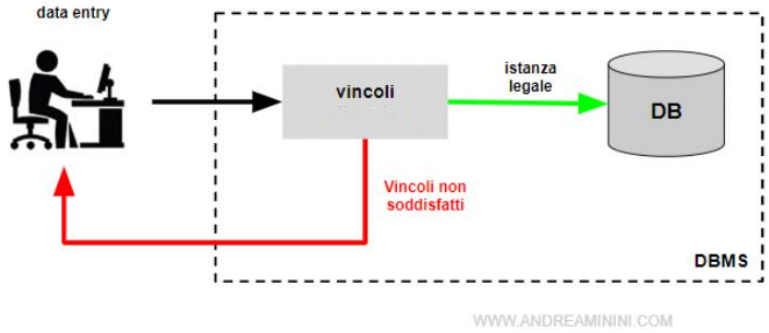
LE BASI DI DATI RELAZIONALI

Vincoli di integrità – Nomi delle chiavi – Normalizzazione - Transazioni

1. CAP 1: VINCOLI DI INTEGRITA'

I vincoli di integrità in un database sono regole che assicurano la validità e la coerenza dei dati. Questi vincoli possono essere applicati a livello di schema e sono fondamentali per mantenere **la qualità dei dati** memorizzati. Quando, attraverso una qualsiasi applicazione si cercherà di aggiungere dei dati, soltanto le istanze che rispettano i vincoli di integrità, potranno essere aggiunte al database.



Definizione

1. Come già detto i **vincoli di integrità** sono le **condizioni** che devono essere rispettate dalle istanze del DB affinchè i dati siano **significativi e validi**.

2. Ogni vincolo può essere espresso come un predicato booleano che, relativamente ad un istanza del DB, può assumere valore vero o falso. L'istanza viene inserita nel DB solo se l'espressione è vera, quindi se il vincolo viene rispettato. I DBMS offrono degli strumenti per facilitare l'inserimento dei vincoli.

Prima di elencare tutti i possibili vincoli definiamo un vincolo particolarmente importante. Il "vincolo di integrità referenziale". Questo vincolo è espresso dalla chiave esterna di un data base relazionale ed è definito nel seguente modo.

Vincolo di integrità referenziale: garantisce che, date due tabelle T1 e T2, dove l'attributo K è chiave primaria di T1 e chiave esterna di T2, non sia possibile aggiungere un riga alla tabella T2 se il valore assunto da K in questa riga non è già presente in T1.

(in altre parole: è necessario che ogni nuova riga nella tabella con chiave esterna, abbia un valore della chiave esterna che **è già presente** come valore della chiave primaria nella tabella collegata)

Vediamo ora tutti i vincoli di integrità e classifichiamoli.

I vincoli si suddividono in:

- vincoli **inter**relazionali (all'interno di una sola relazione)
- vincoli **intra**relazionali (fra elementi diverse reazioni)

INTRA-RELAZIONALI	INTER-RELAZIONALI
Vincoli di chiave 1. il valore della chiave primaria non deve essere null 2. il valore della chiave primaria non deve ripetersi nelle diverse tuple	**Vincolo di integrità referenziale**
Vincoli di tupla Interessano una sola tupla indipendentemente dai valori assunti dalle altre tuple. Se interessa un solo attributo è detto anche **vincolo di attributo o vincolo di dominio**. Possibili vincoli di dominio sono i seguenti: 1. L'attributo non può assumere valore null. esempio: data_nascita!=null (solo se questa espressione è vera, la tupla che contiene il valore data_nascita verrà aggiunta al database) 2. L'attributo può assumere solo determinati valori esempio: voto>=18 and voto<=30 3. Vincolo di tupla su più attributi, si esprime con un predicato dell'algebra di boole che coinvolge più attributi. Esempio: l'attributo booleano "lode" puo' avere valore "true" solo se l'attributo "voto" ha valore 30). !(lode=true and voto!=30) Si può esprimere Anche come: (!(lode=true) or voto=30)	

Già espresso anche nel diagramma e/r con la cardinalità minima=1 dell'attributo.

