CAP 3

Tipologie di architetture: **- OLTP**: Rivolti alla gestione ottimizzata di transazioni affidabili su DB server, specializzati per gestire centinaia o migliaia di transazione al secondo. **OLAP**: Rivolti alla analisi dei dati, operano su server di data warehouse, specializzati per la gestione di dati per sistemi di supporto alle decisioni.

-Paradigmi per la distribuzione dei dati: - ***Architetture Client-Server***: Separazione del DB server dal client. – ***Database distribuiti***: Più DB server utilizzati dalla stessa applicazione. – ***Database paralleli***: Più device di memorizzazione e processori che operano in parallelo per migliorare le performance. – ***Database replicati***: Stessi dati fisicamente memorizzati su più server. – *Data Warehouse*: Server specializzati per la gestione di dati dedicati al supporto alle decisioni.

-Proprietà dei sistemi distribuiti: - ***Portabilità***: Possibilità di trasportare programmi da un ambiente ad un altro. Stabilita al tempo di compilazione – ***Interoperabilità***: Capacità di interagire tra sistemi eterogenei. Stabilita al tempo di esecuzione.

-Architettura Client-Server: Il client ha un ruolo attivo, richiede. Il server ha un ruolo reattivo, risponde. Generalmente un processo client richiede pochi servizi in sequenza ad uno o più processi server, ed un processo server risponde a più richieste di molti processi client. Il computer dedicato al client deve essere appropriato all'interazione con l'utente, il computer server deve avere una grande quantità di memoria(gestione del buffer) e un'elevata capacità disco(per memorizzare l'intera base di dati). Il client può formulare interrogazioni SQL e mandarle al server, i risultati delle interrogazioni sobno calcolati dal server e mandati al client. Spesso il server è multi-threaded,cioè si comporta come un processo singolo che lavora dinamicamente per conto di transazioni differenti. I server gestiscono una coda di input e una di output ed un processo dispatcher distribuisce richieste ai server e ritorna risposte ai client.

-Architettura Two-Tier e Three-Tier: **- *Two-Tier***: Il client funziona sia da interfaccia utente sia da gestore di applicazioni; Il client supporta la logica delle applicazioni(thick client). – ***Three-Tier***: Un secondo server, chiamato application server, è responsabile per la gestione delle applicazioni comuni a molti client; Il client è responsabile solo per l'interfaccia con l'utente finale(Thin client).

-Basi di dati distribuite: Sistema di base di dati nel quale almeno un client interagisce con più server per l'esecuzione di una applicazione. - ***Vantaggi***: - Rispondono ai requisiti delle applicazioni: la distribuzione dei dati permette la loro gestione dove essi sono generati e utilizzati. – Flessibilità e modularità: Possono essere configurate con aggiunte e modifiche progressive dei componenti. – Affidabilità: Possono rispondere ai guasti con una riduzione delle prestazioni anziché con un blocco completo. **-*Autonomia locale e cooperazione***: La base di dati distribuita può essere considerata dal punto di vista astratto come una unica base di dati e deve essere progettata in modo da cercare di avere applicazioni che vengono eseguite su un singolo server, limitando l'interazione e il trasporto di dati.

-Tipi di DBMS: - ***DDB omogenea***: Tutti i server utilizzano lo stesso DBMS. – ***DDB eterogenea***: I server utilizzano diversi DBMS.

-Frammentazione dei dati: - ***Frammentazione orizzontale***: Ogni R_i ha come tuple un sottoinsieme delle tuple di R, ogni R_i può essere interpretato come risultato di una selezione su R. – ***Frammentazione verticale***: Ogni R_i ha come schema un sottoinsieme degli attributi di R, ogni R_i può essere interpretato come risultato di una proiezione su R. **–*Proprietà di correttezza***: - ***Completezza***: Ogni dato di R deve essere presente in qualche suo frammento R_i. – ***Ricostruibilità***: R deve essere interamente ricostruibile a partire dai suoi frammenti. Generalmente i frammenti orizzontali sono disgiunti(non hanno tuple in comune), mentre i frammenti verticali includono la chiave primaria di R(garantisce ricontruibilità).

-Schemi di allocazione: Descrivono il mapping delle relazioni o dei frammenti ai server che li memorizzano. Ogni frammento corrisponde a un file a livello fisico ed è allocato su uno specifico server. L'allocazione può essere: **- *Non ridondante***: Ogni frammento o relazione è allocato su un solo server. – ***Ridondante***: Almeno un frammento o relazione è allocato su più di un server.

-Livelli di trasparenza: - ***Frammentazione***: Il programmatore non ha bisogno di conoscere la frammentazione e non ha bisogno di conoscere l'allocazione. – ***Allocazione***: Il programmatore deve conoscere la struttura dei frammenti ma non ha bisogno di conoscere l'allocazione. – ***Linguaggio***: Il programmatore deve conoscere la struttura dei frammenti e anche l'allocazione. – ***Assenza di trasparenza***: Ogni DBMS accetta il suo "dialetto" SQL: il sistema è eterogeneo e i DBMS non supportano uno standard comune per l'interoperabilità.

-Ottimizzazione delle interrogazioni: - ***Parallelismo***: Si sottomettono richieste in parallelo invece che in sequenza diminuendo il tempo di risposta globale. – ***Conoscenza sulle proprietà logiche dei frammenti***: Query al frammento dove i dati si trovano, aumenta l'efficienza ma diminuisce la flessibilità.

-Livelli progressivi di complessità: - ***Richieste remote*:** Transazioni read-only composte di un numero arbitrario di query SQL, tutte le query sono indirizzate a un singolo DBMS remoto che può essere solo interrogato. – ***Transazioni remote***: Composte da un numero qualsiasi di comandi SQL, tutti indirizzati a un singolo DBMS remoto. Ogni transazione scrive un singolo DBMS. – ***Transazioni distribuite***: Composte da un numero qualsiasi di comandi SQL indirizzati a un numero arbitrario di DBMS remoti. Ogni comando SQL si riferisce a un singolo DBMS e ogni transazione può aggiornare più di un DBMS(richiede two-phase-commit). – ***Richieste distribuite***: Composte da un numero qualsiasi di comandi SQL indirizzati a un numero arbitrario di DBMS remoti. Ogni comando si può riferire a qualsiasi DBMS, richiede un ottimizzatore di query distribuite.

