SOMMARIO

[CAPITOLO 1. Dati 5](#_Toc140697101)

[Dato 5](#_Toc140697102)

[Schema di riferimento 5](#_Toc140697103)

[Informazione 5](#_Toc140697104)

[Tipo 5](#_Toc140697105)

[Maschera 5](#_Toc140697106)

[Dominio 5](#_Toc140697107)

[CAPITOLO 2. Database 6](#_Toc140697108)

[Contenuto 6](#_Toc140697109)

[DBMS – Data Base Management System 6](#_Toc140697110)

[Rappresentazione 6](#_Toc140697111)

[Indipendenza logica 6](#_Toc140697112)

[Indipendenza fisica 6](#_Toc140697113)

[CAPITOLO 3. Modellazione concettuale 7](#_Toc140697114)

[Definizione 7](#_Toc140697115)

[Dominio di un attributo 7](#_Toc140697116)

[Intensione ed estensione 7](#_Toc140697117)

[Attributo univoco 7](#_Toc140697118)

[Chiave 7](#_Toc140697119)

[Superchiave minimale 8](#_Toc140697120)

[Chiavi naturali e chiavi artificiali 8](#_Toc140697121)

[Chiave naturale 8](#_Toc140697122)

[Chiave artificiale 8](#_Toc140697123)

[Associazioni 8](#_Toc140697124)

[Associazioni ternarie 8](#_Toc140697125)

[Associazione totale 9](#_Toc140697126)

[Vincolo d’integrità 9](#_Toc140697127)

[Entità debole 9](#_Toc140697128)

[Associazione part of 9](#_Toc140697129)

[Associazioni ricorsive 9](#_Toc140697130)

[valori mancanti 9](#_Toc140697131)

[Logica trivalente 10](#_Toc140697132)

[CAPITOLO 4. Modelli e/r estesi (extended e/r – ee/r) 11](#_Toc140697133)

[Costrutto di specializzazione e generalizzazione 11](#_Toc140697134)

[CAPITOLO 5. Vincoli 12](#_Toc140697135)

[Vincoli di base 12](#_Toc140697136)

[CAPITOLO 6. Progettazione del software 13](#_Toc140697137)

[Modello waterfall e agile 13](#_Toc140697138)

[Waterfall 13](#_Toc140697139)

[Agile 14](#_Toc140697140)

[CAPITOLO 7. Modello relazionale 15](#_Toc140697141)

[Relazione 15](#_Toc140697142)

[Relazione matematica 15](#_Toc140697143)

[Prodotto cartesiano etichettato 15](#_Toc140697144)

[Multinsieme (Bag) 15](#_Toc140697145)

[Insieme di supporto 15](#_Toc140697146)

[Funzione di molteplicità 15](#_Toc140697147)

[Schema relazionale 15](#_Toc140697148)

[Tabella 16](#_Toc140697149)

[Traduzione degli attributi 16](#_Toc140697150)

[Traduzione delle specializzazioni 16](#_Toc140697151)

[Traduzione delle associazioni 17](#_Toc140697152)

[Reificazione 18](#_Toc140697153)

[CAPITOLO 8. Algebra relazionale 19](#_Toc140697154)

[Definizione 19](#_Toc140697155)

[Espressione di algebra relazionale 19](#_Toc140697156)

[Operatori 19](#_Toc140697157)

[Operatori unari 19](#_Toc140697158)

[Operatori binari 19](#_Toc140697159)

[Albero di interrogazione 20](#_Toc140697160)

[Join 20](#_Toc140697161)

[Riscrittura algebrica 20](#_Toc140697162)

[Operatore 20](#_Toc140697163)

[Query innestate 21](#_Toc140697164)

[Query innestate correlata 21](#_Toc140697165)

[Query gerarchiche 21](#_Toc140697166)

[CAPITOLO 9. SQL 23](#_Toc140697167)

[Definizione 23](#_Toc140697168)

[Interoperabilità 23](#_Toc140697169)

[Differenze con l’algebra relazionale 23](#_Toc140697170)

[Estendere sql 24](#_Toc140697171)

[Conflitto di impedenza 24](#_Toc140697172)

[CAPITOLO 10. Pl/sql 25](#_Toc140697173)

[Vista 25](#_Toc140697174)

[Vista materializzata 25](#_Toc140697175)

[Trigger 25](#_Toc140697176)

[Eventi 25](#_Toc140697177)

[CAPITOLO 11. Transazioni 26](#_Toc140697178)

[Proprietà delle transazioni 26](#_Toc140697179)

[Problemi delle transazioni 26](#_Toc140697180)

[Livelli di isolamento 27](#_Toc140697181)

[Protocolli di gestione della concorrenza 27](#_Toc140697182)

[CAPITOLO 12. Teoria della normalizzazione 28](#_Toc140697183)

[Dipendenza funzionale 28](#_Toc140697184)

[Assiomi di Armstrong 28](#_Toc140697185)

[Chiusura di un insieme di dipendenze funzionali 29](#_Toc140697186)

[Superchiave 29](#_Toc140697187)

[Forme normali 29](#_Toc140697188)

[Operazioni in caso di violazione di integrità referenziale 29](#_Toc140697189)

# Dati

## Dato

Un ***dato*** è un’informazione di qualsiasi *tipo primitivo*.

## Schema di riferimento

Uno ***schema di riferimento*** indica il *contesto* del dato.

