**1**

In questo primo blocco di slide inizieremo a introdurre l'argomento generale delle basi di dati e quindi vedremo cosa sono le basi di dati, a cosa servono, quali sono i vantaggi che porta l'utilizzo delle basi di dati e faremo qualche cenno a come si usano e chi usa le basi di dati.

I più attenti tra di voi avranno notato che ho usato, nelle slide precedenti, circa un centinaio di volte le parole basi di dati e a questo punto è lecito chiedersi: “Ma cosa sono le basi di dati?”. Quindi “alzi la mano chi sa già cos'è una base di dati”, vi chiederei se fossimo in presenza, ma ovviamente non posso farlo, quindi vediamo una prima definizione. Una base di dati è un insieme organizzato di dati utilizzati per il supporto allo svolgimento di attività di un ente, un'azienda, un ufficio, una persona.

Questa è una definizione che sicuramente non è scorretta ma è per ora piuttosto vaga, e in effetti poi la preciseremo e la capiremo meglio. Quindi a cosa servono le basi di dati, qual è l’esigenza, quali sono i problemi che risolvono le basi di dati, perché ne parliamo? Adesso possiamo iniziare a introdurre alcuni punti di vista.

Iniziamo da un punto di vista che può essere più familiare per un informatico. Facciamo conto di essere dei programmatori e un committente ci chiede di sviluppare un programma per gestire le sue attività.

Questo committente potrebbe essere per esempio una università che vuole gestire i dati degli studenti, potrebbe essere una compagnia di assicurazioni che vuole gestire i dati dei suoi assicurati, delle sue polizze, potrebbe essere una anagrafe che vuole gestire i dati dei cittadini di un comune, per esempio.

Quindi siamo dei programmatori, abbiamo questo compito, voi avete già frequentato dei corsi di programmazione e quindi avete dei rudimenti di programmazione di alcuni linguaggi, Java e C, per esempio. Immaginiamo quindi di voler scrivere un programma che gestisce la segreteria studenti dell'Università di Torino.

Immaginiamo quali sono i dati che dobbiamo gestire nella segreteria studenti dell'Università di Torino: dobbiamo tenere conto dei dati anagrafici degli studenti, nome, cognome, indirizzo, recapito telefonico, e-mail, sia domicilio che residenza, e così via, poi dobbiamo tener conto, per esempio, dei dati dei corsi, degli insegnamenti che questi studenti hanno nel proprio piano carriera, tener conto per ogni studente di tutti gli esami a cui questo studente si iscrive/supera, con i relativi voti e le date, bisogna tener conto degli studenti una volta laureati del fatto che si sono laureati, delle loro tesi di laurea.

Quindi immaginate la mole di dati che bisogna gestire per un singolo studente: uno studente può avere più di venti esami, ovviamente uno studente può anche essere iscritto alla magistrale, quindi dobbiamo tenere conto di tutti gli esami che ha dato, sia nella triennale che nella magistrale, moltiplichiamo per il numero di studenti che afferiscono a un corso di laurea, quindi centinaia di studenti che afferiscono al primo anno di un corso di laurea come quello di informatica, moltiplichiamolo per il numero di anni che dura un corso di laurea triennale, sommando il numero di anni di corso di laurea magistrale, e teniamo conto del fatto che ci sono tanti corsi di laurea in tante scuole diverse, informatica, biologia, chimica, lingue, giurisprudenza e così via e teniamo conto del fatto che bisogna memorizzare non solo i dati degli studenti attuali: se uno studente si è laureato, non dobbiamo buttare via i suoi dati, dobbiamo mantenere ancora i suoi dati in qualche modo, perché potrebbero essere utili.

Quindi cosa vuol dire? Vuol dire che abbiamo bisogno di memorizzare una grande quantità di dati e abbiamo bisogno di memorizzare, probabilmente, gigabyte e gigabyte o terabyte di dati in totale.

Ora noi siamo i programmatori e dobbiamo sviluppare questo programma, una cosa che non possiamo fare è quella di pensare di costruire/definire le strutture dati, degli array, delle liste, degli alberi, che poi serializziamo, cioè memorizziamo su memoria secondaria, e poi, quando il programma parte, carica tutti i dati di tutti gli studenti di tutte le università di tutti gli anni, in memoria principale. Questo non possiamo farlo perché la memoria principale non è sufficientemente grande per contenere tutti questi dati, e la gestione di questi dati in memoria principale, quindi, non è fattibile per questioni di capienza della memoria principale. Non possiamo neanche pensare che sia sufficientemente efficiente la gestione in tutti questi dati, anche se riuscissimo a comprare sufficiente RAM per contenere tutti questi dati.

Quindi dal punto di vista del programmatore vediamo che ci sono alcuni casi, come questo della segreteria studenti dell'università, in cui non è comodo gestire tutti questi dati, attraverso il modo nativo fornito da un linguaggio di programmazione all'interno di una applicazione.

Adesso proviamo spostare il punto di vista: invece di essere un programmatore, assumiamo il punto di vista dell'organizzazione, quindi, per esempio, immaginate di essere il direttore della segreteria studenti o, se volete, il rettore dell'università, oppure l'amministratore delegato della compagnia di assicurazioni, o direttore dell’anagrafe. Quindi ora voi non siete dei programmatori, ma siete quella persona che ha la necessità di far funzionare questo ente, questa organizzazione. Non discuto, per brevità, della possibilità di gestire queste informazioni senza computer; anche se non è scontato che il computer sia sempre la scelta migliore per gestire delle informazioni, per ora assumiamo che sia la scelta giusta, quindi vogliamo usare i computer in modo da gestire queste informazioni.

Se voi siete l'amministratore delegato di questa compagnia di assicurazioni, che ha necessità di gestire i dati di tutti i suoi assicurati, di tutte le polizze degli assicurati, tutti i dati di ogni assicurato, comprendendo anche i dati che servono per calcolare il premio di rischio della polizza, quindi il numero di familiari, l'età dei familiari, il luogo in cui uno vive, il lavoro che uno fa e così via. Devo gestire questi dati. Potrei pensare di assumere un programmatore, assegnare questo compito ad una software house di sviluppare un'applicazione che mi aiuti a gestire questi dati.

Quale sarebbe un problema? Se questa software house, che io assoldo per costruire il mio programma, dovesse per esempio fallire, io come faccio ad accedere ai miei dati? La software house mi ha consegnato il prodotto software, che io uso felicemente per cinque anni, dieci anni, per gestire i miei dati: cosa succede se questa società a un certo punto fallisce e ha usato queste tecniche, che discutevamo prima, di programmazione: array, alberi, liste, per accedere ai dati e poi li serializza su hard disk e faccio conto che sia uscita una versione di Windows, MacOS, Linux, che fa in modo che questo programma non funzioni più su questa nuova versione, e non riesco più a lanciare il prodotto software che ho comprato?

Se non riesco a lanciare più quel software, non riesco più ad accedere ai dati che sono memorizzati con quel software, quindi io ho un mio patrimonio, in termini di dati - infatti i dati possono essere un patrimonio su cui si basa un'intera organizzazione - a cui non ho più accesso.

Quindi cosa ci insegnano questi due punti di vista? Ci insegnano un fatto che non è scontato a prima vista, ed è un fatto che non era scontato neanche degli anni ‘70, quando sono nate le prime proposte che riguardavano le basi di dati, cioè di considerare i dati come qualcosa di separato dal programma, dall'applicativo, che serve per risolvere i problemi in una certa organizzazione.

Detto in un altro modo, io sono il rettore dell'università, sono l'amministratore delegato della compagnia di assicurazioni, sono il direttore dell'anagrafe, ho dei prodotti software che mi aiutano a gestire queste aziende, ma questi prodotti software non sono l'unico modo per accedere ai dati, non sono la risorsa che “possiede” i dati, cioè l'unico modo con cui posso leggere/gestire i dati.

Separo i dati e li gestisco con un prodotto a sé stante, cioè i dati hanno una vita a sé, e quindi riconosciamo il fatto che i dati sono un patrimonio importante, sono una risorsa importante, e non vogliamo legarci in modo indissolubile a uno specifico applicativo, non vogliamo che sia l'unico modo che abbiamo per accedere a questi dati.

Dal punto di vista del programmatore ho anche un vantaggio nel fare così, perché posso trovare già risolti “gratis” alcuni problemi in cui devo decidere come faccio a rappresentare questi dati, come faccio a caricare in memoria questa grossa mole di dati, come la rappresento e come faccio ad accedere in modo efficiente a questi dati. Quindi noi separiamo la gestione dei dati dalla gestione del resto del software, e consideriiamo questa base di dati come un'entità a sé stante che merita un trattamento a parte.

Cos'è un sistema informativo? Prima di tutto notate che in sistema informativo c'è una v, non una c: non è un sistema informatico. Si chiama sistema informativo perché è un sistema che gestisce delle informazioni e non è detto che usi un computer, quindi non è detto che sia un sistema informatico.

Ogni organizzazione ha un sistema informativo perché ogni organizzazione, ogni ente, azienda di ogni dimensione ha la necessità di gestire delle informazioni.

Nelle organizzazioni piccole questo sistema informativo può essere “invisibile”, non essere esplicito, non essere presente nella struttura dell'organizzazione. Quando, invece, l'organizzazione ha una certa dimensione significativa, allora questo sistema informativo è esplicitamente presente nella sua organizzazione.

Parliamo di sistema informativo perché stiamo parlando di basi di dati, e le basi di dati sono una componente molto importante di un sistema informativo, anche se non sono l'unica componente di un sistema informativo, e non necessariamente sono presenti, per esempio se il sistema informativo non è automatizzato.

Quando parliamo di gestire le informazioni, cosa stiamo dicendo? Gestire informazioni vuol dire che queste informazioni subiscono alcuni trattamenti, quindi prima di tutto queste informazioni devono essere acquisite, cioè devono essere raccolte, all'interno del sistema informativo. Poi, devono essere conservate, salvate, archiviate. Dopo possono essere elaborate in modo da produrre nuove informazioni, quindi trasformare informazioni esistenti per produrre informazioni nuove. Dopodiché queste informazioni possono essere mandate al di fuori del sistema informativo, distribuite, comunicate, scambiate con altre persone.

Possiamo fare l'esempio della segreteria studenti dell'università. La raccolta delle informazioni avviene quando uno studente, per esempio, elabora un piano carriera, decide quali sono i corsi che vuole mettere nel suo piano carriera, oppure quando il professore verbalizza i risultati di un esame: vengono raccolte queste informazioni all'interno del sistema informativo.

Poi avviene l'archiviazione/conservazione: ovviamente questo sistema informativo della segreteria studenti deve memorizzare queste informazioni in modo da poterle reperire anche in un secondo momento.

Dopo l'elaborazione, trasformazione e produzione avviene, per esempio, quando calcoliamo la media, e calcolare la media dei voti degli esami è un esempio di elaborazione delle informazioni: produciamo un'informazione che non era esplicitamente presente nelle informazioni immesse nel sistema.

Dopodiché questa informazione può essere distribuita/comunicata, per esempio, quando uno studente accede alla propria pagina sul sito dell'università e visualizza i risultati dei vari esami che ha superato. In questo caso il sistema informativo distribuisce l'informazione relativa a questi esami all'esterno, per esempio, attraverso la pagina web del sito del dell'università.

Un sistema informativo non è necessariamente un sistema informatico: sono due concetti che possono essere dipendenti tra di loro. Un esempio che si può fare è quello dei banchieri fiorentini del Rinascimento (nel Rinascimento è nato il sistema bancario). I banchieri fiorentini facevano prestiti a enti, a stati, a sovrani e avevano ovviamente la necessità, come una banca odierna, di tenere traccia di quali sono i prestiti, quando sono stati fatti, quando sono stati restituiti, se in totale o in parte, a quali tassi d'interesse e così via. Per memorizzare queste informazioni usavano dei registri, ovviamente cartacei: questi registri costituivano un sistema informativo di questa organizzazione bancaria e ovviamente, nel Rinascimento italiano, non esistevano ancora i computer, quindi questo è un esempio di un sistema informativo non informatico.

Un sistema informatico quindi cos'è? È la porzione informatica/automatizzata del sistema informativo, quindi quella porzione del sistema informativo che viene gestita con i computer.

In questo schema vediamo una rappresentazione del sistema azienda e la posizione che ha un sistema informatico all'interno di un sistema azienda. Quindi un’azienda ha al suo interno un sottoinsieme che è l'organizzazione dell'azienda; una parte di questa organizzazione è relativa alla gestione delle informazioni dell'azienda, che è il sistema informativo; una porzione del sistema informativo è automatizzata e costituisce il sistema informatico di questa azienda, cioè il sistema come abbiamo detto, che gestisce in modo automatizzato le informazioni dell'azienda.

Finora ho usato queste due parole, informazioni e dati, in modo intercambiabile, e facendo questo ho compiuto un abuso.

Qual è la differenza tra la parola informazione e la parola dato? In questa slide ho riportato le definizioni delle due parole tratte dal vocabolario della lingua italiana Treccani edizione 1987. La Treccani, per informazione ci dice che è una notizia, un dato, un elemento che consente di avere una conoscenza più o meno esatta dei fatti, situazioni e modi di essere.

Per dato, il vocabolario ci dice che è ciò che immediatamente presente alla conoscenza, prima di ogni elaborazione; in informatica sono elementi di informazione costituiti da simboli che debbono essere elaborati. Queste sono definizioni che per i nostri fini informatici possono essere un po’ vaghe e hanno il problema di essere intrecciate tra di loro - ricorsive diciamo - perché per definire informazione, viene usata la parola dato, e per definire la parola dato viene usata la parola informazione. Adesso precisiamo meglio che cosa intendiamo per informazione e dato.

Per illustrare qual è la differenza tra dato e informazione facciamo un esempio, guardiamo queste figure. Quindi 8-17, (8-13) e 8-14 scritto in rosso.

Cosa significano questi numeri? Hanno un significato? Possiamo pensare che questi sono dei dati cioè qualcosa che è immediatamente presente alla percezione, però per capire qual è la conoscenza che sta dietro e che dovrebbero comunicare, hanno bisogno di un contesto, cioè di un'interpretazione.

Una parte dell'interpretazione può essere data dalla prossima figura. Vediamo che c’è il simbolo di divieto di sosta, quindi capiamo che questi sono dei cartelli stradali finlandesi, che immagino si usassero un tempo, e quindi capiamo che quei numeri sono degli orari, cioè la fascia in cui è valido il divieto di sosta.

A questo punto però non conosciamo ancora un pezzo di informazione, cioè non sappiamo qual è la differenza tra questi tre cartelli di sosta: vediamo che c'è una differenza ma non sappiamo quale sia il significato di questa differenza.

Per saperlo abbiamo bisogno di un altro pezzo di informazione. Per esempio, possiamo scoprire che 8-17, scritto in quel modo, vuol dire che divieto di sosta è nei giorni feriali, tra lunedì e venerdì, che se l'orario è scritto tra parentesi tonde significa che è un diritto di sosta, che vale di sabato, e invece quando è scritto in rosso vuol dire che è un divieto di sosta che è valido nei giorni festivi.

Quindi abbiamo visto che c'è una differenza tra un dato e un'informazione, che è qualcosa di diverso da un dato. Senza interpretazione il dato non serve a molto, non ci è molto utile.

Quando parliamo di basi di dati possiamo parlarne usando due accezioni: c'è una accezione generica e una accezione specifica.

L'accezione generica, che abbiamo già visto, è che una base di dati è un insieme organizzato di dati utilizzati per il supporto allo svolgimento dell'attività di un ente.

L'accezione specifica/tecnologica, è che la base di dati è l'insieme dei dati che viene gestito da un DBMS.

Ma cos'è un DBMS? DBMS è un acronimo che sta per DataBase Management Systems, cioè in italiano sistema di gestione delle basi di dati.

È la componente software che usiamo per gestire le basi di dati. Una componente software che non fa parte, come nell'esempio che abbiamo fatto prima del programmatore, dell'applicazione che un programmatore sviluppa, ma è una componente a parte, che è specificamente dedicata a gestire le basi di dati.

Sulla slide vedete i principali database che sono utilizzati al mondo, elenco il nome del database e la data in cui è nato, con qualche caratteristica.

Partiamo da Oracle DB, che è del 1979. È un DBMS proprietario molto diffuso commercialmente, molto potente.

Un altro DBMS è PostgreSQL (si legge postgres) del 1989. È un DBMS open source, cioè i sorgenti sono liberamente disponibili, scaricabili e modificabili, è potente e ampiamente aderente agli standard. È quello che noi useremo nel laboratorio, quindi, quando inizieremo a parlare di SQL, useremo, per far girare le query SQL, PostgreSQL.

Un altro DBMS molto diffuso è MySQL, o Maria DB, del 1995. Inizialmente era libero, poi è stato comprato da Oracle, che quindi possiede due DBMS diversi, Oracle DB, che è quello più costoso, più potente e professionale, e MySQL, che è molto diffuso nelle applicazioni web. Le versioni più vecchie, di qualche anno fa, avevano alcune limitazioni nonostante la grande diffusione e non erano aderenti agli standard, ma negli ultimi anni, dopo l'acquisizione le cose sono un po’ cambiate e quindi è diventato più potente e più aderente agli standard.

Poi un altro DBMS è SQL Server di Microsoft, del 1989, che è un altro DBMS proprietario, ha un limitato supporto a sistemi operativi diversi da Windows.

Microsoft ha un secondo DBMS, molto più limitato di SQL Server, che è Access, del 1992. È un altro DBMS proprietario, ha limitazioni, viene usato per uso personale, quindi SQL Server ha un uso più professionale, mentre Access ha un uso più personale, integra al suo interno un ambiente di sviluppo grafico, ed è disponibile solo su Windows.

L'ultimo DBMS che elenco è SQLite (lite sta per leggero), del 2001. È un DBMS open source, è molto particolare perché, a differenza degli altri, è contenuto in una libreria C, quindi non è un altro pezzo di software che deve essere installato, non è client/server, ma è una libreria C che si linka nella vostra applicazione ed è un DBMS “embedded”, cioè contenutoo all'interno dell'applicazione. È molto leggero e molto veloce, è limitato rispetto a quelli potenti, come PostgreSQL, SQL Server, eccetera. È probabilmente il DBMS più diffuso perché tutti i browser hanno una cronologia della navigazione, hanno i preferiti, e questi sono dei database che sono gestiti da SQLite; quindi ogni browser, che sia Firefox, Chrome, Safari, Edge e così via, avrà all'interno un SQLite che gestisce il database della cronologia di navigazione e dei preferiti.

**2**

Ora vediamo alcune slide in cui caratterizziamo le basi di dati, quindi discutiamo quali sono le caratteristiche principali delle basi di dati - vediamo questo argomento ad alto livello -, e discutiamo quali garanzie forniscono le basi di dati, cioè quali vantaggi abbiamo quando usiamo un DBMS.

Adesso vediamo ognuno dei punti elencati nelle slide. Una base di dati è grande, persistente e condivisa, e un DBMS garantisce le proprietà di privatezza, affidabilità, efficienza ed efficacia. Se volete una discussione ancora più ampia di questi termini, potete trovarla nel libro consigliato, Atzeni-Ceri-Paraboschi. Adesso vediamo, una per una, queste voci.

Una caratteristica importante delle basi dati è che sono grandi. Cosa vuol dire? Vuol dire che gestiscono così tante informazioni, sono composte da così tanti dati, che non possono essere memorizzate completamente nella memoria principale di un computer, non stanno dentro la RAM.

Le dimensioni di una base di dati sono dell'ordine di gigabyte, terabyte, anche petabyte, e quindi sono molto molto grandi. E una caratteristica importante è che sono così grandi che non possono essere memorizzate per intero all’interno della RAM. Questa considerazione avrà poi conseguenze, di cui parleremo, su come sono fatti i DBMS, sull'architettura di DBMS, perché questi devono riuscire a garantire efficienza nel trattamento dei dati nonostante le memorie secondarie in cui vengono memorizzate le basi di dati siano di diversi ordini di grandezza più lente della memoria principale.

Le basi dati sono anche persistenti. Questo vuol dire che hanno un tempo di vita che non è limitato all’esecuzione delle applicazioni, cioè non succede che lanciamo un'applicazione, facciamo qualche operazione nell'applicazione e poi, quando chiudiamo l’applicazione, si perde quello che abbiamo fatto, ma vogliamo che la vita dei dati non sia limitata alla vita dell'applicazione, quindi i dati devono vivere più tempo rispetto al tempo di vita di un’applicazione.

Le basi di dati sono anche condivise. Questo vuol dire che possiamo immaginare una grande organizzazione, composta da diverse parti, e ogni parte di questa organizzazione potrebbe avere un proprio sistema informativo/informatizzato, e le basi dati dovrebbero essere condivise all'interno di questa organizzazione.

In questo schema, vediamo un primo esempio, nel basso della slide vedete questo cilindro, su cui c'è scritto DB, che è la base di dati. L'accesso ai dati della base di dati avviene attraverso un DBMS, questo DBMS potrebbe essere PostgreSQL, Oracle, MySQL, SQL Server e così via. Le applicazioni che gli utenti finali usano non accedono direttamente alla base di dati, ma ogni accesso dell'applicazione è mediato attraverso il DBMS. Se sono un programmatore, quando sviluppo un'applicazione che ha bisogno di una base di dati, non accedo direttamente al disco, ai file nel file system che memorizzano questi dati del DB, ma uso delle API, dei comandi, delle funzioni, che effettuano richieste al DBMS. Ed è il DBMS che avrà accesso alla base di dati. Quindi tutte queste applicazioni condividono il database e passano attraverso l'interfaccia di un DBMS per avere accesso ai dati.

Quando le basi dati sono condivise abbiamo dei vantaggi perché possiamo, grazie alla condivisione di dati tra più applicazioni, evitare ridondanze e incoerenze.

Ridondanza cosa vuol dire? Quando abbiamo una ridondanza, abbiamo un’informazione ripetuta.

Incoerenza cosa vuol dire? Vuol dire che ci sono errori di “allineamento” dei dati.

Se non condividessimo le basi di dati, dovremmo avere più copie dei dati, ognuna, per esempio, per ogni applicazione. Se abbiamo più copie certe informazioni potrebbero essere ripetute tra le varie copie. Se facciamo l'esempio della segreteria studenti, se le basi dati non sono condivise tra i vari uffici, gli uffici centrali potrebbero avere una copia dei dati degli studenti, gli uffici del dipartimento, del corso di studi di informatica potrebbero avere un'altra copia dei dati. Dunque sono due copie diverse e lo stesso dato dello studente, il suo indirizzo, numero di telefono, l'email è ripetuto in due posti diversi, quindi abbiamo ridondanza e abbiamo una possibile incoerenza. Chi ci garantisce che, quando abbiamo due copie dei dati, queste due copie dicano la stessa cosa, cioè sono allineate? Potrei avere uno studente che cambia residenza, viene segnato nel database degli uffici centrali, ma queste informazioni non vengono replicate nella copia del database del corso di studi. Se abbiamo ridondanze, abbiamo anche la necessità di mantenere coerenti le informazioni che sono ripetute.

In questa slide vediamo un esempio, sempre universitario. È un’interfaccia web che potrebbe essere una schermata di un applicativo che gestisce gli orari delle lezioni. Abbiamo un’università degli studi e un corso di studi in ingegneria informatica. Vediamo gli orari delle lezioni per un anno accademico di alcuni anni fa. Potete notare che ci sono vari insegnamenti: analisi 1, basi di dati, chimica, fisica eccetera, con i vari docenti, Luigi Neri, Piero Rossi, Nicola Mori. Ogni lezione è svolta in un'aula, n1, n2, n3 e così via, con un orario tra le 8 e le 9 e mezza, le 9:45 e le 11:15 e così via.

Stiamo immaginando che questa sia una schermata dell'applicativo che gestisce un orario delle lezioni per questo corso di studi. In quest'altra schermata abbiamo un altro esempio, un altro applicativo che, invece di gestire orari delle lezioni, gestisce gli orari di ricevimento dei docenti. Abbiamo il docente Mario Bruni, nella prima riga della tabella, che ha due insegnamenti, fisica 1 e fisica 2, e l'orario di ricevimento al martedì tra le 10 e le 12. Queste, sono due schermate di due applicativi diversi, due applicativi software diversi tra di loro.

Nello schema che vedete immaginiamo che un’applicazione, nel rettangolo “gestione orario elezioni”, memorizza i propri dati in un archivio, un hard disk, una memoria secondaria. In quest'altra slide vediamo un altro rettangolo, che rappresenta il secondo programma, il secondo applicativo, di gestione dei ricevimenti, il quale memorizza i propri dati in un altro archivio che contiene i dati del ricevimento.

Ma questi due archivi sono così indipendenti tra di loro? Potremmo pensare che non lo siano, perché ci sono alcune informazioni che sono in comune: per esempio, i nomi dei professori sono gli stessi, i nomi dei corsi sono gli stessi e abbiamo bisogno di sapere qual è il nome del corso, il nome del professore, sia nell'applicativo che gestisce l'orario delle lezioni, sia nell'applicativo che gestisce gli orari del ricevimento. Se gestiamo queste informazioni in modo integrato, in un'unica base di dati, possiamo condividere le informazioni tra questi due applicativi e se, per esempio, un domani, il professore che insegna analisi 1 cambia, noi non dobbiamo andare ad applicare questo cambiamento, in cui cambia il nome del professore e cambia l’orario di ricevimento, in due archivi diversi, ma possiamo applicarlo in un unico posto integrato. Questo è un esempio molto piccolo, molto semplice, ma se pensiamo che cambiamo non solo il nome del professore, ma tutti i dati anagrafici di questo professore, allora i cambiamenti potrebbero essere onerosi da dover essere applicati in più posti, e aumenta anche il rischio di avere le diverse copie dei dati non sincronizzate tra di loro.

Quindi condividere i dati, avere un unico posto dove memorizzare i dati, dà dei vantaggi, perché una base di dati, essendo condivisa tra le applicazioni è integrata e permette di evitare ridondanze e incoerenze.

Questo ci dà solo vantaggi? Ovviamente no, perché quando condividiamo queste informazioni abbiamo il problema della gestione della concorrenza, nel senso informatico, cioè il problema di gestire accessi contemporanei di diversi agenti alla medesima risorsa. Gli accessi contemporanei non devono interferire tra di loro, e devono garantire l'integrità dei dati.

I DBMS risultano utili perché a fronte di questi problemi derivati dalla concorrenza, che forse avete incontrato nel corso di sistemi operativi dello scorso semestre, il DBMS fornisce meccanismi che permettono di gestire i problemi derivanti dalla concorrenza e regola gli accessi. In questo modo noi programmatori, quando sviluppiamo un software che usa le basi di dati, abbiamo il vantaggio di non dovere gestire e risolvere ogni volta, per ogni applicativo che sviluppiamo, i problemi della concorrenza, ma demandiamo la risoluzione di tali problemi al DBMS.

I DBMS garantiscono, tra i vari vantaggi che danno, anche la privatezza, cioè dentro il DBMS si possono avere dei meccanismi di autorizzazione, cioè, è possibile definire categorie di utenti, oppure specifici utenti, con un proprio account, che hanno diritti diversi di interagire con i dati.

Per esempio pensiamo a una biblioteca, in cui abbiamo il database della biblioteca, e abbiamo la categoria di utente lettore, che ha il diritto di leggere i dati del database e di cercare i dati, per effettuare ricerche bibliografiche, ma non ha il diritto di modificare o inserire i dati, perché è solo un lettore della biblioteca. C'è anche un'altra figura, che è la figura del bibliotecario, che può non solo leggere, cercare dati, ma può anche modificare i dati, nel senso che può aggiungere nuovi libri che sono arrivati alla biblioteca, oppure può modificare i dati, per esempio, segnare che un libro è stato restituito, oppure che è stato mandato in prestito. Quindi abbiamo due categorie di utenti a cui possono essere assegnati diritti/autorizzazioni diversi per interagire sui dati.

I DBMS garantiscono anche affidabilità. Questo significa che sono resistenti a eventuali malfunzionamenti.

I malfunzionamenti possono essere anche hardware, cioè si rompe il computer mentre sta elaborando qualcosa, si rompe l'hard disk o un'altra componente del computer, oppure anche un malfunzionamento software, cioè un bug nell'applicativo, un bug nel sistema operativo.

Dato che nei DBMS si considera che i dati sono una risorsa pregiata, quindi molto importante per un'organizzazione, i DBMS applicano tecniche che garantiscono che i dati non vengano corrotti a seguito di malfunzionamenti sia hardware che software. Si applicano tecniche che fanno in modo che noi possiamo fidarci di cosa è scritto in una base di dati, anche nel caso in cui un applicativo che interagisce con la base di dati ha dei bug, e magari tenta di scrivere delle informazioni scorrette nella base di dati.

Una tecnica fondamentale per la gestione dell'affidabilità è quella della gestione delle transazioni.

Cos’è una transazione? Questo è un concetto che, come gli altri che stiamo introducendo adesso, introduciamo solo ad alto livello, ad un livello astratto, e nel prosieguo del corso queste informazioni saranno molto più precise e leggermente più formali.

Quindi cos'è una transazione? È un concetto fondamentale nelle basi di dati. Di fatto intendiamo una sequenza di operazioni che vengono considerate indivisibili, cioè vengono considerate atomiche. Atomiche vuole dire che sono diverse operazioni che vogliamo vengano eseguite in modo atomico, cioè non è possibile che la sequenza venga interrotta a metà, quindi non deve succedere che venga eseguita solo qualche operazione e qualche altra invece non venga eseguita a causa di malfunzionamenti hardware o software. Vogliamo che tutte le operazioni che fanno parte di una transazione vengano eseguite in modo atomico anche in presenza di concorrenza, e gli effetti dell'operazione devono avere effetti definitivi.

Vediamo un esempio del motivo per cui vogliamo che le operazioni vengono eseguite in modo atomico. Immaginiamo di avere un sistema informativo di una banca. Una banca ha diversi conti correnti e vogliamo trasferire dei fondi, dei soldi, da un conto A ad un conto B. Per esempio, facciamo un bonifico per trasferire 30 euro dal conto A al conto B. Questo trasferimento è in realtà composto da due operazioni, cioè quando trasferiamo questi 30 euro dobbiamo prelevare 30 euro dal conto A e versare 30 euro sul conto B, quindi sono almeno due operazioni, una che preleva e l'altra che versa.

Vogliamo che queste due operazioni vengano eseguite in modo atomico: cosa succede se effettuiamo prima il prelevamento e poi il versamento, quindi togliamo 30 euro dal conto A, e poi accade un malfunzionamento hardware o software, ad esempio l'applicativo esce, si guasta un hard disk o una scheda madre, e quindi la seconda operazione del versamento non viene eseguita?

Questo è un problema per la banca perché abbiamo tolto dal conto A 30 euro e questi 30 euro sono spariti nel nulla perché non sono stati versati sul conto B.

Al contrario, se viene eseguita prima l'operazione di versamento su B e dopo il prelevamento da A, abbiamo comunque un problema, perché se prima versiamo 30 euro sul conto B e poi avviene un malfunzionamento e non possiamo prelevare i 30 euro dal conto A, la banca non è contenta, perché deve comunque dare 30 euro al conto B senza il prelevamento dal conto a.

Quindi queste due operazioni devono essere eseguite in modo atomico, in modo atomico vuol dire che o le eseguiamo entrambe, ed è il caso che noi desideriamo, ma se non si riesce a rientrare in questo caso, perché c'è qualche malfunzionamento, allora, piuttosto che eseguire solo una delle operazioni, è meglio che non venga eseguita nessuna delle due, quindi né il prelevamento e né il versamento.

Faremo cenno verso la fine del corso di teoria a tecniche dei DBMS che fanno in modo che, se solo una delle due operazioni della transazione viene eseguita e poi c'è un malfunzionamento, il DBMS effettua l’UNDO, cioè disfa le operazioni della transazione che sono rimaste sospese, e in questo modo garantisce l’atomicità dell'esecuzione delle transazioni.

Le transazioni sono anche concorrenti, cioè possiamo avere due transazioni diverse - non sto più parlando di due operazioni all'interno della stessa transazione, ma di due transazioni diverse che al loro interno contengono operazioni -, eseguite in modo concorrente, cioè che contemporaneamente usano la stessa risorsa.

Sulla slide riporto due esempi. Ho due assegni, che sono emessi sullo stesso conto e vengono incassati contemporaneamente. Bisogna fare in modo di considerarli entrambi, anche se sono due azioni contemporanee. Queste due azioni contemporanee sono due operazioni che alterano il saldo dello stesso conto corrente e dobbiamo fare in modo che non rimanga il saldo che rimarrebbe dopo l'esecuzione di una sola delle due operazioni, nonostante vengano eseguite nello stesso momento.

Un altro esempio, forse più immediato, è quello della prenotazione dei posti. Le prenotazioni di posti vengono solitamente gestite da database. Se due persone prenotano contemporaneamente lo stesso posto, che risulta libero, su un treno o su un aereo, nonostante le due operazioni di prenotazione, che sarebbero due transazioni, avvengano sulla stessa risorsa, cioè sullo stesso posto libero, bisogna fare in modo che solo una delle due vada a buon fine, anche se sono eseguite contemporaneamente. Se non lo facessimo, assegneremmo la stessa risorsa, cioè lo stesso posto libero, sul treno o sull'aereo, a due persone diverse, che poi avranno una colluttazione quando salgono sul treno o sull'aereo.

Le transazioni devono essere permanenti, cioè, se un'operazione ha successo e viene conclusa positivamente, il risultato dell’operazione sulla base di dati deve essere memorizzato in modo definitivo. Anche se dopo la fine con successo dell’esecuzione di una transazione c'è un malfunzionamento, comunque, nonostante il malfunzionamento successivo, il risultato della transazione deve essere memorizzato in modo permanente nella base di dati. La conclusione positiva di una transazione si chiama, con una parola in inglese, commit, che è una parola che poi riprenderemo e useremo spesso.

I DBMS devono essere efficienti, ovviamente, cioè, nonostante i DBMS gestiscano delle quantità di dati che sono molto grosse, dell'ordine di megabyte, gigabyte, terabyte, petabyte, le operazioni devono essere eseguite in modo molto efficiente, altrimenti nessuno userebbe un database. Su questo punto c’è un intenso lavoro dei DBMS: i DBMS usano tecniche piuttosto sofisticate per garantire l'efficienza delle operazioni, e cercano di usare nel modo più efficiente possibile le risorse che hanno a disposizione, della memoria principale, memoria secondaria, tenendo conto che la memoria secondaria è molto molto più lenta della memoria principale.

Ultima caratteristica dei DBMS è l'efficacia, cioè i DBMS devono essere ovviamente efficaci, cioè deve essere utile usarli, e devono rendere produttive le attività dei loro utilizzatori, offrendo funzionalità articolate, potenti e flessibili.

**3**

Un database fa riferimento a un modello dei dati, che possiamo pensare come una sorta di costruttore di tipo in un linguaggio di programmazione, cioè è un meccanismo che dice come sono strutturati i dati in un database.

Il modello relazionale ha come modello dei dati quello della relazione, che adesso inizio a introdurre in modo informale, e nel prossimo blocco di slide, invece, definiamo in modo decisamente più formale. Riprendiamo l'esempio dell'orario delle lezioni. Abbiamo questo corso di studi, in cui ci sono queste lezioni, per cui abbiamo un insegnamento, un docente, un'aula, e un orario. Come struttureremmo queste informazioni in un database relazionale? Le struttureremmo, probabilmente, nello stesso modo, o in un modo simile, quindi abbiamo un esempio di una relazione, che a prima vista sembra una cosa molto intuitiva e molto semplice, poi, quando lo approfondiremo meglio, vedremo che in effetti è semplice ma presente comunque alcune difficoltà, non è proprio completamente intuitivo.

A prima vista abbiamo una tabella che ha il titolo orario, in cui abbiamo un'intestazione che è insegnamento, docente, aula, ora, sulle colonne, e abbiamo le varie righe della tabella in cui diciamo qual è l'insegnamento, per esempio, analisi matematica 1, quale docente, Luigi Neri, qual è l'aula, N1, e qual è l'ora di inizio della lezione. Abbiamo una riga per ogni insegnamento e riportiamo le stesse informazioni per ognuno degli insegnamenti. Quindi una prima distinzione che possiamo fare è la distinzione tra schema e istanza, lo schema di questa nostra semplicissima base di dati dice che abbiamo una relazione, che si chiama orario, e per questa relazione orario memorizziamo questi attributi che sono, insegnamento, docente, aula e ora. L'istanza della base di dati comprende i vari dati che vogliamo memorizzare all'interno di questo schema. Ogni elemento di questa istanza della base di dati, cioè ogni insegnamento, rispetta lo schema che abbiamo deciso in prima battuta. Quindi - e anticipo cosa sarà scritto nella slide successiva -, abbiamo una porzione di questa base di dati, che consiste nello schema, che è una porzione che non varia nel tempo, che difficilmente varia nel tempo, e che descrive la struttura della base di dati, quindi questo è un aspetto intenzionale della nostra base di dati, poi nella pratica può succedere che la struttura vari, ma è un evento che deve essere eccezionale nelle base di dati relazionali, e poi l'altro aspetto oltre allo schema è quello dell'istanza, cioè gli effettivi valori che assumono i dati, e invece questa è una componente che può variare più volte nel tempo, anche molto frequentemente, e questo è l'aspetto estensionale dei nostri dati, l’aspetto che non descrive la struttura che vogliamo imporre la base di dati, ma il corpo, il contenuto, dei dati che noi abbiamo.

Abbiamo nelle basi di dati due principali modelli per i dati: i modelli concettuali e i modelli logici. Durante il corso parleremo di entrambi questi modelli e li useremo entrambi.

Iniziamo a dire cosa sono i modelli concettuali. Un modello concettuale non è disponibile nei DBMS; li useremo spesso, ne parleremo abbondantemente; sono modelli che usiamo per la progettazione delle basi di dati e che non usiamo direttamente per la rappresentazione dei dati perché per la rappresentazione di dati useremo invece il modello logico.

Un modello concettuale quindi a cosa serve? Ci permette di rappresentare i dati a un livello di astrazione più alto, indipendentemente dal DBMS che useremo effettivamente, quindi ci serve un modello concettuale per descrivere i concetti del modo reale, ci serve per progettare una base di dati, perché descrivendo i dati a questo livello più astratto diventerà più facile progettare una base di dati grazie al fatto che possiamo trascurare i dettagli implementativi.

Il modello concettuale che useremo nel corso e che vedremo nella prima parte delle lezioni di laboratorio è il modello Entity-Relationship, in italiano Entità-Associazione.

Quando parliamo invece di modelli logici, parliamo dei modelli che vengono usati effettivamente per l'organizzazione dei dati. Questi modelli quindi vengono usati effettivamente dai DBMS esistenti.

Ogni DBMS fa riferimento a un modello logico. In questo corso parleremo di un unico modello logico, che è il modello relazionale, e quindi parleremo solo di basi di dati relazionali.

Questo non è l'unico modello logico usato. In passato, agli albori delle basi di dati, si usavano i modelli reticolari e gerarchici, poi ha avuto un certo successo il modello logico basato su oggetti, attualmente ha molto successo un modello alternativo a quello relazionale che è quello NoSQL, che in realtà è una famiglia di modelli, perché esistono i database NoSQL Document based, quelli colonnari, quelli basati su grafo, e così via. Nel corso, però, parliamo esclusivamente dei modelli relazionali, che sono quelli che hanno la maggiore diffusione commerciale, anche se negli ultimi anni hanno un certo successo i DB NoSQL, che però hanno esigenze e scopi un po’ diversi da quelli dei database relazionali.

In questa slide vediamo uno schema in cui c'è un’architettura molto semplificata di un DBMS.

Alla vostra sinistra vedete l'utente; qua per utente, dato che siamo databasisti, non intendiamo solo l'utente finale: potrebbe essere anche il programmatore che programma l'applicativo che usa il database. A sinistra c’è l’utente, che è l’utente della base di dati, alla vostra destra c'è la base di dati e in mezzo abbiamo lo schema logico e lo schema interno. Notate che non c'è lo schema concettuale, perché lo schema concettuale abbiamo detto che si usa per la progettazione della base di dati, e non si usa quando si usa la base di dati.

Abbiamo lo schema logico, ad esempio espresso con il modello relazionale. Un DBMS non usa al proprio interno lo schema logico relazionale, quindi i dati non vengono memorizzati realmente in relazioni, ma vengono memorizzati in un altro modo, secondo un certo schema interno, che è specifico per il particolare DBMS che usiamo. Ad se usiamo PostgreSQL, lo schema interno sarà un certo schema interno, se usiamo Oracle sarà un altro schema interno, se usiamo SQL server, sarà uno schema ancora diverso.

Ogni DBMS ha uno schema interno diverso, ma ogni database relazionale espone verso l'esterno, cioè verso l'utente, verso il programmatore che usa il DBMS, lo stesso schema logico, usando il modello relazionale, e questo è un vantaggio, un grosso vantaggio che ha decretato il successo negli anni delle base di dati relazionali.

Quindi l'utente che usa le basi di dati relazionali non ha bisogno di conoscere come è fatto all'interno il singolo DBMS, e quando cambia DBMS non deve buttare via questa conoscenza per soffermarsi sullo schema interno del nuovo DBMS, ma può interagire con il database facendo esclusivamente riferimento allo schema logico che questo DBMS espone verso l'esterno, e che è indipendente da come è fatta fisicamente, cioè internamente, la base di dati, che potrebbe non avere relazioni, cioè le tabelle che abbiamo visto prima, ma avrà file, in cui ci sono dei record, con dei puntatori che sono ordinati in un certo modo. Tutti questi dettagli implementativi vengono nascosti all'utente finale della base di dati, grazie al meccanismo realizzato dalla presenza di questi due schemi, lo schema logico e lo schema interno.

Possiamo parlare ora dell'indipendenza dei dati. La suddivisione in schema logico e schema interno fa in modo che il livello logico sia indipendente da quello fisico, quindi da utenti usiamo semplicemente le relazioni, o tabelle, nello stesso modo, qualunque sia la realizzazione di queste tabelle a livello fisico.

In questa slide vediamo l'architettura a tre livelli dei database standard ANSI/SPARC. Vediamo in basso il database, o base di dati, BD, che sarà organizzata secondo un certo schema interno, quindi una rappresentazione secondo certe strutture fisiche di memorizzazione.

Al di sopra abbiamo uno schema logico che descrive la base di dati secondo il modello logico principale del DBMS, che per il nostro corso sarà quello relazionale.

Sopra lo schema logico possiamo avere più schemi esterni, e cioè possiamo dare a utenti e applicazioni diversi, modi diversi di vedere gli stessi dati, cioè lo stesso schema logico, in modo specializzato per un certo applicativo. Quindi, per riprendere l'esempio della biblioteca di qualche slide fa, possiamo pensare di avere due schemi esterni diversi, a seconda che l'utente sia il bibliotecario o il lettore della biblioteca. Per esempio, il lettore vedrà il nostro database, avendo lo schema esterno che riguarda solo ciò che si può cercare: non vedrà la cronologia dei prestiti, l’elenco delle persone che hanno preso in prestito un certo libro. Invece un bibliotecario potrebbe fare riferimento a uno schema esterno che dà una visione più ampia del database, e avrebbe accesso alla cronologia dei prestiti, potrebbe vedere da quanto tempo è in ritardo un prestito, che è un'informazione che potrebbe non essere disponibile per l'utente che è solo un lettore.

Vediamo un altro esempio sulla differenza tra uno schema logico e uno schema esterno. Il titolo della tabella è “vista “, una vista è un termine tecnico per le basi di dati.

Nella metà superiore della slide vediamo due relazioni, cioè due tabelle: una tabella dei corsi, in cui abbiamo il nome del corso, il docente, l'aula, e una tabella delle aule, in cui abbiamo il nome dell'aula, DS1, N3, G, l’edificio, OMI, Pincherle, il piano dell’aula, piano terra, piano primo.

Potremmo, sulla base di questo schema logico della base di dati, esporre uno schema esterno diverso, e costruire una lista che mostra all'utente una visione diversa di queste informazioni. Potremmo esporre una tabella virtuale, che non è una tabella che esiste realmente, fisicamente dentro la base di dati, ma viene costruita in modo dinamico partendo da queste due tabelle. Riunendo le informazioni di queste due tabelle possiamo esporre, per esempio, un'unica tabella che contiene corsi e sedi, quindi abbiamo il corso e qual è l'aula, l'edificio, il piano in cui viene tenuto quel corso. Questa è un'informazione che parte dalle informazioni che sono suddivise, sparse, nelle due tabelle di origine. Sappiamo, dalla prima tabella, che il corso è tenuto nell'aula N3, e poi leggendo nella seconda tabella sappiamo che l’aula N3 è nell'edificio OMI, al piano terra.

Il fatto che suddividiamo in due livelli le informazioni dà luogo a due forme di indipendenza dei dati: l'indipendenza fisica e l'indipendenza logica.

Indipendenza fisica significa che possiamo usare una relazione in modo indipendente dalla sua realizzazione fisica. Come ho detto prima ogni DBMS può organizzare i file in modo diverso e a noi utenti della base di dati non importa conoscere la specifica organizzazione fisica: grazie all’ indipendenza fisica usiamo questi dati senza investigare l'effettiva organizzazione fisica. È il DBMS che si occupa di tradurre le nostre richieste di accesso ai dati nelle opportune richieste di accesso alle strutture fisiche.

