Basi di dati

Elia Ronchetti
@ulerich

Marzo 2022

Indice

1 Introduzione - Che cos'è un DB e un DBMS **6**
1.1 Perchè creare un Database 6
1.2 Base di Dati - DB - Data Base 7
1.2.1 Modello dei dati 9
1.3 Schemi e Istanze 9
1.4 Modelli concettuali 9
1.5 Modelli logici - Modello Relazionale 10
1.5.1 Modello Relazionale 10
1.5.2 Linguaggi per basi di dati 10
1.5.3 Creazione di un database 10
1.6 Vantaggi e svantaggi dei DBMS 11

2 Modello Entity Relationship - ER **12**
2.1 Fasi del ciclo di vita 12
2.2 La progettazione di basi di dati 13
2.2.1 Progettazione concettuale 13
2.2.2 Vantaggi della progettazione concettuale 14
2.3 Modello Entità Relazione 14
2.4 I Costrutti del modello ER 14
2.4.1 Entità 15
2.5 Scelta tra entità e attributo 17
2.6 Scelta tra entità e relazione 17
2.7 Cardinalità nelle relazioni 18
2.8 Identificatore di un'entità 18
2.8.1 Esempi di identificatori esterni 19
2.9 Relazione IS-A tra entità 20
2.10 Generalizzazione tra Entità 20
2.11 Svolgimento Esercizi 21

3 Modello Relazionale **22**
3.1 Introduzione al modello relazionale e cenni storici 22
3.1.1 I modelli logici dei dati 22
3.1.2 Il modello relazionale 23
3.1.3 Il termine relazione in 3 accezioni 23
3.2 Modello relazionale - definizione formale 23

4 Progettazione Concettuale **25**
4.1 Reificazione 25
4.2 Stretegie di progetto 25
4.3 Strategia utilizzata in pratica - Mista 25
4.4 Qualità di uno schema concettuale 25
4.5 Consigli per lo svolgimento di esercizi 26

5 Preparazione Primo Parziale **27**
5.1 Schema ER 27
5.2 Modello Relazionale 28

6 Algebra Relazionale **30**
6.1 Operatori dell'algebra relazionale 30
6.2 Operatori insiemistici 31
6.3 Operatorri unari 31
6.3.1 Operatore di ridenominazione 31
6.3.2 Selezione 32
6.4 Proiezione 33
6.5 Join 35
6.6 Interrogazioni in algebra relazionale 38
6.7 Le viste 39
6.8 Plus teorici 39
6.8.1 Rappresentazione delle espressioni tramite alberi 39
6.8.2 Equivalenza di espressioni 39
6.9 Regole base equivalenza 39
6.10 Riassunto simboli 40

7 Progettazione Logica **41**
7.1 Ristrutturazione dello schema ER 42
7.1.1 Carico applicativo 43
7.1.2 Tavola degli accessi 43
7.1.3 Tavola dei volumi 44
7.1.4 Analisi delle ridondanze 44
7.2 Eliminazione gerachie 45

7.2.1 Accorpamento delle figlie nel genitore 45
7.2.2 Accorpamento del genitore nelle figlie 45
7.2.3 Sostituzione della generalizzazione con relazioni 46
7.2.4 Note pratiche 47
7.3 Partizionamento/accorpamento di entità e relationship 47
7.3.1 Accorpamento di entità 48
7.4 Eliminazione di attributi multivalore 48
7.5 Scelta degli identificatori principali 48

8 SQL - Structured Query Language 49
8.1 Confronto con Algebra Relazionale e Istruzioni principali . . . 50
8.1.1 SQL e Algebra Relazionale 50
8.1.2 Istruzioni principali DDL 50
8.1.3 Istruzioni principali DML 51
8.1.4 SQL è dichiarativo 51
8.1.5 Notazione SQL 51
8.1.6 Primo esempio di Query 52
8.2 SQL-DDL 52
8.2.1 Tabelle 52
8.2.2 Definizione dei Dati: I Domini 53
8.2.3 Il tipo Bit 53
8.2.4 Carattere 53
8.2.5 Numerici Esatti 54
8.2.6 Numerici Approssimati 54
8.2.7 Data e Ora 54
8.2.8 Intervalli temporali 55
8.2.9 CLOB e BLOB 55
8.2.10 Domini definiti dall'utente 55
8.2.11 Valori di Deafult 55
8.2.12 Il valore NULL 55
8.2.13 Vincoli di Integrità 56
8.2.14 Chiave 56
8.2.15 Aggiornamenti e Violazioni 58
8.3 Modifiche degli schemi 59
8.3.1 ALTER 59
8.3.2 Cataloghi relazionali 60
8.4 Interrogazioni SQL - Query 61
8.4.1 Istruzione AS 61
8.4.2 Asterisco 62
8.4.3 FROM clausola 62
8.4.4 WHERE 62

8.4.5 WHERE - Operatore Like 63
8.4.6 WHERE - Operatore BETWEEN 63
8.4.7 WHERE - Operatore IN 63
8.4.8 WHERE - Valori nulli 64
8.4.9 SELECT - DISTINCT 65
8.4.10 Espressioni e Funzioni 65
8.5 Operatore JOIN 66
8.5.1 JOIN implicito 67
8.5.2 JOIN esplicito 67
8.5.3 OUTER JOIN 68
8.5.4 Self JOIN 68
8.6 Operatori di ordinamento e aggregazione 69
8.6.1 Ordinamento del risultato: ORDER BY 69
8.6.2 Operatori aggregazione 69
8.6.3 COUNT 70
8.6.4 Altri operatori di aggregazione 71
8.6.5 Operatori di aggregazione e Target List 71
8.6.6 GROUP BY 72
8.6.7 Raggruppamenti e Target List 74
8.6.8 Condizioni sui Gruppi 74
8.6.9 Osservazioni 75
8.7 Interrogazioni di tipo insiemistico 76
8.7.1 UNION 76
8.7.2 INTERSECT, EXCEPT, MINUS 76
8.8 Interrogazioni nidificate 77
8.8.1 Operatori di confronto 78
8.8.2 MAX e MIN in subquery 78
8.8.3 Subquery Correlate 79
8.8.4 EXISTS e NOT EXISTS 80
8.8.5 Note finali subquery 80
8.9 Viste in SQL 81
8.9.1 Creazione di una vista 81
8.9.2 Uso di espressioni e di funzioni 82

Capitolo 1

Introduzione - Che cos'è un DB e un DBMS

Che cos'è un Data Base Una collezione di dati utilizzati per rappresentare le informazioni di interesse di un sistema informativo

Che cos'è un DBMS? Un DBMS (Data Base Management System) è un insieme di programmi che permettono di creare, usare e gestire una base di dati, è quindi un software general purpose che facilita il processo di definizione, costruzione e manipolazione del database per varie applicazioni.

1.1 Perchè creare un Database

Un soggetto, come per esempio un'azienda, ha molti dati da manipolare

- Persone
- Denaro
- Materiali
- Informazioni

Per gestire questi dati è necessario un sistema che li organizzi e li gestisca in modo efficiente e sicuro. Questo sistema è detto **Sistema Informativo**, cioè un componente di una organizzazione che gestisce le informazioni di interesse, con i seguenti scopi:

- Acquisizione/Memorizzazione
- Aggiornamento

- Interrogazione
- Elaborazione

Il **Sistema Informatico** è invece la porzione automatizzata del Sistema informativo, la parte quindi che gestisce informazioni tramite tecnologia informatica.
Il Sistema Informatico ha i seguenti obiettivi:

- Garantisce che i dati siano conservati in modo permanente sui dispositivi di memorizzazione
- Permette un rapido Aggiornamento dei dati
- Rende i dati accessibili alle interrogazoni degli utenti
- Può essere distribuito sul territorio

Gestione delle informazioni Nei sistemi informatici le informazoni vengono rappresentate in modo essenziale attraverso i dati. I Dati hanno bisogno di essere interpretati, ma costituiscono una precisa rappresentazione di forme più ricche di informazioni e conoscenza, inoltre sono più stabili nel tempo rispetto ad altre componenti (come processi, tecnologie, ruoli umani) e restano gli stessi nella migrazione da un sistema al successivo.

1.2 Base di Dati - DB - Data Base

Data Base - DB Collezione di dati utilizzati per rappresentare le informazioni di interesse di un sistema informativo

Altra definizione di DB Insieme di archivi in cui ogni dato è rappresentato logicamente una sola volta e può essere utilizzato da un insieme di applicazioni da diversi utenti secondo opportuni criteri di riservatezza.

Data Base Management System - DBMS Sistema software capace di gestire collezioni di dati che siano grandi, condivise e persistenti, assicurando la loro affidabilità e privatezza.

Elenco caratteristiche DBMS Sistema che garantisce collezioni di dati:

- grandi
- persistenti
- condivise

Garantendo:

- Privatezza - Meccanismi di autorizzazione (come ACL)
- Affidabilità - Resistenza malfunzionamenti hardware e software (tramite tecniche come la gestione delle transazioni)
- Efficienza
- Efficacia

Transazione → Insieme di operazioni da considerare indivisibile (atomico), la sequenza di operazioni sulla base di dati viene eseguita per intero o per niente.
L'effetto di transazioni concorrenti deve essere coerente (ad esempio "equivalente" all'esecuzione separata).
I risultati delle transizioni sono permanenti, la conclusione di una transazione corrisponde a un impegno (in inglese commitment) a mantenere traccia del risultato in modo definitivo.
I DBMS devono essere efficienti cercando di utilizzare al meglio le risorse di spazio di memoria e tempo.
Efficacia intesa come resa produttiva delle attività dei loro utilizzatori.

Caratteristiche di un DB

- Ridondanza minima e controllata
- Consistenza delle informazioni
- Dati disponibili per utenze diverse e concorrenti
- Dati controllati e protetti (da malfunzionamenti hardware e software)
- Indipendenza dei dati dal programma

Riassumendo, un DMSB è un prodotto sfotware in grado di gestire collezoni di dati che siano:

- Grandi
- Persistenti
- Condivise

E che garantiscano

- Affidabilità
- Privatezza
- Efficienza

I DBMS permettono inoltre ai dati di essere indipendenti dalla propria rappresentazione fisica.

1.2.1 Modello dei dati

Insieme di costrutti per organizzare i dati di interesse e descriverne la dinamica. Sono componenti fondamentali che permettono la strutturazione dei dati. Per esempio il modello relazionale prevede il costruttore relazione, che permette di definire insiemi di recordo omogenei.

1.3 Schemi e Istanze

In ogni base di dati esistono:

- Lo schema, sostanzialmente invariante nel tempo, che ne descrive la struttura, il significato (aspetto intensionale). Costituisce quindi la parte astratta delle proprietà.
- L'istanza, che sono i valori attuali e possono cambiare anche molto rapidamente (aspetto estensionale). Costituisce quindi l'aspetto concreto che varia nel tempo.

1.4 Modelli concettuali

Permetton odi rappresentare i dati in modo indipendente da ogni sistema cercando di descrivere i concetti dle modno reale. Sono utilizzati nelle fasi preliminari di progettazione. Il più diffuso è il modello **Entity-Relationship ER**.

1.5 Modelli logici - Modello Relazionale

Sono i modelli adottati nei DBMS esistenti per l'organizzazione dei dati e sono utilizzati dai programmi, sono indipendenti dalle strutture fisiche. L'esempio più diffuso e che noi tratteremo e quello del **modello Relazionale**.

1.5.1 Modello Relazionale

I dati vengono strutturati in tabelle, in particolare un DBMS relazione può essere pensato come un insieme di tabelle, dove ogni tabella mantiene informazioni di tipo omogeneo. Diverse tabelle sono collegate (in relazione) fra loro grazie alla presenza di un campo comune che permette di mettere in relazione i dati delle due tabelle.
In questo caso lo schema è la componente intensionale e descrive la struttura della tabella (ed è stabile nel tempo)
Mentre il'istanza è la componente estensionale e descrive i valori attuali, cioè i dati (ed è variabile nel tempo).

1.5.2 Linguaggi per basi di dati

Ci sono i DDL (Data Definition Language) che permettono di definire il DB. Mentre i DML (Data Manipulation Language) permettono di manipolare i dati, interrogando e aggiornando delle basi di dati. Alcuni linguaggi, come SQL (Structured Query Language) hanno funzioni di entrambe le categorie.

1.5.3 Creazione di un database

Le tre fasi

- Definizione - DDL
- Creazione/Popolazione - DDL
- Manipolazione - DML

Query É fondamentale poter interrogare un DB, attraverso per esempio delle **query**. L'efficacia della query dipende da:

- Conoscenza del contenuto del DB
- Esperienza del linguaggio di interrogazione

1.6 Vantaggi e svantaggi dei DBMS

Pro

- Permettono di considerare i dati come risorsa comune di un'organizzazione, a disposizione di molteplici applicazioni e utenti
- Offrono modello della parte di mondo di interesse che è unificato e preciso, utilizzabile in applicazioni attuali e future
- Controllo centralizzato dei dati, riduce ridondanze e incosistenze
- Indipendenza dei dati: favorisce sviluppi di applicazioni flessibili e facilmente modificabili

Contro

- Costosi, complessi, hanno specifici requisiti in termini di software e hardware
- Difficile separare, tra tutti i servizi offerti da un DBMS, quelli effettivamente utilizzati da quelli inutili
- Inadatti alla gestione di applicazioni con pochi utenti

Capitolo 2

Modello Entity Relationship - ER

In questa parte si studierà la come progettare una base di dati a livello concettuale e logico, partendo dai requisiti di utente. Per capirne l'importanza è utile analizzare il ciclo di vita di un sistema informativo

2.1 Fasi del ciclo di vita

- Studio di fattibilità: definizione costi e priorità
- Raccolta e analisi dei requisiti: studio delle proprietà del sistema
- Progettazione: di dati e funzioni
- Implementazione: realizzazione
- Validazione e collaudo: sperimentazione
- Funzionamento: il sistema diventa operativo

Il ciclo di vita segue un modello a spirale. Per garantire prodotti di buona qualità è fondamentale seguire una metodologia di progetto.

