Progettazione

Considerazioni sui costi e sugli aspetti di progettazione, realizzazione e produzione per sistemi: manifatturieri, informatici, di definizione dei dati.

- *Prodotti manufatturieri: peso preponderante sulla fase di produzione di serie (anche se aumentano sempre più i costi di progettazione)*
- *Prodotti informatici: peso quasi paritetico tra progettazione e produzione del primo singolo manufatto (costo di produzione di serie asintoticamente nullo)*

Modello dati: peso quasi completamente sulla parte progettuale (il processo di produzione è quasi sempre automatizzabile con strumenti CASE e comunque il prodotto è il risultato della progettazione)

Strategie di progetto in generale:

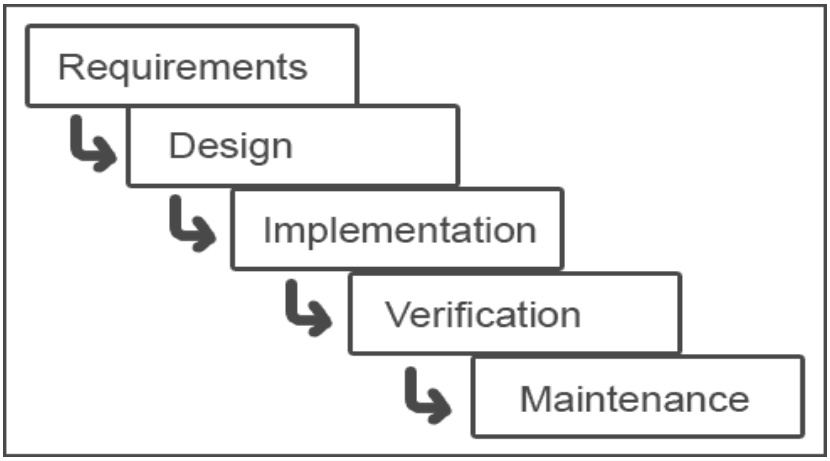
- Top-down
- Bottom-up
- Inside-out
- Prototipale (non su libro)
- Ciclica (non su libro)
- eXtreame Programming (XP - non su libro)

Per la progettazione dei dati non si possono adottare soluzioni prototipali, ne tanto meno XP.

Per i dati la strategia è tipicamente top-down ciclica, cioè fasi di progettazione top-down da un riesame della correttezza e coerenza delle definizioni top sulla base dei risultati down.

Top-down

Modello più anziano, detto anche “a cascata”.



Presuppone la possibilità di completare una fase senza più necessità di riesame sulla base delle attività successive, quindi:

- perfetta conoscenza del dominio
- stabilità dei requisiti
- nessun effetto retroattivo tra dettaglio e generale (il dettaglio non nasconde nulla che non sia

già definito a livello generale)

L'approccio top-down enfatizza la pianificazione ed una completa comprensione del sistema. È ovvio che nessuna codifica può iniziare finché non si è raggiunto almeno un sufficiente livello di dettaglio nella progettazione di una parte significante del sistema. Questo, comunque, ritarda la fase di test finché una parte significativa della progettazione non è stata completata.

Bottom-up

Dopo una fase di "break-down" in cui il sistema viene spezzettato in elementi sempre più piccoli non ulteriormente scomponibili, si analizzano gli atomi per poi procedere ad aggregazioni successive fino ad ottenere il modello completo del sistema.

Aggregando i sistemi parziali si possono individuare:

- T1 - oggetti che hanno tutte le caratteristiche in comune (unificazione di entità)
- T2 - oggetti di sotto-sistemi diversi che hanno un legame (relazione)
- T3 - oggetti di sotto-sistemi diversi che sono casi particolari di un concetto più generale (generalizzazione)
- T4 e T5 - serie di attributi ricorrenti nei vari sotto-sistemi che possono essere aggregati in un'unica entità o relazione

E' un approccio da chimica-molecolare, che presuppone:

- la possibilità d'individuare subito tutte le componenti del sistema
- che il totale non abbia proprietà che trascendono la somma delle parti (problema della vita: come la si spiega partendo dalla sola chimica molecolare ?)