L'indipendenza logica riguarda l’indipendenza del livello logico, cioè possiamo avere una certa indipendenza anche rispetto allo schema logico della base di dati. Grazie al meccanismo di viste possiamo esporre verso l'esterno una sorta di tabella virtuale, cioè delle viste, che fanno in modo che l'utente esterno possa interagire con le viste invece che con le relazioni, e in modo indipendente da quali sono le effettive tabelle o relazioni che soggiacciono a queste viste.

Per andare più sul pratico possiamo dire come si fa a interagire con una base di dati. Quali linguaggi possiamo usare per interagire con le basi di dati?

Esistono vari modi per interagire con le basi di dati. Sulla slide, partendo dal basso, possiamo vedere delle interfacce grafiche relative all'esempio fatto prima. Ho già nominato di Microsoft Access, e noi non useremo questo DBMS perché siamo informatici e ci piace fare le cose in modo serio.

Poi possiamo avere i comandi SQL a vari livelli. Sulla slide lo vedete usato nei primi tre punti. Di SQL parleremo in modo molto diffuso. Vi anticipo che è una sorta di lingua franca delle basi di dati relazionali, cioè ogni base di dati relazionale offre supporto a SQL.

Possiamo usare SQL, come faremo in laboratorio, come linguaggio testuale/interattivo, cioè avremo una certa interfaccia in cui possiamo scrivere direttamente i comandi SQL e poi eseguire questi comandi.

Oppure possiamo “immergere” i comandi SQL in un linguaggio di programmazione ospite. Scriviamo un programma Java - più in là ci sarà un esempio - al cui interno inseriamo comandi SQL.

Oppure la terza possibilità è di usare una specie di sovrainsieme di SQL, in cui SQL diventa un linguaggio di programmazione in cui ci saranno delle variabili. Questi linguaggi sono specifici dei DBMS, quindi PostgreSQL ha un proprio linguaggio di programmazione, Oracle un altro linguaggio di programmazione, in cui all'interno è molto facile eseguire le query SQL.

Vediamo un primo esempio in cui abbiamo queste due tabelle, con i corsi e le aule, e vogliamo ricavare quali i corsi sono tenuti in aule al piano terra. Questo è un esempio di un’interrogazione in cui stiamo porgendo una domanda alla base di dati e ci aspettiamo che la base di dati ci dia una risposta. Questo è un esempio giocattolo molto banale e uno potrebbe dire: “Vabbè, ma basta leggere la tabella: è ovvio quali i corsi sono tenuti nelle aule al piano terra”. Però dobbiamo immaginare di avere non tabelle con tre o quattro righe, ma tabelle con milioni o miliardi di righe, e allora sarebbe meno ovvio rispondere a queste domande. Inoltre un'informazione potrebbe non essere suddivisa in due tabelle diverse, ma essere suddivisa in cinque tabelle diverse, 10 tabelle diverse.

In questa slide vediamo un esempio molto semplice di una query SQL; approfondiremo SQL in seguito.

Abbiamo una query scritta su tre righe, in cui, nella seconda riga usiamo le tabelle aule e corsi, nella terza riga (where), diciamo quale condizione deve essere soddisfatta: vogliamo che il campo nome sia essere uguale al campo aula. Questa è un'operazione di join, in seguito discuteremo in modo ampio dei join sia nelle lezioni di teoria che nelle elezioni di laboratorio. Nella query vogliamo che il piano sia il piano terra e nella prima riga diciamo che vogliamo avere, nel risultato, il corso, il nome dell'aula e il piano, che ovviamente sarà terra.

Sulla slide vedete la risposta: ci sono due corsi, sistemi e reti, che vengono tenuti nell'aula N3, che è al piano terra.

In questa slide, che non mi aspetto che impariate a memoria, vediamo un esempio della stessa query di prima, immersa in un linguaggio ospite, che in questo caso è Java.

Come al solito Java è molto verboso, quindi abbiamo molte istruzioni per svolgere un compito molto semplice.

Vi spiego brevemente il senso.

Apriamo una connessione verso il database PostgreSQL, riuscite a vedere la scritta postgresql da qualche parte nella stringa di connessione

Attraverso questa connessione chiediamo di eseguire un comando, che corrisponde alla query, la vedete in grassetto.

Poi prendiamo i risultati e facciamo un ciclo while in cui mostriamo all'utente con una print ogni record del risultato, dopodiché chiudiamo la connessione.

In quest'altra slide, invece, vediamo un esempio del linguaggio di programmazione specifico di Oracle, che si chiama PL/SQL. In questo caso abbiamo una query che cerca lo stipendio di un particolare impiegato, e se l’impiegato ha uno stipendio maggiore di 30, gli diamo un aumento del 10%, altrimenti gli diamo un aumento del 15%. Come vedete è un po’ particolare, perché abbiamo intervallato comandi che sono SQL, quelli in maiuscolo, Select, From, Where, con comandi che sono tipici di un linguaggio di programmazione, if, then, else.

In questa slide vediamo un esempio di una schermata di Access in cui costruiamo una query in modo grafico, dicendo quali sono i campi o attributi che ci interessano, in quali tabelle vengono presi, e quali sono gli eventuali criteri che devono essere validi perché una riga faccia parte del risultato dell’interrogazione.

Di solito nelle basi di dati si fa una distinzione tra due tipi di linguaggi, cioè i linguaggi di manipolazione dei dati, e il linguaggio di definizione dei dati. Il DML, Data Manipulation Language, serve per interrogare la base di dati, e anche per aggiornare i dati, quindi fare modifiche, inserimenti dei dati. Il DDL, Data Definition Language, invece serve per modificare, non l'istanza della base di dati, ma lo schema della base di dati, per esempio costruire una nuova tabella, oppure modificare gli attributi di una tabella.

Vediamo un esempio di un comando DDL, in cui, come vedremo meglio in laboratorio, creiamo una tabella orario che comprende quattro attributi: insegnamento, docente, aula, ora, che sono delle stringhe, rispettivamente di 20, 20, 24, e 5 caratteri.

Quali sono le principali figure che interagiscono con un DBMS? Abbiamo i realizzatori dei DBMS veri e propri, pochi al mondo, poi ci sono i progettisti, invece, delle singole basi di dati, non del sistema di gestione della basi dati, cioè non i progettisti di Oracle o PostgreSQL, ma dei sistemi di gestione degli orari delle lezioni e delle segreterie degli studenti, e gli amministratori delle basi di dati. Abbiamo anche i progettisti delle applicazioni che interagiscono con queste basi di dati, e abbiamo questa categoria generale di utenti, che possono essere utenti che usano abitualmente le basi di dati, oppure utenti casuali.

Una figura importante è quella del database administrator, abbreviato DBA, che è una persona, o anche un gruppo di persone, che si occupa di controllare e gestire il database, garantendo che le autorizzazioni che vengono date agli altri utenti siano corrette, che il DBMS continui a funzionare in modo affidabile ed efficiente.

Riassumendo, quali sono i pro e i contro dei DBMS?

I pro sono: i database individuano i dati come una risorsa pregiata, che ha una certa importanza, che deve essere quindi anche slegata dalla vita di un singolo applicativo che usa la base di dati. Posso cambiare applicativo, ma, comunque, non cambio la base di dati perché è una risorsa centrale. Può fornire dei servizi perché una risorsa centralizzata permette di ridurre ridondanze e inconsistente e inoltre garantisce la gestione della concorrenza, della privatezza, dell'affidabilità dei dati, garantisce che ci sia dipendenza tra i dati e le applicazioni.

Gli svantaggi sono: un DBMS può avere un costo non indifferente; anche se si usa una soluzione open source, comunque c'è un costo che non è quello dell'acquisto del DBMS, ma è comunque il costo dell'adozione del DBMS, perché un DBMS è un software che non è banale, non è semplicissimo da usare, e quindi richiede una certa formazione e preparazione, cose di cui parleremo durante il nostro corso.

**4**

Un database fa riferimento a un modello dei dati, che possiamo pensare come una sorta di costruttore di tipo in un linguaggio di programmazione, cioè è un meccanismo che dice come sono strutturati i dati in un database.

Il modello relazionale ha come modello dei dati quello della relazione, che adesso inizio a introdurre in modo informale, e nel prossimo blocco di slide, invece, definiamo in modo decisamente più formale. Riprendiamo l'esempio dell'orario delle lezioni. Abbiamo questo corso di studi, in cui ci sono queste lezioni, per cui abbiamo un insegnamento, un docente, un'aula, e un orario. Come struttureremmo queste informazioni in un database relazionale? Le struttureremmo, probabilmente, nello stesso modo, o in un modo simile, quindi abbiamo un esempio di una relazione, che a prima vista sembra una cosa molto intuitiva e molto semplice, poi, quando lo approfondiremo meglio, vedremo che in effetti è semplice ma presente comunque alcune difficoltà, non è proprio completamente intuitivo.

A prima vista abbiamo una tabella che ha il titolo orario, in cui abbiamo un'intestazione che è insegnamento, docente, aula, ora, sulle colonne, e abbiamo le varie righe della tabella in cui diciamo qual è l'insegnamento, per esempio, analisi matematica 1, quale docente, Luigi Neri, qual è l'aula, N1, e qual è l'ora di inizio della lezione. Abbiamo una riga per ogni insegnamento e riportiamo le stesse informazioni per ognuno degli insegnamenti. Quindi una prima distinzione che possiamo fare è la distinzione tra schema e istanza, lo schema di questa nostra semplicissima base di dati dice che abbiamo una relazione, che si chiama orario, e per questa relazione orario memorizziamo questi attributi che sono, insegnamento, docente, aula e ora. L'istanza della base di dati comprende i vari dati che vogliamo memorizzare all'interno di questo schema. Ogni elemento di questa istanza della base di dati, cioè ogni insegnamento, rispetta lo schema che abbiamo deciso in prima battuta. Quindi - e anticipo cosa sarà scritto nella slide successiva -, abbiamo una porzione di questa base di dati, che consiste nello schema, che è una porzione che non varia nel tempo, che difficilmente varia nel tempo, e che descrive la struttura della base di dati, quindi questo è un aspetto intenzionale della nostra base di dati, poi nella pratica può succedere che la struttura vari, ma è un evento che deve essere eccezionale nelle base di dati relazionali, e poi l'altro aspetto oltre allo schema è quello dell'istanza, cioè gli effettivi valori che assumono i dati, e invece questa è una componente che può variare più volte nel tempo, anche molto frequentemente, e questo è l'aspetto estensionale dei nostri dati, l’aspetto che non descrive la struttura che vogliamo imporre la base di dati, ma il corpo, il contenuto, dei dati che noi abbiamo.

Abbiamo nelle basi di dati due principali modelli per i dati: i modelli concettuali e i modelli logici. Durante il corso parleremo di entrambi questi modelli e li useremo entrambi.

Iniziamo a dire cosa sono i modelli concettuali. Un modello concettuale non è disponibile nei DBMS; li useremo spesso, ne parleremo abbondantemente; sono modelli che usiamo per la progettazione delle basi di dati e che non usiamo direttamente per la rappresentazione dei dati perché per la rappresentazione di dati useremo invece il modello logico.

Un modello concettuale quindi a cosa serve? Ci permette di rappresentare i dati a un livello di astrazione più alto, indipendentemente dal DBMS che useremo effettivamente, quindi ci serve un modello concettuale per descrivere i concetti del modo reale, ci serve per progettare una base di dati, perché descrivendo i dati a questo livello più astratto diventerà più facile progettare una base di dati grazie al fatto che possiamo trascurare i dettagli implementativi.

Il modello concettuale che useremo nel corso e che vedremo nella prima parte delle lezioni di laboratorio è il modello Entity-Relationship, in italiano Entità-Associazione.

Quando parliamo invece di modelli logici, parliamo dei modelli che vengono usati effettivamente per l'organizzazione dei dati. Questi modelli quindi vengono usati effettivamente dai DBMS esistenti.

Ogni DBMS fa riferimento a un modello logico. In questo corso parleremo di un unico modello logico, che è il modello relazionale, e quindi parleremo solo di basi di dati relazionali.

Questo non è l'unico modello logico usato. In passato, agli albori delle basi di dati, si usavano i modelli reticolari e gerarchici, poi ha avuto un certo successo il modello logico basato su oggetti, attualmente ha molto successo un modello alternativo a quello relazionale che è quello NoSQL, che in realtà è una famiglia di modelli, perché esistono i database NoSQL Document based, quelli colonnari, quelli basati su grafo, e così via. Nel corso, però, parliamo esclusivamente dei modelli relazionali, che sono quelli che hanno la maggiore diffusione commerciale, anche se negli ultimi anni hanno un certo successo i DB NoSQL, che però hanno esigenze e scopi un po’ diversi da quelli dei database relazionali.

In questa slide vediamo uno schema in cui c'è un’architettura molto semplificata di un DBMS.

Alla vostra sinistra vedete l'utente; qua per utente, dato che siamo databasisti, non intendiamo solo l'utente finale: potrebbe essere anche il programmatore che programma l'applicativo che usa il database. A sinistra c’è l’utente, che è l’utente della base di dati, alla vostra destra c'è la base di dati e in mezzo abbiamo lo schema logico e lo schema interno. Notate che non c'è lo schema concettuale, perché lo schema concettuale abbiamo detto che si usa per la progettazione della base di dati, e non si usa quando si usa la base di dati.

Abbiamo lo schema logico, ad esempio espresso con il modello relazionale. Un DBMS non usa al proprio interno lo schema logico relazionale, quindi i dati non vengono memorizzati realmente in relazioni, ma vengono memorizzati in un altro modo, secondo un certo schema interno, che è specifico per il particolare DBMS che usiamo. Ad se usiamo PostgreSQL, lo schema interno sarà un certo schema interno, se usiamo Oracle sarà un altro schema interno, se usiamo SQL server, sarà uno schema ancora diverso.

Ogni DBMS ha uno schema interno diverso, ma ogni database relazionale espone verso l'esterno, cioè verso l'utente, verso il programmatore che usa il DBMS, lo stesso schema logico, usando il modello relazionale, e questo è un vantaggio, un grosso vantaggio che ha decretato il successo negli anni delle base di dati relazionali.

Quindi l'utente che usa le basi di dati relazionali non ha bisogno di conoscere come è fatto all'interno il singolo DBMS, e quando cambia DBMS non deve buttare via questa conoscenza per soffermarsi sullo schema interno del nuovo DBMS, ma può interagire con il database facendo esclusivamente riferimento allo schema logico che questo DBMS espone verso l'esterno, e che è indipendente da come è fatta fisicamente, cioè internamente, la base di dati, che potrebbe non avere relazioni, cioè le tabelle che abbiamo visto prima, ma avrà file, in cui ci sono dei record, con dei puntatori che sono ordinati in un certo modo. Tutti questi dettagli implementativi vengono nascosti all'utente finale della base di dati, grazie al meccanismo realizzato dalla presenza di questi due schemi, lo schema logico e lo schema interno.

Possiamo parlare ora dell'indipendenza dei dati. La suddivisione in schema logico e schema interno fa in modo che il livello logico sia indipendente da quello fisico, quindi da utenti usiamo semplicemente le relazioni, o tabelle, nello stesso modo, qualunque sia la realizzazione di queste tabelle a livello fisico.

In questa slide vediamo l'architettura a tre livelli dei database standard ANSI/SPARC. Vediamo in basso il database, o base di dati, BD, che sarà organizzata secondo un certo schema interno, quindi una rappresentazione secondo certe strutture fisiche di memorizzazione.

Al di sopra abbiamo uno schema logico che descrive la base di dati secondo il modello logico principale del DBMS, che per il nostro corso sarà quello relazionale.

Sopra lo schema logico possiamo avere più schemi esterni, e cioè possiamo dare a utenti e applicazioni diversi, modi diversi di vedere gli stessi dati, cioè lo stesso schema logico, in modo specializzato per un certo applicativo. Quindi, per riprendere l'esempio della biblioteca di qualche slide fa, possiamo pensare di avere due schemi esterni diversi, a seconda che l'utente sia il bibliotecario o il lettore della biblioteca. Per esempio, il lettore vedrà il nostro database, avendo lo schema esterno che riguarda solo ciò che si può cercare: non vedrà la cronologia dei prestiti, l’elenco delle persone che hanno preso in prestito un certo libro. Invece un bibliotecario potrebbe fare riferimento a uno schema esterno che dà una visione più ampia del database, e avrebbe accesso alla cronologia dei prestiti, potrebbe vedere da quanto tempo è in ritardo un prestito, che è un'informazione che potrebbe non essere disponibile per l'utente che è solo un lettore.

Vediamo un altro esempio sulla differenza tra uno schema logico e uno schema esterno. Il titolo della tabella è “vista “, una vista è un termine tecnico per le basi di dati.

Nella metà superiore della slide vediamo due relazioni, cioè due tabelle: una tabella dei corsi, in cui abbiamo il nome del corso, il docente, l'aula, e una tabella delle aule, in cui abbiamo il nome dell'aula, DS1, N3, G, l’edificio, OMI, Pincherle, il piano dell’aula, piano terra, piano primo.

Potremmo, sulla base di questo schema logico della base di dati, esporre uno schema esterno diverso, e costruire una lista che mostra all'utente una visione diversa di queste informazioni. Potremmo esporre una tabella virtuale, che non è una tabella che esiste realmente, fisicamente dentro la base di dati, ma viene costruita in modo dinamico partendo da queste due tabelle. Riunendo le informazioni di queste due tabelle possiamo esporre, per esempio, un'unica tabella che contiene corsi e sedi, quindi abbiamo il corso e qual è l'aula, l'edificio, il piano in cui viene tenuto quel corso. Questa è un'informazione che parte dalle informazioni che sono suddivise, sparse, nelle due tabelle di origine. Sappiamo, dalla prima tabella, che il corso è tenuto nell'aula N3, e poi leggendo nella seconda tabella sappiamo che l’aula N3 è nell'edificio OMI, al piano terra.

Il fatto che suddividiamo in due livelli le informazioni dà luogo a due forme di indipendenza dei dati: l'indipendenza fisica e l'indipendenza logica.

Indipendenza fisica significa che possiamo usare una relazione in modo indipendente dalla sua realizzazione fisica. Come ho detto prima ogni DBMS può organizzare i file in modo diverso e a noi utenti della base di dati non importa conoscere la specifica organizzazione fisica: grazie all’ indipendenza fisica usiamo questi dati senza investigare l'effettiva organizzazione fisica. È il DBMS che si occupa di tradurre le nostre richieste di accesso ai dati nelle opportune richieste di accesso alle strutture fisiche.

L'indipendenza logica riguarda l’indipendenza del livello logico, cioè possiamo avere una certa indipendenza anche rispetto allo schema logico della base di dati. Grazie al meccanismo di viste possiamo esporre verso l'esterno una sorta di tabella virtuale, cioè delle viste, che fanno in modo che l'utente esterno possa interagire con le viste invece che con le relazioni, e in modo indipendente da quali sono le effettive tabelle o relazioni che soggiacciono a queste viste.

Per andare più sul pratico possiamo dire come si fa a interagire con una base di dati. Quali linguaggi possiamo usare per interagire con le basi di dati?

Esistono vari modi per interagire con le basi di dati. Sulla slide, partendo dal basso, possiamo vedere delle interfacce grafiche relative all'esempio fatto prima. Ho già nominato di Microsoft Access, e noi non useremo questo DBMS perché siamo informatici e ci piace fare le cose in modo serio.

Poi possiamo avere i comandi SQL a vari livelli. Sulla slide lo vedete usato nei primi tre punti. Di SQL parleremo in modo molto diffuso. Vi anticipo che è una sorta di lingua franca delle basi di dati relazionali, cioè ogni base di dati relazionale offre supporto a SQL.

Possiamo usare SQL, come faremo in laboratorio, come linguaggio testuale/interattivo, cioè avremo una certa interfaccia in cui possiamo scrivere direttamente i comandi SQL e poi eseguire questi comandi.

Oppure possiamo “immergere” i comandi SQL in un linguaggio di programmazione ospite. Scriviamo un programma Java - più in là ci sarà un esempio - al cui interno inseriamo comandi SQL.

Oppure la terza possibilità è di usare una specie di sovrainsieme di SQL, in cui SQL diventa un linguaggio di programmazione in cui ci saranno delle variabili. Questi linguaggi sono specifici dei DBMS, quindi PostgreSQL ha un proprio linguaggio di programmazione, Oracle un altro linguaggio di programmazione, in cui all'interno è molto facile eseguire le query SQL.

Vediamo un primo esempio in cui abbiamo queste due tabelle, con i corsi e le aule, e vogliamo ricavare quali i corsi sono tenuti in aule al piano terra. Questo è un esempio di un’interrogazione in cui stiamo porgendo una domanda alla base di dati e ci aspettiamo che la base di dati ci dia una risposta. Questo è un esempio giocattolo molto banale e uno potrebbe dire: “Vabbè, ma basta leggere la tabella: è ovvio quali i corsi sono tenuti nelle aule al piano terra”. Però dobbiamo immaginare di avere non tabelle con tre o quattro righe, ma tabelle con milioni o miliardi di righe, e allora sarebbe meno ovvio rispondere a queste domande. Inoltre un'informazione potrebbe non essere suddivisa in due tabelle diverse, ma essere suddivisa in cinque tabelle diverse, 10 tabelle diverse.

In questa slide vediamo un esempio molto semplice di una query SQL; approfondiremo SQL in seguito.

Abbiamo una query scritta su tre righe, in cui, nella seconda riga usiamo le tabelle aule e corsi, nella terza riga (where), diciamo quale condizione deve essere soddisfatta: vogliamo che il campo nome sia essere uguale al campo aula. Questa è un'operazione di join, in seguito discuteremo in modo ampio dei join sia nelle lezioni di teoria che nelle elezioni di laboratorio. Nella query vogliamo che il piano sia il piano terra e nella prima riga diciamo che vogliamo avere, nel risultato, il corso, il nome dell'aula e il piano, che ovviamente sarà terra.

Sulla slide vedete la risposta: ci sono due corsi, sistemi e reti, che vengono tenuti nell'aula N3, che è al piano terra.

In questa slide, che non mi aspetto che impariate a memoria, vediamo un esempio della stessa query di prima, immersa in un linguaggio ospite, che in questo caso è Java.

Come al solito Java è molto verboso, quindi abbiamo molte istruzioni per svolgere un compito molto semplice.

Vi spiego brevemente il senso.

Apriamo una connessione verso il database PostgreSQL, riuscite a vedere la scritta postgresql da qualche parte nella stringa di connessione

Attraverso questa connessione chiediamo di eseguire un comando, che corrisponde alla query, la vedete in grassetto.

Poi prendiamo i risultati e facciamo un ciclo while in cui mostriamo all'utente con una print ogni record del risultato, dopodiché chiudiamo la connessione.

In quest'altra slide, invece, vediamo un esempio del linguaggio di programmazione specifico di Oracle, che si chiama PL/SQL. In questo caso abbiamo una query che cerca lo stipendio di un particolare impiegato, e se l’impiegato ha uno stipendio maggiore di 30, gli diamo un aumento del 10%, altrimenti gli diamo un aumento del 15%. Come vedete è un po’ particolare, perché abbiamo intervallato comandi che sono SQL, quelli in maiuscolo, Select, From, Where, con comandi che sono tipici di un linguaggio di programmazione, if, then, else.

In questa slide vediamo un esempio di una schermata di Access in cui costruiamo una query in modo grafico, dicendo quali sono i campi o attributi che ci interessano, in quali tabelle vengono presi, e quali sono gli eventuali criteri che devono essere validi perché una riga faccia parte del risultato dell’interrogazione.

Di solito nelle basi di dati si fa una distinzione tra due tipi di linguaggi, cioè i linguaggi di manipolazione dei dati, e il linguaggio di definizione dei dati. Il DML, Data Manipulation Language, serve per interrogare la base di dati, e anche per aggiornare i dati, quindi fare modifiche, inserimenti dei dati. Il DDL, Data Definition Language, invece serve per modificare, non l'istanza della base di dati, ma lo schema della base di dati, per esempio costruire una nuova tabella, oppure modificare gli attributi di una tabella.

Vediamo un esempio di un comando DDL, in cui, come vedremo meglio in laboratorio, creiamo una tabella orario che comprende quattro attributi: insegnamento, docente, aula, ora, che sono delle stringhe, rispettivamente di 20, 20, 24, e 5 caratteri.

Quali sono le principali figure che interagiscono con un DBMS? Abbiamo i realizzatori dei DBMS veri e propri, pochi al mondo, poi ci sono i progettisti, invece, delle singole basi di dati, non del sistema di gestione della basi dati, cioè non i progettisti di Oracle o PostgreSQL, ma dei sistemi di gestione degli orari delle lezioni e delle segreterie degli studenti, e gli amministratori delle basi di dati. Abbiamo anche i progettisti delle applicazioni che interagiscono con queste basi di dati, e abbiamo questa categoria generale di utenti, che possono essere utenti che usano abitualmente le basi di dati, oppure utenti casuali.

Una figura importante è quella del database administrator, abbreviato DBA, che è una persona, o anche un gruppo di persone, che si occupa di controllare e gestire il database, garantendo che le autorizzazioni che vengono date agli altri utenti siano corrette, che il DBMS continui a funzionare in modo affidabile ed efficiente.

Riassumendo, quali sono i pro e i contro dei DBMS?

I pro sono: i database individuano i dati come una risorsa pregiata, che ha una certa importanza, che deve essere quindi anche slegata dalla vita di un singolo applicativo che usa la base di dati. Posso cambiare applicativo, ma, comunque, non cambio la base di dati perché è una risorsa centrale. Può fornire dei servizi perché una risorsa centralizzata permette di ridurre ridondanze e inconsistente e inoltre garantisce la gestione della concorrenza, della privatezza, dell'affidabilità dei dati, garantisce che ci sia dipendenza tra i dati e le applicazioni.

Gli svantaggi sono: un DBMS può avere un costo non indifferente; anche se si usa una soluzione open source, comunque c'è un costo che non è quello dell'acquisto del DBMS, ma è comunque il costo dell'adozione del DBMS, perché un DBMS è un software che non è banale, non è semplicissimo da usare, e quindi richiede una certa formazione e preparazione, cose di cui parleremo durante il nostro corso.

**5**

Vediamo un altro esempio. In questo esempio non abbiamo due relazioni, bensì possiamo pensare che siano una specie di foto di due ricevute per due pasti in un ristorante di Roma, il ristorante Da Filippo in via Roma 2 a Roma (anche se a Roma non c'è nessuna via Roma).

Quindi abbiamo due ricevute in cui ci sono coperti, antipasti, primi, bistecche, orate e caffè, con un totale. In queste ricevute possiamo distinguere delle parti che hanno una variabilità diversa nel mondo, per esempio quelli in rosso sono esempi di dati che variano tra ricevute diverse, fatte in momenti diversi e per clienti diversi: mentre due ricevute dello stesso ristorante avranno un'intestazione uguale, Da Filippo via Roma 2 a Roma, "ricevuta fiscale del..", ci sono altri dati che rappresentano l'istanza dei dati. Quindi cambia numero della ricevuta, cambia la data della ricevuta, cambia il numero di coperti, il totale, il totale parziale e il totale complessivo.

Notiamo anche che i dati possano avere un livello di nidificazione diverso, nel senso che quelli in rosso sono dati in cui abbiamo un esemplare per l'intera ricevuta, quindi il numero della ricevuta è unico per tutta la ricevuta, la data della ricevuta è unica per tutta la ricevuta e anche il totale complessivo è unico, mentre ogni ricevuta può avere più voci, che sono le voci in blu, quindi non è detto che tutti abbiano lo stesso numero di coperti o lo stesso numero di piatti. Nella ricevuta destra abbiamo il caffè, che a sinistra non abbiamo. Questi sono esempi, cioè, di annidamento diverso dei dati.

Questa considerazione ci permette di fare alcune riflessioni riguardo il modello relazionale che non sono così scontate. Per esempio, supponiamo di voler rappresentare queste ricevute nel modello relazionale e nel farlo potremmo essere tentati di rappresentarle come vedete sulla slide, quindi con una relazione che si chiama ricevute e che ha quattro attributi: numero, data, piatti e totale. A sua volta l'attributo piatti contiene una sottorelazione che corrisponde ai singoli piatti/voci di ogni ricevuta: coperti, antipasti, primi, bistecche, caffè e così via.

Questo fatto ci permette di introdurre una considerazione che forse non è così ovvia nel modello relazionale, cioè il modello relazionale è molto semplice, però comporta delle conseguenze che non sono così ovvie, come il fatto che non è possibile rappresentare informazioni in questo modo, cioè questa non è una relazione ammessa nel modello relazionale perché le relazioni nel modello relazionale sono relazioni "piatte", nel senso che ogni attributo ha valori semplici e quindi non sono ammesse sottorelazioni.

Questa restrizione si chiama, per motivi storici, prima forma normale. Ha questo nome particolare che potrebbe causare confusione, perché poi parleremo di terza forma normale, che è molto diversa da questa, comunque il concetto è che gli attributi delle relazioni possono contenere esclusivamente valori semplici, quindi non possono avere questa struttura nidificata.

Quindi come possiamo rappresentare in un database le ricevute di questo ristorante?

Per esempio, usando più tabelle, più relazioni, come nell'esempio della slide.

Quindi l'idea di queste due relazioni è che abbiamo una relazione, che contiene i dati della ricevuta, esclusivamente propri della ricevuta, come il numero della ricevuta, la data e il totale complessivo di ogni ricevuta.

Poi abbiamo una seconda relazione in cui dettagliamo le voci di ogni ricevuta e quindi dettagliamo quanti sono i coperti, qual è l'importo dei coperti, quali sono gli antipasti, l'importo degli antipasti e così via.

Come facciamo a tener traccia di qual è la ricevuta a cui facciamo riferimento?

Tramite il numero della ricevuta: se guardate la relazione dettaglio, il primo tributo della relazione dettaglio si chiama ricevuta e contiene il numero della ricevuta.

Questo numero è un numero univoco nella relazione ricevute e quindi ci permette, dalla relazione dettaglio, di risalire in modo univoco a quale sia la ricevuta a cui questo dettaglio fa riferimento.

Dividere le informazioni in due relazioni diverse ci ha permesso di superare questa limitazione, cioè

che il modello relazionale non può avere strutture nidificate. Quindi questa non è una vera limitazione nella capacità di rappresentazione delle informazioni, perché possiamo comunque rappresentare le stesse informazioni suddividendole in tabelle diverse.

A questo punto facciamo una riflessione: “Ma stiamo veramente rappresentando le stesse informazioni che rappresentavamo attraverso le ricevute?”.

La risposta dipende da ciò che ci serve rappresentare. Quindi ci possiamo, per esempio, chiedere: “Ma una ricevuta contiene tutti i valori distinti tra di loro? L’ordine delle righe è importante in una ricevuta? È uguale se metto prima il coperto e poi il primo e poi secondo, e se invece metto prima il secondo e poi l'antipasto e poi il primo?”.

Se decidiamo che queste cose non sono importanti, la rappresentazione che abbiamo visto nella slide precedente è una rappresentazione che può andare bene, può essere sufficiente. Ricordiamoci che in una relazione l'ordine delle tuple non è importante, quindi noi non possiamo risalire attraverso una relazione a quale era l'ordine delle tuple da cui abbiamo estratto questi dati, perché questa informazione non è memorizzata, non è rappresentata nella relazione, l'ordine non è importante.

Seconda cosa, non solo l'ordine non è importante ma, inoltre, non posso avere ripetizioni nelle tuple, non posso avere nella stessa relazione due tuple uguali e quindi non posso avere per esempio due righe con i coperti con la stessa quantità, stesso totale e coperti nella stessa ricevuta. In ricevute diverse è possibile perché non sarebbe una ripetizione, dato che cambierebbe il numero della ricevuta, ma nella stessa ricevuta non posso avere due volte la stessa voce.

Allora se io ho deciso che, invece, queste cose mi interessa rappresentarle, cioè che voglio rappresentare e non voglio perdere l'ordine delle righe, perché l’ordine per me è rilevante, oppure se posso avere ripetizioni, allora devo rappresentare le informazioni in modo diverso.

Ad esempio, possiamo rappresentare le informazioni come nella slide in questa nuova versione delle relazioni.

Qual è la differenza rispetto a prima? Abbiamo aggiunto l'attributo riga alla relazione dettaglio, quindi la riga ci permette di dire che per esempio la ricevuta 1235 aveva come prima riga i coperti, come seconda riga gli antipasti, terza riga i primi, quarta riga le bistecche.

Quindi questo nuovo tributo ci permette di fare due cose: primo rappresentare questa informazione aggiuntiva del fatto che vogliamo un certo ordine nel presentare ogni voce della ricevuta e secondo possiamo rappresentare anche voci uguali nella stessa ricevuta, perché queste due voci uguali saranno messe in due righe diverse quindi avranno un numero di riga diverso.

Un altro aspetto importante del modello relazionale è che impone una struttura rigida ai dati.

Cosa vuol dire rigida? Vuol dire che le informazioni sono rappresentate per mezzo di tuple e ogni tupla deve essere omogenea all'interno della stessa relazione.

Cosa vuol dire omogenea? Vuol dire che ogni tupla deve conformarsi allo schema della relazione, noi non possiamo avere, ad un certo punto di una relazione, una tupla che rappresenta attributi diversi che non sono previsti dallo schema della relazione, degli attributi in più o attributi in meno.

Il problema è che i dati che noi abbiamo a disposizione potrebbero non combaciare perfettamente con il formato che abbiamo previsto. Ad esempio, sulla slide vedete che c'è una relazione molto semplice che contiene dei nomi di capi di stato di alcuni paesi che hanno vinto la seconda guerra mondiale.

Lo schema della relazione contiene tre attributi: nome, secondo nome e cognome. Abbiamo Franklin Delano Roosevelt, ma gli altri tre capi di stato, Winston Churchill, Charles De Gaulle e Josip Stalin, non hanno un secondo nome. Quindi noi abbiamo deciso, all'inizio, un certo schema di relazione della base dati e poi dobbiamo rappresentare queste quattro tuple all'interno di questa relazione, ma per alcune tuple l'informazione manca completamente.

Quindi si può dire che stiamo contravvenendo a questo principio di omogeneità. Potremmo anche decidere che non vogliamo rappresentare il secondo nome e quindi Franklin Delano Roosevelt diventerà Franklin Roosevelt e quindi “tagliamo” questa relazione, buttando via il secondo attributo che è quello che rappresenta il secondo nome, ma in questo modo stiamo buttando via un pezzo di informazione che magari desideriamo rappresentare.

Per riassumere, quando progettiamo una base di dati dobbiamo decidere qual è lo schema delle varie relazioni, quali sono le relazioni e per ogni relazione qual è suo schema. Questo schema deve essere deciso a priori e deve essere mantenuto per tutte le tuple della relazione, ma non tutte le tuple della relazione potrebbero conformarsi a questo schema che abbiamo deciso. Quindi in questo caso abbiamo delle informazioni incomplete.

Nell'esempio che abbiamo fatto ci mancano proprio le informazioni perché alcune di quelle persone non hanno un secondo nome.

Come possiamo risolvere? Una prima soluzione che ci potrebbe venire in mente è una soluzione tampone, che risolve le cose sul momento ma solo apparentemente perché poi non funziona veramente bene.

Quello che non si deve fare è adottare questa soluzione in cui io uso dei valori particolari, dei valori “strani”, che io suppongo che non siano utilizzati; li uso come segnaposto per dire, "attenzione qua io non ho l'informazione".

Quindi questo valore potrebbe essere uno 0, posso mettere una stringa vuota o stringa nulla, potrei

mettere 99, una sequenza di 99, per esempio un numero di telefono che io non conosco. Il problema è che io penso, quando inizio a riempire/popolare questa relazione, che quel valore non verrà usato. Poi in futuro tra cinque anni, dieci anni, quel valore, quel numero di telefono 999, magari sarà un numero di telefono ammesso e come faccio poi a comunicare alle altre persone che useranno questo database che quel valore che ho messo, tanto per tamponare, per dire “attenzione, non conosco il

valore vero”, oppure “non esiste il valore vero”, cioè questo 99, questa stringa vuota in realtà significa che quel valore non è ammesso e non è un vero 99.

Quindi come faccio a rappresentare questo significato? Nel modello relazionale si adotta una tecnica che si è rivelata abbastanza efficace, anche se rudimentale, ed è una tecnica, che adesso vi spiegherò, che però non è così banale nelle sue conseguenze, nel modo in cui viene usata per esempio in SQL.

Dal punto di vista teorico questo argomento non lo approfondiremo, esso causa in realtà alcune inconsistenze, alcuni problemi noti del modello relazionale. Ora questa tecnica è quella di usare un valore extra, un valore particolare che è garantito essere al di fuori del dominio degli attributi ed è il valore nullo, che si scrive in inglese NULL.

Quindi un valore nullo è un valore che è garantito essere al di fuori del dominio dell'attributo. Se noi

abbiamo un attributo che è un attributo di tipo stringa, sappiamo che non esiste la stringa che corrisponde al valore nullo: il valore nullo è al di fuori del dominio delle stringhe. Cioè può esistere la stringa che contiene le quattro lettere "NULL", ma il valore NULL è al di fuori, non corrisponde a quei 4 caratteri, è un valore extra.

Quindi per ogni tupla t[A], t[A] può assumere un valore nel dominio, quindi t[A] deve appartenere al dominio di A o può anche assumere il valore nullo, che nelle slide indichiamo con questo NULL.

Si può anche decidere, quando decidiamo lo schema di una relazione, di imporre restrizioni sull'utilizzo del valore nullo, cioè possiamo dire che il cognome non può essere nullo ed è obbligatorio, quindi, rappresentare nella relazione il cognome. Quindi dire che il cognoome è obbligatorio equivale a dire che se noi non conosciamo il cognome di una persona, allora non possiamo inserirla, poiché, se abbiamo messo questa restrizione, non può esserci una tupla con il cognome nullo.

Cosa vuol dire valore nullo? Il valore nullo viene usato almeno in tre casi differenti, questi sono i tre casi soliti, più diffusi: il primo caso è quello in cui il valore è sconosciuto, che vuol dire che questo valore esiste ma io non so qual è il suo valore e quindi per esempio posso avere una relazione in cui ho un attributo

numero di telefono e magari di qualche persona non conosco il numero di telefono. Questo non vuol dire che quella persona non ha un telefono, ma semplicemente è un limite della mia conoscenza: quella persona ha un valore per quell'attributo ma io non so quale sia il valore dell'attributo.

Un caso diverso è quello in cui il valore è inesistente, cioè io non posso riportare quel valore non perché esiste e mi è ignoto, ma perché non esiste quel valore. L'esempio che abbiamo visto prima, del secondo nome, è un esempio di questo caso, cioè io non è che non conosco il secondo nome di una persona, di Churchill, ma Churchill non aveva un secondo nome e quindi è un valore proprio inesistente per quella tupla.

Un altro esempio riportato sulla slide è quello del numero di permesso di soggiorno per un cittadino dell'Unione Europea. In questo caso il numero di permesso di soggiorno è proprio inesistente perché un cittadino della UE non ha un permesso di soggiorno.

Un terzo caso è quello in cui il valore è senza informazione, nel senso che potrebbe essere uno dei due casi precedenti ma io non so neanche in quale caso rientra, cioè il valore potrebbe essere sconosciuto o inesistente ma io non so neanche questo, quindi ho ancora meno informazioni.

Per esempio, un campo come numero di passaporto è un esempio in cui non ho informazione. Di una persona io potrei non sapere se ha il passaporto, potrebbe averlo e io non so qual è il suo numero, oppure potrebbe anche non avere neanche il passaporto, perché non l'ha mai richiesto e quindi non ha un passaporto e quindi non c'è neanche il numero di passaporto. Quindi questa terza possibilità è un'unione delle prime due possibilità.

I DBMS relazionali non distinguono tra queste tre possibilità: per tutte queste tre possibilità riportano il valore nullo, non li distinguiamo. Faccio qualche accenno ma non andremo in profondità: questo comporta qualche comportamento non proprio intuitivo e immediato dei database e nel trattamento del valore nullo.

I valori nulli ci causano dei problemi quando sono usati in alcune situazioni.

Qua ho ripreso le tre tabelle con studenti, esami, corsi che abbiamo visto in precedenza.

La differenza rispetto alla versione precedente è che ora abbiamo valori nulli. Ci sono dei casi in cui NULL non ci dà problemi, per esempio nella relazione corsi noi abbiamo che il codice 02, cioè corso con codice 02, ha un titolo NULL e docente NULL, quindi per esempio può voler dire che non è ancora stato deciso il docente del corso e magari non è stato deciso il titolo di quel corso perché è un corso nuovo e il titolo non è ancora stato formalizzato. Questo è un caso in cui la presenza di valore nullo non dà grossi problemi.

Caso diverso, per esempio, è quello che abbiamo nella tabella della relazione studenti: la terza studentessa Maria Rossi ha come matricola NULL. quel matricola NULL cosa significa? Significa che per esempio quella matricola noi non la conosciamo oppure che quella studentessa si è immatricolata da poco e non ha ancora il numero di matricola, ma questo è un problema. Perché è un problema? Perché se noi usiamo la matricola per identificare in modo univoco uno studente e se uno studente non ha numero di matricola, allora noi non possiamo identificare in modo univoco uno studente. In questa situazione noi abbiamo un unico studente con matricola NULL, ma cosa ci impedisce di avere a questo punto due studenti diversi tutti e due con matricola NULL? E a quel punto come facciamo a farvi riferimento, a dire che è proprio questo lo studente che vogliamo e non un altro studente?

Usiamo matricola come unico modo per identificare in modo univoco uno studente, perché potremmo avere due studenti diversi che hanno lo stesso cognome o avere lo stesso cognome e nome, essere nati nello stesso giorno, anche avere tutte e tre questi attributi uguali. Due studenti possono avere stesso nome e cognome ed essere nati nello stesso giorno, anche nello stesso luogo. Non è impossibile che succeda anche se raro.

Se abbiamo il numero di matricola possiamo distinguere gli studenti attraverso il numero di matricola, ma se ammettiamo che esista il valore NULL nel numero di matricola allora non abbiamo più un modo sicuro per identificare uno studente nella relazione studenti.

Sulla relazione esami abbiamo un problema diverso anche se simile. Abbiamo ammesso quindi il valore NULL sia per studente che per corso nella relazione esami, e questo significa non tanto che non possiamo più identificare univocamente un esame, cioè una tupla di una relazione, ma che non sappiamo più a cosa fa riferimento, cioè in questo caso usiamo studente come una chiave esterna (questo è un termine

che useremo tecnicamente), come un modo per collegare la tabella esami con la tabella studenti.

Se ammettiamo il valore NULL per l'attributo studente, allora non possiamo più sapere da quale studente l’esame è stato sostenuto e quindi abbiamo un esame, un 30, e non c'è modo di sapere chi ha avuto questo 30. Oppure, se abbiamo NULL nella relazione esami, nell'attributo corso, non abbiamo più modo di

sapere quel 30 per quale corso è stato conquistato e quindi abbiamo una tupla che è assolutamente inutile e danneggia la nostra base di dati, perché a questo punto se ammettiamo valori nulli per questi attributi abbiamo delle informazioni, dei pezzi di informazione, che hanno perso significato.

Questo è uno dei motivi per cui è possibile, quando si definiscono gli schemi delle relazioni, dire per ogni attributo se il valore nullo è ammesso o non è ammesso, ed è anche un motivo per cui vengono ammessi nelle basi di dati i vincoli di integrità.

Cosa sono vincoli di integrità?

Sono delle regole che vincolano le istanze possibili che può assumere una relazione in una base di dati.

Questo per fare in modo che una base di dati non contenga informazioni scorrette, perché, nel momento in cui noi ammettiamo delle informazioni scorrette in una base di dati, stiamo perdendo la qualità dei dati e quindi non è più una risorsa pregiata, che noi proteggiamo e che ha un valore nella nostra organizzazione, ma è qualcosa su cui non possiamo più fare affidamento. Se facciamo un'interrogazione, una query, in una base di dati scorretta/incoerente non possiamo più fidarci delle risposte alle nostre interrogazioni. Quindi un concetto importante nelle basi di dati relazionali, una caratteristica che distingue le basi di dati relazionali dalle base di dati NoSQL è che nelle base di dati relazionali è fondamentale la coerenza dei dati e quindi si aggiungono dei vincoli che fanno in modo che si abbia la garanzia che la base di dati sia una fonte di verità, si usa dire, cioè contenga informazioni che sono coerenti e corrette.

Ad esempio, sulla slide vedete le due relazioni esami e studenti. L'unica differenza rispetto a prima è che, a parte che ho messo studenti diversi, nella relazione esami abbiamo un attributo in più che è la lode. La lode può essere true o false, vera o falsa, quindi uno studente può avere o meno la lode.

Questo è un esempio di una base di dati scorretta, perché ha delle incongruenze.

Adesso vi lascio qualche secondo per guardare la base di dati sulla slide e provare a pensare a quali sono le

incongruenze, i valori scorretti. C'è più di un valore scorretto nella base basi di dati, quindi vi lascio un po’ di tempo per guardarla, osservate i valori e varie tuple, i valori negli attributi.

Manca la relazione sui corsi, ma facciamo conto che ci sia e che sia corretta e contenga tutti i valori che servono.

Non so se avete individuato alcuni errori, ma iniziamo a discuterne qualcuno.

Per prima cosa guardate la prima tupla della relazione esami. C'è uno studente che ha un voto di 32, questa è un esame universitario e gli esami universitari verbalizzati non possono superare il 30, quindi il voto massimo in trentesimi è 30 e 32 è un voto non ammesso.