Metodologia è un'articolazione in fasi/passi di guida ad una attività di progettazione. Avere una metodologia di progetto:

- Permette di suddividere la progettazione in fasi
- Fornisce una strategia da seguire

- Fornisce modelli di riferimento (linguaggi) per descrivere la realtà che stiamo progettando

Serve per garantire:

- Generalità rispetto ai problemi da affrontare
- Qualità in termini di correttezza, completezza ed efficienza
- Facilità d'uso

La metodologia di basa su un principio semplice ma efficace:

Separazione netta tra decisioni relative a:

- Cosa rappresentare
- Come farlo

2.2 La progettazione di basi di dati

La progettazione si divide in 3 fasi:

- Progettazione concettuale
- Progettazione logica
- Progettazione fisica

Ognuna delle fasi si basa su un modello, che permette di generare una rappresentazione formale (schema) della base di dati ad un dato livello di astrazione (concettuale, logico, fisico).

2.2.1 Progettazione concettuale

Traduce i requisiti del sistema informatico in una descrizione formalizzata, integrata delle esigenze aziendali, espressa in modo **indipendente** dalle scelte implementative.

- Formale - Espressa con un linguaggio non ambiguo e capace di descrivere il sistema analizzato
- Integrata - Deve essere in grado di descrivere nella globalità l'ambiente analizzato
- Indipendete dall'ambiente tecnologico

Nel nostro caso:

- Schema concettuale - **Modello ER**
- Schema logico - **Modello relazionale**

2.2.2 Vantaggi della progettazione concettuale

Permette una descrizione dei dati indipendente dagli aspetti tecnologici con un livello di astrazione intermedio fra utente e sistema. Prevale l'aspetto intensionale.
Si tratta di una rappresentazione prevalentemente grafica. Utile per la documentazione.

2.3 Modello Entità Relazione

Il modello ER è un modello grafico semi-formale per la rappresentazione di schemi concettuali. Si è ormai affermato come standard nelle metodologie di progetto e nei sistemi Software di ausilio alla progettazione.

2.4 I Costrutti del modello ER

- Entità
- Relazione
- Attributo semplice
- Atrributo composto
- Cardinalità
- Cardinalità di un Attributo
- Identificatore interno
- Identificatore esterno
- Generalizzazione
- Sottoinsieme

2.4.1 Entità

Classe di oggetti (fatti, persone, cose) della applicazione di interesse con proprità comuni e con esistenza autonoma e della quale si vogliono registrare fatti specifici.

Definita come sostantivo al singolare (es. studente, classe, docente, ecc.) A livello estensionale un'entità è costituira da un insieme di oggetti che sono chiamati le sue istanze.

Istanza Occorrenza di un'entità, è l'oggetto della classe che entità rappresenta. Nello schema concettuale rappresentiamo le entità, non le singole istanze.
Riassumendo:

- Conoscenza Astratta → Entità
- Conoscenza Concreta → Istanza di entità

Attributi

Un attributo di un entità è una proprietà locale di un'entità di interesse ai fini dell'applicazione. Associa ad ogni istanza di un'entità un valore appartenente a un insieme detto dominio dell'attributo (es. int, string, char, ecc.).
Viene definito quando si vuole rappresentare una proprietà locale delle istanze dell'entità E.
Una proprietà di un oggetto si dice 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.

Attributi composti Si ottengono raggruppando attributi di una medesima entità o relazione che presentano affinità nel loro significato o uso.

Esempio: Indirizzo è composto da Via, Numero, Cap.

Graficamente Sono rappresentati come dei collegamenti con un pallino vuoto.

Relazione-Associazione

Ogni relazione ha un nome che la identifica univocamente nello schema.

Convenzioni

- Singolare
- Sostantivi invece che verbi

A livello estensionale una relazione R tra le entità E ed F è costituita da un insieme di coppie (x, y) tali che x è una istanza di E, ed y è un'istanza di F. Ogni coppia è detta istanza della relazione R.
Ciò significa che se in uno schema S è definita una relazione R sulle entità E ed F

Istanze di associazione Combinazione (aggregazione) di istanze di entità che prendono parte alla associazione.

Esempio: Rossi insegna Basi di Dati

Osservazione importante

Dalla semantica delle relazioni segue immediatamente che non possono esistere due istanze della stessa relazione che coinvolgono le stesse istanze di entità.
Due entità possono essere coinvolte in più relationship.
Le relationship possono coinvolgere più di due entità.

Osservazione sul concetto di relazione Il concetto di relazione sarà spiegato meglio nel capitolo 2-Modello-Relazionale. Una relazione può coinvolgere due o più volte la stessa entità, queste sono dette **associazioni ad anello**. In questi casi è fondamentale definire la specifica dei ruoli, altrimenti non si riesce a capire l'ordine della relazione (esempio del sovrano o del confronto tra i prof. slide 70-78, Link).

2.5 Scelta tra entità e attributo

Un dubbio classico nella risoluzione di questi esercizi è proprio la scelta tra entità e attributo.

Scelgo Entità quando:

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

Scelgo Attributo quando:

- le sue istanze non sono concettualmente significative
- non ha senso considerare una sua istanza indipendentemente dalle altre
- se serve solo a rappresentare un una proprietà locale di un altro concetto

2.6 Scelta tra entità e relazione

Dubbio ancora più classico è la scelta tra entità e relazione.
In linea generale quando è necessario modellare un concetto, perchè esiste a prescindere dalle altre istanze o relazioni allora scelgo entità (es. slide 98). Oltretutto devo considerare che una relazione esiste in funzione delle sue entità, viene identificate da una specifica tupla, quindi se ho un caso come Studente $\rightarrow$ Esame $\leftarrow$ Corso, dove è possibile che una persona possa svolgere più volte un esame, questo schema risulta errato dato che la tupla Studente Corso identifica l'esame e non può identificare più esami svolti dalla stessa persona, questo schema quindi impedisce a una persona di dare più esame, ma come purtroppo sappiamo questo può accadere.

2.7 Cardinalità nelle relazioni

É importante definire il numero minimo e massimo di occorrenze delle relazioni cui ciascuna occorrenza di una entità può partecipare, questo possiamo farlo tramite la **cardinalità**.
La cardinalità è una coppia di valori che si associa a ogni entità che partecipa a una relazione.

- 0,1 è la cardinalità minima
  - 0 = partecipazione opzionale
  - 1 = partezipazione obbligatoria
- 1 e N per la massima - N non pone alcun limite

Esempio: Slide 105, cardinalità Residenza dove una città ha più residenze, mentre uno studente ne ha una sola.
Per quanto riguarda le cardinalità massime, abbiamo relazioni

- 1,1 se la cardinalità massima di entrambe le entità è 1
- Si può avere 1 a molti come nell'esempio della residenza
- Oppure si può avere molti a molti (slide 108 riporta esempi)

Praticità della cardinalità

A livello pratico la cardinalità esprime un limite minimo (cardinalità minima) e massimo (cardinalità massima) di istanze della relazione R a cui può partecipare ogni istanza dell'entità E. Serve a caratterizzare meglio il significato di una relazione.

Attributi e cardinalità Si può assegnare cardinalità anche agli attributi per indicare opzionalità o attributi multivalore.

2.8 Identificatore di un'entità

Super importante è definire un identificatore per ogni entità, necessario per identificare univocamente le occorrenze di un'entità.

- **Identificatore interno** - se costituito da attributi dell'entità
- **Identificatore esterno** - attributi + entità esterne attraverso relationship

Notazione identificatori

Per gli identificatori interni:

- Se l'identificatore è costituito da un solo attributo, si annerisce il corrispondente pallino
- Se l'identificatore è costituito da più attributi, si uniscono gli attributi con una linea che termina con un pallino annerito

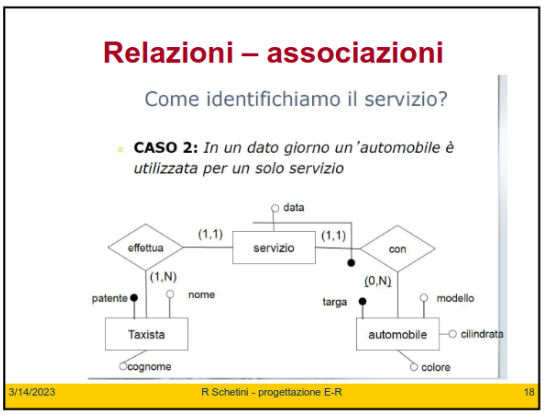
Per gli identificatori esterni:

- Se l'identificatore è formato da attributi e relazioni (o meglio ruoli) si indica unendo gli attributi ed i ruoli con una linea che termina con un pallino annerito

Osservazioni

- Ogni entità deve possedere almeno un identificatore, ma può averne in generale più di uno.
- **Una identificazione esterna è possibile solo attraverso una relationship a cui l'entità da identificare partecipa con cardinalità (1,1)**
- In questo corso **NON si utilizzano identificatori delle relationship**

2.8.1 Esempi di identificatori esterni



Qui invece vediamo la differenza dell'utilizzo di identificatori esterni in base al testo in entrambi i casi si fa riferimento alla data, ma basta cambiare di poco il testo per cambiare l'identificatore esterno. É fondamenale prestare attenzione a frasi come "unico nell'ambito di ..." oppure "identificato con un codice univoco all'interno del ...", tutte frasi che ci fanno capire che la chiave non è tale indipendentemente da tutto, ma è chiave all'interno di un contesto e quindi è chiave se c'è anche l'entità da cui dipende, per questo è un identificatore esterno.
Può anche darsi che l'identificatore esterno dipenda da un fattore esterno allo schema ER, per esempio alla data.

Da ricordare Lo ripeto perchè a volte sfugge ed è un errore grave, **una identificazione esterna è possibile sono attraverso una relationship a cui l'entità da identificare partecipa con CARDINALITÁ (1,1)**.

2.9 Relazione IS-A tra entità

É una relazione di sottoinsieme di un entità, si può definire come entità-padre ed entità figlia, o sottoentità, cioè quella che rappresenta un sottoinsieme della entità padre.

Esempio Persona → Studente, dove Studente è una sottoentità di Persona. Si dice che Studente è in relazione ISA con Persona o in alternativa che Studente ISA Persona. Si tratta di un sottoinsieme specifico di quell'entità, esso eredita tutte le proprietà del padre, quindi i suoi attributi (non vengono riportati nel figlio, ma sono presenti), ciò non toglie che il figlio possa avere attributi aggiuntivi.

Ereditarietà Tutte le proprietà (attributi, relationship, altre generalizzaizoni) dell'entità genitore vengono ereditate dalle entità figlie e non rappresentate esplicitamente.

2.10 Generalizzazione tra Entità

Nella relazione ISA l'entità padre è più generale della sottoentità. Talvolta 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.
Una generalizzazione può essere di due tipi:

- Completa - L'unione delle istanze delle sottoentità è uguale all'insieme delle istanze dell'entità padre
- Non completa

Graficamente La generalizzazione si indica collegando mediante un arco le sottoentità e collegando con una freccia tale arco alle entità padre. La freccia è annerita solo se la generalizzazione è completa.

Esempio Persona → Uomo/Donna (generalizzazione completa)

Esempio Persona → Studente/Docente (generalizzazione non completa)

Regola importante Vige la regola che una entità può avere al massimo una entità padre. In altre parola, il modello ER **NON ammette ereditarietà multipla**. Sia per quanto riguarda le ISA e le generalizzazioni.
Mentre la stessa entità può essere padre di diverse generalizzazioni.

Ereditarietà Anche in questo caso vale il principio di ereditarietà.

Differenze tra due IS-A e una generalizzazione

Le due sottoclassi della generalizzazione derivano da uno stesso criterio di classificazione della superclasse, mentre per quanto riguarda la relazione IS-A le due sottoentità sono indipendenti.

2.11 Svolgimento Esercizi

Risulta fondamentale svolgere gli esercizi da casa per allenare la mente a convertire il testo in schema ER (come sarà all'esame), seguire gli esempi e basta non è sufficiente, dato che ci saranno diversi dubbi e difficoltà che si faranno strada nella vostra mente solo se vi metterete a fare gli esercizi per conto vostro (le esercitazioni sono molto utili).

Capitolo 3

Modello Relazionale

Un modello dei dati è un insieme di concetti per organizzare i dati e descriverne la struttura. Componente fondamentale di ogni modello sono i meccanismi di strutturazione (analogi ai costruttori di tipo).
Ogni modello dei dati prevede alcuni costruttori che permettono di definire nuovi tipi sull abase di tipi predefiniti (elementari).
Il **modello relazionale** è il modello di dati più diffuso.

3.1 Introduzione al modello relazionale e cenni storici

Il modello relazionale permette di definire tipi per mezzo del costruttore **relazione** che permette di organizzare i dati in insiemi di recordo a **struttura fissa**.
Una **relazione** è spesso rappresentata da una tabella, dove:

- Le righe rappresentano specifici record (istanze)
- Le colonne corrispondono ai campi dei record

L'ordine di righe e colonne è sostanzialmente irrilevante. La tabella è il livello logico di cui parlavamo inizialmente (schema DBMS, divisione livello fisico e logico). Ribadiamo che il livello logico è indipendente da quello fisico infatti una tabella è utilizzata nello stesso modo qualunque sia la sua realizzazione fisica. In questo corso vedremo solo il livello logico.