-Tecnologia di basi di dati distribuite: La distribuzione dei dati non influenza la ***consistenza***(i vincoli di integrità descrivono solo proprietà locali) e la ***persistenza***(ogni sistema garantisce persistenza ai dati localmente memorizzati). La distribuzione dei dati influenza isolamento e atomicità e richiede anche modifiche all'ottimizzazione di interrogazioni.

-Ottimizzazione di query distribuite: - Suddivisione della query in sub-query, ognuna indirizzata ad un DBMS specifico. - Costruzione di una strategia di esecuzione distribuita.

-Controllo di concorrenza: In un sistema distribuito, una transazione t_i può eseguire più sotto-transazioni a diversi nodi(t_{ij} esecuzione di t_i al nodo j).

-Atomicità: L'atomicità di transazioni distribuite può essere compromessa da guasti/malfunzionamenti: guasti a nodi, perdite di messaggi che lasciano l'esecuzione di un protocollo in uno stato non certo, guasti a link di comunicazione che possono causare partizionamenti della rete(una transazione può essere simultaneamente attiva in più di una sottorete).

-Serializzabilità globale: La serializzabilità locale non **garantisce** serializzabilità globale. La serializzabilità globale richiede l'esistenza di uno schedule seriale S equivalente a tutti gli schedule locali S_i risultanti a ogni nodo. ***Proprietà***: **- *Schedule globale conflict-serializzabile***,*la serializzabilità globale è* garantita se ogni scheduler usa 2PL stretto ed esegue il commit in modo atomico quando tutte le sotto-transazioni ai vari nodi hanno tutte le risorse. – ***Schedule globale seriale***, *la serializzabilità globale è* garantita se ogni transazione distribuita acquisisce un singolo timestamp e lo usa in tutte le richieste a tutti gli scheduler che fanno controllo della concorrenza basato sui timestamp(richiede assegnamento di TS globale).

-Metodo di Lamport: Permette di assegnare timestamp che riflettano la precedenza tra eventi in un sistema distribuito. Il timestamp ha 2 gruppi di cifre, le meno significative identificano il nodo, le più significative identificano gli eventi che accadono al nodo. Ogni volta che 2 nodi si scambiano un messaggio i timestamp vengono sincronizzati. L'evento ricevente deve avere un timestamp maggiore o uguale dell'evento mittente. Può richiedere l'incremento del contatore locale al nodo ricevente.

-Deadlock distribuiti: Situazioni circolari di attesa tra 2 o più nodi. Soluzioni: **- *timeout***: utilizzato da DBMS distribuiti. – ***Rilevazione di deadlock e risoluzione***: Può essere fatta con un protocollo asincrono e distribuito. Assumiamo che le sotto-transazioni siano attivate usando una RPC.

-Rilevazione di deadlock distribuiti: Attivata periodicamente ai vari DBMS del sistema, ogni DBMS integra nuove sequenze di attesa con le condizioni di attesa locali, analizza il grafo risultante per rilevare il deadlock e comunica le sequenze di attesa ad altri DBMS.

-Protocolli per il commit distribuito: Permettono a una transazione di raggiungere la decisione corretta di commit o abort su tutti i nodi che partecipano alla transazione. Il più diffuso è il two-phase-commit.

-Protocollo two-phase-commit: La decisione commit/abort presa dalle parti è registrata da una terza parte, che ratifica la decisione. Distingue i server che partecipano alla decisione(gestori di risorse RM) e il processo coordinatore(gestore della transazione TM). Funziona tramite scambio rapido di messaggi tra TM e RM e scritture dei record nei log. TM e RM mantengono log per assicurare resistenza ai guasti. Quando un RM si dichiara ready per una transazione perde la propria autonomia e deve rimanere soggetto alla decisione del TM. Prima della dichiarazione della decisione o se si è dichiarato non-ready, RM può abortire autonomamente facendo UNDO dei suoi effetti, senza partecipare al protocollo two-phase-commit. Si compone di 2 fasi: **- *Prima fase***: TM scrive il record di prepare nel suo log e manda un messaggio prepare a tutti gli RM e imposta il timeout. Ogni RM, se in stato affidabile, scrive nel log il record ready e trasmette al TM il messaggio ready, che indica la scelta di partecipare al protocollo; se è in stato non affidabile, manda un messaggio non ready e termina la propria partecipazione al protocollo. TM raccoglie i messaggi di risposta e scrive il log: global commit se tutti gli RM hanno risposto ready, global abort, se almeno un RM ha risposto non ready o il timeout è scattato e non tutti i messaggi sono stati ricevuti.- ***Seconda fase***: TM trasmette la sua decisione globale a tutti gli RM e imposta il timeout. Ogni RM in stato ready scrive nel log il record relativo alla decisione globale e manda un ACK al TM. La decisione globale viene decisa in loco. TM raccoglie tutti gli ACK dagli RM coinvolti nella seconda fase. Se il timeout scade, stabilisce un altro timeout e ripete il messaggio a tutti gli RM dai quali non ha ricevuto ACK. Quando tutti gli ACK sono arrivati scrive il record complete nel suo log

-Log: - ***Transaction manager***(TM): - *Prepare :* Contiene l'identità di tutti i processi RM(identificatori dei nodi e dei processi). – *Global commit o global abort:* Descrive la decisione globale, e la decisione del TM è finale al momento della scrittura nel log. – *Complete*: Registra la fine del protocollo di commit a due fasi. – ***Resource manager***(RM): - *Ready*: Indica la disponibilità irrevocabile a partecipare al protocollo two-phase-commit, e quindi di contribuire alla fase finale. Può essere scritto solo quando RM si trova in uno stato affidabile, cioè ha il lock su tutte le risorse che devono essere scritte. Riporta l'identificatore del TM. – *Begin,Insert,Delete, Update:* della transazione locale.

