Riassunto basi di dati

Dimix

2025-03-13

Indice

Lezione 1 – Introduzione

2 Introduzione alle basi di dati

2.1 Definizione e contesto

Una **base di dati** è una collezione di dati correlati. Per **dato** intendiamo la registrazione simbolica di un elemento della realtà.

Una base di dati possiede alcune caratteristiche distintive:

- Rappresenta un aspetto limitato della realtà, riferito a un sottoinsieme dei dati disponibili denominato **universo del discorso**.
- Costituisce un insieme di dati logicamente coerenti e dotati di un significato intrinseco.
- È progettata, costruita e popolata per uno scopo specifico, rivolto a un gruppo di utenti determinato e ad alcune specifiche applicazioni.

In una base di dati i dati sono correlati per fornire un'interpretazione e trasformarli in **informazione**.

Esempi:

- Dati non correlati: `Roma, 41.89, 12.48`
- Dati correlati: `Roma → Posizione: Latitudine 41.89, Longitudine 12.48`

Esistono tre tipologie di dati

1. **Dati non strutturati**: tipicamente disponibili in formato testuale (memorizzati su file o sistemi di gestione documentale)
2. **Dati semi-strutturati**: tipicamente disponibili in formati come XML, HTML, JSON, RDF, presentano una informazione struttura ma con pochi vincoli di formato e di coerenza dei dati, insieme ad una frequente ridondanza.
3. **Dati strutturati**: tipicamente in formato tabellare, memorizzati in DBMS relazionali o fogli di calcolo come Excel, la loro struttura è rigida, definita da un insieme di campi che si applicano a tutti gli oggetti descritti ma non è detto che siano rispettati vincoli di formato, coerenza dei dati e che non vi sia ridondanza.

2.2 Considerazioni generali sull'uso delle basi di dati

Le basi di dati in confronto alla gestione applicativa

Ogni applicazione possiede un sistema di gestione dei dati, tipicamente in memoria e su file andando a definire di volta in volta le strutture dati necessarie e l'organizzazione dei file, utilizzare invece una base di dati per la gestione dei dati in ausilio delle applicazioni è una scelta che implica alcune caratteristiche specifiche nel modo in cui i dati vengono trattati rispetto alle applicazioni.

Le caratteristiche delle basi di dati

- **Autodescrizione**: Una base di dati contiene sia i dati che una descrizione completa della sua struttura e dei suoi vincoli, attraverso un catalogo.
- **Indipendenza tra programmi e dati**: La struttura di memorizzazione dei dati è astratta e coerente al modello dei dati usato dalla base di dati, in tutti i casi essa è indipendente dalle applicazioni
- **Viste multiple dei dati**: Una base di dati può fornire un accesso a sottoinsiemi e a forme di organizzazione diversi dei dati a diversi utenti e applicazioni per mezzo di viste virtuali
- **Condivisione e controllo della concorrenza**: una base di dati garantisce che utenti e applicazioni diverse possano gestire in modo concorrente gli stessi dati, garantendo al tempo stesso la loro coerenza e correttezza per metto di transazioni.

3 I sistemi di gestione delle basi di dati

3.1 Modelli dei dati

L'organizzare i dati in una base di dati richiede la definizione di un modello logico che detti le regole di organizzazione dei dati stessi.

Il modello dei dati è un insieme di costrutti, formalismi di rappresentazione e operazioni di base necessari alla realizzazione della descrizione astratta e indipendente dalla memorizzazione fisica dei dati, ai vincoli che ne garantiscono la coerenza e correttezza e alla loro gestione.

Il modello relazione è il modello dei dati più diffuso, esso verrà trattato all'interno del nostro corso, esisto anche altri modelli come possono essere quelli *reticolari*, *gerarchici* e ad *oggetti*.

Le nozioni di schema e istanza

- **Schema** (Intensione): è un modello logico che definisce la struttura e i vincoli dei dati da rispettare (fisso nel tempo).
- **Istanza** (Estensione): insieme dei dati organizzati rispettando lo schema (varia nel tempo).

  L'istanza della base di dati in un particolare istante di tempo è detta **stato della base di dati**.

3.2 Architettura di un DBMS

Un **DBMS (Database Management System)** è un insieme di programmi che consente la creazione e manutenzione di basi di dati, in particolare consente di definire, costruire, manipolare e condividere basi di dati per vari utenti e applicazioni.

> **Obiettivo del DBMS** è gestire i dati in modo efficiente evitando duplicazioni, errori, e garantendo integrità e sicurezza.

Architettura a tre livelli

L'architettura concettuale di una base di dati è caratterizzata da tre livelli: 1. **Livello interno** (fisico): memorizzazione fisica dei dati. 2. **Livello logico**: rappresentazione logica della base di dati. 3. **Livello esterno**: viste personalizzate per gli utenti.

3.3 Le categorie di linguaggi nei DBMS

- **DDL (Data Definition Language)**: istruzioni per la creazione e manutenzione dello schema.
- **DML (Data Manipulation Language)**: istruzioni per la manutenzione e interrogazione dei dati.
- **DQL (Data Query Language)**: istruzioni per l'interrogazione dei dati di una base di dati.
- **DCL (Data Control Language)**: istruzioni per gestire permessi e i controlli di accesso alla base di dati.

3.4 Utenze di un DBMS

in una base di dati esistono due macrocategorie di utenti: - **DBA User**: Utenti amministratori dotati di privilegi massimi rispetto ad una o più basi di dati, possono essere in possesso delle funzionalità DDL, DML, DQL, DCL, gestire i privilegi e la sicurezza, gestiscono l'ottimizzazione delle basi di dati e gestiscono lo spazio di memorizzazione fisica. - **DB User**: utenti con privilegi specifici e limitati su una o più basi di dati, essi sono suddivisi in: - Developer user - Occasional user - Casual user

I DBMS adottano un'architettura di comunicazione client/server, l'accesso può avvenire da postazioni locali tramite pipe del sistema operativo o remote tramite socket TCP/IP.

Gli utenti del DMBS utilizzano un software client per comunicare con il server, questo software può essere basato su: - Riga di comando - Interfaccia grafica - Interfaccia web

Lezione 2 – Il modello relazionale

1 Il modello relazionale

1.1 Definizione

Il **modello relazionale** fu proposto da Edgar Codd nel 1970 e introdotto nei DBMS nel 1981.

- Basato sulla nozione di **relazione matematica**, intesa come sottoinsieme del prodotto cartesiano tra insiemi di dati, detti domini.
- Definisce un insieme di vincoli sui dati.
- Associato al linguaggio dell'**algebra relazionale** per eseguire interrogazioni.

Tabella e domini

La rappresentazione più intuitiva di una relazione è la tabella, una base di dati relazionale è rappresentata da una collezione di tabelle.

Ogni tabella ha un nome unico nella base di dati, con le seguenti proprietà: - Ogni **riga** rappresenta un record. - Ogni **colonna** ha un nome distinto (**attributo** (A_k)), associato a un insieme di valori detto **dominio** (D_k).

Per dominio viene intesa intesa una collezione di valori atomici, in termini pratici, i domini a partire dai quali sono costruite le relazioni nei DBMS sono definiti a partire da tipi di dati, come ad esempio stringhe, interi e date.