E' una ottima strategia per affrontare sistemi di grossissime dimensioni in cui un singolo analista non può essere in grado di dominare ogni dettaglio, ma l'integrazione dei modelli è sempre molto complessa ed è soggetta alla capacità di mediazione e di contrattazione tra i vari analisti.

Strategia Bottom-up per i DB
1. Identificare i Tipi d'Entità e, se non esistono, inserire in un Glossario le relative Definizioni
2. Identificare i Tipi di Relazione e, se non esistono, inserire nel Glossario le relative Definizioni
3. Per ogni Tipo d'Entità e ogni Tipo di Relazione, identificare gli Attributi e, se non esistono, inserire nel Glossario le relative Definizioni
4. Introdurre i vincoli di integrità ed eventuali altri vincoli

L'approccio bottom-up enfatizza la codifica e la fase di test precoce, che può iniziare appena il primo modulo è stato specificato. Questo approccio, comunque, induce il rischio che i moduli possano essere codificati senza avere una chiara idea di come dovranno essere connessi ad altre parti del sistema, e quel tipo di link potrebbe non essere facile.

Inside-out

Si procede dall'analisi completa di una porzione del sistema complessivo, per poi aggiungere altre parti da analizzare integrandole con quanto già modellato.

Adatta a sistemi che hanno un "core" (nucleo) molto ben definito e coerente, intorno al quale

ruotano una serie di altri elementi lascamente correlati al nucleo.

Ad esempio la progettazione di un intero sistema ferroviario si focalizza prima di tutto sul treno e le rotaie, poi sulle stazioni e le altre infrastrutture.

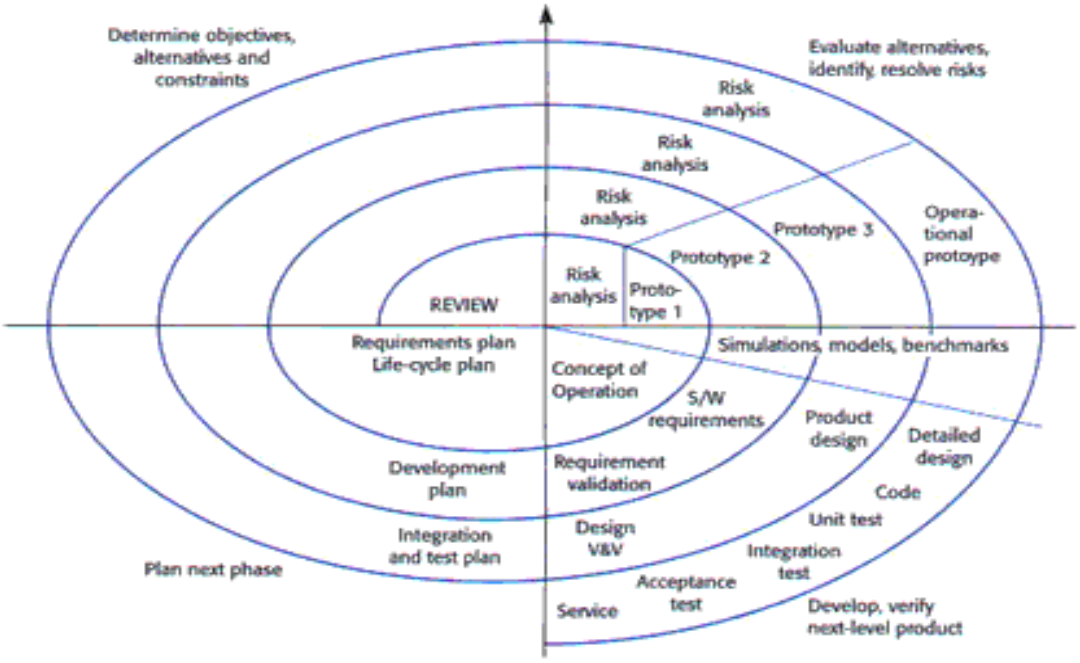
Completata l'analisi si procede poi all'implementazione del sistema completo.

Non facilita una visione globale ed è poco astratta.

