increment

Una colonna di tipo intero può essere dichiarata ad incremento automatico con la keyword AUTO_INCREMENT. Il suo valore viene automaticamente generato all'inserimento del record, tramite un contatore incrementale interno al DBMS associato alla tabella.

Il primo corso avrà codice 1, il secondo 2, ecc.... Il valore iniziale del contatore può essere modificato tramite, ad esempio, "ALTER TABLE univ.corso AUTO_INCREMENT = 100;". I codici che tornano disponibili

```

CREATE TABLE IF NOT EXISTS univ.corso (
    Codice INT NOT NULL AUTO_INCREMENT
PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    ...
) ENGINE = InnoDB;

```

Figura 28 - Auto_increment

```

CREATE TABLE IF NOT EXISTS univ.studente (
    matricola VARCHAR(6) NOT NULL,
    cognome VARCHAR(40) NOT NULL,
    nome VARCHAR(40) NOT NULL,
    PRIMARY KEY (matricola),
    CONSTRAINT un_studente UNIQUE
    (cognome, nome)
) ENGINE=InnoDB;

```

Figura 29 - UNIQUE

```

CONSTRAINT constraint_name
FOREIGN KEY foreign_key_name (columns)
REFERENCES parent_table(columns)
ON DELETE action
ON UPDATE action

CONSTRAINT fk_studente
FOREIGN KEY (matricola)
REFERENCES univ.studente
(matricola)

ON UPDATE CASCADE
ON DELETE RESTRICT,

```

Figura 30 - Vincoli di integrità referenziale

dopo l'eliminazione di corsi non vengono riusati, a meno che il contatore non venga reimpostato al valore iniziale.

10.7.5 Vincoli di tupla

Sono vincoli di unicità. Supponiamo che non possano esistere due studenti con lo stesso nome e cognome.

MySQL non supporta altri vincoli di tupla né vincoli di dominio. I costrutti DOMAIN e CHECK di SQL non sono supportati.

10.7.6 Vincoli di integrità referenziale

La clausola CONSTRAINT permette di definire un nome per il vincolo.

La clausola FOREIGN KEY specifica la colonna nella tabella figlia che si riferisce alla chiave primaria della tabella genitore.

La clausola REFERENCE specifica la tabella genitore e la colonna alla quale si riferisce la tabella figlia.

La clausola ON DELETE e ON UPDATE permettono di definire cosa accade quando il record della tabella genitore viene cancellato/modificato.

10.7.7 Propagation constraints

Consentono di definire l'azione con cui propagare gli aggiornamenti riguardanti valori oggetto di un vincolo di integrità referenziale verso la tabella in cui il vincolo è definito (tabella dipendente).

Azione	In caso di modifica	In caso di eliminazione
CASCADE	Valore automaticamente aggiornato	Record automaticamente eliminato
RESTRICT	Se l'operazione viola l'integrità, viene negata	Se l'operazione viola l'integrità, viene negata
NO ACTION	Secondo lo standard SQL equivale a RESTRICT, ma l'integrità è verificata solo al termine dell'esecuzione della transazione (verifica ritardata). In realtà in MySQL ciò avviene immediatamente, per cui si comporta in maniera identica a RESTRICT	
SET DEFAULT	Valore impostato al valore di default della colonna	Valore impostato al valore di default della colonna
SET NULL	valore impostato a NULL	valore impostato a NULL

10.7.8 Alter table

- Aggiunta di colonne

```
ALTER TABLE univ.studente
```

```
    ADD COLUMN codice_fiscale CHAR(16) NOT NULL AFTER cognome,
    ADD COLUMN laureato TINYINT(1) NOT NULL DEFAULT 0;
```

- rimozione di colonne

```
ALTER TABLE univ.studente DROP COLUMN codice_fiscale;
```

- Modifica di colonne

```
ALTER TABLE univ.studente
```

```
    CHANGE COLUMN matricola numero_matricola CHAR(7) NOT NULL,
    MODIFY COLUMN cognome VARCHAR(100) NOT NULL
```

- Impostazione ed eliminazione del valore di default

```
ALTER TABLE univ.studente
```

```
    ALTER COLUMN laureato DROP DEFAULT,
    ALTER COLUMN codice_fiscale SET DEFAULT "000000000000"
```

- Definizione della chiave primaria

```
ALTER TABLE univ.studente ADD PRIMARY KEY (matricola);
```

- eliminazione della chiave primaria

```
ALTER TABLE univ.studente DROP PRIMARY KEY;
```

- aggiunta di un vincolo di unicità e di integrità referenziale

```
ALTER TABLE univ.studente
```

```
    ADD CONSTRAINT un_codice_fiscale UNIQUE (codice_fiscale),
    ADD CONSTRAINT fk_citta
```

```
        FOREIGN KEY (cod_citta)
```

```
        REFERENCES citta (codice)
```

```
        ON UPDATE CASCADE ON DELETE SET NULL;
```

- eliminazione di un vincolo di unicità

```
ALTER TABLE univ.studente DROP un_codice_fiscale;
```

- eliminazione di un vincolo di integrità referenziale

```
ALTER TABLE univ.studente DROP FOREIGN KEY fk_citta;
```

- ridenominazione di una tabella

```
ALTER TABLE univ.corso RENAME TO univ.insegnamento;
```

```
... oppure utilizzando lo statement RENAME ...
```

```
RENAME TABLE univ.corso TO univ.insegnamento;
```