La tabella

Dati n domini (D1, D2 ,…, Dn), ogni riga di una tabella è una ennupla ordinata di valori (d1, …, dn) con dk appartenente al dominio Dk del corrispondente attributo Ak Una tabella contiene un sottoinsieme di tutte le righe possibili, cioè un sottoinsieme del prodotto cartesiano D1 x D2 x … x Dn.

La relazione matematica

Siano D1, D2 ,…, Dn n insieme di valori anche non distinti, il prodotto cartesiano D1 x D2 x … x Dn. è definito cosi: D1 x D2 x … x Dn = {(d1, d2, …, dn) tale che d1, d2, …, dn appartengono a D1 , D2, …, Dn} Una relazione matematica R su D1, D2 ,…, Dn è cosi definita: R sottoinsieme di D1 x D2 x … x Dn D1, D2 ,…, Dn sono domini di R, una relazione su n domini è detta avere grado n Il numero di ennuple di una relazione è detto cardinalità della relazione, nelle applicazioni reali la cardinalità di una relazione è sempre finita

1.2 Proprietà delle relazioni

Relazione con attributi

La relazione con attributi, nella quale un attributo che descrive il ruolo del dominio nella relazione viene associato ad ogni occorrenza di dominio nella relazione. Una ennupla su un insieme di attributi X è una funzione t[Ak] –> Dk che associa a ciascun attributo un valor del dominio Dk di Ak, quindi t[Ak] denota il valore della ennupla t sull'attributo Ak

La rappresentazione tabellare

I valori di ciascuna colonna sono fra loro omogenei, i valori di un attributo appartengono allo stesso dominio. Le righe sono diverse fra loro, una relazione non contiene mai ennuple identiche per via dell'orientamento ai valori. l'ordinamento delle colonne è irrilevante poiché esse sono sempre identificate per nome e non per posizione, lo stesso vale per le righe in quanto identificate per contenuto

L'orientamento ai valori

In un modello relazione, per via dell'orientamento ai valori, le righe di una tabella non hanno un identificativo fisso, l'unico modo per distinguere le righe tra di loro è guardare i valori che contiene.

Per fare riferimento ad una riga specifica non possiamo fare uso del numero della riga, in quanto queste non hanno un posizione fissa ma variabile nel tempo, quindi per andare a identificare una riga dobbiamo fare riferimento ai suoi valori interni. Nel modello relazionale sono presenti due livelli principali per descrivere una relazione, cioè la tabella, il livello intensionale e il livello estensionale.

Il livello intensionale rappresenta la struttura della tabella ed include il nome della relazione, più l'insieme degli attributi e i domini dei valori che ogni attributo può assumere.

Il livello estensionale rappresenta l'insieme di ennuple che la tabella contiene in un dato momento, a differenza del livello intensionale questo può variare nel tempo.

2 L'informazione incompleta

2.1 Cause e soluzioni

Il modello relazione impone ai dati una struttura rigida, le informazioni sono rappresentate tramite ennuple con formati predefiniti, esso impone anche che tutti i valori vengano specificati ma è possibile che non tutti i valori siano noti o disponibili per cui è necessario trovare un modo per rappresentarli.

Possiamo farlo in due modi tramite l'utilizzo di valori del dominio "non utilizzati", come può essere la stringa vuota o il valore zero, però può capitare che non esistano valori non utilizzati o non significativi, per cui è stato introdotto il valore NULL, questo denota l'assenza di un valore del dominio.

Bisogna stare attenti perché il valore NULL non ha una semantica definita, a seconda dei casi può rappresentare un valore sconosciuto, un valore inesistente o un valore senza informazione.

3 Vincoli di integrità

3.1 Vincoli di integrità

Nelle basi di dati esistono istanze che pur sintatticamente corrette non rappresentano informazioni possibili per l'applicazione di interesse, come può essere un valore negativo associato alla popolazione di un paese.

Il vincolo di integrità indica delle proprietà che devono essere soddisfatte dalle istanze per far si che le informazioni siano corrette per l'applicazione, un vincolo quindi è una funzione booleana che associa ad ogni istanza i valori vero o falso.

Esistono due tipologie principali di vincoli: - I vincoli intrarelazionali: vincoli sui valori o domini, vincoli di ennupla o vincoli di chiave - I vincoli interrelazionali: vincoli di integrità referenziale, vincoli di chiave esterna

I vincoli di dominio

Un vincolo di dominio definisce l'insieme di valori validi che un attributo può assumere, la loro verifica si basa su predicati booleani.

Tali predicati sono valutati al momento dell'inserimento o modifica di un valore nella base di dati andando a garantire che vengano memorizzati solo se il predicato valutato è vero.

I vincoli di ennupla

I vincoli di ennupla sono condizioni da soddisfare sui valori di ciascuna ennupla, indipendentemente dalle altre ennuple, una possibile sintassi è un'espressione booleana che confronta i valori di un attributo con determinati valori scelti oppure tramite espressioni aritmetiche su di essi, anche essi sono verificati al momento dell'inserimento o della modificata.

I vincoli di dominio sono un caso particola di vincoli di ennupla che coinvolgono un solo attributo.

3.2 I vincoli di chiave

Una **chiave** è un insieme di attributi che identificano univocamente le ennuple di una relazione.

Una **superchiave** di una relazione è un insieme di attributi, può essere anche uno singolo, che permette di identificare in modo univoco ogni riga di una tabella, cioè se consideriamo una superchiave di una riga e la andiamo a confrontare con ogni riga della nostra tabella non ne possiamo trovare una uguale.

Quindi possiamo dire che una chiave è una superchiave con il minor numero possibile di attributi, tale che rimuovendo anche solo un attributo si perderebbe la proprietà di unicità, questo viene detto minimalità.

Trovare una superchiave minimale è il nostro obbiettivo all'interno delle basi di dati.

Nelle basi di dati ogni relazione ha almeno una chiave, per cui una relazione non può contenere ennuple uguali ed ogni relazione ha come superchiave l'insieme degli attributi su cui è definita.

L'esistenza delle chiavi garantisce l'accessibilità a ciascun dato della base di dati e tramite l'utilizzo delle chiavi riusciamo ad organizzare i dati in relazioni diverse andando ad usare le chiavi primarie ed esterne.

Nelle chiavi la presenza di valori nulli deve essere limitata, altrimenti non è possibile identificare le ennuple e realizzare facilmente i riferimenti da altre relazioni, quindi vogliamo imporre a ogni relazione la presenza di almeno una chiave priva di valori nulli, in modo che ci sia sicuramente una combinazione di dati che consenta l'accesso univoco alle ennuple della base di dati.

La **chiave primaria** è una chiave su cui non sono ammessi i valori nulli, solitamente è evidenziata tramite una sottolineatura, in SQL vengono definite tramite il vincolo PRIMARY KEY.

3.3 L'integrità referenziale

Vincoli di integrità referenziale

Solitamente informazioni in relazioni diverse sono correlate attraverso valori comuni, per via dell'orientamento ai valori, in particolare tramite i valori delle chiavi, solitamente primarie.

Un vincolo di integrità relazione impedisce l'inserimento di un valore in una chiave esterna se non esiste nella tabella referenziata, una chiave esterna, detta foreign key, indica un valore di una tabella che esiste sicuramente in una tabella correlata, solitamente è associato alla chiave primaria ma basta che sia associato ad una chiave unica, in SQL vengono definite tramite il vincolo FOREIGN KEY.