Poi vedete la terza tupla della relazione esami, che ha due valori che non sono coerenti tra di loro, quindi, mentre nel primo esempio abbiamo un singolo valore che di per sé si può notare che non è corretto, in questo secondo caso noi abbiamo due valori che, separatamente sono valori corretti, cioè il 27 è corretto e la lode a true è corretta, ma insieme non sono coerenti, perché non si può avere il voto di 27 e lode, la lode si può avere solo con il 30, e in tutti gli altri casi la lode deve essere false.

Poi c'è una terza incoerenza, che forse non avete notato, che è nella relazione studenti. Nella relazione studenti ci sono due studenti, Piero Neri e Luca Bianchi, che hanno lo stesso numero di matricola, ma abbiamo detto che il numero di matricola ci serve per identificare in modo univoco uno studente, e se ci sono due studenti che hanno lo stesso numero di matricola noi non abbiamo più modo di identificarli. Quando riportiamo che,un certo studente con un certo numero di matricola ha superato un certo esame, e se l'esame è stato sostenuto dallo studente con matricola 787643, quale dei due, Piero Neri o Luca Bianchi ha preso quel volto in quell'esame?

Quindi in questo caso l'errore non è dato dal fatto che abbiamo due valori della stessa tupla incoerenti tra di loro, come per il 27 e lode, ma dal fatto che abbiamo due tuple diverse che prese insieme generano un'incoerenza, perché c'è una ripetizione sul numero di matricola.

Un'altra incoerenza si ha in questo altro caso - e questa è una incoerenza ancora meno ovvia. L'ultima tupla della relazione esami attribuisce il voto di 24, ovviamente senza lode, per il corso 04, allo studente con matricola 739430. Questo studente però, se guardate la tabella studenti, è uno studente che non

esiste perché non è compreso nella tabella: non c'è nessuno studente con quel numero di matricola e quindi se riportiamo un esame che è stato superato da quello studente, stiamo riportando un

esame superato da un fantasma, da qualcosa che non esiste, per cui non possiamo recuperare come si chiama, qual è il suo nome, il cognome, a che corso è iscritto, quando è nato e tutte le sue informazioni, quindi anche questo è un esempio di incoerenza.

Questo esempio di incoerenza è ancora diverso dagli altri perché per capire che è un'incoerenza, a differenza delle altre situazioni in cui abbiamo usato una singola relazione, abbiamo bisogno di mettere insieme due relazioni diverse, perché è solo guardando la relazione studenti, partendo dalla tabella esami,

che capiamo che lo studente con quel numero di matricola non esiste.

**6**

Per fare in modo che non esista e non possa esistere una relazione o un database scorretto, cioè incoerente, quando si definisce lo schema di una base di dati, si definiscono anche quali sono i vincoli di integrità che la base di dati deve soddisfare.

Il DBMS cosa fa? Prende in considerazione tutti i vincoli di integrità che abbiamo definito quando abbiamo progettato la base di dati, e ogni volta che c'è una modifica alla base di dati - questa modifica può essere un inserimento di una nuova tupla, una cancellazione di una tupla oppure una modifica di qualche tupla esistente - ogni volta che c'è una modifica, quindi, e qualcosa cambia nell’istanza della base di dati, il DBMS controlla tutti i vincoli di integrità. Se c'è almeno un vincolo che non è più soddisfatto, allora quell'operazione viene rifiutata. Cioè un DBMS in nessun modo lascia mai “passare” – cioè non esegue mai - un'operazione che fa in modo che la base di dati diventi scorretta.

Questa affermazione così assolutistica in realtà non è esattamente vera: c'è modo in diversi DBMS di disabilitare momentaneamente i vari vincoli e poi di fare qualche operazione riabilitando i vincoli in seguito, però nel normale uso delle basi di dati quello che ho detto è vero.

Cos'è un vincolo di integrità? Un vincolo di integrità è un predicato logico - quindi può valere true/false - che associa a un'istanza di una base di dati un valore di verità, cioè true o false.

Se il vincolo non è vero, il cambiamento che viene fatto - inserimento, cancellazione, modifica di tuple -

viene rifiutato.

Perché vogliamo una base di dati corretta e quindi abbiamo i vincoli di integrità?

In realtà è una cosa che abbiamo già detto: perché solo con una base di dati corretta e coerente abbiamo la garanzia che i dati che abbiamo sono corretti e coerenti e ci possiamo fidare delle risposte alle interrogazioni.

Adesso vedremo alcuni tipi di vincoli che vengono usati nelle basi di dati.

Non tutti i vincoli possibili immaginabili sono supportati dai DBMS, e nei casi in cui abbiamo bisogno di aggiungere un vincolo che non è supportato dal DBMS, bisogna affidare la responsabilità di verificare la correttezza del vincolo all'utente o al programmatore.

Vediamo alcuni tipi di vincoli.

I vincoli possono essere intrarelazionali o interrelazionali.

Intrarelazionali vuol dire che si prende in considerazione una sola relazione per volta.

Interrelazionali vuol dire che sono dei vincoli che mettono insieme, considerando in modo congiunto, relazioni diverse.

Queste sono le due relazioni che abbiamo visto prima con i vari errori evidenziati che abbiamo visto prima.

I vincoli di tupla ci permettono, per esempio, di evitare gli errori come quello del voto di 32. Possono

essere per esempio vincoli di dominio, quindi coinvolgono un solo attributo, oppure possono essere sulla

stessa tupla ma coinvolgere due attributi diversi.

Per esempio, possiamo controllare che non ci sia un voto di 32, e, assumendo che i voti inseriti nel database siano solo voti sufficienti e che le bocciature non siano verbalizzate dentro il database, allora possiamo imporre il vincolo che il voto deve essere compreso tra 18 e 30, scrivendo il vincolo di integrità, il primo che vedete sulla slide, che dice Voto>=18 And Voto<=30, dove entrambe le condizioni devono essere vere.

Questo è un esempio di un vincolo che coinvolge un singolo attributo, possiamo usare un vincolo di integrità anche per controllare che non ci sia questo 27 e lode che abbiamo visto.

In questo caso dobbiamo coinvolgere due attributi della stessa tupla, quindi un vincolo di tupla su due attributi diversi.

Il secondo esempio ci dice che NOT (Lode = True) OR Voto = 30, questo è un esempio probabilmente meno intuitivo rispetto a quello del voto singolo tra 18 e 30, ma, se vi ricordate, vi ho suggerito nell'introduzione al corso di ripassare la logica proposizionale ed è questo proprio un caso in cui conoscere la logica proposizionale ci aiuta a capire il vincolo di integrità.

Perché questo vincolo impedisce di avere 27 e lode? Possiamo vederlo in due modi.

Nel primo modo vediamo come funziona l’OR. P OR Q è vero se sono veri P, Q o entrambi, basta che non siano falsi sia P che Q. In questo caso a destra dell'OR abbiamo Voto = 30, quindi se il voto è uguale a 30 va sempre bene, sia che la lode ci sia o che non ci sia. A sinistra dell'OR abbiamo NOT (lode = true) , cioè la lode = false, quindi quando la lode è uguale a false va sempre bene, il vincolo è sempre verificato. Va anche bene se il voto è uguale a 30 e la lode c'è. Il caso in cui invece l'OR è false è quello in cui sia a sinistra che a destra abbiamo delle formule false, quindi se c'è la lode e il voto non è 30.

Un altro modo per capire questa formula è quello di ricordarci dell'equivalenza tra implicazione e NOT con OR, cioè P implica Q equivale a NOT P OR Q e questo è proprio un NOT P OR Q.

Quindi noi possiamo pensare questa formula come se fosse (lode = true) implica voto = 30. Se la lode è uguale a true allora per forza il voto deve essere uguale 30, mentre se la lode è uguale a false allora il conseguente può essere sia vero che falso e quindi voto può essere sia 30 che non 30 e va comunque bene.

Questo è un ripasso che adesso non commento sull’implicazione e la sua equivalenza con il NOT e l’OR.

Un altro esempio di un vincolo di tupla è quello in cui, invece di usare una formula proposizionale, usiamo una formula matematica, in cui abbiamo una addizione. Quindi abbiamo una relazione stipendi in cui c'è il cognome dell'impiegato, lo stipendio lordo, le ritenute per le tasse, lo stipendio netto e la formula, che è un vincolo di integrità e che dice che lo stipendio lordo deve essere uguale alle ritenute più il netto.

Se consideriamo questa relazione vediamo che la terza tupla viola questo vincolo, perché il lordo è 50 mila, ma ritenute+netto per Bruni è 11.000 + 36000 = 47.000, quindi questo vincolo non è rispettato e sappiamo che il database contiene un'informazione errata, quindi in realtà questo non è un database, non è un'istanza possibile per la nostra relazione.

Ora iniziamo a precisare questo concetto fondamentale per le basi di dati relazionali, un concetto che iniziamo a introdurre adesso in modo abbastanza intuitivo, e poi, quando parleremo tra qualche lezione di normalizzazione, sarà un concetto che preciseremo e formalizzeremo molto meglio.

Il primo concetto che introduciamo è quello di superchiave. Cos’è una superchiave?

Una superchiave è un insieme di attributi che viene usato per identificare in modo univoco le tuple di una relazione. Quello che abbiamo usato prima, la matricola nella relazione studenti, è un esempio di superchiave perché è un insieme di attributi, in questo caso un insieme composto da un singolo attributo che è matricola, che identifica in modo univoco ogni tupla della relazione studenti.

Detto in un modo leggermente più formale un insieme di attributi K è detto superchiave per una relazione r se e solo se r non può contenere due tuple che hanno valori uguali su K, cioè se r non può contenere due tuple distinte t1 e t2, quindi con t1 diverso da t2 e con t1[K] uguale a t2[K].

Questo ci dice quindi che tutte le tuple devono essere diverse almeno sugli attributi in K.

Facciamo qualche esempio. Questa è la relazione studenti in cui, rispetto a quella precedente, ho aggiunto qualche altro attributo: abbiamo matricola, codice fiscale, cognome, nome, data di nascita e corso di laurea. Adesso vi chiedo: “Quali sono le superchiavi per questa relazione?”.

Se le superchiavi sono un insieme di attributi, vuol dire che possono essere un solo attributo ma anche più attributi, cioè per esempio due attributi presi insieme per cui ogni tupla è diversa da tutte le altre tuple all'interno della relazione.

Vi lascio qualche secondo per pensare a quali sono le possibili superchiavi di questa relazione. Vi do un consiglio: non esiste una sola superchiave, ma esistono più superchiavi, superchiavi composte da un singolo attributo e anche superchiavi composte da più di un attributo.

La prima superchiave è ad esempio quella ovvia di cui abbiamo già parlato tante volte, quindi la prima superchiave è la superchiave composta da un singolo attributo che è matricola: infatti non posso avere due studenti che hanno lo stesso numero di matricola.

Però, ci sono anche altre superchiavi. Per esempio, se la definizione dice che la superchiave è un insieme di attributi per cui non è possibile che due tuple dentro la stessa relazione abbiano lo stesso valore per quegli attributi, allora se matricola è superchiave anche tutti i suoi sovrainsiemi sono una superchiave, ovviamente. Per esempio, matricola e corso di laurea presi insieme sono una superchiave, perché, se non posso avere due studenti con lo stesso numero di matricola, a maggior ragione non posso avere due studenti che hanno sia lo stesso numero di matricola sia lo stesso corso di laurea.

Quindi tutti i sovrainsiemi di matricola sono sicuramente delle superchiavi. Ad esempio, matricola e codice fiscale è un'altra superchiave e all'estremo abbiamo che l'insieme di tutti gli attributi della tabella è sicuramente una superchiave dato che matricola è una superchiave.

Anzi possiamo dire qualcosa di più: dato che una relazione è un insieme di tuple e non posso avere due tuple con lo stesso valore per tutti gli attributi, allora sicuramente in ogni relazione possibile, tutti gli attributi sono sempre una superchiave.

Vediamo invece questo altro esempio in cui prendiamo nome e corso di laurea. Insieme non sono una superchiave perché posso avere due studenti diversi che hanno lo stesso valore sia per il nome che per il corso di laurea, quindi che hanno lo stesso nome e sono iscritti allo stesso corso di laurea, come le ultime due tuple in cui abbiamo due volte Luca ed economia.

Se prendiamo nome e data di nascita ci possano chiedere: “È una superchiave o non è una superchiave?”. Se guardiamo la particolare istanza che adesso vedete sulla slide, vediamo che, mentre ci sono due studenti che si chiamano Luca, comunque differiscono per data di nascita. E ci sono due studenti, i primi due nelle prime tuple, che sono nati nello stesso giorno ma hanno due nomi diversi, quindi apparentemente uno potrebbe dire: “Ma sì, sembra che nome e data di nascita siano superchiave.”.

Però il concetto di superchiave è un concetto che non riguarda una specifica istanza di una base di dati, una specifica istanza di una relazione, ma è un concetto che riguarda tutte le possibili istanze di una relazione, quindi tutte le possibili tuple che possono essere contenute in una relazione.

La domanda che dovremmo farci in realtà dovrebbe essere: “È mai possibile che esistano due studenti che hanno lo stesso nome e sono nati nello stesso giorno?” Se la risposta è sì, allora questo significa che l'insieme di nome e data di nascita non è una superchiave, ed è solo fortuito che in questa caso sembri una superchiave.

Come abbiamo già detto prima quindi in ogni relazione sicuramente tutti gli attributi presi assieme sono sempre banalmente una superchiave. Inoltre abbiamo anche visto che ci sono altre superchiavi che in realtà sono abbastanza banali perché, se sappiamo che matricola è superchiave, allora ogni sovrainsieme di matricola, cioè ogni insieme di attributi della relazione che contiene matricola è banalmente una superchiave.

Questo è il motivo per cui adesso restringiamo il concetto di superchiave in modo da avere un concetto che ci può essere più utile.

Questo concetto è quello di chiave candidata; si può chiamare semplicemente chiave oppure si può chiamare chiave candidata.

Cos'è una chiave candidata? Una chiave candidata è una superchiave minimale. E cosa significa minimale? Significa che non contiene strettamente una superchiave, cioè io prendo una superchiave e provo a togliere qualcuno degli attributi. Se riesco a togliere almeno un attributo e quella continua a essere una superchiave allora vuol dire che quella da cui sono partito non era uno superchiave.

Se invece, nonostante tutti i miei tentativi di eliminare un attributo, perdo la proprietà che è superchiave, allora vuol dire che quella da cui sono partito è veramente minimale e quindi è una chiave o chiave candidata. Quindi ogni chiave è una superchiave: dato che ogni chiave candidata è una superchiave minimale, vuol dire che ogni chiave è banalmente superchiave. Ma non tutte le superchiavi sono chiavi perché alcune superchiavi sono minimali mentre altre, invece, non sono minimali.

Vediamo di nuovo la relazione studenti e proviamo adesso a capire quali sono le chiavi della relazione studenti. Vi lascio qualche secondo per pensare.

Iniziamo da matricola. Abbiamo già detto che matricola è una superchiave e adesso invece ci chiediamo: “Ma è anche una superchiave minimale, e quindi una chiave?”.

In questo caso abbiamo un insieme di attributi che comprende un singolo elemento: dato che comprende un singolo elemento sicuramente è minimale, perché io non posso togliere niente dato che è composto da un singolo elemento perché otterrei un insieme vuoto. Quindi matricola è sicuramente una chiave, e sicuramente tutte le superchiavi composte da un singolo attributo sono delle chiavi.

La seconda superchiave che abbiamo visto prima è quella composto da due attributi matricola e corso di laurea. Questo sappiamo che è una super chiave perché lo abbiamo già detto prima, ma sappiamo che non è minimale, perché non è minimale, perché un suo sottoinsieme proprio è una chiave, cioè matricola da solo, quindi mi è sufficiente matricola da solo per identificare uno studente e quindi corso di laurea è inutile: so che matricola e corso di laurea insieme sono una superchiave ma non sono una chiave, perché non hanno la minimalità.

Se io prendo codice fiscale da solo, ho una chiave, perché in questo caso è composto da un singolo attributo e quindi questo è un esempio del fatto che una relazione non è detto che abbia una singola chiave, ma potrebbe avere più chiavi. In generale, anche se in questo esempio tutte e due le chiavi sono composte un singolo attributo, non è detto che una chiave minimale sia composta da un singolo attributo. Posso avere altre relazioni, poi vedremo degli esempi, in cui la minimalità è garantita da due attributi, per esempio, da tre attributi, quindi la chiave è un insieme di due o tre attributi o anche di più.

Che cos'è una chiave primaria? È un concetto molto importante nelle basi di dati relazionali.

La chiave primaria di una relazione è una particolare chiave, quindi è una superchiave minimale. Tra le varie chiavi che una relazione può avere, il progettista della base di dati ha scelto una certa chiave e l’ha chiamata chiave primaria, e quindi ha deciso che sarà il modo principale per identificare in modo univoco le tuple di una relazione.

Ogni relazione nel modello relazionale ha una e una sola chiave primaria. Eventualmente la chiave primaria può essere anche composta da più attributi, ma comunque anche una chiave primaria composta da più attributi è una chiave primaria, non sono due chiavi primarie.

Guardiamo questo diagramma di Venn. Vediamo che ho un insieme di superchiavi di cui ho un sottoinsieme che è quello delle superchiavi minimali, che chiamiamo chiavi oppure chiavi candidate. Tra queste un singolo elemento, quindi non un insieme ma un singolo elemento delle chiavi è quello che il progettista ha deciso di battezzare chiave primaria.

Notate che il concetto di chiave è un concetto che è legato alla relazione, cioè un certo insieme gli attributi potrebbe essere chiave di una relazione e non essere chiave in un’altra relazione. Per esempio se scopro che matricola è una chiave della relazione studenti, non è detto che matricola sia una chiave di qualsiasi relazione in cui compare l'attributo matricola, pur continuando a rappresentare una matricola di uno studente.

Per esempio, guardiamo di nuovo l'esempio degli studenti degli esami dei corsi: qual è la chiave, quali sono le chiavi della relazione studenti in questa versione minimale?

Abbiamo già detto che sicuramente l'attributo matricola è la chiave di studenti, ma per la relazione corsi qual è la chiave? Due corsi diversi, l'abbiamo già detto, potrebbero avere lo stesso nome e quindi la chiave è il codice del corso, perché non possono esistere due corsi diversi con lo stesso codice.

Nella relazione esami qual è o quali sono le chiavi? L’attributo studente non può essere una chiave, vediamo anche l’esempio nella relazione sulla slide: possiamo vedere che ci sono alcuni studenti che hanno ovviamente sostenuto più esami, superato più esami, ad esempio lo studente 123456 ha superato più di un esame quindi sicuramente non è una chiave, perché è un valore duplicato. Quindi questo è anche un esempio di quello che dicevamo poco fa: la matricola è una chiave della relazione studenti ma per esempio, in questo caso, non è una chiave della relazione esami.

Corso non è una chiave perché per lo stesso corso posso avere più tuple nella relazione esami perché lo stesso corso ha esami sostenuti da più studenti: ad esempio ci sono più studenti che han dato l'esame di inglese, di informatica e di diritto.

Voto non può essere una chiave perché ovviamente più studenti possono prendere lo stesso voto, lode non può essere una chiave, ma neanche data può essere una chiave perché due studenti possono dare due esami diversi o anche lo stesso esame nello stesso giorno.

Quindi questo significa che nessun attributo singolo è una chiave di esami, perché banalmente nessun attributo singolo è neanche una superchiave. Ci rimangono da considerare per esempio le coppie di attributi e quindi possiamo, per esempio, pensare che la coppia studente - corso potrebbe essere una superchiave. Perché potrebbe essere una super chiave? Perché uno stesso studente non può dare l'esame dello stesso corso due volte, una volta che ha superato ed è stato verbalizzato l'esame, non può ridare nuovamente quell'esame. Ci sono alcuni casi in cui queste cose sono ammesse, non so se ad informatica, in altre scuole o altri corsi di laurea in questi casi al corso veniva assegnato un altro codice.

Quindi questo vuol dire che, mentre studente non è una superchiave da solo e corso non è una superchiave da solo per la relazione esami, la coppia studente - corso è una superchiave, e in questo caso è anche una chiave, perché, se tolgo dalla coppia studente – corso lo studente, ottengo solo corso che abbiamo già detto che non è una superchiave. Se tolgo invece corso, ottengo solo studente che abbiamo già detto che non è una superchiave. Quindi questo è anche un esempio di una chiave, cioè una superchiave minimale, che è composta da due attributi perché non esistono delle chiavi composte da un singolo attributo.

Perché parliamo di superchiavi e di chiavi?

Perché sono un concetto basilare nel modello relazionale, perché abbiamo detto che nel modello relazionale per far "comunicare", per "congiungere" relazioni diverse tra di loro si usa unicamente il valore degli attributi.

Ma se usiamo il valore degli attributi allora è necessario potere fare riferimento alle tuple di una relazione in modo univoco. Come facciamo a far riferimento in modo univoco attraverso il valore degli attributi? Usando le chiavi. Poi vedremo molto più nel dettaglio questo concetto, per ora ne parliamo solo a livello intuitivo e introduttivo.

Qual è la relazione tra chiavi e valori nulli? Abbiamo già visto la presenza dei valori nulli può dare dei problemi nelle chiavi. Perché? Rivediamo qualche esempio.

In questa relazione nella prima tupla vediamo che c'è uno studente che ha matricola nulla, ma se ha matricola nulla allora la chiave è incompleta perché possiamo avere, come nell'esempio, due studenti che hanno valore nullo per la matricola. A questo punto non possiamo più usare matricola come chiave perché non è più una superchiave, dato che non ci permette di identificare in modo univoco una tupla nella relazione, cioè uno studente tra gli studenti.

Questo è un altro esempio che elabora l'esempio precedente.

La conseguenza del ragionamento è che la chiave primaria non può assumere valori nulli, e infatti nel modello relazionale non è ammesso che le chiavi primarie assumano valori nulli.

Nella pratica quando in SQL creerete delle tabelle, deciderete quali sono gli attributi che formano la chiave primaria, e automaticamente il DBMS, PostgreSQL o qualunque altro DBMS usiate, aggiungerà in modo implicito il vincolo che quegli attributi che compongono la chiave primaria non possono assumere valori nulli, poiché se assumessero valori nulli non potremmo più usarli come chiave primaria.

Come si rappresenta graficamente la chiave primaria? Sottolineando gli attributi che la compongono. Quindi quando scriviamo lo schema di una relazione mettiamo il nome della relazione e tra parentesi l'elenco degli attributi, e quelli che compongono la chiave primaria li sottolineiamo, quindi non sottolineiamo le chiavi candidate e neanche le super chiavi: sottolineiamo solo

quella che abbiamo deciso che sarà la chiave primaria.

**7**

Per fare in modo che non esista e non possa esistere una relazione o un database scorretto, cioè incoerente, quando si definisce lo schema di una base di dati, si definiscono anche quali sono i vincoli di integrità che la base di dati deve soddisfare.

Il DBMS cosa fa? Prende in considerazione tutti i vincoli di integrità che abbiamo definito quando abbiamo progettato la base di dati, e ogni volta che c'è una modifica alla base di dati - questa modifica può essere un inserimento di una nuova tupla, una cancellazione di una tupla oppure una modifica di qualche tupla esistente - ogni volta che c'è una modifica, quindi, e qualcosa cambia nell’istanza della base di dati, il DBMS controlla tutti i vincoli di integrità. Se c'è almeno un vincolo che non è più soddisfatto, allora quell'operazione viene rifiutata. Cioè un DBMS in nessun modo lascia mai “passare” – cioè non esegue mai - un'operazione che fa in modo che la base di dati diventi scorretta.

Questa affermazione così assolutistica in realtà non è esattamente vera: c'è modo in diversi DBMS di disabilitare momentaneamente i vari vincoli e poi di fare qualche operazione riabilitando i vincoli in seguito, però nel normale uso delle basi di dati quello che ho detto è vero.

Cos'è un vincolo di integrità? Un vincolo di integrità è un predicato logico - quindi può valere true/false - che associa a un'istanza di una base di dati un valore di verità, cioè true o false.

Se il vincolo non è vero, il cambiamento che viene fatto - inserimento, cancellazione, modifica di tuple -

viene rifiutato.

Perché vogliamo una base di dati corretta e quindi abbiamo i vincoli di integrità?

In realtà è una cosa che abbiamo già detto: perché solo con una base di dati corretta e coerente abbiamo la garanzia che i dati che abbiamo sono corretti e coerenti e ci possiamo fidare delle risposte alle interrogazioni.

Adesso vedremo alcuni tipi di vincoli che vengono usati nelle basi di dati.

Non tutti i vincoli possibili immaginabili sono supportati dai DBMS, e nei casi in cui abbiamo bisogno di aggiungere un vincolo che non è supportato dal DBMS, bisogna affidare la responsabilità di verificare la correttezza del vincolo all'utente o al programmatore.

Vediamo alcuni tipi di vincoli.

I vincoli possono essere intrarelazionali o interrelazionali.

Intrarelazionali vuol dire che si prende in considerazione una sola relazione per volta.

Interrelazionali vuol dire che sono dei vincoli che mettono insieme, considerando in modo congiunto, relazioni diverse.

Queste sono le due relazioni che abbiamo visto prima con i vari errori evidenziati che abbiamo visto prima.

I vincoli di tupla ci permettono, per esempio, di evitare gli errori come quello del voto di 32. Possono

essere per esempio vincoli di dominio, quindi coinvolgono un solo attributo, oppure possono essere sulla

stessa tupla ma coinvolgere due attributi diversi.

Per esempio, possiamo controllare che non ci sia un voto di 32, e, assumendo che i voti inseriti nel database siano solo voti sufficienti e che le bocciature non siano verbalizzate dentro il database, allora possiamo imporre il vincolo che il voto deve essere compreso tra 18 e 30, scrivendo il vincolo di integrità, il primo che vedete sulla slide, che dice Voto>=18 And Voto<=30, dove entrambe le condizioni devono essere vere.

Questo è un esempio di un vincolo che coinvolge un singolo attributo, possiamo usare un vincolo di integrità anche per controllare che non ci sia questo 27 e lode che abbiamo visto.

In questo caso dobbiamo coinvolgere due attributi della stessa tupla, quindi un vincolo di tupla su due attributi diversi.

Il secondo esempio ci dice che NOT (Lode = True) OR Voto = 30, questo è un esempio probabilmente meno intuitivo rispetto a quello del voto singolo tra 18 e 30, ma, se vi ricordate, vi ho suggerito nell'introduzione al corso di ripassare la logica proposizionale ed è questo proprio un caso in cui conoscere la logica proposizionale ci aiuta a capire il vincolo di integrità.

Perché questo vincolo impedisce di avere 27 e lode? Possiamo vederlo in due modi.

Nel primo modo vediamo come funziona l’OR. P OR Q è vero se sono veri P, Q o entrambi, basta che non siano falsi sia P che Q. In questo caso a destra dell'OR abbiamo Voto = 30, quindi se il voto è uguale a 30 va sempre bene, sia che la lode ci sia o che non ci sia. A sinistra dell'OR abbiamo NOT (lode = true) , cioè la lode = false, quindi quando la lode è uguale a false va sempre bene, il vincolo è sempre verificato. Va anche bene se il voto è uguale a 30 e la lode c'è. Il caso in cui invece l'OR è false è quello in cui sia a sinistra che a destra abbiamo delle formule false, quindi se c'è la lode e il voto non è 30.

Un altro modo per capire questa formula è quello di ricordarci dell'equivalenza tra implicazione e NOT con OR, cioè P implica Q equivale a NOT P OR Q e questo è proprio un NOT P OR Q.

Quindi noi possiamo pensare questa formula come se fosse (lode = true) implica voto = 30. Se la lode è uguale a true allora per forza il voto deve essere uguale 30, mentre se la lode è uguale a false allora il conseguente può essere sia vero che falso e quindi voto può essere sia 30 che non 30 e va comunque bene.

Questo è un ripasso che adesso non commento sull’implicazione e la sua equivalenza con il NOT e l’OR.

Un altro esempio di un vincolo di tupla è quello in cui, invece di usare una formula proposizionale, usiamo una formula matematica, in cui abbiamo una addizione. Quindi abbiamo una relazione stipendi in cui c'è il cognome dell'impiegato, lo stipendio lordo, le ritenute per le tasse, lo stipendio netto e la formula, che è un vincolo di integrità e che dice che lo stipendio lordo deve essere uguale alle ritenute più il netto.

Se consideriamo questa relazione vediamo che la terza tupla viola questo vincolo, perché il lordo è 50 mila, ma ritenute+netto per Bruni è 11.000 + 36000 = 47.000, quindi questo vincolo non è rispettato e sappiamo che il database contiene un'informazione errata, quindi in realtà questo non è un database, non è un'istanza possibile per la nostra relazione.

Ora iniziamo a precisare questo concetto fondamentale per le basi di dati relazionali, un concetto che iniziamo a introdurre adesso in modo abbastanza intuitivo, e poi, quando parleremo tra qualche lezione di normalizzazione, sarà un concetto che preciseremo e formalizzeremo molto meglio.

Il primo concetto che introduciamo è quello di superchiave. Cos’è una superchiave?

Una superchiave è un insieme di attributi che viene usato per identificare in modo univoco le tuple di una relazione. Quello che abbiamo usato prima, la matricola nella relazione studenti, è un esempio di superchiave perché è un insieme di attributi, in questo caso un insieme composto da un singolo attributo che è matricola, che identifica in modo univoco ogni tupla della relazione studenti.

Detto in un modo leggermente più formale un insieme di attributi K è detto superchiave per una relazione r se e solo se r non può contenere due tuple che hanno valori uguali su K, cioè se r non può contenere due tuple distinte t1 e t2, quindi con t1 diverso da t2 e con t1[K] uguale a t2[K].

Questo ci dice quindi che tutte le tuple devono essere diverse almeno sugli attributi in K.

Facciamo qualche esempio. Questa è la relazione studenti in cui, rispetto a quella precedente, ho aggiunto qualche altro attributo: abbiamo matricola, codice fiscale, cognome, nome, data di nascita e corso di laurea. Adesso vi chiedo: “Quali sono le superchiavi per questa relazione?”.

Se le superchiavi sono un insieme di attributi, vuol dire che possono essere un solo attributo ma anche più attributi, cioè per esempio due attributi presi insieme per cui ogni tupla è diversa da tutte le altre tuple all'interno della relazione.

Vi lascio qualche secondo per pensare a quali sono le possibili superchiavi di questa relazione. Vi do un consiglio: non esiste una sola superchiave, ma esistono più superchiavi, superchiavi composte da un singolo attributo e anche superchiavi composte da più di un attributo.

La prima superchiave è ad esempio quella ovvia di cui abbiamo già parlato tante volte, quindi la prima superchiave è la superchiave composta da un singolo attributo che è matricola: infatti non posso avere due studenti che hanno lo stesso numero di matricola.

Però, ci sono anche altre superchiavi. Per esempio, se la definizione dice che la superchiave è un insieme di attributi per cui non è possibile che due tuple dentro la stessa relazione abbiano lo stesso valore per quegli attributi, allora se matricola è superchiave anche tutti i suoi sovrainsiemi sono una superchiave, ovviamente. Per esempio, matricola e corso di laurea presi insieme sono una superchiave, perché, se non posso avere due studenti con lo stesso numero di matricola, a maggior ragione non posso avere due studenti che hanno sia lo stesso numero di matricola sia lo stesso corso di laurea.

Quindi tutti i sovrainsiemi di matricola sono sicuramente delle superchiavi. Ad esempio, matricola e codice fiscale è un'altra superchiave e all'estremo abbiamo che l'insieme di tutti gli attributi della tabella è sicuramente una superchiave dato che matricola è una superchiave.

Anzi possiamo dire qualcosa di più: dato che una relazione è un insieme di tuple e non posso avere due tuple con lo stesso valore per tutti gli attributi, allora sicuramente in ogni relazione possibile, tutti gli attributi sono sempre una superchiave.

Vediamo invece questo altro esempio in cui prendiamo nome e corso di laurea. Insieme non sono una superchiave perché posso avere due studenti diversi che hanno lo stesso valore sia per il nome che per il corso di laurea, quindi che hanno lo stesso nome e sono iscritti allo stesso corso di laurea, come le ultime due tuple in cui abbiamo due volte Luca ed economia.

Se prendiamo nome e data di nascita ci possano chiedere: “È una superchiave o non è una superchiave?”. Se guardiamo la particolare istanza che adesso vedete sulla slide, vediamo che, mentre ci sono due studenti che si chiamano Luca, comunque differiscono per data di nascita. E ci sono due studenti, i primi due nelle prime tuple, che sono nati nello stesso giorno ma hanno due nomi diversi, quindi apparentemente uno potrebbe dire: “Ma sì, sembra che nome e data di nascita siano superchiave.”.

Però il concetto di superchiave è un concetto che non riguarda una specifica istanza di una base di dati, una specifica istanza di una relazione, ma è un concetto che riguarda tutte le possibili istanze di una relazione, quindi tutte le possibili tuple che possono essere contenute in una relazione.

La domanda che dovremmo farci in realtà dovrebbe essere: “È mai possibile che esistano due studenti che hanno lo stesso nome e sono nati nello stesso giorno?” Se la risposta è sì, allora questo significa che l'insieme di nome e data di nascita non è una superchiave, ed è solo fortuito che in questa caso sembri una superchiave.

Come abbiamo già detto prima quindi in ogni relazione sicuramente tutti gli attributi presi assieme sono sempre banalmente una superchiave. Inoltre abbiamo anche visto che ci sono altre superchiavi che in realtà sono abbastanza banali perché, se sappiamo che matricola è superchiave, allora ogni sovrainsieme di matricola, cioè ogni insieme di attributi della relazione che contiene matricola è banalmente una superchiave.

Questo è il motivo per cui adesso restringiamo il concetto di superchiave in modo da avere un concetto che ci può essere più utile.

Questo concetto è quello di chiave candidata; si può chiamare semplicemente chiave oppure si può chiamare chiave candidata.

Cos'è una chiave candidata? Una chiave candidata è una superchiave minimale. E cosa significa minimale? Significa che non contiene strettamente una superchiave, cioè io prendo una superchiave e provo a togliere qualcuno degli attributi. Se riesco a togliere almeno un attributo e quella continua a essere una superchiave allora vuol dire che quella da cui sono partito non era uno superchiave.

Se invece, nonostante tutti i miei tentativi di eliminare un attributo, perdo la proprietà che è superchiave, allora vuol dire che quella da cui sono partito è veramente minimale e quindi è una chiave o chiave candidata. Quindi ogni chiave è una superchiave: dato che ogni chiave candidata è una superchiave minimale, vuol dire che ogni chiave è banalmente superchiave. Ma non tutte le superchiavi sono chiavi perché alcune superchiavi sono minimali mentre altre, invece, non sono minimali.

Vediamo di nuovo la relazione studenti e proviamo adesso a capire quali sono le chiavi della relazione studenti. Vi lascio qualche secondo per pensare.

Iniziamo da matricola. Abbiamo già detto che matricola è una superchiave e adesso invece ci chiediamo: “Ma è anche una superchiave minimale, e quindi una chiave?”.

In questo caso abbiamo un insieme di attributi che comprende un singolo elemento: dato che comprende un singolo elemento sicuramente è minimale, perché io non posso togliere niente dato che è composto da un singolo elemento perché otterrei un insieme vuoto. Quindi matricola è sicuramente una chiave, e sicuramente tutte le superchiavi composte da un singolo attributo sono delle chiavi.

La seconda superchiave che abbiamo visto prima è quella composto da due attributi matricola e corso di laurea. Questo sappiamo che è una super chiave perché lo abbiamo già detto prima, ma sappiamo che non è minimale, perché non è minimale, perché un suo sottoinsieme proprio è una chiave, cioè matricola da solo, quindi mi è sufficiente matricola da solo per identificare uno studente e quindi corso di laurea è inutile: so che matricola e corso di laurea insieme sono una superchiave ma non sono una chiave, perché non hanno la minimalità.

Se io prendo codice fiscale da solo, ho una chiave, perché in questo caso è composto da un singolo attributo e quindi questo è un esempio del fatto che una relazione non è detto che abbia una singola chiave, ma potrebbe avere più chiavi. In generale, anche se in questo esempio tutte e due le chiavi sono composte un singolo attributo, non è detto che una chiave minimale sia composta da un singolo attributo. Posso avere altre relazioni, poi vedremo degli esempi, in cui la minimalità è garantita da due attributi, per esempio, da tre attributi, quindi la chiave è un insieme di due o tre attributi o anche di più.

Che cos'è una chiave primaria? È un concetto molto importante nelle basi di dati relazionali.

La chiave primaria di una relazione è una particolare chiave, quindi è una superchiave minimale. Tra le varie chiavi che una relazione può avere, il progettista della base di dati ha scelto una certa chiave e l’ha chiamata chiave primaria, e quindi ha deciso che sarà il modo principale per identificare in modo univoco le tuple di una relazione.

Ogni relazione nel modello relazionale ha una e una sola chiave primaria. Eventualmente la chiave primaria può essere anche composta da più attributi, ma comunque anche una chiave primaria composta da più attributi è una chiave primaria, non sono due chiavi primarie.

Guardiamo questo diagramma di Venn. Vediamo che ho un insieme di superchiavi di cui ho un sottoinsieme che è quello delle superchiavi minimali, che chiamiamo chiavi oppure chiavi candidate. Tra queste un singolo elemento, quindi non un insieme ma un singolo elemento delle chiavi è quello che il progettista ha deciso di battezzare chiave primaria.

Notate che il concetto di chiave è un concetto che è legato alla relazione, cioè un certo insieme gli attributi potrebbe essere chiave di una relazione e non essere chiave in un’altra relazione. Per esempio se scopro che matricola è una chiave della relazione studenti, non è detto che matricola sia una chiave di qualsiasi relazione in cui compare l'attributo matricola, pur continuando a rappresentare una matricola di uno studente.

Per esempio, guardiamo di nuovo l'esempio degli studenti degli esami dei corsi: qual è la chiave, quali sono le chiavi della relazione studenti in questa versione minimale?

Abbiamo già detto che sicuramente l'attributo matricola è la chiave di studenti, ma per la relazione corsi qual è la chiave? Due corsi diversi, l'abbiamo già detto, potrebbero avere lo stesso nome e quindi la chiave è il codice del corso, perché non possono esistere due corsi diversi con lo stesso codice.

Nella relazione esami qual è o quali sono le chiavi? L’attributo studente non può essere una chiave, vediamo anche l’esempio nella relazione sulla slide: possiamo vedere che ci sono alcuni studenti che hanno ovviamente sostenuto più esami, superato più esami, ad esempio lo studente 123456 ha superato più di un esame quindi sicuramente non è una chiave, perché è un valore duplicato. Quindi questo è anche un esempio di quello che dicevamo poco fa: la matricola è una chiave della relazione studenti ma per esempio, in questo caso, non è una chiave della relazione esami.

Corso non è una chiave perché per lo stesso corso posso avere più tuple nella relazione esami perché lo stesso corso ha esami sostenuti da più studenti: ad esempio ci sono più studenti che han dato l'esame di inglese, di informatica e di diritto.

Voto non può essere una chiave perché ovviamente più studenti possono prendere lo stesso voto, lode non può essere una chiave, ma neanche data può essere una chiave perché due studenti possono dare due esami diversi o anche lo stesso esame nello stesso giorno.

Quindi questo significa che nessun attributo singolo è una chiave di esami, perché banalmente nessun attributo singolo è neanche una superchiave. Ci rimangono da considerare per esempio le coppie di attributi e quindi possiamo, per esempio, pensare che la coppia studente - corso potrebbe essere una superchiave. Perché potrebbe essere una super chiave? Perché uno stesso studente non può dare l'esame dello stesso corso due volte, una volta che ha superato ed è stato verbalizzato l'esame, non può ridare nuovamente quell'esame. Ci sono alcuni casi in cui queste cose sono ammesse, non so se ad informatica, in altre scuole o altri corsi di laurea in questi casi al corso veniva assegnato un altro codice.

Quindi questo vuol dire che, mentre studente non è una superchiave da solo e corso non è una superchiave da solo per la relazione esami, la coppia studente - corso è una superchiave, e in questo caso è anche una chiave, perché, se tolgo dalla coppia studente – corso lo studente, ottengo solo corso che abbiamo già detto che non è una superchiave. Se tolgo invece corso, ottengo solo studente che abbiamo già detto che non è una superchiave. Quindi questo è anche un esempio di una chiave, cioè una superchiave minimale, che è composta da due attributi perché non esistono delle chiavi composte da un singolo attributo.

Perché parliamo di superchiavi e di chiavi?

Perché sono un concetto basilare nel modello relazionale, perché abbiamo detto che nel modello relazionale per far "comunicare", per "congiungere" relazioni diverse tra di loro si usa unicamente il valore degli attributi.

Ma se usiamo il valore degli attributi allora è necessario potere fare riferimento alle tuple di una relazione in modo univoco. Come facciamo a far riferimento in modo univoco attraverso il valore degli attributi? Usando le chiavi. Poi vedremo molto più nel dettaglio questo concetto, per ora ne parliamo solo a livello intuitivo e introduttivo.

Qual è la relazione tra chiavi e valori nulli? Abbiamo già visto la presenza dei valori nulli può dare dei problemi nelle chiavi. Perché? Rivediamo qualche esempio.

In questa relazione nella prima tupla vediamo che c'è uno studente che ha matricola nulla, ma se ha matricola nulla allora la chiave è incompleta perché possiamo avere, come nell'esempio, due studenti che hanno valore nullo per la matricola. A questo punto non possiamo più usare matricola come chiave perché non è più una superchiave, dato che non ci permette di identificare in modo univoco una tupla nella relazione, cioè uno studente tra gli studenti.

Questo è un altro esempio che elabora l'esempio precedente.

La conseguenza del ragionamento è che la chiave primaria non può assumere valori nulli, e infatti nel modello relazionale non è ammesso che le chiavi primarie assumano valori nulli.

Nella pratica quando in SQL creerete delle tabelle, deciderete quali sono gli attributi che formano la chiave primaria, e automaticamente il DBMS, PostgreSQL o qualunque altro DBMS usiate, aggiungerà in modo implicito il vincolo che quegli attributi che compongono la chiave primaria non possono assumere valori nulli, poiché se assumessero valori nulli non potremmo più usarli come chiave primaria.

Come si rappresenta graficamente la chiave primaria? Sottolineando gli attributi che la compongono. Quindi quando scriviamo lo schema di una relazione mettiamo il nome della relazione e tra parentesi l'elenco degli attributi, e quelli che compongono la chiave primaria li sottolineiamo, quindi non sottolineiamo le chiavi candidate e neanche le super chiavi: sottolineiamo solo

quella che abbiamo deciso che sarà la chiave primaria.

**8**

Con questa lezione inizieremo il discorso sulla teoria della normalizzazione nelle basi di dati relazionali. È un argomento che prenderà diverse lezioni e come lunghezza rivaleggerà con l'algebra relazionale; forse impiegheremo più ore su questo argomento di quante ne abbiamo impiegate sull’algebra relazionale, quindi non vi scoraggiate.

All'inizio sarà un discorso che sembrerà ostico ma poi con pazienza lo affronterete sicuramente.

Questo è l’outline di questa lezione quindi non di tutto il discorso sulla teoria della normalizzazione, ma solo di questa prima lezione. Iniziamo con una introduzione all'argomento poi parliamo delle varie anomalie che si possono incontrare nelle basi di dati. Iniziamo ad introdurre il discorso delle dipendenze funzionali che sono la base della teoria della normalizzazione, poi parleremo della teoria di Armstrong che ci dice come maneggiare queste dipendenze funzionali e poi parleremo della chiusura di dipendenze funzionali e chiusure di insiemi di attributi, che ci servono come strumento per maneggiare la normalizzazione e anche per ripensare a cosa significa appunto concetto di chiave e concetto di super chiave. La base di tutto questo discorso sono le forme normali.

Cos'è una forma normale? È una proprietà di una base di dati relazionale che garantisce che la base di dati abbia una certa qualità. Qualità che in questo contesto significa assenza di alcuni determinati difetti. Quando parliamo di forme normali e di normalizzazione, pensiamo sempre allo schema della base di dati, quindi non tanto al contenuto quanto allo schema e determiniamo se lo schema può dar luogo a delle anomalie oppure no.

Quando una relazione, una base di dati, non è in forma normale, i problemi che abbiamo sono di avere ridondanze e quando compiamo certe operazioni (operazioni di aggiornamento, cancellazione, inserimento di dati) possiamo avere delle anomalie. Notate che queste tre operazioni (aggiornamento, inserimento e cancellazione) sono tutte le operazioni che si possono compiere su una base di dati e che alterano i dati, poi c'è una quarta operazione che è quella di interrogazione della base di dati.

Le forme normali sono definite sul modello relazionale. Alla fine di questo discorso sulla teoria della normalizzazione guarderemo anche qual è il rapporto tra le forme normali e il modello entity-relationship.

La metodologia che abbiamo visto di progettazione e di cui abbiamo parlato nelle lezioni di laboratorio i solito permette di progettare degli schemi di basi di dati che sono già in forma normale e quindi questa proprietà desiderabile delle basi di dati viene “gratis”. Non è detto, comunque, che sia così.

La normalizzazione è il processo che permette di partire da degli schemi di basi di dati relazionali qualunque e trasformarli in schemi che soddisfano una forma normale.

Esistono diverse forme normali, non ne parleremo in questa lezione perché prima abbiamo bisogno di costruire una certa base per riuscire poi a parlare di cosa sono le forme normali. Comunque possiamo già dire che appunto la normalizzazione è questo processo che permette di ottenere delle basi di dati che sono in forma normale.

