Appunti: Basi di dati

I. Introduzione ai DBMS

Informazione = notizia o elemento che consente di avere conoscenza più o meno esatta di fatti, situazioni e modi d'essere.
Dato = elementi di informazione costituiti da simboli che devono essere elaborati e interpretati.
Sistema informativo = componente di un'organizzazione il cui scopo è quello di gestire le informazioni utili ai fini dell'organizzazione stessa. Non è necessariamente automatizzato.
Sistema informatico = porzione automatizzata del sistema informativo. Al suo interno le informazioni sono rappresentate da dati. Questi dati devono essere gestiti in modo persistente dai sistemi informatici.

Approccio per la gestione dei dati:

- Convenzionale: basato su software non specializzati, ovvero sui file.
- Specializzato: basato su software di gestione dei dati.

I problemi dell'approccio convenzionale sono la gestione di grandi quantità di dati e la condivisione e l'accesso concorrente.

DBMS = sistema software che è in grado di gestire collezioni di dati grandi, condivise e persistenti, in maniera efficiente e sicura.
Base di dati = collezione di dati gestita da un DBMS.

Caratteristiche di un DBMS:

- Efficienza: avere adeguate strutture dati per organizzare i dati all'interno di file, e per supportare le operazioni di ricerca e aggiornamento. In genere, si tratta di strutture ad albero o tabelle hash.
- Concorrenza: gestire operazioni concorrenti di accesso ai dati. Si deve garantire che accessi da parte di applicazioni diverse non interferiscano tra loro, lasciando il sistema in uno stato inconsistente. Il Lock Manager è il componente del DBMS responsabile di gestire i lock delle risorse del database e di rispondere alle richieste delle transazioni.
- Affidabilità: fornire appositi strumenti per annullare operazioni non completate e fare roll-back dello stato del sistema. Si deve garantire la persistenza dei dati anche in caso di malfunzionamenti. Il controllore di affidabilità utilizza dei log, nei quali sono indicate tutte le operazioni svolte dal DBMS.
- Sicurezza: implementare politiche di controllo degli accessi ai dati mediante sistemi di permessi.

Scalabilità (orizzontale) = possibilità di gestire grandi moli di dati aumentando il numero di nodi usati per lo storage.

Tre livelli dell'architettura software del DBMS:

1. Schema fisico: come e dove sono memorizzati i dati su memoria secondaria.
2. Schema logico: cosa rappresenta il database (organizzazione logica dei dati).
3. Schema esterno: come si presenta il database agli utenti.

Modelli logici supportati dai DBMS:

- Modello relazionale
- Modello gerarchico
- Modello reticolare
- Modello ad oggetti
- Approcci NoSQL

Linguaggio DDL = definizione dello schema logico.
Linguaggio DML = manipolazione delle istanze.

Le applicazioni che si interfacciano con un DBMS possono integrare codice SQL all'interno del loro codice oppure utilizzare librerie (fornite dal DBMS) per gestire la connessione al DBMS.

II. Modello relazionale

Modello logico = insieme di concetti per strutturare/organizzare i dati relativi ad un certo dominio d'interesse e insieme di regole per modellare eventuali vincoli e restrizioni sui dati.
Le regole e i concetti generali sono indipendenti dal dominio d'interesse che si sta considerando.

Proprietà auspicabili di un DBMS:

- Indipendenza fisica: interagire con il modello logico in modo indipendente dallo schema fisico.
- Indipendenza logica: interagire con il livello esterno in modo indipendente dallo schema logico dei dati.

Modello relazionale = modello logico in cui i dati sono organizzati in record di dimensione fissa e divisi in tabelle (relazioni).

- Colonna: attributo
- Intestazione: schema della relazione
- Riga: istanza della relazione

L'ordinamento delle righe e delle colonne è irrilevante.

Non possono esistere due attributi uguali o due righe uguali. I dati di una colonna devono essere omogenei.
Ogni attributo dispone di un dominio che definisce l'insieme di valori validi per quell'attributo (string, int…).

Prima Forma Normale
Una relazione si dice in PFN se tutti gli attributi sono definiti su domini atomici e non su domini complessi (no attributi o righe ripetute e no attributi array).

I riferimenti tra dati in relazioni (tabelle) differenti sono espressi mediante valori.

Una relazione sui dati può essere vista come una relazione matematica. Una relazione matematica su un tot di insiemi è definita come un sottoinsieme del prodotto cartesiano di quel tot di insiemi.

Uno schema di relazione è un nome R con un insieme di attributi A1, …, An: R(A1, …, An).
Una ennupla su un insieme di attributi X è una funzione che associa a ciascun attributo A in X un valore del dominio di A.
t[A] denota il valore della ennupla t sull'attributo A.
L'insieme r di ennuple su X è definito come l'istanza di relazione su uno schema R(X).

Quando il valore di un attributo è non noto oppure inesistente, si può sostituire con il valore NULL.

I vincoli di un modello relazionale possono essere:

- **Vincoli intra-relazionali**
  - Vincoli di ennupla: esprimono condizioni su ciascuna ennupla considerata singolarmente.
  - Vincoli di chiave: esprimono condizioni su chi può essere la chiave o superchiave di una relazione.
- **Vincoli inter-relazionali**
  I collegamenti tra relazioni differenti sono espressi mediante valori comuni in attributi replicati. Alcuni DBMS consentono di definire delle relazioni senza una chiave primaria associata.

Chiave = insieme di attributi che consente di identificare in maniere univoca le ennuple di una relazione.
Superchiave = sottoinsieme di attributi di una relazione in cui non sono contenute due ennuple uguali (facenti parte della stessa colonna o insieme di colonne). È la chiave più eventualmente altre colonne.

