BASI DI DATI

Laurea triennale in scienze informatiche

Andrea Broccoletti

Università degli studi di Milano Bicocca

A.A. 2022/2023

Indice

Capitolo 1

Introduzione

1.1 Definizioni di base

1.1.1 Base di dati

È l'insieme organizzato di dati utilizzati per il supporto allo svoglimento di attività.

1.1.2 Informatica

scienza del **trattamento razionale**, specialmente per mezzo di macchine automatiche, dell'informazione, considerata come supporto della conoscenza umana e della comunicazione.
L'informatica ha due anime:

Metodologica

I metodi per la soluzione di problemi e la gestione delle informazioni.

Tecnologica

I calcolatori elettronici e i sistemi che utilizzano.

1.1.3 Sistema informativo

Componente, ovvero un sottosistema, di una organizzazione che gestisce le informazioni di interesse, cioè utilizzati per il perseguimento degli scopi dell'organizzazione.

Il concetto di sistema informativo è indipendente da qualsiasi automatizzazione, anche se molti sistemi informativi si sono evoluti verso una razionalizzazione e standardizzazione delle procedure e della gestione delle informazioni.

1.1.4 Sistema informatico

Porzione automatizzata del sistema informativo, che gestisce informazioni con tecnologia informatica.

Sono suoi i compiti di:

- garantire che i dati siano conservati in modo permanente sui dispositivi di memorizzazione
- permettere un rapido aggiornamento dei dati per riflettere rapidamente le loro variazioni
- rendere i dati accessibili alle interrogazioni degli utenti
- essere distribuito sul territorio

1.1.5 Gestione delle informazioni

Nei sistemi informatici, le informazioni vengono rappresentate in modo essenziale attraverso i dati.

Informazione

elemento che consente di avere conoscenza più o meno esatta di dati, situazioni o modi di essere.

Dati

Ciò che è immediatamente presente alla conoscenza, prima di ogni elaborazione. Sono elementi di informazione costituiti da simboliche debbono essere elaborati.
Vengono usati perchè sono una rappresentazione più precisa e costituiscono spesso una risorsa strategica perchè sono stabili nel tempo rispetto ad altri componenti.

1.1.6 Data Base

Collezione di dati utilizzati per rappresentare le informazioni di interesse di un sistema informativo.
Ci sono caratteristiche fondamentali di un database:

- la ridondanza è minima e controllata, assicurando la consistenza delle informazioni
- i dati sono disponibili per utenze diverse e concorrenti
- i dati sono controllati e indipendenti

1.1.7 Data Base Management System

Un DBMS è un sistema software capace di gestire collezioni di dati che siano grandi, condivise e persistenti, assicurando la loro affidabilità e privatezza.

Nella pratica, è un insieme di programmi che permettono di gestire le basi di dati, facilitando il processo di definizione, costruzione e manipolazione del database per varie applicazioni.

Deve garantire privatezza, affidabilità, efficienza ed efficacia attraverso le operazioni consentite.

Vantaggi

- permettono di considerare i dati come risorsa comune, a disposizione di molteplici applicazioni e utenti
- offrono un modello della parte di mondo di interesse che è unificato e preciso
- offrono un controllo centralizzato dei dati, riducendo ridondanze e inconsistenze

Svantaggi

- sono costosi e complessi, con specifici requisiti hardware e software
- difficile filtrare tra tutti i servizi offerti da un DBMS, quelli effettivamente utili
- sono inadatti alle gestione di applicazioni con pochi utenti, in relazione al loro costo

1.1.8 Transazione

Insieme di operazioni da considerare indivisibili, atomiche. La sequenza di operazioni sulla base di dati viene eseguita per intero o per niente.
Sono spesso realizzate con programmi in linguaggio ospite

1.1.9 Organizzazione dei dati

In ogni base di dati esistono:

Schemi

Sostanzialmente invariati nel tempo, descrivono la struttura e l'aspetto estensionale, cioè il significato.

Istanze

I valori attuali, che possono cambiare anche molto rapidamente.

Esempio

Esaminando una tabella di dati, le intestazioni rappresentano lo schema, mentre le righe (tuple) le istanze:

ID	NOME
886155	Pippo
886261	Pluto
887050	Paperino

1.1.10 Modello dei dati

Insieme di costrutti utilizzati per organizzare i dati di interesse e descriverne la dinamica. Come nei linguaggi di programmazione esistono meccanismi che permettono di definire nuovi tipi, così ogni modello dei dati prevede alcuni costruttori.
Ci sono tre tipi principali di modelli.

Logici

Sono indipendenti dalle strutture fisiche e definiscono l'organizzazione dei dati: relazione, a oggetti, gerarchico...

Relazionali

I dati vengono strutturati in tabelle, in particolare un DBMS relazionale può essere pensato come un insieme di tabelle che mantengono informazioni di tipo omogeneo.
Diverse tabelle sono in relazione fra loro grazie alla presenza di un campo comune, che permette di mettere in relazione i dati delle due tabelle.

Concettuali

Permettono di rappresentare i dati in modo indipendente da ogni sistema, cercando di descrivere i concetti del mondo reale e sono utili nelle fasi iniziali di progettazione.
Il più diffuso è il **modello entity-relationship**.

1.1.11 Schemi

Logico

Descrizione della base di dati nel modello logico, come la struttura della tabella.

Interno o Fisico

È la rappresentazione dello schema logico per mezzo di strutture di memorizzazione.

Esterno

Descrizione di parte della base di dati in un modello logico. Non sono altro che viste parziali

Esempio

Considerando una tabella che colleziona i dati degli studenti di un'università, la "vista segreteria" permette di visualizzare tutti i dati relativi agli studenti, mentre la "vista docenti" permette di visualizzare solo nome, cognome e matricola.