3.1.1 I modelli logici dei dati

Tradizionalmente ci sono tre modelli logici:

- Gerarchico - Organizzazione ad albero
- Reticolare - Organizzazione a grafo
- Relazionale - Organizzazione a tabella

Poi ci sono altri modelli più recenti (e meno diffusi)

- a oggetti
- XML
- NoSQL - Basato su documenti

3.1.2 Il modello relazionale

Proposto da E.F. Codd nel 1970 per favorire l'indipendenza dei dati, è diventato disponibile in DBMS reali nel 1981.
É basato sul concetto matematico di relazione a livello formale (con una variante), mentre concettualmente è basato su tabelle.
Essendo uno schema logico definisce come sono organizzati i dati e non come sono memorizzati e gestiti dal sistema informatico.

3.1.3 Il termine relazione in 3 accezioni

1. Relazione matematica - come nella teoria degli insiemi
2. Relazione - secondo il modello relazionale dei dati
3. Relazione (dall'inglese relationship) che rappresenta una classe di fatti, nel modello Entity-Relationship, tradotto anche con associazione e correlazione

3.2 Modello relazionale - definizione formale

Dati due insiemi D_1, D_2 (Estendibile a n insiemi distinti o non), si definisce **prodotto cartesiano** $D_1 x D_2$ come l'insieme di tutte le possibili coppie ordinate (v_1, v_2) tali che $v1 \in D_1$ e $v_2 \in D_2$.

Esempio $A = \{1, 2, 4\}$ e $B = \{a, b\}$ il prodotto cartesiano AxB è composto da $\{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$, cioè le sei coppie ordinate (il primo da A e il secondo da B) senza ripetizioni.
Il prodotto cartesiano rappresenta l'insieme di tutte le n-uple ordinate, mentre la **relazione matematica** rappresentare un sottoinsieme del prodotto cartesiamo fra n insiemi. Gli insiemi di partenza sono chiamati **domini**, mentre il numero delle componenti del prodotto (n) è detto grado della relazione, il numero di n-uple della relazione è la **cardinalità** della relazione.

Capitolo 4

Progettazione Concettuale

4.1 Reificazione

4.2 Stretegie di progetto

- Top-Down
- Bottom-Up
- Inside-Out

Reminder - Rappresentazione dei concetti nella specifica

- **Entità** - Se il concetto ha proprietà significative e descrive oggetti con **esistenza autonoma**
- **Attributo** - Se il concetto è una proprietà locale di un altro e non ha proprietà a sua volta
- **Relazione** - Se il concetto correla due o più concetti
- **Is-a o Generalizzazione** - Se il concetto è caso particolare di un altro

4.3 Strategia utilizzata in pratica - Mista

4.4 Qualità di uno schema concettuale

- Correttezza

- Completezza
- Leggibilità
- Minimalità

4.5 Consigli per lo svolgimento di esercizi

1. Leggere attentamente il testo
2. Suddividere il testo in varie sezioni (in base per esempio alle entità)
3. Iniziare a stendere lo schema ER secondo la suddivisione del testo, scrivendo quindi le entità e le relazioni per ogni sezione
4. Avere già un'idea generale di dove andranno posizionate le entità (giusto per non dover riscrivere tutto dopo)
5. Prosegui alla sezione successiva e ripeti il punto 3
6. Inserire attributi e segnare gli identificatori
7. Aggiungere le **CARDINALITÁ** (da non dimenticare assolutamente)

Capitolo 5

Preparazione Primo Parziale

Il primo parziale sarà composto da due esercizi

- Esercizio 1 - Schema ER
- Esercio 2 - Modello Relazionale

5.1 Schema ER

Ci verrà dato un testo e noi dovremmo creare uno schema ER, quindi dovremo identificare Entità, Relazioni, Attributi...
Sarà possibile disegnare sul testo, per esempio per dividerlo in sezioni a seconda dell'entità o relazione che rappresenta quella parte di testo.
Sarà possibile stendere prima una brutta e poi ricopiare lo schema in bella.

Consigli La prima volta leggere il testo per intero, anche se noterete che non riuscirete a ricordarlo tutto, la complessità di questi testi non permette alla nostra mente di memorizzare tutto. Durante la seconda lettura cercare di Suddividere il testo in sezioni iniziando a identificare entità e relazioni, segnando quelle che sono dubbie (per esempio fare un elenco). La terza lettura inizare a stendere lo schema utilizzando la metodologia mista vista poco fa nella progettazione concettuale.
Iniziare già a segnare gli identificatori e se sono palesi perchè il testo ce le indica, segnare le cardinalità, se invece richiedono un minimo di ragionamento è consigliabile finire prima lo schema e poi segnarle, così si avrà una visione d'insieme più chiara.
Personalmente io parto dalla prima entità e mi espando a macchia d'olio segnandomi le cose su cui ho dubbi, poi quando ho finito di stendere lo schema, mi concentro sulle parti su cui ho dubbi e verifico che abbiano senso.

Last but not least

Ricordarsi di **segnare le cardinalità** e verificare che tutto ciò che è citato nel testo sia presente, capita a volte di dimenticarsi dei pezzi e ciò è male.

Rileggi tutto

Rileggi schema e testo per verificare di non aver modellato cose che non erano citate, oppure di non aver modellato qualcosa di scritto.

- Verifica che tutte le identità abbiano identificatore
- Se c'è qualche entità collegata a una sola relazione, quindi isolata, controllarla due volte perchè potrebbero esserci degli errori, di solito le entità sono collegate a più elementi
- Visto che ce l'ho per vizio di dimenticarmi questa cosa devo ripeterla all'infinito: RICORDATI **LE CARDINALITÁ**
- Infine controlla che tutto abbia senso ("Se siamo in un osepdale fatemi arrivare i pazienti in sala operatoria" cit Schettini)

5.2 Modello Relazionale

Ci verrà dato un testo e ci verrà già dato lo schema con le relazioni, la richiesta sarà quella di:

- Trovare le chiavi primarie per ogni relazione
- Segnare i vincoli di integrità o testualmente (Relazione.attr con Relazione2.attr2) oppure graficamente facendo i collegamenti (io preferisco quest'ultima)
- Trovare almeno un vincolo di dominio (es. CF deve essere lungo 16 caratteri)
- Trovare almeno un vincolo di tupla (es. Data assunzione succesiva a Data nascita)
- Indicare una superchiave non minimale (basta indicare una cosa del tipo, *per la relazione LIBRERIA gli attributi idLib, orariApertura, quantitàLibri costituiscono una superchiave* indicando quindi tutti gli attributi)

- Indicare una chiave che non sia stata scelta come chiave primaria (es. se ho CF come chiave primaria e ho anche telefono come attributo, esso è una chiave che non è stata scelta come primaria)
- Non viene sempre chiesto: Indicare due attributi che possano assumero valore NULL - qui in genere si dovranno indicare attributi che magari inizialmente sono settati a NULL perchè costituiscono misurazoni nel tempo (es numero-giorni-pioggia)

Consigli

- Se nel testo non vengono citati attributi del tipo "id" o "codice" e me li ritrovo nello schema, con buona probabilità quelli saranno possibili chiavi primarie.
- Se sto segnando i vincoli di integrità verso una relazione che ha 2 attributi come chiave primaria, dovrò per forza cerchiare 2 attributi nella relazione d'origine!
- Se ho scelto 2 attributi come vincolo di integrità, non posso spezzarli e prenderne 1 solo per vincolarlo con un'altra relazione
- Occhio al testo, a volte si nascondono delle specifiche diverse dalla realtà a cui siamo abituati
- Indicare esplicitamente che si effettua la risoluzione dei vincoli di integrità graficamente
- Occhio agli attributi cattivi (es. portabandiera indica un atleta)
- Verifica che le chiavi primarie scelte siano minimali

Capitolo 6

Algebra Relazionale

Nell ebasi di dati relazionali esistono 2 tipi di linguaggi di interrogazione

- Procedurali - Specificano le modalità di generazione del risultato - Come
- Dichiarativi - Specificano le proprietà del risultato - Che cosa

L'algebra relazionale è procedurale, mentre SQL è parzialmente dichiarativo. L'algebra relazionale è composta da un insieme di operatori che possono essere utilizzati su relazioni per produrre relazioni. Possono essere composti dando luogo a espressioni algebriche di complessità arbitraria.

6.1 Operatori dell'algebra relazionale

Unarie

- Ridenominazione
- Selezione
- Proiezione

Binarie

- Unione, Intersezione, Differenza (Operatori insiemistici)
- Join (Join naturale, Prodotto cartesiano, Theta-join)

6.2 Operatori insiemistici

Le relazioni sono insiemi e quindi è possibile applicare gli operatori insiemistici, è fondamentale sapere che è possibile applicare queste operazioni **solo a relazioni definite sugli stessi attributi**.

Unione

L'unione di due relazioni r_1 e r_2 è la relazione che contiene le tuple che appartengono ad r_1 oppure ad r_2, oppure ad entrambe.
L'unione è commutativa e associativa.

Intersezione

L'intersezione di due relazioni r_1 e r_2 è la relazione che contiene le tuple che appartengono sia a r_1 che a r_2.
L'intersezione è commutativa e associativa ed è inoltre esprimibile per mezzo della differenza:

$$r(X) = r_1(X) \cap r_2(X) = r_1(X) - (r_1(X) - r_2(X))$$

Differenza

La differenze di due relazioni $r_1(X)$ e $r_2(X)$ definite su un insieme di attributi X è la relazione $r(X) = r_1(X) - r_2(X)$ che contiene le tuple che appartengono a $r_1(X)$, ma non a $r_2(X)$.
La differenza NON è commutativa.

Operatori insiemistici e valori nulli

Gli operatori insiemistici sono definiti anche per relazioni che contengono valori nulli.

6.3 Operatorri unari

6.3.1 Operatore di ridenominazione

Per poter applicare operazioni insiemistiche come unione, intersezione, differenza a relazioni su attributi in parte diversi è necessario ridenominare

attributi, in modo da uniformare i nomi. Questo viene fatto dall'operatore ridenominazione.
Si tratta di un operatore monadico (cioè un solo argomento) che modifica lo schema lasciando inalterata l'istanza dell'operando. Cambia quindi il nome dell'attributo, ma non il valore.

Sintassi Si indica con $\rho_{y \leftarrow x}(r)$ o $\text{REN}_{y \leftarrow x}(r)$, dove x è il nome originale dell'attributo, mentre y è quello nuovo. L'operatore è sempre seguito dal nome della relazione che stiamo considerando.
É possibile rinominare più attributi, in questo caso è importante l'ordine degli attributi dato che la sintassi sarà la seguente:

$$\rho_{y1,y_2 \leftarrow x_1,x_2}(r)$$

Esempio

Padre	**Figlio**
Adamo	Abele
Adamo	Caino

$\text{REN}_{\text{Genitore} \leftarrow \text{Padre}}(\text{Paternità})$

Genitore	**Figlio**
Adamo	Abele
Adamo	Caino

Questa operazione è fondamentale per poter effettuare operazioni insiemistiche tra relazioni con attributi diversi, in questo modo possiamo uniformare i nomi degli attributi.

6.3.2 Selezione

Permette di selezionare un sottoinsieme delle ennuple, producendo un risultato che:

- Ha lo stesso schema dell'operando
- Contiene un sottoinsieme delle ennuple dell'operando
- Contene le ennuple che soddisfano una condizione espressa dall'operatore

Sintassi $\sigma_{\text{condizione}}(r)$

Sintassi altenativa $\text{SEL}_{\text{condizione}}(r)$ Data una relazione r(X) è una formula ottenuta combinando con i connettivi OR, AND e NOT condizioni atomiche del tipo:

- CONFR è un operatore di confronto $(=, <, >, \geq, \leq)$

- A e B sono attributi in X sui cui valori CONFR abbia senso
- c'è una costante per cui il confronto CONFR sia definito

Il risultato contien ele ennuple dell'operando che soddisfano la condizione (cioè su cui la condizione è vera).

Esempi Impiegato che:

- Guadagnano più di 50 - STIPENDIO ¿ 50
- Guadagnano più di 50 e lavorano a Milano - STIPENDIO ¿ 50 AND FILIALE = 'Milano'
- Hanno un cognome uguale al nome della filiale presso cui lavorano - COGNOME = FILIALE

Tradotto in Query in Algebra Realzionale: $\text{SEL}_{\text{Stipendio}>50}$(Impiegati), lo stesso per le altre query, la parte scritta andrà sostituita nella parte condizione (sotto il SEL).

Selezione con valori nulli

La condizione atomica è vera solo per valori non nulli.
Se per esempio effettuo una SEL su una tabella con valori nulli, e la condizione seleziona tutti gli attributi (es. $\text{SEL}_{\text{Età}} > 30 \cup \text{SEL}_{\text{Età}} \leq 30$) il risultato sarà una tabella diversa da quella di partenza, perchè le condizioni atomiche vengono valutate separatamente e i valori nulli non sono valori che possiamo confrontare con un numero dato che rappresentano un valore di verità intermedio tra vero e falso. Anche inserendo tutto in una unica SEL il risultato sarebbe il medesimo, quindi senza valori nulli.
Per questo esistono gli operatori **IS NULL e IS NOT NULL**. Per avere la tabella iniziale per l'esempio Persone basterebbe quindi unire la precedente SEL con la seguente: $\text{SEL}_{\text{Età IS NULL}}$(Persone). In questo modo otteniamo la stessa relazione di partenza dato che consideriamo anche i valori NULL.