Un dato senza schema di riferimento non veicola l’informazione.

## Informazione

Il dato, aggiunto con lo schema di riferimento, costituisce l’***informazione***.

## Tipo

I dati si possono classificare in:

* ***Strutturati***: sono conformi ad uno *schema*, ovvero sono nella forma [attributo = valore];
* ***Non strutturati***: testo libero, immagini, video, suoni;
* ***Semi-strutturati***: come esempio si possono prendere file come XML, JSON, HTML.

## Maschera

La ***maschera*** definisce come viene visualizzato il dato.

## Dominio

Con ***dominio*** si intende l’*insieme di definizione* del dato. Esso dipende assolutamente dal contesto.

# Database

## Contenuto

In un database vi sono:

* ***Dati***;
* ***Metadati***: descrizione dei dati.

Di solito, i database sono sviluppati per un *contesto specifico*.

## DBMS – Data Base Management System

Il ***DBMS*** (***Data Base Management System***) è il sistema di *gestione database*, ovvero un software atto a gestire questi dati: è tramite lui che è possibile leggere/scrivere dati sul database.

Gli utenti (esseri umani, app, ecc.) utilizzano il DBMS per interfacciarsi col database.

Il DBMS ha anche il compito di preservare la ***coerenza*** dei dati.

Con questo sistema si garantisce:

* ***Consistenza*** dei dati;
* ***Affidabilità***: in caso di malfunzionamenti del database, le richieste verranno annullate;
* ***Multiutenza***: accessi multipli in scrittura senza perdita di coerenza;
* ***Persistenza dei dati***: possibilità di salvare i dati per lungo tempo.

Per scrivere si usa il linguaggio **SQL** (**Structured Query Language**).

## Rappresentazione

I database si rappresentano secondo uno schema ***ANSI/SPARC***, con livelli di astrazione differenti. Tali livelli sono:

* ***Vista logica***: quali dati vedono le app.
* ***Logico***: come i dati sono organizzati e come sono relazionati.
* ***Fisico***: come vengono memorizzati su disco.

Vista logica e livello logico fanno parte dell’***indipendenza logica***, mentre il livello fisico appartiene all’***indipendenza fisica***.

### Indipendenza logica

È possibile cambiare ciò che le app vedono, ma non è garantito al 100%.

### Indipendenza fisica

È possibile cambiare le relazioni che i dati hanno tra loro, senza modificare lo schema.

# Modellazione concettuale

## Definizione

La ***modellazione concettuale*** definisce i dati e come essi sono relazionati tra loro. Viene realizzato mediante lo ***schema******E/R*** (***Entity/Relationship***).

Prima cosa di un diagramma E/R è l’***entità***, realizzata mediante un *rettangolo*, con scritto al centro il *nome* dell’entità: significa che nel mio database avrò una certa entità con un certo nome, con certi dati (***attributi***) che dipendono da un contesto; gli attributi sono indicati con un *ovale*, al cui interno viene indicato il rispettivo nome.

Gli attributi si dividono in:

* ***Atomici/univoci***: hanno un unico valore.
* ***Strutturati/multi-valore***: hanno più di un valore; vengono indicati con un *doppio ovale*.
* ***Totale***: attributo che ha valore per la totalità della popolazione di riferimento.
* ***Parziale***: attributo che ha valore per la parzialità della popolazione di riferimento.
* ***Derivabile***: attributo che deriva da un altro; si indica con un *ovale tratteggiato*.
* ***Statico***: attributo che non cambia nel tempo.
* ***Dinamico***: attributo che cambia nel tempo.

Non sono mai presenti attributi non strutturati.

## Dominio di un attributo

Il ***dominio*** di un attributo consiste nell’*insieme di valori* che l’attributo può assumere.

## Intensione ed estensione

Ogni entità ha:

* ***Intensione***, che corrisponde allo schema.
* ***Estensione*** (o ***stato***), che corrisponde alle *istanze* dell’entità.

## Attributo univoco

Posso usare un ***attributo univoco*** per *identificare* un’entità.

## Chiave

Una ***chiave*** gode di tre attributi:

* ***Univocità***: non deve capitare che appaia più di una volta in più istanze.
* ***Totalità***
* ***Costanza***

## Superchiave minimale

La ***superchiave* *minimale*** (oppure ***chiavi* *candidate***) è una superchiave con il *numero minimo di attributi* e corrisponde ad una serie di attributi che possono fungere da ***chiavi primarie***.

## Chiavi naturali e chiavi artificiali

### Chiave naturale

Si dice ***chiave naturale*** una chiave scelta dagli *attributi presenti nell’entità*.

### Chiave artificiale

Si dice ***chiave artificiale*** un attributo *creato appositamente* in modo da poter fungere da chiave primaria; ciò accade quando negli attributi dell’entità, nessuno può fungere da chiave primaria.

Tuttavia, l’effetto collaterale di usare chiavi artificiali è la *perdita di univocità* della chiave (esempio: entità cliente con codice cliente chiave primaria, contatore; posso creare più volte lo stesso cliente, in quanto il codice varierà).

## Associazioni

Le ***associazioni*** collegano *due o più entità*. I tipi di molteplicità delle associazioni sono:

* ***Biunivoca***: associazione a uno a uno; si indica con .
* ***Multi-valore***: associazione uno a molti; si indica con e la chiave è dal lato della .
* ***Univoca***: associazione molti a uno; si indica con .
* ***Multi-valore* *doppia***: associazione molti a molti; si indica con ; con associazioni ternarie, si indica con .