-Gestione dei guasti e ottimizzazioni: Un RM, in stato ready perde la sua autonomia in attesa della decisione del TM. Eventuali guasti possono compromettere la corretta esecuzione del protocollo. Vengono utilizzati protocolli di ripristino dai guasti e orrimizzazioni per la gestione dei guasti e della finestra di incertezza.

-Protocolli di ripristino: Ripristinano la correttezza dello stato dei nodi a seguito di guasti durante l'esecuzione del two-phase-commit: caduta di un RM, caduta del TM, perdita di messaggi, partizionamento della rete. – ***Caduta di un RM***: Ripresa a caldo dipende dall'ultimo record di log. Abort, UNDO della transazione; Commit, REDO della transazione; Ready, guasto successo durante two-phase-commit, bisogna chiedere al TM. – ***Caduta del TM***: Ripresa a caldo dipende dall'ultimo record di log. *Prepare*: alcuni RM potrebbero essere in uno stato di blocco, ci sono 2 soluzioni: o si scrive un global abort nel log e si esegue la seconda fase del protocollo, o si ripete la prima fase. *Global commit/abort*: Alcuni RM potrebbero essere stati correttamente informati mentre altri potrebbero essere ancora bloccati, TM deve ripetere la seconda fase. *Complete*: Non ha effetti sulla transazione. – ***Perdita di messaggi***: La perdita di un messaggio prepare o del successivo ready non sono distinguibili dal TM. In entrambi i casi il timeout scade e viene presa una decisione globale di abortire. La perdita di un messaggio di decisione (commit/abort) o di un ACK non sono distinguibili. In entrambi i casi il timeout della seconda fase scade. Nella seconda fase è ripetuto. - ***Partizione della rete***: Non causa ulteriori problemi; la transazione avrà successo solo se il TM e gli RM appartengono alla stessa partizione.

-Ottimizzazioni two-phase-commit: Il protocollo two-phase-commit è abbastanza oneroso a causa delle scritture sincrone(force) richieste su ogni log. Si utilizzano generalmente 2 ottimizzazioni: - ***Abort presunto***: Un TM che riceve una richiesta di recovery remoto da un RM incerto e non ha alcuna informazione sulla transazione, ritorna un global abort come default. Ciò evita la scrittura dei record prepare e global abort. Il record di complete non è critico e può essere omesso. – ***Read-only***: Un partecipante che ha eseguito solo operazioni di lettura nella transazione può dichiararsi read-only e lasciare il protocollo. Il coordinatore ignora i partecipanti read-only nella seconda fase del protocollo.

-Varianti two-phase-commit:-***Commit a 4 fasi***: il TM è replicato da un processo di backup, ad ogni fase il TM prima informa il backup della sua decisione, poi lo comunica agli RM. Il backup può sostituire il TM in caso di guasto. – ***Commit a 3 fasi***: Introduce una terza fase di pre-commit. Se il TM cade, un partecipante può essere eletto come nuovo TM. Inutilizzabile in pratica perché allunga la finestra di incertezza.

CAP 4

-Dati semistrutturati: possono rispettare in modo parziale il proprio schema oppure non avere uno schema proprio

-XML (eXtended Markup Language): E' un formato di file proposto dal W3C per distribuire documenti elettronici, tipo libri, manuali, cataloghi, ecc.. , sul Word Wide Web.

-XML vs HTML: L'Html ha un insieme fisso di tag, a differenza dell'XML, che è uno standard per creare linguaggi di markup con tag personalizzati. XML non è nato per sostituire l'Html in quanto anno scopi diversi: XML è stato progettato per descrivere i dati ed evidenziare cosa rappresentano mentre l'Html è stato progettato per visualizzare i dati ed evidenziare come farli apparire.

-Uso di XML: XML viene usato per varie funzioni: separare i dati da come vengono visualizzati, per scambiare i dati tra sistemi incompatibili, per scambiare informazioni in sistemi B2B, per condividere dati e per memorizzare i dati.

-Sintassi XML: Ogni documento XML deve avere un elemento tag radice, tutti gli altri elementi dovranno essere compresi in questo tag radice; a loro volta tutti gli elementi dovranno avere un tag di apertura e di chiusura. Questi tag sono "case sensitive" e devono essere annidati correttamente. Ogni documento XML inoltre dovrà iniziare con la dichiarazione della versione XML usata e il tipo di codifica dei caratteri utilizzati nel file.

-Elementi (XML): Gli elementi si possono estendere, possono includere altri elementi figli, hanno un contenuto che può essere anche vuoto e possono avere degli attributi.

-Attributi (XML): Gli attributi sono contenuti negli elementi e consentono di associare il valore agli elementi senza essere considerati parte del loro contenuto, i valori devono essere racchiusi tra "" e differiscono dagli elementi perchè non possono contenere elementi figli. Gli attributi hanno delle limitazioni ovvero: possono contenere un solo valore e quindi non possono contenere valori multipli, non sono facilmente estendibili, non possono descrivere strutture e inoltre sono difficilmente manipolabili dalle applicazioni.

-Documento ben formato (well formed): Un documento è detto ben formato se: -inizia con il prologo; - tutti gli elementi hanno tag iniziali e finali; - la nidificazione dei tag è corretta; - gli attributi sono correttamente codificati; - i valori sono correttamente codificati.

-Namespace: Documenti diversi possono avere elementi/attributi con lo stesso nome. E' una collezione di nomi ed è identificato da un **URI** (stesso formato di una URL, ma non è una URL). Un namespace associato ad un elemento si applica anche ai figli dell'elemento; la dichiarazione del namespace nel prologo del documento si applica a tutti gli elementi del documento; elementi fratelli possono avere namespace diversi.

-DTD e XML schema: Permettono di definire un modello per i documenti ovvero dettano il tipo dei documenti cioè i tag ammessi e le loro priorità e le regole di annodamento.