4 Forme di relazione

Anomalie dovute alla cattiva strutturazione dei dati

L'introduzione dei vincoli di integrità garantisce la correttezza e buona qualità dei dati, ma non risolve tutte le anomalie possibili, quali eccesso di valori ridondanti e/o valori nulli nelle ennuple, possibilità di generare ennuple spurie.

Questo genere di anomalie conduce frequentemente a errori e ulteriori anomalie nelle operazioni di inserimento, aggiornamento e cancellazione dei dati.

Possiamo risolvere queste anomalie seguendo questi principi:

- un singolo oggetto reale deve corrispondere a una e una sola ennupla
- una classe di oggetti con le stesse proprietà deve corrispondere a una sola relazione
- ogni cella di una tabella deve contenere un valore atomico
- relazione in modo corretto i dati di tabelle diverse in modo da evitare di generare corrispondenze scorrette in fase di join

Dipendenze funzionali

Una dipendenza funzionale tra due insiemi di attributi x, y, denotata come $x \to y$, indica che, in una relazione, se due tuple condividono gli stessi valori per gli attributi in x, allora devono avere gli stessi valori per gli attributi in y, cioè i valodi di x determinano univocamente i valori di y.

Le dipendenze funzionali sono vincoli che descrivono come sono collegati tra loro i dati in una tabella, devono essere definite dal progettista sulla base della conoscenza della realtà di interesse, e servono a definire gli stati validi di una relazione, ovvero specificare legami logici fra dati che devono valere sempre, per qualsiasi istanza della relazione.

Un esempio è l'attributo matricola per uno studente universitario, qualsiasi studente è dotato di una matricola univoca.

Regole di inferenze per dipendenze funzionali

Un progettista individua le dipendenze funzionali più evidenti, dato l'insieme F delle dipendenze iniziali individuate, è possibile dedurre altre dipendenze sulla base delle seguenti regole di inferenza. L'insieme F^+ delle dipendenze funzionali individuate dal progettista e quelle inferite prende il nome di **chiusura** di F.

1. Regola *riflessiva*:
2. Regola di *arricchimento*:
3. Regola *transitiva*:
4. Regola di *decomposizione*:
5. Regola di *unione*:
6. Regola *pseudo-transitiva*:

Normalizzazione delle relazioni

Nei nostri database vogliamo ottenere uno schema, cioè un insieme di tabelle collegate, che ha un certo livello di normalizzazione, in base a quanto vogliamo che sia pulito ed efficiente. per fare ciò ci basiamo sulle dipendenze funzionali per controllare se una tabella con le sue chiavi è ben costruita, per fare questo controllo ci serviamo delle forme normali, cioè regole di qualità che una tabella dovrebbe rispettare. Se una tabella non rispetta una forma normale, significa che ha delle anomalie o soffre di ridondanza.

Gli obbiettivi della normalizzazione sono di garantire un join senza perdita di dati, cioè quando spezziamo una tabella in più parti, dovremmo poter ricostruire la tabella iniziale solo tramite operazioni di join, non deve comparire nessuna nuova riga o sparirne alcuna. Un altro obbiettivo è la conservazione delle dipendenze, anche dopo aver spaccato la tabella, le dipendenze funzionali stabilite devono ancora essere valide.

Struttura degli attributi

- **Attributo semplice**: costituito da valori atomici
- **Attributo multivalore**: in cui un possibile valore è un insieme di valori
- **Attributo strutturato**: in cui un possibile valore è una ennupla di valori

4.1 Prima forma normale 1NF FARE ESEMPIO

Una relazione è in **prima forma normale 1NF** se ogni colonna contiene un solo valore, non ci sono gruppi ripetuti di dati dentro una stessa riga, i valori sono atomici.

Nel modello relazione la prima forma normale deve sempre essere rispettata, avendo dati semplici e atomici è facile:

- capire cosa significano
- gestire operazioni come inserimenti, modifiche e cancellazioni
- interpretare i risultati delle ricerche o query

Quindi in presenza di attributi strutturati o multivalore occorre sostituire l'attributo strutturato con attributi atomici e gli attributi multivalore con una relazione separata che contenga la chiave primaria della relazione originaria.

4.2 Seconda forma normale 2NF FARE ESEMPIO

La **seconda forma normale 2NF** si basa sul concetto di dipendenza funzionale completa, cioè una dipendenza funzionale $X \to Y$ si dice completa se tutti gli attributi di X sono necessari per determinare Y, se togli anche solo un pezzo di X, la dipendenza non funziona più.

Quindi se basta solo una parte di X per determinare Y abbiamo una dipendenza parziale e quindi non è adatta per la seconda forma normale.

Uno schema è in seconda forma normale se:

- è gia in prima forma normale
- ogni attributo non-primo, cioè che non fa parte di nessuna chiave, dipende dall'intera chiave primaria, non solo da una sua parte

In caso di una dipendenza parziale, è necessario effettuare la normalizzazione per avere una seconda forma normale valida, per normalizzare dobbiamo:
Spezzare la tabella, creare una nuova tabella per ogni dipendenza parziale, e in questa nuova tabella inserire la parte di chiave da cui dipende e il dato che dipende.

4.3 Terza forma normale 3NF FARE ESEMPIO

La **terza forma normale 3NF** si basa sul concetto di dipendenza transitiva. Una dipendenza transitiva succede quando tre attributi rendono valida la regola di transitività.

Uno schema di relazione è in terza forma normale se:

- è gia in seconda forma normale
- nessun attributo non-primo, cioè che non fa parte della chiave primaria, dipende dalla chiave tramite un altro attributo
- tutti gli attributi non-chiave devono dipendere direttamente dalla chiave, non passando da altri attributi

In caso di dipendenza transitiva bisogna spezzare la tabella, creare una nuova tabella che contiene l'attributo intermedio e gli attributi che lui determina, nella tabella origilinale tieni l'attributo intermedio come ponte di collegamento fra le nuove relazioni.

4.4 Forma normale di Boyce-Codd BCNF FARE ESEMPIO

La **forma normale di Boyce-Codd BCNF** è una versione ancora più rigorosa della terza forma normale, quindi una tabella BCNF sarà anche in terza forma normale, ma non vale il contrario.

Uno schema è in BCNF se ogni dipendenza funzionale $ X →Y$ che esiste nella tabella ha come X una superchiave, cioè se un insieme di attributi X determina un altro attributo Y, allora X deve essere abbastanza forte da identificare univocamente una riga, in pratica X deve essere o contenere una chiave.