6.4 Proiezione

Si occupa di selezionare solo alcune delle colonne della tabella presa in considerazione.
Per fare un confronto con il SEL:

- SEL è un operatore ortogonale di decomposizione orizzontale, infatti riduce il numero di righe
- PROJ è un operatore ortogonale di decomposizione verticale, infatti riduce il numero di colonne

Si tratta anche in questo caso di un operatore monadico.

Sintassi $\text{PROJ}_{\text{lista di attributi}}$(Operando), il risultato conterrà le ennuple dell'perando ristrette ai soli attributi nella ListaAttributi.

Proiezione e Valori Null

Proiezione, unione e differenza continuano a comportarsi usualmente quindi due tuple sono uguali anche se ci sono dei NULL.
Dato che una relazione è un insieme e un insieme non ha elementi uguali il risultato della PROJ non conterrà ennuple uguali, esse saranno scartate.

Cardinalità delle proiezioni

La cardinalità di una relazione è il numero delle sue ennuple e si indica con $|R|$.
Una proiezione:

- Contiene al più tante ennuple quante l'oerando
- Può anche contenerne di meno (come spiegato in precedenza)

Vale la proprietà che **se X è una superchiave di R, allora PROJ**$_X(R)$ contiene esattamente tante ennuple quante R.
Per la definizione di superhciave ogni superchiave compare una sola volta nella relazione.

Selezione e Proiezione

Combinando selezione e proiezione possiamo estrarre interessanti informazioni da una realzione

Esempio Ci viene richiesto matricola e cognome degli impiegati che guadagnano più di 50:

Matricola	Cognome	Filiale	Stipendio
7309	Neri	Napoli	55
5998	Neri	Milano	64
9553	Rossi	Roma	44
5698	Rossi	Roma	64

Soluzione

$$\text{PROJ}_{\text{Matricola, Cognome}}(\text{SEL}_{\text{Stipendio}>50}(\text{Impiegati}))$$

Inserisco quindi come argomento della PROG la SEL delle tuple richieste. Combinando questi due operatori posso estrarre informazioni da una relazione. Non possiamo però correlare, mettere insieme informazioni presenti in relazioni diverse, per questo esiste il JOIN.

6.5 Join

Il Join è senz'altro l'operatore più interessante dell'algebra relazionale dato che permette di correlare, mettere insieme, integrare dati che si trovani in relazioni diverse. Ci sono diversi tipi di Join, partiamo da quello naturale.

Join naturale

Operatore binario (generalizzazione), produce un risultato sull'unione degli attributi degli operandi con ennuple costruitre ciascuna a partire da una ennupla di ognuno degli operandi.

Sintassi Date due relazioni $R_1(X_1)$ e $R_2(X_2)$, $R_1\text{JOIN}R_2$ è una relazione su X_1, X_2. Contribuiscono quindi le ennuple che hanno gli stessi valori negli attributi comuni. Quando ogni ennupla contribuisce al risultato si dice **Join completo**.
Un Join è non completo quando ci sono attributi sulle due relazioni che non corrispondono fra di loro. Se nessun attributo trova una corrispondenza si ottiene un Join vuoto.
Il join si indica anche con $\bowtie$.

Cardinalità del Join

1. Il Join di R_1 e R_2 contiene un numero di ennuple compreso fra zero e il prodotto di $|R_1|$ e $|R_2|$

2. Se il Join coinvolge una chiave di R_2 allora il numero di ennuple è compreso fra zero e $|R_1|$.

3. Se B è chiave in R_2 ed esiste vincolo di integrità referenziale fra B (in R_1) e R_2, allora il numero di ennuple è uguale a $|R_1|$
$|R_1 \,\text{JOIN}\, R_2| = |R_1|$

Il Join è commutativo e associativo.
Il Join naturale non combina due tuple se queste hanno entrambe valore nullo su un attributo in comune (e valori uguali sugli eventuali altri attributi comuni).

Proprietà del Join

In assenza di valor nulli l'intersezione di r_1 e r_2 si può esprimere

- mediante il join naturale $r_1 \cap r_2 = r_1 \text{JOIN} r_2$ oppure
- sfruttando l'uguaglianza $r_1 \cap r_2 = r_1 - (r_1 - r_2)$

In presenza di valori nulli, dalle definizioni date si ha che:

- nel primo caso il risultato non contiene tuple con valori nulli
- nel secondo caso, viceversa, tali tuple compaiono nel risultato

Nel Join naturale le ennuple che non contribuiscono al risultato vengono tagliate fuori, per questo viene utilizzato il JOIN esterno.

Join esterno

Il Join esterno estende, con valori nulli, le ennuple che verrebbero escluse da un join del tipo precedente (interno). Esiste in tre versioni:

- Sinistro $=\!\bowtie$
- Destro $\bowtie\!=$
- Completo $=\!\bowtie\!=$

Sinistro → LEFT mantiene tutte le ennuple del primo operando, estenendole con valori nulli, se necessario.

Destro → RIGHT mantiene tutte le ennuple del secondo operando, estenendole con valori nulli se necessario.

Completo → FULL mantiene tutte le ennuple di entrambi gli operandi estendendole con valori nulli se necessario.

Prodotto Cartesiano

Un Join naturale su relazioni che non hanno attributi in comune restituisce un'istanza di relazione il cui schema contiene tutti i campi di R (nell'ordine originale) seguiti da tutti i campi di S (nell'ordine originale). Contiene sempre un numero di ennuple pari al prodotto delle cardinalità degli operandi (le ennuple sono tutte combinabili).
Il prodotto cartesiano ha senso solo se seguito da selezione (dato che produce dati non reali associando anche tuple tra loro sconnesse a livello semantico). Questa operazione viene chiamata **theta-join**.

Theta-join

L'operazione Theta-join viene indicata con R_1 $\text{JOIN}_{\text{Condizione}} R_2$.
La condizione C è spesso una congiunzione AND di atomi di confronto $A_1 \theta A_2$ dove θ è un operatore di confronto (¿, ¡, =, $\leq$, $\geq$, $\neq$).
Se l'operatore è sempre l'uguaglianza (=) allora si parla di equi-join.

Join naturale e Theta-join

Così come è stato definito, il theta-join richiede in ingresso relazioni con schemi disgiunti, mentre in diversi libri di testo, lavori scientifici e anche nei DBMS viceversa il theta-join accetta relazioni con schemi arbitrati e prende il posto del join naturale, ossia tutti i predicati del join vengono esplicitati. In questo caso per garantire l'univocità degli attributi nello schema risultato è necessario ridenominare gli attributi sovrapposti in una delle relazioni o adottare dei trucchi.
Il join naturale utilizza implicitamente i nomi degli attributi per stabilire la condizione, l'equijoin li indica esplicitamente. I DBMS tipicamente non permettono il join naturale (solo ultime versioni di SQL lo permettono), però possiamo simularlo per mezzo di altri operatori.

Join e Intersezione

Quando le due relazioni hanno lo stesso schema ($X_1 = X_2$) allora due tuple fanno match se e solo se hanno lo stesso valore per tutti gli attributi, ovvero

sono identiche, per cui se $X_1 = X_2$ il join naturale equivale all'intersezione delle due relazioni.

Join e ridenominazione

Per eseguire il Join di una relazione con se stessa in modo significativo bisogna usare la ridenominazione.
$r(X)$ JOIN $r(X) = r(X)$.

6.6 Interrogazioni in algebra relazionale

Dato uno schema R di base di dati, una interrogazione è una funzione che per ogni istanza r di R produce una relazione su un dato insieme di attributi X.
É importante definire una metodologia per effettuare le Query per poter trasformare richieste verbali in algebra relazionale.

Metodologia

1. Individua le relazioni coinvolte nella specifica dell'interrogazione, attraverso gli attributi citati e le condizioni

2. Individua i tipi di operazioni necessarie

3. Individua un possibile ordinamento delle operazioni che porta ad ottenere il risultato richiesto

Se si hanno le tabelle sott'occhio è una buona idea effettuare l'interrogazione visivamente per un solo attributo.
L'ordine delle operazioni è importante! Scambiando l'ordine di alcune operazioni potremmo ottenere espressioni non funzionanti.
Esercitarsi rifacendo e capendo le query delle slide è un ottimo allenamento per assimilare questi concetti.
In algebra relazionale non sono previsti i quantificatori universali, manca per esempio l'equivalente di tutti, in Questi casi è necessario rivedere la query e riformularla in modo tale da poterla esprimere in algebra relazionale, per esempio esprimendo "tutti guadagnao più di n" come "almeno uno guadagna meno di n, oppure n", esprimendo così l'operazione tramite differenza insiemistica.

6.7 Le viste

Si tratta di una relazione temporanea, paragonabile a una variabile definita a run time, che alla fine della query non esiste più. Possono risultare molto comode per salvare risultati intermedi che poi andremo a riutilizzare anche per altre query, in questo modo non dovremo più ricalcolare qulla query.

Sintassi NomeVista = Query

6.8 Plus teorici

6.8.1 Rappresentazione delle espressioni tramite alberi

Ogni espressione dell'algebra relazionale può essere rappresentata tramite un albero, in questo modo rappresentiamo l'ordine di valutazione degli operatori. Ogni operatori corrisponde ad un nodo, quindi gli operaetori unari hanno solo un ramo in ingrasso e uno in uscita, mentre quelli binari hanno 2 rami in entrata e 1 in uscita, la radice è in alto.

6.8.2 Equivalenza di espressioni

Due espressioni sono equivalenti se producono lo stesso risultato.

- Possono essere assolute se non dipendono dallo schema
- Oppure possono dipendere dallo schema

I risultati di due Query equivalenti sono sempre equivalenti, ma la scelta non è indifferente in termini di risorse necessarie, i risultati più interessanti sono quelli che permettono una riduzione dei risultati intermedi portando a una semplificazione dell'espressione.

6.9 Regole base equivalenza

- Il Join naturale è commutativo e associativo
- Selezione e proiezione si possono raggruppare
- Selezione e proiezioone commutano
- Push down della selezione rispetto al join

6.10 Riassunto simboli

- Select - σ
- Proiezione - π
- Rename - ρ
- JOIN - $\bowtie$
- Prodotto cartesiano - $\times$

Capitolo 7

Progettazione Logica

A livello concettuale è la fase intermedia tra la progettazione concettuale e la progettazione fisica. Essa ha diverse fasi:

- Richiede di scegliere il modello dei dati - Relazionale
- Obiettivo - Definizione di uno schema logico relazionale corrispondente allo schema ER di partenza
- Aspetti importanti - Semplificazione dello schema per renderlo rappresentabile mediante il modello relazionale, ottimizzandolo per aumentare l'efficienza delle interrogazioni

L'obiettivo primarioè quello di tradurre lo schema concettuale in uno schema logico che rappresenti gli stessi dati in maniera corretta ed efficiente.

Dati in ingresso

- Schema concettuale
- Informazioni sul carico applicativo
- Modello logico

Dati in uscita

- Schema logico
- Documentazione associata

Non si tratta di una pura e semplice traduzione, dato che alcuni aspetti non sono direttamente rappresentabili.

Esempi

- Entità → Relazione del modello relazionale con gli stessi attributi.
- Generalizzazione → Dipende dalla situazione!

Risulta anche necessario considerare le prestazioni.
Qui di seguito lo schema riassuntivo conversione ER → Modello relazionale.

Tabella 7.1: *
Schema riassuntivo conversione ER - Modello relazionale

Modello ER	**Modello relazionale**
Entità	Relazione (tabella)
Relazione	Riferimento (chiave esterna)
Attributo semplice	Attributo (campo)
Attributo multivalore	Non presente
Generalizzazione	Non presente (conversione in diverse modalità)

7.1 Ristrutturazione dello schema ER

Eliminazione dallo schema ER di tutti i costrutti che non possono essere direttamente rappresentati nel modello logico target (relazionale nel nostro caso).

- Eliminazione attributi multivalore
- Eliminazione generalizzazioni

Inoltre:

- Partizionamento/Accorpamento di entità associazioni
- Scelta degli identificatori primari
- Analisi ridondanze (non dovrebbero esserci)

L'obiettivo è quello di semplificare la traduzione e ottimizzare le prestazioni, teniamo presente che uno schema ER ristrutturato non è più uno schema concettuale nel senso stretto del termine.
Per ottimizzare il risultato abbiamo bisogno di analizzare le prestazioni a

questo livello, ma le prestazioni non sono valutabili con precisione su uno schema concettuale, dipendono dalle caratteristiche del DBMS, dal volume dei dati e dalla caratteristiche delle operazioni.

Indicatori dei parametri di prestazioni

Consideriamo indicatori dei parametri che regolano le prestazioni:

- spazio: numero di occorrenze previste
- tempo: numero di occorrenze (di entità e relationship) viste durante un'operazione

7.1.1 Carico applicativo

Consideriamo degli indicatori dei parametri che regolano le prestazioni:

- Tempo di esecuzione delle operazioni di principale interesse, numero di istanze mediamente accedute durante l'esecuzione dell'operazione
- Spazio di memoria necessario per memorizzare i dati di interesse

Per valutare questi parametri bisogna conoscere oltre allo schema:

- Volume dei dati - Numero di istanze previste di entità e relazioni, dimensioni di ciascun attributo
- Caratteristiche delle operazioni - Tipo (interattiva o batch), frequenza e dati coinvolti

Lo schema di operazione descrive i dati coinvolti in un'operazione, corrisponde al frammento dello schema ER interessato all'operazione sul quale viene disegnato il cammino logico per accedere alle informazioni di interesse.