1.2 Indipendenza dei dati

Il livello logico è indipendente da quello fisico, naturale conseguenza dell'articolazione a livelli.

1.2.1 Indipendenza fisica

Il livello logico e quello esterno sono indipendenti da quello fisico, in particolare una relazione è utilizzata nello stesso modo qualunque sia la realizzazione fisica, che può cambiare senza che debbano essere modificati i programmi che le utilizzano.

1.2.2 Indipendenza logica

Il livello esterno è indipendente da quello logico. Aggiunte o modifiche alle viste non richiedono modifiche al livello logico.

1.3 Linguaggi per basi di dati

Esistono due tipi di linguaggi, che agiscono a diversi livelli e distinguono l'implementazione dall'utilizzo:

- **Data Definition Language** Il DDL definisce gli schemi logici, fisici e delle autorizzazioni di accesso
- **Data Manipulation Language** Il DML permette di interrogare e manipolare la base di dati

1.4 Personaggi e interpreti

1.4.1 Utenti

Si distinguono in **finali** che eseguono operazioni predefinite, le transazioni, e **casuali**, che invece eseguono operazioni non previste a priori usando linguaggi interattivi.

1.4.2 Database Administrator

Persona o gruppo di persone responsabile del controllo centralizzato e della gestione del sistema, delle prestazioni, dell'affidabilità e delle autorizzazioni. Le sue funzioni includono anche quelle di progettazione, anche se in progetti complessi ci possono essere distinzioni

Capitolo 2

Progettazione di basi di dati

2.1 Introduzione

In questa capitolo verrà illustrato ed esemplificato il processo di progettazione concettuale e logica delle basi di dati relazionali, che permette, partendo dai requisiti di utente, di arrivare a produrre strutture di basi di dati di buona qualità.

La **progettazione di basi di dati** è una delle attività del processo di sviluppo dei sistemi informativi, va quindi inquadrata in un contesto più generale, **il ciclo di vita dei sistemi informativi**.

2.2 Ciclo di vita dei sistemi informativi

Insieme e sequenzializzazione delle attività svolte da analisti, progettisti, utenti, nello sviluppo e nell'uso dei sistemi informativi. Essendo un attività iterativa, prevede un ciclo e un susseguirsi di fasi:

- **studio di fattibilità** definzione costi e priorità
- **raccolta e analisi dei requisiti** studio delle proprietà del sistema
- **progettazione** di dati e funzioni
- **validazione e collaudo** una sperimentazione del sistema completo
- **funzionamento** il sistema diventa operativo

2.3 Progettazione

La progettazione si individua in due categorie:

- **dei dati** individua l'organizzazione e la struttura della base di dati
- **delle applicazioni** schematizza le operazioni sui dati e progetta il software applicativo

Per garantire prodotti di buona qualità è opportuno seguire una **metodologia di progetto**, ovvero un'articolazione in fasi/passi di guida ad una attività di progettazione.

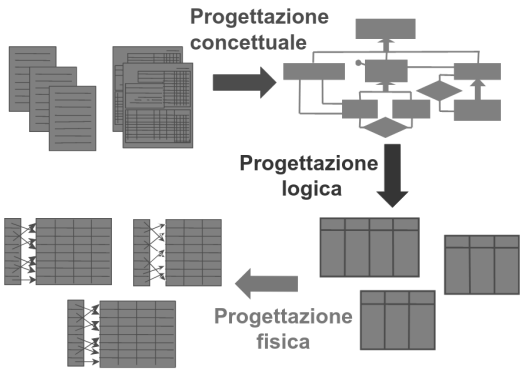
2.3.1 Metodologia di progettazione

Una metodologia permette di **suddividere** la progettazione in fasi successive e indipendenti, fornendo **strategie** da seguire e criteri di scelta in caso di alternative. Deve anche fornire un **modello di riferimento** per descrivere la realtà che stiamo progettando, ovvero dei **linguaggi**. Deve garantire:

- **generalità** rispetto ai problemi da affrontare
- **qualità** in termini di correttezza, completezza ed efficienza
- **facilità d'uso**

Si deve basare su un principio semplice ma efficace: separare nettamente le decisioni di **cosa rappresentare** e **come farlo**. Nella pratica si tratta di prevedere 3 fasi di progettazione: concettuale, logica e fisica.

Figura 2.1: Fasi di progettazione



Ognuna delle fasi si basa su un **modello**, che permette di generare una rappresentazione formale, o **schema**, della base di dati ad un dato livello di astrazione.

Progettazione concettuale

Questa fase traduce i requisiti del sistema informativo in una descrizione **formalizzata** e **integrata** delle esigenze aziendali, espressa in modo **indipendente** dalle scelte implementative. Più nello specifico, è:

- **formale** la descrizione deve essere espressa con un linguaggio non ambiguo e capace di descrivere in modo soddisfacente il sistema analizzato
- **integrata** la descrizione deve essere in grado di descrivere nella globalità l'ambiente analizzato
- **indipendente dall'ambiente tecnologico** la descrizione deve concentrarsi sui dati e sulle loro relazioni, e non sulle scelte implementative.

Questa progettazione permette una descrizione dei dati indipendente dagli aspetti tecnologici con un livello di astrazione intermedio fra utente e sistema, facendo prevalere l'aspetto **intensionale**, utile quindi per la **documentazione**.

Progettazione logica

Questa fase consiste nella traduzione dello schema concettuale nel modello dei dati del DBMS. Ne risulta uno schema logico, espresso nel DDL del DBMS.
In questa fase si considerano anche aspetti legati ai vincoli ed all'efficienza, articolandosi in due sottofasi:

- ristrutturazione dello schema concettuale
- traduzione verso il modello logico

Progettazione fisica

È la fase che completa lo schema logico ottenuto con le specifiche proprie dell'hardware e software. Il risultato è lo **schema fisico** che descrivere le strutture di memorizzazione e di accesso ai dati.