10.7.9 Eliminazione di una tabella

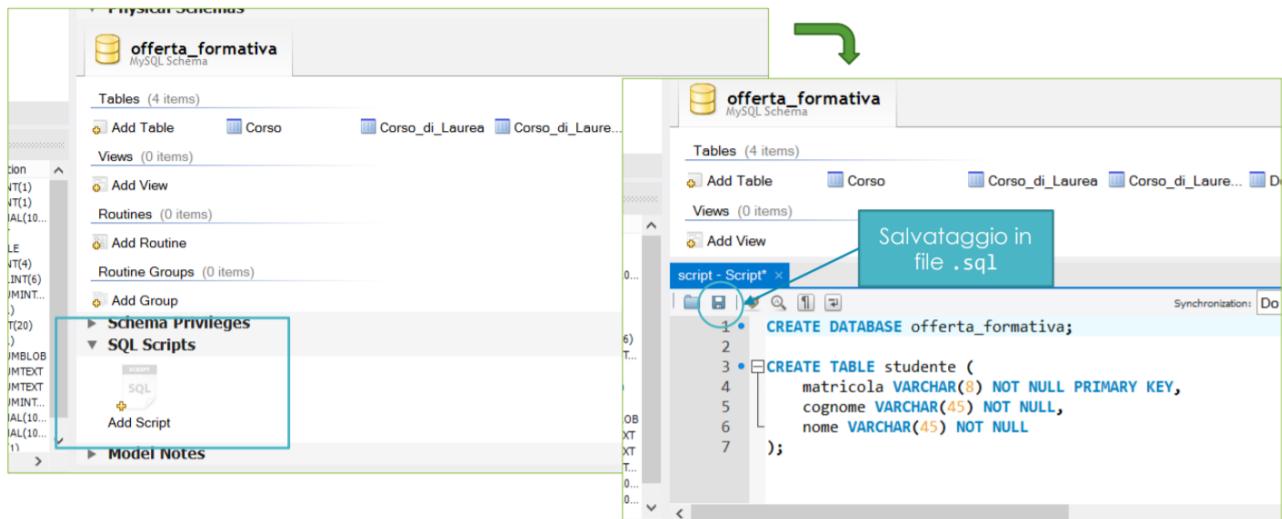
```
DROP TABLE univ.insegnamento;
```

10.7.10 MySQL script

Uno script è un file di testo che contiene statement SQL

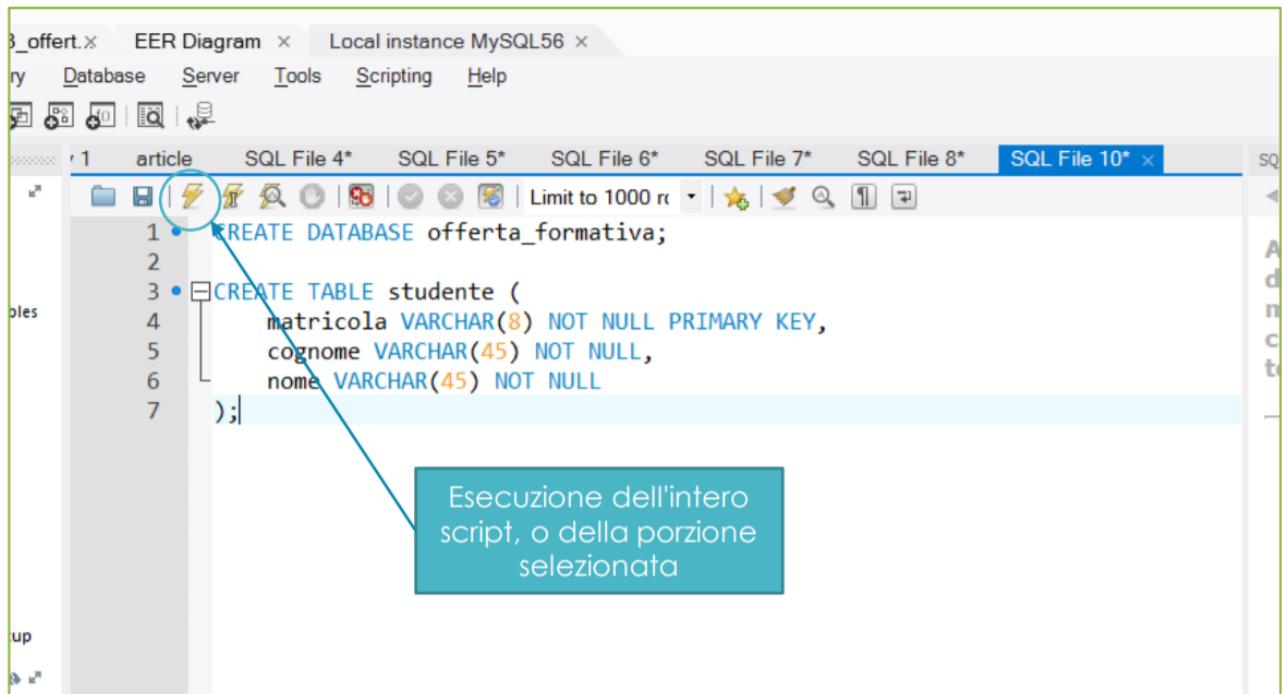
- Utili per creare la struttura di un database o per manipolare i dati in batch
- Uno script di creazione di un database contiene solo statement DDL

MySQL Workbench consente di creare script nel file di modello



oppure si possono creare script dopo aver stabilito una connessione ad un DBMS (locale o remoto).

File > New Query Tab



10.7.11 MySQL Select

```
SELECT *
FROM studente
WHERE cognome = "Rossi";
```

Lo statement SQL SELECT consente di estrarre un insieme di record dal database.

Qua accanto una query per selezionare tutte le colonne tramite l'operatore *, altrimenti si possono esplicitare i nomi delle colonne desiderate.

La clausola WHERE consente di ridurre l'insieme di record selezionati includendo unicamente quelli che rendono vera una specifica espressione booleana. L'espressione può contenere operatori e predicati, applicare funzioni ed eseguire operazioni logiche, binarie e algebriche.

- Operatori logici: AND, OR, XOR, NOT, IS NULL, IS NOT NULL
 - Attenzione alla logica a 3 valori: il confronto con il valore NULL produce NULL!
- Operatori di confronto: =, >, <, >=, <=, <> (oppure !=)

Poiché la SELECT consente di selezionare un sottoinsieme delle colonne di una tabella, l'insieme di risultati può contenere duplicati. La keyword DISTINCT (scritta “SELECT DISTINCT”) consente di rimuovere i duplicati dal risultato.

La keyword DISTINCT considera diversi due record se differiscono per almeno un valore di una colonna.

È possibile applicare operazioni e funzioni sia ai valori da selezionare sia nelle condizioni di selezione

- Operatori aritmetici: +, -, *, /, % (oppure MOD), DIV (divisione intera)
- Operatori bit a bit: & (and), | (or), ^ (xor), ~ (not)
- Funzioni matematiche:
 - ABS(v): valore assoluto di v
 - CEIL(v): minor valore intero $\geq v$
 - FLOOR(v): maggior valore intero $\leq v$
 - SIN(v), COS(v), TAN(v), ...: funzioni trigonometriche seno, coseno, tangente, ...
 - LOG(v), LN(v), LOG10(v), LOG2(v): logaritmo naturale, in base 10 e in base 2 di v
 - RAND(): valore casuale floating-point tra 0.0 e 1.0
 - ROUND(v, d): arrotonda v a d cifre decimali (d di default è 0)

10.7.12 Funzioni su stringhe

- LENGTH(s): numero di byte della stringa s
 - Corrisponde al numero di caratteri tranne nel caso in cui la stringa contenga caratteri multi-byte (ad esempio caratteri accentati nel caso del character set UTF-8)
- CHAR_LENGTH(s): numero di caratteri della stringa s
 - Eventuali caratteri multi-byte vengono contati come un solo carattere
- CONCAT(s1, s2, ...): concatenazione delle stringhe s1, s2, ...
- LCASE(s): converte la stringa s in minuscolo (lowercase)
- UCASE(s): converte la stringa s in maiuscolo (uppercase)
- REPLACE(s, from, to): rimpiazza ogni occorrenza della stringa from presente nella stringa s, con la stringa to
- LEFT(s, n): restituisce i primi n caratteri della stringa s
- RIGHT(s, n): restituisce gli ultimi n caratteri della stringa s

- LTRIM(s), RTRIM(s), TRIM(s): rimuovono eventuali blank rispettivamente all'inizio, alla fine e da entrambe le estremità della stringa s
- SUBSTRING(s, p, l) oppure SUBSTRING(s FROM p FOR l): estrae dalla stringa s una sottostringa di lunghezza l, partendo dal carattere in posizione p
 - Se l è negativo, p indica la posizione a partire dalla fine della stringa

10.7.13 Funzioni temporali

- NOW(), CURRENT_TIMESTAMP(): data e ora attuale
- CURTIME(), CURRENT_TIME(): ora attuale
- CURDATE(), CURRENT_DATE(): data attuale
- DATE(dt): estrae la data dal valore dell'espressione temporale dt
- TIME(dt): estrae l'ora dal valore dell'espressione temporale dt
- YEAR(dt), MONTH(dt), DAY(dt), HOUR(dt), MINUTE(dt), SECOND(dt),
- MICROSECOND(dt): estraggono dall'espressione temporale dt rispettivamente l'anno, il mese, il giorno nel mese, l'ora, i minuti, i secondi e i microsecondi.
- DAYNAME(dt): nome del giorno della settimana rappresentato dall'espressione temporale dt
- DATE_FORMAT(dt, format): formatta la data dt secondo il formato format
 - DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y %H:%i:%s') → "Sunday October 2009 22:23:00"
- ADDDATE(dt, INTERVAL expr unit), SUBDATE(dt, INTERVAL expr unit): rispettivamente aggiunge o sottrae dalla data dt il valore expr di unità temporali unit
 - ADDDATE('2015-05-12', INTERVAL 10 DAYS) → '2015-05-22'
 - SUBDATE('2015-03-08', INTERVAL 2 YEARS) → '2013-03-08'
 - Al posto di ADDDATE e SUBDATE è possibile utilizzare anche gli operatori + e -
- ADDTIME(dt1, dt2), SUBTIME(dt1, dt2): rispettivamente somma o sottrae le espressioni temporali dt1 e dt2
 - ADDTIME('01:00:00.999999', '02:00:00.999998') → '03:00:01.999997'
 - SUBTIME('01:00:00.999999', '02:00:00.999998') → '-00:59:59.999999'
- DATEDIFF(dt1, dt2): numero di giorni di differenza tra dt1 e dt2
 - Se dt1 < dt2 il numero di giorni è negativo

10.7.14 Controllo di flusso

Funzioni di controllo di flusso:

- CASE v WHEN v1 THEN r1 WHEN v2 THEN r2 ... ELSE r END
 - Confronta v con i valori (v1, v2, ...) indicati nelle clausole WHEN, e ritorna il risultato indicato nella corrispondente clausola THEN (r1, r2, ...). Se v è diverso da tutti i valori elencati ritorna il risultato r indicato nella clausola ELSE.
- CASE WHEN cond1 THEN r1 WHEN cond2 THEN r2 ... ELSE r END
 - Valuta le condizioni (cond1, cond2, ...) indicati nelle clausole WHEN, e restituisce il valore (r1, r2, ...) indicato dalla clausola THEN corrispondente alla prima condizione che produce TRUE. Se tutte le condizioni producono FALSE restituisce il valore r indicato dalla clausola ELSE.
- IF(expr, exprTrue, exprElse)
 - Valuta l'espressione expr: se produce TRUE ritorna il valore dell'espressione exprTrue, altrimenti (se FALSE o NULL) restituisce il valore dell'espressione exprElse.
- IFNULL(expr, exprNull)
 - Se l'espressione expr ha valore NULL ritorna il valore dell'espressione exprNull, altrimenti ritorna il valore di expr.