5 Proprietà delle decomposizioni

5.1 Proprietà

Nella progettazione di un database possiamo usare la teoria delle normalizzazione per pensare al database come una unica grande relazione universale R che contiene tutti gli attributi dello schema. Da questa relazione iniziale andiamo ad ottenere uno schema composto da più relazioni tramite le decomposizioni. Lo schema finale è detto decomposizione di R e denotato $D = R_1, R_2, \cdots, R_m$. Ogni attributo di R deve essere

presente in almeno una relazione R_i di D per conservare tutti gli attributi, in modo che vale:

$$\bigcup_{i=1}^{m} R_i = R$$

Una buona decomposizione dovrebbe rispettare alcune proprietà:

- Conservazione delle dipendenze: dato un insieme di dipendenze F su R, andiamo a definire **proiezione** di F su R_i$ l'insieme delle dipendenze $X →Y4 di F^+ tali che gli attributi $X \cup Y$ siano tutti contenuti in R_i. Una decomposizione D conserva le dipendenze di R se:

  $$(\pi R_1(F) \cup \cdots \cup \pi R_m(F))^+ = F^+$$

  Cioè le regole devono rimanere valide anche dopo aver diviso la tabella.
- Proprietà di join non-additivo (senza perdita): una decomposizione D è senza perdita rispetto all'insieme F di dipedenze di R se, per ogni stato di relazione di R che soddisfa F, data l'operazione di JOIN vale che:

  $$*(\pi R_1(r), \cdots, \pi R_m(r)) = r$$

  Cioè riusciamo a ricostruire la tabella iniziale, senza errori sui dati.

5.2 Algoritmi di progettazione

Sintesi relazione in 3NF con conservazione delle dipendenze

Abbiamo come obiettivo la costruzione di un insieme di tabella che siano in terza forma normale e mantenga tutte le dipendenze originali. il nostro input è una relazione universale R e un insieme di dipendenze funzionali F.

I passi da seguire sono:

- Semplificare F togliendo le dipendenze o attributi ridondanti nelle dipendenze, in modo da trovare la copertura minimale G di F.
- Creare una tabella che contiene X insieme agli attributi che X determina, se da X derivano più attributi, li metti tutti nella stessa tabella, nella quale avremo le dipendenze in G con X come parte sinistra.
- Mettere tutti gli attributi rimanenti in un'unica relazione per assicurare la conservazione degli attributi.

Decomposizione in BCNF senza perdita

Abbiamo come obiettivo di rendere lo schema ancora più pulito, come input abbiamo la relazione universale R e l'insieme di dipendenze funzionali F.

I passi da seguire sono:

- inizializzazione $D = R$
- ripetere le seguenti finche esiste una tabella Q in D che non è in BCNF
- trovare una dipendenza $X \to Y$ in Q che viola la BCNF
- sostituire Q con due nuove tabelle; una contenente gli attributi $X \cup Y$ e l'altra quelli $Q - Y$

Lezione 3 - Modellazione concettuale (ADD IMMAGINI)

1 Sistemi di basi di dati

1.1 Criteri di progettazione

Raccolta e analisi dei requisiti

Per primo passo è necessaria la comprensione degli obbiettivi della base di dati, anche tramite una interazione con gli utenti che descrivono le necessità, i problemi, gli obbiettivi facendo uso del linguaggio naturale, ideare una formulazione di descrizioni informali dei dati, cioè i loro requisiti, e delle operazioni che verranno effettuate sui dati, cioè requisiti funzionali.

Progettazione della base di dati

Progettazione concettuale: utilizzo di uno schema concettuale per rappresentare i requisiti informali sulla realtà di interesse tramite una descrizione concettuale formale e completa. Questo schema è indipendente dallo specifico DBMS che verrà utilizzato per la gestione della base di dati.

Progettazione logica: traduzione dello schema concettuale nel modello dei dati adottato dal DBMS scelto per la gestione della base di dati. Questo tipo di progettazione è dipendente dal DBMS scelto.

Progettazione fisica: la schema logico ottenuto viene completato con la specifica dei parametri fisici di memorizzazione dei dati, quindi l'organizzazione dei file e indici.

Modelli dei dati

Modelli concettuali: utilizzati nella fase di progettazione concettuale per costruire schemi concettuali, ovvero descrizioni dei dati ad alto livello di astrazione, indipendente dallo specifico DBMS. Un esempio di esso è il modello entità-relazione ER.

Modelli logici: utilizzati nella fase di progettazione logica per ottenere schemi logici, ovvero descrizioni dei dati processabili dal DBMS. Un esempio è il modello relazionale.

La nozione di astrazione

Il criterio fondamentale per la progettazione è l'astrazione, ovvero il processo mentale per estrapolare le proprietà e caratteristiche essenziali di un insieme di oggetti, ignorando le differenze e i dettagli non essenziali, ovvero una selezione delle proprietà rilevanti di un insieme di oggetti.

L'obbiettivo dell'astrazione è le definizione di descrizioni generali di classi di oggetti simili, prescrindendo dalle caratteristiche specifiche di ogni oggetto.

Criteri di astrazione

- **Classificazione**: individuazione di una classe o insieme che possa caratterizzare un gruppo omogeneo di individui specifici. Per esempio; simone, adriano, kevin, … → Studente.
- **Aggregazione**: individuazione di un concetto che possa essere definito dall'insieme delle sue parti. Per esempio; Docente, Aula e Studenti → Corso
- **Generalizzazione**: individuazione di una classe che possa rappresentare entità simili tramite una entità generale. Per esempio; Studente, Docente → Persona

Interpretazione logica dei meccanismi di astrazione

I meccanismi di astrazione hanno un immediato corrispettivo in termini insiemistici, eccetto l'aggregazione che corrisponde alla relazione "essere parte di".

La classificazione è rappresentata dal simbolo di appartenenza $\in$, per esempio se x è classificato in una classe Y si dice $x \in Y$

La generalizzazione è rappresentata dal simbolo di sottoinsieme $\subseteq$, per esempio date due classi X e Y, possiamo dire che X generalizza Y, $X \subseteq Y$, se $\forall x | x \in X \rightarrow x \in Y$

La definizione di generalizzazione implica che tutte le caratteristiche proprie degli oggetti classificati in Y saranno necessariamente proprie anche degli oggetti classificati in X, possiamo dire che X eredita le caratteristiche di Y.

2 Progettazione concettuale

2.1 Il modello ER

Modello Entity-Relationship (ER)

Il principale modello dei dati usato per la progettazione concettuale è il modello ER. Ne esistono anche versioni estese o diverse per notazione, cosi come esistono anche altre alternative al modello ER per la progettazione concettuale, come UML.

Costrutti

I costrutti principali del modello ER sono:

- Entità
- Relazione
- Attributo

I costrutti secondari sono:

- Cardinalità
- Identificatori
- Attributi composti
- Gerarchie di generalizzazione

2.2 Entità, relazioni e attributi TO DO ESEMPI

Entità

L'entità rappresenta una classe di oggetti che possiedono proprietà comuni ed esistena autonoma ai fini dell'applicazione di interesse.
Possono essere oggetti con esistenza fisica, come per esempio Territory, oppure oggetti che sistono a livello concettuale, come per esempio Name.

Ogni entità all'interno di uno schema ha un nome che la identifica univocamente ed è rappresentato graficamente da un rettangolo.

Relazione o associazione

La relazione rappresenta un legame logico tra due o più entità, per esempio l'associazione *has_name* fra country e name rappresenta il fatto che una nazione può avere diversi nomi in lingue diverse.

Un'associazione R all'interno di uno schema ER ha un nome che la identifica univocamente, essa è rappresentata da un rombo che collega le entità interessate.

Vediamo degli esempi:

Attributi

Gli attributi sono necessari per descrivere una proprietà elementare delle entità e delle associazioni di interesse. Ogni istanza di entità o relazione possiede un valore per ciascuno dei suoi attributi. Il valore di un attributo appartiente a un dominio dell'attributo che contiene i valori ammissibili, come per esempio intero, stringa, data eccetera.