7.1.2 Tavola degli accessi

Con lo schema di operazione si può fare una stima del costo di un'operazione contando il numero di accessi alle istanze di entità e relazioni, il risultato può essere riassunto in una tavola degli accessi. Il tipo distingue gli accessi in scrittura (S) e in lettura (L), le operazioni di scrittura sono in genere più onerose (esecuzione in modo esclusivo, aggiornamento degli indici).
Solitamente per produrre la tavola di accessi si fa riferimento allo schema di navigazione, cioè la parte dello schema ER interessata dall'operazione estesa con delle frecce che indicano in che modo l'oerazione naviga i dati.

Concetto	Costrutto	Accessi	Tipo

Tabella 7.2: Table with four columns

7.1.3 Tavola dei volumi

Specifica il numero stimato di istanze per ogni entità E e associazione R dello schema, i valori sono necessariamente approssimati, ma indicativi. Il numero medio di partecipazioni di una istanza di entità alle istanze di relazione dipende dalla cardinalità delle relazioni.
I volumi sono quindi influenzati da:

- Cardinalità dello schema
- Numero medio di volte che le istanze delle entità partecipano alle relazioni

Il numero delle istanze si ricava dalla tavola dei volumi mediante semplici operazioni.

7.1.4 Analisi delle ridondanze

Una ridondanza in uno schema ER è una informazione significativa ma derivabile da altre, in questa fase si decide se eliminarle o tenerle.
Ci sono sia vantaggi che svantaggi nel mantenere delle ridoondanze:

- Vantaggi - Semplificazione delle interrogazioni
- Svantaggi - Maggiore occupazione di spazio, aggiornamenti più pesanti

Forme di ridondanza

- Attributi derivabili da altri attributi della stessa entità o di altre entità o relazioni
- Associazioni derivabili dalla composizione di altre relazioni in presenza di cicli

É necessario considerare la semantica della relazione coinvolta nel ciclo, perchè in casi simili potrebbero esserci delle ridondanze che non sono tali.

Esempio slide 35 Docenza - Tesi

7.2 Eliminazione gerachie

Il modello relazionale non supporta la rappresentazione diretta delle generalizzazioni, mentre entità e relazioni sì, quindi si eliminano le gerarchie sostituendole con entità o relazioni.
Ci sono tre possibilità:

1. Accorpamento delle figlie della generalizzazione nel genitore
2. Accorpamento del genitore della generalizzazione nelle figlie
3. Sostituzione della generalizzazione con relazioni

7.2.1 Accorpamento delle figlie nel genitore

- Le entità figlie sono eliminate e al loro posto viene creato un attributo che le rappresenta nell'entità padre (un attributo per distinguere il tipo, quale figlia è o nessuna)
- Gli attributi della figlia vengono spostati nel padre
- Gli attributi che provengono dalla figlia possono essere nulli
- Lre relazioni che provengono da una sola figlia hanno cardinalità minima pari a 0

Pro e Contro

- Vantaggi - Accesso contestuale agli attributi del padre e della figlia
- Svantaggi - Si ha spreco di memoria per i valori nulli

7.2.2 Accorpamento del genitore nelle figlie

- L'entità padre si elimina
- Gli attributi, le associazioni e l'identificatore del padre sono aggiunti alle figlie
- Ogni associazione definita sul padre genera una associazione distinta per ogni figlia

Pro

- Migliore se si effettuano prevalentemente operazioni solo su istanze di una classe figlia
- Non ci sono di principio valori nulli

Contro Possibile solo se la generalizzazione è totale, dato che non si possono rappresentare istanze che non sono in nessuna delle classi figlie.

7.2.3 Sostituzione della generalizzazione con relazioni

- Si introduce una relazione uno-a-uno fra l'entità padre e ciascuna entità figlia
- Occorre inserire il vincolo che ogni istanza dell'entità padre può partecipare solo ad una relazione di legame con le entità figlie
- Se la generalizzazione è totale ogni istanze dell'entità padre partecipa necessariamente ad una (sola) delle relazioni di legame con le figlie

Pro

- Conviene quando la generalizzazione non è totale e ci sono operazioni che fanno distinzione fra le entità padre e figlie
- Non è necessario introdurre valori nulli
- Genera entità con pochi attributi (le tabelle corrispondenti sono più piccole e più tuple possono essere gestite in memoria principale)

Contro

- Si incrementa il numero di accessi per mantenere la consistenza delle istanze rispetto ai vincoli introdotti

La scelta fra le alternative si può fare con metodo simile a quello visto per l'analisi delle ridondanze (però non basato solo sul numero degli accessi).

Regole generali

- **Accorpamento delle figlie della generalizzazione nel genitore** conviene se gli accessi al padre e alle figlie sono contestuali
- **Accorpamento del genitore nelle figlie** conviene se gli accessi alle figlie sono distinti e solo se la generalizzazione è totale
- **Sostituzione della generalizzazione con relazioni conviene** se gli accessi alle entità figlie sono separati dagli accessi al padre

Sono possibili anche soluzioni ibride, soprattutto in gerarchie a più livelli.

7.2.4 Note pratiche

L'accorpamento nel padre è sempre applicabile, ma è sconsigliato in presenza di elevato numero di relazioni e/o attributi provenienti dalle entità figlie (necessità vincoli di integrità).
L'accorpamento nelle figlie è applicabile solo se la generalizzazione è totale, è sconsigliabile applicare questa tecnica per una copertura parziale.
La tecnica non è adatta per coperture sovrapposte dato che risulterebbe difficile gestire degli identificatori duplicati relative alle istanze comuni a più entità figlie.
Mentre per quanto riguarda la creazione di relazioni, è la soluzione più semplice, sempre applicabile, ma può essere dispendiosa per ricostruire l'informazione di partenza.

7.3 Partizionamento/accorpamento di entità e relationship

Ristrutturazioni effettuate per rendere più efficienti le operazioni in base a un semplice principio:

Gli accessi si riducono

- Separando attributi di un concetto che vengono acceduti separatamente
- Raggruppando attributi di concetti diversi acceduti insieme

Casi principali

- Partizionamento verticale di entità
- Partizionamento orizzontale di relationship
- Accorpamento di entità/relationship

7.3.1 Accorpamento di entità

Due entità legate da una associazione possono essere fuse in un'unica entità contenente gli attributi di entrambe quando le operazioni fanno sempre riferimento a tutti gli attributi delle due entità. In questo modo si risparmiano gli accessi per recuperare i dati attraverso la relazione che lega le due entità. Si effettuando per associazioni uno a uno.

7.4 Eliminazione di attributi multivalore

Il modello relazionale non consente la rappresentazione di attributi multivalore, si crea quindi una relazione che collega l'entità dell'attributi designato con una nuova relazione rappresentante l'attributo multivalore. La cardinalità sarà uno a molti (dato che ogni entità avrà una sola istanza dell'attributo multivalore trasformato in entità associato).

7.5 Scelta degli identificatori principali

Operazione indispensabile per la traduzione nel modello relazionale.

Criteri

- Assenza di opzionalità (NO valori nulli)
- Semplicità (preferebza agli identificatori interni, dimensioni ridotte)
- Utilizzo nelle operazioni più frequenti o importanti
- Se nessuno degli identificatori soddisfa i requisiti precedenti si introducono nuovi attributi (codici) contenenti valori speciali generati appositamente per questo scopo

Capitolo 8

SQL - Structured Query Language

SQL è un linguaggio per la definizione e la manipolazione dei dati in database relazionali adottato da molti DBMS.
Ci sono diverse versioni, la prima versione ufficiale risale al 1986. Poi sono state rilasciate altre versioni come SQL-89, SQL-2, SQL-3...
Noi faremo riferimento principalmente a SQL-2. Questa versione è ricca e complessa, tanto che nessun sistema commerciale lo implementa in maniera completa.
Esistono 3 livelli di conformità:

- Entry level: molto simile a SQL-89
- Intermediate level: versione che soddisfa le esigenze di mercato
- Full level: versione completa anche delle funzioni avanzate che non sono realizzate in alcun DBMS

La maggior parte dei database è conforme solo all'entry level.
Alcune famose implementazioni di SQL sono:

- ORACLE
- DB2 (IBM)
- Access (Microsoft)
- MSSQL server (Microsoft)
- MySQL
- Firebird

8.1 Confronto con Algebra Relazionale e Istruzioni principali

8.1.1 SQL e Algebra Relazionale

SQL è relazionalmente completo: ogni espressione logica può essere tradotta in SQL. Viene adottata la logica dei 3 valori (T, F, U) dell'Algebra relazionale (U = Unknown).
Il modello dati di SQL è basato su tabelle anzichè relazioni (possono essere presenti righe duplicate).
SQL è computazionalmente completo, ha istruzioni di controllo.

Logica a 3 valori Logica che utilizza:

- True
- False
- Unknown

Che segue la seguente tabella di verità

Tabella 8.1: SQL Three-Value Logic Truth Table

Value 1	**Value 2**	**AND**	**OR**
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	UNKNOWN	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	TRUE
UNKNOWN	FALSE	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

8.1.2 Istruzioni principali DDL

Operazioni di definizione schema e modifica

- CREATE - Definisce database, tabelle, domini, viste, vincoli e autorizzazioni

- ALTER - Modifica attributi e vincoli
- DROP - Elimina database e tabelle

8.1.3 Istruzioni principali DML

Operazione di interrogazione: SELECT. Formula Query come nell'AR o anche richieste più elaborate.

Aggiornamento

- INSERT - Inserisce nuove tuple nelle tabelle
- DELETE - Elmina tuple nelle tabelle
- UPDATE - Modifica tuple

Queste ultime possono basarsi sul risultato di una query.

8.1.4 SQL è dichiarativo

Essendo principalmente dichiarativo non possiamo scegliere l'ordine in cui avvengono le operazioni è necessario attenersi alla struttura sintattica delle istruzioni.

8.1.5 Notazione SQL

- Termini del linguaggio in MAIUSCOLO
- Termini variabili (specificati dall'utente) in minuscolo
- ¡x¿ usate per isolare un termine x

  x indicano che il parametro è opzionale

- — seprara opzioni alternative

8.1.6 Primo esempio di Query

```
SELECT ListaAttributi
FROM ListaTabelle
[WHERE Condizione]
```

Dove SELECT e FROM sono obbligatori, mentre WHERE è opzionale (in realtà c'è praticamente sempre in quanto serve per filtrare i risultati).

8.2 SQL-DDL

Uno schema di base di dati è una collezione di oggetti: domini, tabelle, asserzioni, viste, privilegi. La sintassi è la seguente:

```
CREATE SCHEMA
  [Nome Schema]
  [[AUTHORIZATION] Autorizzazione]
  {DefinizioneElementoSchema}
```

Nome schema se omesso indica l'utente che ha lanciato il comando, mentre autorizzazione è il proprietrio dello schema (utente che lo ha definito).

8.2.1 Tabelle

Tramite CREATE TABLE si definisce una tabella. Una tabella non è altro che uno schema di relazione, con CREATE viene creata un'istanza vuota.

```
CREATE TABLE NomeTabella
  (NomeAttributo1 TipoDominio1 [Valore di Default] [Vincolo1],
  NomeAttributo2 TipoDominio2 [Valore di Default] [Vincolo2],
  ...
  [AltriVincoli])
```

In questo ambito parleremo di tabelle e non di relazioni, e di righe e non di tuple, perchè rispetto al modello relazionale possiamo avere righe duplicate. Qui di seguito un esempio reale di definizione di tabella:

```
CREATE TABLE Impiegato(
  Matricola CHAR(6) PRIMARY KEY,
  Nome CHAR(20) NOT NULL,
  Cognome CHAR(20) NOT NULL,
  Dipart CHAR(15),
```

```
    Stipendio NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(Dipart) REFERENCES
    Dipartimento(NomeDip),
    UNIQUE (Cognome,Nome)
)
```

8.2.2 Definizione dei Dati: I Domini

I domini specificano i vlori ammessi da ciascun attributo. SQL ha 6 domini elementari predefiniti:

- Carattere- VARCHAR - Stringa di lunghezza variabile tra 0 e n
- Numerico Esatto - INTEGER, SMALLINT, NUMERIC
- Numerico Approssimato - FLOAT, REAL, DOUBLE PRECISION
- Data/Ora TIMESTAMP, DATE, TIME
- Intervallo Temporale
- Bit (SQL-2) - BOOLEAN (SQL-3) - con dominio 0,1

Bit è stato poi eliminato e sostituito parzialmente da BOOLEAN in SQL-3
L'utente pu definire dei domini custom (semplici, ma riutilizzabili).

BLOB, CLOB sono oggetti di grandi dimensioni, costituiti da binari (BLOB) o caratteri (CLOB), vengono memorizzati in maniera differente dagli altri dati e sono usati per memorizzare informazioni non strutturate (immagini, video, testi...).

8.2.3 Il tipo Bit

Utilizzato spesso per definire se è presente o meno una certa proprietà, dato che si tratta di un BOOLEAN.

8.2.4 Carattere

Si indica:

```
CHAR(n) - Stringa di lunghezza fissa n
VARCHAR(n) - Stringa di lunghezza variabile tra 0 e n
```

8.2.5 Numerici Esatti

Rappresentano numeri interi o numeri decimali in virgola fissa (con un numero prefissato di decimali, come per i valori monetari). Precisionè il numero di cifre significative, scala il numero di cifre dopo la virgola.