- `NULLIF(expr1, expr2)`
 - Ritorna NULL se `expr1 = expr2`, altrimenti ritorna il valore dell'espressione `expr1`

10.7.15 Selezione da più tabelle

```
SELECT st.matricola, cl.nome AS
corso_laurea
FROM studente AS st, corso_laurea
AS cl
WHERE st.id_corso_laurea = cl.id;
```

Due (o più) tabelle possono essere combinate attraverso l'operazione di join (prodotto cartesiano). Produce tutte le possibili combinazioni tra le tuple delle tabelle coinvolte.

Lo statement SELECT effettua il join delle tabelle elencate nella clausola FROM, In questo caso si parla di join implicito.

L'eventuale condizione di join è definita nella clausola WHERE.

10.7.16 Join

```
SELECT st.matricola, cl.nome AS
corso_laurea
FROM studente AS st
JOIN corso_laurea AS cl ON cl.id =
st.id_corso;
```

Il join in SQL è esprimibile tramite la sintassi "JOIN table ON condition". In questo caso si parla di join esplicito. La clausola ON definisce la condizione di join.

10.7.17 Tipologie di join

Il join visto fin qui è detto join interno (inner join) perché esclude tutte le tuple per le quali la condizione di join non restituisce valori NULL. JOIN e INNER JOIN sono equivalenti.

SQL prevede anche alcune forme di join esterno (outer join), che preserva le tuple la cui condizione di join produce un valore NULL

- LEFT OUTER JOIN (o LEFT JOIN): mantiene solo le tuple della tabella alla sinistra dell'operatore di join
- RIGHT OUTER JOIN (o RIGHT JOIN): mantiene solo le tuple della tabella alla destra dell'operatore di join
- FULL OUTER JOIN: mantiene le tuple di entrambe le tabelle coinvolte nel join
 - Non è supportato da MySQL (e anche da molti altri DBMS), ma emulabile tramite UNION ALL

10.7.18 Ordinamento

```
SELECT s.matricola, s.comune
FROM studente AS s INNER JOIN comune
AS c ON s.comune = c.nome
ORDER BY c.abitanti DESC,
YEAR(s.data_iscrizione) ASC;
```

L'insieme dei risultati dell'interrogazione può essere ordinato rispetto ad una o più colonne utilizzando la clausola ORDER BY.

L'ordinamento può essere effettuato anche rispetto a colonne di tabelle diverse o rispetto a valori calcolati tramite operazioni e funzioni.

10.7.19 Aggregazione

```
SELECT COUNT(voto_laurea) AS
laureati, AVG(voto_laurea) AS media
FROM studente
WHERE YEAR(data_iscrizione) < 2006;
```

Gli operatori di aggregazione sono funzioni applicabili all'insieme di tuple risultanti da una interrogazione. Poiché si applicano all'intero set risultante, quando vengono utilizzati la SELECT non può selezionare valori diversi da quelli calcolati con gli operatori di aggregazione.

Valutano l'espressione data per ogni tupla del risultato, e aggregano i risultati.

- COUNT(expr): conta il numero di valori non NULL
 - COUNT(*) è un'eccezione! Conta il numero di risultati anche se NULL
- SUM(expr): calcola la somma dei valori
- AVG(expr): calcola la media dei valori
- MIN(expr): calcola il minimo dei valori
- MAX(expr): calcola il massimo dei valori

10.7.20 Raggruppamento

```
SELECT
    YEAR(data_iscrizione) AS anno,
    comune,
    COUNT(*) AS studenti,
    AVG(voto_laurea) AS voto_medio
FROM studente
WHERE YEAR(data_iscrizione) < 2006
GROUP BY YEAR(data_iscrizione),
comune
ORDER BY anno ASC, comune ASC
HAVING studenti >= 2;
```

La clausola GROUP BY consente di partizionare l'insieme di tuple risultanti dall'interrogazione in gruppi omogenei di tuple

Il raggruppamento può avvenire in base a valori uguali:

- di una o più colonne
- di un'espressione calcolata su ogni tupla

Quando si effettua un raggruppamento gli operatori di aggregazione vengono valutati per ogni gruppo

I valori selezionabili dalla SELECT possono essere solamente:

- uno dei valori calcolati dagli operatori di aggregazione (se utilizzati)

- uno dei valori delle colonne o delle espressioni utilizzate come criterio di raggruppamento

Attraverso la clausola HAVING è possibile filtrare il risultato finale. Equivalente alla clausola WHERE, ma valutata solo dopo il raggruppamento. Permette di applicare anche condizioni sui risultati degli operatori di aggregazione.

10.7.21 Interrogazioni nidificate

```
SELECT ...
FROM ...
WHERE expr operator
(SELECT ... FROM ...)
...;
```

```
SELECT nome
FROM comune
WHERE codice <> ALL (
    SELECT comune
    FROM studente
    WHERE nome = "Alice"
);
```

Consentono di costruire interrogazioni complesse, il cui risultato dipende dal risultato di altre sotto-interrogazioni (o subquery). Una subquery può essere utilizzata:

- Nella clausola WHERE: per filtrare i record selezionati dalla query principale sulla base dell'esito della subquery
- Nella clausola FROM: per arricchire l'insieme di tuple su cui viene effettuata l'interrogazione principale con un set di tuple fornito dalla subquery

Non c'è limite al numero di subquery e di livelli di nidificazione. Aumentano la potenza espressiva del linguaggio. Molte query semplici possono essere riscritte utilizzando query nidificate. A volte migliora la leggibilità (ma spesso a scapito delle performance).

Attenzione: non è vero il contrario! Alcune interrogazioni sono esprimibili solo utilizzando query nidificate.

Con gli operatori di confronto è possibile applicare dei quantificatori:

- ANY (oppure SOME): quantificatore esistenziale, Produce true se l'operatore restituisce true nel confronto con almeno una delle tuple risultanti dalla subquery
- ALL: quantificatore universale, Produce true se l'operatore restituisce true nel confronto con tutte le tuple risultanti dalla subquery

I quantificatori consentono di applicare gli operatori di confronto anche se la subquery produce più di un valore.

SQL consente di utilizzare il quantificatore esistenziale anche per verificare l'esistenza di almeno un record nel risultato della subquery

- EXISTS: true se la subquery produce almeno un risultato
- NOT EXISTS: true se la subquery non produce risultati

```
SELECT ...
FROM ..., (SELECT ... FROM ...)
AS alias, ...
```

```
WHERE ...
...;
```

In questo caso non si confronta il valore di un'espressione con il risultato della subquery, ma si verifica solo il numero di risultati prodotti da essa

le subquery possono essere inserite anche nella clausola from per generare una tabella ristretta. Questa deve essere però rinominata tramite un alias.

10.7.22 Interrogazioni insiemistiche

Consentono di effettuare operazioni su insiemi di risultati di interrogazioni differenti, ossia sui risultati di due SELECT. I due insiemi devono essere omogenei: devono avere colonne in uguale numero e tipo di dati

Operatori insiemistici:

- UNION: produce l'unione dei due insiemi di risultati
- INTERSECT: produce l'intersezione dei due insiemi di risultati
- EXCEPT: produce la differenza tra i due insiemi, restituendo tutte e sole le tuple del primo insieme che non sono presenti nel secondo insieme

Il risultato è un insieme di risultati privo di duplicati. La definizione matematica di insieme impone che contenga solo elementi distinti. Per mantenere i duplicati è possibile specificare anche la keyword ALL

Attenzione! INTERSECT ed EXCEPT non sono supportati da MySQL, sono però emulabili tramite join e/o subquery.

10.7.23 Inserimento

Ci sono due modi di inserire valori:

- Inserimento singolo
- Inserimento multiplo

Inserimento singolo	Inserimento multiplo
<pre>INSERT INTO studente SET matricola = "972061", cognome = "Rossi", nome = "Mario";</pre>	<pre>INSERT INTO studente (matricola, cognome, nome) VALUES ("972061", "Rossi", "Mario"), ("092682", "Brambilla", "Roberto");</pre>

10.7.24 Modifica/aggiornamento

```
UPDATE studente
SET cognome = "Verdi", codice_fiscale =
"VRDPLA92E03F205L"
WHERE matricola = "073448"
LIMIT 1;
```

LIMIT serve a indicare il numero di risultati da modificare.

L'aggiornamento può coinvolgere dati di più tabelle. Le tabelle coinvolte possono essere selezionate con join implicito o esplicito, o con una subquery.

10.7.25 Eliminazione

```
DELETE FROM studente
WHERE matricola = "073448"
LIMIT 1;
```

Ha una sintassi simile a quella del SELECT.

Anche l'eliminazione può coinvolgere dati di più tabelle.