-Elementi (DTD): contenenti altri elementi figli, <!ELEMENT PRODOTTO (DESCRIZIONE)>; con PCDATA <!ELEMENT DESCRIZIONE (#PCDATA)>; contenuto misto, <!ELEMENT ARTICOLO (#PCDATA|PRODOTTO)>; elementi vuoti , <!ELEMENT ARTICOLO EMPTY>

-Cardinalità di elementi DTD: 1 volta, <!ELEMENT LISTA (PRODOTTO)>; 1 o più volte <!ELEMENT LISTA (PRODOTTO+)>; 0 o più volte <!ELEMENT LISTA (PRODOTTO*)>, 0 o 1 volta, <!ELEMENT LISTA (PRODOTTO?)>.

-Vincoli Attributi DTD: #REQUIRED: il valore deve essere specificato; #IMPLIED: il valore può mancare; #FIXED "valore": se presente deve essere "valore"; "valore": default se nessun valore è specificato.

-XML schema vantaggi rispetto ai DTD: A differenza dei DTD gli schema XML hanno vari vantaggi quali: - sono estendibili, - sono file XML a differenza dei DTD che non lo sono, - sono più ricchi e completi, - gestiscono namespace.

-Elementi semplici (XML Schema): Gli elementi semplici non possono contenere altri elementi o attributi, possono essere solo tipi standard o tipi di dati derivati da questi, possono avere valori di default, possono avere valori fissi e possono avere associate delle restrizioni.

-Elementi complessi (XML Schema): Gli elementi complessi possono contenere attributi o altri elementi e possono utilizzare degli indicatori.

-Indicatori (XML Schema): Indicatori di ordinamento: - **Any**, qualunque elemento, in qualunque ordine; - **All**, tutti gli elementi , in qualunque ordine; **-Choice**, uno e un solo elemento; - **Sequence**, tutti gli elementi, nell'ordine specificato. **Indicatori della cardinalità**: - **max0ccurs**, max numero di occorrenze; - **min0ccurs**, min numero di occorrenze. Incatori di raggruppamento: - **Group name; - Group reference**.

-Attributi (XML Schema): Gli attributi possono: -essere solo attributi standard, - possono avere valori di default, - possono avere valori fissi, - possono essere obbligatori o opzionali, - possono avere associate delle restrizioni.

-XQuery: Permette di esprimere query su documenti XML, basato su Xpath ed espressioni FLWOR.

-XPath: Permette di indirizzare parti di un documento. Una path expression seleziona oggetti che corrispondono ad un insieme di cammini sull'albero, ritorna: insieme di nodi, valore booleano, numero, stringa alfanumerica. La valutazione di una path expression avviene in bade al nodo contesto.

-Espressioni FLWOR: Sono simili al costrutto SELECT-FROM-WHERE di SQL ma anziché essere applicate a tabelle associano variabili e valori e utilizzano tali associazioni per produrre risultati. Sono: - FOR, per dichiarare variabili che permettono di iterare sugli elementi di un documento; - LET, per dichiarare variabili associate al risultato dell'espressione, eventualmente associandole a quelle introdotte con for; - WHERE, per esprimere condizioni e filtra le tuple prodotte dal for e dalla let; - ORDER, per ordinare le tuple prodotte dal for e dalle let; - RETURN, per generare il risultato.

-RETURN (Espressioni FLWOR): Genera l'output di un'espressione FLWOR che può essere: un nodo, una foresta ordinata di nodi, un valore. Il return può contenere costruttori di elementi, riferimento a variabili definite da for e let oppure espressioni annidate.

-ORDER BY (Espressioni FLWOR): Ordina il risultato della return.

CAP 5

-Basi di dati attive: Supportano la definizione e gestione di *regole di produzione* (**regole attive**), che sono: -il paradigma *Evento-Condizione-Azione*, secondo cui a seguito dell'evento, se la condizione é soddisfatta, esegui l'azione; -*indipendenza della conoscenza*, secondo cui la conoscenza di tipo reattivo viene sottratta ai programmi applicativi e codificata sottoforma di regole attive. Quasi tutti i DBMS hanno supporto per semplici regole attive, ovvero per i trigger.

-Trigger: DBMS diversi possono differire nella gestione dell'attivazione dei trigger. La creazione dei trigger fa parte del Data Definition Language. Le sue componenti sono: ***evento*** (primitive SQL per la manipolazione dei dati, ad es.insert, delete, update), ***condizione*** (predicato booleano espresso in SQL) e *azione* (sequenza di primitive SQL generiche o procedura). I trigger presentano **2 livelli di granularità**: -*tupla* (row-level), in cui l'attivazione viene avviene per ogni tupla coinvolta nel'evento; -*primitiva* (statement-level), in cui l'attivazione avviene una sola volta per ogni evento e si applica a tutte le tuple coinvolte nell'evento. La **modalità di attivazione** dei trigger può essere: -*immediata*: valutazione immediatamente dopo (opzione after) o prima (opzione before) dell'evento che lo ha attivato; -*differita*: la valutazione avviene alla fine (commit) della transazione. Nella sintassi SQL dei trigger, sono presenti comandi come *statement-level*, che è di default, e *referencing*, che permette di inserire variabili (*old e new* si riferiscono allo stato precedente/successivo della tupla, mentre *old_table* e *new_table* si riferiscono allo stato precedente/successivo della tabella). Vi sono 4 possibilità di trigger in SQL: before row, before statement, after row, after statement. I **before trigger** vengono usati solo per determinare errori e per modificare i valori delle variabili new (possono chiederne la variazione anticipata), non possono contenere comandi DML e non attivano altri trigger. I passi per l'**esecuzione dei trigger** in SQL sono i seguenti: si eseguono prima i trigger statement e poi, in ordine, quelli before now, i test di integrità sulla tabella, i trigger after row, e infine quelli after statement. Se vi sono più trigger della stessa categoria, l'ordine di esecuzione viene scelto dal sistema in un modo che dipende dall'implementazione. I trigger sono gestiti in un **Trigger Execution Context** (**TEC**) e la loro esecuzione può produrre eventi che fanno scattare altri trigger, che sono valutati in un altro TEC interno. Inoltre, in ogni istante possono esserci più TEC per una transazione, uno dentro l'altro, ma uno solo può essere attivo (struttura a stack); per il trigger row-level, invece, il TEC tiene conto di quali tuple sono già state considerate e quali sono ancora da considerare.