Prototipale (non su libro)

Quando il dominio non è noto, un approccio possibile è quello prototipale, che prevede la creazione appunto di un prototipo sul quale verificare i concetti di base e discutere le caratteristiche del sistema con l'utente, e poi sulla base di quanto raccolto procedere con una strategia classica di implementazione.

Ciclica o a Spirale(non su libro)

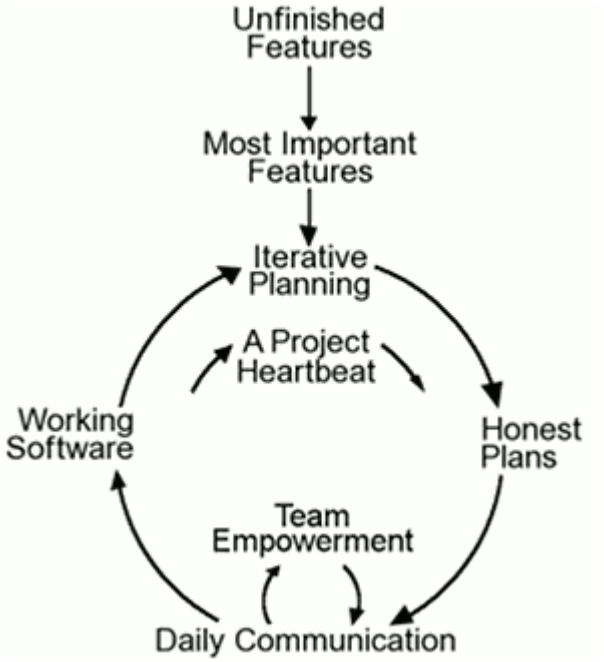


Si procede come nell'Inside-out, ma spingendo oltre l'analisi fino all'implementazione completa della parte, per poi fare nuovi cicli di analisi e implementazione di parti sempre più ampie del sistema.

eXtreame Programming (XP - non su libro)

Tra gli aspetti caratteristici dell'extreme programming vi sono la programmazione a più mani (generalmente in coppia), la verifica continua del programma durante lo sviluppo per mezzo di programmi di test e la frequente reingegnerizzazione del software, di solito in piccoli passi incrementali, senza dover rispettare fasi di sviluppo particolari.

E' un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente, ottenendo in tal modo una elevata reattività alle sue richieste.



Feedback a scala fine

- Pair Programming - significa che tutto il codice viene prodotto da due programmatori che lavorano insieme su una sola workstation.
- Planning Game - è una riunione di pianificazione che avviene una volta per iterazione, tipicamente una volta a settimana.
- Test-driven Development - i test unitari vengono scritti prima di scrivere il codice. Test funzionali e unitari.
- Whole Team - in XP, il "cliente" non è colui che paga il conto, ma la persona che realmente utilizza il sistema. Il cliente deve essere presente e disponibile a verificare (sono consigliate riunioni settimanali).

Processo continuo

- Continuous Integration - Integrare continuamente i cambiamenti al codice eviterà ritardi più avanti nel ciclo del progetto, causati da problemi d'integrazione.
- Refactoring o Design Improvement - riscrivere il codice senza alterarne le funzionalità esterne, cambiando l'architettura, in modo da renderlo più semplice e generico.

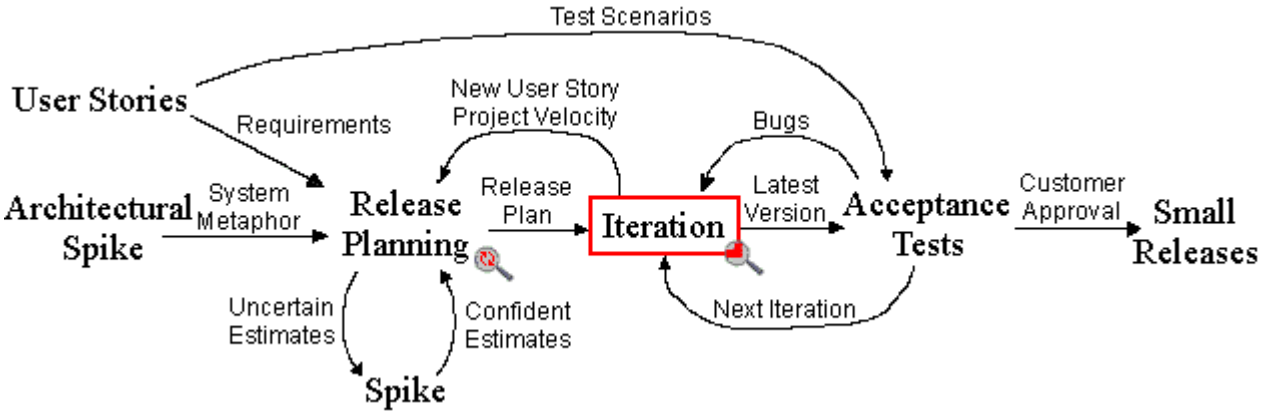
- Small Releases - consegna del software avviene tramite frequenti rilasci di funzionalità che creano del valore concreto.