10.7.26 Safe mode

MySQL per impostazione di default cerca di impedire modifiche massive accidentali. Impedisce DELETE e UPDATE se la condizione nel WHERE non coinvolge la chiave primaria.

Il safe mode è bypassabile in due modi:

- Includendo la clausola LIMIT 1
- Disabilitandolo per la sessione in corso tramite "SET SQL_SAFE_UPDATES = 0;"

10.7.27 Importazione da CSV

MySQL consente di importare data set da file CSV. Un file CSV rappresenta dati tabellari, un record per riga, e il valore di ogni colonna è separato da una virgola (o da un altro carattere separatore). La prima riga descrive i nomi delle colonne.

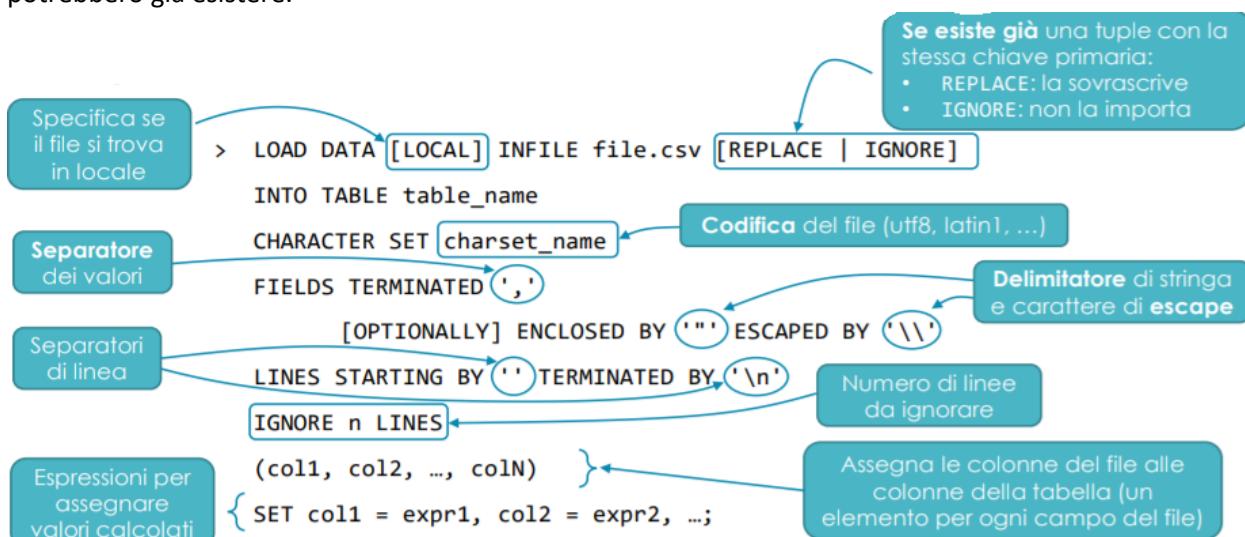
```
id,first_name,last_name,birth_date,address,city,country
1,Kaitlin,Holman,1980-07-04,5279 Parturient Road,Gonars,France
2,Graham,Williams,1978-06-02,843-1422 Fermentum Road,Enna,Togo
3,Quail,Burns,1973-04-06,"184-1266 Sem, Rd.",Broxburn,Canada
```

```
LOAD DATA INFILE file.csv
INTO TABLE table_name;
```

Per importare da file CSV si usa lo statement LOAD DATA INFILE.

Il formato CSV è molto "variabile" (Diversi separatori, linee in eccesso, diversi caratteri di fine riga, diverse codifiche...).

Potrebbe essere necessario escludere delle colonne, o elaborare i valori prima di inserirli e alcune tuple potrebbero già esistere.



MySQL Workbench consente di importare dati da file CSV, ma non consente di definire alcuna opzione.

- Non possono esistere righe o colonne in eccesso
- I valori possono essere separati unicamente da virgole
- I caratteri separatori di linea devono essere quelli utilizzati dal sistema operativo in uso
- Non devono esistere duplicati
- Non è possibile applicare espressioni sui valori da inserire

10.7.28 Esportare un dataset

```
SELECT ...
    INTO OUTFILE file.csv
        FIELDS TERMINATED BY ','
        [OPTIONALLY] ENCLOSED BY ""
        LINES TERMINATED BY '\n'
        CHARACTER SET utf8
    FROM ... WHERE ...;
```

da file CSV, Workbench non consente di specificare opzioni.

10.7.29 Viste in MySQL

```
CREATE
[ALGORITHM =
{UNDEFINED|MERGE|TEMPTABLE}]
VIEW view_name AS SELECT ... FROM
... ;
```

UPDATE e DELETE.

Gli algoritmi di processing definiscono la modalità di processing delle query effettuate

sulla vista

- MERGE: La query viene riscritta effettuando il merge tra essa e la SELECT che definisce la vista
- TEMPTABLE: Il DBMS esegue prima l'interrogazione che definisce la vista e inserisce il risultato in una tabella temporanea. La query viene poi eseguita sulla tabella temporanea, che viene poi eliminata

MERGE è più efficiente ma utilizzabile solo se la vista non è definita con:

- Operatori di aggregazione: COUNT(), MIN(), MAX(), AVG(), ...
- Raggruppamenti: GROUP BY
- Operatori insiemistici: UNION, UNION ALL
- Clausole DISTINCT, HAVING e LIMIT

Se l'algoritmo è UNDEFINED, MySQL utilizza, se possibile, MERGE.

10.7.30 Viste aggiornabili

Una vista è aggiornabile se c'è un mapping 1 a 1 tra le sue tuple e quelle di una tabella fisica.

In pratica, in MySQL una vista è aggiornabile solo se usa l'algoritmo di processing MERGE e se la definizione non contiene:

- Operatori di aggregazione: COUNT(), MIN(), MAX(), AVG(), ...
- Raggruppamenti: GROUP BY

- Operatori insiemistici: UNION, UNION ALL
- Clausole DISTINCT e HAVING
- Subquery nella clausola SELECT, o anche nella clausola WHERE se si tratta di subquery correlate
- Riferimenti ad altre viste non aggiornabili nella clausola FROM
- Outer join: LEFT JOIN, RIGHT JOIN

Inoltre, la INSERT è supportata solo se la vista seleziona tutte le colonne delle tabelle coinvolte che non hanno un valore di default, e se le colonne selezionate non sono espressioni definite sulle colonne delle tabelle referenziate.

Solo una delle tabelle coinvolte nella vista può essere aggiornata: Nel caso di UPDATE, la clausola SET può fare riferimento solamente a colonne di una sola delle tabelle.

10.7.31 Modifica/eliminazione viste

```
ALTER
[ALGORITHM = { UNDEFINED | MERGE | TEMPTABLE}]
VIEW view_name AS SELECT ... FROM ... ;

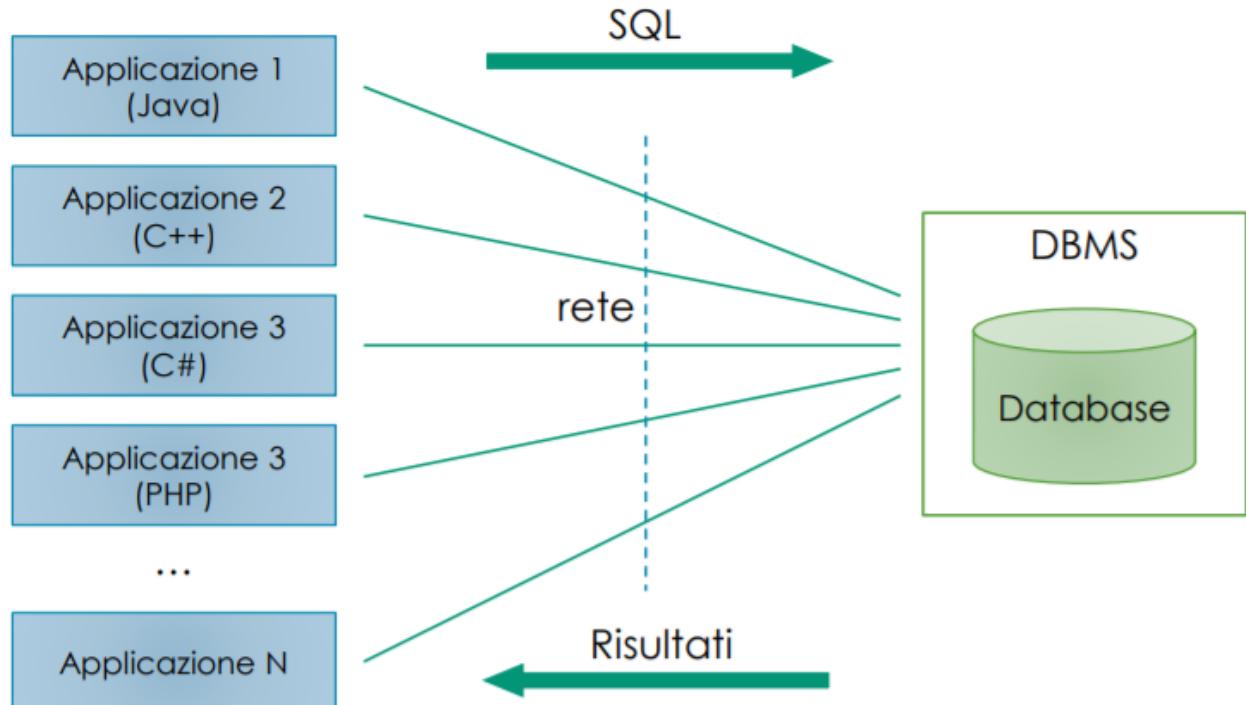
DROP VIEW [IF EXISTS] media_voti;
```