-Evoluzione delle regole attive: Rispetto ai trigger relazionali, alcuni sistemi e prototipi evoluti hanno caratteristiche che aumentano il potere espressivo delle regole attive in base a 3 elementi: -***Eventi*:** si dividono in *temporali* o periodici (es.:21/03/2006 alle ore 12:00), *applicativi* (definiti dall'utente), e *combinazioni booleane di eventi*. -***Attivazione***: le regole possono essere attivabili/disattivabili e inoltre alcuni sistemi permettono di definire gruppi di regole, ognuno attivabile/disattivabile. La fase di attivazione é inoltre caratterizzata dalla clausola *instead of* (modalità alternativa a before e after che afferma che quando la condizione é vera, l'azione viene eseguita al posto dell'operazione che ha attivato la regola (evento), e dalla modalità *detached*, in cui la regola viene gestita in una transazione separata (é una modalità che si aggiunge a *immediate*, in cui il trigger viene considerato ed eseguito con l'evento che lo ha attivato, ed a *deferred*, in cui invece il trigger viene gestito al termine della transazione). -***Priorità*:** regola l'ordine di esecuzione delle regole quando ve ne sono diverse attivate contemporaneamente (nei sistemi spesso si usa l'ordine temporale di definizione).

-Proprietà delle regole attive: E' necessario avere garanzie che l'interferenza tra le diverse regole e l'attivazione a catena non generi anomalie, del tipo di: -***terminazione***: per qualunque stato iniziale e qualunque sequenza di modifica, le regole producono uno stato finale, cioé non devono esserci cicli infiniti di attivazione (é questa l'anomalia più importante). -***confluenza***: per qualunque stato inziale e qualunque sequenza di modifiche, le regole producono uno stato finale (terminazione) e inoltre producono un unico stato finale, indipendente dall'ordine in cui i trigger vengono eseguiti (é significativa quando il sistema presenta del non determinismo nella scelta delle regole da eseguire). -***determinismo delle osservazioni***: per qualunque stato iniziale e qualunque sequenza di modifiche, le regole terminano e producono un unico stato finale, indipendente dall'ordine in cui le regole vengono eseguite (confluenza), e producono la stessa sequenza di azioni visibili.

-Analisi di terminazione: E' basata sul ***grafo di attivazione***, dove vi é un nodo per ogni regola, e un arco da un nodo Ri a un nodo Rj se l'esecuzione dell'azione di Ri può attivare la regola Rj. Se il grafo é *aciclico*, si ha la garanzia che il sistema é terminante (da notare che l'aciclità è una condizione sufficiente ma non necessaria per la terminazione).

-Applicazioni delle regole attive: E' possibile fare una distinzione tra ***regole interne alla base di dati***, generate dal sistema e non visibili all'utente (come gestione dei vincoli di integrità referenziale, derivazione o replicazione dei dati, ecc.), e *regole* ***esterne*** **(o *regole aziendali*)** che invece esprimono conoscenza di tipo applicativo.

-Regole aziendali: Vincoli o controlli non esprimibili con i costrutti del DBMS, ad esempio ***integrità*** (es.:un impiegato deve avere uno stipendio minore del direttore del dipartimento al quale afferisce) o ***derivazione*** (es.: il fatturato totale é la somma dei totali delle fatture emesse).

CAP 6

-Basi di dati a oggetti(ODBMS): Questi sistemi sono stati sviluppati indipendentemente senza nessuna standardizzazione. Nonostante ciò dopo un breve periodo inizia ad esserci una convergenza sul modello e il linguaggio.La prima generazione di ODBMS è composta da linguaggi di programmazione a oggetti persistenti, incompatibile con gli RDBMS. Due tecnologie: **OODBMS**(object-oriented) e **ORDBMS** (object-relational).Una base di dati a oggetti è una colezione di oggetti.Ciascun oggetto ha un **identificatore**, uno **stato** e un **comportamento**. L'identificatore(OID) garantisce l'individuazione in modo univoco dell'oggetto e permette di realizzare riferimenti tra oggetti. Lo stato è l'insieme dei valori assunti dalle proprietà dell'oggetto.Il comportamento è descritto dall'insieme dei metodi che possono essere applicati all'oggetto.Un tipo descrive le proprietà di un oggetto(parte statica) e l'interfaccia dei suoi metodi(parte dinamica).

-Oggetti e valori: Un oggetto è una coppia(OID, valore) dove OID è un valore atomico e valore è un valore complesso. Il valore assunto da una proprietà di un oggetto può essere l'OID di un altro oggetto.

-**Identità e uguaglianza:**Tra gli oggetti sono definite le seguenti relazioni:-*Identità:* Richiede che gli oggetti abbiano lo stesso identificatore. –***Uguaglianza superficiale***:Richiede che gli oggetti abbiano lo stesso stato,cioè lo stesso valore per proprietà omologhe.-***Uguaglianza profonda***: Richiede che le proprietà che si ottengono seguendo i riferimenti abbiano gli stessi valori, non richiede l'uguaglianza dello stato.

-**CGI (Common Gatway Interface**): E' il primo e il più semplice standard architetturale proposto per risolvere il problema della creazione dinamica delle pagine. Si basa su un semplice concetto, quello di utilizzare l'URL della richiesta HTTP per invocare un programma presente sul server, che calcolerà la pagina da restituire al client.

-**Tipo**: Un tipo descrive le proprietà di un oggetto (parte statica) e l'interfaccia dei suoi metodi (parte dinamica). Relativamente alla parte statica, i tipi vengono costruiti a partire da un insieme di tipi atomici (numeri, stringhe, booleani, …). **Tipo complesso:** Con i costruttori di tipo, si definiscono tipi complessi. I costruttori di tipo (tra loro ortogonali), sono: record of, set of, bag of, list of. Dato un tipo complesso T, un oggetto che ha per tipo T si dice istanza di T.

-Classi: Gli oggetti sono associati ad un tipo(intensione) e ad una classe(implementazione). Una classe descrive l'implementazione di un tipo; struttura dei dati e implementazione di metodi tramite programmi. La definizione di una classe è separata in due parti: l'interfaccia descrive il tipo degli oggetto appartenenti alla classe e la segnatura di tutti i metodi, mentre l'implementazione descrive il codice dei metodi.