L’associazione ha dei propri attributi, e ogni istanza di un’associazione formata dalla *coppia di chiavi delle due entità*, più eventuali attributi dell’associazione. Anche le associazioni necessitano di una **chiave**.

### Associazioni ternarie

Le ***associazioni ternarie*** si utilizzano quando si verificano dipendenze di grado 3.

### Associazione totale

L’**associazione totale** si indica con *doppia linea*.

## Vincolo d’integrità

Il ***vincolo d’integrità*** vuole che tutte le entità e tutte le associazioni debbano avere una *chiave primaria*.

Inoltre, tutti gli attributi devono essere *atomici*. Un esempio può essere la memorizzazione di un indirizzo, che si divide in via, numero civico, provincia.

## Entità debole

Un’***entità debole*** si ha quando:

* Si ha una *dipendenza di esistenza* tra l’entità debole e l’entità forte.
* Ci dev’essere una relazione .
* Deve avere una *chiave parziale o debole*, ovvero un attributo chiave che da solo non riesce a fungere da chiave dell’associazione.

Si rappresenta con *doppio rettangolo* e l’associazione collegata ad un’entità debole si chiama ***associazione identificante*** (rappresentata con *doppio rombo*); la chiave dell’entità debole si indica con una *linea tratteggiata*.

## Associazione part of

Le ***associazioni part-of*** sono con relazione , e si dividono in:

* ***Aggregazione***: si ha quando vi sono entità forti, quindi con chiavi forti, senza dipendenza di esistenza.
* ***Composizione***: si ha quando vi è un’entità debole, quindi con chiave debole, con dipendenza di esistenza.

## Associazioni ricorsive

Nelle ***associazioni ricorsive*** riguardano istanze della *stessa entità*: l’entità di partenza e quella di arrivo sono *uguali*.

## valori mancanti

I ***valori mancanti*** differiscono nelle seguenti casistiche:

* ***Non disponibile***: attributo *facoltativo* (attributo che non tutte le istanze dell’entità hanno).
* ***Non applicabile***: attributo che non ha alcun significato con quell’entità.
* ***Sconosciuto***: attributo che se esiste non lo conosciamo, oppure non esiste affatto.

È consigliabile non inserire valori mancanti.

## Logica trivalente

La ***logica trivalente*** sussiste quando, oltre ai valori *vero* o *falso*, abbiamo anche il valore ***NULL***. Tale valore non vale 0, “”, ‘NULL’.

Per consentire il confronto con valori NULL, esistono due costrutti:

* ***is***
* ***is not***

# Modelli e/r estesi (extended e/r – ee/r)

## Costrutto di specializzazione e generalizzazione

Il costrutto di ***specializzazione*** (o ***generalizzazione***) dipende dalla relazione *is\_a*, ovvero corrisponderebbe all’operazione di *inclusione* insiemistica. Tale relazione viene indicata con un *cerchio* (che si legge appunto *is\_a*); Un esempio può essere il seguente:

Immagine che contiene diagramma

Descrizione generata automaticamente

Il simbolo *semicerchio*, sovrapposto sulle linee in figura, caratterizza l’operazione di *sottoinsieme*.

Se nei due cerchi viene inserita:

* “***O***” (*overlap* – *sovrapposizione*): una persona può essere sia studente che lavoratore.
* “***D***” (*disjoint* – *disgiunto*): una persona è studente oppure lavoratore.
* “***U***” (*union* – *unione*): l’entità viene definita ***selettiva***, ovvero tra più entità madri, eredita gli attributi della sola entità madre selezionata.

La chiave dell’entità selettiva viene definita ***surrogata***, il quale è una *chiave artificiale*; ciò succede a causa dell’inesistenza di chiavi in comune tra le entità madri.

Se, invece, vi è totalità da persona, significa che vi sono esclusivamente i dati di studenti e lavoratori.

Le specializzazioni aiutano ad eliminare *ridondanze* e *valori mancanti*.

Si definisce specializzazione o generalizzazione in base al *verso* in cui si legge. Nell’esempio precedente:

* Se una “persona” si specializza in studente/lavoratore, allora si ha la *specializzazione*.
* Se uno “studente/lavoratore” va verso “persona”, allora si ha la *generalizzazione*.

# Vincoli

Un ***vincolo*** si divide in:

* ***Statico***: immutabile nel tempo.
* ***Dinamico***: può variare nel tempo.

Si ha un ***vincolo estensionale*** quando il vincolo riguarda *estensioni di associazioni*.

Si ha un ***vincolo intensionale*** quando il vincolo riguarda i *domini* sullo schema.

I ***vincoli*** si distinguono anche in: *vincoli di business* e *vincoli dinamici*.

Per la gestione dei vincoli, usiamo le ***asserzioni***. Un esempio è:

create assertion troppeSquadre as check not exists (select idPartita from Gioca group by idPartita having count(\*) <> 2);

## Vincoli di base

Vincolo **:PRIMARY KEY**: vedere file [README.md](file:///Users/roccodelprete/Library/CloudStorage/OneDrive-Uniparthenope/basi%20di%20dati/README.md) (sezione “VINCOLI DI BASE”).