Per modificare una vista si usa lo statement ALTER VIEW. La sintassi è identica a quella dello statement CREATE VIEW. Poiché le viste sono "virtuali", e quindi non contengono dati, è anche possibile semplicemente eliminare la vista e ricrearla

Per eliminare una vista si usa lo

statement DROP VIEW, Specificando la clausola IF EXISTS la vista viene eliminata solo se esiste.

10.8 ACCESSO A DATABASE DA APPLICAZIONI



10.8.1 Problemi

- Impedance Mismatch
 - È il termine utilizzato per indicare l'insieme di difficoltà tecniche nell'accesso e nell'uso del database da parte delle applicazioni
- Diverse modalità di accesso
 - SQL è un linguaggio set-oriented (opera su insiemi di tuple), mentre le applicazioni operazionali agiscono su una singola tupla per volta (record-oriented)
- Diversi tipi di dati
 - ogni linguaggio di programmazione definisce i propri tipi di dati, che possono differire da quelli dell'SQL (e del DBMS)
 - Es: Java definisce un solo tipo per rappresentare le stringhe (`java.lang.String`), SQL ne definisce diversi: CHAR, VARCHAR, TEXT, LONGTEXT, ...
- Nel caso in cui si cerchi di mappare la struttura relazionale del DB con un modello ad oggetti vi sono anche altre difficoltà
 - Object-Relational Impedance Mismatch
- Incapsulamento
 - il mapping tra oggetti e tabelle espone necessariamente tutti i dati di un oggetto, violando l'incapsulamento
- Accesso
 - al contrario dei linguaggi OO, i DB non distinguono tra dati intrinsecamente pubblici o privati
- Ereditarietà e Polimorfismo
 - il modello relazionale non prevede ereditarietà, che viene spesso implementata tramite associazioni tra più tabelle
- Differenze strutturali
 - I linguaggi OO consentono di rappresentare la composizione di oggetti (tramite liste, collezioni, ...)
 - Nel modello relazionale è necessario rappresentare relazioni M-N con tabelle aggiuntive

10.8.2 Connettori ODBC e JDBC

I DBMS forniscono connettori (driver) specifici per alcuni linguaggi di programmazione. Bassa portabilità: ogni DBMS ha una propria API, quindi l'applicazione diventa dipendente dallo specifico DBMS

Nel 1992 Microsoft introduce ODBC (Open DataBase Connectivity)

- Definisce un API unificata di accesso ai database in C
- Ogni DBMS fornisce un proprio driver specifico che implementa l'API ODBC
- Unifica i tipi di dati
- Utilizzabile da qualsiasi linguaggio in grado di utilizzare librerie C (anche Java con JNI)

Java introduce JDBC (Java DataBase Connectivity)

- Equivalente a ODBC, ma fornisce un API specifica per Java (`java.sql.*`)
- Non richiede di interfacciarsi con librerie non interpretabili dalla JVM
- Ogni DBMS deve implementare il proprio driver JDBC

10.8.3 Connettori per MySQL

MySQL fornisce diversi connettori

- Connector/.NET: driver ADO.NET per framework Microsoft .NET
- Connector/ODBC: driver ODBC
- Connector/J: driver JDBC type 4 (pure Java)
- Connector/Python: driver nativo per Python
- Connector/C++: driver nativo per C++
- Connector/C: driver nativo per C

10.8.4 JDBC driver manager

Il DriverManager (java.sql.DriverManager) gestisce i driver JDBC disponibili e crea le connessioni ai DBMS

Per ottenere una connessione è necessaria una stringa di connessione, in forma di URN (Uniform Resource Name) e specifica per DBMS, che indica:

- Il tipo di driver
- Il tipo di DBMS
- L'host del server DBMS
- Le credenziali di accesso
- Il database a cui connettersi

11 ESEMPI DI ESAME

Si richiede di implementare con l'utilizzo di MySQL Workbench il database relativo al tema sotto riportato, secondo lo schema E-R dato.

- 1) Definire lo schema logico del database mediante l'EER Diagram;
- 2) Creare il database e le relative tabelle;
- 3) Popolare secondo quanto specificato di seguito;
- 4) Implementare ed eseguire almeno 3 delle 8 query sotto riportate.

TEMA DEL DATABASE

Il database "mondiale2009" contiene i dati relativi ai risultati del campionato mondiale Di formula 1 della stagione 2009.

La base di dati dovrà contenere alcune semplici informazioni sui piloti che hanno preso parte alle Gare del mondiale di formula 1, anno 2009 (codice pilota, cognome, nome e nazionalità). Inoltre per le squadre si dovrà specificare anche il nome della squadra, il motore montato sulle Autovetture (si suppone che tutte le macchine di ogni squadra montino lo stesso motore) e un codice Identificativo.

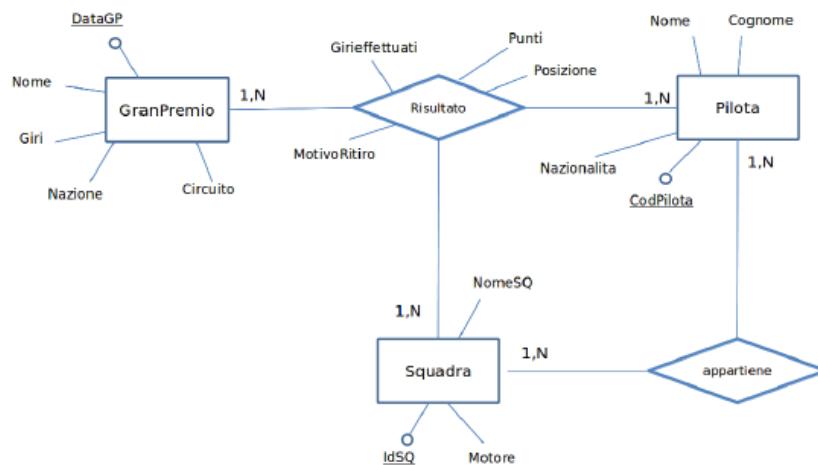
Dovranno essere registrate le informazioni relative ai singoli gran premi che sono stati disputati. Per ogni gran premio si dovrà tenere traccia, in particolare, della nazione in cui si è svolto, del Nome, del circuito, del numero di giri che i piloti devono compiere per arrivare al traguardo, e della Data in cui si svolge.

Sarà poi necessario tenere traccia dei risultati ottenuti dai piloti in ciascun gran premio disputato, in Particolare data di svolgimento della gara, posizione di arrivo, eventuale motivo del ritiro, giri Effettuati, punti guadagnati.

Per la relazione risultato si noti che:

- l'attributo relativo alla posizione di arrivo, deve essere nullo nel caso in cui il pilota non Abbia terminato la gara;
- l'attributo relativo ai giri compiuti dal pilota, è inferiore o uguale al numero di giri totali del Gran premio;
- l'attributo relativo al motivo del ritiro è nullo quando il pilota termina regolarmente la gara.