Esistono gli **attributi composti**, i quali raggruppano attributi che hanno affinità d'uso o di significato. L'insieme di questi attributi prende il nome di attributo composto.

2.3 Cardinalità

Vincoli di cardinalità

I vincoli di cardinalità esprimono le regole che stabiliscono la partecipazione delle istanze di un'entita a una relazione con altre entità. Data una relazione R tra due entità, E1 ed E2, avremo due vincoli di cardinalità differenti quello che va da E1 a R e quello da E2 a R.

Ogni vincolo di cardinalità è rappresentato da una coppia di valori (x,y), dove x indica la cardinalità minima e y la cardinalità massima.

Cardinalità minima

La cardinalità minima esprime il numero minimo di partecipazioni che una entità deve avere in una relazione, in pratica definisce se la partecipazione è obbligatoria o facoltativa.

Cardinalità massima

La cardinalità massima esprime il numero massimo di partecipazioni che una entità può avere in una relazione, quindi serve a limitare il numero di coinvolgimenti.

Tipologie di associazioni binarie

In base ai valori delle cardinalità si hanno le seguenti tipologie di relazioni binarie:

- (1,1) uno a uno; entrambe le entità hanno cardinalità 1
- (1,n) uno a molti; la cardinalità massima è n, la minima 1
- (n,m) molti a molti; entrambe le entità hanno cardinalità n,m

Cardinalità degli attributi

La cardinalità degli attributi rappresenta quanti valori può assumere un attributo per ogni singola istanza di entità o relazione. esso è rappresentato sempre da una coppia di valori.

La coppia (1,1) si verifica nella maggioranza dei casi, e di defualt è omesso, esso indica la obbligatorietà di uno e un solo valore.
Un valore di una coppia può essere nullo, solitamente la cardinalità minima può essere nulla ed indica che è un valore di attributo opzionale.
La cardinalità massima può essere n, ovvero quando un attributo può assumere diversi valori, fino ad un massimo di n.

2.4 Identificatori

Un identificatore di una entità E è un insieme di attributi e/o entità connesse ad E che permettono di identificare univocamente le istanze di E. Vi sono due tipi di identificatori, quello interno e quello esterno.

Gli identificatori interni

Un identificatore si dice interno quando è costituito esclusivamente da attributi dell'entità identificata. Può essere semplice, cioè costituito da un solo attributo oppure composto, cioè costituito da più attributi.

Gli identificatori esterni

Un identificatore esterno è necessario quando gli attrivuti di E non sono sufficienti per formare un identificatore per E, in questo caso E è detta **entità debole**. In questo caso si utilizzano entità con cui E ha un vincolo di dipendenza, per esempio si considerano tipi di associazioni binarie con cardinalità (1,1).

2.5 Gerarchie di generalizzazione

Una gerarchia di generalizzazione è quando un'entità E più generale, detta *superclasse*, viene specializzata in entità più specifiche E1, E2,… dette *sottoclassi*, che ereditano attributi e relazioni dalla superclasse.

Da questa definizione ne esce fuori il concetto di **ereditarietà**, infatti ogni proprietà di una superclasse E è ereditata da tutte le sottoclassi di E. Le proprietà ereditate non vanno rappresentate esplicitamente.

vincoli sulle specializzazioni

Una gerarchia è detta **totale** se ogni entità della superclasse è istanza di almeno una sottoclasse della specializzazione, altrimenti è detta **parziale**

Una gerarchia è detta **disgiunta** o **esclusiva** se un'istanza può essere istanza di una sola sottoclasse, altrimenti è detta **sovrapposta** quando può essere istanza di più sottoclassi contemporaneamente.

Esistono quindi le seguenti tipologie di gerarchie:

- Totale-Esclusiva (TE)
- Totale-Sovrapposta (TO)
- Parziale-Esclusiva (PE)
- Parziale-Sovrapposta (PO)

Interpretazione insiemistica

Data una gerarchia di generalizzazione fra un'entità padre E e le entità figlie E1, E2, …, En

La gerarchia è totale se e solo se vale:

$$\bigcup_{i=1}^{n} E_i = E$$

La gerarchia è esclusiva se e solo se vale:

$$\bigcap_{i=1}^{n} E_i = \emptyset$$

Lezione 5 – L'algebra relazionale

L'**algebra relazionale** è il linguaggio procedurale per interrogare basi di dati relazionali. Il risultato di un'interrogazione è una relazione che può essere manipolata utilizzando gli operatori dell'algebra stessa.

Operatori dell'algebra relazione

- Gli operatori si suddividono in: *operatori insiemistici e operatori su relazioni*
- Operatori fondamentali: Unari(σ, π) e Binari($\cup$, $\times$, $-$)
- Operatori derivati: Binari($\bowtie$, $\bowtie_\theta$, $\cap$, $\div$)

Operatori insiemistici

- Due relazioni si dicono compatibili all'unione se sono dello stesso grado e sono definite sugli stessi attributi

Date due relazioni r e s compatibili all'unione, è possibile operare su esse con i seguenti operatori insiemistici:

- ***Unione***: l'unione $r \cup s$ è costituita dalle ennuple appartenenti a r o a s o ad entrambe
- ***Differenza***: la differenza $r - s$ è costituita dalle ennuple apparteneti a r e non a s
- ***Intersezione***: l'intersezione $r \cap s$ è costituita dalle ennuple appartenenti sia a r e sia a s

Compatibilità all'unione

$R(A_1, A_2, \dots, A_n)$ *e* $S(B_1, B_2, \dots, B_n)$ sono compatibili all'unione se:

1. R e S hanno lo stesso grado, cioè stesso numero di attributi
2. Domini corrispondenti degli attributi, gli attributi corrispondenti nelle due relazioni devono appartenere allo stesso dominio ($Dom(A_i) = Dom(B_i) \quad \forall i = 1, \dots, n$)

Operatori unari

Selezione (σ): Restituisce una relazione contenente l'insieme delle sole ennuple della relazione operando che soddisfano particolari condizioni, espresse mediante una formula proposizionale.

Proiezione (π): Restituisce come risultato una relazione contenente l'insieme delle ennuple della relazione operando limitate su particolari attributi.

Nella selezione si tratta di una **decomposizione orizzontale** in quanto va a selezionare alcune ennuple della relazione originale Nella proiezione si tratta di una **decomposizione verticale** in quanto va a selezionare alcuni insiemi di ennuple della relazione originale

Operatori binari

Prodotto cartesiano ($\times$)

Date due relazioni $r = \mathrm{R(X)}$ e $s = \mathrm{S(Y)}$ con $X \cap Y = \emptyset$ il prodotto cartesiano $r \times s$ è una nuova relazione definita sull'insieme di attributi XY contenente tutte le possibili combinazioni delle tuple di r e s.