Nel nostro utilizzo, lo schema concettuale è il **modello E-R**, mentre lo schema logico è il **modello relazionale**.

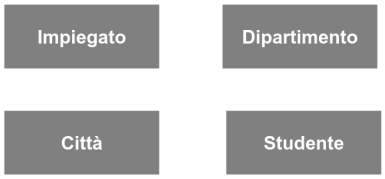
2.4 Modello Entità-Relazione

Il modello ER è un linguaggio grafico semi-formale per la rappresentazione di schemi concettuali.
Il modello si è ormai affermato come uno standard nelle metodologie di progetto e nei sistemi software di ausilio alla progettazione. Ne esistono di molte versioni, può o meno diverse.
Il modello utilizza diversi costrutti, di seguito elencati e esemplificati.

2.4.1 Entità

Una entità è una classe di oggetti dell'applicazione di interesse con proprietà comuni e con esistenza **autonoma** e della quale si vogliono registrare fatti specifici.

Figura 2.2: Rappresentazione grafica di entità



Convenzioni

Ogni entità ha un **nome** che la identifica univocamente nello schema, sono nomi espressivi, convenzionalmente al **singolare**.

A livello estensionale, un'entità è costituita da un insieme di oggetti, che sono chiamati le sue **istanze**: non è un valore che identifica un oggetto, ma è l'oggetto stesso.

Istanza

L'occorrenza o l'instanza di entità è un oggetto della classe che l'entità rappresenta, una **conoscenza concreta**. Nello schema rappresentiamo le entità, la **conoscenza astratta**.

2.4.2 Attributi

Un attributo di entità è una proprietà locale di un'entità, di interesse ai fini dell'applicazione.
Associa quindi ad ogni istanza di entità un valore appartenente ad un insieme detto **dominio** dell'attributo: si definisce un attributo per l'entità E quando si vuole rappresentare una proprietà **locale** delle istanza di E.

Una proprietà è detta quindi **locale** quando in ogni istanza dello schema il valore di tale proprietà dipende solamente dall'oggetto stesso, e **non ha alcun rapporto** con altri elementi dell'istanza dello schema.

Anche gli attributi hanno un nome che li identificano in modo univoco nell'ambito della entità.

Figura 2.3: Rappresentazione grafica degli attributi

Dominio

Un attributo è definito su un dominio di valori ed, in particolare, associa ad ogni istanza di entità o associazione un valore nel corrispondente dominio. Solitamente, i domini non sono indicati, ma si può indicare il **tipo** dell'attributo.

Esempio

Se esiste un attributo Cognome, che possiamo ipotizzare essere una stringa, possiamo indicare l'attributo con l'etichetta *Cognome*/*stringa*.

Smart Trick

Un concetto verrà modellato come una entità se:

- se le sue istanze sono concettualmente significative indipendentemente da altre istanze
- se ha o potrà avere delle proprietà indipendenti dagli altri concetti
- se il concetto è importante nell'applicazione

Mentre come attributo se:

- se le sue istanze non sono significative
- se non ha senso considerare una sua istanza indipendente
- se serve solo a rappresentare una proprietà locale

Composti

Gli attributi composti si ottengono raggruppando attributi di una medesima entità o relazione che presentano affinità nel loro significato d'uso.

Esempio

Un attributo *Indirizzo* può a sua volta essere composto dagli attributi *Via*, *Numero*, *CAP*.

2.4.3 Relazione

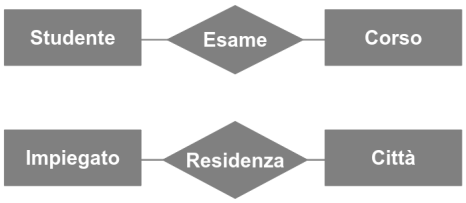
Detta anche **associazione**, è un fatto che descrive un'azione o una situazione che **stabilisce legami logici** tra istanze di entità nella realtà che stiamo considerando. I legami possono essere fra più di due entità, e il numero di entità coinvolte ne determina il **grado**.

Ogni relazione ha un nome che la identifica univocamente nello schema. Valgono le convenzioni di utilizzo del singolare e sostantivi invece che verbi.

Istanza di associazione

È una combinazione, o aggregazione, di entità che prendono parte all'associazione. Dalla semantica delle relazioni segue immediatamente che non

Figura 2.4: Rappresentazione grafica di associazione



possono esistere due istanze della stessa relazione che coinvolgono le stesse istanze di entità.

Attributo di relazione

Un attributo di relazione è una proprietà locale di una relazione, che associa ad ogni istanza di relazione un valore appartenente ad un dominio.
Ha un nome che lo identifica in modo univoco nell'ambito della relazione ed è rappresentato come un normale attributo.

Ricorsiva o ad anello

È un'associazione che coinvolge due o più volte la stessa entità.
In queste relazioni, è necessario aggiungere la specifica degli **ruoli** successore e predecessore.

2.5 Altri costrutti ER

In questa sezione si presentano altri costrutti utilizzabili nei modelli ER.

2.5.1 Cardinalità delle relazioni

È una coppia di valori che si associa a ogni entità che partecipa ad una relazione. Questi valori specificano il **numero minimo** e **massimo** di occorrenze delle relazioni cui ciascuna occorrenza di un' entità può partecipare.