La normalizzazione, dato che viene ottenuta quasi sempre attraverso la metodologia di progettazione di cui abbiamo parlato, non deve essere vista come una metodologia di progettazione ma piuttosto come una tecnica che verifica che il risultato della progettazione sia corretto, cioè che abbia una certa qualità perché rispetta le forme normali.

Inoltre, possiamo pensare di applicare la normalizzazione a basi di dati che sono state progettate per esempio da altre persone, non con la metodologia che abbiamo spiegato, quindi potremmo trovarci a lavorare su un progetto già esistente che magari viene ereditato dal passato e che non è stato progettato bene. La normalizzazione ci permette di capire quali sono i problemi di questa base di dati preesistente e come risolvere questi problemi modificandola e quindi normalizzando questa base di dati.

Ora possiamo iniziare a discutere di quali sono i problemi che possiamo trovare in una base di dati che non è normalizzata. Per introdurre gli esempi introduciamo questa relazione che si chiama Esami, che è diversa dalle relazioni Esami che abbiamo già visto durante le scorsi lezioni perché come vedete sulla slide ha molti più attributi. Questa relazione Esami ha l’attributo matricola dello studente che ha sostenuto l'esame, poi il nome dello studente, il suo indirizzo, il CAP dello studente, il codice fiscale dello studente, la sua data di nascita, poi abbiamo il corso di cui lo studente ha sostenuto l'esame, il voto che ha ricevuto, se ha avuto la lode o no, quindi booleano, qual è stata la data d’esame, qual è il codice del professore che è titolare del corso, come si chiama questo professore, qual è la sua qualifica nel senso se è professore ordinario, professore associato o ricercatore o docente a contratto e poi qual è il tipo di ufficio del professore.

Quindi una relazione che può essere un po’ strana perché è una relazione molto grossa che ha un mucchio di attributi che non metteremmo dentro la stessa relazione se noi costruissimo prima l’entity-relationship. Comunque, dal punto di vista del modello relazionale è una relazione valida, non viola nessun problema, nessun requisito delle basi di dati relazionali.

La chiave primaria di questa relazione è composta dai due attributi matricola e corso perché appunto questa è una relazione esami e quindi uno studente non può superare due volte l'esame di un certo corso e assumiamo di registrare solo i voti sufficienti e non rifiutati. Quindi la chiave primaria appunto è la matricola dello studente insieme al codice del corso che è stato superato.

Dato che useremo spesso questa relazione ci conviene introdurre delle abbreviazioni per gli attributi in modo da rendere un po’ più snelle le slide e anche la nostra trattazione quindi matricola diventa MATR, nome studente diventa NS, indirizzo studente diventa IS, CAP dello studente diventa CAP , codice fiscale dello studente è CF, data di nascita dello studente diventa DN, il codice del corso è Co con la o minuscola, il voto è VO, la lode è LO, la data dell'esame DE ,il codice del professore CP, nome professore NP, la qualifica Q, tipo ufficio TU, quindi usiamo adesso queste abbreviazioni.

Ora vediamo un'istanza di questa relazione in cui mettiamo le prime 4 tuple, vediamo che la prima tupla è dello studente che ha matricola 341, si chiama Piero, abita a Torino, il CAP è 101, il codice fiscale è PX (non mettiamo tutti i dati per non esagerare, il codice fiscale ovviamente non può essere solo PX, il CAP non può essere solo 101).

La prima tupla ci dice che lo studente Piero che ha matricola 341 ha superato l'esame di basi di dati con 27 con il professore Anselma che ha codice p1 e che è un ricercatore; poi la seconda tupla ci dice che lo studente Giorgio che ha matricola 343 ha superato l'esame di programmazione 1 con 30 e lode, del professore Cardone che è un professore associato; la terza tupla ci dice che lo studente Piero, quindi il primo studente, ha superato anche l'esame di programmazione 1 con 25 col professor Cardone e l'ultima delle quattro tuple ci dice che lo studente 343 ha superato anche l'esame di basi di dati con 18 con il professore Anselma e poi ci sono puntini perché ci possono essere altre tuple.

Questa è un'istanza ragionevole, non è un'istanza strana, è un’istanza normale e vediamo che adesso usando questa istanza ragionevole e provando a inserire, cancellare e modificare dei dati ci possono essere delle anomalie, cioè dei comportamenti strani di questa base di dati e poi vedremo che questi comportamenti strani sono causati proprio dalla progettazione della base di dati che è stata una progettazione errata.

Prima anomalia: anomalia di inserimento.

Supponiamo che ci sia un nuovo studente che si immatricola, allora questa relazione esami che abbiamo è la relazione che contiene tutti i dati. Nella base di dati non c'è una relazione studenti, che ci sarebbe stata se avessimo usato la procedura di progettazione delle basi dati che abbiamo visto. Questa è l'unica relazione di tutto il database, perché se è fatta così non ci servono altre relazioni dato che contiene tutti i dati di cui abbiamo bisogno sugli studenti, tutti i dati di cui abbiamo bisogno sui professori e tutti i dati sugli esami.

Quindi se un nuovo studente si immatricola dove lo inseriamo?

Lo inseriamo ovviamente in questa relazione che è l’unica che c'è. Se lo studente non lo inseriamo, allora non c'è traccia di questo studente nel nostro database e quindi è come se non si fosse mai iscritto: non abbiamo scritto nessuna parte nel nostro sistema informativo che c'è un nuovo studente in più.

Questo nuovo studente però, se si è appena immatricolato, ovviamente non ha ancora superato nessun esame quindi non posso non inserirlo, altrimenti non ce n'è traccia.

C’è una studentessa che si chiama Laura e ha matricola 444, io ovviamente posso inserire tutti gli attributi che riguardano lo studente ma non posso inserire nessun attributo che è correlato con l’esame che questa studentessa ha superato perché non ha superato nessun esame e quindi cosa scrivo al posto del codice del corso e del voto e della lode?

Scriverò NULL come anche saranno NULL tutti gli attributi relativi al professore che ha tenuto il corso di cui la studentessa ha superato l'esame dato che non ha superato nessun esame. Quindi una parte della tupla è valorizzata con i dati della studentessa e un'altra parte non è valorizzata perché non ha superato ancora nessun esame.

È ammessa questa istanza della base di dati nuova che abbiamo? Osserviamo la chiave primaria, che è composta da matricola e codice del corso, e vediamo che è un errore. Non possiamo avere questa tupla inserita in questo modo perché avremmo un attributo che fa parte della chiave primaria che ha valore nullo, cioè il codice del corso di cui la studentessa ha superato l'esame è NULL, quindi non possiamo averlo.

Una situazione simile si ha quando viene assunto un nuovo docente: se il nuovo docente viene assunto allora non ha ancora verbalizzato nessun esame, ma dobbiamo inserirlo nel database, altrimenti non ce n'è traccia nel sistema informativo.

Se inseriamo questo professore allora dobbiamo inserire una tupla in cui abbiamo solo i dati del professore, quindi il suo codice, il suo nome, la qualifica e il tipo di ufficio ma tutti gli altri attributi della tupla sono NULL e quindi sono necessariamente NULL le due tuple che fanno parte della chiave primaria: matricola e codice del corso e quindi non possiamo inserirlo.

Questi due sono esempi di anomalie di inserimento, cioè noi abbiamo provato a inserire una tupla e non ci siamo riusciti, quindi abbiamo una criticità di esprimibilità, nel senso che noi vogliamo inserire una informazione che è sensata ed è significativa per il nostro sistema informativo ma non possiamo farlo perché lo schema della base di dati ci impedisce di rappresentare questa informazione, cioè quella di uno studente che non ha ancora superato nessun esame o di un professore che non ha ancora verbalizzato nessun esame.

Nel caso della cancellazione, abbiamo di nuovo un'anomalia, quindi le anomalie sono presenti non solo con le operazioni di inserimento: supponiamo di voler cancellare gli esami di tutti gli studenti che sono laureati.

Questa è un'operazione che può essere sensata da eseguire ciclicamente ogni tanto (una volta l’anno). Possiamo pensare di archiviare in un altro database che non è online i dati degli studenti che si sono laureati e quindi non abbiamo più bisogno di tenere sempre a disposizione i voti dei loro esami e vogliamo rimuovere queste informazioni dal nostro database.

Se io lo faccio e quindi cancello gli esami di tutti gli studenti laureati che cosa succede ai professori? Perché non è impossibile, ma può succedere, che qualche professore non rimanga più nel database, perché questo professore magari non ha verbalizzato nessun esame di studenti che devono ancora laurearsi, ha solo verbalizzato esami di studenti che si sono già laureati. Questo può succedere magari per un giorno due giorni, non un periodo lungo, però può succedere.

Quindi supponiamo che, nel nostro esempio, sia lo studente 341 che lo studente 343 si siano laureati e ora vogliamo cancellare dal database questi dati dello studente e facendolo cancelliamo queste quattro tuple che vedete sulla slide.

Se lo facciamo però stiamo cancellando anche necessariamente i dati dei professori se questi professori, ad esempio Cardone, non ha altre tuple con qualche studente che non si è laureato più in basso, nelle tuple non mostrate.

Quindi noi vogliamo in questo caso cancellare un'informazione, quella sugli esami degli studenti e lo schema della base di dati ci obbliga a cancellare anche delle informazioni che noi vorremmo comunque tenere nel nostro database.

Anche questo è un caso di criticità di esprimibilità, cioè non ci permette di esprimere alcuni concetti da soli, senza tenere necessariamente accoppiati altri concetti, come il concetto di professore senza tenere accoppiato il concetto dell'esame di studenti.

Nel caso dell’aggiornamento, cioè di modifica di tuple, noi abbiamo un altro tipo di problema ma è comunque un’anomalia.

Supponiamo che uno studente, per esempio lo studente 341 abbia cambiato indirizzo e quindi anche CAP. Se noi vogliamo memorizzare questo nuovo indirizzo come facciamo a memorizzarlo? Dobbiamo andare in tutte le tuple della relazione Esami che riguardano lo studente 341 e modificare l'indirizzo e il CAP associati a questo studente.

Non possiamo fare questa modifica solo ad alcune tuple dello studente e ad altre no perché a quel punto avremmo un database incoerente e inconsistente, perché poi non potremmo fidarci di quale tupla contiene il dato vero dell'indirizzo dello studente. Quindi dobbiamo modificare tutte le tuple.

Una situazione simile ma anche peggiore è quella in cui un docente cambia ufficio. Se vogliamo cambiare ufficio a un docente siamo costretti, da come è fatta la base di dati, ad andare in tutte le tuple in cui quel docente ha verbalizzato qualche esame e cambiargli l'ufficio in quella tupla.

Se uno studente dà qualche decina di esami e quindi una modifica dell'indirizzo dello studente implica una modifica di 20, 30, 40 esami , quando un docente cambia d'ufficio allora può aver verbalizzato decine di migliaia di esami, quindi per cambiare un solo dato dobbiamo cambiarlo decine di migliaia di volte e questa è una criticità di efficienza nel senso che non è una criticità di esprimibilità: questa informazione la possiamo esprimere, la possiamo rappresentare, però il modo in cui è stata progettata la base di dati ci costringe a eseguire questa operazione in modo molto inefficiente.

Ora ci possiamo chiedere: “Ma a che cosa sono dovute queste anomalie? È necessario averle oppure si possono eliminare? E se vogliamo eliminarle come si può fare?”.

Possiamo avere un'idea intuitiva su come rispondere a queste domande: per esempio, se guardiamo la relazione esami, intuitivamente possiamo renderci conto che stiamo rappresentando con un’unica relazione tre concetti che sono diversi tra di loro.

Quindi se per esempio progettassimo attraverso entity-relationship un database di questo genere, sicuramente non ci verrebbe in mente di mettere un’unica entità che ha tutti questi dati, ma metteremmo tre entità o associazioni, tre concetti diversi che sono studenti, professori ed esami.

Però questo è solo un modo intuitivo di affrontare il problema e intuitivamente ci rendiamo conto anche che ci sono un mucchio di ridondanze, quindi l’indirizzo di uno studente per esempio è replicato n volte, una volta per ogni esame. La stessa cosa succede per l'ufficio del professore.

Però noi adesso affrontiamo questo problema da un punto di vista molto intuitivo, poi daremo un modo più sistematico di affrontare questo problema e più formale.

Quindi formalizziamo questa idea del fatto che ci sono concetti diversi legati tra di loro attraverso il concetto di dipendenze funzionali, poi discuteremo con la teoria di Armstrong come maneggiare questa nozione di dipendenze funzionali, quindi vedremo tecniche per riuscire a maneggiarle.

Poi descriveremo a cosa servono, e quindi anche qual è l'impatto, delle dipendenze funzionali su concetti di superchiave o di chiave. E poi useremo, non in questa lezione ma nelle prossime lezioni, l'idea di dipendenza funzionale per descrivere le forme normali e qual è il processo di normalizzazione per partire da una relazione, come questa di esami che abbiamo visto, e arrivare a una versione che rappresenta le stesse informazioni ma in modo più efficiente ed efficace minimizzando quindi le varie anomalie che abbiamo visto.

**9**

Quindi ora è il momento di affrontare le dipendenze funzionali.

Nel sistema informativo che abbiamo appena visto, che è quello della segreteria studenti e che è composto da una sola relazione esami, abbiamo intuitivamente visto che ci sono degli attributi che caratterizzano concetti diversi.

Per esempio, c'è il concetto di studente che è caratterizzato dagli attributi matricola, nome studente, indirizzo studente, CAP e così via.

Certo è che il concetto di studente non è caratterizzato dall’attributo codice professore, quindi c'è una “mescolanza” in quel senso

Il concetto di docente è caratterizzato dagli attributi codice professore, nome professore, eccetera. In particolare, questi attributi non sono “uguali” tra di loro, nel senso che non hanno lo stesso livello di importanza.

Per esempio, se io parto dal nome dello studente, posso avere studenti diversi che hanno lo stesso nome e quindi anche indirizzo studente, mentre, se io ho due tuple diverse con la stessa matricola, so che in realtà quello è lo stesso studente perché ha lo stesso numero di matricola.

Quindi io posso pensare di partire dall’attributo matricola per arrivare a determinare gli altri attributi che fanno parte di studente, cioè che caratterizzano il concetto di studente. Quindi c'è una correlazione tra alcuni attributi come nell'esempio della matricola dello studente e la descrizione dello studente in termini di altri attributi.

Detto in altro modo, se noi prendiamo due tuple della relazione esami t1 e t2 che sono relative a due esami diversi, se queste due tuple hanno lo stesso numero di matricola allora noi pensiamo che riguardino lo stesso studente, quindi non saranno uguali su tutti gli attributi, però ci aspettiamo che, dato che riguardano lo stesso studente poiché hanno lo stesso numero di matricola, allora siano uguali su tutti gli attributi che riguardano gli studenti, cioè che siano uguali sugli attributi nome studente, indirizzo studente, CAP, codice fiscale, data di nascita dello studente.

Questa considerazione cattura il fatto che esiste un vincolo, per ora implicito in questa base di dati, che noi ci aspettiamo che sia soddisfatto dalla relazione esami: cioè, prese due tuple nella relazione esami, se queste due tuple hanno lo stesso valore di matricola, allora devono avere lo stesso valore su tutti gli attributi dello studente, cioè quelli che abbiamo elencato: nome studente, eccetera.

È questa l'idea che sta alla base del concetto di dipendenza funzionale.

Ora vediamo la definizione formale di dipendenza funzionale.

Data una relazione r(A) dove r è il nome della relazione e A è l'insieme di attributi che compongono lo schema di r e dati i due sottoinsiemi X e Y degli attributi di A (quindi X e Y sono degli attributi dello schema di r), allora la dipendenza funzionale si scrive come X◊Y.

La freccia che usiamo non è la freccia che abbiamo usato per l'implicazione ma è una freccia disegnata in modo diverso, in questo caso insomma una sola linea, questo per evitare di confonderla. Questa è la rappresentazione grafica della dipendenza funzionale, quindi noi diciamo che scriviamo X◊Y e leggiamo “X determina Y”.

Attenzione a non confonderla con il concetto logico di implicazione. Per implicazione si intende un concetto che fa parte della logica e che noi abbiamo usato nelle basi di dati ma comunque è una concetto di logica, mentre questo è un concetto di dipendenza funzionale che non si studia quando studiate la logica perché è un concetto che fa parte esclusivamente delle basi di dati.

Noi usiamo una freccia graficamente diversa da quell dell’implicazione logica e leggiamo “X determina Y” dove X e Y sono insieme di attributi che fanno parte dello schema della relazione.

Il vincolo di dipendenza funzionale X determina Y - terminiamo la definizione - è soddisfatto se e solo se per ogni coppia di tuple t1 e t2 della relazione che stiamo prendendo in esame, se t1 e t2 sono uguali sugli attributi X, allora (questo è un allora che è un'implicazione logica) devono essere uguali anche sugli attributi Y.

Vediamo un esempio e prendiamo di nuovo esami. Prima di tutto, da dove nascono queste dipendenze funzionali?

Nascono dalla conoscenza che abbiamo del dominio, cioè della realtà che vogliamo modellare con il database e quindi è il progettista del database, che, sapendo come funziona il mondo che vuole modellare, scrive l'elenco delle dipendenze funzionali che esistono in quella porzione del mondo modellata.

Noi sappiamo come funzionano gli esami all'università e quindi possiamo scrivere le dipendenze funzionali che esistono tra i vari attributi che abbiamo messo nella relazione esami. Sulla slide trovate la relazione esami sia con gli attributi scritti per esteso (li ho scritti per esteso in modo che ce li abbiate sott'occhio e che sia più intuitivo che cosa vogliono dire), sia con gli attributi scritti come abbreviazione (e l’abbreviazione ci è più utile per scriverli un po’ più velocemente).

Una prima dipendenza funzionale è quella che dice che la matricola dello studente determina il nome dello studente, l'indirizzo dello studente, il CAP dello studente, il codice fiscale dello studente e la data di nascita dello studente.

Ora ci sono altre dipendenze funzionali in questa relazione esami, almeno altre cinque. Come esercizio potete provare a mettere in pausa il video e provate a scrivere quali sono le altre dipendenze funzionali che esistono in questo dominio.

Ricordate il concetto e la definizione che abbiamo visto nella slide precedente quindi vedendo l'esempio della matricola? Perché c'è questa dipendenza funzionale scritta sulla slide? Perché se noi abbiamo due tuple di esami, stiamo considerando esami, se le due tuple hanno lo stesso valore per matricola allora devono avere lo stesso valore per indirizzo studente eccetera.

Sulla base di questo potete quindi adesso mettere in pausa ed esaminare quali sono le altre dipendenze funzionali che esistono. Un suggerimento è per esempio quello di partire dal codice fiscale: che cosa determina il codice fiscale? Oppure che cosa determina il codice del professore? Ecco, oltre a quella scritta sulle slide ce ne sono altre cinque. Adesso potete mettere in pausa e poi vi svelo la soluzione.

E qua vediamo le altre dipendenze funzionali, quindi c'è una dipendenza funzionale che dice che codice fiscale determina la matricola dello studente. Quindi, se ho due tuple diverse con lo stesso codice fiscale, sicuramente devono avere la stessa matricola perché parlano dello stesso studente. Se l'indirizzo dello studente è un indirizzo completo in cui c’è sia nome della via che numero civico, allora possiamo dire che l'indirizzo dello studente determina il CAP. Poi possiamo dire che, se due tuple hanno stessa matricola e lo stesso codice del corso, allora, dato che identifica lo stesso esame, questa coppia di attributi messi insieme determina il voto, la lode e la data dell'esame, il codice del professore e il nome del professore.

Adesso questo non è l'unico modo di scrivere le dipendenze funzionali che esistono su questa relazione, ci sono anche altri che possono essere equivalenti. Poi abbiamo un'altra dipendenza funzionale che dice che il codice del professore determina il nome del professore e la qualifica, quindi, se abbiamo due tuple con lo stesso codice del professore, sicuramente devono avere lo stesso nome del professore e la stessa qualifica. Poi la qualifica del professore determina il tipo di ufficio. Possiamo immaginare che i professori più importanti abbiano gli uffici più belli e quelli meno importanti quelli meno belli. Questo può derivare dalla nostra conoscenza del dominio.

Poi possiamo immaginare che esista un vincolo per cui ogni docente è titolare di un solo corso e non di due corsi e questo è un vincolo che magari non esiste nel corso di studi in informatica però magari esiste in altri corsi di studi e quindi, se esistesse questo vincolo, allora potremmo dire che il codice del professore determina il codice del corso perché se abbiamo due tuple con lo stesso codice professore allora devono avere lo stesso codice corso seguendo questa assunzione.

Poi possiamo chiederci - anzi vi chiedo - se queste due dipendenze funzionali abbiano senso: cioè che il voto determina la lode oppure se la lode determina il voto. Provate a ragionarci, mettendo anche in pausa questo video e poi vi do una risposta: provate a immaginare cosa succede a due tuple diverse che hanno lo stesso voto oppure che hanno lo stesso valore per la lode.

Adesso vi do la risposta: se noi immaginiamo di avere due tuple diverse che hanno lo stesso voto, avranno anche lo stesso valore per la lode? Se consideriamo dui voti come ad esempio 28 si, i due 28 devono avere per forza la lode uguale a FALSE. Però questo non vale sempre perché, per esempio se noi abbiamo due tuple con il voto 30 allora alcune possono avere lode TRUE, altre lode FALSE. Quindi non è sempre vero che questa dipendenza funzionale vale per ogni coppia di tuple, perché non varrebbe per le coppie di tuple che hanno il valore 30 per il voto.

L'inverso vale? Se abbiamo due tuple diverse che hanno lo stesso valore per la lode, allora queste hanno lo stesso voto?

Questo vale per il TRUE, perché se due tuple hanno TRUE allora sicuramente il voto è 30, ma non vale se la lode è FALSE poiché il voto può essere 28 o 18 o anche 30. Quindi non vale per ogni coppia di tuple perché non vale per ogni valore degli attributi. Quindi queste sono due dipendenze funzionali che non sono valide e non le inseriamo nell'elenco delle dipendenze funzionali.

Le dipendenze, come abbiamo detto, vengono raccolte attraverso un'analisi della realtà. Quindi è una persona, un progettista analista che, conoscendo com’è fatta la realtà che stiamo modellando oppure intervistando delle persone, raccoglie un insieme di dipendenze funzionali che noi chiamiamo F nei nostri esercizi.

Nell'esempio che abbiamo visto, le dipendenze funzionali che abbiamo raccolto e che chiamiamo F sono scritte sulla slide (non le elenco una per una), però vedete che sono tra parentesi graffe perché sono un insieme. Ogni elemento dell'insieme è una dipendenza funzionale in cui troviamo l'antecedente, una freccia e il conseguente. Sono separate da un punto e virgola perché è più intuitivo dato che usiamo già la virgola per separare i singoli attributi all'interno della stessa dipendenza funzionale.

Possiamo avere diversi insiemi di dipendenze funzionali equivalenti tra di loro: possiamo avere un progettista che raccoglie un certo insieme di dipendenze funzionali, un altro progettista che raccoglie un altro insieme di dipendenze funzionali, come nell'esempio sulla slide. Ci possiamo chiedere: “Ma sono la stessa cosa? E in che senso diciamo che sono la stessa cosa?”.

Il primo progettista ha individuato tre dipendenze funzionali: la prima dice che la matricola determina nome studente, indirizzo, CAP, codice fiscale, data di nascita dello studente. La seconda dice che il codice fiscale dello studente determina la matricola. La terza dice che l'indirizzo dello studente determina il CAP dello studente.

Poi c'è l'altro progettista che vede un altro insieme di dipendenze funzionali: la prima dice che la matricola determina nome studente, indirizzo studente. La seconda dice che la matricola determina codice fiscale e data di nascita dello studente. La terza dice che il codice fiscale dello studente determina la matricola, il nome dello studente, la data di nascita, sempre dello studente. La quarta dice che l'indirizzo dello studente determina il suo CAP. Quindi ci stiamo chiedendo: “Questi insiemi sono uguali o sono diversi?”.

Da un certo punto di vista sono sicuramente diversi perché basta vedere che ci sono scritte cose diverse, quindi dal punto di vista sintattico sono diversi.

Ma anche se sono sintatticamente diversi (quindi se la forma è diversa), la sostanza è uguale? Cioè sono equivalenti? E poi che cosa vuol dire equivalenti?

È questo l'argomento che affrontiamo nelle prossime slide: studiamo che cosa significa l'equivalenza delle dipendenze funzionali e come si fa a determinare l'equivalenza delle dipendenze funzionali.

Lo facciamo prima con un approccio che usa dimostrazioni e poi lo facciamo con altri approcci che sono più comodi, che non richiedono di fare ogni volta una dimostrazione.

Iniziamo prendendo in considerazione una prima dimostrazione: noi abbiamo un primo insieme di dipendenze funzionali. Non prendiamo tutte le dipendenze funzionali che abbiamo visto due slide fa ma prendiamo solo queste tre dipendenze, quindi due insiemi, perché altrimenti sarebbe troppo scomodo e poco utile fare una dimostrazione lunga.

Quindi abbiamo un primo insieme di dipendenze funzionali che chiamiamo F’ e che contiene una singola dipendenza funzionale che chiamiamo f’1 che dice che la matricola determina il nome, l’indirizzo, il CAP, il codice fiscale e la data di nascita dello studente. Poi abbiamo un secondo insieme di dipendenze funzionali F’’ che contiene invece due dipendenze funzionali, la prima è f’’1 che dice che la matricola determina nome indirizzo e CAP dello studente e la seconda che chiamiamo f’’2 che dice che la matricola determina il codice fiscale e la data di nascita dello studente.

Questi due insiemi F’ ed F’’ sono equivalenti tra di loro? Possiamo dare una risposta a questa domanda applicando semplicemente la definizione di dipendenza funzionale e o riusciamo a dimostrare l’equivalenza oppure falliamo la dimostrazione e troviamo un contro esempio.

In questo caso dimostriamo che se vale f’’1 and f’’2 allora vale f’1 e anche viceversa, cioè se vale f’1 allora vale anche f’’1 e f’’2. Quindi è una dimostrazione “se e solo se” tra le dipendenze funzionali in “and” del secondo insieme e l'unica dipendenza funzionale del primo insieme.

Vediamo il primo verso di questa dimostrazione: quindi se vale f’’1 and f’’2 allora vale f’1.

Se scriviamo la definizione di dipendenza funzionale dell'antecedente, f’’1 and f’’2, abbiamo quelle due righe scritte sulla slide. Se quindi leggendole diciamo che per f’’1, la definizione ci dice che, per ogni coppia di tuple t1 e t2 che appartengono a r, se queste due tuple sono uguali per l'antecedente della dipendenza funzionale (quindi se sono uguali su MATR), allora devono essere uguali sulla conseguenza della dipendenza funzionale, cioè devono essere uguali su NS, IS e CAP. Deve essere vera anche la seconda dipendenza funzionale dell'insieme, cioè, per ogni coppia di tuple in R, se sono uguali su MATR allora sono uguali anche su CF e DN.

La conseguenza di questo è che, se valgono tutte e due queste regole, allora possiamo dire che, per ogni coppia di tuple, se sono uguali su MATR allora devono essere uguali su NS, IS e CAP. E se sono uguali su MATR, devono essere uguali anche su CF e DN.

Quindi applicando gli operatori logici e le proprietà degli operatori logici possiamo dire che se le tuple sono uguali su MATR allora devono essere uguali su MS, IS e CAP e devono essere uguali su CF e DN. Quindi raggruppando i vari attributi della conseguenza, dato che è equivalente, possiamo dire che per ogni coppia di tuple in R, se sono uguali su MATR allora devono essere uguali anche su NS, IS, CAP, CF e DN e, se noi guardiamo con attenzione questo ultimo passaggio, ci accorgiamo che è proprio la definizione della dipendenza funzionale f’1. Quindi abbiamo dimostrato questo verso dell'equivalenza.

L'altro verso dell'equivalenza che dobbiamo dimostrare è che se vale f’1 allora vale l’and di f’’1 e f’’2. Anche qua applichiamo la definizione, quindi scriviamo la definizione di f’1 che dice che per ogni coppia di tuple t1 e t2 in R, se sono uguali su MATR, allora devono essere uguali su NS, IS, CAP, CF e DN e quindi a maggior ragione se sono uguali su questo sono uguali anche su un sottoinsieme di questi attributi cioè devono essere uguali anche su NS, IS, CAP. Questa è proprio la definizione di f’’1.

Se noi applichiamo lo stesso ragionamento possiamo derivare anche f’’2. Basta che prendiamo nel conseguente di f’1 un altro insieme di attributi e quindi ricaviamo la definizione di f’’2.

Quindi abbiamo proprio dimostrato che da f’1 derivano f’’1 and f’’2 e abbiamo dimostrato tutti e due i versi delle implicazioni, quindi il se e solo se. Quindi abbiamo dimostrato che i due insiemi di dipendenze funzionali da cui siamo partiti sono esattamente equivalenti.

Cosa significa però equivalenza? A questo punto possiamo dire che significa che, se noi abbiamo due basi di dati in cui in una imponiamo i vincoli F’ e nell’altra imponiamo i vincoli F’’, dove questi vincoli sono insiemi di dipendenze funzionali (li chiamiamo vincoli perché “vincolano”, cioè permettono solo alcune combinazioni di tuple e di valori rispetto a quelle possibili se non avessimo il vincolo.

Quindi se noi imponiamo alternativamente vincoli F’ a una base di dati e altri vincoli F’’ alla stessa base di dati e poi proviamo a effettuare delle operazioni su queste due basi di dati che all'inizio sono uguali e poi cerchiamo di farle evolvere, allora, se i due vincoli di dipendenze funzionali sono equivalenti, allora anche l'evoluzione di questa base di dati dopo alcune operazioni deve essere esattamente uguale, quindi devono evolvere esattamente allo stesso modo altrimenti questi due insiemi non sono equivalenti.

Ora vediamo un altro esempio in cui abbiamo un insieme di vincoli F’ con f’1 che dice che da MATR determiniamo NS, IS, CF e DN; f’2 che dice che da IS determiniamo il CAP. Poi abbiamo un insieme di vincoli F’’ in cui il primo vincolo f’’1 dice che da MATR determiniamo NS, IS, CAP, CF e DN e f’’2 che dice che da IS determiniamo il CAP e ci chiediamo se le due formulazioni sono equivalenti.

Se notate, già nella dimostrazione scorsa abbiamo usato la definizione di dipendenza funzionale e per usare la definizione non abbiamo avuto bisogno di ricordarci che cosa vuol dire veramente MATR, cosa vogliono dire veramente NS e IS. Abbiamo semplicemente usato in modo simbolico la definizione. Il significato di IS, NS, CF, eccetera ci serve solo per la fase di progettazione e quindi per decidere quali sono le dipendenze funzionali che esistono nel nostro dominio. Anche in questo caso noi useremo quindi in questo modo più sintattico questi attributi basandoci esclusivamente su quali sono le dipendenze funzionali.

Per risolvere questo problema e quindi decidere se le due formulazioni sono equivalenti, dobbiamo verificare di nuovo che valga il se e solo se, cioè f’1 and f’2 se e solo se f’’1 and f’’2. Se vale questo allora vuol dire che sono equivalenti.

Se osservate i due insiemi, vedete che il secondo vincolo in tutte e due gli insiemi è uguale, cioè abbiamo sempre IS determina CAP. Questa osservazione ci permette di semplificare la dimostrazione, perché possiamo togliere in questa dimostrazione dal conseguente dell'implicazione il vincolo che è uguale nei due casi. Non possiamo toglierlo dall’antecedente altrimenti staremmo dimostrando una cosa diversa, ma dal conseguente possiamo toglierlo.

Quindi adesso dobbiamo dimostrare che f’1 and f’2 implica f’’1 e poi l'altro verso cioè che f’’1 and f’’2 implica f’1.

Dimostriamo il primo verso: scriviamo le definizioni delle due dipendenze funzionali e poi seguiamo gli altri procedimenti. Quindi possiamo, una volta scritte le due definizioni, osservare che possiamo ricavare dei sottoinsiemi dei conseguenti. Nel primo passaggio dopo la definizione abbiamo che IS viene portato fuori dalle implicazioni in una implicazione a sé stante e quindi possiamo scorporare questo t1(MATR) uguale a t2(MATR) implica t1(IS) uguale a t2(IS).

Poi possiamo per transitività dire che, se da MATR deriviamo IS e da IS deriviamo CAP, allora possiamo dire per transitività che, se due tuple sono uguali su MATR, allora devono essere uguali anche su CAP. Quindi abbiamo il terzo passaggio e poi rimettendo insieme le implicazioni in and che hanno lo stesso antecedente, stiamo ricavando proprio f’’1 che è quello che volevamo ottenere.

L'altro verso parte con la definizione di f’’1. Dovremmo aggiungere anche la definizione di f’’2, ma in realtà non ci serve perché già così possiamo vedere che, se vale f’’1 e quindi se due tuple sono uguali su MATR, allora devono essere uguali anche su NS, IS, CAP, CF e DN e a maggior ragione sono uguali su un loro sottoinsieme e quindi su NS, IS, CF e DN, che è proprio f’1 e quindi abbiamo dimostrato che anche quei due insiemi di dipendenze funzionali sono equivalenti.

**10**

Ora possiamo affrontare la teoria di Armstrong, che è correlata a quello che stavamo appena dicendo. È correlata perché ci possano chiedere: “Ma ogni volta che dobbiamo stabilire se due insiemi di dipendenze funzionali sono equivalenti dobbiamo seguire tutto questo procedimento? Cioè è indispensabile fare quello che abbiamo fatto?”.

La risposta è no: in realtà non si fa mai. Noi abbiamo fatto questo passaggio della dimostrazione per due motivi: il primo motivo è per far vedere che è possibile farlo e come si fa, il secondo motivo è per fare vedere che non è intuitivo e non è veloce. E poi, anche se ho detto due motivi, ne dico un terzo: il terzo motivo è quello di esercitarci sulla definizione di dipendenza funzionale. Quindi ci è servito come esercizio per ricordarci bene la definizione di dipendenza funzionale.

In realtà c'è un modo molto più comodo ed efficiente di procedere che è appunto quello della teoria di Armstrong.

La teoria di Armstrong ci dà delle regole che servono per maneggiare le dipendenze funzionali e quindi derivare da quelle che noi abbiamo scritto e senza ricorrere alle definizioni e alle dimostrazioni le dipendenze funzionali che sono implicite.

Partiamo dando i primi assiomi della teoria di Armstrong. Se notate nel titolo la parola “assiomi” è scritta tra virgolette perché in letteratura sono chiamati assiomi, però non è corretto, è un abuso della parola “assioma”, perché questi sono dimostrabili. Un assioma invece non è dimostrabile e quindi uno dovrebbe chiamarli “proprietà” oppure “regole inferenziali”, dato che è qualcosa che si può dimostrare partendo dalle definizioni di dipendenza funzionale. Però l'uso comune è quello di chiamarlo “assiomi”, quindi tutti sanno che non è vero che sono assiomi ma comunque tutti li chiamano ancora così, perché Armstrong quando li ha definiti li ha chiamati così e sono diventati noti con questo nome. Quindi noi continueremo a chiamarli assiomi ma ci ricorderemo all'interno di noi stessi che non è vero che sono assiomi ma sono regole.

Primo assioma: l’assioma di riflessività. Se abbiamo due insiemi di attributi Y e X e se Y è un sottoinsieme di X, allora vale la dipendenza funzionale che X determina Y.

Il secondo assioma si chiama assioma di unione e dice che se X determina Y e X determina Z allora si può scrivere che X determina YZ dove YZ è l'unione dei due insiemi di attributi Y e Z.

Poi abbiamo il terzo assioma che è quello di transitività, che dice che se X determina Y, Y determina Z, allora per appunto transitività X determina Z.

In queste tre regole, in questi tre assiomi, abbiamo il primo che si chiama di riflessività perché comprende un sottoinsieme di se stesso: dice come una dipendenza funzionale “parla” di un sottoinsieme di se stesso. Il secondo si chiama di unione perché permette di unire i conseguenti di due dipendenze funzionali e il terzo si chiama di transitività per motivi ovvi.

Vediamo nel dettaglio i vari assiomi partendo dall'assioma di riflessività che dice che se prendo un sottoinsieme Y di X allora X determina Y, ad esempio io parto da un insieme X di attributi CF,Co,Q e prendo un suo sottoinsieme Co,Q, allora c'è la dipendenza funzionale CF,Co,Q determina Co e Q. Questo assioma di riflessività è un assioma particolare diverso dagli altri perché non parte da una dipendenza funzionale, o da due dipendenze funzionali, per produrne un'altra che deriva da quelle, ma invece crea, si può dire, delle dipendenze funzionali nuove da zero, semplicemente da un insieme di attributi.

Un esempio di applicazione ce l’abbiamo con i due insieme di dipendenze funzionali F’ ed F’’.

F’ contiene una sola dipendenza funzionale f’1: MATR determina NS,IS,CAP,CF,DN. F’’ contiene invece due dipendenze funzionali f’’1 che è uguale a f’1 e f’’2 che è CF,Co,Q determina Co,Q. Noi possiamo sapere che questi due insiemi di dipendenze funzionali sono equivalenti tra di loro perché possiamo partire da F’, applicare l’assioma di riflessività creando la nuova dipendenza funzionale f’’2 e quindi, dato che abbiamo ricavato F’’ da F’, sappiamo appunto che sono equivalenti.

Un altro assioma è l'assioma di unione che dice che se abbiamo le due dipendenze funzionali X determina Y e X determina Z, allora vale la dipendenza funzionale X determina YZ. Ad esempio abbiamo la dipendenza funzionale F’ che contiene solo f’1: MATR determina NS,IS,CAP,CF,DN. Poi abbiamo F’’ che contiene le due dipendenze funzionali in cui MATR determina NS, IS e CAP e l'altra dipendenza funzionale f’’2 che dice che MATR determina CF,DN. Allora noi possiamo applicare l'assioma di unione. Partendo da F’’ uniamo le due dipendenze funzionali e ricaviamo F’, quindi l'unica dipendenza funzionale è di F’, applicando appunto l'assioma di unione e stabiliamo quindi che i due insiemi F’ ed F’’ sono equivalenti tra di loro, senza applicare la definizione di dipendenza funzionale.

Il terzo assioma è l'assioma di transitività che dice che, se abbiamo le due dipendenze funzionali X determina Y e Y determina Z, allora vale la dipendenza funzionale X determina Z. Ad esempio abbiamo F’ con f’1: MATR determina NS,IS,CF,DN ed f’2 IS determina CAP. Poi abbiamo F’’ con MATR che determina NS,IS,CAP,CF,DN quindi è simile a f’1, ma nel conseguente ha in più anche CAP e poi abbiamo f’’2 che dice che IS determina CAP, quindi è uguale a f’2. Possiamo applicare l'assioma di transitività, cioè noi sappiamo che da f’1 ricaviamo IS, dato che da IS ricaviamo CAP, allora sappiamo che da MATR possiamo ricavare non solo IS, ma anche CAP e quindi sappiamo che i due insiemi sono equivalenti tra di loro.

Si può dimostrare che la teoria di Armstrong è una teoria corretta e completa.

Per teoria corretta intendiamo questo: consideriamo corretto ciò che si può ricavare tramite le definizioni di dipendenza funzionale e quindi come correttezza della teoria di Armstrong intendiamo che se noi deduciamo qualcosa applicando gli assiomi di Armstrong, allora la stessa cosa si può ricavare anche usando la definizione di dipendenza funzionale. Quindi non ricaviamo qualcosa che non è derivabile anche applicando la definizione di dipendenza funzionale.

Non vediamo la dimostrazione, come non vediamo neanche diverse altre dimostrazioni in questo corso, ma comunque l'idea della dimostrazione è che si considera la definizione di dipendenza funzionale e si vede che ogni assioma è corretto rispetto alla definizione e poi per induzione si dimostra che gli assiomi generano dipendenze funzionali che soddisfano la definizione di dipendenza funzionale.

Poi abbiamo detto che la teoria di Armstrong non è solo corretta ma è anche completa. Per completa intendiamo qualcosa simile all'inverso della correttezza, cioè per correttezza intendiamo che se qualcosa è ricavabile dalla teoria di Armstrong allora è ricavabile anche tramite le definizioni di dipendenza funzionale, per completezza intendiamo che se qualcosa è ricavabile tramite le definizioni dipendenze funzionali allora è ricavabile anche tramite gli assiomi di Armstrong e quindi si ha la completezza, cioè gli assiomi di Armstrong derivano tutto ciò che si può derivare anche attraverso le definizioni di dipendenza funzionale.

La teoria di Armstrong è corretta e completa e la conseguenza di questa proprietà è che, se noi usiamo gli assiomi di Armstrong, otteniamo tutto e solo ciò che possiamo ottenere tramite la definizione di dipendenza funzionale. Questo ci garantisce che possiamo “archiviare” quelle dimostrazioni lunghe e poco intuitive in cui usiamo la definizione di dipendenza funzionale e possiamo limitarci a usare gli assiomi di Armstrong.

Non abbiamo in Armstrong solo degli assiomi, ma abbiamo anche altre regole aggiuntive. Queste regole non sono regole indispensabili o necessarie nel senso che non ci permettono di ricavare qualcosa in più che non sia già ricavabile usando anche gli assiomi, ma sono regole che sono utili perché ci permettono di ricavare ciò che è ricavabile tramite anche gli assiomi però in modo più comodo cioè usando meno passaggi e quindi possono risultare intuitive e utili da utilizzare.

Le regole che vediamo nelle prossime slide sono le regole dell'espansione, della decomposizione, della pseudotransitività e del prodotto. Queste si chiamano ufficialmente non assiomi ma regole quindi non si usa la parola assioma, perché queste sono dimostrabili ricavandole dagli assiomi di Armstrong. Quindi per queste regole non è necessario - è possibile ma non è necessario - usare la definizione di dipendenza funzionale, ma si possono ricavare semplicemente usando gli assiomi di Armstrong e questo fa anche capire perché Armstrong ha definito le altre tre regole assiomi e queste no. Perché ha stabilito che quelle tre regole che abbiamo visto prima sono quelle necessarie per ricavare anche queste regole aggiuntive, anche se non sono dei veri e propri assiomi.

La regola di espansione dice che se partiamo da una dipendenza funzionale X determina Y e abbiamo un insieme di attributi W allora possiamo aggiungere questo insieme gli attributi W sia all'antecedente che al conseguente della dipendenza funzionale e ottenere WX determina WY. Ad esempio, prendendo la dipendenza funzionale Q determina TU quindi la qualifica del professore determina il tipo di ufficio e l’attributo codice fiscale, posso ottenere codice fiscale, Q determina codice fiscale, TU. Se volete come esercizio potete applicare la definizione di dipendenza funzionale per vedere che questo esempio è corretto. Si può dimostrare che questa regola funziona ed è corretta applicando gli assiomi, non vado nei dettagli di questa dimostrazione, comunque potete leggerla sulla slide: applichiamo l'assioma di riflessività due volte e poi l'assioma di transitività e l’assioma di unione per dimostrare che vale questa regola.

La regola di decomposizione dice che se partiamo da una dipendenza funzionale X determina Y e Z, allora possiamo decomporla in due regole X determina Y e X determina Z. Ad esempio, partendo da matricola che determina codice fiscale e data di nascita, ricaviamo le due regole o meglio i due vincoli di dipendenza funzionale matricola determina CF e matricola determina DN.

Fate attenzione perché questa regola di decomposizione permette di decomporre solo il conseguente di una dipendenza funzionale non vale assolutamente come decomposizione su un antecedente, cioè su ciò che sta a sinistra della freccia. Non è affatto vero che se Y determina Z allora si può ricavare X determina Z o Y determina Z.

Intuitivamente si può pensare che in effetti, se noi decomponessimo l’antecedente, avremmo due regole che hanno un antecedente più debole e meno restrittivo e quindi non è detto che valga, mentre possiamo tranquillamente decomporre il conseguente attraverso questa regola di decomposizione, perché appunto è antecedente e continua ad essere uguale e quindi la restrizione che eventualmente applica è uguale in tutti e tre i vincoli.

Si può dimostrare questa regola applicando gli assiomi di riflessività di transitività poi di nuovo di riflessività e di nuovo di transitività, però anche per questo non andiamo nei dettagli.

Abbiamo la regola di pseudo transitività che dice che, se abbiamo 2 dipendenze funzionali X determina Y e WY determina Y, non possiamo applicare direttamente l’assioma di transitività, ma comunque possiamo dire che WX determina Z, quindi aggiungiamo questo W all’antecedente di X.

Ad esempio se abbiamo che codice fiscale determina matricola, corso e matricola determinano il codice del professore, possiamo dire che vale che codice del corso e codice fiscale determinano il codice del professore.

La dimostrazione della regola di pseudo transitività è molto breve perché possiamo applicare il teorema di espansione cioè la regola di espansione aggiungendo W sia all'antecedente che al conseguente di X determina Y e a questo punto possiamo applicare l'assioma di transitività.

Abbiamo anche la regola del prodotto che dice che partendo da X determina Y e W determina Z allora vale XW determina YZ, cioè combiniamo i due antecedenti delle due regole e i due conseguenti delle due regole.

Ad esempio, se il codice fiscale determina la matricola e il codice del professore determina il nome del professore allora possiamo dire che il codice fiscale e il codice del professore determinano la matricola e il nome del professore. La dimostrazione della regola del prodotto applica i vari assiomi di riflessività e transitività e poi l'assioma dell'unione.