Operatori Join

Theta-Join ($\bowtie_\theta$)

Date due relazioni $r = \mathrm{R(X)}$ e $s = \mathrm{S(Y)}$ con X e Y disgiunti, il Theta-Join $r \bowtie_\theta s$ è una relazione definita su XY composta dalle ennuple risultato della concatenazione di ennuple di r con ennuple di s che soddisfano le condizioni di confronto $A\theta B$ con $\mathrm{A} \in \mathrm{X}$ e $\mathrm{B} \in \mathrm{Y}$

- θ è una espressione composta per mezzo di operatori di confronto ($=, \neq, >, <, \geq, \leq$)
- Se il confronto è di uguaglianza, l'operazione è detta **Equijoin**
- Possiamo notare che il join $\mathrm{R} \bowtie_{A=B} \mathrm{S}$ è del tutto equivalente all'operazione $\sigma_{A=B}(\mathrm{R} \times \mathrm{S})$

Join naturale ($\bowtie$)

Date due relazioni r(YX) e s(XZ), il join naturale di r e s è l'operazione che combina le tuple delle due relazioni basandosi sull'uguaglianza dei valori dell'attributo X, il quale è comune ad entrambe.

Possiamo dare le seguenti definizioni:

- Due ennuple sono *joinabili* se è possibile effettuare l'operazione di join su di esse
- Una ennupla è detta *dangling* quando non contribuisce all'operazione di join
- Una operazione di join viene detta *completa* quando non contiene ennuple dangling
- Tramite una *ridenominazione*(ρ) possiamo rendere compatibili all'operazione di join due attribuiti uguali concentualmente ma con sintassi differente

Proprietà del join naturale

- Il join naturale è commutativo e associativo
- Può essere espresso in termini di:
  - Prodotto cartesiano
  - Selezione ennuple con valori uguali per attributi in comune
  - Proiezione di r e s eliminando duplicazioni
- Se r(X), s(Y) con X $\cap$ Y = $\emptyset$, allora $r \bowtie s = r \times s$
- Se r(X), s(Y) con X = Y, allora $r \bowtie s = r \cap s$

Outer Join

L'outer join è una variante del join che conserva anche le ennuple dangling (quelle che non trovano corrispondenza), inserendo valori NULL nei campi mancanti delle relazioni. Questo permette di evitare la perdita di dati che si verifica nei join naturali e theta join.

Tipologie di Outer Join:

1) Left outer join (Left Join)

- Mantiene tutte le ennuple della relazione di sinistra

2) Right Outer Join (Right Join)

- Mantiene tutte le ennuple della relazione di destra

3) Full Outer Join (Full Join)

- Mantiene tutte le ennuple di entrambi le relazioni

Divisione

La divisione si applica solo a relazioni r(Z) e s(X), dove X $\subseteq$ Z. La divisione va a restituire una relazione T(Y), dove Y = Z - X, contenente tutte le ennuple t tali che:

- Esistano ennuple tr in r con tr[Y] = t
- Per ogni ennupla ts in s, esista una ennupla tr in r con tr[X] = ts

Intuitivamente la divisione verifica una condizione o corrispondenza fra una o più ennuple di r e tutte le tuple di s.

Equivalenza di espressioni algebriche

Caratteristiche generali

-
-
-
-

regole di trasformazione

-
-
-
-
-

Lezione 6 – SQL

Introduzione

Le operazioni basilari per i database sono le DML, Data Manipulation Language, queste permettono di modificare i dati all'interno di una tabella.

L'operazione di **inserimento** di valori in una tabella di un database avviene tramite il seguente schema di comandi:

INSERT INTO *NomeTabella* (*ListaAttributi*) VALUES (*ListaDiValori*);

Un esempio pratico è il seguente:

```
INSERT INTO studenti (matricola, nome, età) VALUES (345216, 'Luigi', 21);
```

Da tenere presente che è anche possibile andare ad importare insiemi di righe di dati da altre tabelle del nostro database o anche in maniera estenstiva scrivendo le varie ennuple separate da parantesi rotonde all'interno del comando citato in precedenza.

L'operazione di **modifica** dei valori di una tabella di un database avviene tramite il seguente schema di comandi: UPDATE *NomeTabella* SET *Attributo* = (*Valore*)

Un esempio pratico è il seguente:

```
UPDATE studenti SET età = 20;
```

Prestiamo attenzione che il seguente comando lavora sull'intera tabella, se vogliamo andare a modificare in modo più selettivo dobbiamo fare l'utilizzo dell'opzione WHERE, per esempio:

```
UPDATE studenti SET età = 20 WHERE matricola = 345216;
```

L'operazione di **Cancellazione** di valori in una tabella di un database avviene tramite il seguente schema di comandi:

DELETE FROM *NomeTabella*

Un esempio pratico è il seguente:

```
DELETE FROM studenti;
```

Anche in questo caso il comando precedente lavora sull'intera tabella, quindi se vogliamo una cancellazione selettiva dobbiamo fare utilizzo del comando aggiuntivo WHERE, per esempio:

```
DELETE FROM studenti WHERE matricola = 345216;
```

Interrogazioni

Introduzione alle interrogazioni

Per effettuare le interrogazioni sui dati salvati nei nostri database avremo un linguaggio SQL dedicato detto DQL, Data Query Languange. il DML, come SQL, è un linguaggio dichiarativo cioè l'utente descrive l'obbiettivo dell'operazione ma non specifica i dettagli su come il database deve eseguire l'operazione. l'interprete del DBMS analizza l'istruzione della query e genera il corrispettivo codice in linguaggio procedurale, il tutto in modo nascosto dall'utente.

Il Query Optimizer è un modulo del DBMS che ottimizza la query trovando il modo più efficiente per eseguire l'interrogazione.

il linguaggio SQL può essere incluso anche in altri programmi/applicazioni, scritti anche con linguaggi differenti, per interagire con i nostri database.

L'istruzione SELECT

l'istruzione SELECT è il comando fondamentale del DQL, questo viene usato per recuperare dei dati da una o più tabelle.

La sintassi di base del SELECT è la seguente:

SELECT *ListaAttributi* FROM *ListaTabelle* (WHERE *Condizione*)

- SELECT: Indica gli attributi, colonne, dalle quali recuperare i dati cercati.
- FROM: Nome delle tabelle sulle quali la query verrà valutata.
- WHERE: Espressione booleana di ricerca che filtra le ennuple in base ad una condizione.

Esempi di utilizzo:

Selezione di una intera tabella:

```
SELECT * FROM clienti;
```

Selezione di colonne specifiche:

```
SELECT nome, cognome FROM clienti;
```

Selezione di una intera tabella o di colonne specifiche con filtrazione sui dati:

```
SELECT * FROM clienti WHERE città = 'roma';
SELECT nome, cognome FROM clienti WHERE città = 'roma';
```

La clausola WHERE

La clausola WHERE necessità di espressioni booleane costruite tramite l'utilizzo di operatori di confronto, che possono essere combinati tra di loro tramite gli operatori logici AND, OR, NOT.

Gli operatori di confronto basici disponibili in SQL sono i seguenti: $=, <>, <, >, <=, >=, IS NULL, IS NOT NULL. Inoltre sono presenti anche operatori di confronto avanzanti come possono essere BETWEEN, IN, EXISTS e LIKE.

Alcuni esempi di utilizzo:

```
SELECT * FROM clienti
WHERE Città = 'Roma';

SELECT nome, cognome FROM clienti
WHERE età > 18;
```

Se vogliamo andare a filtrare i risultati in base a pattern di testo possiamo fare uso dell'operatore LIKE, tramite "%" possiamo rappresentare una sequenza di caratteri, mentre tramite "_" un carattere singolo, per esempio:

```
SELECT nome, cognome FROM clienti
WHERE nome LIKE 'Mar%';
```

Questo esempio seleziona tutti i nomi che incominciano con "Mar".

```
SELECT nome, cognome FROM clienti
WHERE nome LIKE '_a___';
```

Questo esempio invece filtra a parole di 5 caratteri, dove la seconda lettera è il carattere 'a'

Possiamo andare ad eseguire la **ridenominazione degli attributi** ricavati dalla clausola WHERE, questo si effettua tramite l'operatore 'AS' che indicherà il nome della nuova colonna, contenente il dato cercato.