4. Vincolo di unicità: l'attributo (o insieme di attributi) non può ripetersi nelle tuple Esempio: in una tabella utenti ci sono sia id_utente che codice_fiscale, se all'id_utente è assegnato il vincolo di chiave, si può aggiungere un ulteriore vincolo per indicare che il codice fiscale non possa ripetersi, questo è un vincolo di unicità UNICO (codice_fiscale)	

2. CAP. 2: I MERAVIGLIOSI NOMI DELLE CHIAVI

Abbiamo già specificato cosa si intende per chiave primaria (PK) e chiave esterna (FK), ma nello studio delle basi di dati si sentirà parlare anche di **superchiave** e di **chiave candidata**. Chiariamo questi ulteriori due concetti.

Riprendiamo l'esempio del database "Libri" visto precedentemente

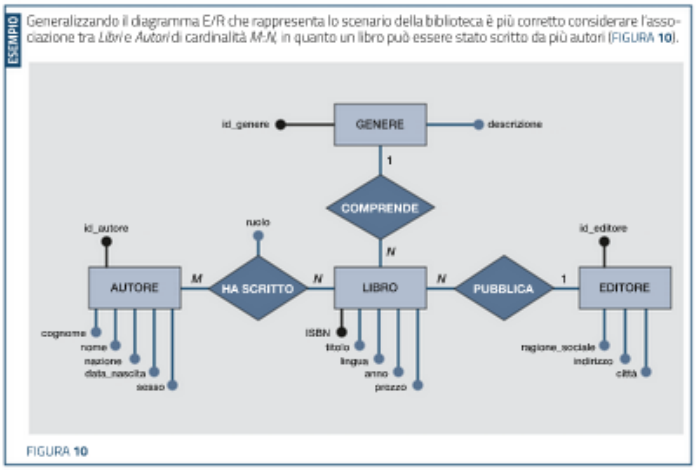


FIGURA 10

Lo **schema logico completo della base di dati** è dato quindi dai seguenti **schemi**:

Libri (**ISBN**, titolo, lingua, anno, prezzo, *id_genere*, *id_editore*)

Autori (**id_autore**, cognome, nome, nazione, data_nascita, sesso)

Editori (**id_editore**, ragione_sociale, indirizzo, città)

Generi (**id_genere**, descrizione)

Autori_Libri (**id_autore**, **ISBN**, ruolo)

Consideriamo la relazione "Autori_Libri". La chiave primaria è una chiave **composta** data dagli attributi **id_autore** e **ISBN**. Supponiamo, per rendere più agevole la ricerca di informazioni, di aggiungere un ulteriore attributo, un numero progressivo che si incrementa ogni volta che viene aggiunta una nuova tupla, chiamiamo questo campo **id_ruolo**.

La tabella che rappresenta la relazione diviene quindi la seguente:

Id_ruolo	**id_autore**	**ISBN**	**ruolo**
1	A002	123-456-789	autore
2	A001	987-654-321	autore
3	A001	012-345-678	autore
4	A004	876-543-210	autore
5	A003	876-543-210	coautore

Riferendoci a questa tabella per esemplificare i concetti diamo quindi le seguenti definizioni:

Superchiave: sottoinsieme degli attributi di una relazione **(non necessariamente minimale)** che identifica univocamente una tupla. Ossia tale che, considerando ogni coppia di tuple della relazione, il valore della superchiave è diverso.

Esempi di superchiave:

- {Id_ruolo}
- {Id_autore, ISBN} (si ipotizza che un autore possa avere un solo ruolo in ogni libro, se ha più ruoli si memorizza solamente il ruolo principale)
- {Id_ruolo, Id_autore, ISBN}
- {id_ruolo, id_autore, ISBN, ruolo} (tutti gli attributi)

Esempio di elenco di attributi che non è una superchiave

- {id_autore, ruolo} (non è superchiave perché non è univoca, vedi le righe 2 e 3)

Poiché il modello logico relazionale prevede che non ci siano tuple uguali, in una relazione ci sarà sempre una superchiave, al limite formata da tutti gli attributi.

Chiave candidata (rappresenta una possibile chiave primaria): sottoinsieme di attributi che rispetta entrambi i seguenti requisiti:

A. è una superchiave
B. è minimale. Quindi è una superchiave tale che, eliminando uno qualsiasi degli attributi, non è più una superchiave. (in altre parole: esiste un attributi che, se eliminato, fa sì che l'insieme non sia più superchiave)

Esempi di chiave candidata

- {Id_ruolo}
- {Id_autore, ISBN}

Esempi di elenco di attributi che non è una chiave candidata

- {Id_autore, ISBN, ruolo}
  non è chiave candidata perché **è** una superchiave **ma** se elimino ruolo è ancora una superchiave
- {Id_autore, ruolo}
  non è chiave candidata perché non è una superchiave

Chiave primaria: fra le chiavi candidate ne viene scelta una (solitamente quella con il minor numero di attributi) che viene definita chiave primaria.