Una chiave di una relazione r è una superchiave minimale di r.
Esiste sempre almeno una superchiave per ogni relazione e possono esisterne di più per la stessa. Lo stesso vale per le chiavi.

Chiave primaria = chiave di una relazione su cui non sono ammessi valori NULL. Ogni relazione deve disporre di una chiave primaria.

Pro del modello relazionale:
- È intuitivo
- È basato su proprietà algebrico/logiche
- Garantisce indipendenza dallo schema fisico
- Riflessività

Contro del modello relazionale:
- È poco flessibile (tutte le istanze di una relazione devono possedere la stessa struttura)
- Ridondanza dei dati causata dai vincoli

III. SQL: Costrutti DDL

SQL = linguaggio per basi di dati basate sul modello relazionale.

Due componenti principali:
- **Data Definition Language (DDL)**: contiene i costrutti necessari per la creazione e la modifica dello *schema* della base di dati.
- **Data Manipulation Language (DML)**: contiene i costrutti per le interrogazioni e quelli di inserimento, eliminazione e modifica di *dati*.

Comandi

Costruire uno schema di una base di dati	`create database NomeDB`

Costruire una tabella all'interno dello schema	`create table NomeTabella (`
	`nomeAttributo1 Dominio`
	`[valDefault][Vincoli],`
	`nomeAttributo2 Dominio`
	`[valDefault][Vincoli],`
	`…`
	`)`

Costruire un proprio dominio di dati a partire da domini elementari	`create domain NomeDominio as TipoDati` `[Valore di default] [Vincolo]`
Specificare un valore di default	`default [valore | user | null]`
Esprimere vincoli di ennupla	`check (condizione)`
Imporre che uno o più attributi siano una superchiave della tabella	`unique`

Imporre che uno o più attributi siano una chiave primaria	`primary key`
Definire dei vincoli di integrità referenziale tra due tabelle	`foreign key(nomeAtt1, nomeAtt2)` `references nomeTabella(Att1, Att2)`
Associare azioni specifiche da eseguire sulla tabella interna quando i vincoli di integrità vengono violati	`on (delete | update)` `(cascade | set null | set default| no action)`

Modificare o cancellare gli schemi di dati precedentemente creati	`drop (schema|domain|table|view) NomeElemento`
	`drop (restrict|cascade) NomeElemento`
	`alter NomeTabella`
	`alter column NomeAttributo`
	`add column NomeAttributo`
	`drop column NomeAttributo`
	`add contraint DefVincolo`
	…

Domini elementari di SQL:

- character
- numeric
- decimal
- integer
- smallint
- float
- real
- double precision
- date
- time
- timestamp
- blob
- cblob

La keyword `auto_increment` permette di creare dei campi numerici che si auto-incrementano ad ogni nuovo inserimento nella tabella.

IV. SQL: Costrutti DML

Comandi e operatori

Interrogazione	`SELECT (Attributo/i)`
	`FROM (Tabella/e)`
	`WHERE (Condizione/i)`

Espressioni booleane	`AND`

	OR NOT
Confrontare delle stringhe	LIKE _ (carattere arbitrario) % (sequenza di caratteri arbitrari)
Verificare l'appartenenza ad un certo insieme di valori	BETWEEN
Verificare se un valore è nullo o meno	IS NULL IS NOT NULL
Rinominare le colonne del risultato o le tabelle	AS
Rimuovere i duplicati nel risultato	DISTINCT
Ordinare le righe del risultato in base al valore di un attributo specificato	ORDER BY ListaAttributiOrdinamento
Operatori aggregati	Sum() Max() Min() Avg() Count() * conta tutte le righe
Dividere la tabella in gruppi, ognuno caratterizzato da un valore comune dell'attributo specificato nell'operatore. Si applica la select su ciascun gruppo, quindi ogni gruppo produce una sola riga nel risultato finale. ListaAttributi1 deve essere un sottoinsieme di ListaAttributi2.	SELECT ListaAttributi1 FROM ListaTabelle WHERE Condizione GROUPBY ListaAttributi2
Filtrare i gruppi in base a determinate condizioni	SELECT ListaAttributi1 FROM ListaTabelle WHERE Condizione GROUPBY ListaAttributi2 HAVING Condizione
Effettuare operazioni insiemistiche tra tabelle (i tipi devono essere compatibili)	UNION INTERSECT EXCEPT
Inserire una riga esplicitando i valori degli attributi	insert into NomeTabella [ListaAttributi] values (ListaValori)
Inserire in una riga estraendo le righe da altre tabelle del database	insert into NomeTabella SQLSelect
Cancellare tutte le righe che soddisfano una condizione	delete from Tabella where Condizione
Aggiornare il contenuto di uno o più attributi che rispettano una certa condizione	update NomeTabella set attributo=expr|SELECT|null|default [where Condizione]
Implementare il join tra tabelle	SELECT ListaAttributi

	FROM Tabella JOIN Tabella ON CondizioneJoin [WHERE Condizione]

V. SQL: Interrogazioni annidate e viste

Query annidata = nella clausola WHERE, oltre ad espressioni semplici, possono comparire espressioni complesse in cui il valore di un attributo viene confrontato con il risultato di un'altra query.

```
SELECT
FROM
WHERE (Attributo expr SELECT
                    FROM
                    WHERE)
```

Esistono operatori speciali di confronto nel caso di interrogazioni annidate:

- any: la riga soddisfa la condizione se è vero il confronto tra il valore dell'attributo ed almeno uno dei valori ritornata dalla query annidata.
- all: la riga soddisfa la condizione se è vero il confronto tra il valore dell'attributo e tutti i valori ritornati dalla query annidata.