La ridenominazione permette di eseguire le operazioni matematiche di base (somma, moltiplicazione…) sui dati selezionati, ma anche operazioni più avanzate come l'arrotondamento, la potenza eccetera.

Vediamo un esempio di moltiplicazione e di potenza:

```
SELECT Nome, Età, Età * 12 AS Eta_in_Mesi
FROM Studenti
WHERE Età > 18;
```

```
SELECT Nome, Età, POWER(Età,2) AS EtaAlQuadrato
FROM Studenti
WHERE Età > 18;
```

Join

L'operatore Join ha la funzione di combinare righe da due o più tabelle sulla base di una condizione comune, solitamente tramite una chiave primaria ed una esterna.

I tipi principali di JOIN sono:

- Inner Join: Restituisce solo le righe che corrispondono in entrambe le tabelle
- Left Join: Tutte le righe della prima tabella, anche se non c'è corrispondenza
- Right Join: Tutte le righe della seconda tabella, anche se non c'è corrispondenza
- Full Join: Tutte le righe di entrambe le tabelle, combinate dove possibile

Esiste anche il natural join, nel quale non è specificata alcuna condizione, ma verrà effettuato un equijoin implicito di due colonne con lo stesso nome, condizione necessaria perchè funzioni, se non presenti è necessario effettuare una ridenominazione.

Vediamo degli esempi per ogni tipologia:

Inner Join:

```
SELECT m.id, m.official_title, imdb.rating.movie, imdb.rating.score
FROM imdb.movie m
INNER JOIN imdb.rating on m.id = rating.movie
WHERE imdb.rating.score > 7;
```

Left join:

```
SELECT m.id, m.official_title, imdb.rating.movie
FROM imdb.movie m
LEFT JOIN imdb.rating on m.id = rating.movie;
```

Right join:

```
SELECT m.id, m.official_title, imdb.rating.movie
FROM imdb.movie m
RIGHT JOIN imdb.rating on m.id = rating.movie;
```

Altre condizioni di selezione

Tramite il predicato **is null** possiamo selezionare le righe con valori nulli, esso può essere anche negato per non andarle a selezionare.

Tramite l'operatore **SELECT DISTINCT** possiamo andare ad eliminare eventuali righe uguali dal risultato di una interrogazione.

Tramite gli attributi *ASC* e *DESC* abbinati all'operatore **ORDER BY** possiamo eseguire l'ordinamento delle ennuple risultate da una interrogazione. teniamo presente che il valore NULL ha significato più altro, quindi in caso di ordinamento decrescente viene messo in cima.

Interrogazioni complesse

Operatori aggregati

Gli operatori aggregati permettono di calcolare valori riassuntivi su insiemi di righe, come somme, medie, conteggi eccetera. I principali operatori sono COUNT, SUM, AVG, MIN, MAX. l'operatore aggregato viene applicato al risultato di una interrogazione avvenuta tramite gli operati FROM e WHERE.

L'operatore COUNT

L'operazione COUNT (*) restituisce il numero di righe di una tabella specificata, COUNT('attributo') restituisce il numero di righe della tabella che possiedono valori diversi da null, a COUNT possiamo applicare l'attributo DISTINCT che conterà solamente i valori diversi all'interno di una tabella. vediamo alcuni esempi:

```
SELECT COUNT(*) AS TotaleStudenti
FROM Studenti;

SELECT COUNT(nome) AS TotaleStudenti
FROM Studenti;

SELECT COUNT(DISTINCT nome) AS TotaleStudenti
FROM Studenti;
```

Gli altri operatori

Gli operatori di MIN e MAX permettono di ottenere il valore minimo e massimo dei valori presenti in una determinata colonna, l'operatore SUM permette di sommare i vari valori della colonna specificata, mentre l'operatore AVG permette di calcolare la media dei valori della colonna specificata. vediamo degli esempi:

```
SELECT AVG(Età) AS EtaMedia
FROM Studenti;

SELECT SUM(Età) AS SommaEta
FROM Studenti;

SELECT MIN(Età) AS EtaMinima, MAX(Età) AS EtaMassima
FROM Studenti;
```

Il raggruppamento

Il raggruppamento ci permette di appplicare gli operatori aggregati a sottogruppi di tuple di una tabella in base al valore degli attributi, la sintassi base è la seguente:

SELECT colonna1, funzione_aggregata(colonna2) FROM tabella GROUP BY colonna1;

Vediamo un esempio in merito:

```
    SELECT Città, AVG(Età) AS EtaMedia
    FROM Studenti
    GROUP BY Città;
```

Teniamo a mente che ogni colonna nel SELECT deve essere una colonna nel GROUP BY oppure il risultato di funzione aggregata. Inoltre il GROUP BY viene sempre prima di HAVING e dopo WHERE, se presenti.

Ricordiamo che la clausola WHERE filtra le righe prima del raggruppamento, mentre la clausa HAVING che filtra dopo il raggruppamento, per esempio:

```
    SELECT Città, COUNT(*) AS NumeroStudenti
    FROM Studenti
    GROUP BY Città
    HAVING COUNT(*) > 1;
```

Interrogazioni di tipo insiemistico

Le operazioni insiemistiche presenti in SQL sono UNION, INTERSECT e EXCEPT, di default eseguono l'eliminazione dei duplicati, se vogliamo tenere questi ultimi dobbiamo specificarlo con l'aggiunta di ALL.

Vediamo degli esempi per ogni tipo di operazione:

```
Unione tra studenti non laureati e laureati
    SELECT Nome FROM Studenti
    UNION
    SELECT Nome FROM Laureati;

Selezione di studenti laureati
    SELECT Nome FROM Studenti
    INTERSECT
    SELECT Nome FROM Laureati;

Selezione di studenti non laureati
    SELECT Nome FROM Studenti
    EXCEPT
    SELECT Nome FROM Laureati;
```

Interrogazioni nidificate

Una interrogazione nidificata consiste in una query che al suo interno possiede una subquery, essa è una query interna scritta tra parentesi, che viene eseguita prima della query esterna e i cui risultati vengono usati da quest'ultima.

Esistono tre tipologie di subquery:

- subquery scalare: restituisce un solo valore
- subquery che restituisce una colonna: spesso usata con IN, ANY, ALL

- subquery correlata: fa riferimento a una colonna della query esterna

vediamo degli esempi, uno per tipologia:

```
Subquery scalare; trova il valore della popolazione con il PIL più alto

    SELECT value AS popolazione
    FROM Measure
    WHERE label = 'Population'
        AND country = (
          SELECT country
          FROM Measure
          WHERE label = 'GDP'
          ORDER BY value DESC
          LIMIT 1);

Subquery normale; trova il nome dei paesi monarchici che hanno dati di popolazione

    SELECT name
    FROM Country
    WHERE iso3 IN (
      SELECT country
      FROM Measure
      WHERE label = 'Population'
    )
      AND government LIKE '%monarchy%';

Subquery correlata; mostra i paesi e la loro popolazione solo se superiore
alla media globale

    SELECT name, (
        SELECT value
        FROM Measure
        WHERE label = 'Population' AND country = c.iso3
    ) AS popolazione
    FROM Country c
    WHERE (
        SELECT value
        FROM Measure
        WHERE label = 'Population' AND country = c.iso3
    ) > (
        SELECT AVG(value)
        FROM Measure
        WHERE label = 'Population');
```

Predicato EXISTS

Il predicato EXISTS(subquery) restituisce TRUE se la subquery restituisce almeno una tupla; altrimenti restituisce FALSE.