- Nel nostro esempio la chiave primaria è dunque id_ruolo. Se non avessimo aggiunto tale attributo, la chiave primaria sarebbe stata l'unica chiave candidata vale a dire la chiave composta {Id_autore, ISBN}.

Diamo infine altre due definizioni:

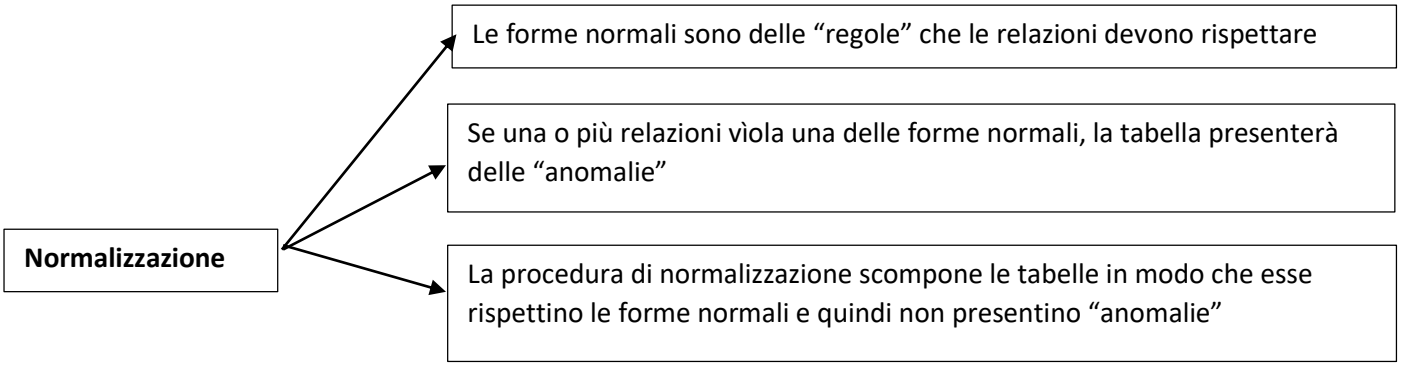
Attributo primo (o attributo chiave): attributo membro di almeno una **chiave candidata (ATTENZIONE: candidata. Non necessariamente primaria!)**. (nel nostro esempio sono attributi primi: ISBN, id_autore, id_ruolo)

Attributo non primo (o attributo non chiave): attributo che non è membro di alcuna **chiave candidata. (**nel nostro esempio: ruolo**).**

OSSERVAZIONE IMPORTANTE: quando si usa genericamente il nome **chiave** si intende un insieme di attributi che può svolgere il ruolo di chiave, quindi **chiave candidata**.

3. CAP 3: NORMALIZZAZIONE

Spiegheremo la normalizzazione nel seguente modo:



Se la progettazione concettuale e logica è svolta correttamente, le tabelle rispettano le forme normali quindi non è necessaria la normalizzazione

Come mostrato in precedenza il modello logico relazionale per essere tale deve soddisfare solamente 5 requisiti. In realtà Edgar Codd (Frank) nel 1985 ha definito più precisamente il modello relazionale in un articolo intitolato "*Is your DBMS really relational?"* elencando 12 requisiti (si possono trovare alla p. A46 del libro) che devono essere soddisfatti da un DBMS affinché esso si possa definire relazionale, ossia un RDBMS (Relational DBMS). Dal punto di vista concettuale il modello relazionale consente in maniera abbastanza semplice di descrivere la realtà. Questa semplicità può portare dei problemi in fase di progettazione, può infatti accadere che venga progettato un modello in cui, pur soddisfacendo i requisiti previsti dal modello relazionale, vi è **ridondanza** di dati (ossia dati che si ripetono). Tale ridondanza può causare problemi chiamati **anomalie**, problemi che, se presenti e rilevati in fase di progettazione, possono essere risolti applicando al modello determinate procedure note con il nome di **normalizzazione**. **La normalizzazione non è una tecnica di progettazione ma una tecnica per rilevare e correggere errori di progettazione.**

Vediamo innanzitutto quali possono essere le anomalie, e in seguito come svolgere la normalizzazione.