Il costrutto in restituisce true se un certo valore è contenuto nel risultato di una interrogazione nidificata, false altrimenti.
Il costrutto exists restituisce true se l'interrogazione nidificata restituisce un risultato non vuoto.

```
SELECT ListaAttributi
FROM TabellaEsterna
WHERE Valore/i IN|EXISTS SELECT ListaAttributi2
                 FROM TabellaInterna
                 WHERE Condizione
```

Le interrogazioni nidificate possono essere:

- Semplici: non c'è passaggio di binding tra un contesto e l'altro. Le interrogazioni vengono valutate dalla più interna alla più esterna.
- Complesse: c'è passaggio di binding attraverso variabili condivise tra le varie interrogazioni. In questo caso, le interrogazioni più interne vengono valutate su ogni tupla.

Vista = "tabella virtuale" ottenuta da dati contenuti in altre tabelle del database. Ogni vista ha associato un nome ed una lista di attributi, e si ottiene dal risultato di una select.

```
create view NomeView [ListaAttributi]
```

```
as SELECTSQL
[with [local | cascade] check option]
```

I dati delle viste non sono fisicamente memorizzati a parte. Le viste esistono a livello di schema, ma non hanno istanze proprie.

Le viste possono servire a:

- Implementare meccanismi di indipendenza tra il livello logico ed il livello esterno.
- Scrivere interrogazioni complesse, semplificandone la sintassi.
- Garantire la retro-compatibilità con precedenti versione dello schema del database, in caso di ristrutturazione dello stesso.

Common Table Expression (CTE) = viste temporanee che possono essere usate in una query come se fossero una vista a tutti gli effetti, ma non esistono a livello di schema del database.

Asserzioni = costrutto per definire vincoli generici a livello di schema. Consentono di definire vincoli non altrimenti definibili con altri costrutti. Il vincolo può essere immediato o differito (ossia verificato al termine di una transazione).

```
create assertion NomeAsserzione check Condizione
```

VI. SQL: Procedure, trigger e permessi

Stored procedure = frammenti di codice SQL, con la possibilità di specificare un nome, dei parametri di ingresso e dei valori di ritorno.

Le estensioni procedurali permettono di aggiungere strutture di controllo al linguaggio SQL, di dichiarare variabili e tipi di dato user-defined, di definire funzioni avanzate ed ottimizzate (ritenute sicure dal DBMS).

```
CREATE FUNCTION function_name
RETURN type_return
List of SQL routine statements
```

Trigger = meccanismi di gestione della base di dati basati sul paradigma Evento/Condizione/Azione (ECA).

Evento: primitive per la manipolazione dei dati.
Condizione: predicato booleano.
Azione: sequenza di istruzioni SQL.

I trigger garantiscono il soddisfacimento di vincoli di integrità referenziale e specificano meccanismi di reazione ad hoc in caso di violazione dei vincoli.

Inoltre, specificano regole aziendali (business rules).

```
CREATE TRIGGER nome
BEFORE/AFTER INSERT/DELETE/UPDATE ON tabella
[REFERENCING referenza]
[FOR EACH ROW/STATEMENT]
[WHEN (IstruzioneSQL)]
Istruzioni/procedure SQL
```

Permessi = di default, ogni risorsa appartiene all'utente che l'ha definita e su ciascuna sono definiti dei privilegi.

Comandi che cambiano i permessi:

- **`Grant`**: consente di assegnare privilegi su una certa risorsa ad utenti specifici (`with grant option` permette di propagare il privilegio ad altri utenti).
- **Revoke**: consente di revocare privilegi su una certa risorsa ad utenti specifici (`cascade` agisce ricorsivamente sui privilegi eventualmente concessi da quell'utente.

Ruolo = contenitore di privilegi per l'accesso alle risorse di un database (`create role/set role`).

VII. RDBMS: MySQL

[non per esame]

VIII. Gestione delle transazioni

Transazione = unità di lavoro elementare (insieme di istruzioni SQL) che modifica il contenuto di una base di dati.

```
start transaction
update SalariImpiegati
set conto=conto*1.2
where (CodiceImpiegato = 123)
commit|rollback work
```

Proprietà **ACID** delle transazioni:

- Atomicità: la transazione deve essere eseguita con la regola del "tutto o niente".
- Consistenza: la transazione deve lasciare il database in uno stato consistente, eventuali vincoli di integrità non devono essere violati.
- Isolamento: l'esecuzione di una transazione deve essere indipendente dalle altre.
- Persistenza: l'effetto di una transazione che ha fatto `commit work` non deve essere perso.

Gestore dell'affidabilità: garantisce atomicità e persistenza usando log e checkpoint.
Gestore della concorrenza: garantisce l'isolamento in caso di esecuzione concorrente di più transazioni.

Schedule = sequenza di operazioni di lettura/scrittura di tutte le transazioni così come eseguite sulla base di dati. Uno schedule si dice seriale se le azioni di ciascuna transazione appaiono in sequenza, senza essere inframezzate da azioni di altre transazioni.

Problemi dell'esecuzione concorrente:
- Perdita di aggiornamento
- Lettura sporca
- Letture inconsistenti
- Aggiornamento fantasma

Uno schedule si dice serializzabile se produce lo stesso risultato di un qualunque scheduler seriale delle stesse transazioni.

Per implementare il controllo della concorrenza i DBMS usano il meccanismo dei **lock**. È necessario avere acquisito il lock (in lettura o scrittura) per poter effettuare una qualsiasi operazione su una risorsa.
I lock possono essere richiesti o rilasciati (unlock).