Per semplicità la notazione fa uso di tre simboli:

- **0** partecipazione opzionale
- **1** partecipazione obbligatoria e massima
- **N** non pone alcun limite

Classificazione

Con riferimento alle cardinalità massime, si possono avere relazione classificate come:

- **uno a uno** se le cardinalità massime di entrambe le entità sono uno
- **uno a molti**
- **molti a molti**

Vincoli di cardinalità

Un vincolo tra un'entità e una relazione esprime un **limite minimo** ed un **limite massimo** di istanze della relazione a cui può partecipare ogni istanza dell'entità.

Smart Trick

È possibile associare delle cardinalità anche agli attributi, con lo scopo di indicare opzionalità e attributi multivalore.

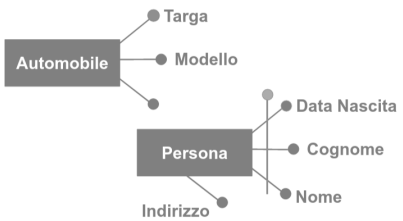
2.5.2 Identificatore di entità

È uno strumento per l'identificazione univoca delle occorrenze di un'entità. È costituito da attributi di entità, ovvero un **identificatore interno**, e da entità esterne attraverso relationship, con un **identificatore esterno**.

Notazione

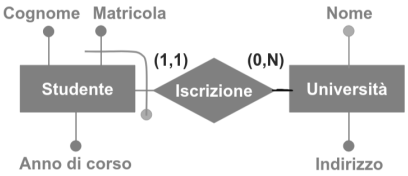
Gli identificatori interni, se è formati da un solo attributo, si annerisce il corrispondente pallino, sese invece sono formati da più attributi, si uniscono gli attributi con una linea che termina con pallino annerito.

Figura 2.5: Identificatore interno



Mentre gli identificatori esterni, se formati da attributi e relazioni, si indica unendo gli attributi ed i ruoli con una linea che termina con pallino annerito.

Figura 2.6: Identificatore esterno



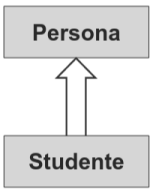
Osservazioni

- ogni entità deve possedere almeno un identificatore
- una identificazione esterna è possibile solo attraverso una relationship a cui l'entitò da identificare partecipa con cardinalità *(1,1)*.

2.5.3 Relazione IS-A

Due entità possono avere istanze in comune, ovvero può sussistere una relazione IS-A, o **di sottoinsieme**, cioè ogni istanza di una è anche istanza dell'altra.
Questa relazione si può definire tra due entità, rispettivamente **padre** e **figlia** o **sottoentità**.
Si rappresenta mediante una freccia dalla sottoentità alla entità padre.

Figura 2.7: Rappresentazione grafica della relazione IS-A



Ereditarietà

Ogni proprietà dell'entità padre è anche una proprietà della sottoentità e non si riporta esplicitamente nel diagramma. L'entità figlia può avere ovviamente ulteriori proprietà.

2.5.4 Generalizzazione

Finora, abbiamo considerato la relazione IS-A che stabilisce che l'entità padre è più generale della sottoentità. Talvolta, però, l'entità padre può **generalizzare** diverse sottoentità rispetto ad un unico criterio. In questo caso si parla di generalizzazione.
Nella generalizzazione, le sottoentità hanno **insiemi di istanze disgiunti a coppie** (anche se in alcune varianti del modello ER, si può specificare se due sottoentità della stessa entità padre sono disgiunte o no).

Una generalizzazione è detta anche **completa** se l'unione delle istanze delle sottoentità è uguale all'insieme delle istanze dell'entità padre. Altrimenti, è **non completa**.

La generalizzazione si indica collegando mediante un arco le sottoentità, e collegando con una freccia tale arco alla entità padre. La freccia è **annerita** se la generalizzazione è completa.

Vige la regola che una entità può avere al massimo una entità padre, ovvero il modello ER **non ammette** ereditarietà **multipla**.

Ereditarietà

Il principio di ereditarietà vale anche per le generalizzazioni: ogni proprietà dell'entità padre è anche una proprietà della sottoentità, e non si riporta esplicitamente nel diagramma. L'entità figlia può avere ovviamente ulteriori proprietà.

Smart Trick

Si usa una **generalizzazione** se le due sottoclassi derivano da uno stesso criterio di classificazione delle istanze della superclasse; mentre si usa la **relazione IS-A** se le due sottoentità sono indipendenti, nel senso che il loro significato non deriva dallo stesso criterio di classificazione delle istanze dell'entità padre.

2.5.5 Altre proprietà

Esistono altre proprietà secondarie, che non verranno trattate nello specifico:

- possono esistere gerarchie a più livelli e multiple generalizzazioni allo stesso livello
- un'entità può essere incluse in più gerarchie, come genitore e/o come figlia
- se una generalizzazione ha solo un'entità figlia si parla di **sottoinsieme**.

2.6 Vincoli non esprimibili nel diagramma ER

Fino ad ora si sono trattati gli schemi ER nella loro capacità di cogliere la maggior parte delle interazioni tra i dati del dominio d'interesse.
Tuttavia alcune inter-relazioni non possono essere colte direttamente: tali inter-relazioni vanno in ogni caso tenute presenti attraverso delle asserzioni aggiuntive dette **vincoli esterni al diagramma**, o semplicmente **vincoli esterni**.

Rappresentazione

Vengono rappresentati attraverso formalismi opportuni, in **logica matematica**, oppure come asserzioni in **linguaggio naturale**, che devono essere il più possibile precise e non ambigue.