- INTEGER/SMALLINT rappresentano valori interi - La precisione varia a seconda della specifica implementazione di SQL, SMALLINT richiede meno spazio di memorizzazione
- NUMERIC/DECIMAL rappresentano valori decimali - La differenza fra questi due è che il primo deve essere implementato esattamente con la precisione richiesta, mentre il secondo può avere una precisione maggiore.

8.2.6 Numerici Approssimati

Sono utili per rappresentare valori reali approssimati, ad esempio grandezze fisiche (rappresentazione in virgola mobile, in cui a ciascun numero corrisponde una coppia di valori: mantissa e esponente).
REAL e DOUBLE PRECISION rappresentano valori a singola/doppia precisione in virgola mobile. FLOAT permette di richiedere la precisione che si desidera.

8.2.7 Data e Ora

Permettono di descrivere informazioni temporali, rappresentando istanti di tempo:

- DATE rappresenta le date espresse come anno (4 cifre), mese (2 cifre) e giorno (2 cifre) - DATE 'yyyy-mm-dd'
- TIME [WITH TIME ZONE] rappresenta l'ora del giorno, espressa come ora (2 cifre), minuti (2 cifre) e secondi (2 cifre)
- TIMESTAMP

Ciascuno di questi domini è strutturato e decomponibile in un insieme di campi (anno, mese, giorno, ora, minuti, secondi).

8.2.8 Intervalli temporali

Permette di rappresentare intervalli di tempo come durate di eventi.
INTERVAL rappresenta una durata temporale, esistono interval ani e mesi, oppure giorni e ore, ma non mesi e giorni poichè i mesi non hanno tutti lo stesso numero di giorni.

8.2.9 CLOB e BLOB

Permettono di includere nel database oggetti molto grandi (come dati multimediali). Sono implementati come valore e non possono essere usati come criterio di selezione per le query.

8.2.10 Domini definiti dall'utente

```
CREATE DOMAIN Voto AS SMALLINT
  DEFAULT 0
  CHECK (VALUE >= 18 AND VALUE <= 30)
```

Possono mettere vincoli può o meno utili.
Al contrario dei meccanismi di definizione dei tipi nei linguaggi di programmazione, SQL-2 non mette a disposizione dei costruttori di dominio come record o array. Questa caratteristica deriva dal modello relazionale dei dati il quale richiede che ogni attributo sia definito su un dominio elementare.

8.2.11 Valori di Deafult

Definiscono il valore che deve assumere l'attributo quando non viene specificato un valore durante l'inserimento di una tupla.
DEFAULT < ValoreGenerico — user — null >

- ValoreGenerico rappresenta un valore compatibile con il dominio, rappresentato come una costante o come un'espressione.
- user è l'identificativo dell'utente che effettua il comando di aggiornamento della tabella.
- null è il valore di DEFAULT di base.

8.2.12 Il valore NULL

É un valore polimorfico che appartiene a tutti i domini con il significato di valore non noto.

- Il valore esiste ma non è noto al database
- Il valore è inapplicabile (es. numero patente per minorenni)
- Non si sa se il valore è inapplicabile o meno (es. numero patente per un maggiorenne)

8.2.13 Vincoli di Integrità

Un vincolo è una regola che specifica delle condizioni sui valori di un elemento dello schema del database. Un vincolo può essere associato ad una tabella, ad un attributo, ad un dominio.

- Vincoli Intrarelazionali - Proprietà sempre valida all'interno di una relazione
- Vincoli Interrelazionali - Proprietà sempre valida tra relazioni diverse

Sintassi

- NOT NULL - Valore non nullo
- UNIQUE - Valore unico
- PRIMARY KEY - Chiave primaria
- CHECK - Definisce condizioni complesse (sia intra che inter relazionali)

```
Nome CHAR(20) NOT NULL
Cognome CHAR(20) NOT NULL
UNIQUE (Nome, Cognome)
```

NOT NULL su Nome e Cognome è un vincolo intrarelazionale, UNIQUE è Interrelazionale (non posso avere istanze uguali ripetute)

8.2.14 Chiave

Insieme di attributi che identificano univocamente una tupla all'interno di una relazione. Il vincolo PRIMARY KEY può essere definito una volta sola all'interno della relazione (implicitamente NOT NULL e UNIQUE).

Eccezioni Il valore NULL può comparire su diverse righe senza violare il vincolo.

CHECK e FOREING KEY

Utilizzato per definire vincoli complessi, sia intrarelazionali che interrelazionali.

```
CREATE TABLE Studente (
  CREATE DOMAIN Voto AS SMALLINT
  DEFAULT 0
  CHECK (Voto>=18 AND VOTO<=30)
)
```

Per quanto riguarda invece i vincoli interrelazionali, si utilizza FOREIGN KEY e REFERENCES.

FOREIGN KEY Un vincolo di intergrità referenziale ("FOREIGN KEY") fra gli attributi X di una relazione R_1 e un'altra relazione R_2 impone ai valori su X, in R_1 di comparire come vlaori della chiave primaria di R_2.
Un vincolo di intergrità referenzialie fra gli attributi $X = A_1, A_2, ...$ di una tabella figlio interna e un'altra tabella padre esterna impone ai valori degli attributi X nella tabella Figlio di comparire come valori della chiave primaria nella Tabella Padre.
Alcuni attributi della tabella figlio sono definiti come FOREIGN KEY e si devono riferire (REFERENCES) ad alcuni attributi della tabella padre che costituiscono una chiave (devono essere UNIQUE e NOT NULL, oppure PRIMARY KEY).
I valori contenuti nella FOREIGN KEY devono essere sempre presenti nella tabella padre.
La tabella Figlio viene anche definita **interna**, mentre quella Padre **esterna**.

Due sintassi

1. Nella parte di definizione degli attributi con il costrutto sintattico REFERENCES

   ```
   AttrFiglio CHAR(3) REFERENCES TabellaPadre(AttrPadre)
   ```

2. Oppure dopo le definizione degli attributi con i costrutti FOREIGN KEY e REFERENCES

   ```
   FOREIGN KEY (AttrFiglio) REFERENCES TabellaPadre(AttrPadre)
   ```

Quando si hanno più attributi da riferire si utilizza sempre FOREIGN KEY e REFERENCES, se si omettono gli attributi destinazione, vengono assunti quelli della chiave primaria.

8.2.15 Aggiornamenti e Violazioni

Se viene eseguita un'operazione di aggiornamento che viola un vincolo di integrità referenziale la base di dati diventa non valida, per questo sono definite un insieme di politiche per evitare che questo accada. Per i vincoli di integrità SQL permette di scegliere delle reazioni da adottare in caso di violazioni, tali politiche vanno dichiarate nella dichiarazione DDL dello schema. Si possono introdurre violazioni operando sulle righe della tabella padre (esterna) o sulle righe della tabella efiglio (interna)

Modifiche Tabella Figlio

- Inserimento nuova riga
- Modifica della FOREIGN KEY

Non venogno proposte delle reazioni, le operazioni vengono rifiutate

Tabella Padre

- Cancellazione riga
- Modifica dell'attributo riferito

Vengono proposte diverse reazioni:

- CASCADE - L'operazione modifica/cancellazione viene propagata alla tabella figlio (o tabella interna)
- SET NULL - NULL nella tabella figlio in entrambi i casi
- SET DEFAULT - default nella tabella figlio in entrambi i casi
- NO ACTION - rifiutata in entrambi i casi

Le politiche di reazione possono essere definite in modo diverso per eventi di modifica/cancellazione

```
Dipart CHARACTER(15) REFERENCES Dipartimento(NomeDip)
  ON DELETE SET NULL
  ON UPDATE CASCADE
```

Con CASCADE in pratica se effettuo una cancellazione cancello anche tutte le relazioni legati a quel valore, con NULL invece pongo a NULL la FOREIGN KEY, ma mentengo gli altri valori. É possibile assegnare un nome a questi vincoli, per rendere più chiaro il codice e rendere quindi più semplice il debugging e la scrittura della documentazione.

```
Stipendio INTEGER CONSTRAINT StipendioPositivo
CHECK (Stipendio > 0),
...
CONSTRAINT ForeignKeySedi
FOREIGN KEY (Sede) REFERENCES Sedi
```

8.3 Modifiche degli schemi

Necessarie per garantire l'evoluzione della base di dati a fronte di nuove esigenze.
Ci sono 2 comandi SQL appositi:

- ALTER - Modifica oggetti
- DROP - Cancella oggetti dallo schema

8.3.1 ALTER

- ALTER DOMAIN - modifica domini (default, vincoli)
- ALTER TABLE - modifica tabelle

Attenzione: Quando si definisce un nuovo vincolo deve essere soddisfatto dalla istanza presente, altrimenti viene rifiutato.

Modifiche degli schemi - DROP

- DROP DOMAIN - cancellazione del dominio (attributo)
- DROP TABLE - cancellazione della tabella

Opzioni

- RESTRICT - Un oggetto (dominio, tabella) non è rimosso se non è vuoto
- CASCADE - viene rimosso l'oggetto e tutto ciò che è coinvolto nella definizione dell'oggetto

```
ALTER DOMAIN NomeDominio <
  SET DEFAULT ValoreDefault|
  DROP DEFAULT|
  ADD CONSTRAINT DefVincolo |
  DROP CONSTRAINT NomeVincolo >
```

Esempio ALTER su tabella

```
ALTER TABLE NomeTabella <
  ALTER COLUMN NomeAttributo<
    SET DEFAULTNuovoDefault |
    DROP DEFAULT> |
  DROP COLUMN NomeAttributo |
  ADD COLUMN DefAttributo |
  DROP CONSTRAINTNomeVincolo
  ADD CONSTRAINTDefVincolo >
```

```
DROP <schema, domain, table, view, ...> NomeElemento
  [RESTRICT | CASCADE]
```

- RESTRICT - Impedisce drop se gli oggetti comprendono istanze non vuote
- CASCADE - Applica drop agli oggetti collegati, Potenziale pericolosa reazione a catena

8.3.2 Cataloghi relazionali

Il catalogo contiene il dizionario dei dati (data dictionary) ovvero la descrizione della struttura dei dati contenuti nel database. Anche questa descrizione di dati (metadati) è rappresentata tramite una struttura relazionale.
SQL-2 Organizza il catalogo su due livelli:

- Definition_Schema (composto da tabelle che descrivono tutte le strutture della base di dati)
- Information_Schema (composto da viste che costituiscono l'interfaccia verso il dizionario dei dati)

8.4 Interrogazioni SQL - Query

Le interrogazioni avvengono usando l'istruzione SELECT

```
SELECT ListaAttributi
FROM ListaTabelle
[ WHERE Condizione ]
```

- SELECT indica quali attributi vogliamo nel risultato
- FROM indica le tabelle da cui estrarre i dati
- WHERE indica che condizioni devono essere soddisfatte

SELECT non significa selezione, sceglie le colonne output della query.
FROM se vengono indicate più tabelle si effettua il loro prodotto cartesiano, salvo diversamente specificato (es. JOIN).

8.4.1 Istruzione AS

Si possono fare ridenominazioni su attributi e anche su relazioni con la clausola AS, molto utile per evitare di scrivere ogni volta l'attributo o la tabella per esteso (specialmente se i nomi sono lunghi).

```
SELECT AttrEspr [[AS] Alias]{,AttrEspr [[AS] Alias]}
FROM Tabella [[AS] Alias]{,Tabella [[AS] Alias]}
[WHERE Condizione]
```

Esempio pratico:

```
SELECT Nome AS N, Stipendio AS Salario
FROM Dipendente AS Dip
WHERE Dip.Stipendio > 1000
```

8.4.2 Asterisco

Per selezionare tutti gli attributi viene utilizzato *.

8.4.3 FROM clausola .

La clausola PUNTO . permette di specificare a che relazione appartiene un attributo (o anche una tabella o un database).
Esempio:

```
SELECT Dip.Nome, Dip.Stipendio
FROM Dipendente AS Dip
WHERE Dip.Stipendio > 1000
```

Si possono ovviamente usare gli alias denominati con AS dato che viene prima letta la parte del FROM.

8.4.4 WHERE

Parte fondamentale della Query (seppur opzionale), specifica la condizione di selezione. L'argomento do WHERE è un'espressione booleana:

```
WHERE [NOT] PredicatoSemplice {< AND | OR > [NOT] PredicatoSem
```

Operatori di confronto

- =, ¡¿, !=, ¿, ¿=, ¡, ¡=
- BETWEEN valore1 AND valore2
- IN (valore1, valore2, ...)
- LIKE 'stringa' (con % e _ per sostituire caratteri)
- IS NULL, IS NOT NULL

Si selezionano per il risultato solo le tuple per le quali l'espressione che segue la clausola WHERE è vera.

Precedenza operatori Booleani

- NOT
- AND
- OR

Per stabilire la precedenza **usare sempre le parentesi**.

8.4.5 WHERE - Operatore Like

L'operatore LIKE permette di esprimere dei pattern su stringhe di caratteri mediante la seguente notazione:

- _ indica un singolo carattere arbitrario
- % indica una stringa di caratteri arbitraria (anche lunga 0)

Esempio Tutti nomi degli impiegati che finiscono con una i e hanno una 0 in seconda posizione:

```
SELECT Nome
FROM Impiegato
WHERE Nome LIKE '_o%i'
```

8.4.6 WHERE - Operatore BETWEEN

L'operatore BETWEEN permette di esprimere condizioni di appartenenza a un intervallo.

```
WHERE Attributo BETWEEN Valore1 AND Valore2
```

8.4.7 WHERE - Operatore IN

L'operatore IN permette di esprimere condizioni di appartenenza a un insieme.