Lock manager = componente del DBMS responsabile di gestire i lock delle risorse del database e di rispondere alle richieste delle transazioni.

Il **Two Phase Lock (2PL)** è un metodo di controllo della concorrenza il quale prevede che una transazione, dopo aver rilasciato un lock, non può acquisirne un altro. Ovvero, una transazione deve acquisire prima tutti i lock delle risorse di cui necessita.
Uno schedule che rispetta questo metodo è serializzabile e non può incorrere in configurazioni erronee dovute a aggiornamenti fantasma, letture inconsistenti o perdite di aggiornamento.
Lo **Strict Two Phase Lock (S2PL)** è una variante del 2PL in cui i lock di una transazione sono rilasciati solo dopo aver effettuato le operazioni di commit/abort.
Uno schedule che rispetta questo metodo eredita tutte le proprietà del 2PL e in più non presenta anomalie causate da problemi di lettura sporca.

Questi protocolli possono però generare delle situazioni di **deadlock** che possono essere gestite nei seguenti modi:
- Uso dei time-out: ogni operazione di una transazione ha un time-out entro il quale deve essere completata, pena annullamento della transazione.
- Deadlock avoidance: utilizza time-stamp o classi di priorità tra transazioni.
- Deadlock detection: utilizza algoritmi per identificare eventuali situazioni di deadlock e prevede meccanismi di recovery dal deadlock.

Un metodo alternativo al 2PL è l'utilizzo dei **time-stamp delle transazioni (TS)**. Ad ogni transazione è associato un time-stamp che rappresenta il momento di inizio della transazione. Ogni transazione non può leggere o scrivere un dato scritto da una transazione con time-stamp maggiore. Ogni transazione non può scrivere su un dato già letto da una transazione con un time-stamp maggiore.

Scheduler di sistema = verifica se un'eventuale azione eseguita su una transazione con un certo time-stamp può essere eseguita o meno.

Quattro livelli di **isolamento**:

- Read uncommitted
- Read committed
- Repeteable read
- Serializable

Ripresa a caldo = ripristinare il sistema dopo malfunzionamenti del software.
Ripresa a freddo = ripristinare il sistema dopo malfunzionamenti dell'hardware.

Il controllore di affidabilità utilizza un **log**, nel quale sono indicate tutte le operazioni svolte dal DBMS. Tramite il log, è possibile ripristinare lo stato completo del DBMS in caso di malfunzionamenti o guasti.
Log = file sequenziale suddiviso in record.

Il record può essere di due tipi:

- Di transizione: tengono traccia delle operazioni svolte da ciascuna transizione sul DBMS.
    - Begin/commit/abort(nome)
    - Update(nome, oggetto modificato, stato precedente, stato attuale)
    - Insert(nome, oggetto modificato, stato attuale)
    - Delete(nome, oggetto modificato, stato precedente)
- Di sistema:
    - Dump(eventi di backup)
    - Checkpoint(transizioni attive)

Buffer temporaneo = buffer temporaneo di informazioni in memoria principale che viene periodicamente trascritto su memoria secondaria per aumentare l'efficienza prestazionale.

Quando si effettua un checkpoint:

1. Si sospendono tutte le operazioni in corso sul DBMS.
2. Si rendono effettive anche su memoria secondaria tutte le operazioni effettuate da transazioni che hanno fatto commit, i cui dati si trovino ancora nel buffer (flush).
3. Si scrivono nel record di checkpoint del log tutte le transazioni attive del sistema.
4. Si riprende l'esecuzione delle operazioni.

Regole di scrittura del log:

- Write Ahead Log: la parte Before State di ogni record deve essere scritta prima che la corrispondente operazione venga effettuata, in modo tale da rendere possibile un'operazione di undo.
- Commit Precedence: la parte After State di ogni record deve essere scritta prima che venga effettuato il commit della transazione, in modo tale da rendere possibile un'operazione di redo.

I guasti possono essere dovuti a:

- Malfunzionamenti del software (perdita del contenuto della memoria principale)
- Malfunzionamenti dell'hardware con perdita di dati nella memoria secondaria
- Malfunzionamenti dell'hardware con perdita di dati nella memoria secondaria e stabile

Fasi della ripresa a caldo:

1. Si accede al log e lo si scorre fino all'ultimo record di checkpoint.
2. Si decidono le transazioni da annullare o da rifare.
3. Si disfano tutte le azioni effettuate da ogni transazione (da annullare)
4. Si applicano le azioni effettuate da ogni transazione (da rifare)

Fasi della ripresa a freddo:

1. Si copia il dump nel database attuale (solo la parte deteriorata).
2. Si scorre il log dal dump in poi fino all'ultimo checkpoint e si effettuano le azioni delle transazioni concluse con un commit.
3. Si effettua la ripresa a caldo.

IX. Introduzione ai database NoSQL

NoSQL = movimento che promuove l'adozione di DBMS non basati sul modello relazionale (anche chiamato NoT Only SQL, ma più correttamente NoRel).

Proprietà dei sistemi NO-SQL:

- Database distribuiti
- Strumenti generalmente open-source
- Non dispongono di schema
- Non supportano operazioni di join
- Non implementano le proprietà ACID delle transazioni
- Sono scalabili orizzontalmente
- Sono in grado di gestire grandi moli di dati
- Supportano le repliche dei dati

Motivi della diffusione dei sistemi NoSQL

- Gestione dei Big-data

  Big data = moli di dati, eterogenee, destrutturate, difficili da gestire attraverso tecnologie tradizionali. Denota sia la tipologia dei dati, che le tecnologie e i tool usati per gestirli.