Dizionario dei dati

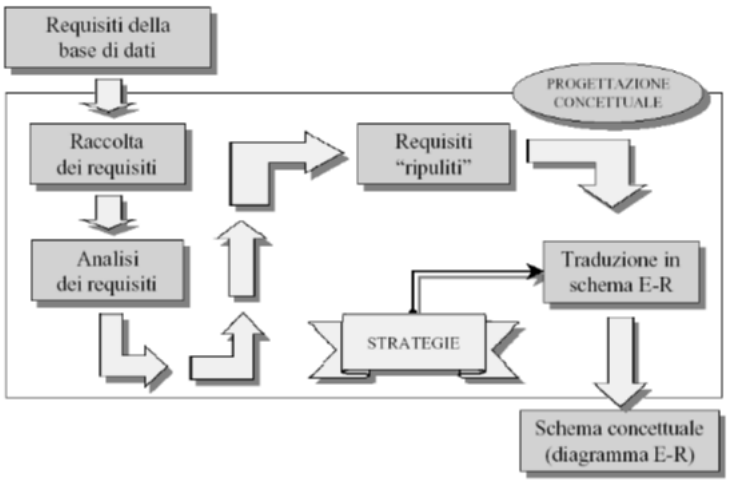
Oltre al diagramma ER, lo schema concettuale è descritto dal dizionario, che è costituito dalle tabelle di entità, relazioni, attributi e vincoli esterni e dalle loro proprietà.

Capitolo 3

Progettazione Concettuale

La progettazione concettuale prevede diverse fasi successive, che portano come fine ultimo alla definizione di uno **schema concettuale**.
Alla base di tutto ci sono i **requisiti** della base di dati.

Figura 3.1: Fasi di progettazione concettuale



3.1 Analisi dei requisiti

L'analisi dei requisiti comprende attività di acquisizione dei requisiti, successiva analisi e costruzione dello schema concettuale, con relativa costruzione del glossario.

I requisiti vengono dedotti da diverse **fonti**:

- **utenti** attraverso interviste e documentazione apposita
- **documentazione esistente** come le diverse normative e regolamenti del settore, oltre che a regolamenti interni, procedure aziendali e realizzazioni pre-esistenti

Il reperimento dei requisiti è un'attività difficile e non standardizzabile, a cui fa seguito la fase di analisi che spesso indirizza verso altre fonti di acquisizione.

3.1.1 Documentazione descrittiva

La raccolta dei requisiti porta alla generazione di una documentazinoe associata, che segue regole generali per la sua stesura:

- scegliere il corretto livello di astrazione
- standardizzare la struttura delle fasi
- suddividere le frasi articolate
- separare le frasi sui dati da quelle sulle funzioni
- costruire un glossario dei termini
- individuare omonimi e sinonimi e unificare i termini
- riorganizzare le frasi per concetti

3.2 Design pattern

Sono soluzioni progettuali a problemi comuni, largamente usati nell'ingegneria del software. Alcuni pattern sono comuni e degni di spiegazione.
Sono nella pratica concetti astratti, basati su un processo di **reificazione**: il procedimento di creazione di un modello di dati basato su un concetto astratto predefinito.

Reificazione di attributo di entità

L'entità può essere estratta da un'altra entità in quanto rappresenta un'entità separata.

Part-of

A volte un'entità può essere legata a un'altra entità creando una relazione di tipo *(1, N)*. Il concetto di part of rappresenta un'entità che fa parte di un'altra entità, relazione che può essere di **dipendenza** o meno

Esempio

Un cinema è composto da molte sale: le sale non esisterebbero senza un cinema.
Allo stesso modo un team è composto da diverse persone, ma ogni persone è autonoma, può esistere anche senza l'esistenza di un team.

Instance-of

A volte sovviene la necessità di creare un'entità astratta che prende concretezza in un'entità istanza.

Esempio

Un'entità *Volo* possiede informazioni astratte sul *volo di linea* che invece rappresenta il volo che avviene giornalmente.

3.3 Strategie di progetto

Ciascuna strategia prevede opportune **primitive di raffinamento** che specificano in che modo sostituire o integrare una parte dello schema con una versione più raffinata della stessa.

3.3.1 Tipologie di strategia

Top-Down

Si parte da uno schema iniziale molto astratto ma completo, che viene successivamente raffinato fino ad arrivare allo schema finale.
Non è inizialmente necessario specificare i dettagli, ma richiede sin dall'inizio una visione globale del problema non sempre ottenibile in casi complessi.

Bottom-Up

Si suddividono le specifiche in modo da sviluppare semplici schemi parziali ma dettagliati, che poi vengono integrati tra loro.

Permette una ripartizione delle attività, ma richiede una successiva fase di integrazione.

Inside-Out

Lo schema si sviluppa dall'interno verso l'esterno, partendo dai concetti più importanti, aggiungendo quelli ad essi correlati.
Non richiede passi di integrazione, ma richiede invece, ad ogni passo, di esaminare tutte le specifiche per trovare i concetti non ancora esaminati.

3.3.2 Strategia ibrida

Combina i vantaggi delle strategie top-down e bottom-up, ovvero la suddivisione dei requisiti in **componenti separate** e la definizione di uno **schema scheletro** per i concetti principali.

È la strategia più flessibile perchè permette di suddividere il problema in sottoproblemi e di procedere per raffinamenti progressivi.

Schema scheletro

È uno schema che facilità le fasi di integrazione:

- si individuano i concetti più importanti, i più citati o quelli indicati come cruciali
- si organizzano tali concetti in un semplice schema concettuale
- ci si concentra sugli aspetti essenziali

3.4 Schema concettuale

Per arrivare ad uno schema concettuale, considerando tutti gli aspetti precedentemente esplicitati, bisogna passare attraverso quattro fasi:

- **analisi dei requisiti** in cui si analizzano i requisiti e si eliminano le ambiguità, costruendo un glossario dei termini e raggruppando i requisiti in insiemi omogenei
- **passo base** definire uno schema scheletro con i concetti più rilevanti
- **passo iterativo** da ripetere fino alla soddisfazione, in questo passo si raffinano i concetti presenti sulla base delle loro specifiche, aggiungendo concetti per descrivere più nello specifico.