Come esercizio potete provare a porre come assioma la proprietà dell'espansione e dimostrare all'indietro la regola dell'unione.

**11**

Ora passiamo a parlare della chiusura di un insieme di dipendenze funzionali.

Abbiamo iniziato con un approccio al problema volto a determinare l'equivalenza di due insiemi di dipendenze funzionali dicendo che questi due insiemi sono equivalenti se fanno evolvere una base di dati nello stesso modo.

Abbiamo poi dimostrato l’equivalenza di due insiemi di dipendenze funzionali usando la definizione di dipendenza funzionale e poi dimostrando formalmente se sono equivalenti o no e adesso abbiamo introdotto le regole di Armstrong che ci permettono di evitare di utilizzare direttamente la definizione di dipendenza funzionale.

Adesso quindi possiamo introdurre un altro approccio al problema dell'equivalenza, cioè determiniamo l'equivalenza tra due insiemi di dipendenze funzionali applicando le regole di Armstrong , vedendo che cosa riusciamo a derivare attraverso le regole di Armstrong, partendo da queste due insiemi di dipendenze funzionali e vedendo se ciò che riusciamo a derivare da uno e dall'altra è uguale e quindi se i due insiemi derivati di dipendenze funzionali sono uguali tra di loro.

Questa intuizione, un'idea informale, adesso la formalizziamo e la precisiamo per capirla meglio e definirla in modo più preciso introducendo l'idea di chiusura e cioè la definizione di chiusura di un insieme di dipendenze funzionali.

Definiamo la chiusura di un insieme di dipendenze funzionali F come un insieme F+, quindi scriviamo il più in alto, in apice e quello rappresenta il fatto che stiamo determinando la chiusura di F.

Quindi F+ che cos'è? È la chiusura di F e cioè l'insieme di tutte le dipendenze funzionali derivabili da F.

Come le deriviamo queste dipendenze funzionali? Possiamo derivarle come vogliamo; il modo più comodo di farlo è applicare le regole di Armstrong. Ma quante regole di Armstrong applichiamo? Applichiamo tutte le regole di Armstrong che è possibile applicare, quindi continuiamo ad applicarle continuamente, anche più volte le stesse regole, in tutti i modi possibili e immaginabili, fino a che non raggiungiamo una situazione in cui non è più possibile applicarne nessuna. A quel punto abbiamo terminato il calcolo della chiusura dell'insieme di dipendenze funzionali F.

Ci possiamo chiedere se questo algoritmo termina, questo algoritmo che applica iterativamente tutte le possibili regole di Armstrong e la risposta è sì, perché l'insieme di dipendenze funzionali F è finito perché partiamo da un insieme finito di dipendenze funzionali.

Perché è finito? Perché l'insieme di attributi da cui partiamo è necessariamente finito, perché altrimenti non siamo in un modello relazionale e un insieme di attributi finito ha un numero finito di modi di essere raggruppato nei vincoli di dipendenza funzionale.

Questa non è una dimostrazione formale ma comunque dovrebbe dare l'idea del perché sappiamo che sicuramente l'insieme F+ è finito.

La nuova definizione di equivalenza è quindi che F è equivalente a G, dove F e G sono due insiemi di dipendenze funzionali, se e solo se F+ è uguale a G+.

Detto in parole povere abbiamo ricondotto il problema di determinare l'equivalenza tra F e G ad un problema sintattico, non più semantico, quindi non dobbiamo andare a vedere se veramente due database evolvono nello stesso modo, ma basta che calcoliamo la chiusura di F e di G, quindi calcoliamo F+ e G+ e vediamo se contengono gli stessi identici vincoli di dipendenza funzionale.

Se lo facciamo e sono diversi, allora sappiamo che F non è equivalente G. Se controlliamo che F+ e G+ siano esattamente uguali e vediamo che sono esattamente uguali, allora possiamo dire con sicurezza che F è equivalente a G.

Il problema è che è vero che F+ e G+ sono finiti, non sono infiniti, quindi prima o poi termineremo il calcolo della chiusura di F e della chiusura di G, però non è efficiente calcolarle perché la complessità è almeno esponenziale nel numero di dipendenze funzionali da cui partiamo cioè nella cardinalità di F e nella cardinalità di G.

Questo perché intuitivamente si può vedere che se noi partiamo da un insieme di dipendenze funzionali F dove abbiamo n dipendenze funzionali fatte così: A determina B1, A determina B2 fino ad A determina Bn, quindi appunto la cardinalità di F è n.

Se noi ci limitiamo ad applicare due regole, quella di decomposizione e quella di unione allora - eventualmente potete andare a ripassare cosa dicono queste due regole - sappiamo che possiamo ricavare tutte le regole A determina Z in cui Z fa parte dell'insieme delle parti di B1, B2, Bn, quindi dell'unione di B1, B2, Bn.

Dato che stiamo applicando solo queste due regole e otteniamo qualcosa che è l'insieme delle parti e l'insieme delle parti ha cardinalità esponenziale, infatti è due alla n meno uno (l'insieme delle parti di un insieme di n elementi è composto da due alla n meno uno insiemi) allora sappiamo che va sicuramente peggio, perché qua ci siamo limitati ad applicare decomposizione e unione, ma potremmo applicare ancora altre regole e otterremmo ancora più vincoli, quindi sappiamo che la chiusura di F come minimo contiene due alla n meno uno elementi. Quindi il calcolo di un numero esponenziale di risultati è esponenziale di per sé, quindi è molto costoso calcolare la chiusura di F e la chiusura di G.

Ora siamo arrivati al punto in cui abbiamo due definizioni diverse di equivalenza tra due insiemi di dipendenze funzionali. Prima di commentare questa slide faccio una parentesi e dico che, in realtà, quello che ci interessa veramente non è proprio esclusivamente determinare l'equivalenza tra due insiemi di dipendenze funzionali, ma tutto l'armamentario che adesso stiamo sviluppando e che stiamo capendo ci servirà comunque dopo quando parleremo di forme normali e di normalizzazione. È un percorso a tappe e questa è una delle tappe che dobbiamo affrontare.

Quindi, chiusa parentesi, abbiamo detto che abbiamo due definizioni di equivalenza: abbiamo una prima definizione, che è un primo approccio che abbiamo usato, che dice che F è equivalente a G se e solo se è possibile derivare G da F o viceversa e questo approccio l'abbiamo seguito già all'inizio quando abbiamo usato la definizione di dipendenza funzionale, direttamente la definizione, poi abbiamo fatto la dimostrazione.

Una seconda volta abbiamo seguito questo approccio quando abbiamo applicato gli assiomi di Armstrong e abbiamo raggiunto degli insiemi dipendenze funzionali che erano uguali. Adesso abbiamo un secondo approccio, che abbiamo introdotto ora: diciamo che F è equivalente a G se e solo se F e G hanno la stessa chiusura. Quindi sono due approcci che sono diversi tra di loro.

Possiamo riformulare questi due approcci, quindi riscrivere questi due approcci modo più formale e lo facciamo in questa slide. In questa slide l’equivalenza è questo simbolo di “uguale” con un trattino in più.

Diciamo che F è equivalente a G, nel punto 1, se e solo se G è deducibile da F, con questo simbolo logico di deduzione, quindi se e solo se G è deducibile da F e F è deducibile da G.

Cosa vuol dire che è deducibile? G è deducibile da F quando, se noi prendiamo una qualunque dipendenza g di G, quindi un elemento di G, allora questa dipendenza g è deducibile da F.

Questo vuol dire che G è deducibile da F. Questo vuol dire che è deducibile se è possibile dedurre ogni dipendenza di G.

Il contrario, ovviamente, cioè F deducibile da G, vuol dire che, se noi prendiamo una qualunque dipendenza f di F, allora possiamo dedurre questa dipendenza f partendo da G.

Il secondo approccio non usa la deduzione ma usa la chiusura delle dipendenze funzionali. Quindi con il secondo approccio diciamo che F è equivalente a G se e solo se la chiusura di F è uguale alla chiusura di G. Quindi questa è un’uguaglianza con solo due trattini orizzontali, quindi una nozione sintattica non semantica.

Per vedere se sono uguali basta vedere che contengono esattamente gli stessi elementi fatti nello stesso modo. Adesso ci possiamo chiedere: “Ma queste due definizioni di equivalenza tra dipendenze funzionali dicono la stessa cosa oppure sono due definizioni di equivalenze diverse, quindi due concetti diversi di equivalenza?”.

Possiamo dimostrare che in realtà sono lo stesso concetto di equivalenza perché possiamo dimostrare che G è deducibile da F e F deducibile da G se solo se la chiusura di F è uguale alla chiusura di G, quindi vuol dire che dicono esattamente la stessa cosa. La dimostrazione si può fare, anzi si deve fare, su entrambi i versi della dimostrazione perché è un se e solo se.

Su questa slide riportiamo per brevità un solo verso della dimostrazione cioè dimostriamo che se G è deducibile da F e F è deducibile da G, allora anche G+ è uguale F+.

Cosa vuol dire che F+ è uguale a G+? Dato che F+ è un insieme di elementi, cioè un insieme di dipendenze funzionali e G+ è un insieme di dipendenze funzionali, per dire che i due insiemi sono uguali possiamo dire che F+ è un sottoinsieme di G+ e anche G+ è un sottoinsieme di F+. Se sono vere tutte e due queste condizioni allora F+ è esattamente uguale a G+.

Dimostriamo quindi il primo sottoinsieme, quindi dimostriamo che se G è deducibile da F e F è deducibile da G allora G+ è un sottoinsieme di F+.

La dimostrazione è molto semplice perché noi prendiamo una qualunque dipendenza g+ di G+, un arbitrario elemento di G+.

Per definizione di chiusura di un insieme di dipendenze funzionali sappiamo che G+ è derivabile da G perché per ottenere la chiusura non facciamo altro che applicare regole di Armstrong e se applichiamo le regole di Armstrong vuol dire che possiamo derivare quella dipendenza funzionale e quindi possiamo dire che G+ è derivabile da G.

Per la proprietà transitiva, se noi sappiamo che G è deducibile da F e che g+ è deducibile da G, allora possiamo concludere che g+ è deducibile da F, ma se g+ è deducibile da F, vuol dire che g+ appartiene alla chiusura, perché appunto per dedurre dobbiamo applicare le regole di Armstrong.

Quindi abbiamo dimostrato che g+, se appartiene a G+, allora appartiene anche a F+. Se appartiene a tutti e due vuol dire che G+ è un sottoinsieme di F+. Quindi appunto abbiamo dimostrato la prima relazione di sottoinsieme.

Per la seconda relazione di sottoinsieme facciamo un ragionamento molto simile e quindi possiamo concludere (io non lo faccio per passo per passo ma comunque possiamo concludere) quindi che F+ è un sottoinsieme di F+ quindi vuol dire che F+ è esattamente uguale a G+.

Notate che nelle due dimostrazioni abbiamo scelto un g e un f arbitrari, quindi vuol dire che possiamo ripetere la stessa dimostrazione per ogni elemento di G+ ed F+ e quindi insomma non siamo legati alla particolare scelta che abbiamo fatto. Quindi abbiamo dimostrato un verso dell'implicazione, l'altro verso non lo dimostriamo ma potete trovarlo sui testi.

**12**

Facciamo un passo ulteriore e parliamo di chiusura di un insieme di attributi.

Per risolvere il problema dell'equivalenza tra due insiemi di dipendenze funzionali abbiamo discusso due approcci diversi: un primo approccio in cui abbiamo usato la definizione e le dimostrazioni per determinare l'equivalenza e un secondo approccio in cui abbiamo usato le regole di Armstrong per determinare la chiusura di un insieme di dipendenze funzionali. Abbiamo visto che questi due metodi sono equivalenti, nel senso che danno lo stesso risultato, ma la cattiva notizia è che sono tutti e due troppo costosi da applicare e hanno una complessità troppo alta per riuscire a essere di qualsiasi utilità pratica.

Quindi sembra che apparentemente abbiamo perso del tempo ma in effetti adesso vediamo un altro concetto che è correlato alla chiusura di un insieme di dipendenze funzionali che invece è molto più efficiente.

L'idea che applichiamo è quella di non calcolare la chiusura di un insieme di dipendenze ma di calcolare la chiusura solo di un insieme di attributi.

Nelle prossime slide questo processo diventerà più chiaro.

Definiamo la chiusura di un insieme di attributi: partiamo da un insieme di attributi R che corrisponde a tutti gli attributi di uno schema di un database e prendiamo un sottoinsieme X di questi attributi.

Associato a un database abbiamo un insieme di dipendenze funzionali F.

Definiamo la chiusura di X quindi dell'insieme di attributi X che è un sottoinsieme di R rispetto a F e la scriviamo come X e in pedice scriviamo F cioè quell'insieme di dipendenze funzionali che usiamo per calcolare la chiusura e in apice un più, che rappresenta il fatto che stiamo calcolando una chiusura.

Se non c'è ambiguità su quale insieme di dipendenze funzionali stiamo usando possiamo anche non scrivere F.

Quindi la chiusura dell’insieme di attributi X è definita come l'insieme degli attributi A tale che la dipendenza funzionale X determina A, appartiene alla chiusura di F.

Notate due fatti o tre, cioè che noi prendiamo non solo le dipendenze funzionali in F ma prendiamo le dipendenze funzionali in F+, quindi nella chiusura di F quindi stiamo riutilizzando i concetti che abbiamo sviluppato prima, quindi quello della chiusura di un insieme di dipendenze funzionali.

Poi notate che l'antecedente della dipendenza funzionale deve essere X, cioè l'insieme di attributi di cui stiamo calcolando la chiusura e di questo noi prendiamo tutti i conseguenti, quindi la chiusura di un insieme di attributi è un altro insieme di attributi (quindi non è più, come prima, un insieme di dipendenze funzionali, ma in questo caso un insieme gli attributi) che sono i conseguenti di tutte le dipendenze funzionali nella chiusura di F che sono applicabili partendo dall’insieme di attributi X.

Vediamo un esempio: vogliamo calcolare la chiusura dell'insieme gli attributi che contiene un singolo attributo MATR rispetto alle dipendenze funzionali F che vedete sulla slide.

Quindi in questa slide riporto in modo più compatto, nel primo punto della slide, ciò che vogliamo calcolare e per il primo passo dobbiamo notare che, per l'assioma di riflessività di Armstrong, sicuramente MATR appartiene alla chiusura dell'attributo MATR alla chiusura di se stesso perché per l’assioma di riflessività MATR è un sottoinsieme di se stesso, dell'insieme di attributi che contiene solo se stesso, quindi la dipendenza funzionale MATR determina MATR appartiene sicuramente alla chiusura di F, quindi MATR appartiene alla chiusura X+.

Questa è una regola generale nel senso che ogni volta che noi calcoliamo la chiusura di un insieme di attributi, per prima cosa dobbiamo ricordarci di mettere quell’insieme di attributi nella chiusura perché appartengono sempre sicuramente alla chiusura. Se non lo facciamo subito rischiamo di dimenticare.

Poi possiamo considerare che la prima dipendenza funzionale di F cioè MATR determina NS,IS,CAP,CF e DN è applicabile in questo caso perché l'antecedente è MATR di questa regola quindi possiamo applicare questa dipendenza funzionale e possiamo aggiungere oltre a MATR anche il conseguente di questa dipendenza funzionale all'insieme degli attributi nella chiusura. Quindi aggiungiamo alla chiusura anche NS,IS,CAP,CF e DN.

Se proviamo ad andare avanti ci possiamo accorgere che non c'è altro che possiamo aggiungere quindi la chiusura di X uguale MATR è composta da MATR, per riflessività e poi da NS,IS,CAP,CF e DN.

Adesso guardiamo un algoritmo per calcolare direttamente la chiusura di un insieme di attributi quindi quello che abbiamo visto nella slide precedente era solo un esempio per capire meglio, ma per calcolare effettivamente la chiusura di un insieme di attributi useremo poi questo algoritmo che adesso introduciamo.

L'algoritmo lo vediamo da due punti di vista: ad alto livello in modo che intuitivamente possa darvi un'idea di come funziona e poi l'algoritmo effettivo che gestisce i vari dettagli che questo alto livello ignora.

Quindi ad alto livello noi partiamo da X cioè dall’insieme di attributi di cui vogliamo calcolare la chiusura e poi facciamo un ciclo in cui consideriamo tutte le dipendenze funzionali in F che sono “applicabili”.

Applicabili vuol dire che l'antecedente è compreso tra gli attributi che abbiamo già ricavato nei cicli precedenti o nell’inizializzazione e finché ci sono delle dipendenze funzionali applicabili allora applichiamo queste dipendenze funzionali, cioè ricaviamo i conseguenti, cioè nuovi attributi che possiamo ricavare e andiamo avanti in questo ciclo finché non terminiamo.

Più nel dettaglio vediamo che ci sono questi passi in cui c'è la fase di inizializzazione in cui C che è un insieme di attributi lo inizializziamo con X, X è l’insieme di attributi di cui vogliamo calcolare la chiusura e F’ è un insieme di dipendenze funzionali che inizializziamo con F.

A questo punto iniziamo con il ciclo, quindi per ogni dipendenza funzionale Y determina Z che appartiene a F’, che possiamo applicare cioè per cui Y è un sottoinsieme di C.

“Possiamo applicare” vuol dire che l’antecedente di questa dipendenza funzionale è composto da attributi che noi abbiamo già ricavato o nell'inizializzazione o dai cicli precedenti e abbiamo messo in C.

Quindi per ogni dipendenza funzionale Y che determina Z applicabile, allora noi la applichiamo, cioè prendiamo il conseguente di questa dipendenza funzionale che è Z, che è un insieme di attributi, aggiungiamo questo attributo in Z a C, agli attributi che abbiamo già ricavato in precedenza con quel passo di unione, poi dato che questa dipendenza funzionale l'abbiamo già applicata, per evitare di finire in un gruppo infinito, dobbiamo ricordarci che non dobbiamo applicarla più e quindi la sottraiamo e la eliminiamo dall'insieme F’ delle dipendenze funzionali.

Quando terminiamo il ciclo restituiamo C che è l'insieme degli attributi che abbiamo ricavato nei passi precedenti.

Ora vediamo un esempio di applicazione di questo algoritmo in cui vogliamo calcolare la chiusura di CP cioè un insieme di attributi che contiene un solo attributo che è CP, cioè il codice del professore.

Sulla slide vedete in alto a sinistra l'insieme F’, quindi delle dipendenze funzionali che dobbiamo ancora applicare e che è inizializzato al primo passo come F, cioè tutte le dipendenze funzionali che abbiamo a disposizione.

In basso a destra vedete l'algoritmo in modo da averlo sempre sott'occhio per capire dove siamo.

Vogliamo calcolare la chiusura di CP, quindi inizializziamo X uguale a CP, inizializziamo C uguale a CP ed F’ uguale a F (i primi due passi dell'algoritmo).

Poi cerchiamo una dipendenza funzionale in F’ che ha come antecedente CP quindi una dipendenza funzionale che è applicabile e vediamo che esiste questa dipendenza ed è CP determina NP,Q.

Il conseguente NP e Q lo aggiungiamo a C dove si trova già CP che deve appartenere a C per l'assioma di riflessività.

Contestualmente eliminiamo da F’ anche la dipendenza funzionale che abbiamo appena utilizzato: CP determina NP,Q.

Ora facciamo un altro passo del ciclo e cerchiamo una dipendenza funzionale applicabile cioè il cui antecedente è compreso in CP,NP e Q. Esiste? Sì, esiste. Per esempio, possiamo applicare Q determina TU perché appunto l'antecedente Q è compreso in C, quindi aggiungiamo il conseguente TU all’insieme C ed eliminiamo la dipendenza funzionale da F’.

Poi cerchiamo un’altra dipendenza funzionale il cui antecedente sta in CP,NP,Q e TU. Questa esiste e possiamo applicare per esempio CP determina CO, il conseguente della dipendenza funzionale è CO, quindi aggiungiamo l'attributo CO a C ed eliminiamo la dipendenza funzionale che abbiamo appena applicato.

Poi cerchiamo un’altra dipendenza funzionale tra quelle rimanenti in F’ che sia applicabile, cioè il cui antecedente sia in C. Non esiste perché le dipendenze funzionali rimanenti hanno come antecedente MATR, CF IS e CO . Nessuna di queste è applicabile, in particolare MATR,CO determina Vo, DE,CP,NP e non è applicabile perché è vero che abbiamo CO ma tutti gli antecedenti devono essere compresi in C e in C non abbiamo la matricola quindi non possiamo applicare questa dipendenza funzionale e quindi non c'è nessuna dipendenza funzionale applicabile.

Siamo arrivati al termine del ciclo e usciamo dall'algoritmo. La chiusura di CP rispetto a F e quindi composta da CP stesso, NP,Q.TU, CO. Quindi abbiamo terminato il ciclo e abbiamo ricavato qual è la chiusura di questo insieme di attributi CP rispetto a F.

Ora ci potremmo chiedere: “Ma qual è il rapporto, se c'è un rapporto, tra la definizione di chiusura di un insieme di attributi e l’algoritmo che abbiamo appena visto?”.

Be’ si può dimostrare che questi due modi di procedere ricavano esattamente la stessa cosa, cioè si può dimostrare che l'algoritmo è corretto e completo.

Corretto vuol dire che ciò che ottiene è corretto, cioè è ciò che ricaveremmo anche applicando la definizione di chiusura di un insieme di attributi.

Intuitivamente quindi si può dire che dato che C è l'insieme degli attributi che l'algoritmo ricava alla fine della sua esecuzione esso è un sottoinsieme di XF+.

Per dimostrarlo si può ragionare sul fatto che l'algoritmo aggiunge a C passo per passo degli attributi che sono ricavabili anche in F+ applicando gli assiomi di Armstrong di riflessività, transitività e unione. Poi si può dire anche che l'algoritmo è completo cioè che tutti gli attributi che io ricavo tramite la definizione sono anche ricavati dall'algoritmo e cioè che XF+ è un sottoinsieme di C.

Intuitivamente si può pensare, per ragionare su questo, che, presa una qualunque dipendenza funzionale X determina Ai che appartiene alla chiusura di F, prima o poi il conseguente di questa dipendenza funzionale, cioè Ai, viene aggiunto a C dall'algoritmo perché con gli assiomi di riflessività, transitività e unione, che sono quelli che realizzano in effetti l’algoritmo, si può dimostrare che si ricava tutta la chiusura di F.

Questa è solo un'idea ma non parliamo della dimostrazione comunque se riusciamo a dimostrare questi due contenimenti sappiamo che C, che è ciò che ricava l'algoritmo è esattamente uguale a XF+.

Quindi adesso sappiamo che questo algoritmo ricava esattamente ciò di cui abbiamo bisogno, cioè il calcolo della chiusura di un insieme di attributi X.

La complessità dell'algoritmo: questo è un aspetto importante perché se la complessità di questo algoritmo fosse esponenziale, non avremmo fatto nessun passo avanti rispetto alla situazione da cui siamo partiti, perché la chiusura di un insieme di dipendenze funzionali era esponenziale, quindi speriamo di aver fatto meglio con la chiusura di un insieme di attributi.

La buona notizia è che questa complessità è polinomiale rispetto al numero di dipendenze funzionali e rispetto al numero di attributi.

Perché è polinomiale? Perché il ciclo contiene delle operazioni di unione e differenza tra insiemi che si possono risolvere in tempo polinomiale nel numero degli attributi.

Ogni dipendenza funzionale viene presa in considerazione una sola volta perché prendiamo in considerazione ogni dipendenza funzionale non nella chiusura dell'insieme delle dipendenze funzionali, che sarebbe un numero esponenziale di dipendenze funzionali, ma nell'insieme originale di dipendenze funzionali F.

Adesso non ci interessa dire esattamente quale sia la complessità dell'algoritmo ma è sufficiente dire che sia polinomiale per sapere che già facciamo molto meglio rispetto all'algoritmo che calcola la chiusura di un insieme di dipendenze funzionali.

Ci possiamo chiedere quale sia il rapporto tra le due chiusure di cui abbiamo parlato finora, cioè abbiamo iniziato parlando di chiusura di un insieme di dipendenze funzionali, adesso abbiamo definito la chiusura di un insieme di attributi e possiamo chiederci ma cosa c'entrano l'uno con l'altro?

In effetti una proprietà li lega infatti si può dimostrare che la dipendenza funzionale X determina Y appartiene alla chiusura di F se solo se Y è un sottoinsieme della chiusura di X.

Quindi abbiamo un modo tramite questa proprietà di calcolare la chiusura di un insieme di attributi e riuscire a determinare se una certa dipendenza funzionale appartiene alla chiusura di un insieme di dipendenze funzionali. Poi capiremo perché è utile avere questa proprietà.

La dimostrazione di questa proprietà, dato che c’è un se e solo se, è una dimostrazione nei due versi, una doppia implicazione.

Teniamo a mente qual è la definizione di chiusura di un insieme di attributi che è riportata in alto sulla slide, cioè sono gli attributi del conseguente di tutte le dipendenze funzionali che appartengono alla chiusura di F.

Da un verso l’implicazione dice che, se la dipendenza funzionale X determina Y appartiene alla chiusura, allora dobbiamo dimostrare che Y è un sottoinsieme della chiusura di X.

Supponiamo che Y, cioè il conseguente della dipendenza funzionale sia composto da n attributi A1…An.

Se applichiamo la regola di Armstrong della decomposizione allora abbiamo che alla chiusura di F appartengono anche X determina A1 fino a X determina An.

Tutte queste n dipendenze funzionali appartengono alla chiusura di F.

Per la definizione della chiusura di X rispetto F abbiamo ovviamente che A1,A2,…An appartengono alla chiusura di X e quindi abbiamo dimostrato che Y è un sottoinsieme della chiusura di X.

L'altro verso della implicazione è: se Y è un sottoinsieme di Xf+ allora X determina Y appartiene a F+.

Quindi partiamo sapendo che per ipotesi Y è un sottoinsieme della chiusura di X.

Per definizione di chiusura di un insieme di attributi abbiamo che necessariamente devono appartenere all’insieme della chiusura delle dipendenze funzionali F tutte le dipendenze funzionali che ci hanno permesso di mettere Y dentro la chiusura di X, cioè tutte le dipendenze X determina A1… fino a X determina An.

Stiamo continuando ad assumere che Y sia composto da questi n attributi, il che è senza perdita di generalità.

A questo punto possiamo applicare la regola dell'unione, per cui sappiamo che dato che X◊A1 fino a X◊An appartengono alla chiusura di F allora anche X determina Y appartiene alla chiusura di F.

Questo chiude la dimostrazione.

Applichiamo in un esempio questa proprietà. Noi abbiamo un insieme di dipendenze funzionali F che vedete sulla slide. È lo stesso insieme che abbiamo già visto prima e vogliamo verificare se la dipendenza funzionale MATR determina CAP appartiene ad F+.

Non ci basta guardare l'insieme delle dipendenze funzionali perché non c'è MATR determina CAP direttamente in F quindi dobbiamo sapere se però, anche se non è in F, è nella chiusura di F.

Se vogliamo verificare se MATR determina CAP è nella chiusura di F, possiamo sfruttare la proprietà che abbiamo appena visto e calcolare la chiusura di MATR rispetto a F e poi vedere se CAP appartiene a questa chiusura di MATR rispetto a F.

La proprietà che abbiamo appena visto ci garantisce che una condizione necessaria e sufficiente per decidere se quella dipendenza funzionale appartiene alla chiusura.

La chiusura di MATR l'abbiamo già ricavata prima in un esempio precedente e sappiamo che comprende MATR, NS, IS, CAP, CF e DN.

Vediamo che CAP appartiene alla chiusura e quindi possiamo concludere che anche la dipendenza funzionale MATR determina CAP appartiene alla chiusura di F e quindi ci sarà un modo, anche se adesso non sappiamo dire esattamente quale, ma sappiamo che c'è sicuramente un modo per derivare la dipendenza funzionale MATR determina CAP, dall’insieme delle dipendenze funzionali F.

Facciamo un altro esempio: ora vogliamo sapere se MATR determina VO, il voto, cioè se questa dipendenza funzionale è derivabile in F.

La chiusura di MATR è la stessa di prima, perché l'abbiamo già ricavata sempre rispetto all'insieme F che non cambia, quindi la chiusura di MATR comprende MATR, NS, IS, CAP, CF e DN e vediamo che VO, il voto non è incluso nella chiusura di MATR e quindi dato che quella proprietà era una proprietà necessaria e sufficiente possiamo anche concludere che la dipendenza funzionale MATR determina VO, non è deducibile da F.

Ora finalmente possiamo capire a cosa è servito parlare della chiusura di un insieme di attributi perché noi siamo partiti dal parlare del problema dell'equivalenza tra due insiemi di dipendenze funzionali, dicendo che in effetti calcolando la chiusura di un insieme di dipendenze funzionali siamo in grado di risolvere il problema dell'equivalenza, ma il problema è che per calcolare la chiusura di un insieme di dipendenze funzionali abbiamo bisogno di un algoritmo che è necessariamente esponenziale e che quindi non è applicabile in pratica.

Poi abbiamo visto che però la chiusura non di un insieme di dipendenze funzionali ma di un insieme di attributi è invece efficiente da calcolare, perché polinomiale e ora possiamo capire come fare per sfruttare questa chiusura di insiemi di attributi per risolvere il problema dell'equivalenza e quindi abbiamo questo terzo approccio al problema dell'equivalenza, cioè controlliamo se ogni dipendenza funzionale X determina Y di F è deducibile in G, questo per calcolare l'equivalenza o meno tra F e G dove F e G sono due insiemi di dipendenze funzionali.

Quindi se noi vogliamo sapere se ogni dipendenza funzionale X determina Y di F è deducibile nell'altro insieme, cioè G, possiamo farlo verificando se Y è nella chiusura di X usando - e questo è sottolineato perché è l'aspetto meno ovvio - le dipendenze funzionali non di F ma le dipendenze funzionali di G, cioè dell'insieme di dipendenze funzionali di arrivo, con cui vogliamo verificare se c'è una equivalenza.

Quindi scrivendolo con una formula dobbiamo verificare se Y è un sottoinsieme di XG+.

L'idea è che - provo a ripetere questo primo punto - l'idea è che noi vogliamo verificare se F e G sono equivalenti tra di loro dove F e G sono due insiemi di dipendenze funzionali. Per farlo, applicando l'idea della chiusura di un insieme di attributi, invece che di un insieme di dipendenze funzionali, operiamo in questo modo: noi prendiamo una per una ogni dipendenza funzionale di F e guardiamo se è possibile ricavare Y partendo da X anche dentro G.

Dentro F è ovvio perché abbiamo questa dipendenza funzionale e quindi ovviamente è possibile ricavare Y partendo da X, ma quello che non è ovvio è che dobbiamo verificare, è se è possibile ottenere Y partendo da X anche in G.

Per farlo calcoliamo la chiusura dell'insieme di attributi X rispetto alle dipendenze funzionali di G, quindi sfruttiamo quelle di G, e verifichiamo se Y appartiene a questa chiusura. Se la risposta è “no, non appartiene”, allora sappiamo che sicuramente non sono equivalenti.

Quello che otteniamo con questo primo punto non è direttamente equivalenza. Da questo primo punto sappiamo solo che F+ è un sottoinsieme di G+.

Dobbiamo controllare anche l'altro verso cioè se G+ è un sottoinsieme di F+. Per farlo facciamo la stessa cosa ma dal punto di vista opposto, quindi dobbiamo controllare anche che ogni dipendenza funzionale X determina Y di G è deducibile in F, verificando se Y e nella chiusura di X, usando le dipendenze funzionali non di G ma di F e quindi controlliamo se Y è un sottoinsieme di XF+, dove X determina Y è una dipendenza funzionale di G. Se verifichiamo questa proprietà anche per ogni dipendenza funzionale di G allora possiamo dire che la chiusura di G è un sottoinsieme della chiusura di F.

Nel primo punto abbiamo verificato il verso opposto e quindi se verifichiamo tutti e due i punti possiamo dire che le due chiusure sono esattamente uguali e quindi che i due insiemi di dipendenze funzionali sono equivalenti tra di loro.

Rivedendo in modo più dettagliato, più formale, la formula, possiamo dire che F è equivalente a G se e solo se, per ogni dipendenza funzionale X determina Y in F, vale che la dipendenza funzionale X determina Y appartiene a G+ e, per ogni dipendenza funzionale X determina Y in G, questa dipendenza funzionale appartiene anche alla chiusura di F.

Questa prima parte della slide è quello che in realtà già sapevamo prima e già abbiamo visto che questa strada però è troppo costosa da percorrere. Adesso che abbiamo parlato di chiusura di un insieme di attributi possiamo fare un passo ulteriore e avere il vero meccanismo che possiamo usare per controllare l'equivalenza. “Vero” perché effettivamente applicabile perché è sufficientemente efficiente.

Quindi nella seconda parte della slide riscriviamo questa regola ma sfruttando la chiusura di insiemi di attributi, quindi possiamo dire che F è equivalente a G se e solo se per ogni dipendenza funzionale X determina Y in F, Y è un sottoinsieme di XG+, cioè - notate la G in pedice - controlliamo che per ogni dipendenza funzionale in F sia possibile derivare Y in G partendo da X e deve valere anche che per ogni dipendenza funzionale X determina Y in G noi riusciamo a derivare Y dentro F partendo da X.

Se valgono queste condizioni possiamo avere la conclusione che F e G sono equivalenti tra di loro.

Vediamo un esempio: abbiamo due insiemi di dipendenze funzionali nella tabella uno a sinistra, F che sono magari le dipendenze funzionali che ha ricavato il primo progettista, conoscendo il dominio, poi abbiamo da altre dipendenze funzionali nella seconda colonna della tabella in G che sono state ricavate da un altro progettista e vogliamo verificare se questi due insiemi di dipendenze funzionali siano equivalenti tra di loro.

Dobbiamo controllare tutti e due i lati dell'inclusione di sottoinsieme.

Per l'esempio facciamo solo uno dei due lati e l'altro lato può essere lasciato come esercizio.

Quindi adesso controlliamo se ogni vincolo, quindi ogni dipendenza funzionale di F può essere derivata in G. Prima di tutto possiamo considerare che ci sono molte dipendenze funzionali che sono uguali nei due insiemi e se sono uguali è ovvio che queste appartengono a tutte e due le chiusure dato che appartengono direttamente a F e G e quindi possiamo risparmiare del lavoro ed evitare di considerare queste che sono uguali e vedere se sono nella chiusura dell'altro tanto è ovvio che la risposta è sì.

Quali sono le dipendenze funzionali che sono uguali nei due casi? Vediamo che c'è per esempio a sinistra CF determina MATR, IS determina CAP, che c'è anche a destra in G e poi abbiamo CP determina NP,Q e Q determina TU e CP determina CO, quindi anche le ultime tre.

Quelle che rimangono quindi da verificare sono quelle riportate in fondo sulle slide cioè MATR determina NS, IS, CAP, CF e DN e MATR,CO determina VO,LO,DE,CP,NP in F. Quindi queste due dipendenze funzionali che sono in F dobbiamo capire se sono anche nella chiusura di G, sfruttando il meccanismo di chiusura di insiemi di attributi.

Iniziamo con la prima delle due dipendenze funzionali, quindi vediamo se MATR determina NS, IS, CAP, CF e DN che è una dipendenza di F, se appartiene a G+.

Per controllare se appartiene a G+ sappiamo che basta controllare se la chiusura di MATR che è l'antecedente, applicando le dipendenze funzionali di G, comprende, quindi racchiude, la conseguenza della dipendenza funzionale cioè NS, IS, CAP, CF e DN.

Quindi adesso cosa facciamo? Partiamo da MATR cioè l'antecedente e raccogliamo applicando l'algoritmo che abbiamo visto prima la chiusura di MATR.

Nella chiusura di MATR prima di tutto c'è MATR stesso che è la prima cosa da mettere, altrimenti rischiamo di dimenticarci, quindi c'è MATR stesso poi oltre a MATR stesso cosa mettiamo?

Guardiamo quali sono le dipendenze funzionali che sono applicabili, cioè in cui l’antecedente è MATR. Quelle applicabili sono la prima e la seconda, non possiamo applicare MATR, CO determina voto lode e data esame perché ci manca un pezzo dell'antecedente, quindi non basta MATR, ma bisogna avere tutti gli antecedenti.

La stessa cosa per quella successiva che è MATR,CO, NS non possiamo applicarla. Quindi aggiungiamo al nostro insieme di attributi che abbiamo raccolto, oltre a MATR, aggiungiamo NS,IS che è la prima delle dipendenze funzionali di G, poi aggiungiamo DN,CF che sono nella seconda dipendenza funzionale di G.

Non abbiamo altre dipendenze funzionali che “scattano” per l'antecedente MATR e che vengono attivate da MATR, ma possiamo considerare quali sono quelle che adesso vengono attivate dagli attributi che abbiamo raccolto finora.

Ci accorgiamo che la terza viene attivata però da CF, cioè CF determina MATR, ma MATR ce l'abbiamo già quindi non cambia l'insieme degli attributi che abbiamo raccolto.

La quarta è IS determina CAP e IS è nell’insieme degli attributi che abbiamo raccolto finora, CAP non ce l'abbiamo ancora e quindi possiamo aggiungere CAP all'insieme degli attributi che abbiamo raccolto come chiusura.

Poi le altre dipendenze funzionali che rimangono, vediamo che non sono attivate dagli attributi che abbiamo raccolto perché nessuna ha a sinistra uno degli attributi dell’insieme.

Quindi abbiamo terminato un calcolo della chiusura di MATR, sappiamo quindi che la chiusura di MATR comprende questi attributi MATR, NS,IS,DN, CF e CAP.

Noi vogliamo controllare se il conseguente della dipendenza funzionale che stiamo analizzando è incluso in questo insieme e vediamo sì che è un sottoinsieme, un sottoinsieme proprio. Quindi possiamo concludere che questa dipendenza funzionale che stiamo analizzando di F, cioè MATR determina NS, IS, CAP, CF, DN appartiene effettivamente alla chiusura di G.

Ora possiamo ripetere lo stesso ragionamento con la seconda dipendenza funzionale che dobbiamo controllare cioè MATR,CO, cioè il codice del corso determina voto, lode, data esame CP,NP. Questa è l'altra dipendenza funzionale di F che dobbiamo controllare e anche l'ultima.

Vogliamo sapere se questa dipendenza appartiene alla chiusura di G. Per farlo cosa dobbiamo controllare? Usiamo ancora il meccanismo della chiusura di un insieme di attributi: partiamo dall'insieme di attributi MATR,CO che è l’antecedente della dipendenza funzionale e calcoliamo la chiusura usando le dipendenze funzionali non di F ma di G quindi partendo da MATR,Co, cosa possiamo ricavare? Dato che abbiamo MATR, sappiamo già che possiamo ricavare tutti gli attributi che sono attivati da MATR, che sono quelli che abbiamo calcolato nella slide precedente, quindi sappiamo già che possiamo raccogliere nella chiusura di MATR,Co: NS, IS,DN,CF e CAP. inoltre abbiamo anche Co quindi tra MATR, Co, riusciamo a ricavare anche attivando la quinta dipendenza funzionale di G, ricaviamo anche il voto o Vo, la lode e la data dell'esame. Nell'insieme che abbiamo ricavato finora, oltre a MATR,Co, abbiamo ricavato NS tramite la prima delle dipendenze funzionali quindi sappiamo che possiamo ricavare CP e da CP ricaviamo NP e Q da Q ricaviamo TU. Quindi cosa abbiamo fatto? Siamo partiti da MATR, Co e abbiamo ricavato tutti gli attributi del nostro database che abbiamo, insomma tutti gli attributi A. Dato che abbiamo ricavato tutti gli attributi, sicuramente gli attributi del conseguente nella dipendenza funzionale da cui siamo partiti sono inclusi in questi, quindi possiamo dire che anche la seconda dipendenza funzionale MATR determina Vo,Lo,DE,CP,NP, appartiene alla chiusura di G. Dato che le altre dipendenze funzionali, come abbiamo detto, di F sono esattamente uguali sappiamo sicuramente che appartengono dalla chiusura. Allora grazie a questo ragionamento possiamo dire che la chiusura di F è un sottoinsieme della chiusura di G.

Ci rimane da calcolare l'altro verso quindi considerare le varie dipendenze funzionali di G e vedere se sono incluse nella chiusura di F, per sapere se anche la chiusura di G è un sottoinsieme della chiusura di F e determinare quindi che F e G sono equivalenti tra di loro. Però non facciamo anche questo passo, potete svolgerlo voi come esercizio.

**13**

Ora possiamo chiudere questa prima lezione sulle dipendenze funzionali esaminando qual è il rapporto tra le dipendenze funzionali, le chiavi e le superchiavi.

In effetti si possono usare le dipendenze funzionali per ricercare le superchiavi di una relazione.

Iniziamo richiamando qual è la definizione di superchiave, che è quella che trovate sulla slide che dice che data una relazione r su attributi A il sottoinsieme K degli attributi A è una superchiave se e solo se per ogni coppia di tuple ti, tj di r se ti e tj sono uguali su K allora devono essere uguali anche su A e, se riguardate adesso questa definizione, noterete che è molto simile alla definizione che abbiamo visto su cosa sono le dipendenze funzionali.

C'è una differenza, la differenza è che non è detto, nelle dipendenze funzionali, che si operi nel conseguente di queste implicazioni, su tutto A, cioè su tutti gli attributi di una relazione, perché in generale si considera solo qualche attributo, che possono essere anche tutti.

Quindi questa definizione è identica alla definizione di una dipendenza funzionale K determina A dove A sono tutti gli attributi di una relazione. Cioè se applichiamo la definizione di dipendenza funzionale alla dipendenza funzionale K determina A otteniamo proprio la definizione di superchiave K.

Grazie a questa osservazione possiamo trovare quindi un'altra definizione di superchiave, perché possiamo dire che un insieme di attributi K che è un sottoinsieme di A, dove A sono tutti gli attributi di R, è una superchiave se e solo se la dipendenza funzionale K determina A appartiene alla chiusura di F.

Per dire questo basta che noi controlliamo se A è uguale alla chiusura degli attributi K in F, quindi se A è uguale a KF+.

Come esempio di applicazione di questa definizione nuova di superchiave, possiamo considerare la relazione Esami e l'insieme di dipendenze funzionali F che abbiamo già visto prima e chiederci se MATR è una superchiave per la relazione esami. Per rispondere alla domanda dobbiamo calcolare la chiusura di MATR, che in realtà abbiamo già calcolato e sappiamo che è composta da MATR,NS,IS,CAP,CF,DN.

Non ci sono tutti gli attributi di Esami, per esempio manca il voto e quindi sappiamo che MATR da solo non è una superchiave per Esami.

Ora possiamo chiederci se MATR,Co,NS è una superchiave per esami. Quindi aggiungiamo Co, il codice del corso e nome studente e applichiamo l'algoritmo di calcolo di chiusura di un insieme di attributi a X che è uguale MATR,Co,NS.

Per riflessività sappiamo già che questa chiusura conterrà sicuramente tutti gli attributi di X cioè MATR,Co,NS. Poi sappiamo che ricaviamo anche almeno tutti gli attributi che abbiamo già ricavato usando solo MATR e quindi abbiamo sicuramente che C contiene almeno MATR,NS,IS,CAP,CF,DN,Co.

In più ci sono tutte le dipendenze funzionali che vengono attivate dai nuovi attributi.

In particolare, se noi abbiamo MATR,Co, possiamo “attivare” la dipendenza funzionale che dà anche Vo,Lo,DE,CP. Per convincervene potete guardare nella slide precedente le dipendenze funzionali.

Poi CP da solo ci dà inoltre NP e Q e Q da solo ci dà anche TU.

Se controlliamo gli attributi che abbiamo ottenuto vediamo che in effetti abbiamo ottenuto tutti gli attributi della relazione e cioè nella chiusura di X abbiamo tutti gli attributi della relazione. Quindi possiamo dire che X è una superchiave e cioè che MATR,Co,NS è una superchiave.

Il fatto che sia una superchiave non ci basta, facciamo un passo ulteriore, infatti vogliamo verificare se questa superchiave è anche una superchiave minimale e quindi può essere una chiave candidata.

Come verifichiamo se MATR,Co,NS è minimale? Proviamo con i sottoinsiemi di MATR, Co,NS e verifichiamo se a loro volta sono delle superchiavi.

Se nessun sottoinsieme è una superchiave, allora sappiamo che quella da cui siamo partiti è minimale, altrimenti sappiamo che non è così. Proviamo con la chiusura di Co ed NS e per esercizio potete provare a vedere che in effetti non si riescono a ricavare tutti gli attributi della relazione.

Poi proviamo a chiudere MATR,CO quindi senza NS e possiamo verificare, e anche questo è un esercizio, che in effetti riusciamo a ricavare tutti gli attributi di A. Quindi MATR,Co è una superchiave a sua volta.

Mi posso chiedere ma è una superchiave minimale? Io so già che MATR da solo non è una superchiave, poi posso verificare che Co da solo non è una superchiave, ma in effetti non è neanche necessario perché se già so che Co insieme a NS non è una superchiave, allora nessun suo sottoinsieme può essere una superchiave, perché nella chiusura ci saranno sicuramente meno attributi o al massimo tutti gli attributi che ho ottenuto con Co,NS e quindi posso dire che MATR,Co è una superchiave minimale perché nessun suo sottoinsieme proprio è una superchiave.