Comprensione condivisa

- Coding Standards - Scegliere ed utilizzare un preciso standard di scrittura del codice. Questo rappresenta un insieme di regole concordate, che l'intero team di sviluppo accetta di rispettare nel corso del progetto.
- Collective Code Ownership - significa che ognuno è responsabile di tutto il codice; quindi contribuisce alla stesura chiunque sia coinvolto nel progetto.
- Simple Design - i programmatori dovrebbero utilizzare un approccio del tipo "semplice è meglio" alla progettazione software. Progettare al minimo e con il cliente.
- System Metaphor - descrivere il sistema con una metafora, anche per la descrizione formale. Questa può essere considerata come una storia che ognuno - clienti, programmtori, e manager - può raccontare circa il funzionamento del sistema.

Benessere dei programmatori

- Sustainable Pace - il concetto è che i programmatori o gli sviluppatori software non dovrebbero lavorare più di 40 ore di lavoro settimanali.



Strumenti di progettazione assistita (CASE)

Computer-Aided Software Engineering.

Gli strumenti C.A.S.E. aiutano lo sviluppo del software attraverso interfacce grafiche, automatismi, generatori e librerie di funzionalità.

Obiettivi degli strumenti CASE :

- semplificare la scrittura di codice
- automatizzare i passi ripetitivi (un buon programmatore non ripete tre volte lo stesso processo, ma scrive un programma che lo esegue infinite volte!)
- garantire conformità agli standard
- facilitare il lavoro cooperativo

Tipologie:

1. **Tools** - focalizzati su singoli task del processo di sviluppo del software
2. **Workbenches** - insieme di tool coordinati per supportare un intero segmento dello sviluppo
3. **Environments** - ambiente integrato che copre “tutto” il ciclo di sviluppo

Progettazione Logica

Fasi della progettazione logica:

- analisi delle prestazioni
- ristrutturazione degli schemi
- traduzione in modello relazionale

Non si tratta di un processo lineare in tre passi ma di una serie di cicli di raffinamento, soprattutto tra i primi due passi.

Infatti una particolare ristrutturazione può richiedere una nuova verifica delle prestazioni per controllare il raggiungimento e/o il mantenimento degli obiettivi di performance desiderati, ed un'analisi delle prestazioni insoddisfacenti può scatenare una ristrutturazione dello schema con lo scopo di migliorarle.

Analisi delle prestazioni

L'analisi delle prestazioni si basa sulla valutazione del costo di un'operazione in termini di accessi al DB e di memoria necessaria.

Alla base vi è la descrizione del modello dati e l'individuazione delle attività svolte dal sistema.

E' necessario quindi avere:

- volume dei dati del modello (numerosità e dimensione media)
- caratteristiche delle operazioni
  - tipologia (batch / onLine)
  - frequenza
  - struttura dell'operazione (le query coinvolte per poter capire quali sono i dati toccati)

Ristrutturazione degli schemi

Passi di ristrutturazione

- analisi delle ridondanze
- trasformazione delle generalizzazione
- eliminazione attributi multi-valore
- accorpamento entità (relazioni 1-1)
- partizionamento verticale di entità (parte dei dati di un'entità non vengono quasi mai usati)
- identificatori principali

La ristrutturazione deve essere guidata di due obiettivi:

1. pulizia del disegno
2. ottimizzazione delle prestazioni

Questi due obiettivi spesso vanno di pari passo, ma a volte divergono, ed è quindi necessario trovare il giusto equilibrio tra “perfezione formale” del disegno e “sporca ottimizzazione” delle operazioni.