  Quattro V dei big data:
  - Volume: grosse moli di data
  - Velocity: stream di dati
  - Variety: dati eterogenei, multi-sorgente
  - Valore: possibilità di estrarre conoscenza dai big-data

  Data mining: tecniche di apprendimento computerizzato per analizzare ed estrarre conoscenze da collezioni di dati. I dati presentano pattern e relazioni non note a priori e non immediatamente identificabili. È una disciplina complessa che utilizza tecniche di machine learning, intelligenza artificiale e statistiche.

- Limitazioni del modello relazionale

  1. Il modello relazionale presuppone una rappresentazione tabellare, ma i dati possono non essere in quella forma.
  2. Alcune operazioni non possono essere implementate in SQL.
  3. Scalabilità orizzontale dei DBMS relazionali.

  Scalabilità = capacità di un sistema di migliorare le proprie prestazioni per un certo carico di lavoro quando vengono aggiunte nuove risorse al sistema.
  Scalabilità verticale: aggiungere più potenza di calcolo ai nodi che gestiscono il database.
  Scalabilità orizzontale: aggiungere più nodi al cluster.

- Teorema CAP

  Teorema di Brewer (CAP Theorem)
  Un sistema distribuito può soddisfare al massimo solo due di queste tre proprietà:
  - Consistency: tutti i nodi della rete vedono gli stessi dati.
  - Availability: il servizio è sempre disponibile.
  - Partition tolerance: il servizio continua a funzionare correttamente anche in presenza di perdita di messaggi o di partizionamenti della rete.

Modelli logici alternativi al modello relazionale

- Database chiave/valore

I dati del database sono rappresentati come liste di coppie chiave/valore dove la chiave è un valore univoco per operazioni di ricerca e il valore può essere qualsiasi cosa.

- Database document-oriented
  Adatto alla gestione di dati eterogenei e complessi, è scalabile orizzontalmente e i dati sono rappresentati come documenti (coppie chiave/valore JSON). Forniscono funzionalità per aggregazione o analisi dei dati.
- Database column-oriented
  I dati sono organizzati su colonne anziché su righe, ciò rende lo schema più flessibile e più efficiente nello storage garantendo una maggiore possibilità di compressione di dati.
  Column family: contenitore di colonne. Ogni column family è scritta su un file diverso.
- Database graph-oriented
  I dati sono strutturati sotto forma di grafi (i nodi sono attributi o righe, gli archi sono le relazioni tra di essi).

X. Introduzione a MongoDB

MongoDB è un database NoSQL di tipo **document-oriented**. Gestisce dati eterogenei e complessi, è scalabile orizzontalmente e fornisce funzionalità per aggregazione e analisi dei dati. Utilizza **documenti** di coppie **chiave-valore** (JSON).

Tabella -> Collezione
Riga -> Documento
Colonna di una riga -> Campo

JSON = formato per lo scambio di dati tra applicazioni.
I dati hanno la forma di { nome : “valore” }
Il valore può essere un numero, una stringa (racchiusa tra apici), un booleano, un array (tra parentesi quadre), un oggetto (tra parentesi graffe).

Nella stessa collezione si possono inserire documenti con strutture campo-valore differente, senza dover ricorrere a null.

Ogni documento ha un valore _id che viene definito automaticamente e corrisponde alla **chiave primaria**.

Comandi

Avvio del server	`mongod`
Avvio della shell	`mongo`
Utilizzo/creazione di un database	`use` *`nomeDB`*
Creazione di una collezione	`db.createCollection(“`*`nomeCollezione`*`”);`

Inserimento di un documento	db.*nomeCollezione*.insert(*Documento*)
Rimozione di un documento	db.*nomeCollezione*.remove({}) db.*nomeCollezione*.remove(*Selettore*)
Aggiornamento di un documento	db.*nomeCollezione*.update(*Selettore, Campi*) db.*nomeCollezione*.update(*Selettore, {$set: Campi}*) db.*nomeCollezione*.update(*Selettore, {$push: Campi}*) db.*nomeCollezione*.update(*Selettore, Campi, Opzioni*)
Ricerca su una collezione	db.*nomeCollezione*.find() db.*nomeCollezione*.find(*Selettore*) db.*nomeCollezione*.find(*Selettore, Projection*)
Ordinamento sulla find	db.*nomeCollezione*.find(…).sort(*Campo/i*)
Conteggio sulla find	db.*nomeCollezione*.find(…).count()
Filtro duplicati sulla find	db.*nomeCollezione*.distinct([*Campo*], *Selettore*)
Eseguire uno script	mongodb *myFile*.js
Connessione al server	conn = new Mongo();
Database su cui operare	db = conn.getDB(*"nomeDB"*);
Funzione aggregate	db.*nomeCollezione*.aggregate([*Operatore1, Operatore2, …, OperatoreN*])

Le **correlazioni** possono essere espresse mediante campi _id replicati tra più collezioni (non ci sono costrutti di vincoli di integrità referenziale).

Le funzioni **aggregate** consentono di implementare una pipeline di operazioni da eseguire sulla base di dati. Tutte le operazioni prendono in input dei documenti JSON e restituiscono documenti JSON.

Sharding = processo di suddivisione dei dati su un cluster di server MongoDB.
L'allocazione dei diversi segmenti della collezione (chunks) ai diversi nodi del cluster può essere di tipo range-based o hash-based.

Ottimizzazioni a run-time:

- **Splitting** = quando un chunk è troppo grande viene diviso in più parti, non implica la migrazione del chunk.
- **Balancing** = il Balancer tiene traccia del numero di chunk in ogni server, se non sono bilanciati migra i chunk da un server all'altro.