# Progettazione del software

Nella ***progettazione software*** abbiamo due aspetti:

* ***Real Word***: divisa in configurazioni, rappresenta il mondo reale.
* ***Information system***: il modello.

Si ha ***ambiguità*** quando ad uno stato dell’information system corrispondono *due stati diversi del real word*; al contrario, si ha ***ridondanza*** quando ad uno stato del real word *corrispondono due stati dell’information system*.

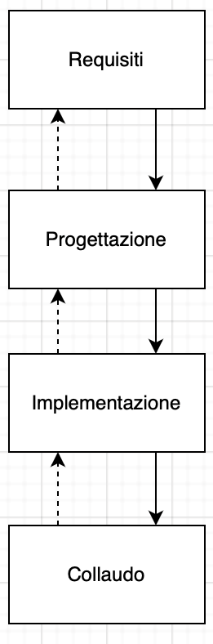
Si ha uno ***schema perfetto*** quando ad ogni stato del real world *corrisponde uno ed un solo stato* dell’information system e viceversa.

## Modello waterfall e agile

Nella progettazione del software si hanno due modelli: ***waterfall*** (*cascata*) e ***agile***.

### Waterfall

Il ***modello waterfall*** segue il seguente schema:



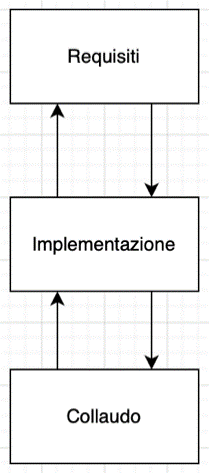
Le frecce sono tratteggiate perché si può tornare indietro *solo* in caso di eccezioni.

Il modello waterfall prevede che il prodotto, al rilascio, abbia necessità di assistenza in vie eccezionali; infatti, garantisce una qualità maggiore.

Tuttavia, il tempo di rilascio è decisamente lungo e i costi sono maggiori.

### Agile

Il ***modello agile*** segue il seguente schema:



Il modello agile non prevede una progettazione, infatti è molto probabile che un prodotto che segue questo schema è in continua assistenza, ed il numero di persone partecipanti al progetto è più basso.

Tuttavia, i tempi di rilascio sono decisamente minori.

Il prodotto realizzato seguendo questo modello prende il nome di ***prototipo***.

È preferibile utilizzare questo schema quando i requisiti sono molto variabili.

# Modello relazionale

Il ***modello relazionale*** è composto da ***relazioni*** e si ottiene dal modello concettuale.

Il processo di passaggio da modello concettuale a modello logico si definisce ***traduzione***.

## Relazione

Una ***relazione*** è definita sui ***multinsiemi***.

Essa è diversa da una *relazione matematica* perché:

* sono ammessi valori ripetuti;
* le coppie possono essere scambiate.

Viene rappresentata attraverso una ***tabella***.

### Relazione matematica

La ***relazione matematica*** è un *sottoinsieme* del ***prodotto cartesiano*** tra due o più insiemi.

### Prodotto cartesiano etichettato

Il ***prodotto cartesiano etichettato*** è dato dall’insieme delle ***coppie etichettate***, in cui ciascun elemento appartiene agli insiemi presi in considerazione.

## Multinsieme (Bag)

Un ***multinsieme*** è definito da un ***insieme di supporto*** e da una ***funzione di molteplicità***.

### Insieme di supporto

Un ***insieme di supporto*** è l’*insieme di elementi* contenuti nel multinsieme.

### Funzione di molteplicità

La ***funzione di molteplicità*** indica il *numero* di volte in cui un elemento si ripete nel multinsieme.

## Schema relazionale

Uno ***schema relazionale*** è definito dalla terna costituita da:

Dove:

* indica il numero di ***relazioni***;
* indica il numero di ***vincoli***;
* indica il numero di ***dipendenze*** ***funzionali***.

La *rappresentazione* di uno schema relazionale avviene attraverso una ***tabella***.

### Tabella

Una ***tabella*** ha anch’essa uno *schema*, ovvero la *rappresentazione grafica* di una relazione, che corrisponde allo schema dell’entità di riferimento.

Ogni colonna della tabella viene definita ***campo*** (o ***attributi***), mentre ogni riga viene definita ***tupla*** (o ***istanza***).

## Traduzione degli attributi

Gli *attributi strutturati* vengono inseriti in tabella eliminandone la struttura; un esempio è l’indirizzo, composto da via, CAP e città: in tabella vanno memorizzati solo i campi via, CAP e città.

Gli *attributi derivabili* non vanno memorizzati in tabella.

Gli *attributi multi-valore* vanno memorizzati creando una nuova tabella avente come campi:

* la ***chiave esterna*** dell’entità l’attributo di cui è multi-valore;
* i *valori* che esso può assumere.

Inoltre, bisogna collegare con una *freccia* la chiave esterna alla *chiave primaria* che rappresenta.

Anche queste tabelle hanno bisogno di una *chiave primaria*, che generalmente può essere la coppia ***chiave esterna – valore***.

## Traduzione delle specializzazioni

Le specializzazioni vengono tradotte in una tabella, utilizzando varie tecniche.

#### Tecnica con discriminatore

La tabella prevede i seguenti campi:

* gli attributi della specializzazione;
* gli attributi di ogni singola istanza della specializzazione
* un discriminatore che serve a distinguere le singole istanze della specializzazione, andando ad indicare a quale entità figlia appartiene; esso è obbligatorio.