-Classi ed estensioni:I concetti di classi ed estensioni non sono identici: una classe è un' implementazione di un tipo. Un'estensione è una collezione di oggetti aventi lo stesso tipo.

-Relazioni:I riferimenti permettono di rappresentare relazioni tra due tipi A e B. Le relazioni si distinguono in:uno a uno, uno a molti e molti a molti.

-Metodi: Un metodo è una procedura utilizzata per incapsulare lo stato di un oggetto ed è caratterizzato da un'interfaccia e un'implementazione. I metodi possono essere dei seguenti tipi:-***Costruttori***: per costruire oggetti a partire da parametri d'ingresso;-***Distruttori***: per cancellare gli oggetti;-***Accessori***:funzioni che restituiscono informazioni sul contenuto degli oggetti;-***Trasformatori***:procedure che modificano lo stato degli oggetti.

-Mismatch di impedenza: In un RDBMS esiste un mismatch di impedenza tra i linguaggi con i quali vengono scritte le applicazioni e l'SQL, che estrae insiemi di ennuple

-Persistenza: Gli oggetti possono essere temporanei o persistenti. La persistenza può essere specificata in vari modi:inserimento di una classe persistente, raggiungibilità da oggetti persistenti e la definizione di un nome per un oggetto.

-**Gerarchie di generalizzazione (o di ereditarietà):**Tra i tipi (e le classi) di una base di dati a oggetti è possibile definire una gerarchia di ereditarietà.Una sotto-classe eredita lo stato e il comportamento della sopra-classe.Tutti gli oggetti delle sotto-classi appartengono automaticamente alle sopra-classi.

-Ereditarietà multipla: In alcuni sistemi è possibile che una classe erediti da più sopra-classi .L'ereditarietà multipla può generare conflitti di nome, qualora due o più sopra-classi posseggano proprietà o metodi con lo stesso nome.

-Dichiarazioni fra classi: Nel corso della propria esistenza un oggetto deve mantenere la propria identità, tuttavia è possibile che un oggetto cambi tipo, ed è quindi necessario un meccanismo di acquisizione e perdita di tipi.

-Ridefinizione dei metodi:L'***overriding*** consiste nel ridefinire il corpo di un metodo nell'ambito di una sotto-classe.Si hanno quindi diverse versioni dello stesso metodo(***overloading***).La scelta del metodo da invocare dipende dalla classe cui appartiene l'oggetto. La scelta del metodo da invocare dipende dalla classe cui appartiene l'oggetto; se la classe non è nota al momento della compilazione, è necessario effettuare il ***late binding***.

-Ridefinizione dei metodi con raffinamenti di tipo: La modifica dell'interfaccia dei metodi richiede una certa attenzione. Se si ridefinisce un metodo di una sottoclasse, i suoi parametri possono essere definiti in 2 modi: -***co-variante***: i parametri sono sotto-tipi dei parametri della sopraclasse; -***contro-variante***: i parametri sono sopra-tipi dei parametri della sopraclasse. La soluzione co-variante é più diffusa, ma crea dei problemi nei parametri di ingresso.

-Manifesto delle basi di dati a oggetti(OODBMS): E' una lista di funzionalità per la definizione di OODBMS ed include funzionalità obbligatorie, funzionalità opzionali e scelte aperte.Alcune funzionalità obbligatorie: Oggetti complessi, Identità di oggetto, Persistenza, Concorrenza, Recovery..Alcune funzionalità opzionali: Ereditarietà multipla, Distribuzione, Gestione delle versioni..Alcune scelte libere: Paradigma di programmazione, Sistema di tipi, Uniformità.

-ODMG (Object Database Management Group): è composto da alcuni esperti che hanno proposto uno standard per OODBMS, il ODMG-93 che comprende un modello a oggetti, ODL(un linguaggio di definizione basato su IDL) e OQL(un linguaggio di interrogazione).

-Manifesto dei 3G DBMS o Contro-manifesto: E' una risposta al manifesto OODBMS. I principi del contro-manifesto sono: i DBMS di terza generazione dovranno essere una generalizzazione compatibile con DBMS della seconda generazione;oltre a fornire servizi tradizionali di gestione dei dati, dovranno permettere la definizione di oggetti complessi e regole; dovranno essere aperti ad altri sottoinsiemi.

-Modello dei dati di SQL-3: E' possibile definire dei tipi: **- *tipi ennupla*** in cui le ennuple sono gli oggetti, le relazioni sono le classi, gli identificatori possono essere manipolabili e si possono usare riferimenti e/o incorporare oggetti. – ***tipi astratti***: permettono di definire tipi da utilizzare per i singoli attributi; possono avere funzioni associate.

-Tecnologia delle basi di dati a oggetti: La gestione di basi di dati a oggetti solleva alcune problematiche: -rappresentazione dei dati e degli identificatori, indici complessi, architettura client-server, modello transazionale, architettura a oggetti distribuiti.

-Rappresentazione dei dati: -***approccio orizzontale***: ogni oggetto viene rappresentato in modo contiguo entro la classe più specifica di appartenenza. Questo approccio favorisce l'accesso agli oggetti nel loro complesso. –***approccio verticale***: gli oggetti sono suddivisi nelle proprie componenti(proprietà), le quali sono memorizzate contiguamente. Questo approccio favorisce la ricerca di oggetti sulla base di una loro proprietà.

-Rappresentazione degli identificatori: Esistono diversi approcci per rappresentare gli OID: mediante indirizzo fisico(memoria di massa), mediante surrogato, cioè un valore simbolico associato unicamente ad un oggetto(strutture di accesso).

-Architettura Client-Server nei sistemi a oggetti: Un meccanismo possibile è quello di delegare al server le operazioni di check-out e check-in degli oggetti,importando gli oggetti sul client e manipolandoli manualmente.