XI. Progettazione di basi di dati

Problemi della progettazione di un database:

- Dimensionamento del problema (avere decine di tabelle)

- Analisi dei requisiti (specifiche del sistema, dati d'interesse, operazioni sui dati da gestire)
- Traduzione del modello relazionale (passare da una specifica informale a uno schema relazionale)

Senza una buona progettazione, possono emergere anomalie ed errori nella traduzione nel modello relazionale. Esistono tecniche e metodologie per una buona progettazione.

Fasi della progettazione di un database:

- Studio di fattibilità
- Raccolta e analisi dei requisiti
- Progettazione
  - Progettazione concettuale -> schema concettuale
  - Progettazione logica -> schema logico
  - Progettazione fisica -> schema fisico
- Implementazione
- Validazione
- Funzionamento

Metodologia "classica" di progettazione di una base di dati:

- Analisi dei requisiti
- Progettazione concettuale: ci si focalizza sul contenuto informativo dei dati ad alto livello di astrazione. Si produce un modello concettuale indipendente dallo schema logico (relazionale) e indipendente dal DBMS in uso.
  Si possono usare modelli concettuali come il modello entità-relazione o UML.
- Progettazione logica (dipende dal modello dei dati): si rappresenta la base di dati nello schema logico del DBMS.
  Comprende la traduzione dello schema concettuale e l'ottimizzazione dello schema logico ottenuto.
  Si rimuovono le ridondanze e si analizzano le prestazioni (trovare quale schema è il più efficiente).
- Progettazione fisica (dipende dal DBMS in uso): si descrivono le strutture per la memorizzazione dei dati su memoria secondaria, e l'accesso ai dati.

XII. Analisi dei requisiti

Raccolta/analisi dei requisiti = completa individuazione dei problemi che il sistema informativo deve risolvere e le caratteristiche che il sistema deve avere. Serve sapere quali dati devono essere gestiti (informazioni e vincoli) e quali operazioni sui dati saranno consentite.

Le fonti dell'analisi dei requisiti sono gli utenti dell'applicazione, la documentazione esistente e le realizzazioni/applicazioni preesistenti.

Passaggi della raccolta dei requisiti:

1. Produrre un documento di specifica: deve essere astratto, avere delle frasi strutturare in modo standard, evitare frasi contorte, individuare omonimi e sinonimi e esplicitare il riferimento tra i termini.
2. Costruire un glossario dei termini.
3. Definire le operazioni sui dati (è utile perché verifica la completezza dei modelli, le prestazioni e fornisce linee guida per l'implementazione).

XIII. Modello Entità Relazione

Modello Entità-Relazione = modello per la rappresentazione concettuale dei dati ad alto livello di astrazione. È basato su rappresentazione grafica (diagramma). È indipendente dal modello logico in uso e dal DBMS di riferimento.

Componenti di un diagramma ER:

- **Entità**
  Classe di oggetti (fatti, persone, cose) della realtà di interesse con proprietà comuni e con esistenza autonoma.
  Un'entità può essere tradotta in una relazione.
  L'istanza di un'entità è uno specifico oggetto appartenente a quell'entità.

- **Relazioni**
  Legame logico fra due o più entità, rilevante nel sistema che si sta modellando.
  Una relazione può essere tradotta in una relazione del modello relazionale.
  L'istanza di una relazione è una combinazione di istanze dell'entità che prendono parte all'associazione.
  Una relazione può coinvolgere un qualsiasi numero di entità, e può anche essere ricorsiva (in questo caso si può definire un ruolo per ciascun ramo della relazione).

- **Attributi**
  Proprietà elementare di un'entità o di una relazione del modello.
  È possibile definire attributi composti come unione di attributi affini di una certa entità o relazione.

- **Cardinalità delle relazioni**
  Coppia di valori (min, max) che specificano il numero minimo e massimo di occorrenze delle relazioni a cui ogni occorrenza di entità può partecipare.
  Il valore minimo può essere 0 (partecipazione opzionale) o 1 (partecipazione obbligatoria). Il valore massimo può essere 1 o N.

In base al valore delle cardinalità massime in una relazione, si distinguono tre casi: relazioni uno-ad-uno, uno-a-molti e molti-a-molti.

- **Cardinalità degli attributi**
  Come per le relazioni, anche per gli attributi è possibile definire una cardinalità minima e massima. La cardinalità di default è 1,1.

- **Identificatori**
  Strumento per identificare in maniera univoca le istanze di una entità.
  Corrisponde al concetto di chiave del modello relazionale.
  L'identificatore può essere interno se è composto da uno o più attributi dell'entità (questi hanno per forza cardinalità 1,1).
  L'identificatore esterno è composto, invece, da altre entità collegato attraverso relazioni (l'entità esterna deve essere in relazione 1,1).

- **Generalizzazioni**
  Definisce una gerarchia tra entità basata sul concetto di ereditarietà.
  Può essere parziale (esistono occorrenze dell'entità padre che non sono occorrenze delle entità figlie) o totale (tutte le occorrenze dell'entità padre sono almeno di una delle entità figlie).

Dizionario dei dati = tabella contenente la descrizione delle entità e relazioni del modello ER.

I vincoli che non possono essere espressi nel modello ER si esprimono attraverso le **Business Rules**.
I vincoli di integrità possono essere espressi mediante **asserzioni**, ossia affermazioni che devono essere sempre verificate sulla base di dati.
Per esprimere le operazioni aritmetiche o logiche che consentono di ottenere un concetto derivato si usano le **regole di derivazione**.

XIV. Progettazione concettuale

La costruzione di uno schema concettuale deve essere corretta (utilizzo corretto dei costrutti del modello E-R) e completa (rappresentare tutti i dati di interesse descritti nel documento di specifica).
Per garantire queste proprietà esistono delle metodologie di progettazione concettuale.