Se è minimale vuol dire che è anche una chiave candidata. In generale io posso avere più di una chiave candidata per la stessa relazione e lo stesso insieme di dipendenze funzionali. Se applichiamo lo stesso procedimento, e volendo potete farlo come esercizio, potete scoprire che io ho come chiavi candidate MATR,Co; CF,Co; MATR,CP; CF,CP. Quindi ho quattro chiavi candidate per la mia relazione.

Adesso vediamo un esempio su MATR,CP e verifichiamo se MATR,CP è una chiave (sottointeso: candidata).

La chiusura di MATR sappiamo già che ci dà ciò che c'è scritto sulla slide, che adesso non ripeto per l'ennesima volta, poi calcolo la chiusura di CP.

Con la chiusura di CP so che ricavo sicuramente CP per riflessività, poi applicando le dipendenze funzionali ricavo NP,Q e Co. Co in particolare vedete che è in grassetto. Perché l'ho scritto in grassetto? Perché io so che adesso posso fermarmi qua, non è necessario che continui a calcolare la chiusura di questi attributi perché se da MATR,CP ho ottenuto almeno MATR,Co e so già, perché l’ho calcolato prima, che MATR, Co è una chiave, allora so già che MATR,CP è una superchiave, perché so già che io potrò ricavare tutti gli attributi della relazione, quindi so già che K+ è uguale ad A.

Poi posso chiedermi se MATR,CP è una superchiave minimale, ma anche questo io l’ho già verificato perché so già che MATR da solo non è una superchiave, io so già che CO da solo non è una superchiave e quindi posso concludere che MATR,CP è una superchiave minimale e cioè una chiave candidata.

Quindi ho quattro chiavi candidate diverse che possono essere scelte dal progettista per ricoprire il ruolo di chiave primaria della relazione Esami. Vale questa proprietà che dice che se io ho due insiemi di dipendenze funzionali F e G e so che questi due insiemi sono equivalenti allora avranno le stesse identiche chiavi candidate e questa proprietà vale banalmente perché io so che se sono equivalenti hanno la stessa chiusura e quindi, dato che le chiavi vengono calcolate attraverso la chiusura, sicuramente hanno le stesse chiavi.

Vediamo un nuovo database che poi ci permette di fare una riflessione ulteriore. Questo database è composto da una sola relazione che si chiama CC che rappresenta dei conti correnti. La relazione è composta dagli attributi titolare, numero del conto corrente, numero dell'agenzia in cui il conto corrente è attivo, la città dell'agenzia, il direttore di questa agenzia, il saldo del conto corrente e la data di ultima movimentazione.

Usiamo le abbreviazioni degli attributi per semplicità. Sono quelli rappresentati sulla slide quindi IT,NCC eccetera e abbiamo un insieme delle dipendenze funzionali F in cui NCC determina NA, CA, cioè dal numero del conto corrente possiamo ricavare il numero dell'agenzia e la città dell'agenzia e poi la seconda dipendenza funzionale dice che dal numero del conto corrente possiamo ricavare il saldo del conto corrente e la data di ultima movimentazione. Il terzo vincolo ci dice che dal numero dell'agenzia e dalla città dell'agenzia possiamo determinare qual è il direttore dell'agenzia, cioè ogni agenzia ha un unico direttore e poi da DA, cioè dal direttore dell'agenzia possiamo determinare la città dell'agenzia. Cioè un direttore non può dirigere agenzie in città diverse, quindi può dirigere agenzie diverse ma devono stare tutte nella stessa città.

Vogliamo ricavare ora le chiavi candidate di CC, dato l'insieme di dipendenze funzionali con il metodo che abbiamo già visto, ma se guardiamo le dipendenze funzionali possiamo notare che l'attributo titolare cioè abbreviato T, non è nella conseguenza di nessuna dipendenza funzionale, non sta a destra di nessuna dipendenza funzionale.

Questo cosa significa? Significa che, se noi proviamo a usare un qualunque insieme di attributi che non comprende T, sicuramente non possiamo derivare T tramite chiusura perché non c'è nessuna dipendenza funzionale che deriva T, perché nessuna ha T a destra.

Quindi, se non riusciamo mai a scoprire T tramite nessuna dipendenza funzionale, significa che dobbiamo aggiungerlo noi nella chiave, dato che non verrà mai “da solo”. Quindi l'attributo titolare deve far parte necessariamente di ogni chiave candidata, perché, se non fa parte di una chiave candidata, allora la chiusura di questa chiave candidata sicuramente non comprende tutti gli attributi della relazione CC, perché sicuramente non comprenderà l'attributo Titolare.

Inoltre, Titolare non solo non è nel conseguente di nessuna dipendenza funzionale, non è neanche in nessun antecedente di una dipendenza funzionale, quindi Titolare da solo non può essere una superchiave, perché se io calcolo la chiusura di Titolare da solo, dato che non c'è nessuna dipendenza funzionale che viene attivata da questo attributo, la sua chiusura comprenderà solo Titolare stesso, per l’assioma di riflessività e quindi devo considerare, per calcolare le chiavi candidate di questa relazione, tutti i sovrainsiemi di T.

Ad esempio posso provare con T,NCC per vedere se è una superchiave e quindi una chiave candidata. Adesso non lo facciamo, potete svolgerlo come esercizio.

Come nota di chiusura di questa lezione, possiamo osservare che gli attributi coinvolti dalle dipendenze funzionali non sempre coprono tutti gli attributi della relazione, come in questo esempio del conto corrente.

E quando questo succede, allora, gli attributi che non vengono nominati da nessuna dipendenza funzionale, necessariamente devono far parte delle chiavi candidate che esaminiamo, perché appunto non possiamo derivarli tramite le dipendenze funzionali.

Un'ultima nota: ogni volta che troviamo una relazione R che ha attributi A e una chiave primaria K, allora implicitamente, anche se non è stata dichiarata, ci sarà una dipendenza funzionale K determina A.

**14**

In questa seconda lezione sulla teoria della normalizzazione introdurremo brevemente cos'è la normalizzazione, poi parleremo delle proprietà delle composizioni e vedremo una forma normale che si chiama Boyce-Codd Normal Form.

L'esempio su cui ci baseremo sarà quello che abbiamo già usato e che riguarda la relazione Esami e comprende gli attributi che potete leggere sulla slide. Useremo sempre la forma abbreviata degli attributi con le dipendenze funzionali riportate sulle slide. In effetti quando si applica la teoria della normalizzazione la comprensione del significato degli attributi serve per scrivere le dipendenze funzionali, quindi il progettista usa la loro “semantica”, usa la propria conoscenza del mondo per sapere quali dipendenze funzionali esistono su questo dominio. Una volta scritte le dipendenze funzionali, per analizzarle alla luce della teoria della normalizzazione, non è necessario comprendere cosa significhino gli attributi, ma è sufficiente trattarle in modo simbolico, utilizzando le varie dipendenze funzionali che sono state stabilite.

Questo è l’outline della lezione e iniziamo discutendo cos'è la normalizzazione. Abbiamo già introdotto, in generale, il concetto di normalizzazione nella lezione scorsa dicendo che è il processo che porta da una o più relazioni qualunque a relazioni che rispettano una forma normale.

Come si fa a compiere questo processo?

La normalizzazione decompone uno o più schemi di partenza ottenendo più schemi di arrivo che rispettino la forma normale. Vedremo in seguiito che esistono più forme normali. Per ogni forma normale esiste un processo specifico per ottenere quella forma normale, cioè per decomporre uno schema di partenza e ottenere degli schemi decomposti che rispettino queste forme normali.

Ora, invece di iniziare direttamente a parlare di forme normali specifiche, introduciamo prima questo processo di decomposizione in generale. I concetti che adesso introduciamo varranno per ogni processo di normalizzazione.

Iniziamo parlando delle decomposizioni senza perdita di informazione, quindi “di informazione” è sottointeso sulla slide.

Consideriamo una versione più semplice della relazione esami in cui utilizziamo qualche attributo in meno per compattezza e per semplicità. In questa versione della relazione esami che chiamiamo S, abbiamo solo questi sei attributi: la matricola dello studente, il nome dello studente, il voto che ha conseguito all'esame, qual è il corso di cui ha superato l'esame, qual è il codice del corso e qual è il titolare del corso. Quindi questo è lo schema della relazione.

Abbiamo nella figura tre tuple per dare un esempio. Abbiamo lo studente Piero con matricola 31 che ha superato sia l’esame di basi di dati con il professor Pensa, che l'esame di algoritmi col professor Damiani prendendo 27 e 26 come voti rispettivamente. Poi abbiamo un secondo studente, Giovanni, che è la terza tupla, con matricola 37, che ha superato l'esame di basi di dati non con il professor Pensa, ma con il professor Anselma, che infatti ha un codice diverso, il codice 03 che è il codice del corso. Decomponiamo questa relazione, con questi sei attributi, in due sottorelazioni. Una sottorelazione, che chiamiamo S1, comprende solo gli attributi matricola, nome studente, voto e corso, quindi non ha i due attributi codice del corso e titolare del corso.

La relazione S2 invece ha tre attributi che sono corso, codice del corso e il titolare. Ovviamente ogni decomposizione deve avere qualcosa in comune, perché le relazioni decomposte devono “parlare” tra di loro, cioè deve essere possibile metterle in join per riunire le informazioni che abbiamo suddiviso in due relazioni diverse. Quindi per esempio non avrebbe avuto senso decomporre la relazione S in due relazioni S1 ed S2 che non hanno neanche un attributo comune, quindi che non hanno nemmeno corso in comune, perché, se noi togliessimo corso dalla relazione S2, non avremmo nessun attributo su cui “fare ponte” per riuscire a riunire le informazioni che abbiamo decomposto. Quindi ovviamente ci deve essere almeno un attributo in comune.

Ora vediamo le istanze di queste due relazioni S1 e S2.

La relazione S1 avrà queste tre tuple, due tuple per lo studente 31 Piero e una tupla per lo studente 37 Giovanni. Piero ha superato l'esame del corso di base di dati e l'esame di algoritmi, mentre Giovanni quello di basi di dati.

La relazione S2 avrà tre tuple: la prima tupla è basi di dati col professor Pensa e codice del corso 01, la seconda tupla algoritmi col professor Damiani con il codice del corso 02, la terza tupla basi di dati con il codice del corso 03, con il professor Anselma.

Come abbiamo ottenuto queste istanze? Semplicemente siamo partiti dalla relazione di partenza S e abbiamo proiettato le tuple di S usando l'operatore dell'algebra relazionale di proiezione, usando come attributi quelli che sono nello schema di S1 e S2, quindi semplicemente abbiamo fatto questo per ottenere le istanze di S1 ed S2 con questi nuovi schemi.

Ora ci possiamo porre la domanda: “Dato che in seguito alla decomposizione ho delle tuple che rappresentano informazioni sul mondo che io sto modellando nella mia base di dati, queste informazioni rappresentate sono le stesse della base di dati originale? Oppure sto rappresentando delle informazioni diverse, quindi per esempio ho introdotto informazioni spurie, oppure ho perso informazioni?”.

Per esempio, se riguardo le slide scorse - adesso io nel video non mando indietro però voi potete rivedere quella decomposizione - la prima tupla della relazione S ci dice che lo studente Piero che ha matricola 31 ha sostenuto l'esame di basi di dati col professor Pensa.

Con la decomposizione ho S1 e S2: come faccio a ottenere l’informazione che avevo in origine? Posso ricomporre le informazioni che ho suddiviso nelle due relazioni, usando un join, in questo caso per esempio un natural join. Dato che l'attributo in comune tra le due relazioni ha lo stesso nome, posso usare il natural join. Se lo applico ottengo ciò che potete vedere sulle slide. In particolare, facendo ponte sull’attributo corso, accoppio lo studente Piero con tutte e due le versioni del corso di base di dati, perché hanno lo stesso nome in tutte le versioni: sono due corsi diversi con codice diverso, tenuti da due professori diversi, Pensa e Anselma.

Lo studente Piero, però, se guardate in basso sulle slide, nel natural join compare tre volte perché una volta compare per l'esame del corso di algoritmi e due volte viene replicato con lo stesso voto per le due versioni del corso di base di dati e lo stesso succede per lo studente Giovanni, che compare anch'esso due volte, una per ognuna delle due versioni del corso di basi di dati.

Mi potrei chiedere se il problema consiste nel fatto che ho usato un natural join e se usando un'altra operazione per riunire le informazioni avrei avuto i risultati corretti, ma la risposta è no, perché l'unica operazione che posso svolgere su queste due relazioni per mettere insieme le informazioni è un'operazione di join. Invece del natural join avrei potuto usare un equijoin, ma comunque avrei dovuto “fare ponte” sull'attributo corso quindi in realtà il problema è il fatto che sto usando l'attributo corso per fare ponte nel join.

Confrontiamo le due relazioni. Ho evidenziato le tuple che sono state aggiunte: la seconda e la quarta tupla che vedete sulla slide e che abbiamo già commentato prima. Cosa è successo? È successo che applicando il natural join ho introdotto delle tuple in più. Cioè, decomponendo la relazione S di partenza nelle due relazioni S1 e S2, ho ottenuto delle tuple in più. Queste tuple in più sono tuple scorrette che rappresentano delle informazioni che non erano presenti nella base di dati originale, cioè ho introdotto delle tuple che chiamiamo spurie.

In questo processo io ho in effetti perso informazioni, quindi ho questa situazione, che sembra paradossale, in cui introducendo delle tuple in più in realtà ho perso delle informazioni. Ho perso le informazioni nel senso che non so più attribuire l'esame di uno studente al corso esatto perché ho perso l’informazione di quale sia esattamente il vero corso di cui lo studente ha superato l'esame.

Vedendo in modo più formale quello che è successo: sono partito da una relazione r(A) dove A è l'insieme degli attributi che compongono uno schema di r e ho decomposto r in due relazioni r1(A1) e r2(A2), dove A1 e A2 sono due sottoinsiemi di A la cui unione forma A, quindi non ho buttato via attributi, perché la mia intenzione è di rappresentare le stesse informazioni.

r1 ed r2 li ho ottenuti tramite la proiezione di r sullo schema di r1 cioè gli attributi A1 e la proiezione di r sullo sche,a di r2, cioè gli attributi A2. La differenza insiemistica r1(A1) natural join r2(A2) meno r(A) mi dà esattamente ciò che è cambiato a causa della decomposizione. Cioè io con il natural join cerco di ricomporre le informazioni originali, con la differenza dell'algebra relazionale vedo che cosa ho ottenuto in più, quali sono le tuple spurie che ho introdotto a causa della mia decomposizione.

Quando sono contento? Sono contento quando applico questa differenza insiemistica e la differenza è vuota, perché questo significa che non ho introdotto nessuna tupla spuria.

Ora, se io osservo questa formula in cui ho questo join e questa differenza e impongo che il risultato sia un insieme vuoto, ottengo esattamente la definizione di decomposizione con join senza perdita di informazione.

Quindi immaginate che in questa formula io aggiunga “uguale insieme vuoto” a destra e ottenga esattamente questa definizione in cui la relazione di partenza deve essere esattamente uguale al join tra la relazione r1 e la relazione r2, dove r1 e r2 sono due decomposizioni della relazione originale. Questa è la definizione di decomposizione con join senza perdita di informazione.

Facciamo un altro esempio: supponiamo di avere decomposto la relazione S originaria in un modo diverso. Cioè ho decomposto la relazione tenendo, al posto dell’S1 che avevo ottenuto prima (l’S1 di prima aveva come attributi matricola, nome studente, voto e corso), un S1 in cui invece di corso metto codice del corso e quindi ottengo la decomposizione S1’ in cui l'istanza di S1’ è quella che vedete in basso sulla slide e in cui appunto ho codice corso al posto di corso. Se ora provo a ricavare di nuovo le informazioni originali, cioè applico il natural join, riesco a ottenere esattamente la relazione S originale, senza introdurre delle tuple spurie.

Il concetto di riuscire a riottenere le informazioni originali è un concetto importante, cioè questa è la proprietà di decomposizione senza perdita di informazione che vogliamo che assolutamente venga rispettata da ogni normalizzazione.

Quindi definiremo delle normalizzazioni che decompongono la relazione originale o le relazioni originali in sottorelazioni, quindi effettuano decomposizioni, ma ogni normalizzazione che definiremo dovrà assolutamente rispettare questa proprietà, perché vogliamo che sia possibile ricostruire le informazioni originali senza che la nostra decomposizione “inventi” delle informazioni in più, cioè crei le tuple spurie. Questa proprietà deve valere su ogni istanza della base di dati e quindi delle relazioni originali e delle loro decomposizioni; non deve essere una cosa che noi verifichiamo e che vale solo occasionalmente per una decomposizione, ma deve valere in generale per ogni possibile evoluzione del nostro database in cui verranno inserite altre tuple diverse.

In particolare in questa decomposizione che stiamo vedendo sulla slide immaginiamo cosa succede se nella relazione S2, quindi quella a destra in alto, non abbiamo nel database la combinazione con il codice corso 03, quindi la terza tupla non esiste, facciamo conto che non sia ancora stata inserita. In questo caso riusciremmo a ottenere da ogni natural join, anche quello precedente (quello che dava problemi perché introduceva tuple spurie) la relazione originale, cioè quella di partenza, ma comunque, anche in questo caso, la decomposizione sarebbe stata con perdita di informazione perché una proprietà che deriva da com’è fatto lo schema e non da quali sono le specifiche occasionali tuple che sono del database in un certo istante.

Abbiamo bisogno quindi di un criterio per stabilire quando abbiamo una decomposizione che siamo sicuri sia senza perdita di informazione. Introduciamo questo teorema che specifica esattamente qual è il criterio di cui abbiamo bisogno.

Il teorema dice che una decomposizione è senza perdita di informazione se e solo se A1 intersezione A2, quindi l'intersezione degli schemi delle due decomposizioni, è superchiave di A1, oppure, simmetricamente, se l'intersezione di A1 e A2 è superchiave di A2.

Qual è l’idea alla base di questo criterio? In modo intuitivo l'intersezione di A1 e A2 comprende esattamente gli attributi che usiamo per riuscire a ricostruire la relazione originale quando facciamo il join. Quindi, se l'intersezione di A1 e A2 è una superchiave di almeno una delle due relazioni, sappiamo che, quando faremo il join, riusciremo a individuare una e una sola tupla della relazione che abbiamo decomposto e quindi sappiamo che in questo modo non generiamo delle tuple spurie.

Possiamo riformulare lo stesso teorema alla luce di ciò che sappiamo adesso essere la definizione di superchiave, perché adesso sappiamo che, alla luce della teoria della normalizzazione, possiamo definire una superchiave come la chiusura di un insieme di attributi e quindi possiamo dire che una decomposizione di r in r1 e r2 è senza perdita di informazione se solo se A1 è un sottoinsieme della chiusura dell'intersezione di A1 e A2 oppure A2 è un sottoinsieme della chiusura dell'intersezione di A1 e A2.

Se riguardiamo alla fine della lezione scorsa cos'è la chiusura di un insieme gli attributi e quale rapporto ha con le superchiavi, vediamo che questa è una condizione esattamente equivalente a quella della slide precedente, perché se calcoliamo la chiusura dell’intersezione tra A1 e A2, otteniamo un sovrainsieme di A1 e sappiamo che appunto questa chiusura ci dà almeno tutti gli attributi di A1 e quindi sappiamo che stiamo ottenendo una superchiave per la relazione r1.

Allo stesso modo, simmetricamente, ragioniamo sulla parte destra. Vediamo un esempio. Consideriamo la relazione S di prima e la decomponiamo in S1’ e S2, dove S1’ e S2 sono le relazioni che abbiamo già visto prima, cioè S1’ è composto dagli attributi matricola, nome dello studente, voto e codice del corso e S2 è composto da corso, codice corso, e titolare.

Proviamo a calcolare l’intersezione dei due schemi e otteniamo che l'intersezione tra i due è codice corso. Codice corso è esattamente una superchiave - anzi è una chiave perché è minimale - di S2. Sappiamo che è una chiave di S2 perché se proviamo a calcolarne la chiusura, alla luce dei vincoli di dipendenza funzionale che abbiamo introdotto, possiamo ricavare tutti gli attributi perché c'è un vincolo che dice che codice corso determina corso, titolare e poi per riflessività otteniamo anche codice corso stesso.

Alla luce del teorema possiamo dire che questa decomposizione, la seconda che abbiamo visto, è senza perdita di informazione.

Riconsideriamo il teorema e dimostriamo che vale.

Per non appesantire troppo la trattazione, non dimostriamo l'intero teorema ma solo una parte rappresentativa perché permette di capire lo spirito del teorema. In particolare, dimostriamo solo la condizione di sufficienza, cioè la condizione che dice che se A1 intersezione A2 è una superchiave di r1 oppure una superchiave di r2, allora il join è senza perdita di informazione.

Detto in modo più formale, assumiamo per ipotesi che A1 sia un sottoinsieme della chiusura dell'intersezione di A1 e A2, oppure che A2 sia un sottoinsieme della chiusura dell'intersezione di A1 e A2. Questo significa che, se A1 è sottoinsieme della chiusura dell'intersezione, allora la dipendenza funzionale “A1 intersezione A2 determina A1” appartiene alla chiusura delle dipendenze funzionali di F, che sono le dipendenze funzionali di r che è la relazione che vogliamo decomporre.

Oppure, se A1 intersezione A2 è chiave di A2, vuol dire che in modo equivalente stiamo dicendo che la dipendenza funzionale “A1 intersezione A2 determina A2” appartiene alla chiusura delle dipendenze funzionali di F. Queste sono le ipotesi.

La tesi è che il join sia senza perdita di informazione cioè che r sia esattamente uguale al join tra r1 e r2. Che sia esattamente uguale lo possiamo dimostrare con le due inclusioni, cioè dicendo che r è un sottoinsieme di r1 join r2 e anche che r1 join r2 sia un sottoinsieme di r.

Per non appesantire troppo la dimostrazione, dimostriamo solo una delle due inclusioni, cioè dimostriamo che r1 natural join r2 sia un sottoinsieme di r. L'altra inclusione in realtà è più semplice.

Quindi riassumiamo quali sono le ipotesi e qual è la tesi: le ipotesi sono che dentro la chiusura delle dipendenze funzionali F abbiamo almeno una di queste due dipendenze funzionali “A1 intersezione A2 determina A1” oppure “A1 intersezione A2 determina A2”. La tesi è che il natural join tra r1 e r2 è un sottoinsieme di r.

Per dimostrare questa tesi, assumiamo di prendere un qualunque elemento dell'insieme di sinistra, cioè una qualunque tupla che appartiene al natural join e dimostriamo che questa tupla deve appartenere anche a r, così stiamo dimostrando l'inclusione insiemistica.

Per dimostrare questo, usiamo le definizioni degli operatori. Iniziamo in particolare dalla definizione di natural join. Per rendere più semplice la trattazione, introduciamo una rappresentazione grafica che comunque non ci fa perdere in generalità.

Prendiamo una qualunque tupla t che appartiene al risultato del natural join tra r1 e r2. Lo schema del risultato del natural join, per ipotesi, deve essere uguale a quello dello schema di r. Per ipotesi perché stiamo considerando una decomposizione e quindi assumiamo che l'unione dei due schemi di r1 r2 deve essere A (cioè lo schema di r) e l'intersezione tra A1 e A2 è non vuota.

Quindi il risultato del natural join comprenderà alcuni attributi che appartengono solo ad A1 quindi sono in A1 meno A2, altri attributi che appartengono solo ad A2 e quindi ad A2 meno A1, oppure attributi che appartengono sia ad A1 che ad A2 e quindi ad A1 intersezione A2. Questo è lo schema del risultato del natural join.

Supponiamo che la tupla t appartenga al natural join, quindi la situazione sarà quella in figura, in cui questa tupla t avrà alcuni valori, che indichiamo generalmente con i segni grafici --- per gli attributi in A1 meno A2, /// per gli attributi nell'intersezione tra A1 e A2, +++ per gli attributi che sono solo in A2, quindi in A2 meno A1.

Ciò che vogliamo dimostrare è che questa tupla t generica appartiene anche a r.

Per farlo applichiamo la definizione di natural join “al contrario”. Noi sappiamo che il risultato del natural join è la tupla t e quindi, se il risultato è la tupla t, devono esistere anche le due tuple che noi mettiamo in join e quindi una tupla t1 in r1 e una tupla t2 in r2.

Queste due tuple devono essere fatte in modo abbastanza preciso, perché, se il risultato è la tupla t che abbiamo visto prima, quella con ---, /// e +++, allora sappiamo che t1 negli attributi propri di r1, che sono quelli solo di A1 o condivisi tra A1 e A2, dovrà essere fatta come la tupla t1 in figura. Avrà certi valori generici --- che sono gli stessi che abbiamo nel risultato del natural join per A1 meno A2 e altri valori /// per gli attributi che sono sia in A1 che in A2 e anche questi sono gli stessi che abbiamo visto per la tupla t che sta nel risultato.

Per la tupla t2 abbiamo un discorso analogo, quindi i valori degli attributi che sono solo in A2 sono quelli del risultato cioè +++ in A2 meno A1. I valori degli attributi che sono sia in A1 che in A2, dato che t2 combinata in join con t1 ci deve dare la tupla t, deve per forza avere ///, cioè lo stesso valore che abbiamo messo per la tupla t1.

Ricordiamoci come facciamo a ottenere r1 e r2 dato che conosciamo lo schema A1 e A2.

In una decomposizione per ottenere l'istanza della relazione decomposta, prendiamo l’istanza della relazione di partenza e poi applichiamo la proiezione sugli attributi propri della decomposizione, quindi proiettiamo su A1 per ottenere r1 e proiettiamo su A2 per ottenere r2.

Se facciamo così - ora stiamo continuando a seguire gli operatori al contrario, dal risultato agli operandi -sappiamo che per ottenere t1, che abbiamo visto nella slide precedente, ci devono essere dentro la relazione r di partenza delle tuple t’1. Non è detto che t’1 sia una sola tupla perché il join può anche partire da tuple diverse e poi farle collassare in un'unica tupla se hanno lo stesso valore sugli attributi su cui sto proiettando.

Quindi ci possono essere delle tuple t’1 nella relazione di partenza r che sono fatte come vediamo sulla slide: cioè, se il risultato della proiezione di t1 aveva --- ///, dato che adesso consideriamo il risultato della proiezione t1’ su A1 preso da r e non più da r1, per avere quel risultato ci deve essere almeno una tupla che ha come valore --- per gli attributi che sono solo in A1, e come valore /// per gli attributi che sono in A1 intersezione A2. Di questi valori siamo certi perché abbiamo già il risultato della proiezione. Invece per gli attributi che abbiamo eliminato con la proiezione, cioè quelli che sono solo in A2, cioè in A2 meno A1, questa tupla avrà certi valori che non conosciamo e che indichiamo come ???.

Per r2 facciamo un discorso analogo: cioè, se noi abbiamo ottenuto la tupla t2 in seguito a una proiezione, applicando al contrario la definizione dell'operatore di proiezione, sappiamo che questa tupla t2 deriva da almeno una - ma possono essere anche di più - tupla che ha negli attributi propri di A2, quindi in A2 meno A1, il valore +++ dato che il risultato della proiezione ha +++ per questi attributi. Poi per gli attributi che sono nell'intersezione di A1 e A2 ha i valori /// e poi invece per gli attributi che sono propri solo di A1 e quindi la proiezione non sappiamo che cosa ha buttato via, mettiamo genericamente i simboli \*\*\*.

Quindi adesso stiamo ripercorrendo le definizioni all'indietro e abbiamo determinato che nella relazione r ci sono almeno queste due tuple.

Per ipotesi sappiamo che nella chiusura delle dipendenze funzionali di F ci deve essere una di queste due dipendenze funzionali: o “intersezione tra A1 e A2 determina A1” oppure “intersezione tra A1 e A2 determina A2”, quindi queste sono le ipotesi da cui stiamo partendo. Almeno una delle due dipendenze funzionali deve stare nella chiusura di F.

Assumiamo che sia la seconda dipendenza che appartiene alla chiusura. Perché lo facciamo? Abbiamo perdita di generalità? No, perché poi possiamo ripetere la dimostrazione assumendo che sia l'altra dipendenza dentro la chiusura. Non lo facciamo esplicitamente però lo possiamo fare.

Quindi assumiamo che la seconda dipendenza appartenga alla chiusura cioè che “intersezione tra A1 e A2 determina A2” appartiene a F+, cioè alla chiusura di F.

La relazione r deve per forza rispettare tutti i vincoli imposti sulla relazione r, cioè deve essere per ipotesi una relazione valida rispetto a tutti i vincoli imposti, quindi deve rispettare tutti i vincoli che sono nelle dipendenze funzionali in F, sia quelli inseriti esplicitamente in F sia quelli che derivano implicitamente dalla chiusura di F.

Quindi, se r deve rispettare tutti i vincoli, deve rispettare anche il vincolo che dice che l'intersezione tra A1 e A2 determina A2. Se riprendiamo la relazione r che abbiamo ricostruito al contrario nella slide precedente, che comprende almeno le due tuple t1’ e t2’, e applichiamo questa dipendenza funzionale, possiamo notare che queste due tuple hanno lo stesso valore per l’intersezione tra A1 e A2.

Se vale il vincolo di dipendenza funzionale allora devono avere lo stesso valore per A2: dato che hanno lo stesso valore per l'antecedente della regola, devono avere anche lo stesso valore per il conseguente. Quindi gli attributi che sono in A2 meno A1 devono avere lo stesso valore per queste due tuple. Quindi abbiamo concluso che quel ??? che avevamo inserito perché non sapevamo quale fosse il valore di quella tupla, in realtà deve essere per forza uguale a +++, altrimenti violeremmo il vincolo di dipendenza funzionale.

Nella relazione r queste tuple t1’ e t2’ devono essere fatte come nella slide, cioè per t1’ dobbiamo avere --- , /// e +++; per t2’ dobbiamo avere \*\*\*,/// e +++.

Ora confrontiamo la relazione r che abbiamo appena ottenuto, in cui abbiamo detto che come minimo devono esserci queste due tuple t1’ e t2’ che appartengono a r, fatte nel modo che abbiamo discusso e che vediamo anche adesso sulla slide. Confrontiamo questa relazione r con la relazione ottenuta con r1 natural join r2 che avevamo all'inizio della dimostrazione, in cui andiamo a prendere una qualunque tupla t da questo natural join. Se li confrontiamo possiamo vedere che in effetti la tupla t deve appartenere a r.

Riassumendo, come abbiamo fatto la dimostrazione?

Seguendo tutte le definizioni “al contrario”. Applicando l'ipotesi abbiamo concluso che questa tupla t generica che abbiamo scritto in questo modo ---, /// e +++ sicuramente deve appartenere anche alla relazione r e quindi abbiamo dimostrato la tesi almeno limitandoci alla porzione di teorema che abbiamo deciso di dimostrare.

**15**

Passiamo a parlare di un'altra proprietà che deve valere per le decomposizioni, cioè la conservazione delle dipendenze.

Cosa vuol dire conservare le dipendenze? Iniziamo a considerare una decomposizione di una relazione R su cui valgono le dipendenze funzionali che sono in F e immaginiamo di decomporla nelle due relazioni R1 e R2, dove lo schema di R1 è A1 e lo schema di R2 è A2.

Le dipendenze funzionali in F sono delle dipendenze funzionali che “parlano” degli attributi che sono in A e possiamo “tradurle” in dipendenze funzionali relative a R1 e R2.

Perché vogliamo fare questo? Perché le dipendenze funzionali esprimono dei vincoli che una base di dati deve rispettare per essere coerente. Se il progettista ha inserito questi vincoli per R e noi decomponiamo R, allora dobbiamo riuscire a mantenere, per quanto possibile, queste dipendenze funzionali anche nella decomposizione, perché se non valessero la base di dati potrebbe evolvere in modo incoerente, perché ci sarebbero istanze di relazioni che una volta riunite darebbero tuple incoerenti con quelle che avremmo nella relazione R se non avessimo fatto la decomposizione.

Quindi occorre tradurre queste dipendenze funzionali in F in dipendenze che sono relative alla decomposizione cioè in A1 e A2. Associamo a ogni decomposizione un proprio insieme di dipendenze funzionali. R1 avrà l'insieme di dipendenze funzionali F1 e R2 avrà l'insieme di dipendenze funzionali F2. Come calcoliamo F1 e F2?

Sulla slide vediamo la definizione della restrizione di un insieme dipendenze funzionali rispetto a una decomposizione.

Partiamo da uno schema R(A) e consideriamo una decomposizione che si chiama Ri(Ai), cioè Ai sono gli attributi di Ri. La restrizione Fi è composta dall’insieme delle dipendenze funzionali X determina Y tali che X determina Y appartiene alla chiusura di F e tutti e due gli insiemi di attributi X e Y, che sono nell’antecedente e nel conseguente, sono sottoinsiemi degli attributi in Ai.

Notate un particolare che è poco ovvio, cioè noi non consideriamo solo F: nella definizione imponiamo che la dipendenza funzionale X determina Y debba appartenere alla chiusura di F e non solo a F e quindi non consideriamo solo i singoli vincoli che sono esplicitamente inclusi nell'insieme F, ma tutti quelli anche impliciti che derivano da quelli compresi in F.

Perché avanziamo questa richiesta poco ovvia? Vediamo due esempi.

Nel primo esempio supponiamo di non applicare la definizione corretta che abbiamo appena visto, ma di applicare la definizione in cui, invece di avere F+, abbiamo F e quindi non consideriamo la chiusura ma solo i vincoli esplicitamente elencati.

Partiamo da una relazione R(A,B,C), quindi con tre attributi, con un insieme di dipendenze funzionali F che ha le due dipendenze funzionali A determina B e B determina C. Decomponiamo R in R1, che ha A,B, e in R2, che ha B,C, e applichiamo questa definizione sbagliata di restrizione che considera solo F. Mettiamo in F1 la prima dipendenza funzionale perché parla degli attributi A e B e in F2 mettiamo la seconda dipendenza funzionale perché parla degli attributi B e C. Quindi F1 è uguale ad A determina B e F2 è uguale a B determina C. In questo caso funziona, quindi questa restrizione che abbiamo calcolato è corretta ma ora vediamo un altro esempio.

Nel secondo esempio continuiamo a sbagliare cioè a non applicare la definizione corretta ma a considerare solo F e non F+. Partiamo dalla stessa relazione R di prima che ha lo stesso insieme di dipendenze funzionali ma con una decomposizione diversa. R1 è sostituito da R’1 che comprende gli attributi A,C quindi abbiamo R’1, che ha gli attributi A,C, e R2, che ha gli attributi B,C, e vogliamo calcolare le restrizioni.

Per R2 non abbiamo problemi e calcoliamo la restrizione B determina C, ma per R’1 quali sono le dipendenze funzionali che vogliamo includere?

Se non consideriamo la chiusura di F ma solo le dipendenze funzionali che sono direttamente in F, vediamo che R’1 ha gli attributi A,C e non c'è nessuna dipendenza funzionale che considera gli attributi A,C perché ogni dipendenza ha l’attributo B, che non sta in R’1.

Quindi non attribuiamo nessuna dipendenza funzionale a R’1. Questo è un errore e perché è un errore? Perché nella relazione originale abbiamo queste due dipendenze funzionali che ci dicono che, se ci sono due tuple che hanno lo stesso valore su A, allora devono avere lo stesso valore su B e, se hanno lo stesso valore su B, devono avere anche lo stesso valore su C.

Questo vuol dire che, se prendiamo due tuple che hanno lo stesso valore su A, sicuramente devono avere anche lo stesso valore su C. Ma questo non succede nella relazione R’1 se noi non mettiamo dei vincoli di dipendenza funzionale, perché in R’1 non c'è nessun vincolo e quindi ammettiamo tuple che sono uguali su A e sono diverse su C, e questo è incoerente rispetto alla relazione originale.

Il controesempio dimostra che, se noi non consideriamo la chiusura di F, abbiamo dei problemi nel calcolo delle restrizioni e quindi non calcoliamo correttamente le dipendenze funzionali delle decomposizioni. Quindi, se nella definizione di restrizione applichiamo correttamente F+ invece di F, sappiamo, per effetto della regola di transitività, che c'è un'altra dipendenza funzionale implicita A determina C, che è in grassetto sulla slide, che poi va a finire nella restrizione di R’1 cioè in F’1.

Possiamo introdurre la proprietà di decomposizione che conserva le dipendenze, che definiamo come la decomposizione le cui restrizioni delle dipendenze funzionali permettono di derivare l'insieme di dipendenze funzionali da cui siamo partiti, cioè della relazione R.

Ora ci possiamo chiedere: “Ma questa proprietà vale per ogni possibile decomposizione oppure ci sono casi in cui la proprietà non vale?”. La risposta è che ci sono casi in cui la proprietà non vale.

Consideriamo come esempio la relazione R con i tre attributi A, B e C, che ha l'insieme di dipendenze funzionali F in cui R1 ha gli attributi A,B e R2 ha gli attributi A,C.

Sappiamo che la decomposizione è senza perdita di informazione perché R1 e R2 hanno in comune l’attributo A. Se calcoliamo l’intersezione rimane A, che è la chiave di una delle due relazioni, infatti è la chiave di R2 - questo lo ricaviamo dai vincoli di dipendenze funzionali, perché A determina C e quindi la chiusura di A comprende tutti gli attributi di R2. Quindi sappiamo che A è superchiave ma è anche chiave, perché è minimale in quanto è composta da un solo attributo.

Ora sappiamo che la decomposizione è senza perdita di informazione, ma cosa possiamo dire della conservazione delle dipendenze?

Calcoliamo le restrizioni delle dipendenze funzionali. Possiamo vedere che per R1, che ha gli attributi A,B, non ci sono dipendenze funzionali nelle restrizioni, perché non possiamo ereditare né A determina B, né B determina C e non ci sono altre dipendenze funzionali utili nella chiusura di F.

Per F2, cioè le dipendenze funzionali di R2 che ha attributi A,C, invece possiamo mettere nella restrizione la dipendenza funzionale A determina C e quindi abbiamo F1 che è vuoto e F2 che ha una sola dipendenza funzionale A determina C.

Cosa succede della dipendenza funzionale che non abbiamo portato da nessuna parte che era in F, cioè B determina C e non è andata a finire né in F1 né in F2?

Sappiamo che non possiamo metterla in nessuna delle due restrizioni perché la relazione R1 ha gli attributi A e B e quindi manca l’attributo C per potere mettere questa dipendenza funzionale. La relazione R2 ha gli attributi A,C e quindi manca B per potere mettere la dipendenza funzionale B determina C, quindi non possiamo metterla in nessuna delle due restrizioni. Se calcoliamo l'unione di F1 e F2 e quindi abbiamo semplicemente A determina C e calcoliamo la chiusura, sicuramente non possiamo ottenere B determina C, quindi la decomposizione non conserva le dipendenze.

Cosa succede se non conserva le dipendenze? Se consideriamo solo le dipendenze funzionali che siamo riusciti a mettere nelle restrizioni, cioè F1, F2, allora il database potrebbe avere delle istanze inconsistenti rispetto alle dipendenze funzionali decise dal progettista, che sono quelle in F.

Ad esempio potremmo avere due tuple t1 e t2 che appartengono al natural join di R1 e R2 (stiamo ricostruendo R e possiamo ricostruirlo senza perdita di informazione, senza tuple spurie, perché abbiamo verificato che vale la proprietà). t1 e t2 appartengono al natural join e possiamo avere queste due tuple che sono uguali su B e sono diverse su C e quindi stiamo violando la dipendenza funzionale B determina C che dice che appunto se sono uguali su B, devono essere uguali anche su C. Quindi possiamo avere una istanza del database che è incoerente.

Le dipendenze funzionali che non possiamo mettere nelle restrizioni andrebbero aggiunte comunque come vincoli globali, cioè come dipendenze funzionali e vincoli che non sono specifiche per una certa relazione, ma coinvolgono più relazioni tra di loro. Se non lo facciamo, come abbiamo detto, abbiamo dei problemi perché abbiamo una base di dati incoerente, ma se lo facciamo abbiamo comunque dei problemi perché la base di dati sarà sì coerente, però la verifica dei vincoli globali, che coinvolgono più relazioni, è generalmente molto costosa. Normalmente viene ammessa per i vincoli di integrità referenziale e non per questi che sono diversi.

Perché è costosa? Perché quando bisogna controllare un vincolo - e un vincolo deve essere controllato quando si modifica una delle tuple che sono coinvolte dal vincolo - bisogna ogni volta effettuare un join tra tutte le relazioni coinvolte, per poi controllare se il vincolo è rispettato dal join. L'operazione di join è costosa e va effettuata il minor numero di volte possibili, solo quando è necessario.

Ora possiamo finalmente iniziare a parlare di forme normali. Prima parleremo in generale di forme normali e poi vedremo due forme normali specifiche. La prima è la Boyce-Codd normal form, abbreviata BCNF, la seconda che vediamo sarà la terza forma normale, abbreviata 3NF.

Cosa sono le forme normali? Sono delle “ricette” di buona progettazione che sono il risultato di un processo di normalizzazione. Abbiamo un processo, cioè un algoritmo (parleremo proprio di algoritmi di normalizzazione) che parte da uno schema relazionale qualunque, che quindi può non rispettare la forma normale e lo trasforma in forma normale. La forma normale permette di dire che un certo schema di una base di dati gode di alcune proprietà, quindi per esempio minimizza le ridondanze e minimizza le anomalie di cui abbiamo parlato all'inizio della lezione sulla normalizzazione, quindi le anomalie di aggiornamento, di inserimento e di cancellazione.

Esistono diverse forme normali perché ogni forma normale trova un diverso punto di compromesso tra le proprietà delle decomposizioni desiiderabili.

Infatti esistono alcune proprietà che non è possibile conciliare completamente, come per esempio la compattezza della rappresentazione e la minimizzazione delle anomalie. Alcune forme normali riescono a eliminare qualunque anomalia e a garantire tutte le proprietà, però danno origine a una rappresentazione che è troppo poco compatta che quindi darebbe luogo a troppi join, che sono operazioni costose in un database. Quindi non è desiderabile per esempio avere il massimo della decomposizione delle relazioni e il massimo della minimizzazione delle anomalie.

In questa slide vedete una tabella di cui non importa se non riuscite a leggere i dettagli perché serve solo a dare un'idea del quadro delle diverse forme normali proposte. Non esistono infatti solo le forme normali di cui parleremo ma ne esistono molte altre.

Esiste una prima forma normale (la prima forma normale è in realtà quella che già usiamo e siamo abituati a usare senza parlarne esplicitamente perché richiede che nel modello relazionale ogni attributo abbia domini semplici).

La seconda forma normale non viene usata, ha solo un valore storico.

Della terza forma normale e della forma normale di Boyce-Codd parliamo esplicitamente, perché sono quelle largamente usate nella pratica.

Esistono formule normali ulteriori quali la quarta, la quinta, la domain-key normal form, la sesta forma normale, che di solito danno luogo a delle decomposizioni che sono troppo “decomposte”, cioè troppo piccole e quindi darebbero molti join.

Di questa slide ho già parlato e possiamo andare alla prossima, in cui appunto diciamo che tratteremo queste due forme normali la Boyce-Codd normal form abbreviata BCNF (in italiano forma normale di Boyce-Codd) e la terza forma normale, abbreviata 3NF.

In questa lezione parleremo solo della forma normale di Boyce-Codd. La definiamo però poi non vedremo è il relativo algoritmo di normalizzazione, mentre parleremo dell'algoritmo solo per la terza forma normale.

La Boyce-Codd normal form trae il proprio nome dalle due persone che l'hanno definita: Boyce, che è uno dei due inventori di SQL, e Codd, che abbiamo già incontrato perché è la persona che ha definito il modello relazionale.

Partiamo da una relazione R con schema già in prima forma normale - quindi ogni attributo ha un dominio semplice – con un insieme di dipendenze funzionali F.

Si dice che la relazione R è in forma normale di Boyce-Codd, cioè in BCNF, se e solo se, per ogni dipendenza funzionale X determina Y che appartiene ad F, si verifica almeno una delle condizioni sulla slide.

Notate che non devono valere per forza entrambe, ma basta che ne valga una.

La prima condizione è che Y è un sottoinsieme di X, cioè, per ogni dipendenza funzionale, il conseguente è un sottoinsieme dell'antecedente. Un altro modo per dire questo è dire che X determina Y è una dipendenza riflessiva. Se vi ricordate una regola di Armstrong parla delle dipendenze riflessive. Quindi il primo caso è che sia una dipendenza banale, banale perché appunto è riflessiva, in cui un conseguente è sottoinsieme dell’antecedente.

Il secondo caso invece è quello più interessante. In alternativa al primo caso, X può essere una superchiave per R.

Vediamo un esempio: partiamo dalla relazione Esami che ha le dipendenze funzionali che vedete nella slide. Possiamo verificare che questa relazione Esami non è in Boyce-Codd normal form. Ad esempio, guardate la prima dipendenza funzionale elencata: quella non rispetta nessuna delle due condizioni perché non è riflessiva (infatti il conseguente non è un sottoinsieme dell'antecedente) e inoltre MATR non è una superchiave. Sappiamo già che non è una superchiave perché abbiamo già calcolato prima le superchiavi di questa relazione e abbiamo già visto che MATR non è una superchiave.

Quindi sappiamo che almeno una dipendenza funzionale non rispetta nessuna delle due condizioni e quindi sappiamo che la relazione esami non è in Boyce-Codd normal form.

Consideriamo un altro schema con una sola relazione studente in cui abbiamo gli attributi matricola, nome studente, codice fiscale, data di nascita, con le due dipendenze funzionali “matricola determina codice fiscale, nome studente, data di nascita” e “codice fiscale determina matricola”.