In SQL esiste anche NOT EXISTS che funziona al contrario.

vediamo un esempio:

```
Trovare le nazioni con lo stesso nome in lingue diverse
    SELECT country, gazeteer, script FROM name N

    WHERE EXISTS (
        SELECT * FROM name X
        WHERE N.gazeteer = X.gazeteer AND N.script <> X.script );
```

divisione

Le subquery correlate e l'operatore di NOT EXISTS sono necessari per implementare l'operazione di divisione in SQL.

La specifica della divisione in SQL fa uso dell'operatore NOT EXISTS e richiede di ragionare in base al concetto di controesempio; una persona A verifica l'interrogazione se non esiste una repubblica che A non abbia visitato, ovvero data una tupla A di persona non deve esistere una tupla C di country tale che C sia una repubblica e che non esista una relazione TRAVEL fra A e C.

vediamo un esempio SQL:

```
SELECT code, name, surname FROM person A
WHERE NOT EXISTS (
    SELECT * FROM country C
    WHERE government LIKE '%republic%'
    AND NOT EXISTS (
        SELECT * FROM travel T
        WHERE T.person=A.code AND T.country=C.iso3
));
```

Piano di esecuzione delle interrogazioni

Explain

Le interrogazioni SQL sono trasformare in una successione di operazioni atomiche sui dati denominata **piano di esecuzione**, è possibile accedere al piano di esecuzione attraverso la clausola EXPLAIN.

Vediamo un esempio in merito:

```
    EXPLAIN
    SELECT name
    FROM Country
    WHERE iso3 IN (
        SELECT country
        FROM Measure
        WHERE label = 'Population'
          AND value > (
        SELECT AVG(value)
        FROM Measure
        WHERE label = 'Population'
    )
);
```

Interrogazioni avanzate TO COMPLETE

La clausola WITH

Una Common Table Expression (CTE) è una tabella temporanea che esiste e può essere utilizzata solo nell'ambito di una interrogazione, le CTE sono realizzate attraverso la clausola WITH, tramite la seguente sintassi:

WITH [RECURSIVE] with query [, …]
SELECT …

In cui la with query 'e:
with query name [(column name [, …])] AS (SELECT …)

Vediamo il seguente esempio:

Interrogazioni ricorsive

In generale la clausola WITH è prevalentemente una facilitazione sintattica per interrogazioni che sarebbe comunque possibili attraverso query nidificate, correlate e viste. Aggiungendo la clausola RECURSIVE diventà però possibile usare WITH per interrogazioni ricorsive altrimenti impossibili in SQL.

Vediamo un esempio astratto che genera ricorsivamente numeri:

```
WITH RECURSIVE t(n) AS (
SELECT 1
UNION ALL
SELECT n+1 FROM t WHERE n < 100
)
SELECT n FROM t;
```

Una query ricorsiva è sempre composta da un termine di passo (non ricorsivo) seguito da UNION (ALL) e da un termine ricorsivo. Solo il termine ricorsivo può contenere un riferimento all'output della query.

La differenza tra UNION e UNION ALL è che il primo scarta i duplicati, mentre il secondo no.

L'esecuzione di una query ricorsiva inizia dal passo base, una volta eseguito questo si inizia l'esecuzione del termine ricorsivo finchè la condizione di chiusura non è soddisfatta. Le query ricorsive sono spesso utilizzare per interrogare tabelle che rappresentano relazioni gerarchiche o relazioni di connessioni fra oggetti.

vediamo degli esempi di interrogazioni ricorsive:

Viste TO COMPLETE

Uso di viste in interrogazioni SQL

Una vista in SQL è una tabella derivata a partire da altre tabelle, essa è una tabella virtuale, in quanto non ci sono tuple istanza della vista memorizzate nella base di dati. Anche se virtuale, può essere utilizzata nella formulazione di interrogazioni come se fosse memorizzata. Da tenere presente che possono anche esisistere viste materializzate e quindi salvate nella base di dati.

Generazione di viste

```
CREATE VIEW NomeVista [(ListaAttributi)] AS SelectSQL
[WITH [LOCAL | CASCADED] CHECK OPTION]
```

In generale le viste sono utili per:

- specificare una nuova tabella che frequentemente viene utilizzata nelle interrogazioni, anche se non esiste nella base di dati.
- specificare una tabella contenente un sottoinsieme dei dati di una o più tabella nella base di dati, al fine di restringere l'accesso al solo sottoinsieme contenuto nella vista (sicurezza)

Viste in SQL

Vediamo un esempio di implementazione di una vista in SQL:

Aggiornamento di viste

Le operazioni di aggiornamento sulle viste sono tradotte in opportuni comandi di modifica sulle tabelle di base da cui la vista è derivata. Non sempre è possibile determinare in modo univoco come riportare la modifica sulla vista in termini di modifiche sulle tabelle di base; i problemi sorgono con viste ottenute tramite join di più tabelle.
I sistemi considerano aggiornabili viste definite su una sola tabella, in alcuni casi si richiede che la vista contenga la chiave primaria.

- CHECK OPTION: si applica a viste aggiornabili
- Sono ammessi aggiornamenti sulle righe della vista e dopo l'aggiornamento le righe devono continuare ad appartenere alla vista.
- LOCAL vs CASCADE: profondità con cui applicare i controlli a seguito di aggiornamento

Trigger e basi di dati attive TO DO

Asserzioni

In SQL un ASSERTION consente di specificare un ulteriore vincolo sulla base di dati che non è altrimenti rappresentabile dallo schema.

Sintassi:

```
CREATE ASSERTION <Constraint name>
CHECK (search condition)
[ <constraint attributes> ]
```

Un'asserzione può avere come attributo DEFERRABLE o NOT DEFERRABLE, il primo significa che il DBMS può posticipare il controllo del vincolo fino al termine della transazione, altrimenti controlla il vincolo immediatamente dopo l'esecuzione di ogni SQL statement in una transazione.

Un'asserzione può essere INITIALLY DEFFERED O INITIALLY IMMEDIATE, la prima significa che il vincolo è necessariemente DEFERRABLE e controllato all'inizio di ogni transazione, la seconda significa che il vincolo può essere sia DEFERRABLE o NOT DEFERRABLE e il tempo di controllo è immediato all'inizio di ogni transazione.

vediamo degli esempi:

Trigger

I trigger sono regole attive, essi sono utili per formare basi di dati attive, cioè che hanno un comportamento reattivo, sono in grado di eseguire delle regole oltre che alle transazioni utente.

Lezione 7 - Progettazione fisica

Lezione 8 - Elementi di sicurezza

Lezione 9 - Cenni su transazioni