Strategie di progettazione

- Strategia top-down: lo schema concettuale viene ottenuto attraverso una serie di raffinamenti successivi a partire da uno schema iniziale molto astratto.

- Strategia bottom-up: le specifiche iniziali sono suddivise in componenti via via più piccole ed in un secondo momento i vari schemi sono integrati tra loro.
- Strategia inside-out: si individuano una serie di concetti importanti e poi si procede a partire da questi verso concetti correlati.
- Strategia mista: combinazione delle strategie precedenti.

Pattern di progettazione

Regole concettuali del diagramma E-R:

- Entità: concetto che ha proprietà significative e descrive oggetti con esistenza autonoma.
- Relazione: concetto che correla due o più entità.
- Generalizzazioni: concetto che è un caso particolare dell'altro.

Pattern = soluzione di problemi ricorrenti.

1. Concetti di tipo "parte di" -> utilizzo di relazioni uno a molti.
2. Gestione di duplicati -> introduzione di nuove entità in relazioni uno a molti.
3. Tenere traccia della storia di un concetto -> utilizzo di generalizzazioni.
4. Tenere traccia dell'evoluzione nel tempo di un certo concetto -> utilizzo di generalizzazioni.

Analisi di prestazione

Indici di prestazione:

- Costo operazionale: numero di entità/associazioni mediamente visitate per implementare una certa operazione.
- Occupazione di memoria: spazio di memoria necessario per memorizzare i dati.

Tavola dei volumi = fornisce una stima del numero di occorrenze di entità/relazioni presenti nel modello E-R.
Tavola delle operazioni = definisce l'insieme delle operazioni che devono essere implementate sui dati, con la tipologia (batch/interattive) e la frequenza.

Costo di un'operazione: $C(O) = f(O) * w * (\alpha * NC(w) + NC(r))$

Costo dello schema: dato uno schema e un insieme di operazioni con il relativo costo, C(S) = somma di tutti i costi delle operazioni.

Occupazione di memoria dello schema: M(S) = somma della tabella dei volumi delle entità moltiplicata per la dimensione (in Byte) + somma della tabella dei volumi delle relazioni moltiplicata per la dimensione (in Byte).

Si cerca di determinare il miglior trade-off tra occupazione di memoria e costo delle operazioni dello schema.

XV. Progettazione logica

L'obiettivo della progettazione logica è la realizzazione del modello logico a partire dalle informazioni del modello ER.

Per garantire la qualità dello schema prodotto la progettazione logica include due step:

1. **Ristrutturazione del modello concettuale**: modificare lo schema ER per semplificare la traduzione ed ottimizzare il progetto.
    - Eliminazione delle generalizzazioni
        - Soluzione 1: accorpare le entità figlie nel genitore come attributi. È utile quando non ci sono troppe distinzioni tra l'entità padre e i figli.
        - Soluzione 2: accorpare gli attributi e le relazioni dell'entità genitore nei figli. È utile quando ci sono solo operazioni che coinvolgono le entità figlie, ma non il padre.
        - Soluzione 3: sostituire la generalizzazione con relazioni tra padre e figli. È utile quando ci sono operazioni che si riferiscono solo alle singole entità.
    - Eliminazione degli attributi multi-valore
      Possono essere sostituiti introducendo una nuova entità con una relazione uno a molti.
    - Partizionamento/accorpamento di concetti
      È possibile ridurre il numero di accesso separando attributi di un concetto che vengono acceduti separatamente o raggruppandoli se acceduti insieme.
      Gli accorpamenti riguardano in genere relazioni uno-ad-uno.
    - Scelta degli identificatori
      È necessario scegliere l'identificatore minimale, preferendo quelli interni.
    - Analisi delle ridondanze
      Le ridondanze possono essere vantaggiose se rendono le operazioni sui dati più efficienti, mentre sono svantaggiose se occupano troppa memoria e rendono molto più complessi gli aggiornamenti.
      Per scegliere se mantenere o eliminare le ridondanze si calcolano il costo e l'occupazione di memoria e si confrontano nei due casi.

      C(senza) / C(con) = miglioramento dello speedup
      M(con) – M(senza) = overhead di memoria

2. **Traduzione del modello logico**: traduzione dei costrutti del modello ER nei costrutti del modello relazionale.
    - Le entità diventano tabelle sugli stessi attributi. L'identificatore è la chiave primaria.
    - Le relazioni diventano tabelle sugli identificatori delle entità coinvolte (ma può non essere così a seconda della cardinalità).
    - Nelle relazioni uno-a-molti si può inglobare la relazione nell'entità con cardinalità massima 1.
    - Nelle relazioni uno-a-uno si può inglobare la relazione in una delle due entità.

XVI. Normalizzazione

Le **ridondanze** sui dati possono essere di due tipi:

- Ridondanza concettuale: non ci sono duplicazioni dello stesso dato, ma sono memorizzate informazioni che possono essere ricavate da altre già contenute nel database.
- Ridondanza logica: esistono duplicazioni sui dati, che possono generare anomalie nelle operazioni.

Dipendenza funzionale = vincolo di integrità per il modello relazionale che descrive legami di tipo funzionale tra gli attributi di una relazione.
Si ha una dipendenza funzionale tra due attributi se due tuple che sono uguali per il primo attributo sono uguali anche per il secondo attributo.
$Y \rightarrow Z$

Le dipendenze funzionali sono definite a livello di schema e non di istanza.

Sono una generalizzazione del vincolo di chiave (e di superchiave).
Esiste un vincolo di dipendenza funzionale tra una superchiave di una tabella e qualsiasi attributo della tabella.