-Modello transazionale nei sistemi a oggetti: Modelli transazionali più complessi di quelli basati sul locking a causa di transazioni di lunga durata e di transazioni complesse. Idee utilizzate: Check-out e check-in di oggetti, versioni di oggetti, versioni di collezioni di oggetti e versioni dello schema degli oggetti.

-CORBA: E' una proposta di standard dell'OMG con lo scopo di garantire l'interoperabilità di oggetti distribuiti mediante un ORB(object request broker). L'ORB è un bus software per oggetti distribuitim responsabile della comunicazione tra oggetti.Ciascun oggetto ha un'interfaccia(IDL), un'implementazione e risiede su un nodo del sistema. Un client può invocare un metodo di un oggetto in modo statico con un meccanismo simile all'RPC. Questo meccanismo rende trasparente la distribuzione dell'oggetto, l'effettiva locazione, la selezione dell'oggetto e la conversione di formato dei parametri. L'invocazione dei metodi degli oggetti può essere anche di tipo dinamico.

-Interrogazione su dati multimediali: Le query devono spesso integrare due diversi aspetti. Condizioni sulla parte strutturata delle informazioni e condizioni di "somiglianza". La ricerca all'interno di testi è il caso più tipico di query su dati multimediali. Le tecniche tipiche sono: esclusione di parole irrilevanti, riduzione a parole chiave, considerazione delle frequenze delle parole.

-Rappresentazioni dei dati spaziali(GIS): I GIS sono sistemi specifici per la gestione di informazioni geografiche e spaziali. Le strutture **2d-tree** e **quad-tree** costruiscono una rappresentazione ad albero di punti nello spazio che permette la gestione efficiente di query spaziali.

Cap 7

-Limiti delle basi di dati relazionali: La basi di dati sono adatte per la gestione efficiente di dati in linea (**OLTP**) ma offrono supporto limitato all'analisi dei dati (**OLAP**): l'SQL non é adatto agli analisti di alto livello, anche perché le applicazioni sono complesse e rigide (é infatti difficile ottimizzare applicazioni in modo che soddisfino sia le esigenze della gestione in linea sia quelle di analisi). Si necessità quindi di soluzioni che rendano i dati prodotti per la gestione operativa utilizzabili anche per la gestione strategica.

-OLTP (**On Line Transaction Processing**): Rappresenta la tradizionale elaborazione di transazioni, che realizzano i processi operativi, in cui ogni operazione (predefinita e relativamente semplice), coinvolge "pochi" dati (dati di dettaglio, aggiornati), e vi sono le proprietà ACIDe delle transazioni essenziali.

-OLAP (**On Line Analytical Processing**): Rappresenta l'elaborazione di operazioni per il supporto alle decisioni, in cui ogni operazione (complessa e casuale) può coinvolgere molti dati (dati aggregati, storici, anche non aggiornati), e vi sono proprietà ACIDe non rilevanti, tipicamente per operazioni di lettura.

-Differenze tra OLTP e OLAP: La configurazione di sistemi dedicati a uno solo dei 2 compiti è un problema gestibile, mentre é estremamente difficile far convivere i 2 carichi di lavoro insieme, a causa della disomogeneità di utenti e requisiti, e delle differenze tecniche che vi sono. -***Conflitto di lock*:** OLTP, tante transazioni rapide con lock esclusivi; OLAP: poche transazioni lunghe con lock condivisi; OLTP+OLAP, le transazioni OLTP sono molto rallentate o le query OLAP non riescono ad essere eseguite. -***Uso degli indici***: OLTP, pochi e solo se servono; OLAP, tanti per coprire ogni esigenza; OLTP+OLAP, o le transazioni OLTP rallentano per l'aggiornamento di molti indici, o le query OLAP non hanno a disposizione gli indici necessari. -***Precomputazione di query***: OLTP, molto raramente, per problemi di consistenza e di carico; OLAP, aspetto chiave per abbassare i tempi di risposta. -***Modello logico***: OLTP, elevata frammentazione e tante tabelle, generalmente normalizzate; OLAP, poche tabelle denormalizzate. Concludendo, OLTP e OLAP hanno un conflitto intrinseco, che non sparisce con l'aumentare della potenza di calcolo. La soluzione é quindi quella di separare i 2 ambienti: basi di dati (OLTP) e data warehouse (OLAP).

-Data warehouse: Base di dati per il supporto alle decisioni (OLAP) con le seguenti caratteristiche: -***Integrata***: riunisce i dati di diverse sorgenti informative preesistenti e rappresenta i dati in modo univoco, riconciliando le eterogeneità delle diverse rappresentazioni (nomi, struttura, codifica). -***Dati in forma aggregata***: i dati delle sorgenti sono aggregati sulla base di opportune coordinate (es.: tempo, collocazione geografica) ed é orientata ai dati (non alle appllicazioni), cioé le basi di dati operazionali sono costruite a supporto dei singoli processi operativi o applicazioni, e la data warehouse é costruita attorno alle principali entità del patrimonio informativo aziendale. -***Dati storici/temporali***: tiene l'evoluzione storica delle informazioni con un ampio orizzonte temporale e indicazione di elementi di tempo. -***Fuori linea e non volatile***: tipicamente formata in modo asincrono e periodico rispetto alle sorgenti. -***Autonoma***: fisicamente separata dalle sorgenti informative, per vari motivi fra cui: ragioni tecniche; non esiste un'unica base di dati operazionale che contiene tutti i dati di interesse; la data warehouse deve essere integrata; i dati di interesse sarebbero comunque diversi (in quanto devono essere mantenuti dati storici ed aggregati); l'analisi dei dati richiede organizzazioni speciali dei dati e metodi di accesso specifici; degrado generale delle prestazioni senza la separazione.

-OLAP e controllo di gestione: Le applicazioni per il controllo di gestione hanno alcune caratteristiche in comune con le applicazioni OLAP, fra cui la lettura di dati, la necessità di svolgere aggregazioni e i confronti con serie storiche. Presentano comunque importanti differenze, come la staticità delle query (non query ad hoc, ma report fissi) e la necessità di mantenere il dettaglio dello schema. Le applicazioni per il controllo di gestione, possono comunque condividere informazioni con l'OLAP, ma si deve trattare di progetti diversi.