Può essere rappresentato come:

* singolo campo, avente come valori un identificativo per ogni istanza della specializzazione;
* campi multipli, di cardinalità pari al numero di istanze della specializzazione, avente come valori un tipo booleano.

Tuttavia, questi due tipi di rappresentazioni sono eccessivamente vulnerabili agli errori.

Soluzione a questa problematica, è il partizionamento orizzontale.

#### Partizionamento orizzontale

Il ***partizionamento orizzontale*** prevede la creazione di tante tabelle quante sono le istanze della specializzazione, dove ogni tabella ha come campi:

* gli attributi della specializzazione (inseriti come primi);
* gli attributi di ogni istanza della specializzazione.

Aspetto negativo, in questa rappresentazione, è la ridondanza dei dati.

Tuttavia, se ho associazioni sulle classi figlie, il partizionamento orizzontale permette di mantenere tali associazioni, ma, se ho un’associazione con la classe madre, con questa tecnica l’associazione si perde.

#### Partizionamento verticale

Il ***partizionamento verticale*** prevede la creazione di tante tabelle quante sono le istanze della classe madre, più la tabella per la classe madre; ogni tabella ha i propri campi.

La chiave primaria dell’entità madre viene portata nelle varie tabelle delle entità figlie, diventando chiave esterna, ed anche *chiave primaria* di ogni entità figlia.

#### Partizionamento – tipo unione

Con il tipo ***unione*** vi è un tipo di partizionamento leggermente differente: la chiave primaria dell’entità figlia viene portata nelle entità madri come chiave esterna, ma non assume la funzione di chiave primaria.

## Traduzione delle associazioni

La ***traduzione delle associazioni*** avviene mediante ***tabella***, i cui campi corrispondono alle *chiavi* delle entità a cui sono associati, più altri eventuali attributi. Inoltre, anche la tabella delle associazioni necessita di una ***chiave***.

Le associazioni permettono di risparmiare l’aggiunta di una tabella portando il campo chiave dell’entità con cardinalità nell’entità con molteplicità ; la chiave che verrà copiata in un’entità prenderà il nome di ***foreign key*** (***chiave esterna***) e rappresenta un *puntatore logico* all’entità di riferimento.

Analogamente succede nelle associazioni , ovvero si porta la chiave primaria di un’entità come chiave esterna dell’entità a cui è associata, soltanto che bisogna precisare che la chiave esterna deve essere ***univoca***.

Nelle associazioni la traduzione prevede la creazione di un’ulteriore tabella, definita ***tabella di transizione***, perché le chiavi primaria *transitano* in questa nuova tabella. Inoltre, nella tabella di transizione la chiave primaria corrisponde alla *coppia* composta dalle due *foreign keys*.

Gli eventuali attributi dell’associazione vanno inseriti nella tabella di transizione.

## Reificazione

Vi sono scuole di pensiero che affermano: un’associazione non deve avere attributi; nel caso li abbia, essa dev’essere trasformata in entità.

Il processo che prevede la trasformazione di un’associazione in un’entità viene definito ***reificazione***.

# Algebra relazionale

## Definizione

L’***algebra relazionale*** è un linguaggio:

* ***Chiuso***: operano su ***relazioni*** e producono ***relazioni***.
* ***Procedurale***: le operazioni seguono un ***ordine***.

## Espressione di algebra relazionale

Un’***espressione di algebra relazionale*** rappresenta un’*interrogazione* (***query***) e si ottiene applicando gli *operatori delle relazioni***.**

L’***ordine*** di esecuzione di un’espressione di algebra relazionale è *dall’interno verso l’esterno*.

## Operatori

Gli ***operatori*** si dividono in ***unari*** e ***binari***.

### Operatori unari

Degli ***operatori*** ***unari*** fanno parte:

* (***restrizione***): si applica una condizione ***restrizione***, ovvero si *selezionano* le istanze (righe) di una tabella. Il risultato sarà una tabella con *schema identico,* ma con stato composto dalle *righe selezionati*.
* (***proiezione***): a differenza della proiezione, necessita di una *lista di selezione*: una lista di *attributi* dove si selezionano i *campi* della tabella (non le righe come nella restrizione). Il risultato sarà una tabella avente come:
  + *Campi*: campi indicati nella lista degli attributi.
  + *Righe*: valori del campo indicato presente in tabella, *senza duplicati*.
* (***ridenominazione***): permette di ***rinominare*** un campo di una tabella. Il risultato sarà una tabella identica alla precedente, ma col campo rinominato.

### Operatori binari

Degli ***operatori*** ***binari*** fa parte il (***prodotto cartesiano***), che consiste nella *concatenazione* degli attributi delle due tabelle.

Altro operatore binario è l’operatore ***join*** (), che è molto simile al prodotto cartesiano, solo che è meno complesso di esso. Inoltre, nella ***natural join***, i campi duplicati non faranno parte del risultato.

## Albero di interrogazione

L’***albero di interrogazione***è formato dagli operatori, che compongono i *nodi* dell’albero.