UML

UML : Unified Modeling Language - "linguaggio di modellazione unificato"

Linguaggio di modellazione e specifica basato sul paradigma object-oriented.

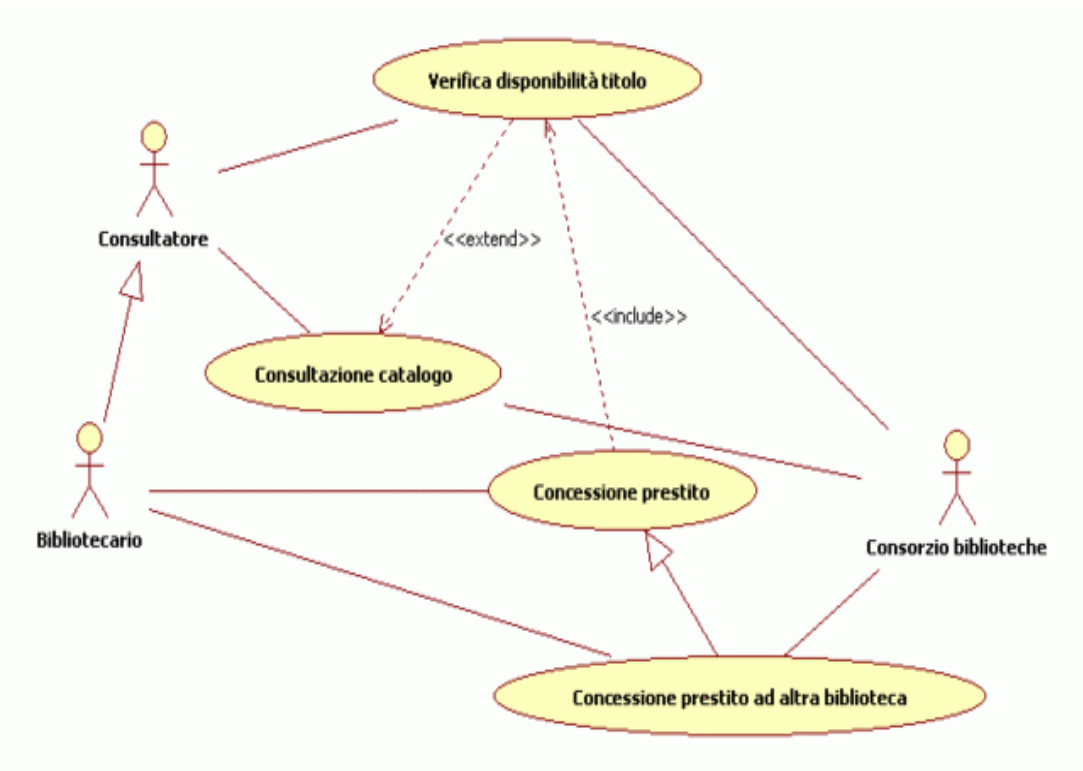
Il nucleo del linguaggio fu definito nel 1996 da "i tre amigos" (Grady Booch, Jim Rumbaugh e Ivar Jacobson) all'interno dell'OMG (Object Management Group), raccogliendo le best practices per definire uno standard unico.

UML svolge la funzione di "lingua franca" della progettazione e programmazione a oggetti, descrivendo soluzioni progettuali in modo standard, sintetico e comprensibile a tutti.