Le dipendenze funzionali possono essere "buone" (e quindi non causare anomalie) se sulla sinistra della relazione si ha una chiave o una superchiave.

Forma Normale di Boyce e Codd (FNBC)
Uno schema si dice in FNBC se per ogni dipendenza funzionale $Y \rightarrow Z$ definita su di esso, Y è una superchiave dello schema.

Per normalizzare una tabella si possono creare tabelle separate ognuna delle quali rispetta la FNBC.

Uno schema R(X) si **decompone senza perdita** negli schemi R1(X1) ed R2(X2) se, per ogni possibile istanza r di R(X), il join naturale delle proiezioni di r su X1 ed X2 produce la tabella di partenza (quindi non genera tuple spurie).

Per **conservare le dipendenze** in una decomposizione, ogni dipendenza funzionale dello schema dovrebbe coinvolgere attributi che compaiono tutti insieme in uno degli schemi decomposti.

Ogni decomposizione dovrebbe quindi soddisfare tre proprietà:

- Soddisfare la FNBC (ogni tabella prodotta deve rispettare la FNBC)
- Decomporre senza perdite (il join delle tabelle decomposte deve produrre la relazione originaria)
- Conservare le dipendenze (il join delle tabelle decomposte deve rispettare tutte le DF originarie)

Non è tuttavia sempre possibile decomporre una tabelle in FNBC, si introduce una nuova forma normale meno restrittiva: la terza forma normale.

Terza Forma Normale (TFN)
Una tabella è in TFN se per ogni dipendenza funzionale Y → Z dello schema almeno una di queste condizioni è verificata:

- Y contiene una chiave della tabella
- Z appartiene almeno ad una chiave della tabella

Questa forma, al contrario della FNBC, è sempre ottenibile tramite l'**algoritmo di normalizzazione in Terza Forma Normale**.

Come prima, per decomporre la tabella nelle relazioni è necessario che queste siano in TFN, che la decomposizione sia senza perdite e che conservi tutte le dipendenze.

Idee alla base dell'algoritmo di normalizzazione:

1. **Semplificare l'insieme di dipendenze** (nella parte destra deve comparire un singolo attributo).
2. **Raggruppare gli attributi coinvolti nelle stesse dipendenze.**
3. **Assicurarsi che almeno una delle relazioni prodotte contenga la chiave della relazione originaria.**

Chiusura di un insieme di attributi rispetto a un insieme di dipendenze = insieme degli attributi che dipendono funzionalmente dall'insieme di attributi di partenza (ovvero, tutti gli elementi a destra delle frecce che a sinistra hanno il nostro attributo, anche indirettamente).

Un insieme di attributi K è una (super)chiave di una tabella R(U) se F (insieme di dipendenze) implica K → U.

Due insiemi di dipendenze funzionali sono **equivalenti** se il primo implica ogni dipendenza del secondo e viceversa.

Un insieme di dipendenze funzionali si dice **ridondante** se rimuovendo una dipendenza dall'insieme, questo implica comunque la dipendenza che è stata tolta.
Un insieme di dipendenze funzionali si dice **ridotto** se non è ridondante e non è possibile ottenere un insieme equivalente eliminando attributi a sinistra delle frecce.

Passi per trovare una **copertura ridotta** di un insieme di DF:

1. Sostituire l'insieme con uno equivalente che a destra delle frecce ha sempre un singolo attributo.
2. Eliminare gli attributi estranei dalla parte sinistra delle frecce (quelli superflui). Per sapere se un attributo può essere eliminato si calcola la chiusura dell'altro attributo a sinistra e si verifica se include l'attributo a destra, se sì si può cancellare il primo attributo.
3. Eliminare le ridondanze non necessarie. Per sapere se l'intera dipendenza è ridondate la si elimina dall'insieme e si calcola la chiusura dell'attributo a sinistra (della ridondanza presa in considerazione) e si verifica se tale insieme include ancora l'attributo a destra, se sì allora era una dipendenza ridondante.

Passi dell'algoritmo di normalizzazione in TFN:

I. **Ridurre l'insieme di DF**
   Costruire una copertura ridotta dell'insieme.
II. **Decomporre l'insieme di DF**
   Decomporre l'insieme in sottoinsiemi tali che in ogni sottoinsieme ci siano dipendenze che a sinistra di ogni freccia hanno lo stesso attributo (o più attributi insieme).
III. **Fondere gli insiemi**
   Se due o più lati sinistri delle dipendenze si implicano a vicenda, si fondono i relativi insiemi.
IV. **Costruire le relazioni associate**
   Trasformare ciascun insieme in una tabella con gli attributi contenuti in ciascuna dipendenza. La chiave della tabella è il lato sinistro (comune a tutte le dipendenze in quell'insieme).
V. **Verificare l'esistenza della chiave**
   Se nessuna tabella contiene una chiave della tabella originaria allora bisogna inserire una nuova tabella contenente gli attributi della chiave.

Prima Forma Normale (PFN)
Una relazione è in PFN se rispetta i requisiti del modello relazionale, ossia ogni attributo è elementare e non ci sono righe o colonne ripetute.

Seconda Forma Normale (SFN)
Una relazione è in SFN (variante debole della Terza) quando non presenta dipendenze parziali, dove l'attributo a sinistra è un sottoinsieme proprio della chiave e l'attributo a destra è un qualsiasi sottoinsieme della tabella.

XVII. Sistemi informativi sul web

[non per esame]

XVIII. Implementazione di WIS con PHP e MySQL

[non per esame]

XIX. Data Mining

[non per esame]

XX. Tecniche di data mining

[non per esame]

XXI. Weka

[non per esame]