Esempio Cognome e numero uffcio degl impiegati negli uffici 10 e 20:

```
SELECT Cognome, ufficio
FRM Impiegato
WHERE ufficio IN ('10', '20')
```

Si poteva anche usare:

```
WHERE ufficio = '10' OR ufficio = '20'
```

8.4.8 WHERE - Valori nulli

SQL-2 usa una logica a tre valori per valutare il valore di verità di una clausola WHERE: True (T), Flase (F), Unkown (?).
Un predicato semplice valutato su un attributo a valore nullo dà come risultato della valutazione ?.
Una tupla per cui il valore di verità è ? non viene restituita dalla query.
Se la valutazione del predcato di un constraint è ? il constraint non è violato.

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

NOT	
T	F
F	T
?	?

Figura 8.1: Tabella di verità

Tabella di verità Per valutare gli attributi nulli è necessario usare IS NULL o IS NOT NULL.

- IS NULL - vero se attributo ha valore nullo, falso altrimenti
- IS NOT NULL - vero se attributo ha valore specificato, falso altrimenti

Esempio per la seguente tabella

A	B	C
a	?	c1
a1	b	c2
a2	?	?

```
SELECT *
FROM Tabella
WHERE B IS NULL
```

Restituisce t1 e t3.

Esempio reali Query: gli impiegati che **sono** o **potrebbero essere** ingegneri.

```
SELECT *
FROM Impiegato
WHERE mansione = 'Ingegnere' OR mansione IS NULL
```

8.4.9 SELECT - DISTINCT

In algebra relazionale i risultati della interrogazioni non contengono elementi duplicati, mentre in SQL le tabelle prodotte dalle interrogazioni possono contenere più righe identiche tra loro.
I duplicati possono essere rimossi usando la parola chiave DISTINCT.

```
SELECT [DISTINCT] AttrEspr
FROM Tabella
[WHERE Condizione]
```

8.4.10 Espressioni e Funzioni

I predicati usati nelle interrogazioni possono coinvolgere, oltre a nomi di colonna anche espressioni. Le espressioni sono formulate applicando operatori ai valori delle colonne delle tuple.
Esempi di espressioni e funzioni sono que40lle aritmetiche su stringhe, su date e tempi. Le espressioni possono comparire nelle clausole SELECT, WHERE e UPDATE.

SELECT Una espressione usata nella clausola SELECT dà luogo ad una nuova colonna che non corrisponde a nessuna colonna della tabella su cui si effettua la selezione (argomento di FROM).
Questa colonna è detta Colonna **virtuale**. Le colonne virtuali non sono fisicamente memorizzate, ma sono materializzate (esistono) solo come risultato delle interrogazioni.
Anche alle colonne virtuali è possibile assegnare un alias (con il costrutto AS).

Funzioni predefinite

- su stringhe (UPPER, UCASE, LENGTH)

- su date, intervalli (+, -, DATE, DAYOFWEEK, ...)
- matematiche (*, +, -, /, TAN, SQRT, SIN, ...)
- informazioni di istema (USER, CURRENT_DATE, ...)

Esempio Trovare stipendio mensile degli impiegati che guadagnano più di 40.
Dato che lo stipendio memorizzato nel DB è annuale, bisogna dividere per 12.

```
SELECT stipendio/12 AS stipendio_mensile
FROM Impiegato
WHERE stipendio > 40
```

Se non inserisco un ALIAS la colonna virtuale avrà un nome generato (non significativo).

Funzioni per le stringhe Gli operatori più comuni sono:

- —— - concatenazione
- UPPER, UCASE - Trasforma la stringa in caratteri maiuscoli
- LOWER, LCASE - Trasforma la stringa in caratteri minuscoli

8.5 Operatore JOIN

L'operatore JOIN rappresenta un'importante funzionalità in quanto permette di correlare dati in tabelle diverse.
Matematicamente si tratta del prodotto cartesiano tra le due tabelle, che avrà un numero di righe pari al prodotto del numero di righe delle due tabelle e viene applicata una condizione di matching tra le colonne specificate.
Il puro prodotto cartesiano avviene fra le tabelle inserite nel FROM, mentre la condizione di matching è inserita o nel WHERE (**JOIN implicito**) o nel FROM esplicitando il JOIN (**JOIN esplicito**).
Ad un operatore JOIN sono applicati uno o più predicati di join.
Un predicato di join esprime una condizione che deve essere verificata dalle tuple del risultato dell'interrogazione. Il predicat di join permette di estrarre dal DB un sottoninsieme opportuno del prodotto cartesiano.

8.5.1 JOIN implicito

Quando la condizione di selezione viene inserita nel WHERE allora sto effettuando un JOIN implicito.

```
SELECT *
FROM Impiegato, Dip
WHERE Impiegato.Dipart = Dip.Nome;
```

In questo caso ho estratto tutte le informazioni sugli impiegati e sui dipartimenti in cui lavorano.

Esempio Esami Estrarre Matricola, Cognome, Nome Voto di tutti gli studenti che hanno sostenuto l'esame di 'Analisi I'.

```
SELECT S.Matricola, S.Cognome, S.nome, E.Voto
FROM Studenti AS S, Esami AS E
WHERE (S.Matricola = E.Studente) AND (E.Corso = 'Analisi␣I')
```

Come già detto in precedenza il FROM efettua il prodotto cartesiano, mentre la condizione di WHERE indica il predicato di join.

8.5.2 JOIN esplicito

Introdotto in SQL-2, è possibile effettuare esplicitamente il JOIN nella clausola FROM

```
SELECT AttrExpr [ [AS] Alias ] {, AttrExpr [[AS] Alias ] }
FROM Tabella [ [AS] Alias ]
{ [ TipoJoin ] JOINTabella [[AS] Alias ] ONCondizJoin}
[ WHERE CondizQuery ]
```

In questo modo esplicitiamo il JOIN sulle tabelle e separiamo le condizioni di JOIN dalle condizioni di QUERY.

Tipi di JOIN

- INNER (questo prefisso non si mette dato che è il JOIN di default)
- OUTER JOIN
  - RIGTH [OUTER]

- LEFT [OUTER]
- FULL [OUTER]

ATTENZIONE l'ordine degli operandi è importante! Scambiandoli il risultato può cambiare, è sempre consigliabile effettuare i JOIN nell'ordine in cui il testo ce lo chiede e nell'ordine con cui percorriamo i collegamenti fra le chiavi esterne, se per esempio devo fare il JOIN tra corso, docente e residenza, dato che la chiave esterna di corso riferisce a docente e la chiave esterna di docente riferisce a città, effettuerò il JOIN in questo ordine.

Esempio JOIN sul seguente schema

ID_impiegato	Nome	Cognome	Dipart	Ufficio	Stipendio	premioprod	Mansione

Nome	Indirizzo	Città

```
SELECT I.Nome, I.Cognome, Dip.Citta
FROM Impiegato AS I JOIN Dip ON I.Dipart = Dip.Nome
```

Non si ha traccia delle tuple di impiegato che non corrispondono ad alcuna tupla di Dip.

8.5.3 OUTER JOIN

L'operatore di OUTER JOIN aggiunge l risultato le tuple di Impiegato e Dip che non hanno partecipato al JOIN, completandole con NULL.
Consideriamo il seguente esempio:

```
FROM Impiegato AS I [FULL|LEFT|RIGHT] OUTER JOIN Dip ON ...
```

LEFT Le tuple di impiegato che non partecipano al join vengono completate ed inserite nel risultato.

RIGHT Le tuple di Dip che non partecipano al join vengono completate ed inserite nel risultato.

FULL (Sia Left che Right) Sia le tuple di impiegato che quelle di Dip che non partecipano al join vengono completate ed inserite nel risultato.

8.5.4 Self JOIN

JOIN effettuato sulla stessa tabella.

```
SELECT G1.Genitore AS Nonno
FROM Genitori G1, Genitori G2
WHERE G1.Figlio = G2.Genitore
AND G2.Figlio = 'Anna'
```

In questo caso vengono ricercati i nonni di Anna. É possibile fare questo perchè quando seleziono le tabelle nel FROM tratto delle loro copie, quindi posso anche prenderle due volte.

8.6 Operatori di ordinamento e aggregazione

8.6.1 Ordinamento del risultato: ORDER BY

Le righe sono ordinate in base al primo attributo, se i valori sono uguali, viene ordinata in base al secondo attributo ecc.
Ordinamento: ascendente (ASC), discendente (DESC), (ASC è default e può essere omesso).

```
SELECT *
FROM Impiegato
ORDER BY Stipendio
```

L'opzione DESC è relativa al singolo attributo, quindi va specificata per ciascun attributo che elenco nell'ORDER BY.

8.6.2 Operatori aggregazione

Sono un'estensione rispetto all'Algebra Relazionale. Essi operano su gruppi di tuple per:

- Effettuare calcoli sull'insieme di tuple
- Verificare condizioni relative all'insieme di tuple

- COUNT - Conta numero di tuple di tabella
- SUM - Somma valori o espressioni di attributi
- MAX - Valore massimo di un attributo di tabella

- MIN - Valore minimo di un attributo di tabella
- AVG - Valore medio di un attributo di tabella
- GROUP BY - Raggruppamento delle tuple in sottogruppi gestiti come estensione delle normali interrogazioni

Come vengono applicati

- Si esegue l'interrogazione sulla base di clausole FROM e WHERE.
- Si applica l'operatore aggregato alla tabella risultato dell'interrogazione.

8.6.3 COUNT

```
SELECT COUNT(< * | [DISTINCT | ALL] ListaAttr >)
```

- DISTINCT numero di valori distinti diversi da NULL di ListaAttr
- ALL numero di valori diversi da NULL di ListaAttr
- COUNT (*) conta numero di righe di una tabella

Se non è specificato nulla, ALL è l'opzione di default.

Esempio Estrarre il numero di impiegati del dipartimento Produzione

```
SELECT COUNT(*) AS impiegati_produzione
FROM Impiegato
WHERE dipart='Produzione'
```

L'operatore aggregato COUNT viene applicato al risultato dell'interrogazione.

COUNT e valori nulli COUNT(*) conta anche le righe con valori NULL, dato che conta le righe totali della tabella.
COUNT(Attributo) non conta le righe NULL.
COUNT(DISTINCT Attributo) non conta le righe NULL e non conta i valori duplicati.

8.6.4 Altri operatori di aggregazione

```
< SUM | MAX | MIN | AVG > (< * | [DISTINCT | ALL] AttrExpr >)
```

Valutano espressioni analizzando tuple del risultato nella query che li contiene. Operano su tuple di tabella prese come insieme di elementi (non su elementi individuali). DISTINCT e ALL si applicano come per il caso di COUNT.

SUM Restituisce la somma dei valori di AttrExpr relativamente alle tuple di tabella specificate nella query. Opera su attributi numerici.

MIN MAX Restituisce il minimo/massimo dei valori di AttrExpr relativamente alle tuple di tabella specificate nella query. Opera su atributi ordinabili (stringhe, numer, ecc.).

AVG Restituisce la media dei valori di AttrExpr relativamente alle tuple di tabella specificate, opera su valori numerici.

8.6.5 Operatori di aggregazione e Target List

SQL impedisce di includere in una stessa target list funzioni aggregate e espressioni al livello di riga.

```
SELECT Cognome, MAX(Stipendio) -- ERRORE
FROM Impiegato
WHERE Dipart = 'Amministrazione'
```

La soluzione è effettuare una query nidificata (spiegata in seguito).

```
SELECT Cognome, Stipendio
FROM Impiegato
WHERE Dipart = 'Amministrazione'
AND Stipendio = (SELECT MAX(I1.Stipendio)
                 FROM Impiegato AS I1
                 WHERE I1.Dipart = 'Amministrazione')
```

8.6.6 GROUP BY

La clausola GROUP BY serve a definire gruppi omogenei di tuple, specificando una o più colonne (di raggruppamento) sulla base della/e quale/i le tuple sono raggruppate per valori uguali.

Esempio Data la seguente tabella estrarre la somma degli stipendi degli impiegati che afferiscono allo stesso dipartimento.

Nome	Cognome	Dipart	Ufficio	Stipendio	Città
Mario	*Rossi*	*Amministrazione*	*10*	*45*	*Milano*
Carlo	**Bianchi**	**Produzione**	**20**	**36**	**Torino**
Giuseppe	*Verdi*	*Amministrazione*	*20*	*40*	*Roma*
Franco	Neri	Distribuzione	16	45	Napoli
Carlo	Rossi	Direzione	14	80	Milano
Lorenzo	Lanzi	Direzione	7	73	Genova
Paola	*Borroni*	*Amministrazione*	*75*	*40*	*Venezia*
Marco	**Franco**	**Produzione**	**20**	**46**	**Roma**

Figura 8.2: Tabella esercizio

```
SELECT Dipart, SUM(Stipendio)
FROM Impiegato
GROUP BY Dipart
```

Come viene eseguita la query? Analizziamola passo per passo:

Esecuzione senza GROUP BY In questo caso mi restituisce tutti gli attributi della SELECT in un'unica tabella.
Con GROUP BY invece vengono raggruppate le righe del risultato come specificato nel GROUP BY (nell'esempio valori Dipart).