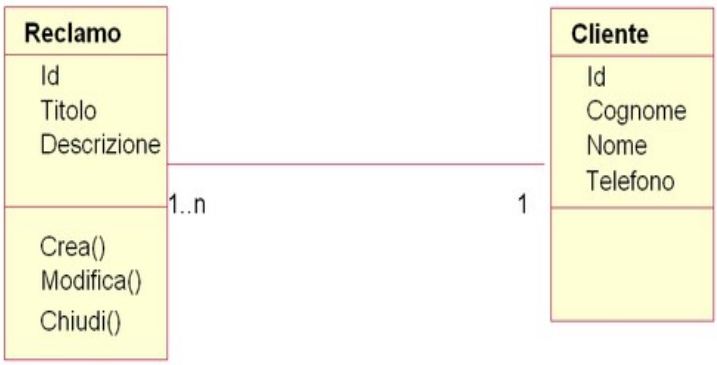
I diagrammi classici UML

- Use Case Diagram
- Class Diagram
- Object Diagram
- Statechart Diagram
- Activity Diagram
- Sequence Diagram
- Communication Diagram / Collaboration Diagram
- Component Diagram
- Deployment Diagram

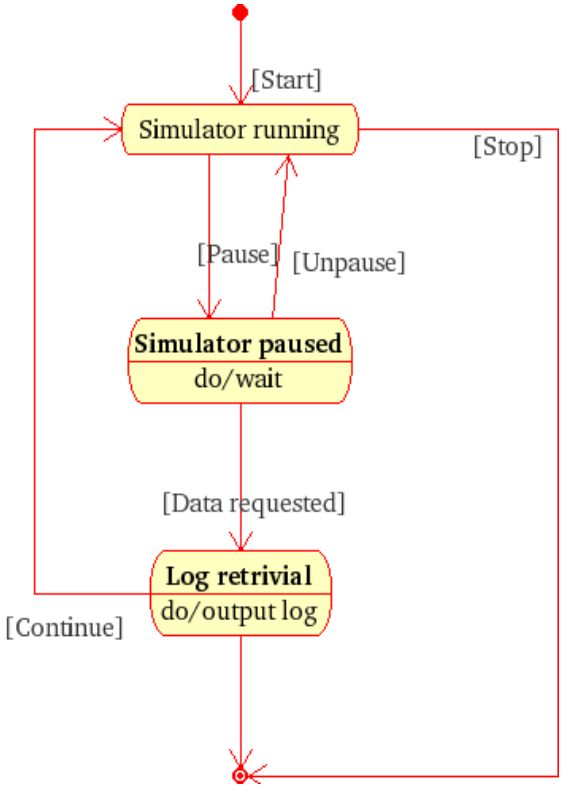
Use Case Diagram



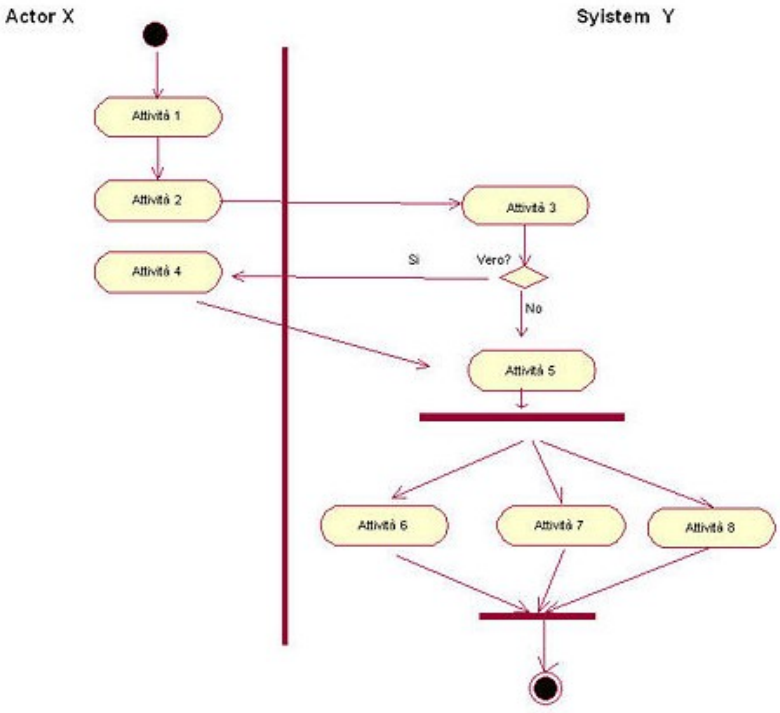
Class Diagram



Statechart Diagram



Activity Diagram



Sequence Diagram

Normalizzazione

Anomalie

Ridondanza
→ anomalia di aggiornamento
→ anomalia di inserimento
→ anomalia di cancellazione