Ci possiamo chiedere se questo schema è in Boyce-Codd normal form. Per rispondere dobbiamo trovare le superchiavi e vediamo che ovviamente ci sono due superchiavi che sono anche minimali, quindi sono chiavi candidate, che sono MATR e CF. Lo vediamo chiaramente perché da MATR possiamo derivare tutti gli attributi e anche da CF ricaviamo MATR, da cui poi deriviamo tutti gli attributi. Quindi sono sicuramente delle superchiavi e dato che sono composte da un solo attributo sappiamo che sono anche minimali e quindi sono chiavi candidate.

Tutte e due le dipendenze funzionali sono in Boyce-Codd normal form perché sono del tipo 2, cioè l'antecedente è una superchiave. Infatti, MATR è superchiave e anche CF è superchiave.

Qual è il senso della Boyce-Codd normal form? Il senso è che si parte dalla considerazione che le dipendenze funzionali rappresentano dipendenze tra attributi e cioè insiemi di attributi che identificano lo stesso concetto. La Boyce-Codd normal form rappresenta l’idea che in una relazione non dobbiamo rappresentare concetti diversi, infatti ammettiamo solo o i casi banali in cui il conseguente di una dipendenza funzionale è un sottoinsieme dell'antecedente, oppure dipendenze funzionali che dipendono dalle superchiavi. Quindi in ogni relazione ammettiamo solo attributi che, dipendendo da superchiavi, rappresentano “compattamente” un concetto, senza “sottodipendenze” tra gli attributi. Altre dipendenze funzionali non sono ammesse: se sono presenti significa che la relazione che stiamo esaminando non è in Boyce-Codd normal form e quindi va normalizzata per renderla in Boyce-Codd normal form.

Discutiamo informalmente il motivo per cui la BCNF evita di avere anomalie e ridondanze.

Parliamo prima delle ridondanze: una ridondanza è una ripetizione che è evitabile, perché, se ho un’informazione che viene rappresentata dalla ridondanza, posso ricavare l’informazione dalle altre che ho nel mio database.

Ogni dipendenza funzionale introduce una possibile ripetizione. Per esempio, se ho la dipendenza funzionale “matricola determina nome studente, data di nascita”, vuol dire che in ogni relazione in cui vale questa dipendenza funzionale, se due tuple hanno la stessa matricola, devono essere obbligatoriamente uguali anche nome studente e data di nascita.

In che modo interviene la Boyce-Codd normal form su questo? Fa in modo che in ogni relazione l'antecedente di ogni dipendenza funzionale, se la dipendenza non è banale, sia una superchiave.

Se l'antecedente è una superchiave, non posso avere due tuple che hanno lo stesso valore sulla superchiave e quindi queste dipendenze funzionali non possono introdurre ridondanze sugli attributi nel conseguente della dipendenza funzionale.

Consideriamo le anomalie: anomalie di aggiornamento, anomalie di inserimento e anomalie di cancellazione.

Iniziamo con le anomalie di aggiornamento, tenendoci sempre a livello informale. Abbiamo per esempio la relazione esami con gli attributi che potete leggere sulla slide. Nella stessa relazione memorizzo non solo l'esame ma anche tutti i dati dello studente come il nome, indirizzo, CAP e tutti i dati del professore. Quando aggiorno i dati dello studente o i dati del professore, dato che nella relazione ogni studente può essere ripetuto tante volte e anche ogni professione viene ripetuto tante volte (una volta per ogni esame), devo effettuare tanti aggiornamenti dei dati dello studente o del professore, uno per ognuno degli esami. Dato che in BCNF non abbiamo ridondanze, per il motivo che abbiamo appena discusso, non ci possono essere anomalie di aggiornamento.

Riguardo invece le anomalie di inserimento, consideriamo come esempio la relazione E, anche questa non in Boyce-Codd normal form, con matricola, nome studenti, data di nascita, codice del corso, voto e data esame, chiave primaria matricola e corso e due dipendenze funzionali: “matricola determina nome studente e data di nascita” e “matricola, corso determinano voto e data esame”. Quando ho un'anomalia di inserimento, succede che, ad esempio, nella relazione voglia inserire un nuovo studente; per inserire i dati di uno studente, cioè il suo numero di matricola, il suo nome e la sua data di nascita, non posso inserire solo questi attributi ma devo inserire anche come minimo la chiave primaria della tabella, perché gli attributi della chiave primaria non possono essere nulli, quindi dovrei inserire anche matricola e corso. Quindi la relazione E presenta un’anomalia di inserimento.

Se invece la relazione fosse in BCNF, avrebbe soltanto o dipendenze funzionali banali o dipendenze funzionali in cui ogni attributo della relazione dipende solo da una superchiave. Se succede questo, non posso avere in una stessa relazione più concetti diversi e quindi ogni concetto viene rappresentato in una relazione a sé stante. Quando voglio inserire un fatto nuovo e non posso inserire anche un altro concetto correlato,non è un problema perché posso inserire ogni concetto in una relazione a sé stante (vediamo questo solo a livello intuitivo).

Per le anomalie di cancellazione il discorso è analogo a quello che abbiamo appena fatto per le anomalie di inserimento.

Se abbiamo una relazione che non è in Boyce-Codd normal form e vogliamo decomporla per renderla in Boyce-Codd normal form, possiamo ricorrere a un algoritmo di normalizzazione che hanno proposto Boyce e Codd. Non vediamo questo algoritmo, perché abbiamo limitazioni di tempo e perché comunque vediamo l’algoritmo di normalizzazione in terza forma normale. Per darvi un'idea ho comunque scritto queste tre righe sulla slide per avere un'idea ad alto livello di come funziona.

L’algoritmo prende in considerazione ogni dipendenza funzionale della relazione R che vogliamo normalizzare.

Se questa dipendenza funzionale non viola la BCNF allora si va avanti a considerare la successiva dipendenza.

Se invece la dipendenza funzionale viola la BCNF allora la si usa per decomporre R.

Come decomponiamo? Definiamo una nuova relazione che comprende solo gli attributi che sono nell’antecedente e nel conseguente della dipendenza funzionale, quindi comprende gli attributi X,Y e assegniamo a questa nuova relazione, che chiamiamo RX, anche la dipendenza funzionale X determina Y. Se attribuiamo questa dipendenza funzionale a RX, cioè calcoliamo la chiusura dell’antecedente, vediamo che X è una chiave di RX perché da X posso derivare, per chiusura, tutti gli attributi di RX, cioè X,Y.

Quindi con questa decomposizione ho la garanzia che la nuova relazione R(X) sia in BCNF perché l’antecedente della dipendenza funzionale di RX è una chiave.

Poi prendiamo la relazione originale R e da quella eliminiamo Y, cioè tutti gli attributi nel conseguente della dipendenza funzionale che abbiamo usato per decomporre. Eliminiamo Y e non X: X lo dobbiamo tenere in R perché altrimenti non riusciremo a effettuare poi i join. Servono degli attributi nell'intersezione degli schemi e quindi teniamo X.

Andiamo avanti poi col ciclo a considerare tutte le altre dipendenze funzionali che non sono in BCNF. Quando arriviamo al termine di questo ciclo abbiamo decomposto la relazione R originale eliminando tutte le dipendenze funzionali che violavano la BCNF, quindi abbiamo la garanzia che il risultato sia in BCNF.

Vediamo un esempio (anche se abbiamo visto l'algoritmo solo ad alto livello, possiamo comunque vedere un esempio).

Consideriamo lo schema che contiene i dati sui conti correnti che ha come attributi titolare, numero conto, che insieme sono chiave primaria, poi numero dell'agenzia, città agenzia e saldo.

C'è una dipendenza funzionale che dice che il numero del conto determina il numero dell'agenzia, la città dell'agenzia e il saldo. Possiamo osservare che la relazione CC non è in Boyce-Codd normal form, perché la dipendenza funzionale che abbiamo non è riflessiva e ha nell’antecedente numero conto, che non è una superchiave, perché per essere una superchiave dovrebbe contenere almeno anche titolare.

Se però dalla relazione CC eliminiamo l'attributo titolare, possiamo trasformarla e decomporla in una relazione in Boyce-Codd normal form e otteniamo la decomposizione che c'è nella prossima slide.

Abbiamo CC’, che è una relazione che non ha più titolari ma ha numero conto (che è chiave primaria), numero agenzia, città agenzia e saldo con il vincolo di dipendenza funzionale uguale a prima.

Poi abbiamo una nuova relazione titolari, che ha i due attributi titolare e numero conto, che sono entrambi chiave primaria, e un insieme vuoto di dipendenze funzionali, a patto che siamo d'accordo che non stiamo riportando le dipendenze funzionali riflessive che sono banali e la dipendenza funzionale che deriva dalla chiave primaria. La domanda ora è: “Questo nuovo database è in Boyce-Codd normal form?”. Potete pensarci qualche secondo mettendo in pausa la slide e poi vi do la risposta.

La risposta è sì, è in Boyce-Codd normal form, perché abbiamo solo dipendenze funzionali che hanno come antecedente una chiave primaria che quindi ovviamente è una superchiave.

La Boyce-Codd normal form ha alcune limitazioni che fanno sì che non sempre sia possibile raggiungere questa forma normale.

Vediamo un esempio per capire quali sono. In questo esempio abbiamo una relazione con i tre attributi dirigente, progetto e sede e abbiamo due vincoli di dipendenza funzionale: il primo dice che progetto e sede determinano dirigente, quindi individuano anche la chiave primaria della relazione che sarà progetto, sede e la seconda dipendenza funzionale dice che il dirigente determina la sede.

Possiamo osservare che questa relazione non è in Boyce-Codd normal form perché la seconda dipendenza funzionale ha un antecedente che non è una superchiave, mentre la prima sì, ma comunque noi cerchiamo il controesempio che è la seconda dipendenza funzionale.

Lo scopo della decomposizione è di decomporre questa relazione in modo da avere delle sottorelazioni che siano in BCNF. Questo è possibile?

La risposta è no, non è possibile, perché ogni decomposizione che noi possiamo operare su questi tre attributi necessariamente fa in modo che la prima dipendenza funzionale, che comprende tutte e tre gli attributi, non possa essere conservata nella decomposizione. Quindi non è possibile trovare una decomposizione che conservi questa dipendenza funzionale, perché appunto coinvolge tutti gli attributi.

Questo vuol dire che se noi la decomponiamo siamo poi costretti ad aggiungere quel vincolo come un vincolo globale, che quindi ha quei problemi di efficienza di cui abbiamo già parlato.

Quindi la Boyce-Codd normal form dà molti vantaggi perché ci assicura di evitare tutte le anomalie che abbiamo individuato all'inizio della lezione sulla normalizzazione.

Purtroppo, però, non è sempre possibile arrivare a questa forma normale perché ci sono casi in cui ogni decomposizione possibile che possa ristabilire la forma normale violerebbe la proprietà di conservazione delle dipendenze. Quindi esistono casi in cui questa normalizzazione non è praticabile e questo è il motivo per cui nella prossima lezione considereremo un’altra forma normale: la terza forma normale.

**16**

Questa lezione è dedicata alla trattazione della terza forma normale e questo è l’outline della lezione.

Inizieremo a introdurre la terza forma normale, poi la definiremo (la abbreviamo come 3NF: terza forma normale in inglese). Poi parleremo di insieme di copertura minimale, che cos'è, lo definiamo e vediamo come ricavarlo. Poi parleremo della normalizzazione in terza forma normale.

Mentre per la BCNF non abbiamo discusso approfonditamente l'algoritmo che serve per partire da una relazione, un database qualunque e decomporlo in relazioni in BCNF, invece per la terza forma normale parliamo anche dell'algoritmo. Infine concludiamo discutendo le proprietà di questo algoritmo.

Per iniziare richiamiamo la BCNF che abbiamo visto nella lezione scorsa. La Boyce-Codd normal form è una forma normale per cui ogni dipendenza funzionale deve rispettare almeno una delle seguenti condizioni: è una dipendenza banale, cioè data la dipendenza funzionale X determina Y che appartiene all’insieme delle dipendenze funzionali, Y è un sottoinsieme di X, quindi è una dipendenza riflessiva, oppure l'antecedente X è una superchiave per la relazione R.

La BCNF è una proprietà molto desiderabile perché permette di definire schemi che eliminano le anomalie di aggiornamento, di inserimento e di cancellazione.

Purtroppo, non è sempre possibile raggiungere la BCNF. In alcuni schemi di database, decomposti per raggiungere la BCNF, non riusciamo a conservare le dipendenze funzionali, perché ci sono alcune dipendenze funzionali che “parlano” di attributi in relazioni diverse e questo, come abbiamo già discusso, non è una cosa desiderabile.

Quindi la BCNF ha un grosso vantaggio perché elimina le anomalie, però non è sempre possibile raggiungerla conservando le dipendenze.

A questo punto dobbiamo decidere se ci interessa di più conservare le dipendenze oppure non avere le anomalie. La scelta che si fa di solito è di conservare le dipendenze, perché è più importante avere efficienza nel controllo dei vincoli, quindi avere vincoli che non richiedono necessariamente sempre di effettuare un join.

Quindi è stata definita una ulteriore forma normale che si chiama terza forma normale. Si chiama terza forma normale perché ne esistono storicamente altre due prima, però noi parliamo solo della terza forma normale. Abbiamo fatto qualche cenno sulla prima, invece della seconda non abbiamo parlato, perché ha solo interesse storico.

Quindi definiamo una nuova forma normale, la terza forma normale che è un po’ più debole, nel senso che non permette di eliminare tutte le anomalie, però ha un vantaggio: è sempre possibile raggiungere la terza forma normale conservando le dipendenze funzionali, quindi accettiamo un compromesso rispetto a quello che ci darebbe la BCNF. Adesso possiamo definire cos'è la terza forma normale.

Prima di definire cos'è la terza forma normale dobbiamo introdurre la definizione di attributo primo. Partiamo dalla relazione che vediamo sulla slide, la relazione esami, che ha, in questa versione, solo quattro attributi: matricola studente, codice corso, voto studente e codice professore con le dipendenze funzionali “matricola, corso determinano voto” e “codice professore determina codice corso”.

La relazione ha quindi come chiavi candidate “matricola, corso” e “matricola, codice professore”. Potete verificare che sono chiavi candidate calcolando la chiusura di questi due insiemi di attributi e quindi constatare che comprende tutti gli attributi della relazione e poi verificare che sono minimali provando a togliere matricola o codice corso o codice professore e verificare che queste non risultano essere superchiavi.

Quindi abbiamo questa relazione, con queste due chiavi: “matricola, corso” e “matricola, codice professore”. Codice corso, da solo, non è una chiave primaria, non è neanche una chiave candidata e non è neanche una superchiave, ma è un attributo comunque contenuto in una chiave della relazione esami. Codice corso è un attributo primo per la relazione esami.

Possiamo ora definire che cos'è un attributo primo: data una relazione R con schema A, gli attributi Y, che sono un sottoinsieme di A, sono detti attributi primi se e solo se sono un sottoinsieme di K, dove K è una chiave della relazione R. Quindi, detto in parole povere, gli attributi primi sono degli attributi che appartengono a una delle chiavi della relazione.

Questa definizione ci serve per introdurre la terza forma normale.

La definizione di terza forma normale è la seguente: una dipendenza funzionale X determina Y, che appartiene a F, è in 3NF se e solo se si verifica almeno una di queste tre condizioni.

La prima dice che la dipendenza funzionale è banale, cioè il conseguente è un sottoinsieme dell'antecedente, cioè X determina Y è riflessiva.

La seconda condizione dice che l’antecedente è una superchiave, quindi X è una superchiave.

Abbiamo infine una terza condizione che dice che Y, cioè il conseguente della dipendenza funzionale, è composto solo da attributi primi.

Notate il rapporto tra la definizione di terza forma normale e la definizione della forma normale di Boyce-Codd. Infatti, vediamo che i punti 1 e 2 sono in comune con la forma normale di Boyce-Codd. La terza forma normale aggiunge un punto ulteriore, quindi c'è una possibilità in più, possibilità in più perché queste tre condizioni sono in “or” perché deve valere almeno una di queste tre condizioni.

Se aggiungiamo una terza condizione, stiamo ammettendo qualcosa in più che la forma normale di Boyce-Codd non ammetteva, cioè ammettiamo un terzo caso ulteriore che dice che il conseguente è composto da attributi primi.

Quindi è una definizione meno stretta della forma normale di Boyce-Codd: alcune relazioni possono essere in terza forma normale ma non essere nella forma normale di Boyce-Codd.

Ad esempio, possiamo vedere in questa slide la relazione esami che è composta da matricola, codice corso, voto, codice professore, con le due dipendenze funzionali che abbiamo detto qualche slide fa e possiamo verificare che questa relazione è in terza forma normale ma non è in forma normale di Boyce-Codd.

Infatti la prima dipendenza funzionale, cioè “matricola, codice corso determina voto”, è in Boyce-Codd perché l'antecedente è una superchiave, anzi è addirittura una chiave.

La seconda dipendenza funzionale, cioè “codice professore determina codice corso” non è in forma normale di Boyce-Codd perché l’antecedente CP non è una superchiave e inoltre “CP determina CO” non è una dipendenza funzionale riflessiva, ma comunque possiamo notare che il conseguente di questa dipendenza funzionale cioè CO, codice del corso, è un attributo che fa parte di una delle due chiavi della relazione, cioè fa parte della chiave MATR,CO e quindi è un attributo primo.

Quindi questo è un esempio di una relazione che non è in forma normale di Boyce-Codd, ma è in terza forma normale.

Vale il viceversa, cioè dato che la forma normale di Boyce-Codd è più restrittiva, allora se una relazione è in forma normale di Boyce-Codd , sicuramente è anche in terza forma normale.

A differenza della forma normale di Boyce-Codd, una relazione in terza forma normale può presentare ancora delle anomalie.

Vediamo ad esempio la relazione agenzie che ha tre attributi: numero agenzia, città agenzia e direttore. “Numero agenzia, città agenzia” è chiave primaria e ci sono due dipendenze funzionali, una che è quella di chiave primaria, cioè “numero agenzia, città agenzia determinano direttore” e la seconda “direttore determina la città agenzia”, cioè un direttore può lavorare in una sola città e non può dirigere l'agenzia in due città diverse.

Possiamo verificare che questa relazione non è in Boyce-Codd normal form perché, sebbene la prima dipendenza funzionale abbia come antecedente una superchiave, la seconda dipendenza funzionale non è in forma normale di Boyce-Codd perché direttore non è una chiave, ma comunque il conseguente della seconda dipendenza funzionale, cioè città agenzia è un attributo primo perché è uno degli attributi che appartengono alla chiave. Quindi la seconda dipendenza funzionale è in terza forma normale e quindi tutta la relazione è in terza forma normale.

Vediamo un esempio di istanza di questa relazione per fissare le idee.

Le prime due tuple sono dell'agenzia numero 341: una sta a Torino e l'altra sta a Roma, quindi sono due agenzie diverse. Una è diretta da Chiara e la seconda è diretta da Susanna. La terza tupla è l'agenzia 343 di Torino, diretta ancora da Chiara e la quarta è l'agenzia 341 di Torino, diretta da Mario. Riporto sulla slide anche le due dipendenze funzionali così le abbiamo sott'occhio.

Possiamo notare che in questa relazione ci sono delle anomalie. Infatti se voglio aggiungere un direttore, dato che dal direttore dipende la città dell’agenzia, devo aggiungere anche una città dell’agenzia, che però è in chiave e quindi devo inserire obbligatoriamente, quando inserisco un nuovo direttore, anche tutta un'agenzia. Quindi la relazione ha un’anomalia di inserimento simile a quella che avevo detto nell'esempio che avevamo fatto qualche lezione fa sul database degli esami, in cui non potevo inserire un professore, perché avrei dovuto inserire degli esami e non potevo inserire un nuovo studente, perché avrei dovuto inserire anche almeno un suo esame.

Posso avere anche anomalie di cancellazione, perché per esempio se cancello tutte le agenzie di un direttore, per esempio dal database cancello la prima e la terza agenzia (341 di Torino e 343 di Torino), sto cancellando tutte le agenzie della direttrice Chiara e quindi non ho più traccia nel mio database di questa direttrice. Questa è un’anomalia di cancellazione, che è simile all'anomalia di cancellazione che abbiamo visto sull’esempio degli esami in cui, se io avessi cancellato tutti gli esami di un professore, non avrei più avuto traccia del professore nel mio database.

Inoltre, ma questa è una cosa rara, se io cambiassi la città dell'agenzia per un direttore, per effetto della seconda dipendenza funzionale, dovrei aggiornare tutte le agenzie che dirige e questa è un'anomalia di aggiornamento, ma comunque è una anomalia che succede raramente perché io dovrei fare un cambiamento di un attributo che sta in chiave e questo è un cambiamento che è raro e limitato.

Quindi - per concludere questo discorso di introduzione alla terza forma normale - siamo partiti dalla forma normale di Boyce-Codd che eliminava le anomalie e siamo arrivati alla terza forma normale, che invece non riesce a eliminare tutte le anomalie. Abbiamo dovuto farlo perché la forma normale di Boyce-Codd non è sempre raggiungibile conservando le dipendenze.

Quindi da dove arrivano queste anomalie? Derivano da ciò che è diverso tra la terza forma normale e la forma normale di Boyce-Codd, cioè da quelle dipendenze funzionali che noi ammettiamo nella terza forma normale e che non ammetteremmo in quella di Boyce-Codd , cioè le dipendenze funzionali di tipo 3, quelle in cui il conseguente è un attributo primo.

In effetti storicamente lo sviluppo delle forme normali è stato inverso rispetto a quello con cui le ho presentate. È stata definita prima, da Codd, la terza forma normale e poi Codd si è reso conto che questa forma normale ammetteva ancora delle anomalie e ha fatto un passo avanti definendo insieme a Boyce la forma normale di Boyce-Codd. Ma noi le abbiamo presentate al contrario del processo storico, perché è più intuitivo.

Comunque, le anomalie rimaste, quando la terza forma normale non riesce a eliminarle, sono anomalie limitate e di solito tollerabili.

Ora possiamo introdurre l'insieme di copertura minimale, che è il primo passo che dobbiamo percorrere per spiegare qual è il processo di normalizzazione per raggiungere la terza forma normale. Per normalizzare in terza forma normale dobbiamo manipolare le dipendenze funzionali, cioè dobbiamo trasformare le dipendenze funzionali che vengono fornite dal progettista, per avere un insieme diverso, ma equivalente, di dipendenze funzionali, che riusciamo poi a sfruttare nel processo di normalizzazione.

In particolare, dobbiamo trovare un insieme di dipendenze funzionali che sia equivalente a quello di partenza e anche minimale.

Per raggiungere la minimalità introduciamo prima le nozioni di attributo estraneo e di dipendenza ridondante, perché dobbiamo eliminarli per riuscire a raggiungere la minimalità.

Vediamo la definizione di attributo estraneo: un attributo in una dipendenza funzionale in F è estraneo se e solo se possiamo rimuovere questo attributo dalla dipendenza funzionale - quindi non da ogni dipendenza funzionale, ma da una precisa dipendenza funzionale - e continuiamo ad avere un insieme di dipendenze funzionali equivalente a quelle da cui siamo partiti.

Detto in altre parole, un attributo è estraneo, in una dipendenza funzionale, se possiamo eliminarlo dalla dipendenza funzionale e non cambia niente, perché ciò che possiamo rappresentare con il nuovo insieme di dipendenze funzionali è esattamente ciò che potevamo rappresentare con l'insieme di partenza.

Ad esempio, abbiamo le due dipendenze funzionali “ABCD determina E” e “B determina C”. Possiamo verificare che l’attributo C nella prima dipendenza funzionale è estraneo, perché se eliminiamo l'attributo C dalla prima dipendenza funzionale otteniamo una dipendenza funzionale “ABD determina E” e poi abbiamo anche l'altra dipendenza “B determina C”. Se abbiamo nella prima dipendenza funzionale B, applichiamo la seconda dipendenza funzionale e riusciamo a ottenere C, significa che non è necessario avere C nell’antecedente della prima dipendenza funzionale. Quindi eliminando C otteniamo quel secondo insieme F: “ABD determina E” e “B determina C”.

Per verificare se un attributo è estraneo, possiamo sfruttare il meccanismo della chiusura di un insieme di attributi, cioè, dato un insieme di dipendenze funzionali F, considerando una dipendenza funzionale in questo insieme, che sarà fatta come “X (che è un insieme di attributi) determina A”, possiamo dire che l'attributo B in X è estraneo per questa dipendenza funzionale se e solo se la chiusura di X-B in F comprende A.

Perché questa chiusura fatta in questo modo? Perché se noi partiamo dall’antecedente della dipendenza funzionale e rimuoviamo l'attributo che vogliamo verificare se sia estraneo e poi calcoliamo la chiusura, sfruttando tutti i vincoli che sono originariamente in F, e possiamo raggiungere lo stesso A - quindi se possiamo determinare A facendo a meno di B -, significa che B non era necessario per raggiungere A, neanche nella dipendenza funzionale che stiamo esaminando e quindi significa che B è un attributo estraneo.

Ad esempio partiamo dall'insieme di dipendenze funzionali F che comprende due dipendenze funzionali “ABCD determina E” e “B determina C”. Possiamo verificare che C è estraneo nella prima dipendenza funzionale perché se rimuoviamo C e proviamo a calcolare la chiusura di ABD sfruttando queste due dipendenze funzionali, vediamo che, alla chiusura sicuramente per riflessività appartengono A, B e D. Poi da B possiamo determinare C e quindi aggiungiamo anche C. A questo punto abbiamo A,B,C,D che è l'antecedente della prima dipendenza funzionale, che quindi ci permette di ricavare E.

Quindi possiamo dire che E è nella chiusura di A,B,D e quindi l’attributo C è estraneo, perché non è necessario per ricavare E.

Similmente possiamo dire che B, invece, non è estraneo nella prima dipendenza funzionale perché, se rimuoviamo D e quindi calcoliamo la chiusura dell'antecedente cioè A,B,C, vediamo che della chiusura faranno parte A, B e C. Poi la seconda dipendenza funzionale non ci permette di ricavare nuovi attributi perché da B determiniamo C, ma C ce l'abbiamo già nell'insieme e non riusciamo a ricavare E perché ci mancherebbe D e quindi vediamo che D non è estraneo perché, appunto, non appartiene alla chiusura dell'antecedente da cui abbiamo rimosso D stesso.

Abbiamo un concetto molto simile che riguarda le dipendenze. Invece di usare la parola estraneo, usiamo l'aggettivo “ridondante” e diciamo che una dipendenza funzionale è ridondante in un insieme di dipendenze funzionali F se e solo se possiamo rimuovere questa dipendenza da F e continuiamo ad avere un insieme di dipendenze funzionali equivalente.

Anche per questo caso posso sfruttare il meccanismo della chiusura di un insieme di attributi per verificare se una dipendenza è ridondante. In particolare, partendo da un insieme di dipendenze funzionali F la dipendenza funzionale “X determina Y” in F è ridondante se e solo se l'attributo Y appartiene alla chiusura di X. La chiusura non si calcola nell'insieme originale di dipendenze funzionali, ma in quello che ottengo rimuovendo la dipendenza X determina Y.

Qual è il senso di questo? Il senso è che verifico se posso derivare, partendo da X, l'attributo Y dall'insieme F anche senza sfruttare la dipendenza funzionale “X determina Y”.

Se partendo da X riesco a raggiungere Y in F anche se tolgo da F questa dipendenza funzionale, vuol dire che questa dipendenza è ridondante e quindi non mi serve, posso buttarla via, perché tanto riesco a ottenere gli stessi attributi Y.

Ad esempio, consideriamo un insieme di dipendenze funzionali F che è composto da tre dipendenze funzionali: “A determina B”, “B determina C” e “A determina C”.

Posso dire che “A determina C” è ridondante sia perché posso vedere chiaramente che posso derivarlo per transitività, ma anche tramite la chiusura, perché posso verificare che posso partire dall’antecedente e calcolare la chiusura di A usando il nuovo insieme di dipendenze funzionali in cui ho tolto questa dipendenza funzionale che potrebbe essere ridondante. Quindi tengo solo “A determina B” e “B determina C” e vedo che la chiusura di A comprende A stesso per transitività, poi contiene anche B e poi, grazie a B, posso derivare anche C.

Dato che C è contenuto in questa chiusura, vuol dire che non è necessario utilizzare la dipendenza funzionale “A determina C” e quindi posso rimuoverla dall'insieme F avendo un insieme di dipendenze funzionali equivalente a quello da cui sono partito.

Alternativamente posso verificare che “B determina C”, invece, non è ridondante, perché se calcolo la chiusura dell’attributo nell’antecedente della dipendenza funzionale, cioè di B rispetto a F meno “B determina C” e quindi mi rimane “A determina B” e “A determina C”, quindi se calcolo la chiusura di B, nessuna delle due dipendenze funzionali è “attivata” da B, quindi la chiusura contiene B e basta. C non è contenuto nella chiusura, quindi questa dipendenza funzionale non è ridondante.

**17**

Questa lezione è dedicata alla trattazione della terza forma normale e questo è l’outline della lezione.

Inizieremo a introdurre la terza forma normale, poi la definiremo (la abbreviamo come 3NF: terza forma normale in inglese). Poi parleremo di insieme di copertura minimale, che cos'è, lo definiamo e vediamo come ricavarlo. Poi parleremo della normalizzazione in terza forma normale.

Mentre per la BCNF non abbiamo discusso approfonditamente l'algoritmo che serve per partire da una relazione, un database qualunque e decomporlo in relazioni in BCNF, invece per la terza forma normale parliamo anche dell'algoritmo. Infine concludiamo discutendo le proprietà di questo algoritmo.

Per iniziare richiamiamo la BCNF che abbiamo visto nella lezione scorsa. La Boyce-Codd normal form è una forma normale per cui ogni dipendenza funzionale deve rispettare almeno una delle seguenti condizioni: è una dipendenza banale, cioè data la dipendenza funzionale X determina Y che appartiene all’insieme delle dipendenze funzionali, Y è un sottoinsieme di X, quindi è una dipendenza riflessiva, oppure l'antecedente X è una superchiave per la relazione R.

La BCNF è una proprietà molto desiderabile perché permette di definire schemi che eliminano le anomalie di aggiornamento, di inserimento e di cancellazione.

Purtroppo, non è sempre possibile raggiungere la BCNF. In alcuni schemi di database, decomposti per raggiungere la BCNF, non riusciamo a conservare le dipendenze funzionali, perché ci sono alcune dipendenze funzionali che “parlano” di attributi in relazioni diverse e questo, come abbiamo già discusso, non è una cosa desiderabile.

Quindi la BCNF ha un grosso vantaggio perché elimina le anomalie, però non è sempre possibile raggiungerla conservando le dipendenze.

A questo punto dobbiamo decidere se ci interessa di più conservare le dipendenze oppure non avere le anomalie. La scelta che si fa di solito è di conservare le dipendenze, perché è più importante avere efficienza nel controllo dei vincoli, quindi avere vincoli che non richiedono necessariamente sempre di effettuare un join.

Quindi è stata definita una ulteriore forma normale che si chiama terza forma normale. Si chiama terza forma normale perché ne esistono storicamente altre due prima, però noi parliamo solo della terza forma normale. Abbiamo fatto qualche cenno sulla prima, invece della seconda non abbiamo parlato, perché ha solo interesse storico.

Quindi definiamo una nuova forma normale, la terza forma normale che è un po’ più debole, nel senso che non permette di eliminare tutte le anomalie, però ha un vantaggio: è sempre possibile raggiungere la terza forma normale conservando le dipendenze funzionali, quindi accettiamo un compromesso rispetto a quello che ci darebbe la BCNF. Adesso possiamo definire cos'è la terza forma normale.

Prima di definire cos'è la terza forma normale dobbiamo introdurre la definizione di attributo primo. Partiamo dalla relazione che vediamo sulla slide, la relazione esami, che ha, in questa versione, solo quattro attributi: matricola studente, codice corso, voto studente e codice professore con le dipendenze funzionali “matricola, corso determinano voto” e “codice professore determina codice corso”.

La relazione ha quindi come chiavi candidate “matricola, corso” e “matricola, codice professore”. Potete verificare che sono chiavi candidate calcolando la chiusura di questi due insiemi di attributi e quindi constatare che comprende tutti gli attributi della relazione e poi verificare che sono minimali provando a togliere matricola o codice corso o codice professore e verificare che queste non risultano essere superchiavi.

Quindi abbiamo questa relazione, con queste due chiavi: “matricola, corso” e “matricola, codice professore”. Codice corso, da solo, non è una chiave primaria, non è neanche una chiave candidata e non è neanche una superchiave, ma è un attributo comunque contenuto in una chiave della relazione esami. Codice corso è un attributo primo per la relazione esami.

Possiamo ora definire che cos'è un attributo primo: data una relazione R con schema A, gli attributi Y, che sono un sottoinsieme di A, sono detti attributi primi se e solo se sono un sottoinsieme di K, dove K è una chiave della relazione R. Quindi, detto in parole povere, gli attributi primi sono degli attributi che appartengono a una delle chiavi della relazione.

Questa definizione ci serve per introdurre la terza forma normale.

La definizione di terza forma normale è la seguente: una dipendenza funzionale X determina Y, che appartiene a F, è in 3NF se e solo se si verifica almeno una di queste tre condizioni.

La prima dice che la dipendenza funzionale è banale, cioè il conseguente è un sottoinsieme dell'antecedente, cioè X determina Y è riflessiva.

La seconda condizione dice che l’antecedente è una superchiave, quindi X è una superchiave.

Abbiamo infine una terza condizione che dice che Y, cioè il conseguente della dipendenza funzionale, è composto solo da attributi primi.

Notate il rapporto tra la definizione di terza forma normale e la definizione della forma normale di Boyce-Codd. Infatti, vediamo che i punti 1 e 2 sono in comune con la forma normale di Boyce-Codd. La terza forma normale aggiunge un punto ulteriore, quindi c'è una possibilità in più, possibilità in più perché queste tre condizioni sono in “or” perché deve valere almeno una di queste tre condizioni.

Se aggiungiamo una terza condizione, stiamo ammettendo qualcosa in più che la forma normale di Boyce-Codd non ammetteva, cioè ammettiamo un terzo caso ulteriore che dice che il conseguente è composto da attributi primi.

Quindi è una definizione meno stretta della forma normale di Boyce-Codd: alcune relazioni possono essere in terza forma normale ma non essere nella forma normale di Boyce-Codd.

Ad esempio, possiamo vedere in questa slide la relazione esami che è composta da matricola, codice corso, voto, codice professore, con le due dipendenze funzionali che abbiamo detto qualche slide fa e possiamo verificare che questa relazione è in terza forma normale ma non è in forma normale di Boyce-Codd.

Infatti la prima dipendenza funzionale, cioè “matricola, codice corso determina voto”, è in Boyce-Codd perché l'antecedente è una superchiave, anzi è addirittura una chiave.

La seconda dipendenza funzionale, cioè “codice professore determina codice corso” non è in forma normale di Boyce-Codd perché l’antecedente CP non è una superchiave e inoltre “CP determina CO” non è una dipendenza funzionale riflessiva, ma comunque possiamo notare che il conseguente di questa dipendenza funzionale cioè CO, codice del corso, è un attributo che fa parte di una delle due chiavi della relazione, cioè fa parte della chiave MATR,CO e quindi è un attributo primo.

Quindi questo è un esempio di una relazione che non è in forma normale di Boyce-Codd, ma è in terza forma normale.

Vale il viceversa, cioè dato che la forma normale di Boyce-Codd è più restrittiva, allora se una relazione è in forma normale di Boyce-Codd , sicuramente è anche in terza forma normale.

A differenza della forma normale di Boyce-Codd, una relazione in terza forma normale può presentare ancora delle anomalie.

Vediamo ad esempio la relazione agenzie che ha tre attributi: numero agenzia, città agenzia e direttore. “Numero agenzia, città agenzia” è chiave primaria e ci sono due dipendenze funzionali, una che è quella di chiave primaria, cioè “numero agenzia, città agenzia determinano direttore” e la seconda “direttore determina la città agenzia”, cioè un direttore può lavorare in una sola città e non può dirigere l'agenzia in due città diverse.

Possiamo verificare che questa relazione non è in Boyce-Codd normal form perché, sebbene la prima dipendenza funzionale abbia come antecedente una superchiave, la seconda dipendenza funzionale non è in forma normale di Boyce-Codd perché direttore non è una chiave, ma comunque il conseguente della seconda dipendenza funzionale, cioè città agenzia è un attributo primo perché è uno degli attributi che appartengono alla chiave. Quindi la seconda dipendenza funzionale è in terza forma normale e quindi tutta la relazione è in terza forma normale.

Vediamo un esempio di istanza di questa relazione per fissare le idee.

Le prime due tuple sono dell'agenzia numero 341: una sta a Torino e l'altra sta a Roma, quindi sono due agenzie diverse. Una è diretta da Chiara e la seconda è diretta da Susanna. La terza tupla è l'agenzia 343 di Torino, diretta ancora da Chiara e la quarta è l'agenzia 341 di Torino, diretta da Mario. Riporto sulla slide anche le due dipendenze funzionali così le abbiamo sott'occhio.

Possiamo notare che in questa relazione ci sono delle anomalie. Infatti se voglio aggiungere un direttore, dato che dal direttore dipende la città dell’agenzia, devo aggiungere anche una città dell’agenzia, che però è in chiave e quindi devo inserire obbligatoriamente, quando inserisco un nuovo direttore, anche tutta un'agenzia. Quindi la relazione ha un’anomalia di inserimento simile a quella che avevo detto nell'esempio che avevamo fatto qualche lezione fa sul database degli esami, in cui non potevo inserire un professore, perché avrei dovuto inserire degli esami e non potevo inserire un nuovo studente, perché avrei dovuto inserire anche almeno un suo esame.

Posso avere anche anomalie di cancellazione, perché per esempio se cancello tutte le agenzie di un direttore, per esempio dal database cancello la prima e la terza agenzia (341 di Torino e 343 di Torino), sto cancellando tutte le agenzie della direttrice Chiara e quindi non ho più traccia nel mio database di questa direttrice. Questa è un’anomalia di cancellazione, che è simile all'anomalia di cancellazione che abbiamo visto sull’esempio degli esami in cui, se io avessi cancellato tutti gli esami di un professore, non avrei più avuto traccia del professore nel mio database.

Inoltre, ma questa è una cosa rara, se io cambiassi la città dell'agenzia per un direttore, per effetto della seconda dipendenza funzionale, dovrei aggiornare tutte le agenzie che dirige e questa è un'anomalia di aggiornamento, ma comunque è una anomalia che succede raramente perché io dovrei fare un cambiamento di un attributo che sta in chiave e questo è un cambiamento che è raro e limitato.

Quindi - per concludere questo discorso di introduzione alla terza forma normale - siamo partiti dalla forma normale di Boyce-Codd che eliminava le anomalie e siamo arrivati alla terza forma normale, che invece non riesce a eliminare tutte le anomalie. Abbiamo dovuto farlo perché la forma normale di Boyce-Codd non è sempre raggiungibile conservando le dipendenze.

Quindi da dove arrivano queste anomalie? Derivano da ciò che è diverso tra la terza forma normale e la forma normale di Boyce-Codd, cioè da quelle dipendenze funzionali che noi ammettiamo nella terza forma normale e che non ammetteremmo in quella di Boyce-Codd , cioè le dipendenze funzionali di tipo 3, quelle in cui il conseguente è un attributo primo.

In effetti storicamente lo sviluppo delle forme normali è stato inverso rispetto a quello con cui le ho presentate. È stata definita prima, da Codd, la terza forma normale e poi Codd si è reso conto che questa forma normale ammetteva ancora delle anomalie e ha fatto un passo avanti definendo insieme a Boyce la forma normale di Boyce-Codd. Ma noi le abbiamo presentate al contrario del processo storico, perché è più intuitivo.

Comunque, le anomalie rimaste, quando la terza forma normale non riesce a eliminarle, sono anomalie limitate e di solito tollerabili.

Ora possiamo introdurre l'insieme di copertura minimale, che è il primo passo che dobbiamo percorrere per spiegare qual è il processo di normalizzazione per raggiungere la terza forma normale. Per normalizzare in terza forma normale dobbiamo manipolare le dipendenze funzionali, cioè dobbiamo trasformare le dipendenze funzionali che vengono fornite dal progettista, per avere un insieme diverso, ma equivalente, di dipendenze funzionali, che riusciamo poi a sfruttare nel processo di normalizzazione.

In particolare, dobbiamo trovare un insieme di dipendenze funzionali che sia equivalente a quello di partenza e anche minimale.

Per raggiungere la minimalità introduciamo prima le nozioni di attributo estraneo e di dipendenza ridondante, perché dobbiamo eliminarli per riuscire a raggiungere la minimalità.

Vediamo la definizione di attributo estraneo: un attributo in una dipendenza funzionale in F è estraneo se e solo se possiamo rimuovere questo attributo dalla dipendenza funzionale - quindi non da ogni dipendenza funzionale, ma da una precisa dipendenza funzionale - e continuiamo ad avere un insieme di dipendenze funzionali equivalente a quelle da cui siamo partiti.

Detto in altre parole, un attributo è estraneo, in una dipendenza funzionale, se possiamo eliminarlo dalla dipendenza funzionale e non cambia niente, perché ciò che possiamo rappresentare con il nuovo insieme di dipendenze funzionali è esattamente ciò che potevamo rappresentare con l'insieme di partenza.

Ad esempio, abbiamo le due dipendenze funzionali “ABCD determina E” e “B determina C”. Possiamo verificare che l’attributo C nella prima dipendenza funzionale è estraneo, perché se eliminiamo l'attributo C dalla prima dipendenza funzionale otteniamo una dipendenza funzionale “ABD determina E” e poi abbiamo anche l'altra dipendenza “B determina C”. Se abbiamo nella prima dipendenza funzionale B, applichiamo la seconda dipendenza funzionale e riusciamo a ottenere C, significa che non è necessario avere C nell’antecedente della prima dipendenza funzionale. Quindi eliminando C otteniamo quel secondo insieme F: “ABD determina E” e “B determina C”.

Per verificare se un attributo è estraneo, possiamo sfruttare il meccanismo della chiusura di un insieme di attributi, cioè, dato un insieme di dipendenze funzionali F, considerando una dipendenza funzionale in questo insieme, che sarà fatta come “X (che è un insieme di attributi) determina A”, possiamo dire che l'attributo B in X è estraneo per questa dipendenza funzionale se e solo se la chiusura di X-B in F comprende A.

Perché questa chiusura fatta in questo modo? Perché se noi partiamo dall’antecedente della dipendenza funzionale e rimuoviamo l'attributo che vogliamo verificare se sia estraneo e poi calcoliamo la chiusura, sfruttando tutti i vincoli che sono originariamente in F, e possiamo raggiungere lo stesso A - quindi se possiamo determinare A facendo a meno di B -, significa che B non era necessario per raggiungere A, neanche nella dipendenza funzionale che stiamo esaminando e quindi significa che B è un attributo estraneo.

Ad esempio partiamo dall'insieme di dipendenze funzionali F che comprende due dipendenze funzionali “ABCD determina E” e “B determina C”. Possiamo verificare che C è estraneo nella prima dipendenza funzionale perché se rimuoviamo C e proviamo a calcolare la chiusura di ABD sfruttando queste due dipendenze funzionali, vediamo che, alla chiusura sicuramente per riflessività appartengono A, B e D. Poi da B possiamo determinare C e quindi aggiungiamo anche C. A questo punto abbiamo A,B,C,D che è l'antecedente della prima dipendenza funzionale, che quindi ci permette di ricavare E.

Quindi possiamo dire che E è nella chiusura di A,B,D e quindi l’attributo C è estraneo, perché non è necessario per ricavare E.

Similmente possiamo dire che B, invece, non è estraneo nella prima dipendenza funzionale perché, se rimuoviamo D e quindi calcoliamo la chiusura dell'antecedente cioè A,B,C, vediamo che della chiusura faranno parte A, B e C. Poi la seconda dipendenza funzionale non ci permette di ricavare nuovi attributi perché da B determiniamo C, ma C ce l'abbiamo già nell'insieme e non riusciamo a ricavare E perché ci mancherebbe D e quindi vediamo che D non è estraneo perché, appunto, non appartiene alla chiusura dell'antecedente da cui abbiamo rimosso D stesso.

Abbiamo un concetto molto simile che riguarda le dipendenze. Invece di usare la parola estraneo, usiamo l'aggettivo “ridondante” e diciamo che una dipendenza funzionale è ridondante in un insieme di dipendenze funzionali F se e solo se possiamo rimuovere questa dipendenza da F e continuiamo ad avere un insieme di dipendenze funzionali equivalente.

Anche per questo caso posso sfruttare il meccanismo della chiusura di un insieme di attributi per verificare se una dipendenza è ridondante. In particolare, partendo da un insieme di dipendenze funzionali F la dipendenza funzionale “X determina Y” in F è ridondante se e solo se l'attributo Y appartiene alla chiusura di X. La chiusura non si calcola nell'insieme originale di dipendenze funzionali, ma in quello che ottengo rimuovendo la dipendenza X determina Y.

Qual è il senso di questo? Il senso è che verifico se posso derivare, partendo da X, l'attributo Y dall'insieme F anche senza sfruttare la dipendenza funzionale “X determina Y”.

Se partendo da X riesco a raggiungere Y in F anche se tolgo da F questa dipendenza funzionale, vuol dire che questa dipendenza è ridondante e quindi non mi serve, posso buttarla via, perché tanto riesco a ottenere gli stessi attributi Y.