MODELLO E-R



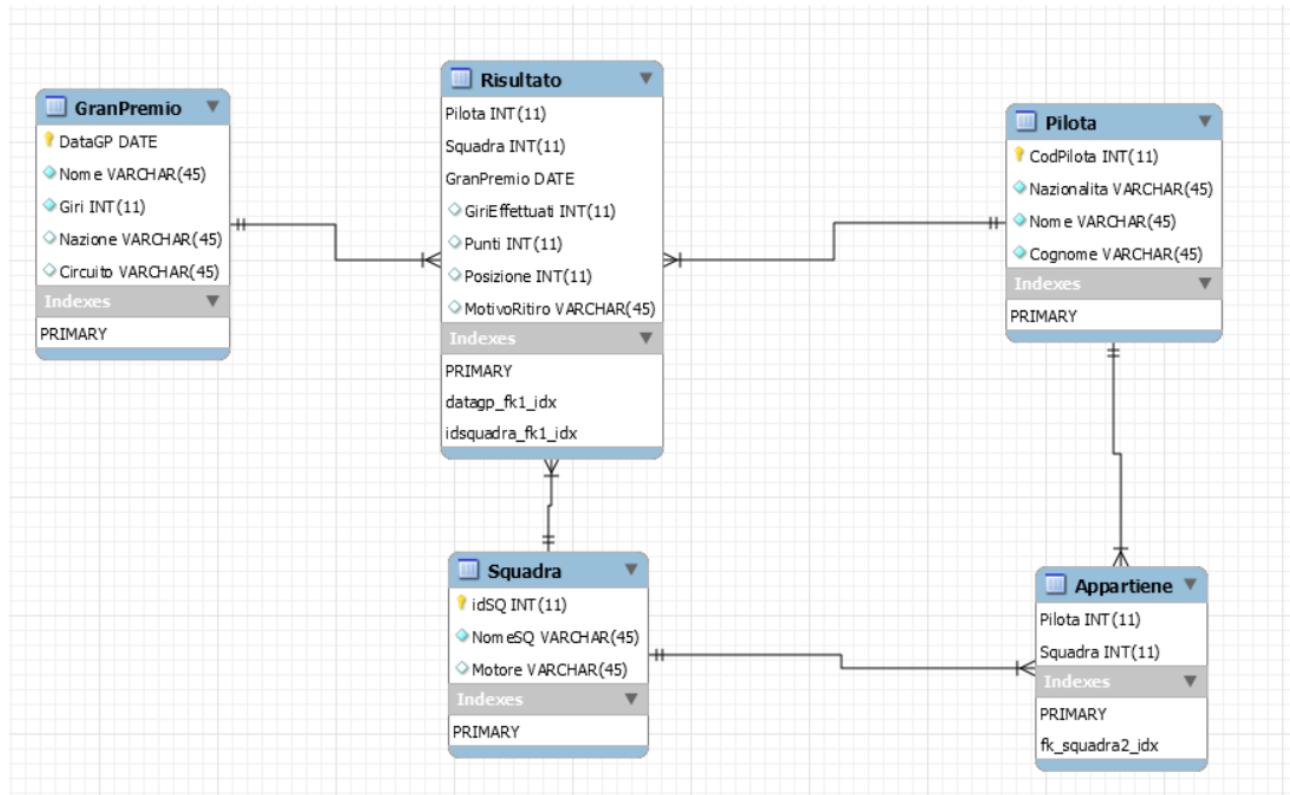
AVVERTENZE PER IL POPOLOAMENTO DEL DATABASE:

Per il popolamento del database potete utilizzare dati reali relativi al mondiale di f1 del 2009 Presenti in internet sui vari siti specializzati, per esempio:

Http://it.wikipedia.org/wiki/campionato_mondiale_di_formula_1_2009

In ogni caso, il popolamento del database dovrà tenere conto delle seguenti informazioni:

- una squadra può cambiare i propri piloti nel corso della stagione e, analogamente, un pilota Può correre per squadre diverse (in gare diverse, ovviamente).
- la tabella che si riferisce alle squadre dovrà avere almeno 5 record.
- ogni squadra concorre in ogni gran premio con due autovetture.
- in totale sono stati disputati almeno 6 gran premi.



11.1 CREAZIONE TABELLE

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;  
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;  
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';  
  
-----  
-- SCHEMA MYDB  
-----  
-----  
-- SCHEMA MONDIALE2009  
-----  
-----  
-- SCHEMA MONDIALE2009  
-----  
-----  
CREATE SCHEMA IF NOT EXISTS `MONDIALE2009` DEFAULT CHARACTER SET LATIN1 ;  
USE `MONDIALE2009` ;  
  
-----  
-- TABLE `MONDIALE2009`.`PILOTA`  
-----  
CREATE TABLE IF NOT EXISTS `MONDIALE2009`.`PILOTA` (  
`CODPILOTA` INT(11) NOT NULL,  
`NAZIONALITA` VARCHAR(45) NOT NULL,  
`NOME` VARCHAR(45) NOT NULL,  
`COGNOME` VARCHAR(45) NOT NULL,  
PRIMARY KEY (`CODPILOTA`))  
ENGINE = INNODB  
DEFAULT CHARACTER SET = LATIN1;
```

```
-- TABLE `MONDIALE2009`.`SQUADRA`
```

```
-----  
CREATE TABLE IF NOT EXISTS `MONDIALE2009`.`SQUADRA` (
```

```
  `IDSQ` INT(11) NOT NULL,  
  `NOMESQ` VARCHAR(45) NOT NULL,  
  `MOTORE` VARCHAR(45) NULL DEFAULT NULL,  
  PRIMARY KEY (`IDSQ`))
```

```
ENGINE = INNODB
```

```
DEFAULT CHARACTER SET = LATIN1;
```

```
-----  
-- TABLE `MONDIALE2009`.`APPARTIENE`
```

```
-----  
CREATE TABLE IF NOT EXISTS `MONDIALE2009`.`APPARTIENE` (
```

```
  `PILOTA` INT(11) NOT NULL,  
  `SQUADRA` INT(11) NOT NULL,  
  PRIMARY KEY (`PILOTA`, `SQUADRA`),  
  INDEX `FK_SQUADRA2_IDX` (`SQUADRA` ASC),  
  CONSTRAINT `FK_PILOTA2`  
    FOREIGN KEY (`PILOTA`)  
      REFERENCES `MONDIALE2009`.`PILOTA`(`CODPILOTA`)
```

```
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE,
```

```
  CONSTRAINT `FK_SQUADRA2`  
    FOREIGN KEY (`SQUADRA`)
```

```
    REFERENCES `MONDIALE2009`.`SQUADRA`(`IDSQ`)
```

```
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE)
```

```
ENGINE = INNODB
```

```
DEFAULT CHARACTER SET = LATIN1;
```

```
-- TABLE `MONDIALE2009`.`GRANPREMIO`
```

```
CREATE TABLE IF NOT EXISTS `MONDIALE2009`.`GRANPREMIO` (
```

```
  `DATAGP` DATE NOT NULL,  
  `NOME` VARCHAR(45) NOT NULL,  
  `GIRI` INT(11) NOT NULL,  
  `NAZIONE` VARCHAR(45) NULL DEFAULT NULL,  
  `CIRCUITO` VARCHAR(45) NULL DEFAULT NULL,  
  PRIMARY KEY (`DATAGP`))
```

```
ENGINE = INNODB
```

```
DEFAULT CHARACTER SET = LATIN1;
```

```
-- TABLE `MONDIALE2009`.`RISULTATO`
```

```
CREATE TABLE IF NOT EXISTS `MONDIALE2009`.`RISULTATO` (
```

```
  `PILOTA` INT(11) NOT NULL,  
  `SQUADRA` INT(11) NOT NULL,  
  `GRANPREMIO` DATE NOT NULL,  
  `GIRIEFFETTUATI` INT(11) NULL DEFAULT NULL,  
  `PUNTI` INT(11) NULL DEFAULT NULL,  
  `POSIZIONE` INT(11) NULL DEFAULT NULL,  
  `MOTIVORITIRO` VARCHAR(45) NULL DEFAULT NULL,  
  PRIMARY KEY (`PILOTA`, `SQUADRA`, `GRANPREMIO`),  
  INDEX `DATAGP_FK1_IDX` (`GRANPREMIO` ASC),  
  INDEX `IDSQUADRA_FK1_IDX` (`SQUADRA` ASC),  
  CONSTRAINT `COD_PILOTA_FK1`
```

```
FOREIGN KEY (`PILOTA`)
REFERENCES `MONDIALE2009`.`PILOTA`(`CODPILOTA`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `DATAGP_FK1`
FOREIGN KEY (`GRANPREMIO`)
REFERENCES `MONDIALE2009`.`GRANPREMIO`(`DATAGP`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `IDSQUADRA_FK1`
FOREIGN KEY (`SQUADRA`)
REFERENCES `MONDIALE2009`.`SQUADRA`(`IDSQ`)
ON DELETE CASCADE
ON UPDATE CASCADE)
ENGINE = INNODB
DEFAULT CHARACTER SET = LATIN1;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

11.2 POPOLAMENTO DATABASE

```
CREATE DATABASE IF NOT EXISTS `MONDIALE20092016` /*!40100 DEFAULT CHARACTER SET LATIN1 */;
```

```
USE `MONDIALE20092016`;
```

```
-- MYSQL DUMP 10.13 DISTRIBUT 5.6.19, FOR OSX10.7 (I386)
```

```
--
```

```
-- HOST: LOCALHOST  DATABASE: MONDIALE20092016
```

```
-- -----
```

```
-- SERVER VERSION      5.6.10
```