## Join

#### Inner join

Del gruppo degli ***inner join*** fanno parte:

* ***Natural join***: la condizione di join è *implicita*.
* ***Cross join***
* ***Equi join ()***: la condizione di join è *esplicita*; gli attributi omonimi vengono ripetuti e non devono avere obbligatoriamente stesso nome.
* ***Theta join ()***
* ***Left join***: effettua la join e proietta tutti gli attributi dell'entità sinistra.
* ***Right join***: effettua la join e proietta tutti gli attributi dell'entità destra.

#### Outer join

Del gruppo ***outer join*** fanno parte:

* ***Left outer join***: ottengo una tabella costituita dalla tabella risultante della join insieme alle tuple della tabella di sinistra che non rispettano le condizioni di join (la tabella finale ha anche i campi della tabella di destra → valori NULL).
* ***Right outer join***: ottengo una tabella costituita dalla tabella risultante della join insieme alle tuple della tabella di destra che non rispettano le condizioni di join (la tabella finale ha anche i campi della tabella di sinistra → valori NULL).
* ***Outer join***: ottengo una tabella costituita dalla tabella risultante della join insieme alle tuple della tabella di sinistra che non rispettano le condizioni di join insieme alle tuple della tabella di destra che non rispettano le condizioni di join.
* ***Anti-join***: restituisce le tuple *non associate* alla join.

## Riscrittura algebrica

La ***riscrittura algebrica*** consiste nel riscrivere una query in modo da ottenere un risultato più efficiente, utilizzando un ***approccio greedy***.

Per attuare ciò necessitiamo di certe proprietà:

* ***Distributività***: se ho un operatore di *restrizione* applicato ad una *giunzione*, esso corrisponde ad una *giunzione delle restrizioni*.
* ***Associatività***: le join godono di proprietà *associativa*, quindi l’ordine di parentesizzazione può essere variabile. Consiglio: eseguire la parentesizzazione tra tabelle di *grandezza* *minore*.

## Operatore

L’operatore viene definito ***operatore di funzioni aggregate***. Le funzioni sono:

* ***Min***: ritorna il *valore minimo* di un dato campo indicato. È possibile rinominare le colonne.
* ***Sum***
* ***Avg*** (*media)*
* ***Max***: ritorna il *valore massimo* di un dato campo indicato.
* ***Count***

Alla destra dell’operatore gamma si indicano anche gli ***attributi di raggruppamento***. Quindi, il risultato di una query con operatore gamma sarà uno schema formato da:

* Risultato delle funzioni aggregate indicate.
* Attributi di raggruppamento.

## Query innestate

Una query si definisce ***innestata*** se il *valore di confronto* è il risultato di un’altra query.

### Query innestate correlata

Nelle ***query innestate correlate*** i risultati sono ***correlati*** tra loro.

Esempi:

-- nome e cognome di tutti gli studenti che hanno un perfetto coetaneo (stessa data di nascita)

select \*

from Studente S1

where exists (

select dataNascita

from Studente S2

where S1.matricola <> S2.matricola and S1.dataNascita = S2.dataNascita

)

-- trovare gli studenti che hanno superato gli stessi esami di Mario Rossi (123)

select distinct matricola

from EsamiSuperati E1

where not exists (

select \*

from EsamiSuperati E2

where matricola = '123' and not exists (

select \*

from EsamiSuperati E3

where E3.matricola <> E1.matricola and E2.nomeEsame <> E3.nomeEsame

)

)

### Query gerarchiche

Esempi:

-- gerarchia impiegati di ogni livello

select \*

from impiegato

connect by prior cf = cf\_supervisore

start with cf\_supervisore is null

Se nell’esempio precedente si vuole iniziare da un supervisore esistente:

select \*

from impiegato

connect by prior cf = cf\_supervisore

start no cycle with cf\_supervisore = ‘111’

# SQL

Quando si parla di ***SQL***, la proprietà cardine è la ***coerenza dei dati***, ovvero che seguono uno *schema* preciso.

I sistemi ***NO-SQL*** sono sistemi *schemaless*, ovvero che non seguono uno *schema*.

## Definizione

SQL è acronimo di ***Structured Query Language***, ovvero un linguaggio di *interrogazione*, ma non solo per tale funzionalità.

Si dice che SQL è:

* ***DDL*** (*Data Definition Language*): di questo gruppo fanno parte i comandi:
  + ***CREATE***
  + ***ALTER***
  + ***DROP***

Questi comandi modificano lo *schema* di una tabella.

* ***DML*** (*Data Manipulation Language*): di questo gruppo fanno parte i comandi:
  + ***INSERT***
  + ***UPDATE***
  + ***DELETE***

Questi comandi modificano le *tuple* di una tabella.

* ***QL*** (*Query Language*)
* ***DCL*** (*Data Control Language*)
* ***VDL*** (*View Definition Language*)

L’SQL segue uno standard.

## Interoperabilità

L’***interoperabilità*** è garantita dallo standard di SQL e permette di scegliere *qualsiasi motore di database*, senza bisogno di dover cambiare nessun valore del database già presente.

## Differenze con l’algebra relazionale

Le differenze tra SQL e algebra relazionale sono:

1. L’SQL è ***dichiarativo***, mentre l’algebra relazionale è ***procedurale***: l’SQL non segue un ordine di esecuzione della query, mentre l’algebra relazionale fornisce risultati diversi in base all’ordine della query.
2. L’SQL usa la logica dei ***bag***, ovvero ammette elementi ripetuti, mentre l’algebra relazionale usa la logica ***set***, opposta alla bag.
3. L’SQL e l’algebra relazionale usano sintassi diverse.