Ad esempio, consideriamo un insieme di dipendenze funzionali F che è composto da tre dipendenze funzionali: “A determina B”, “B determina C” e “A determina C”.

Posso dire che “A determina C” è ridondante sia perché posso vedere chiaramente che posso derivarlo per transitività, ma anche tramite la chiusura, perché posso verificare che posso partire dall’antecedente e calcolare la chiusura di A usando il nuovo insieme di dipendenze funzionali in cui ho tolto questa dipendenza funzionale che potrebbe essere ridondante. Quindi tengo solo “A determina B” e “B determina C” e vedo che la chiusura di A comprende A stesso per transitività, poi contiene anche B e poi, grazie a B, posso derivare anche C.

Dato che C è contenuto in questa chiusura, vuol dire che non è necessario utilizzare la dipendenza funzionale “A determina C” e quindi posso rimuoverla dall'insieme F avendo un insieme di dipendenze funzionali equivalente a quello da cui sono partito.

Alternativamente posso verificare che “B determina C”, invece, non è ridondante, perché se calcolo la chiusura dell’attributo nell’antecedente della dipendenza funzionale, cioè di B rispetto a F meno “B determina C” e quindi mi rimane “A determina B” e “A determina C”, quindi se calcolo la chiusura di B, nessuna delle due dipendenze funzionali è “attivata” da B, quindi la chiusura contiene B e basta. C non è contenuto nella chiusura, quindi questa dipendenza funzionale non è ridondante.

**18**

Ora siamo pronti per descrivere l'algoritmo per normalizzare una relazione e renderla in terza forma normale.

Vediamo prima ad alto livello come funziona all'algoritmo di normalizzazione e poi nella prossima slide vedremo più nel dettaglio ogni passo.

Partiamo da una relazione R su uno schema A che ha come insieme di dipendenze funzionali F.

Come primo passo calcoliamo la copertura minimale F’ di F, con l'algoritmo che abbiamo visto poco fa.

Al secondo passo decomponiamo R. La decomposizione funziona in questo modo: prendiamo ogni dipendenza funzionale che sta in F’ e costruiamo una nuova relazione per ognuna di queste dipendenze funzionali. Se ci sono più dipendenze funzionali che hanno lo stesso antecedente, queste daranno luogo alla stessa relazione.

Al terzo e al quarto passo “aggiustiamo” questa decomposizione che abbiamo creato. Al terzo passo cerchiamo se ci siano delle relazioni che sono un sottoinsieme di altre relazioni dal punto di vista dello schema, cioè i cui attributi sono contenuti negli attributi di altre relazioni. Se succede, eliminiamo le relazioni contenute.

Al quarto passo cerchiamo se una relazione, tra quelle che abbiamo aggiunto, contiene una chiave di R e se nessuna di esse contiene una chiave di R, aggiungiamo un'altra nuova relazione che la contiene.

Ora vediamo nel dettaglio l’algoritmo di normalizzazione in terza forma normale. Quindi partiamo da una relazione R con un insieme di dipendenze funzionali F.

Primo passo: calcoliamo la copertura minimale F’ di F. Come risultato del primo passo avremo un insieme di dipendenze funzionali in forma canonica, cioè nel loro conseguente c'è un solo attributo e avremo eliminato gli attributi estranei e le dipendenze ridondanti.

Al secondo passo dell'algoritmo consideriamo tutte le dipendenze funzionali che hanno lo stesso antecedente e prendiamo un insieme per volta considerando tutte le dipendenze funzionali che hanno lo stesso antecedente. Gli insiemi saranno fatti in questo modo: X determina A1, X determina A2, fino a X determina An. Se c'è una sola dipendenza con un determinato antecedente, ovviamente sarà un insieme che contiene una singola dipendenza funzionale. Per ogni insieme di dipendenze funzionali creiamo una nuova relazione. Questa nuova relazione, che chiamiamo - per ricordarci a cosa corrisponde - RX, conterrà gli attributi che stanno a sinistra, quindi X, e anche gli attributi che stanno a destra nelle varie dipendenze funzionali cioè A1, A2, …, An. Associate a questa relazione ci saranno le dipendenze funzionali che raggruppiamo di nuovo assieme e quindi torniamo indietro dalla forma canonica, X determina A1, A2, …, An.

Nel passo successivo “aggiustiamo” questa decomposizione che abbiamo già operato, cercando le relazioni che sono sottoinsiemi di altre relazioni, cioè consideriamo ogni coppia di relazioni RX con gli attributi X, Y dove XY sono insiemi di attributi e delle relazioni RX’ con gli attributi X’, Y’, in cui vediamo che X, Y sono un sovrainsieme di X’,Y’: quindi si può dire che la relazione RX’ è un sottoinsieme della relazione RX dal punto di vista dello schema perché tutti gli attributi di RX’ sono contenuti negli attributi di RX.

Se troviamo delle relazioni fatte in questo modo, eliminiamo la relazione RX’, cioè quella più piccola, quella contenuta e, per non perdere le dipendenze funzionali che, comunque, apportano delle informazioni al nostro database, prendiamo tutte le dipendenze funzionali di RX’ e le aggiungiamo a quelle di RX.

Quindi il passo 3 serve per eliminare le relazioni che sono contenute nelle altre relazioni e facciamo attenzione a non perdere le dipendenze, perché, se buttassimo via le dipendenze di R(X)’, la decomposizione finale avrebbe un’evoluzione diversa da quella della relazione da cui siamo partiti e questo non lo vogliamo: vogliamo che sia equivalente.

Passo 4: cerchiamo se c'è almeno una relazione che contiene una chiave K di R, la relazione da cui siamo partiti. Controlliamo ogni relazione che abbiamo creato e vediamo se ce n'è almeno una che contiene tutta la chiave K della relazione R. Non per forza la chiave primaria, basta che contenga una qualunque chiave di R. Se succede allora va bene e non facciamo niente. Se invece non c'è nessuna relazione della decomposizione che contiene la chiave di R, dobbiamo crearne una nuova.

Ne troviamo una nuova, quindi creiamo una nuova relazione che chiamiamo RK, che contiene tutti e soli gli attributi che fanno parte della chiave.

Vediamo un esempio di funzionamento dell'algoritmo in cui decomponiamo la relazione Esami che contiene matricola, nome studente, data di nascita, codice del corso, voto, data esame, codice professore, nome professore. Le dipendenze funzionali sono: “matricola, corso determinano voto, data esame e codice professore”, “matricola determina nome studente e data di nascita dello studente”, “codice professore determina nome del professore”.

Passo 1: calcoliamo la copertura minimale F’ di F. Scriviamo le dipendenze funzionali in forma canonica e abbiamo l’F che vediamo sulla slide. Possiamo esaminare poi queste dipendenze funzionali in forma canonica e notare che nessuna di queste ha attributi estranei. Questo potete farlo come esercizio.

Notiamo anche che non ci sono dipendenze funzionali ridondanti e anche questo potete farlo come esercizio. Quindi F è già una copertura minimale.

Possiamo passare al passo 2 dell'algoritmo, in cui effettuiamo la decomposizione vera e propria, cioè prendiamo ogni insieme di dipendenze funzionali che ha lo stesso antecedente e creiamo una nuova relazione con questi attributi. Quindi abbiamo due dipendenze funzionali nell'insieme di copertura minimale che hanno a sinistra l’attributo MATR e sono “MATR determina NS” e “MATR determina DN”.

Da queste ottengo la prima relazione di decomposizione, che qua chiamo R1, ma potrei chiamare anche RMATR, che contiene MATR a sinistra e due attributi a destra, cioè NS e DN. Eredito le due dipendenze funzionali e quindi MATR è una chiave primaria di R1 (perché le due dipendenze funzionali “MATR determina NS” e “MATR determina DN” equivalgono a dire che da MATR posso derivare tutti gli attributi e cioè MATR è una superchiave e, dato che è composta un solo attributo, sarà anche una chiave). Non scrivo le dipendenze funzionali perché posso semplicemente sottolineare l'attributo MATR per dire che MATR è una chiave primaria, e in questo modo sto rappresentando anche queste dipendenze funzionali.

C'è un altro insieme di attributi di dipendenze funzionali che hanno come attributi a sinistra MATR e Co: sono “MATR,Co determina voto”, “MATR,Co determina data esame” e “MATR,Co determina codice professore”. Queste tre dipendenze sono nello stesso insieme e danno luogo a una nuova relazione. Questa relazione, che qua chiamiamo R2, conterrà i due attributi che stanno a sinistra, cioè MATR e Co, che saranno chiave primaria, e poi i tre attributi che stanno a destra, che sono il voto, la data dell'esame e il codice professore.

Poi abbiamo l'ultima dipendenza funzionale che non abbiamo ancora considerato, che è un insieme che contiene una singola dipendenza funzionale cioè “CP determina NP”. Non c'è nessun'altra dipendenza funzionale che ha a sinistra CP, quindi questa sarà da sola e darà luogo alla relazione R3, che ha i due attributi CP,NP, con CP che è chiave primaria.

Poi eseguo il passo 3 dell'algoritmo, in cui cerco se esistono relazioni i cui attributi sono contenuti in altre relazioni e posso notare che delle tre che ho creato non ce n’è nessuna che è contenuta in un’altra, quindi posso passare a eseguire il punto 4 dell'algoritmo, in cui cerco se c'è almeno una relazione che contiene una chiave K della relazione di partenza.

La relazione di partenza era la relazione Esami e noi sappiamo già, perché lo abbiamo già discusso, che una chiave per Esami è composta da MATR,CO, cioè la matricola dello studente e il codice del corso dell'esame che è stato superato. Posso notare che la relazione R2 contiene proprio quella chiave, quindi non ho bisogno di aggiungere una nuova relazione.

Ottengo questo risultato in cui ho decomposto la relazione di partenza Esami nelle tre relazioni R1,R2,R3. R1 ha la matricola dello studente, che è chiave primaria, poi il nome dello studente e la data di nascita dello studente. R2 ha come chiave primaria la matricola dello studente e il codice del corso e gli altri attributi sono il voto, la data dell’esame e il codice professore. R3 ha come chiave primaria il codice del professore e come altro attributo il nome del professore.

Non è necessario che rappresenti nessuna dipendenza funzionale perché implicitamente sono già rappresentate dalle chiavi primarie, cioè in R1 c'è già sicuramente una dipendenza funzionale che dice che da MATR determino NS e DN, quindi non è necessario che io aggiunga questa perché è ovvia. Se controllo queste relazioni, posso constatare che sono tutte di tipo 2, cioè ogni relazione ha una dipendenza funzionale in cui l'antecedente è una superchiave della relazione stessa, quindi R1 ha questa dipendenza funzionale implicita che dice che dalla chiave primaria determino tutti gli attributi che ovviamente è di tipo 2, della Boyce-Codd normal form, perché l'antecedente è una superchiave, anzi è proprio una chiave.

Per R2 e R3 è la stessa cosa, quindi questo schema che abbiamo creato non è solo in terza forma normale, ma è anche in Boyce-Codd normal form.

Vediamo un ulteriore esempio in cui vogliamo decomporre CC, cioè conto corrente, che ha attributi titolare, numero conto, numero agenzia, città agenzia e saldo, con le dipendenze funzionali F, che dicono che “numero conto determina numero agenzia”, “numero conto determina città agenzia” e “numero conto determina saldo”.

Passo 1 dell'algoritmo: cerchiamo una copertura minimale di F e come esercizio potete verificare che F è già una copertura minimale.

Passo 2 dell'algoritmo: decomponiamo e vediamo che c'è un unico insieme di dipendenze funzionali perché tutte e tre hanno lo stesso attributo a sinistra, quindi danno luogo alla stessa relazione che chiamiamo R1, che ha come attributi numero conto, che è quello nell’antecedente e quindi diventa anche chiave primaria e poi altri tre attributi che sono numero agenzia, città agenzia e saldo.

Passo 3 dell'algoritmo: cerchiamo delle sottorelazioni, ma dato che ce n'è una sola non ha senso cercarle. Passo 4 dell'algoritmo: consideriamo se le relazioni che abbiamo generato contengono una chiave della relazione di partenza. Possiamo vedere che nella relazione di partenza c'è l'attributo titolare che non compare in R1, quindi sicuramente non c'è in R1 la chiave della relazione di partenza, di CC, perché se ci fosse dovrebbe riuscire a generare tramite chiusura anche l'attributo titolare, ma è impossibile che lo faccia perché non è presente in R1. Quindi sicuramente in R1 non c'è la chiave di CC.

Quindi deriviamo la chiave di CC, come abbiamo già fatto come esercizio in qualche lezione passata. Abbiamo visto che la chiave è titolare e conto, quindi aggiungiamo una nuova relazione R2 che contiene solo la chiave, cioè titolare e conto e tutte e due sono chiave primaria di questa nuova relazione R2.

Vediamo ancora un esempio con la relazione CC che contiene gli attributi titolare, numero conto, numero agenzia, città agenzia, saldo, direttore, qualifica e stipendio. I vincoli di dipendenza funzionale che insistono su questa relazione sono quelli in F’, cioè dal numero conto possiamo derivare il numero dell'agenzia, la città dell'agenzia e il saldo; da numero agenzia e città agenzia possiamo derivare il direttore; dalla qualifica deriviamo stipendio e dal direttore deriviamo la città dell'agenzia. Se guardate queste dipendenze funzionali, potete verificare che sono già una copertura minimale, perché sono in forma canonica, non ci sono attributi estranei e non ci sono dipendenze funzionali ridondanti. Adesso ricaviamo la terza forma normale.

Nel primo passo raccogliamo una copertura minimale e abbiamo detto che è già minimale.

Secondo passo costruiamo una relazione per ogni dipendenza funzionale, facendo attenzione a raggruppare tutte le dipendenze che hanno lo stesso antecedente nella stessa relazione e quindi abbiamo R1 che ha il numero conto che è chiave primaria, poi numero agenzia, città agenzia e saldo. R2 che ha numero agenzia e città agenzia in chiave primaria e poi direttore. R3 che ha qualifica come chiave primaria e stipendio. R4 che ha direttore come chiave primaria e città agenzia.

Al terzo passo verifichiamo se una relazione è un sottoinsieme di un'altra relazione e possiamo vedere che R4, che ha come attributi direttore e città agenzia, ha uno schema che è un sottoinsieme di R2, che in effetti ha città agenzia, direttore e in più anche numero agenzia. Quindi possiamo eliminare R4 e aggiungiamo le dipendenze funzionali di R4 a R2. La dipendenza funzionale è quella di chiave primaria, cioè da direttore io ricavo città agenzia e quindi R2 sarà quella di prima, con in più la nuova dipendenza funzionale. Quindi R2 sarà una relazione che ha la sua dipendenza funzionale di chiave primaria, con in più questa dipendenza funzionale: “direttore determina agenzia”.

Nel passo 4 dobbiamo verificare se almeno una di queste relazioni che abbiamo ottenuto contiene una chiave per la relazione che cerchiamo di decomporre, cioè per la relazione CC. Nella relazione CC possiamo notare che c'è un attributo che non viene nominato da nessuna di queste quattro relazioni ed è l’attributo titolare, quindi sicuramente nessuna di queste quattro relazioni ha una chiave della relazione CC, perché nessuna di queste quattro relazioni permette di raggiungere, tramite le dipendenze funzionali, l'attributo titolare.

Quindi dobbiamo aggiungere una nuova relazione che contiene la chiave della relazione CC e possiamo verificare che è titolare, numero conto e qualifica e quindi aggiungiamo una nuova relazione che chiamiamo R5 che è composta da titolare, numero conto e qualifica, dove tutti e tre gli attributi sono in chiave primaria.

Ora possiamo discutere le proprietà della normalizzazione in terza forma normale.

L'algoritmo di normalizzazione in terza forma normale gode delle seguenti proprietà: prima di tutto nelle relazioni che vengono generate dalle dipendenze funzionali “X determina Ai”, X è una chiave della relazione.

L'algoritmo genera relazioni che sono effettivamente in terza forma normale, quindi fa esattamente ciò che vogliamo che faccia: è corretto.

L'algoritmo ha complessità polinomiale e le decomposizioni che genera godono delle due proprietà importanti che abbiamo discusso la lezione scorsa, cioè la decomposizione conserva le dipendenze e garantisce che la decomposizione sia senza perdita.

Nelle prossime slide, per semplicità, ignoriamo alcune delle ottimizzazioni che l'algoritmo introduce, cioè supponiamo che la decomposizione abbia la forma R1(X1,A1),R2(X2,A2), fino a Rn(Xn,An), cioè non consideriamo che l'algoritmo raggruppa le dipendenze funzionali diverse che hanno lo stesso antecedente, generando una sola relazione, e per semplicità supponiamo che non faccia quel passo che è un’ottimizzazione. Poi c'è l'ultima relazione che non è detto venga aggiunta, cioè Rn+1, che contiene K, dove K è una chiave della relazione originale.

La prima proprietà, che chiamiamo lemma perché ci serve per dimostrare le altre proprietà, dice che X è una chiave di R(X). Cioè, data una qualsiasi relazione che chiamiamo Ri che comprende gli attributi Xi e Ai, se questa relazione è stata generata a partire da una dipendenza funzionale “Xi determina Ai”, che viene scelta tra le dipendenze funzionali che sono nella copertura minimale F’, allora Xi è una chiave di Ri.

La dimostrazione inizia dimostrando che Xi è una superchiave e poi prosegue dimostrando che non è solo una superchiave ma è anche una chiave.

Perché Xi è una chiave? Perché se vale la dipendenza funzionale Xi determina Ai, allora ovviamente nella chiusura di Xi troviamo sia Xi stesso, per riflessività, che l'attributo Ai e quindi possiamo ricavare tutti gli attributi della relazione nella chiusura, quindi Xi è una superchiave di Ri.

Adesso facciamo un passo ulteriore e dimostriamo che è una superchiave minimale e cioè una chiave e lo facciamo per assurdo, assumendo che sia una superchiave non minimale. Se non è minimale, allora esiste un suo sottoinsieme proprio W che è una chiave per Ri, se è una chiave per Ri vuol dire che se calcoliamo la chiusura di W otteniamo tutti gli attributi di Ri, cioè XiAi e questo significa che gli attributi che sono in Xi ma non sono in W sono degli attributi estranei per la dipendenza funzionale “Xi determina Ai”. Perché sono attributi estranei? Perché se consideriamo che nella chiusura di W troviamo XiAi allora sicuramente a maggior ragione troviamo Ai, cioè da W possiamo derivare, tramite le dipendenze funzionali F’, Ai. Quindi per derivare Ai è sufficiente usare W e non tutto X, perché appunto W è un sottoinsieme proprio, cioè la dipendenza funzionale “W determina Ai” appartiene alla chiusura di F’. Ma se c'è almeno un attributo estraneo nella dipendenza Xi determina Ai, allora siamo in contraddizione con l’ipotesi che questa dipendenza funzionale “Xi determina Ai” appartiene alla copertura minimale, perché, se appartenesse alla copertura minimale, tutti gli attributi estranei sarebbero già stati rimossi. Quindi abbiamo trovato una contraddizione e quindi Xi è una superchiave minimale, cioè una chiave.

La proprietà successiva dice che l'algoritmo è in effetti corretto, nel senso che gli schemi che genera sono schemi in terza forma normale.

La dimostrazione procede per assurdo, cioè assumiamo che una delle relazioni Ri, formata dagli attributii XiAi, con Xi chiave primaria, non sia in terza forma normale e cioè che Ri ci sia almeno una dipendenza funzionale che non è in terza forma normale. Questa dipendenza funzionale sarà del tipo Y determina Bj con Y e Bj che fanno parte dello schema della relazione. Prima di tutto Bj deve essere uguale a Ai perché se Bj non fosse uguale ad Ai, dato che deve fare parte degli attributi della relazione, dovrebbe appartenere a Xi, ma se appartenesse a Xi allora sarebbe un attributo primo proprio per la definizione di attributo primo e quindi sarebbe una dipendenza funzionale in terza forma normale, mentre stiamo assumendo per assurdo che non sia in terza forma normale.

Quindi ora sappiamo che questa dipendenza funzionale che viola la terza forma normale è di tipo “Y determina Ai”.

Adesso esaminiamo l'antecedente Y: Y deve essere diverso da Xi perché altrimenti la dipendenza sarebbe ridondante, cioè, se fosse esattamente uguale a Xi, allora la chiave primaria implicherebbe già una dipendenza funzionale “Xi determina Ai” e da qui troviamo la ridondanza, che è un assurdo, perché stiamo assumendo che le dipendenze funzionali appartengano alla copertura minimale.

Quindi per costruzione Y deve essere un sottoinsieme proprio di Xi, ma dato che Y determina Ai, allora da Y posso derivare tutti gli attributi della relazione. Cioè Y sarebbe una chiave per Ri ma se Y è un sottoinsieme proprio di Xi ed è una chiave per Ri allora Xi non può essere una chiave e sarà solo una superchiave. Questo è assurdo perché abbiamo già dimostrato che Xi deve essere una chiave, cioè è minimale.

L'algoritmo di normalizzazione in terza forma normale, in effetti spesso genera degli schemi che sono anche più “forti”, non presentano anomalie perché sono in Boyce-Codd normal form.

Infatti, se rivediamo l'algoritmo, notiamo che il primo passo dell'algoritmo ricava una copertura minimale e poi il secondo passo genera degli schemi di relazione del tipo R(XiAi) dove Xi è la chiave e che hanno la dipendenza funzionale “Xi determina Ai”, quindi il passo 2 può generare sono relazioni in Boyce-Codd normal form, perché l'antecedente delle dipendenze funzionali è una superchiave, anzi non è solo una superchiave ma è addirittura una chiave.

Il passo 4, se non trova una chiave della relazione di partenza, aggiunge una nuova relazione RK dove K è la chiave sia della relazione di partenza, sia di questa nuova relazione che stiamo aggiungendo. Anche questa relazione è in Boyce-Codd normal form, perché non ha dipendenze funzionali che non siano banali, cioè da K, la chiave, ricaviamo K stesso.

Quindi l'unico passo che può generare delle dipendenze funzionali che non sono in Boyce-Codd normal form è il passo 3, cioè quello in cui cerchiamo se una relazione è contenuta in un’altra ed eventualmente la accorpiamo. Quindi appunto bisogna fare attenzione al passo 3, non perché si può evitare, ma perché sappiamo che quando viene eseguito può generare una relazione che è in terza forma normale ma non in Boyce-Codd.

Altra proprietà: l'algoritmo ha una complessità polinomiale. Infatti il calcolo della copertura minimale è polinomiale (questo è il passo uno ed è già stato discusso prima), poi il passo 2, in cui facciamo la decomposizione, è anch’esso polinomiale perché itera su tutte le dipendenze funzionali, quindi considera una volta ogni dipendenza funzionale e compie per ognuna una operazione che è polinomiale nella lunghezza delle dipendenze funzionali; il controllo del passo 3, in cui verifichiamo se una dipendenza funzionale sia contenuta in un’altra, può essere svolto con una complessità polinomiale e infine il passo 4, in cui cerchiamo se una chiave è già presente, calcola la chiusura di un insieme di attributi e questa operazione, che abbiamo già discusso, è polinomiale.

Altra proprietà dell’algoritmo: possiamo dimostrare che l'algoritmo conserva le dipendenze. Cioè ogni dipendenza che troviamo nella copertura minimale la possiamo poi ritrovare, al termine dell'algoritmo, in una sola relazione, perché o per costruzione è una dipendenza funzionale che usiamo per la decomposizione del passo 2 e quindi sarà una dipendenza funzionale che determina la chiave della relazione Ri della decomposizione, oppure quando controlliamo il contenimento tra due relazioni e poi trasferiamo le dipendenze funzionali della relazione contenuta in un'altra, comunque la ritroviamo nella restrizione della nuova relazione. In tutte e due i casi comunque la decomposizione conserva le dipendenze.

L'ultima proprietà di cui parliamo è la proprietà di decomposizione senza perdita, cioè la normalizzazione in terza forma normale garantisce che possiamo ottenere una decomposizione senza perdita.

Non vediamo la dimostrazione di questa proprietà perché sarebbe troppo lunga.

Quindi, concludendo il discorso sulla normalizzazione in terza forma normale, possiamo notare che è successo, in almeno uno degli esempi che abbiamo visto, che la decomposizione che abbiamo generato fosse una decomposizione in forma normale di Boyce-Codd, cioè abbiamo applicato la normalizzazione in terza forma normale e ciò che abbiamo ottenuto non era solo in terza forma normale, ma era anche in questa forma più restrittiva che è la forma normale di Boyce-Codd.

Quali sono i casi in cui otteniamo una normalizzazione in Boyce-Codd? Sono i casi in cui non vengono generate delle dipendenze funzionali di tipo 3.

Riprendiamo il discorso affrontato poco fa in una dimostrazione. Se riguardate le definizioni di terza forma normale e di forma normale di Boyce-Codd, potete notare che la differenza tra le due è che la terza forma normale ammette un caso in più che la forma normale di Boyce-Codd non ammette, cioè appunto le dipendenze funzionali di tipo 3, in cui il conseguente di una dipendenza funzionale contiene attributi primi.

Quali sono i casi in cui vengono generate queste dipendenze funzionali tipo 3? Se riguardiamo l'algoritmo, non sono generate sicuramente dai passi 2 e 4 dell'algoritmo perché generano esclusivamente dipendenze funzionali in cui l’antecedente è una chiave. Sono, invece, i casi in cui abbiamo applicato il passo 3 dell'algoritmo, cioè quelli in cui abbiamo verificato se non ci fosse un contenimento tra gli schemi delle relazioni, perché sono i casi in cui per conservare la dipendenza funzionale della relazione che stiamo eliminando la aggiungiamo a un'altra relazione e in questa relazione non è detto che la dipendenza rispetti la condizione di tipo 2.

Quindi non abbiamo visto, quando abbiamo parlato della forma normale di Boyce-Codd, nei dettagli l'algoritmo di normalizzazione in forma normale di Boyce-Codd, perché comunque conviene procedere nel modo che abbiamo visto adesso: cioè applichiamo sempre e soltanto l'algoritmo di normalizzazione in terza forma normale, perché ha dei vantaggi rispetto alla normalizzazione in Boyce-Codd , dato che permette di conservare le dipendenze e ha una complessità polinomiale e poi possiamo analizzare il risultato che otteniamo. Se il risultato è in forma normale di Boyce-Codd, siamo contenti, perché in questo caso siamo sicuri che le anomalie saranno annullate. Se invece il risultato non è in forma normale di Boyce-Codd, allora sappiamo che possono rimanere delle anomalie, ma comunque possiamo accettarle perché la forma normale di Boyce-Codd non è sempre raggiungibile.

**19**

In questa breve lezione parliamo di un argomento che è molto importante nelle basi di dati, che è quello delle transazioni. In una base di dati non ci sono solo le interrogazioni, di cui abbiamo parlato ampiamente, ma ci sono anche operazioni che modificano l'istanza della base di dati, cioè il contenuto della base di dati.

Queste sono operazioni che corrispondono ai comandi SQL di INSERT, DELETE e UPDATE, cioè un comando che inserisce un nuovo record, uno che cancella un record e uno che modifica il valore di un record esistente. Quando avviene una modifica della base di dati, il DBMS accetta questa modifica solo se la modifica rispetta i vincoli che sono stati imposti dal progettista sulla base di dati.

Questi vincoli sono, per esempio, quelli di chiave, per esempio chiave primaria, vincoli unique, vincoli di integrità referenziali, vincoli di dominio e anche vincoli imposti dall’utente con delle espressioni logiche.

In ogni caso, quando c'è un'operazione che viene eseguita sulla base di dati, prima il DBMS controlla se quest'operazione lascia il database in uno stato coerente. Se la risposta è no, allora l'operazione va in errore e viene rifiutata, non viene eseguita sul database, perché nelle basi di dati relazionali un concetto importante è che la base di dati deve essere sempre coerente, altrimenti non potremmo fidarci delle risposte alle interrogazioni fatte sulla base di uno stato incoerente della base di dati.

Vediamo un esempio per capire che questo comportamento potrebbe risultare problematico.

Nell'esempio abbiamo dei clienti di una banca con i loro conti correnti: abbiamo tre tabelle/relazioni: conto, con il numero del conto corrente e altri attributi, cliente, che ha il codice fiscale del cliente come chiave primaria e altri attributi, e titolare, che ha due attributi, codice fiscale del cliente e numero del conto corrente, tutte e due in chiave primaria perché un titolare può essere titolare di più conti correnti e un conto corrente potrebbe anche avere più titolari.

Abbiamo due vincoli di integrità referenziale: il primo dice che TITOLARE(CF) referenzia CLIENTE(CF), quindi, quando si inserisce una tupla/record dentro la relazione TITOLARE, ci deve essere un cliente già esistente nella relazione cliente. Il secondo vincolo di integrità referenziale dice che TITOLARE(CC) referenzia CONTO(CC), cioè, quando si inserisce un nuovo titolare nella tabella TITOLARE, deve esserci un numero del conto corrente corrispondente nella relazione CONTO.

Consideriamo anche un nuovo vincolo che non deriva dall’organizzazione relazionale, ma viene imposto dalla banca, cioè l'azienda richiede che, quando un nuovo cliente arriva - e quindi viene inserito nel database - questo cliente deve essere titolare di un conto corrente. Quindi abbiamo in realtà un nuovo vincolo all'integrità referenziale, che dice che CLIENTE(CF) deve referenziare TITOLARE(CF): in questo modo ci assicuriamo che sia titolare di un controcorrente.

Quindi, riassumendo, abbiamo queste tre relazioni con questi tre vincoli di integrità referenziale.

Ora assumiamo che sul database vogliamo eseguire queste tre operazioni: l'operazione a, che inserisce un nuovo conto nei conti correnti, l'operazione b, che inserisce un nuovo cliente nella relazione cliente, e l'operazione c, che inserisce un nuovo titolare, cioè una nuova coppia con il nuovo cliente che abbiamo inserito e un nuovo conto che abbiamo inserito nelle altre due operazioni a e b.

Supponiamo di eseguirle nell'ordine della slide, cioè prima a, poi b, poi c, quindi eseguiamo a, che inserisce un nuovo conto nella relazione conto, e poi eseguiamo b, che inserisce un nuovo cliente in CLIENTE. Ricordiamoci però che esistono i vincoli di integrità referenziale, e quindi, quando inseriamo un nuovo cliente in CLIENTE, c'è un vincolo chi dice: “No, questo cliente deve essere già titolare di un conto corrente”, che è il terzo vincolo di integrità referenziale, quello imposto dalla banca, cioè CLIENTE(CF) referenzia TITOLARE(CF). Quando eseguiamo l'operazione b questo vincolo di integrità referenziale è violato, perché non siamo ancora arrivati ad eseguire l'operazione c, quella in cui inseriamo un nuovo titolare.

Proviamo a considerare un diverso ordine di eseguire le operazioni: congetturiamo che il problema sia nato dal fatto che non c'era il nuovo titolare, e quindi dopo l'operazione a eseguiamo, invece dell'operazione b, c, cioè inseriamo il nuovo titolare, quindi inseriamo prima il conto, come abbiamo fatto prima, e poi inseriamo il nuovo titolare. Cosa succede? Stiamo cercando di inserire il nuovo titolare, però c'è un vincolo di integrità referenziale che dice che, quando c'è un nuovo titolare, questo deve avere già una tupla corrispondente nella relazione CLIENTE. Ciò che dice questo primo vincolo di integrità referenziale è che non possiamo inserire un nuovo titolare di un conto se questo non è già un cliente della banca, e quindi abbiamo una violazione anche di questo vincolo.

Quindi abbiamo provato ad eseguire le operazioni nell'ordine a, b, c e abbiamo fallito quando abbiamo cercato eseguire b, poi abbiamo considerato un altro ordine, prima a, poi c e poi b, e abbiamo fallito di nuovo, perché non va bene neanche in quest'ordine.

Se ci pensiamo, non c'è un modo per eseguire queste operazioni in modo che, dopo l'esecuzione di ogni operazione, questa base di dati sia in uno stato coerente. Però, se noi potessimo considerare queste tre operazioni come un blocco unico e sospendere il controllo dei vincoli di integrità referenziale dopo l'esecuzione della singola istruzione, ma invece controllare questi vincoli solo dopo l'esecuzione del blocco di istruzioni, allora vedremmo che la base di dati sarebbe in uno stato coerente e i vincoli sarebbero rispettati.

Questo sta alla base dell'idea di transazione, cioè una transazione permette di considerare un blocco di operazioni deciso dai programmatori. Un blocco di operazioni si considera come se fosse un’istruzione unica in cui i vincoli devono essere rispettati solo prima dell'esecuzione e dopo l'esecuzione, ma non necessariamente durante l'esecuzione.

Quindi una transazione è un'unità di programma, cioè è composta da una sequenza di istruzioni che iniziano con un comando SQL di begin transaction, che comunica al DBMS che la transazione ha inizio e il DBMS assegna alla transazione un id in modo da identificare questa transazione, poi vengono comunicati al DBMS i vari comandi di inserimento, cancellazione, modifica e anche interrogazioni all'interno della transazione, e poi, quando si arriva alla fine del blocco di istruzioni, il programmatore può scrivere due istruzioni, cioè o una istruzione di commit, che significa che la transazione è andata a buon fine, oppure un’istruzione di rollback, che significa che la transazione invece ha fallito.

Questi sono due comandi di SQL che poi vediamo meglio nel seguito.

Vediamo più nel dettaglio il ciclo di vita di una transazione attraverso questo automa a stati finiti, che inizia con un comando di begin transaction, che porta la transazione nello stato di transazione attiva. Poi la transazione comprenderà dei comandi di inserimento, di cancellazione, di aggiornamento e anche di interrogazione del database, e questo continua a far rimanere la transazione nello stato di transazione attiva. Si esce da questo stato attraverso uno dei due comandi, o il comando di commit o il comando di rollback.

Quando si esegue il comando di commit? Quando una transazione emette questo comando significa che segnala al DBMS che la transazione ha avuto un buon fine, almeno dal punto di vista del programmatore, e quindi la transazione entra nello stato di transazione parzialmente terminata.

Perché non entra direttamente in uno stato di transazione terminata? Perché a questo punto il DBMS deve controllare tutti i vincoli, i vincoli di chiave primaria, di integrità referenziale eccetera, per verificare se questi vincoli sono rispettati o non sono rispettati.

Se tutti i vincoli sono soddisfatti, la transazione entra nello stato di committed. Committed significa che la transazione ha avuto buon fine e quindi i cambiamenti che la transazione ha apportato vengono resi definitivi sul database vero e proprio memorizzato su memoria secondaria.

Se invece c'è almeno un vincolo che non è rispettato, quindi non è soddisfatto, allora la transazione non va nella fase di commit, ma entra nello stato di transazione fallita. Questo è lo stesso stato in cui la transazione entra a seguito di un commando rollback.

Cosa vuol dire rollback? Vuol dire che il programmatore si è accorto che a un certo punto la transazione non può essere portata a conclusione con successo. Un esempio: immaginiamo di avere un database che serve per gestire la prenotazione dei posti su un treno o un aereo. Se richiediamo con un commit di chiudere la transazione e il DBMS rileva che il posto è già stato prenotato da qualche altra transazione in modo concorrente mentre stavamo eseguendo queste operazioni, allora non c'è più modo di riuscire a concludere la transazione con successo. Quindi il comando di rollback viene scelto dal programmatore - non dal DBMS stesso - per chiedere al DBMS di "fare tornare indietro" la transazione.

La transazione entra, quando incontra i due casi di richiesta di rollback o di vincoli non rispettati, nello stato di transazione fallita, e da questo stato esce nello stato di aborted.

Per entrare nello stato di aborted il DBMS deve disfare tutti i comandi di inserimento, cancellazione e aggiornamento che sono stati eseguiti fino a quel momento, perché deve riportare il database allo stato che aveva prima dell'inizio della transazione, nella situazione che si avrebbe se la transazione non fosse stata eseguita per nulla.

Nelle slide successive, che adesso non commento, trovate la descrizione di quello che ho commentato direttamente sull’automa stati finiti e quindi vado avanti velocemente, potete mettere in pausa se volete leggere ciò che è scritto sulle slide.

Quali sono buone pratiche nella scrittura delle transazioni? Si deve cercare di scrivere transazioni brevi per migliorare la concorrenza perché non si tengono occupate le risorse per troppo tempo. Di questo argomento poi parleremo meglio nella parte di architettura delle basi di dati.

Questo è un esempio di una transazione in SQL. Abbiamo il comando di “start transaction”, che fa partire la transazione - in SQL i comandi sono sempre terminati dal punto e virgola.

Abbiamo anche un comando che cambia il valore di una tupla, cioè il comando di update, che è seguito dal nome della tabella, e il comando set dice di prendere l'attributo saldo e assegnargli un nuovo valore, cioè saldo uguale saldo più 10, cioè aumentiamo di 10 euro il saldo di questo conto corrente per il conto corrente che numero “12202”.

Poi abbiamo un secondo comando update contoCorrente, di un altro conto, che ha un altro numero, e a questo altro conto, invece, togliamo 10 euro.

Infine arriviamo al comando di commit della transazione che segnala che la transazione è andata a buon fine, almeno dal punto di vista del programmatore, e quindi i cambiamenti possono essere resi permanenti sulla base di dati.

Notate che una transazione non può essere eseguita solo parzialmente, perché se eseguissimo parzialmente la transazione - e cioè se ci fermassimo a eseguire solo il primo comando, cioè dopo lo start transaction eseguissimo solo il primo update - allora avremmo assegnato 10 euro a questo conto ma non avremmo tolto 10 euro dall'altro conto corrente, e quindi avremmo creato 10 euro dal nulla. Questo è un segnale di una cosa che poi di cui parleremo meglio, cioè del principio per cui le transazioni devono essere considerate atomiche, e cioè eseguite o per tutto o per niente, e non a metà.

Questo è un altro esempio di una transazione che è leggermente più complessa. Ha una prima parte che è uguale alla prima parte della transazione che abbiamo già visto. Dopo, però, c'è anche una decisione, cioè questo comando if, che è un'estensione di SQL. Prima, con una select, leggiamo il saldo del conto corrente 42177 e scriviamo il risultato nella variabile a, poi, usando i costrutti di questo linguaggio di programmazione immerso in SQL, controlliamo se ha un saldo positivo, cioè se il conto corrente da cui abbiamo tolto i soldi, 42177, è ancora positivo. Se è positivo richiediamo un commit, cioè terminiamo con successo, se invece il saldo è negativo, assumiamo di avere sbagliato a fare questo bonifico, cioè assumiamo che la richiesta di effettuare un bonifico non sia stata una richiesta corretta perché non c’erano abbastanza soldi sul conto corrente, e richiediamo un roll back per tornare indietro e disfare tutta la transazione.

In un database relazionale vengono rispettate le proprietà delle transazioni che adesso commentiamo.

Queste proprietà sono molto importanti, hanno garantito il successo delle basi di dati relazionali e motivano molte delle scelte sull'architettura delle basi di dati, perché alla base di ogni proprietà deve esserci un modo per garantire questa proprietà, e questo non è sempre banale. Qualche accenno a questo lo faremo nella parte che inizieremo dopo e che riguarda l'architettura delle basi di dati, e che tratteremo in modo tangenziale e non approfonditamente.

Queste proprietà di solito non sono invece rispettate dalle basi di dati non relazionali, come, ad esempio, le basi di dati NoSQL, in cui si è scelto di lasciare cadere qualcuna di queste proprietà per potere garantire prestazioni maggiori. Quindi le basi di dati NoSQL non forniscono solo vantaggi, nonostante siano più recenti rispetto alle basi di dati relazionali, ma comportano anche svantaggi perché non tutte queste proprietà, desiderabili in una base di dati, sono garantite.

Le proprietà sono quattro. Per ricordarle facilmente in modo mnemonico di solito traggono il loro nome dalle iniziali, proprietà ACID, in inglese.

Queste proprietà sono di Atomicità, quindi una transazione deve essere atomica, poi la commentiamo, deve essere Consistente, deve essere Isolata e deve essere Durabile, in italiano, consistente si dovrebbe dire coerente, perché in italiano consistente vuol dire una cosa diversa, e durabile bisognerebbe dirlo persistente.

Adesso vediamo, una per una, queste proprietà, iniziando dall’Atomicità.

Una transazione che rispetta questa proprietà viene o eseguita interamente oppure non deve essere eseguita per niente, cioè nessuna delle azioni di cui è composta deve essere eseguita: non possono esistere casi in cui viene eseguita solo una parte di una transazione.

Di primo acchito può sembrare una proprietà che non è così difficile da ottenere, ma questa proprietà deve valere sempre in una base di dati, cioè, se abbiamo un guasto in una base di dati, e si può guastare la CPU per esempio, comunque un DBMS deve garantire che sia valida la proprietà di atomicità. Immaginiamo che il DBMS stia eseguendo una transazione e ad un certo punto si verifica un guasto a metà dell'esecuzione della transazione: anche in questo caso la transazione deve essere o eseguita interamente o non essere eseguita per niente.

Come si fa a garantire questa proprietà? In seguito faremo qualche cenno nella parte di architettura. I DBMS esguono UNDO e REDO, cioè disfare e riapplicano le operazioni, disfano e riapplicano una transazione. Se il guasto avviene prima che si raggiunga un comando di commit, che è il comando che, come abbiamo detto, segnala che la transazione ha avuto successo, allora, se il guasto avviene prima del commit, il DBMS disfa tutte le operazioni che sono già state svolte; se invece il guasto avviene dopo il commit, allora la transazione deve essere eseguita completamente anche se è accaduto un guasto, e quindi il DBMS ripete le operazioni che possono non essere ancora state apportate al database ma erano presenti nella transazione.

Come si fa, quindi, a garantire questa proprietà? I DBMS fanno uso di solito di file di log, in cui tengono traccia, come in un registro, delle operazioni, e grazie al log possono rieseguire o disfare le operazioni.

L'esito di una transazione, lo abbiamo già detto, può essere un caso di commit, in cui si ha successo ed è il caso più frequente, oppure un caso di abort, che causa un rollback. Un caso di abort può essere richiesto dalla applicazione e in quel caso è come se la transazione stessa stesse commettendo un suicidio, cioè richiede di disfare tutto ciò che ha chiesto di fare in un primo momento, oppure può essere il sistema che fa una richiesta di abort alla transazione perché magari sono stati violati dei vincoli, oppure ci sono altri motivi, per esempio per garantire la concorrenza, forse qualcosa avremo tempo di dire su quest’ultimo motivo, in cui appunto è il sistema che uccide la transazione e quindi anche la transazione deve essere disfatta eseguendo questo rollback.

Un'altra proprietà importante è quella di Consistenza, oppure, se vogliamo dire meglio in italiano, di coerenza, però di solito si usa questa strana traduzione non completamente corretta dall'inglese: consistenza.

Cosa vuol dire consistenza? Vuol dire che la base di dati deve essere in uno stato consistente, cioè deve partire, prima della transazione, da uno stato consistente, la transazione deve essere eseguita, e dopo l’esecuzione il database deve essere lasciato in uno stato corretto e consistente. I programmatori, cioè, durante la transazione devono fare in modo di eseguire delle operazioni che lasciano, alla fine della transazione, la base di dati in uno stato consistente, se invece non lo fanno, al termine del commit, il DBMS controlla i vincoli, si accorge che qualche vincolo è violato, e quindi richiede di abortire la transazione, anche dopo che ha raggiunto la fase di commit. La transazione viene disfatta, tutte le operazioni vengono disfatte, e comunque il DBMS fa in modo che la le transazioni lascino sempre lo stato della base di dati in uno stato consistente.

La terza proprietà delle ACID è l’Isolamento. I DBMS non accettano una transazione per volta in esecuzione: un DBMS può avere molti utenti che sono collegati contemporaneamente, cioè molti programmi che vengono eseguiti contemporaneamente, e ognuno sta eseguendo una transazione.

Quindi ci sono transazioni che devono essere eseguite in modo concorrente sulla stessa base di dati, alcune di queste azioni, all'interno delle transazioni, potrebbero agire anche sulla stessa tupla e, quindi, c'è un problema di accesso concorrente alle risorse.

Il DBMS deve fare rispettare la proprietà di isolamento, cioè ha dei meccanismi che fanno in modo che l'esecuzione concorrente, cioè concomitante/contemporanea di transazioni, anche sulla stessa risorsa, cioè anche sulla stessa tupla, sulla stessa tabella, diano lo stesso risultato che si avrebbe se ogni azione, ogni transazione, venisse eseguita da sola, come se fosse l'unica ad essere eseguita sul sistema.

Questo viene garantito attraverso certi meccanismi di architettura che fanno in modo che, anche nei casi in cui questo non si riesce a garantire, le transazioni vengono abortite in modo che vengano fatte ripartire e non violino il principio di isolamento.

L'ultima delle proprietà ACID è la proprietà di Durabilità, o in italiano persistenza, che è una proprietà di cui in realtà abbiamo già parlato. Nell'automa a stati finiti abbiamo visto che, quando viene richiesto un commit, il DBMS controlla se tutti i vincoli della base di dati sono soddisfatti e, in questo caso, la transazione entra in uno stato di committed. Questo è quello a cui facciamo riferimento quando diciamo che le modifiche sono andate a buon fine e, in questo caso, tutte le modifiche che devono essere apportate alla base di dati devono essere rese persistenti: anche se, per esempio, mancasse la corrente o ci fosse un guasto, comunque queste modifiche devono continuare a essere presenti nella base di dati.