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
```

```
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
```

```
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
```

```
/*!40101 SET NAMES UTF8 */;
```

```
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
```

```
/*!40103 SET TIME_ZONE='+00:00' */;
```

```
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
```

```
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
```

```
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
```

```
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
```

```
--
```

```
-- TABLE STRUCTURE FOR TABLE `APPARTIENE`
```

```
--
```

```
DROP TABLE IF EXISTS `APPARTIENE`;
```

```
/*!40101 SET @SAVED_CS_CLIENT = @@CHARACTER_SET_CLIENT */;
```

```
/*!40101 SET CHARACTER_SET_CLIENT = UTF8 */;
```

```
CREATE TABLE `APPARTIENE` (
```

```
    `PILOTA` INT(11) NOT NULL,
```

```
    `SQUADRA` INT(11) NOT NULL,
```

```
    PRIMARY KEY (`PILOTA`,`SQUADRA`),
```

```
    KEY `FK_SQUADRA2_IDX` (`SQUADRA`),
```

```
CONSTRAINT `FK_PILOTA2` FOREIGN KEY (`PILOTA`) REFERENCES `PILOTA` (`CODPILOTA`) ON DELETE CASCADE ON UPDATE CASCADE,
```

```
CONSTRAINT `FK_SQUADRA2` FOREIGN KEY (`SQUADRA`) REFERENCES `SQUADRA` (`IDSQ`) ON DELETE CASCADE ON UPDATE CASCADE
```

```
) ENGINE=INNODB DEFAULT CHARSET=LATIN1;
```

```
/*!40101 SET CHARACTER_SET_CLIENT = @SAVED_CS_CLIENT */;
```

```
--
```

```
-- DUMPING DATA FOR TABLE `APPARTIENE`
```

```
--
```

```
LOCK TABLES `APPARTIENE` WRITE;
```

```
/*!40000 ALTER TABLE `APPARTIENE` DISABLE KEYS */;
```

```
INSERT INTO `APPARTIENE` VALUES
```

```
(4,1),(11,1),(5,2),(12,2),(13,3),(1,6),(3,6),(2,7),(7,7),(4,8),(14,8),(8,9),(10,10),(15,10),(16,10);
```

```
/*!40000 ALTER TABLE `APPARTIENE` ENABLE KEYS */;
```

```
UNLOCK TABLES;
```

```
--
```

```
-- TABLE STRUCTURE FOR TABLE `GRANPREMIO`
```

```
--
```

```
DROP TABLE IF EXISTS `GRANPREMIO`;
```

```
/*!40101 SET @SAVED_CS_CLIENT = @@CHARACTER_SET_CLIENT */;
```

```
/*!40101 SET CHARACTER_SET_CLIENT = UTF8 */;
```

```
CREATE TABLE `GRANPREMIO` (
```

```
 `DATAGP` DATE NOT NULL,
```

```
 `NOME` VARCHAR(45) NOT NULL,
```

```
 `GIRI` INT(11) NOT NULL,
```

```
 `NAZIONE` VARCHAR(45) DEFAULT NULL,
```

```
 `CIRCUITO` VARCHAR(45) DEFAULT NULL,
```

```
 PRIMARY KEY (`DATAGP`)
```

```
) ENGINE=INNODB DEFAULT CHARSET=LATIN1;  
/*!40101 SET CHARACTER_SET_CLIENT = @SAVED_CS_CLIENT */;  
  
--  
-- DUMPING DATA FOR TABLE `GRANPREMIO`  
  
--  
  
LOCK TABLES `GRANPREMIO` WRITE;  
/*!40000 ALTER TABLE `GRANPREMIO` DISABLE KEYS */;  
  
INSERT INTO `GRANPREMIO` VALUES ('2009-03-29','ING AUSTRALIAN GRAND  
PRIX',58,'AUSTRALIA','MELBOURNE GRAND PRIX CIRCUIT'),('2009-04-05','PETRONAS MALAYSIAN GRAND  
PRIX',56,'MALESIA','SEPANG INTERNATIONAL CIRCUIT'),('2009-04-19','CHINESE GRAND  
PRIX',56,'CINA','SHANGHAI INTERNATIONAL CIRCUIT'),('2009-04-26','GULF AIR BAHRAIN GRAND  
PRIX',57,'BAHARAIN','BAHRAIN INTERNATIONAL CIRCUIT'),('2009-05-10','GRAN PREMIO DE ESPAÑA  
TELEFÓNICA',66,'SPAGNA','CIRCUIT DE CATALUNYA'),('2009-05-24','GRAND PRIX DE  
MONACO',78,'MONACO','CIRCUIT DE MONACO'),('2009-09-13','GRAN PREMIO SANTANDER  
D\ITALIA',53,'ITALIA','AUTODROMO NAZIONALE DI MONZA');  
/*!40000 ALTER TABLE `GRANPREMIO` ENABLE KEYS */;  
UNLOCK TABLES;  
  
--  
-- TABLE STRUCTURE FOR TABLE `PILOTA`  
  
--  
  
DROP TABLE IF EXISTS `PILOTA`;  
/*!40101 SET @SAVED_CS_CLIENT = @@CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_CLIENT = UTF8 */;  
  
CREATE TABLE `PILOTA` (  
    `CODPILOTA` INT(11) NOT NULL,  
    `NAZIONALITA` VARCHAR(45) NOT NULL,  
    `NOME` VARCHAR(45) NOT NULL,  
    `COGNOME` VARCHAR(45) NOT NULL,  
    PRIMARY KEY (`CODPILOTA`)  
) ENGINE=INNODB DEFAULT CHARSET=LATIN1;
```

```
/*!40101 SET CHARACTER_SET_CLIENT = @SAVED_CS_CLIENT */;

--  
-- DUMPING DATA FOR TABLE `PILOTA`  
  
--  
  
LOCK TABLES `PILOTA` WRITE;  
  
/*!40000 ALTER TABLE `PILOTA` DISABLE KEYS */;  
  
INSERT INTO `PILOTA` VALUES  
(1,'ING','JENSON','BUTTON'),(2,'GER','SEBASTIAN','VETTEL'),(3,'BRA','RUBENS','BARRICELLO'),(4,'ITA','GIAN  
CARLO','FISICHELLA'),(5,'ING','LEWIS','HAMILTON'),(7,'AUS','MARK','WEBBER'),(8,'ITA','JARNO','TRULLI'),(9,'F  
IN','KIMI ','RÄIKKÖNEN'),(10,'SPA',' FERNANDO ','ALONSO'),(11,'BRA','FELIPE','MASSE'),(12,'FIN','HEIKKI  
' , 'KOVALAINEN'),(13,'POL','ROBERT ','KUBICA'),(14,'GER',' ADRIAN ','SUTIL'),(15,'FRA','ROMAIN  
' , 'GROSJEAN'),(16,'BRA','NELSON ','PIQUET ');  
  
/*!40000 ALTER TABLE `PILOTA` ENABLE KEYS */;  
  
UNLOCK TABLES;  
  
--  
-- TABLE STRUCTURE FOR TABLE `RISULTATO`  
  
--  
  
DROP TABLE IF EXISTS `RISULTATO`;  
  
/*!40101 SET @SAVED_CS_CLIENT = @@CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_CLIENT = UTF8 */;  
  
CREATE TABLE `RISULTATO` (  
    `PILOTA` INT(11) NOT NULL,  
    `SQUADRA` INT(11) NOT NULL,  
    `GRANPREMIO` DATE NOT NULL,  
    `GIRIEFFETTUATI` INT(11) DEFAULT NULL,  
    `PUNTI` INT(11) DEFAULT NULL,  
    `POSIZIONE` INT(11) DEFAULT NULL,  
    `MOTIVORITIRO` VARCHAR(45) DEFAULT NULL,  
    PRIMARY KEY (`PILOTA`,`SQUADRA`,`GRANPREMIO`),
```

```

KEY `DATAGP_FK1_IDX` (`GRANPREMIO`),
KEY `IDSQUADRA_FK1_IDX` (`SQUADRA`),
CONSTRAINT `COD_PILOTA_FK1` FOREIGN KEY (`PILOTA`) REFERENCES `PILOTA` (`CODPILOTA`) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `DATAGP_FK1` FOREIGN KEY (`GRANPREMIO`) REFERENCES `GRANPREMIO` (`DATAGP`) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `IDSQUADRA_FK1` FOREIGN KEY (`SQUADRA`) REFERENCES `SQUADRA` (`IDSQ`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB DEFAULT CHARSET=LATIN1;
/*!40101 SET CHARACTER_SET_CLIENT = @SAVED_CS_CLIENT */;