· **analisi di qualità** ripetuta e distribuita, verifica le qualità dello schema e lo modifica per definirlo al meglio della qualità

3.4.1 Qualità di uno schema concettuale

Correttezza

Uso corretto dei costrutti, sintattici e semantici.

Completezza

Tutti i dati devono essere rappresentati, e tutte le operazioni possono essere eseguite. In particolare, tutti i dati previsti da un'operazione sono raggiungibili **navigando** il diagramma ER.

Leggibilità

Lo schema deve essere il più possibile autoesplicativo, anche in termini di nomenclatura e layout dello schema.

Minimalità

Lo schema non contiene ridondanze a livello intensionale ed estensionale, ovvero concetti ripetuti o che possono essere derivati da altri, senza la necessità di averli come entità, relazioni o attributi veri e propri.

Capitolo 4

Modello Relazionale

4.1 Modello dei dati

Un modello dei dati è un insieme di **concetti** per organizzare i dati e descriverne la struttura.
Il modello deve essere comprensibile da un elaboratore e deve prevedere alcuni costruttori che permettano di definire nuovi tipi sulla base di tipi predefiniti, detti **elementari**. il **modello relazionale** è quello più diffuso.

4.1.1 Modello Relazionale

Questo modello permette di definire tipi per mezzo del costruttore **relazione** che permette di organizzare i dati in insiemi di record a **struttura fissa**. Una relazione è spesso rappresentata da una tabella in cui:

- **le righe** rappresentano specifici record
- **le colonne** corrispondono ai campi dei record

L'ordine di righe e colonne è sostanzialmente irrilevante.

4.2 Tabelle e relazioni

Una tabella rappresenta una relazione se:

- i valori di ogni colonna sono fra loro omogenei
- le righe sono diverse fra loro
- le intestazioni delle colonne sono diverse tra loro

4.3 Chiave

Insieme di attributi che identificano univocamente le tuple di una relazione. Dal punto di vista formale, un insieme di attributi è detto **superchiave** se due tuple distinte non hanno mai la stessa chiave. È una **chiave** se è l'unica superchiave presente.
Infine, è detta **minimale** se contiene un solo attributo.

4.3.1 Chiave primaria

Chiave su cui non sono ammissibili valori nulli. Viene notificata con una sottolineatura.

4.4 Informazione incompleta

Il modello relazionale impone ai dati una struttura rigida, ma i dati disponibili possono non corrispondere al formato previsto. In questi casi, si adotta il **valore nullo** che denota l'assenza di un valore del dominio. Si rappresenta con NULL.
Un DBMS non sa distinguere i vari casi di valore nullo, quindi bisogna imporre restrizioni sulla presenza di questi.

4.5 Vincoli di integrità

Risulta evidente che solo alcune configurazioni di valori nulli sono ammissibili.
Esistono vincoli di integrità che possono essere espressi per ognuno de domini della relazione e non solo relativamente ai valori nulli.

4.5.1 Tipi di integrità

Non tutte le tuple rappresentano informazione corretta per un'applicazione:

- valori nulli
- valori fuori del dominio di un attributo
- tuple inconsistenti
- tuple con valori uguali per attributi identificanti
- valori inesistenti in attributi usati per corrispondenze tra relazioni

Non tutte le proprietà di interesse sono rappresentabili per mezzo di vincoli di integrità esprimibili direttamente.

4.5.2 Chiave primaria

Due righe distinte non possono avere lo stesso valore sui campi scelti come chiave primaria, inoltre non può assumere valore nullo.

4.5.3 Chiave univoca

Implementa una caratteristica simile alla chiave primaria, con la particolarità che gli attributi coinvolti pososno assumere valore nullo.

4.5.4 Vincoli di tupla

È un vincolo che può essere valutato su ciascuna tupla indipendentemente dalle altre. Un vincolo definito come riferimento ai singoli valori viene detto vincolo sui valori o **vincolo di dominio** in quanto impone una restrizione sul dominio dell'attributo.

4.5.5 Vincoli interrelazionali

Il può usato è il vincolo di integrità referenziale: gli attributi di una data tabella possono assumere soltanto dei valori specificati in un'altra tabella.

Vincoli di integrità referenziale

Un vincolo di integrità referenziale (**foreign key** fra gli attributi di una relazione e di un'altra relazione, impone ai valori della prima relazione di comparire come valori della chiave primaria della seconda relazione.

Capitolo 5

Algebra relazionale

5.1 Operatori insiemistici, ridenominazione e proiezione

I modelli concettuali e logici permettono di descrivere informazioni, ma non sono direttamente interpretabili da un elaboratore. Dai modelli si deve passare ai **linguaggi**, dotati di:

- **sintassi** che definisce le frasi corrette del linguaggio
- **semantica** che definisce le operazioni effettuate quando vegnono eseguite dagli operatori (dette anche istruzioni o comandi) del linguaggio

Esistono principalmente due tipi di linguaggi:

- **procedurali** che specificano le modalità di generazione del risultato; èdi questo tipo l'algebra relazionale
- **dichiarativi** che specificano le proprietà del risultato; sono di questo tipo il calcolo relazionale, l'SQL e il QBE

In questo capitolo si evidenzia l'algebra relazionale, ovvero un insieme di operatori su relazioni, che producono relazioni e possono essere composti tra loro a formare nuove interrogazioni.