-Architettura della Data Warehouse: La data warehouse é composta dai seguenti elementi: -sorgente dei dati; -data warehouse server (dedicato all'OLAP): spesso gestisce anche viste materializzate (data mart) e può avere associati metadati e strumenti per l'assistenza allo sviluppo; -sistema di alimentazione, per l'estrazione dei dati dalle sorgenti, la pulizia (elimina errori e inconsistenze) e la trasformazione, e per il caricamento nella data warehouse; -strumenti di analisi, divisa in: analisi multidimensionale (operazioni interattive di aggregazione/disaggregazione dei dati) e data mining (ricerche sofisticate per inferire nuovi dati e correlazioni fra i dati).

-Modello multidimensionale: La rappresentazione dei dati ad alto livello prescinde dai criteri di memorizzazione e favorisce l'analisi. I concetti rilevanti di questo modello, sono: il ***fatto***, che é il concetto sul quale centrare l'analisi; la ***misura***, che é una proprietà atomica del concetto da anlizzare; e la ***dimensione***, che descrive una prospettiva lungo la quale effettuare l'analisi. Esempio: in una catena di negozi, la vendita é il fatto, l'incasso é la misura, e il tempo e il luogo sono le dimensioni.

-Dimensioni e gerarchie di livelli: Ciascuna dimensione é organizzata in una gerarchia che rappresenta i possibili livelli di aggregazione per i dati della dimensione stessa. **Operazioni su dati multidimensionali**: -***Slice-and-dice*** (seleziona e proietta): seleziona un sottoinsieme delle celle di un cubo. -***Roll-up*** (aggrega i dati): applica una funzione aggregativa (tipicamente somma) sui dati aggregati di un cubo; é possibile effettuarla su misure additive (su cui ha senso fare la somma lungo una dimensione). -***Drill-down*** (disaggrega i dati): é l'operazione inversa del roll-up e aggiunge dettaglio ad un cubo disaggregandolo lungo una o più dimensioni.

-Realizzazione di Data warehouse: -***ROLAP (Relational OLAP)***: utilizza tecnologia relazionale, opportunamente adattata e estesa, in cui i dati sono memorizzati in tabelle e le operazioni di analisi sono espresse come istruzioni SQL; vi sono inoltre strutture di accesso ai dati speciali per ottimizzare le operazioni. E' generalmente costruito in modo incrementale, per collezione di data mart settoriali, ed utilizza uno schema *dimensione a stella*, basato su modello concettuale a stella, con restrizioni sulla struttura dello schema. -***MOLAP (Multidimensional OLAP)***: memorizza i dati direttamente in forma multidimensionale e le strutture dati sono tipicamente proprietarie.

-**Schema a stella**: E' costituito da 3 componenti principali: -***Tabella dei fatti***: una tabella principale che memorizza i dati del data mart; la sua chiave è composta da attributi che sono riferimenti alle chiavi delle tabelle dimensione. –***Tabelle dimensione***: sono relazioni ausiliare che memorizzano i dati relativi alle dimensioni dell'analisi; hanno una chiave semplice (composta da un solo attributo) e gli attributi non chiave rappresentano i livelli della dimensione o qualche loro proprietà, e sono tipicamente testuali/descrittivi. –***Vincoli di integrità referenziale***, ognuno dei quali collega un attributo della tabella dei fatti a una tabella dimensione.

-**Schema a fiocco di neve (snow flake)**: Risulta da una normalizzazione (anche parziale) di uno schema a stella, e permette di evitare ridondanze eccessive nelle dimensioni. E' una normalizzazione da fare con attenzione, in quanto generalmente porta degrado nelle prestazioni.

-Operazioni su ROLAP: I dati sono presentati all'utente con un modello di alto livello (multidimensionale) e le interrogazioni sullo schema multidimensionale sono trasformate dal sistema in istruzioni SQL.

-Aggregazione in SQL: Lo standard SQL offre operatori per aggregazione, che esprimono tutte le possibili aggregazioni delle tuple di una tabella, e utilizzano il nuovo valore polimorfo ALL (presenti in tutti i domini e corrispondente all'insieme di tutti i possibili valore del dominio). Le opzioni di aggregazione in SQL sono: ***with data cube***, che aggrega su tutte le possibili dimensioni, e ***with roll up***, che aggrega in modo progressivo su una dimensione per volta, secondo un ordine specificato.

-Ottimizzazioni: -***Indici bitmap***: permettono di valutare in modo efficiente disgiunzioni, congiunzioni e predicati di uguaglianza; -***indici join***: precomputano il join fra le tabelle dimensione e la tabella dei fatti; -***materializzazione delle viste***: vengono precalcolate le viste che servono per rispondere alle query più frequenti.

-Data mining: E' una tecnica di analisi per estrarre informazione non esplicita nei dati. E' un processo svolto in modo iterativo e adattativi con costruzione progressiva della conoscenza, mediante: comprensione del dominio applicativo, preparazione dell'insieme di dati, scoperta e valutazione dei pattern, e utilizzo dei risultati. Le applicazioni del data mining, contengono: ***regole di associazione***, che ricercano regolarità nei dati e sono costituite da una premessa della regola (corpo) e da una conseguenza della regola (testa); ***regole di classificazione***, che classificano nuovi fenomeni in classi predefinite e presentano fondamentalmente 2 proprietà: ***supporto***, che rappresenta la probabilità che siano presenti in una transazione entrambi gli elementi di una regola, e ***confidenza***, che costituisce invece la probabilità che sia presente in una transazione la conseguenza di una regola, essendo presente la premessa. Il **data minino per associazione** consiste nell'estrarre tutte le regole con supporto e conseguenza superiori a certi valori prefissati.

-Classificazione: Corrisponde ad una catalogazione di un fenomeno particolare in una classe predefinita. E' quindi un processo costituito da: un ***fenomeno da classificare***, presente sottoforma di fatto elementare (tupla); e un ***classificatore***, algoritmo che svolge la classificazione, costruito automaticamente tramite un insieme di dati di prova (training set), applicato per la classificazione di fenomeni generici, e rappresentato come albero di decisione.