## Estendere sql

Per estendere l’SQL abbiamo varie possibilità:

* ***Embedded SQL***: consiste nell’integrare l’SQL con un altro linguaggio (C, C#, C++, Java, ecc.).
* ***API***.
* ***4GL (Fourth Generation Language)***: consiste nell’aggiungere all’SQL cicli e controlli; non vi è *conflitto di impedenza*.

È l’approccio migliore in quanto garantisce sicurezza (previene *SQL injections*) ed è più efficiente.

### Conflitto di impedenza

Il ***conflitto di impedenza*** consiste nella diversa tipizzazione tra l’SQL e il linguaggio (nel caso embedded) o l’API.

# Pl/sql

Vedere file [README.md](file:///Users/roccodelprete/Library/CloudStorage/OneDrive-Uniparthenope/basi%20di%20dati/README.md).

## Vista

Una ***vista*** è una tabella virtuale, su cui è possibile eseguire le operazioni di creazione, aggiornamento ed eliminazione. Essa non occupa spazio su disco.

Esempio di creazione di una vista:

create view StudentiInformatica as (select \* from Studenti where corsoDiLaurea = 'INF');

Vantaggi di una vista sono la *gestione dei permessi*, la *semplificazione della scrittura delle query innestate*.

Svantaggi invece sono l’*impossibilità di aggiornamento* di una vista, se la tabella da cui deriva quest’ultima viene cambiata, e la *quantità di calcolo maggiore* nel caso in cui più utenti vanno a generare più viste.

### Vista materializzata

create materialized view StudentiInformatica as (select \* from Studenti where corsoDiLaurea = 'INF');

refresh on commit;

Nella ***vista materializzata*** è possibile *aggiornare* la vista (ricalcolarla) in base a delle opzioni.

È conveniente usare la vista materializzata quando si hanno molte richieste.

## Trigger

Un ***trigger*** è un paradigma ***ECA*** (*Evento Condizionato Azione*): “quando si verifica un evento ed è vera la condizione, allora esegui l’azione”.

È conveniente creare *un trigger per ogni evento*.

### Eventi

Gli ***eventi*** si verificano in DML, ovvero con i comandi *INSERT*,*UPDATE*,*DELETE* e possono verificarsi prima (***before***) o dopo (***after***)

Per poter conoscere il valore di un campo che stiamo per inserire, quindi con la *insert*, si utilizza la clausola ***:NEW***; con la *delete* si usa la clausola ***:OLD***, ovvero si indica una tupla che stiamo per cancellare. Con l’*update*, invece, si usano entrambe le clausole in modo da conoscere il valore precedente (*:OLD*) e aggiornarlo con il nuovo valore (*:NEW*).

# Transazioni

Le ***transazioni*** sono un *blocco di istruzioni* che sono trattate come un'unica operazione atomica. La transazione va a buon fine se tutte le operazioni vanno a buon fine; se il sistema incorre in un errore durante l'esecuzione del blocco di istruzioni, sono annullate tutte le operazioni del blocco.

Garantiscono la coerenza dei dati e permettono a più persone di leggere e scrivere dati in maniera concorrente senza creare incoerenze.

Una transazione può avere due ***stati di terminazione***:

* ***Committed***: tutte le operazioni sono confermate.
* ***Rollback***: tutte le operazioni devono essere annullate.

## Proprietà delle transazioni

Di seguito vengono elencate le ***proprietà*** delle transazioni:

* ***Atomicità***: le operazioni sono eseguite come se fossero un tutt'uno, o si eseguono tutte o nessuna.
* ***Consistenza***: una transazione lascia il database in uno stato *consistente*, che rispetta i vincoli.
* ***Isolamento***: una transazione viene eseguite indipendentemente dalle altre: se più transazioni eseguono in concorrenza, l’effetto è equivalente a quello di un'esecuzione sequenziale.
* ***Durability – Persistenza***: gli effetti di una transazione che ha terminato correttamente la sua esecuzione devono essere persistenti nel tempo.

## Problemi delle transazioni

In caso di multiutenza, alcune operazioni potrebbero causare dei conflitti. Ciò potrebbe succedere quando gli utenti stessero effettuando sia operazioni di lettura che di scrittura.

I problemi che potrebbero sorgere sarebbero:

* ***Dirty Read***: si verifica quando viene letto un dato nel mentre viene effettuata una transazione su di questo, che ad esempio ne cambia il valore. Se tale transazione fallisce viene effettuato il rollback e il dato ritorna al suo valore originale; tuttavia, la lettura era stata effettuata durante la transazione, e ci ritroviamo quindi con un valore *non vero* (**lettura sporca**).
* ***Non repeatable reads***: una transazione potrebbe effettuare più operazioni di lettura sullo stesso dato, che allo stesso tempo potrebbe subire operazioni di scrittura che ne cambiano il valore. Ciò significa che ogni volta che tale dato verrà letto, anche se si tratta della stessa variabile, avremo un valore diverso.
* ***Phantoms***: con “fantasmi” facciamo riferimento alle tuple fantasma, in particolar modo ne distinguiamo di due tipi:
  + *tuple che vediamo ma non esistono* (es. facciamo una select durante l’inserimento di tuple. Se tale inserimento viene alla fine abortito, vedremo delle tuple che in realtà non esistono).
  + *tuple che non vediamo ma esistono* (es. facciamo una select durante una cancellazione di tuple. Se tale cancellazione venisse alla fine abortita, non vedremmo delle tuple che in realtà esistono).
* ***Incorrect summary***: si verifica quando effettuiamo una SUM su una tabella in cui sta avvenendo una cancellazione. Di conseguenza la somma non sarà̀ corretta.