Per chiarire meglio i concetti partiamo dalla seguente relazione "Inventario" che rappresenta la quantità di determinati prodotti presenti in diversi magazzini.

Inventario (**codice_prodotto, codice_magazzino**, quantità, indirizzo_magazzino)

codice_prodotto	**codice_magazzino**	quantità	indirizzo_magazzino
545	CA1	800	Via Tonale, 12
545	PA2	700	Via Mazzini, 25
545	VE1	356	Calle Corta, 5
100	CA1	245	Via Tonale, 12
200	VE1	230	Calle Corta, 5
545	PA1	370	Via Garibaldi, 38
100	PA2	350	Via Mazzini, 25
100	VE1	720	Calle Corta, 5

La chiave primaria è composta dagli attributi **codice_prodotto** e **codice_magazzino.**

Anomalia di inserimento: non è possibile aggiungere a questa tabella un nuovo magazzino senza indicare un prodotto in esso depositato (perché il codice prodotto essendo chiave primaria non può avere valore null), quindi non è possibile, ad esempio, aggiungere l'informazione relativa all'indirizzo di un nuovo magazzino vuoto (senza prodotti).

Questa anomalia indica, pertanto, l'impossibilità di inserire correttamente alcuni dati.

Anomalia di cancellazione: se elimino tutti i prodotti che stanno nel magazzino CA1, perdo anche l'informazione sull'indirizzo del magazzino CA1 (perdere quell'informazione è un errore, poiché anche se nessun prodotto si trova più in quel magazzino, l'indirizzo del magazzino è un'informazione che dovrebbe comunque rimanere!).

Questa anomalia indica quindi l'indesiderata eliminazione di alcune informazioni in conseguenza dell'eliminazione di altre.

Anomalia da modifica (o da aggiornamento): se vi è la necessità di modificare l'indirizzo del magazzino CA1, ad esempio perché il magazzino viene spostato, tale modifica deve essere ripetuta per ogni tupla in cui è presente un prodotto che si trova in quel magazzino.

Questa anomalia indica quindi la necessità di dover ripetere lo stesso aggiornamento più volte in corrispondenza di una modifica nei dati. Se la modifica, per errore, non avvenisse in tutte le tuple necessarie si arriverebbe alla situazione in cui vi è **incongruenza (o inconsistenza)** dei dati, poiché uno stesso dato, presente più volte, assume valori distinti. Quando si verifica l'incongruenza dei dati si diche che la base di dati è **inconsistente**. Quando una base si dati è inconsistente è impossibile ottenere da essa informazioni.

Quale è la soluzione del problema?

Tutte le anomalie sono risolte sostituendo la tabella con le due tabelle seguenti

Inventario (**codice_prodotto, codice_magazzino**, quantità)

Magazzino (**codice_magazzino**, indirizzo_magazzino)

Inventario

codice_prodotto	**codice_magazzino**	quantità
545	CA1	800
545	PA2	700
545	VE1	356
100	CA1	245
200	VE1	230
545	PA1	370
100	PA2	350
100	VE1	720

Magazzino:

codice_magazzino	indirizzo_magazzino
CA1	Via Tonale, 12
PA2	Via Mazzini, 25
VE1	Calle Corta, 5
PA1	Via Garibaldi, 38

La separazione in due tabelle non ha causato perdita di dati (infatti la tabella di partenza si può ricostruire con l'operazione di congiunzione fra le due tabelle finali, operazione di congiunzione che vedremo meglio più avanti)

La separazione evita la ridondanza di dati (ogni dato è scritto una sola volta), in questo modo non si hanno più i problemi di anomalie. Le due tabelle non sono state generate in maniera casuale ma eseguendo un processo chiamato **normalizzazione.**

Spiegazione: per evitare che le tabelle di un modello logico relazionale abbiano problemi di anomalie, tali relazioni devono soddisfare determinati criteri. Le tabelle che sodisfano tali criteri sono dette **in forma normale (FN)**. Esistono 4 forme normali che ora elenchiamo e poi andremo a definire:

- prima forma normale (1NF)
- seconda forma normale (2NF)
- terza forma normale (3NF)
- forma normale di Boyce Codd

In realtà esistono ulteriori due forme normali (4FN e 5 FN) ma solitamente non vengono considerate perché normalizzare rispetto ad esse consente una ulteriore riduzione di ridondanza in alcuni casi ma provoca un degrado delle prestazioni)

Quando una relazione (tabella) vìola una di queste forme normali si dice che "**non è ben strutturata**".