5.1.1 Relazioni e insiemi

Le relazioni sono insiemi: dal punto di vista matematico una relazione è un **sottoinsieme del prodotto cartesiano** di due o più insiemi. Prendendo il prodotto cartesiano tra due insiemi si generano delle **n-uple**, un sottoinsieme di queste n-uple è una relazione.

Una relazione, essendo un insieme, ha proprietà:

- non c'è ordinamento tra le diverse tuple
- le tuple sono distinte, non ce ne possono essere due uguali
- ciascuna tupla è aul suo interno ordinata: l'i-esimo valore proviene dall'i-esimo dominio

Quindi, una tabella è una relazione **se e solo se** non ci sono due righe uguali. Su ogni relazione possibamo applicare degli operatori, purchè valgano le seguenti regole:

- è possibile applicare unione, intersezione, differenza solo a relazioni **definite sugli stessi attributi**
- i risultato debbono essere a loro volta relazioni

5.1.2 Operatore Unione

L'operatore unione, $R \cup R'$, produce tutte le n-uple delle relazioni R e R'.

5.1.3 Operatore Intersezione

L'operatore intersezione $R \cap R'$, produce tutte le n-uple che appartengono sia a R sia a R'.

5.1.4 Operatore Differenza

L'operatore differenza $R - R'$, produce tutte le n-uple di R che non compaiono anche in R'.

5.1.5 Operatore di ridenominazione

Gli operatori insiemistici devono essere applicati a relazioni che hanno la stessa struttura e nomi degli attributi, per poter rilasciare questo vincolo si deve poter uniformare i nomi degli attributi: questa funzione è permessa dall'operatore di **ridenominazione**.

L'operatore è definito come $REN_{y \leftarrow x(r)}$, è per questo un operatore **monadico** (dotato di un solo argomento) che produce come effetto la ridenominazione dell'attributo x della relazione r in y: si **modifica lo schema** lasciando inalterata l'istanza di r.

5.1.6 Operatore di Proiezione

in alcune situazioni non è sufficiente usare l'operatore di ridenominazione perchè alcune relazioni potrebbero avere degli attributi in meno, oppure attributi che hanno un significato diverso: la ridenominazione è sintatticamente corretta in questi casi, ma semanticamente errata. Ci si deve quindi **limitare ad un sottoinsieme di attributi** usando la proiezione.

L'operatore è definito come $\sqcap_{listaAttributi}(r)$, è anche questo un operatore monadico e produce una relazione:

- definita su **una parte** degli attributi della relazione r, quelli specificati nella listaAttributi
- che contiene ennuple a cui contribuiscono tutte le n-uple di r

5.1.7 Cardinalità

La cardinalità di una relazione r è il numero delle sue ennuple e si indica con $|r|$.

5.2 Selezione, join naturale, theta join

5.2.1 Operatore di Selezione

È l'operatore **ortogonale** alla proiezione, definito come $\varphi_{condizione}(r)$: decompone orizzontalmente una relazione, producendo una relazione che:

- ha lo **stesso schema** dell'operando
- contiene un **sottoinsieme delle ennuple** dell'operando: quelle che soddisfano la condizione espressa dall'operatore

Condizione

Detta anche **formula proposizionale**, data una relazione r, è una **espressione booleana** ottenuta combinando con i connettivi OR, AND, NOT delle condizioni atomiche.
Ogni condizione atomica è del tipo A op B oppure A op c, dove:

- op è un operatore di confronto
- A e B sono attributi di R su cui valori op abbia senso

· c è una costante per cui il confronto op abbia senso

Si applicano le normali regole di precedenza degli operatori booleani (NOT, AND, OR).

Esempio

Docenti che hanno la classe di stipendio 3 e che hanno il nome uguale al cognome.

$Nome = CognomeANDClasse_stipendio = 3$

Esempio

Professori ordinari o che hanno classe di stipendio superiore a 3.

$Ruolo = 'Ordinario' ORClasse_stipendio > 3$

5.2.2 Operatore Join (Interni)

È un operatore fondamentale dell'AR che ci permette di combinare tuple appartenenti a relazioni diverse: produce un **sottoinsieme del prodotto cartesiano** di due relazioni in cui il valore di determinati attributi coincide. Ne esistono diverse varianti.

Join Naturale

Data una relazione $r1(X1)$ definita sugli attributi X1, e una relazione $r2(X2)$ definita sugli attributi X2, ogni tupla che compare nel risultato del join naturale di r1 e r2 è ottenuta come combinazione di una tupla di r1 con una tupla di r2 sulla base dell'**uguaglianza dei valori degli attributi comuni** alle due relazioni, cioè quelli in $X1 \cup X2$.

Non tutte le tuple degli operandi possono contribuire al risultato: le tuple che non partecipano sono chiamate **dangling** e il join viene detto **incompleto**. Se nessuna tupla partecipa al risultato, il join viene detto **vuoto**.

Se due relazioni sono **definite sugli stessi attributi**, allora il join naturale equivale all'intersezione delle due relazioni.

Si formalizza con $r_1 \triangleright \triangleleft r2$.

Theta Join

Il prodotto cartesiano ha senso solo se seguito da selezione: per la **selezione sul prodotto cartesiano** esiste questo operatore dedicato che si applica solo quando X1 e X2 sono operatori disgiunti e introduce una **condizione** da rispettare dopo aver calcolato il prodotto cartesiano.

Si formalizza con $r_1 \triangleright \triangleleft_{condizione} r2$.

Equi-Join

È un Theta Join in cui la condizione da rispettare è una **uguaglianza**.

5.2.3 Join Esterno

Un problema dei join interni sono le tuple dangling, che non partecipano al join e fanno diventare il join incompleto. Questo succede perchè in un join iterno si hanno solo **le tuple che si accoppiano**. Ma si potrebbe anche voler avere una vista completa, anche quelle tuple non interessate dal join.