## Livelli di isolamento

Di seguito vengono riportati i 4 ***livelli di isolamento***:

1. ***Read Uncommitted***: una transazione può leggere i valori di un’altra transazione in corso anche se non è committed.  
   *Non c’è isolamento*, e per tal motivo presenta tutti i problemi visti prima.
2. ***Read Committed***: una transazione può leggere i valori di un’altra transazione solo se tali valori sono confermati.  
   Si risolve solo il problema della ***dirty read***.
3. ***Repeatable Read***: se una transazione legge due volte uno stesso dato, ottiene lo stesso valore.

Si risolve il problema della ***dirty read*** e delle ***non repeatable read***.

1. ***Serializable***: tutte le transazioni vengono messe in fila. Non abbiamo alcun problema legato alla concorrenza in quanto, di fatto, la perdiamo (è come se diventasse un sistema monoutente). Per tal motivo, si tratta di una situazione molto inefficiente che deve essere usata solo in casi estremi.

## Protocolli di gestione della concorrenza

Di seguito vengono elencati i ***protocolli per gestire la concorrenza*** tra le varie operazioni:

* ***Lock***: quando una transazione ha bisogno di un dato. Il problema di tale soluzione è che si possano formare dei ***deadlock***, ovvero situazioni in cui due transazioni aspettano il rilascio del lock dell’altra transazione.
* ***Multiversioning*** (usato da Oracle): di ogni dato vengono create molte versioni.
* ***Timestamp***: marca temporale, viene controllato l'ordine delle operazioni    
  (es: vendo il biglietto al primo che ha prenotato).
* ***Ottimistici***: viene permessa l’esecuzione delle operazioni a tutte le transazioni e poi alla fine si fanno i controlli.

# Teoria della normalizzazione

Nella ***teoria della normalizzazione*** ci sono dei criteri che misurano la qualità di uno schema EE/R.

Si può usare per decidere come scomporre una relazione.

## Dipendenza funzionale

Primo concetto della teoria della normalizzazione è la ***dipendenza funzionale***, che dice: se per ogni coppia (tuple), allora:

Ovvero: se la proiezione di su (*insieme di attributi*) è uguale alla proiezione di su , allora avranno uguale anche un altro insieme di attributi (); ciò significa che ***determina*** .

La dipendenza funzionale è una proprietà ***intensionale*** e ***semantica***: per avere la dipendenza funzionale bisogna conoscere il *significato* degli attributi.

Inoltre, per dipendenza funzionale possiamo scrivere:

Ovvero, *la chiave determina tutto l’insieme di attributi*.

Dato come insieme di dipendenze funzionali, allora:

Da qui si determinano regole ***corrette*** e ***complete***.

Una ***regola corretta*** è definita come:

Una ***regola completa*** è definita come:

### Assiomi di Armstrong

#### Riflessività

Se

#### Arricchimento

Se allora : è possibile aggiungere attributi alla dipendenza funzionale.

#### Transitività

Se e allora .

### Chiusura di un insieme di dipendenze funzionali

La ***chiusura*** di un insieme di dipendenze funzionali corrisponde all’insieme di tutte le dipendenze funzionali derivabili.

### Superchiave

Se (tutti gli attributi), allora è ***superchiave***.

## Forme normali

Le ***forme normali*** che vedremo sono quattro e sono:

* ***Prima forma normale***: gli attributi devono essere *atomici*.
* ***Seconda forma normale***: implica la prima forma normale. Riguarda le chiavi multi-attributo e recita: quando si ha una chiave multi-attributo, devono esistere ***dipendenze totali*** e non parziali.
* ***Terza forma normale***: implica la prima e la seconda forma normale, elimina a transitività (vedere *Transitività*) e tutti gli attributi non-chiave *dipendono soltanto dalla chiave primaria*.
* ***Forma normale di Boyce e Codd***: implica le precedenti forme normali e dice che tutti gli attributi non-chiave *hanno dipendenze funzionali solo con la chiave primaria*.

## Operazioni in caso di violazione di integrità referenziale

Quando l’***integrità referenziale*** è violata, il database può assumere tre diversi comportamenti che prevengono il fallimento delle operazioni:

* ***Cascade***: quando viene eliminata una tupla, tutte le tuple collegate con quella tupla verranno eliminate “*a cascata*”: se cancello la tupla di una tabella a cui fanno riferimento altre tuple (tramite FK), anche queste altre tuple verranno eliminate. Quando invece viene aggiornata la PK di una tupla, verrà aggiornato automaticamente anche il valore delle FK delle altre tuple (es. se cancello uno studente, verranno eliminati anche tutti i suoi esami superati).
* ***Set NULL***: quando viene eliminata una tupla (o aggiornata la sua PK), il valore delle FK delle altre tuple associate viene impostato a *NULL*.
* ***Default***: quando viene eliminata una tupla (o aggiornata la sua PK), il valore delle FK delle altre tuple associate viene impostato ad un valore di *default*.