Quando in un modello logico relazionale nessuna relazione vìola una FN si dice che le relazioni sono "**ben strutturate**".

La normalizzazione è un processo che scompone una relazione in più relazioni, quando la relazione di partenza vìola una FN. La normalizzazione fa sì che nelle relazioni ottenute, le FN non siano più vìolate e quindi "ben strutturate".

La normalizzazione consente quindi di evitare i problemi di anomalia dovuti alla ridondanza dei dati, il "prezzo" da pagare è una maggiore complessità nella ricerca delle informazioni (operazione che sarà però svolta dal DBMS in maniera automatica). Infatti per ottenere le informazioni sarà necessario "unire" le tabelle ottenute scomponendo la relazione iniziale, per formare nuovamente la tabella iniziale. Si può dire che la normalizzazione di una tabella rende più complessa la ricerca dei dati ma **evita le anomalie** nelle operazioni di **aggiornamento (inserimento, cancellazione e modifica)**, e quindi evita il pericolo principale nella gestione delle basi di dati che è **l'inconsistenza** dei dati.

Dipendenza funzionale e dipendenza transitiva

Prima di spiegare le 4 forme normali è necessario introdurre due ulteriori concetti, il concetto di **dipendenza funzionale** e di **dipendenza transitiva** tra insiemi di attributi di una tabella.

Dato un attributo A di una tabella (o un insieme di attributi A) si ha **dipendenza funzionale** quando il valore assunto dall'attributo A (o dall'insieme di attributi A) **determina** il valore assunto da un altro attributo B (o da un insieme di altri attributi B). Si dice che **B dipende funzionalmente da A** oppure che **A determina B** e si può indicare con **A→B**.

Cioè se "per determinati valori di A" nella tabella i valori di B "sono sempre gli stessi", naturalmente questo deve valere in generale per tutte le possibili tuple, non solo per quelle specifiche istanze presenti in tabella.

Ad esempio nella relazione:

LOCALITA: (id_localita, nome, comune, provincia, regione) provincia→regione

Tornando alla tabelle Inventario dell'esempio iniziale:

codice_magazzino→indirizzo

codice_prodotto, codice_magazzino →quantità

(perché non vale il contrario? Perché potrei avere nello stesso indirizzo due magazzini, ad esempio al piano terra e al piano primo, ma essi avranno codici diversi.)

OSSERVAZIONE: si noti che se l'insieme determinante A è una chiave candidata essa determina qualunque altro attributo non chiave.

Dipendenza transitiva: si ha quando **un attributo A** (o un insieme di attributi A) **è determinante per un attributo B** (o un insieme di attributi B) **e B è determinante per un attributo C** (o un insieme di attributi C), in tal caso si dice che **A è determinante per C transitivamente**, oppure che **C dipende transitivamente da A**. Si può indicare con **A→B e B→C** oppure **A→C transitivamente**.

Le dipendenze funzionali e transitive non sono sempre di facile individuazione analizzando i dati, è spesso necessario analizzare attentamente la realtà descritta dalla base di dati per individuarle.

Ad esempio nella seguente relazione che rappresenta i dipendenti di un'azienda e i dipartimenti in cui essi lavorano

DIPENDENTI (id_dipendente, nome_dipendente, nome_dipartimento, budget_dipartimento,)

id_dipendente → nome_dipendente

id_dipendente → nome_dipartimento A→B

nome_dipartimento→budget_dipartimento B→ C

budget_dipatimento dipende transitivamente da id_dipendente. La dipendenza è transitiva poiché il budget del dipartimento dipende direttamente dal nome del dipartimento (dato il nome del dipartimento, ogni tupla con quel nome avrà sempre lo stesso valore di budget)

Grazie a questi concetti possiamo ora descrivere le prime 3 forme normali (la quarta, la forma normale di Boyce Codd la analizzeremo successivamente dopo aver imparato le prime tre).

Prima forma normale (1NF).

Una relazione è in 1FN se tutti i suoi attributi sono atomici ed esiste per essa una chiave primaria.

Seconda forma normale (2FN) (per chiavi primarie composte)

Una relazione si dice in 2FN se:

- **è in 1FN**
- **tutti i suoi attributi NON CHIAVE dipendono funzionalmente dall'intera chiave, ossia non devono esserci dipendenze funzionali parziali dalla chiave.** (In altre parole: una relazione vìola la 2FN se vi è almeno un attributo non chiave che ha dipendenza funzionale PARZIALE da una chiave, ossia che dipende solo da una parte della chiave).