In un join esterno compaiono **tutte le tuple, anche quelle non coinvolte**, che vengono associate a valori **nulli** laddove mancano dei valori negli attributi. Esistono, anche in questo caso, diversi tipi di join.

La sintassi è del tipo $r_1 \triangleright \triangleleft_{LEFT/RIGHT/FULL} r2$.

Left Join

Mantiene tutte le ennuple del primo operando, estendendendole con valori nulli, se necessario.

Right Join

Mantiene tutte le ennuple del secondo operando, estendendendole con valori nulli, se necessario.

Completo

mantiene tutte le ennuple di entrambi gli operandi, estendendendole con valori nulli, se necessario.

5.2.4 Self Join

Si effettua un join di una relazione con una copia di se stessa per:

- **confrontare tuple** di una relazione tra loro
- trovare elementi **duplicati**
- quando una tupla fa **riferimento** a dati nella stessa tabella, come nelle relazioni:
  - gerarchiche
  - sequenziali
  - a grafo

5.3 Valori Nulli

Non sempre l'informazione nelle basi di dati è completa, e questo influisce sulle interrogazioni.

5.3.1 Logica a 3 valori

Oltre ai valori di verità Vero e Falso, si introduce il valor **Unkown**: vale sempre la regola che la selezione produce le sole tuple per cui l'espressione di predicati risulti vera.

5.3.2 Is (not) NULL

Per riferirsi ai valori nulli esistono le apposite condizioni IS NULL e IS NOT NULL, che esplicitano come un attributo può essere, o non essere, NULL.

5.4 Viste

Per semplificare, soprattutto nel caso di sotto espressioni spesso ripetute, è utile avere delle **relazioni derivate** a partire dalle relazioni definite nello schema di base di dati. A questo scopo, in algebra relazionale è possibile definire delle viste, che altro non sono che **espressioni a cui viene assegnato un nome**.
È quindi possibile utilizzare le viste all'interno di altre espressioni, il che semplifica la scrittura di espressioni complesse. La sintassi è $Nomevista = EspressioneAR$.

5.5 Interrogazioni in AR

Dall'espressione della interrogazione in linguaggio naturale, bisogna prima trovare l'**insieme di relazioni** su cui applicare gli operatori. Si procede secondo la strategia dall'interno all'esterno:

1. prima cercare di capire se servono dei join sulle relaizoni in R che devono essere collegate per costruire il risultato finale
2. poi cercare di capire se vi sono operazioni insiemistiche, che non alterano la struttura della relazione ma avvicinano al risultato finale
3. poi individuare eventuali selezioni necessarie per rappresentare analoghe selezioni esistenti nella interrogazione in linguaggio naturale
4. infine, definisce su quali attributi fare proiezioni, ed in particolare la proiezione finale.

Capitolo 6

Linguaggio SQL

6.1 Strucuted Query Language

Il linguaggio SQL è il linguaggio per la definizione e la manipolazione dei dati in database relazionali, adottato da tutti i principali DBMS.
Nonostante sia uno standard, esistono diverse features che possono essere implementate o meno.

6.1.1 SQL e AR

L'SQL è un **relazionalmente completo**: ogni espressione dell'algebra relazionale può essere tradotta in SQL, infatti adotta la logicaa 3 valori ed è computazionalmente completo.

6.1.2 Caratteristiche

È un linguaggio **dichiarativo**: non possiamo scegleire l'ordine con cui avvengono le operazioni ma ci si deve attenere ad una **struttura sintattica** delle istruzioni.

6.1.3 SELECT-FROM-WHERE

l'operatore SELECT seleziona tra le n-uple del prodotto cartesiano delle tabelle citate nella FROM, quelle che soddisfano la condizione presente nel WHERE, e di esse ne rappresenta solo gli attributi nella ListaAttributi.

6.1.4 Condizione LIKE

Seleziona stringhe di caratteri che corrispondano ad una certa regola, che contiene anche gli operatori:

- _ indica un singolo carattere arbitrario
- % indica una stringa con un numero arbitrario di caratteri

6.1.5 NULL

La gestione dei valori null avviene esattamente come in AR, attravreso gli operatori IS NULL e IS NOT NULL.

6.2 SQL Query

6.2.1 Prodotto cartesiano

Per produrre un prodotto cartesiano tra più tabelle, elencare più tabelle nella clausola FROM, a prescindere dall'insieme di attributi delle tabelle.

6.2.2 Theta Join

Se al prodotto cartesiano precedente applichiamo la condizione WHERE, otteniamo una selezione sul prodotto cartesiano, ovvero un Theta Join.
A differenza dell'AR, Il theta join ha **sempre senso**, perché si combinano solo le tuple che hanno lo stesso valore per gli attributi specificati nella condizione di join.

6.2.3 Clausola Punto

La clausola **.** permette di specificare a che relazione appartiene un attributo, o anche una tabella a un database.

6.2.4 Ridenominazioni

Si possono fare ridenominazioni su attributi e anche su relazioni con la clausola **AS**.

6.2.5 Costrutto Join

Si può esplicitare il predicato Join direttamente nella clausola FROM utilizzando il costrutto **JOIN ON** anche detto join esplicito.
Esplicitare il Join consente di dichiarare anche quale tipo di join si vuole utilizzare, tra INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER.

Esempio

La clausola definita come:

```
SELECT *
FROM Personale_docente, Stipendio
WHERE Classe_stipendio=Classe AND
Valore>=60000 AND Ruolo='Ricercatore'
```

Viene esplicitata come:

```
SELECT *
FROM Personale_docente JOIN Stipendio ON
Classe_stipendio=Classe
WHERE Valore>=60000 AND Ruolo='Ricercatore'
```