Dipendenza Funzionale

•sia data una relazione ***r*** su ***R(X)***
•si considerino due sottoinsiemi non vuoti ***Y*** e ***Z*** di ***X***
•si dice che esiste in ***r*** una ***dipendenza funzionale*** (FD) da ***Y*** a ***Z*** se, per ogni coppia di tuple ***t1*** e ***t2*** di ***r*** aventi ***Y*** identico (cioè con gli stessi valori su Y), risulta che ***t1*** e ***t2*** sono identiche anche su ***Z*** (cioè hanno gli stessi valori anche su Z)

In altri termini si può dire che Y implica Z :

Y→ Z

Si hanno delle dipendenze funzionali BANALI quando
•Y → A con A appartiene a Y
Per simmetria
•Y → Z è non banale se nessun attributo in Z appartiene a Y

Una dipendenze funzionali può causare anomalie se la componente Y "di partenza" non corrisponde ad una chiave della relazione in esame

le FD non si "ricavano" dall'analisi dei dati,
ma ragionando sugli attributi dello schema

Forme normali

Si definisce che una base di dati è in Forma Normale (o soddisfa una certa forma normale) se sono rispettate delle proprietà che garantiscono l'assenza di determinati difetti strutturali.

Queste proprietà sono dette "Forme Normali".

Le "Forme Normali" sono quindi proprietà che si riferiscono ad una o più relazioni (intese in senso "relazionale") o a strutture ad esse equivalenti, e che indicano se tale relazioni garantiscono un determinato livello di consistenza dei dati.

Vengono definite 5+1 forme normali, che sono via via più stringenti.

Prima Forma Normale (1NF)

Una relazione è in 1NF se:

- non presenta attributi non atomici (ad esempio multipli)
- ha definita una chiave primaria

Tabelle devono descrivere entità singole - Una colonna un valore

Seconda Forma Normale (2NF)

Una relazione è in 2NF se:

- è in 1NF
- tutti i campi non in chiave dipendono funzionalmente dall'intera chiave primaria (e non da una parte di essa)

La chiave, solo la chiave, nient'altro che la chiave
2NF viene ottenuta spezzando tabelle in parti normalizzate che descrivano una singola entità

Terza Forma Normale (3NF)

Una relazione r è in 3NF se:

- è in 2NF
- per ogni dipendenza funzionale non banale $X \rightarrow Y$ definita su r
  - X contiene una chiave K di r
  - ogni attributo in Y è contenuto in almeno una chiave di r

Rimuovere colonne calcolate e creare tabelle di lookup

Vi è un importante teorema :

Ogni relazione può essere trasformata in modo da soddisfare la 3NF

Forma normale di Boyce e Codd (BCNF)

Una relazione r è in forma normale di Boyce e Codd se e solo se:

- per ogni dipendenza funzionale (non banale) $X \rightarrow Y$ definita su di essa, X contiene una chiave K di r

Si può dire anche che X dev'essere una superchiave di r

Se una relazione ha una sola chiave, allora essa è in BCNF se e solo se è in 3NF .

La BCNF garantisce un ottimo livello di consistenza dei dati, ma:
non sempre si può trasformare uno schema in modo che soddisfi la BCNF.

Quarta Forma Normale (4NF)

Una **base di dati** è in 4NF se:

- per ogni relazione di dipendenza funzionale molti a molti $X \rightarrow Y$, X è una superchiave

Quinta Forma Normale (5NF)

Una **base di dati** è in 5NF se:

- è in 4NF
- ogni relazione di dipendenza funzionale è implicata dalle chiavi delle relative tabelle

Processo di normalizzazione

La normalizzazione di un modello è il processo che mira ad individuare se e dove intervenire sul modello stesso al fine di eliminare attraverso la trasformazioni in una delle varie forme normali le anomalie, e con esse i rischi d'inconsistenza e ridondanza.