Dipart	Stipendio
Amministrazione	45
Amministrazione	40
Amministrazione	40
Produzione	36
Produzione	46
Distribuzione	45
Direzione	80
Direzione	73

Figura 8.3: Tabella con GROUP BY

Applicazione SUM dopo GROUP BY Così facendo l'applicazione dell'operatore SUM avviene su gruppi di tuple con lo stesso nome.

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

Tabella 8.2: Tabella somma stipendi per dipartimento

Il GROUP BY raggruppa sempre gli gli attributi specificati nella clausola GROUP BY con lo stesso nome, permettendo poi di applicare condizioni particolari nella SELECT (per esempio AVG, COUNT, ecc.).

8.6.7 Raggruppamenti e Target List

Importante restrizione Una clausola di proiezione di una query contenente la clausola GROUP BY può solo includere:

1. Una o più colonne se compaiono nella clausola GROUP BY
2. Operatori di aggregazione (COUNT, SUM, AVG, ...)

Nella SELECT deve apparire l'attributo specificato nel GROUP BY, altrimenti la Query risulterà scorretta.

```
SELECT nome, cognome -- nome e' ERRORE!
FROM persone
GROUP BY cognome

SELECT cognome, AVG(Eta) -- Corretta
FROM persone
GROUP BY cognome
```

Tutti gli attributi nella SELECT devono essere sepcificati anche nel GROUP BY.

8.6.8 Condizioni sui Gruppi

Verificano condizioni su gruppi di tuple aggregate con GROUP BY.

- WHERE - Verifica condizioni su tuple individuali
- HAVING - Verifica condizioni (espressioni booleane) su gruppi di tuple (opera su raggruppamenti)

Le condizioni in clausola HAVING sono verificate dopo la creazione dei sottogruppi di tuple specificati da GROUP BY, in pratica le condizioni sono verificate dopo il WHERE e dopo il GROUP BY, mentre il WHERE le applica prima che il GROUP BY effettui i raggruppamenti.

Esempio Trovare i dipartimenti che spendono più di 100 in stipendi.

```
SELECT Dipart, SUM(Stipendio) AS SommaStipendi
FROM Impiegato
GROUP BY Dipart
HAVING SommaStipendi > 100
```

Risolviamo la Query passo per passo.

1. Esecuzione della query senza considerare GROUP BY e operatori aggregati

2. Raggruppamento delle righe del risultato come specificato da GROUP BY

3. Applicazione dell'operatore di aggregazione SUM su Stipendio ai gruppi di righe precedentemente costruiti

4. Selezione dei gruppi risultanti come specificato dalla clausola HAVING

Qui vediamo proprio la differenza di HAVING rispetto a WHERE, dato che WHERE opera su tuple individuali (selezione di tuple prima di GROUP BY), mentre HAVING opera su predicati che operano su gruppi di tuple creati da GROUP BY (selezone raggruppamenti).

Esempio molto utile Andare alla slide N 247 (62 come Pagina PDF) delle slide del prof Slide Prof.

8.6.9 Osservazioni

```
GROUP BY attr1, attr2
-- Equivalente a
GROUP BY attr2, attr1
```

HAVING osservazioni La sintassi permette di definire la clausola HAVING anche senza la clausola GROUP BY. In questo caso l'intero insieme di righe è trattato come un unico raggruppamento, è necessario ricordarsi che per HAVING devono essere usati solo predicati in cui compaiono operatori aggregati.

8.7 Interrogazioni di tipo insiemistico

Permettono di costruire query concatenando due query SQL

- UNION
- INTERSECT
- EXCEPT (differenza)

```
SelectSQL { < UNION/INTERSECT/EXCEPT > [ALL] SelectSQL }
```

I duplicati vengono eliminati (a meno che si usi ALL).

8.7.1 UNION

Restituisce tutte le tuple distinte restituire da almeno una delle sottointerrogazioni a qui è applicata.
L'operatore UNION elimina i duplicati dal risultato, mentre UNION ALL li mantiene.
UNION impone alcune importanti restrizioni sulle interrogazioni su cui opera:

- Le interrogazioni devono restituire lo stesso numero di colonne, e le colonne corrispondenti devono avere lo stesso dominio (non è richiesto che abbiano la stessa lunghezza) o domini compatibili
- La corrispondenza si basa sulla posizione delle colonne, indipendentemente dal loro nome
- Se si usa la clausola ORDER BY questa deve essere usata una sola volta alla fine dell'interrogazione e non alla fine di ogni SELECT

8.7.2 INTERSECT, EXCEPT, MINUS

- L'operatore INTERSECT esegue l'intersezione
- Gli operatori EXCEPT e MINUS eseguono la differenza

Per questi operatori valgono le stesse condizioni di applicabilità viste per l'operatore UNION. Le interrogazioni devono restituire lo stesso numero di clonne e le colonne corrispondenti devono avere lo stesso dominio (non è

richiesto che abbiano la stessa lunghezza) o domini compatibili. La corrispondenza si basa sulla posizione delle colonne.

8.8 Interrogazioni nidificate

Una delle ragioni che rendono SQL un linguaggio potente è la possibilità di esprimere interrogazioni più complesse in termini di interrogazioni più semplici, tramite il meccanismo delle **subqueries** (sottointerrogazioni).
La clausola WHERE di una query (detta query esterna) può infatti contenere un'altra query (detta subquery).
La subquery viene usata per determinare uno o più valori da usare come valori di confronto in un predicato della query esterna.
Le subquery possono essere usate nel FROM per arricchire l'insieme di tuple su cui viene effettuata l'interrogazione principale oppure nel WHERE per filtrare le tuple dell'interrogazione principale.
Nella clausola WHERE possono comparire predicati che confrontano un attributo (o un'espressione sugli attributi) con il risultato di una query SQL:

```
ScalarValue Operator <ANY | ALL > SelectSQL
```

- ANY - Il predicato è vero se almeno una riga restituita dalla query SelectSQL soddisfa il confronto
- ALL - Il predicato è vero se tutte le righe restituite dalla query SelectSQL soddisfano il confronto
- Operator - Uno qualsiasi tra $=, <>, >, <, >=, <=$

La query che appare nella clausola WHERE è detta **query nidificata**.

Esempio Estrarre il cognome degli impiegati che lavorano in dipartimenti con sede a Milano.
*(*IN = ANY).

```
SELECT Cognome
FROM Impiegato
WHERE dipart ANY (SELECT nome
                  FROM Dip
```

```
                    WHERE Citta = 'Milano')

-- Equivale a

SELECT cognome
FROM Impiegato JOIN Dip
  ON dip.Nome = impiegato.Dipart
  WHERE Dip.Citta = 'Milano'
```

La forma ndificata è meno dichiarativa, ma talvolta più leggibile (richiede meno variabili). La forma piana e quella nidificata possono essere combinate.

Osservazione Le sottointerrogazioni non possono contenere operatori insiemistici (si fanno solo a livello esterno), la limitazione non è significativa.

8.8.1 Operatori di confronto

Gli operatori di confronto $=, <$, ... si possono usare solo se la subquery restituisce non più di una tupla (detta subquery "scalare").
Se la subquery può restituire più di un valore allora si devono usare le forme:

- ANY - la relazione $=$ vale per almeno uno dei valori
- ALL - la relazione $>$ vale per tutti i valori

Ricordiamo che la forma ANY $=$ IN.

8.8.2 MAX e MIN in subquery

Gli operatori aggregati max e min possono essere usati in una subquery.
Anche in questo caso è necessario usare ANY o ALL se vengono restituiti più valori:

- ANY la relazione vale per aleno uno dei valori
- ALL la relazione vale per tutti i valori

É inoltre possibile selezionar epiù di una colonna tramite una sottointerrogazione, in tal caso è necessario apporre delle parentesi alla lista delle colonne a sinistra del'operatore di confronto (Costruttore di tupla).

```
SELECT Cognome, Nome
FROM Impiegato
WHERE (Cognome, Nome) IN (SELECT Cognome, Nome
```

```
                    FROM Impiegato
                    WHERE Dipart = 'Ricerca')
```

Interpretazione semplice L'interrogazione viene eseguita prima dell'interrogazione di esterna. Il risultato può essere salvato in una tabella temporanea, in modo da eseguire una volta sola l'interrogazione nidificata.

8.8.3 Subquery Correlate

Query correlate A volte la query più interna fa riferimento a una variabile definita nella query più esterna, in questi casi l'interpretazione semplice non va più bene. Riconsideriamo l'interpretazione:

1. Costruisamo prima il prodotto cartesiamo delle tabelle coinvolte dalla clausola FROM e poi a ciascuna riga si applica la clausola WHERE,

2. Poi le subqueries sono eseguite ripetutamente per ogni tupla candidata considerata nella valutazione della query esterna.

L'interrogazione interna viene eseguita una volta per ciascuna tupla dell'interrogazione esterna.
Due sono i concetti alla base dalla correlazione:

- Una subquery correlta fa riferimento ad un attributo selezionato dalla query esterna

- Se una subquery seleziona tuple della stessa tabella riferita dalla query esterna è necessario definire un alias per tale tabella della query esterna

La subquery deve usare alias per riferire i valori di attributo nelle tuple candidate nella query principale.

Visibilità delle variabili Una variabile è usabile solo nell'ambito della query in cui è definita o nell'almbito di una query nidificata (a qualsiasi livello) all'interno di essa. Se una interrogazione possiede delle interrogazioni nidificate allo stesso livello le variabili introdotto nella clausola FROM di una query NON possono essere usate dall'altra query.
Le subquery correlate sono in grado di restituire mano a mano i risultati interessati della subquery alla query principale, per poi riprendere a calcolare i vari risultati nella subquery, quindi non viene interamente calcolata la subqeury e poi restituita alla query, ma vengono mano a mano calcolati i risultati.

8.8.4 EXISTS e NOT EXISTS

EXISTS e NOT EXISTS ammettono come parametro le query nidificate. L'operatore logico EXISTS restituisce il valore booleano True se la subquery restituisce almeno una tupla, altrimenti restituisce false, viceversa l'operatore NOT EXISTS. La valutazione di predicati con questi due operatori non restituisce mai il valore Unkown (NULL).

Esempio Query: dipartimenti con almeno un programmatore.

```
SELECT Nome
FROM Dip AS S -- Alias per la subquery
WHERE EXISTS (SELECT *
              FROM Impiegato
              WHERE Mansione = 'Programmatore'
              AND Dipart = S.Nome) -- S.Nome fa riferimento al
```

Il risultato della subquery dipende dal dipart specifico (selezionato nella query esterna) e quindi diventa: Per ogni tupla del blocco esterno, considera il valore di S.nome e risolvi la query innestata.
É spesso possibile ricondursi alla forma "piatta" (senza quindi subquery), ma la cosa non è sempre così ovvia e soprattutto NON è sempre possibilie! Alcune Query possono essere espresse solo tramite subquery. Per quanto riguarda l'esempio precedente:

```
SELECT S.Nome
FROM Dip AS S, Impiegato AS I
WHERE S.Nome = I.Dipart AND I.Mansione = 'Programmatore'
```

8.8.5 Note finali subquery

- In una subquery non è possibile usare operatori insiemistici (UNION, INTERSECT, EXCEPT)
- una subquery può comparire solo come operando destro in un predicato
- In una subquery scalare (cioè che restituisce un solo valore) non ci possono essere le clausole GROUP BY e HAVING
- Le subquery possono comparire anche nelle espressioni della clausola select (non solo nella clausola WHERE) e nelle clausole HAVING

8.9 Viste in SQL

In SQL è possibile definire delle viste alternative degli stessi dati. Una vista (**view**) è una relazione virtuale attraverso cui è possibile vedere dati memorizzati nelle relazioni reali (dette **di base**). Una vista non contiene tuple, ma può essere quasi usata come una relazione di base. Una vista è definita da un'interrogazione su una o più relazioni di base oppure su altre viste. Una vista è materializzata eseguendo l'interrogazione che la definisce.
Il meccanismo delle viste è utile per semplificare l'accesso ai dati e garantirne la privatezza.

8.9.1 Creazione di una vista

Si definiscono associando un nome ed una lista di attributi al risultato di un'interrogazione.

```
CREATE VIEW NomeVista [(ListaAttributi)] AS query
```

- NomeVista è il nome della vista che viene creata, deve essere univoco nel database, unico quindi rispetto sia a nomi di relazioni che di viste definite dallo stesso utente.
- Query è l'interrogazione di definizione della vista
- ListaAttributi è una lista di nomi da assegnare alle colonne della vista, tale specifica non è obbligatoria, tranne nel caso in cui l'interrogazione contenga nella clausola di proiezione funzioni di gruppo e/o espressioni

Esempio Si vuole creare una vista costituita da un sottoinsieme delle tuple della relazione Impiegati, più precisamente la vista deve elencare le colonne Codice, Nome e Mansione degli impiegati del 'produzione'.

```
CREATE VIEW Imp10 AS
SELECT Codice, Nome, Mansione
FROM Impiegati WHERE dipart = 'produzione'
```

I nomi delle colonne della vista Imp10 sono rispettivamente: Codice, Nome, Mansione.
Su una vista si possono eseguire sia interrogazioni che modifiche (con alcune importanti restrizioni).

8.9.2 Uso di espressioni e di funzioni

É possibile definire viste tramite interrogazioni che contengono espressioni o funzioni, queste espressioni appaiono del tutto simili alle altre colonne della vista, ma il loro valore è calcolato dalle relaizoni di base ogni volta che la vista è materializzata.
Tali colonne sono spesso chiamate colonne virtuali, **è obbligatorio specificare un nome nella vista per tali colonne**.
Quando si specifica una vista il sistema sostituisce nell'interrogazione la vista con la sua definizione.