-- 
-- DUMPING DATA FOR TABLE `RISULTATO`

-- 

LOCK TABLES `RISULTATO` WRITE;
/*!40000 ALTER TABLE `RISULTATO` DISABLE KEYS */;
INSERT INTO `RISULTATO` VALUES (1,6,'2009-03-29',58,10,1,NULL),(1,6,'2009-04-05',56,10,1,NULL),(1,6,'2009-04-19',56,8,3,NULL),(1,6,'2009-04-26',57,10,1,NULL),(1,6,'2009-05-10',66,10,1,NULL),(1,6,'2009-05-24',78,10,1,NULL),(1,6,'2009-09-13',53,9,2,NULL),(2,7,'2009-04-19',56,10,1,NULL),(2,7,'2009-04-26',57,9,2,NULL),(2,7,'2009-05-24',5,NULL,NULL,'GUASTO MOTORE'),(3,6,'2009-03-29',58,9,2,NULL),(3,6,'2009-04-05',56,6,5,NULL),(3,6,'2009-04-26',57,6,5,NULL),(3,6,'2009-05-10',66,9,2,NULL),(3,6,'2009-05-24',78,9,2,NULL),(3,6,'2009-09-13',53,10,1,NULL),(5,2,'2009-04-26',57,7,4,NULL),(7,7,'2009-04-19',56,9,2,NULL),(7,7,'2009-05-10',66,8,3,NULL),(8,9,'2009-03-29',58,8,3,NULL),(8,9,'2009-04-05',56,7,4,NULL),(8,9,'2009-04-26',57,8,3,NULL),(8,9,'2009-05-10',32,NULL,NULL,'PENALTY'),(9,1,'2009-05-10',8,NULL,NULL,'GUASTO MOTORE'),(9,1,'2009-05-24',78,8,3,NULL),(9,1,'2009-09-13',53,8,3,NULL),(10,10,'2009-03-29',58,6,5,NULL),(10,10,'2009-09-13',53,6,5,NULL),(11,1,'2009-03-29',12,NULL,NULL,'GUASTO MOTORE'),(11,1,'2009-04-19',0,NULL,NULL,'SQUALIFICA'),(11,1,'2009-05-24',78,7,4,NULL),(13,3,'2009-04-05',11,NULL,NULL,'GUASTO MOTORE'),(13,3,'2009-05-10',66,1,11,NULL),(13,3,'2009-09-13',46,NULL,NULL,'SQUALIFICA');

/*!40000 ALTER TABLE `RISULTATO` ENABLE KEYS */;
UNLOCK TABLES;

-- 
-- TABLE STRUCTURE FOR TABLE `SQUADRA`

-- 

```

```

DROP TABLE IF EXISTS `SQUADRA`;

/*!40101 SET @SAVED_CS_CLIENT = @@CHARACTER_SET_CLIENT */;

/*!40101 SET CHARACTER_SET_CLIENT = UTF8 */;

CREATE TABLE `SQUADRA` (
    `IDSQ` INT(11) NOT NULL,
    `NOMESQ` VARCHAR(45) NOT NULL,
    `MOTORE` VARCHAR(45) DEFAULT NULL,
    PRIMARY KEY (`IDSQ`)
) ENGINE=INNODB DEFAULT CHARSET=LATIN1;

/*!40101 SET CHARACTER_SET_CLIENT = @SAVED_CS_CLIENT */;

-- 
-- DUMPING DATA FOR TABLE `SQUADRA`
-- 

LOCK TABLES `SQUADRA` WRITE;

/*!40000 ALTER TABLE `SQUADRA` DISABLE KEYS */;

INSERT INTO `SQUADRA` VALUES (1,'SCUDERIA FERRARI MARLBORO','FERRARI 056'),(2,'VODAFONE MCLAREN MERCEDES','MERCEDES FO 108W'),(3,'BMW SAUBER F1 TEAM','BMW P86/9'),(4,'SCUDERIA TORO ROSSO','FERRARI 056'),(5,'AT&T WILLIAMS F1 TEAM','TOYOTA RVX-09'),(6,'BRAWN GP F1 TEAM[44]','MERCEDES FO 108W'),(7,'RED BULL RACING','RENAULT RS27'),(8,'FORCE INDIA F1 TEAM','MERCEDES FO 108W'),(9,'PANASONIC TOYOTA RACING','TOYOTA RVX-09'),(10,'RENAULT F1 TEAM','RENAULT RS27');

/*!40000 ALTER TABLE `SQUADRA` ENABLE KEYS */;

UNLOCK TABLES;

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;

/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;

/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;

/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;

```

```
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;  
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

```
-- DUMP COMPLETED ON 2016-05-18 23:00:25
```

11.3 QUERY

- 1) Stilare una classifica dei piloti (composta da nome, nazionalità e totale dei punti), Ordinandola per punteggio decrescente;
- 2) Stilare una classifica delle squadre (composta da nome, motore e totale dei punti), Ordinandola per punteggio decrescente;
- 3) Relativamente ai piloti che hanno partecipato a meno di 4 gran premi, selezionare il nome Dei piloti e il numero di gran premi disputati;
- 4) Per ciascun pilota, selezionare il nome e il numero di gare in cui ha effettuato almeno il 50% Dei giri totali;
- 5) Per ciascun gran premio, indicarne il nome, la nazione, il circuito e il numero di piloti che si Sono ritirati;
- 6) Effettuare una classifica delle cause di ritiro (escludendo il valore null);
- 7) Individuare i piloti che hanno corso per due squadre diverse. Stampare il nome del pilota ed Il nome delle due squadre;
- 8) Individuare le squadre per cui hanno corso più di due piloti. Stampare il nome della squadra Ed il numero di piloti corrispondente.

USE MONDIALE20092016;

-- Q1

```
SELECT P.NOME, P.COGNOME, P.NAZIONALITA, SUM(R.PUNTI) PUNTI_TOTALI
FROM PILOTA P
JOIN RISULTATO R ON R.PILOTA=P.CODPILOTA
GROUP BY R.PILOTA
ORDER BY PUNTI_TOTALI DESC;
```

-- Q2

```
SELECT S.NOMESQ, S.MOTORE, SUM(R.PUNTI) PUNTI_TOTALI
FROM SQUADRA S
JOIN RISULTATO R ON R.SQUADRA=S.IDSQ
GROUP BY R.SQUADRA
ORDER BY PUNTI_TOTALI DESC;
```

-- Q3

```
SELECT P.NOME, P.COGNOME, COUNT(*) PARTECIPAZIONI
FROM RISULTATO R
```

```
JOIN PILOTA P ON R.PILOTA=P.CODPILOTA
```

```
GROUP BY R.PILOTA
```

```
HAVING PARTECIPAZIONI<4;
```

-- Q4

```
SELECT P.NOME, P.COGNOME,COUNT(*)
```

```
FROM RISULTATO R
```

```
JOIN PILOTA P ON R.PILOTA=P.CODPILOTA
```

```
JOIN GRANPREMIO G ON G.DATAGP = R.GRANPREMIO
```

```
WHERE R.GIRIEFFETTUATI>0.5*G.GIRI
```

```
GROUP BY R.PILOTA;
```

-- Q5

```
SELECT G.NOME,G.NAZIONE, G.CIRCUITO,COUNT(*)
```

```
FROM GRANPREMIO G
```

```
JOIN RISULTATO R ON G.DATAGP = R.GRANPREMIO
```

```
WHERE R.POSIZIONE IS NULL
```

```
GROUP BY G.DATAGP;
```

-- Q6

```
SELECT R.MOTIVORITIRO, COUNT(*) NUMERO_RITIRI
```

```
FROM RISULTATO R
```

```
WHERE R.MOTIVORITIRO IS NOT NULL
```

```
GROUP BY R.MOTIVORITIRO
```

```
ORDER BY NUMERO_RITIRI DESC;
```

-- Q7

```
SELECT DISTINCT P.NOME,P.COGNOME,S1.NOMESQ,S2.NOMESQ
```

```
FROM PILOTA P
```

```
JOIN APPARTIENE R1 ON R1.PILOTA=P.CODPILOTA
```

```
JOIN APPARTIENE R2 ON R2.PILOTA=P.CODPILOTA
```

```
JOIN SQUADRA S1 ON S1.IDSQ=R1.SQUADRA  
JOIN SQUADRA S2 ON S2.IDSQ=R2.SQUADRA  
WHERE R1.SQUADRA > R2.SQUADRA;
```

-- Q8

```
SELECT S.NOMESQ,COUNT(*) NUMERO_PILOTI  
FROM APPARTIENE A  
JOIN SQUADRA S ON S.IDSQ=A.SQUADRA  
GROUP BY A.SQUADRA  
HAVING NUMERO_PILOTI > 2;
```