La normalizzazione non è una metodologia di progettazione delle Basi di Dati ma uno strumento di verifica.
La verifica pre normalizzazione mira ad identificare le anomalie che sono presenti in un modello relazionale e quindi i punti dove intervenire.

Più alto è il livello di normalizzazione raggiunto da un modello e maggiore è l'affidabilità dei dati in esso contenuti, ma maggiore è anche lo sforzo per gestire tali dati.

Uno schema in 4 o 5 NF ha un altissimo livello di rigore formale e garantisce un alto grado di non ridondanza e non anomalie, ma a costo di un'elevato degrado delle prestazioni, quindi non si utilizzano quasi mai, o meglio non si spinge il processo di normalizzazione di un modello fino al 4 o 5 NF.

La terza forma normale è meno restrittiva della forma normale di Boyce e Codd (e ammette relazioni con alcune anomalie) , ma ha il vantaggio di essere sempre “raggiungibile” .
Normalmente quindi ci si “accontenta” di un modello in 3NF.

Il processo di normalizzazione mira ad eliminare da un modello relazionale le anomalie identificate da una precedente analisi, ed è un percorso che permette di trasformare schemi non normalizzati in schemi che soddisfano una determinata forma normale.

In tale processo ad esempio si sostituiscono le relazioni che non soddisfano la BCNF con il risultato di una decomposizione sulla base delle dipendenze funzionali di tali relazioni in altre relazioni che soddisfano la BCNF.

Primo passo semplice di normalizzazione, non sempre valido ed applicabile:

- per ogni relazione che contiene una dipendenza funzionale $X \rightarrow Y$ che viola la BCNF si definisce una nuova relazione XY e si eliminano dalla relazione originaria gli attributi Y

Decomposizioni senza perdite (*lossless*)

Non sempre scomponendo una relazione per ottenere un modello BCNF si ottiene un insieme di dati che preserva tutte e sole le informazioni del modello precedente
Scomponendo la relazione si creano delle nuove relazioni che riaggregate introducono dei dati precedentemente non presenti.

•Una relazione r si decompone senza perdita su X1 e X2 se il join delle proiezioni di r su X1 e X2 è uguale a r stessa (cioè non contiene ennuple spurie)

•La decomposizione senza perdita è garantita se gli attributi comuni contengono una chiave per almeno una delle relazioni decomposte

Uno schema R(X) si decompone senza perdita negli schemi R1(X1) eR2(X2) se, per ogni istanza legale r su R(X), il join naturale delle proiezioni di r su X1 e X2 è uguale a r stessa.

Mantenimento delle dipendenze

•Una decomposizione conserva le dipendenze se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti

Decomposizione in 3NF

•si crea una relazione per ogni gruppo di attributi coinvolti in una dipendenza funzionale
•si verifica che alla fine una relazione contenga una chiave della relazione originaria

- 1NF: Una colonna un valore. Rimuovere gruppi ripetuti
- 2NF: Spezzare in tabelle che descrivano entità separate. Spezzare le PK composte
- 3FN: Rimuovere colonne calcolate e creare tabelle di lookup

Denormalizzazione

La normalizzazione non va intesa come un obbligo, in quanto in alcune situazioni le anomalie che si riscontrano in schemi non normalizzati sono un male minore rispetto alla situazione che si viene a creare normalizzando

In particolare:

- Normalizzare elimina le anomalie, ma può appesantire l'esecuzione di certe operazioni
- La frequenza delle modifiche incide su qual è la scelta più opportuna (relazioni "quasi statiche" danno meno problemi se non normalizzate)
- La ridondanza presente in relazioni non normalizzate va quantificata, per capire quanto incida sull'occupazione di memoria, e sui costi da pagare quando le repliche di una stessa informazione devono essere aggiornate

A volte si decide di non normalizzare o addirittura di denormalizzare strutture normalizzate.

Quando Denormalizzare

- Performance
- Quando gli utenti lo richiedono (anche se questo puo' essere evitato)

Se Denormalizzate

- **Farlo deliberatamente**
- **Avere una ottima regione per farlo**
- **Essere ben consci di cosa questo comporti in termini di performance**
- **Documentare la decisione**