Nel nostro esempio di partenza la tabella "Inventario" è in prima forma normale ma vìola la 2FN perché l'attributo (non chiave) indirizzo_magazzino dipende da UNA PARTE della chiave primaria composta, ossia da codice_magazzino (che forma, insieme a codice_prodotto la chiave primaria).

Per risolvere il problema che la tabella vìola la 2NF operiamo sulla tabella una normalizzazione con l'apposita procedura.

Procedura di normalizzazione per relazioni che violano la 2FN:

1) Dalla relazione di partenza in cui vi sono attributi la cui dipendenza funzionale vìola la 2FN si crea una nuova relazione (indichiamola con R1). In questa nuova relazione vi sono **tutti gli attributi che violano la 2FN**, sia i determinanti (codice_magazzino) che quelli dipendenti (indirizzo_magazzino**). L'attributo determinante diventa la chiave primaria** di questa nuova tabella:

   R1: (**codice_magazzino**, indirizzo_magazzino)

2) Si costruisce una nuova relazione (R2) che contiene gli attributi di partenza **ad eccezione degli attributi che hanno la dipendenza parziale** dalla chiave primaria

   R2: (**codice_prodotto, codice_magazzino**, quantità, ~~indirizzo_magazzino~~)

Se alcune delle tabelle ottenute violano ancora la 2FN Il procedimento viene **reiterato** finchè tutte le relazioni sono in 2FN.

Il risultato è proprio quello che avevamo ottenuto con la soluzione mostrata quando è stato presentato il problema.

Esempio schematico per la normalizzazione in 2FN

Indichiamo con **R** una generica relazione e con le altre lettere i suoi attributi

R (A, B, C, D, E)

Nella quale {A, B} → C
A → D D dipende *parzialmente* dalla chiave
B → E E dipende *parzialmente* dalla chiave

Considerando la prima delle due dipendenze funzionali (A→D) si ottengono le seguenti due relazioni:

R1 (A, D)
R2 (A, B, C, E)

Così è stata risolta la prima dipendenza funzionale.
Risolviamo la seconda (B→E) ottenendo due relazioni (R2A ed R2B) da R2

R2A (B, E)
R2B (A, B, C)

Alla fine la relazione R è stata scomposta nelle relazioni normalizzate R1, R2A, R2B

Terza forma normale (3NF).

Una relazione è in 3FN se:

- **e' in 2FN**
- **tutti gli attributi non chiave hanno dipendenza funzionale DIRETTA dalla chiave. Ossia non devono esserci attributi non chiave che dipendono da altri attributi non chiave**

ATTENZIONE: attributi non chiave sono quelli che non fanno parte di ALCUNA chiave candidata. Quindi valuto ogni attributo NON PRIMO.

OSSERVAZIONE: le eventuali vìolazioni della 3NF possono avvenire SOLO per gli attributi NON CHIAVE, vale a dire quelli che non fanno parte di alcuna chiave candidata.

La seguente tabella è un esempio di tabella che viola la 3FN

Archivio per la gestione di immatricolazioni auto

Targa	Modello	colore	Numero_posti	anno
AJ321EE	punto	Nero	5	2015
DS232PE	multipla	Nero	6	2020
DF435TR	F40	Nero	2	2017
FF435TG	X1	Rosso	5	2015
JG938EE	Clio	Blu	5	2019
GH989RT	X1	Nero	5	2020
RE434RT	A3	Rosso	5	2018
RT432WW	punto	blu	5	2019

- è in 1NF poiché non ci sono righe uguali
- è in 2NF poiché non ci sono attributi che dipendono parzialmente dalla chiave (essendo la chiave non composta)
- Non è in 3FN poiché: modello → numero_posti

Dipendenza di attributo non-chiave da attributo non-chiave, ossia dipendenza transitiva dell'attributo cilindrata dalla chiave primaria, infatti:

targa → modello → numero_posti

quindi

targa → numero_posti transitivamente

Per risolvere il problema si applica alla tabella la procedura di normalizzazione per la 3FN (che praticamente uguale a quella per la 2FN)

Procedura di normalizzazione per relazioni che violano la 3FN:

1. Si crea una relazione contenente **gli attributi che violano le caratteristiche della 3FN** dove l'attributo **determinante** diventa la **chiave primaria**

   automobile: (**modello**, numero_posti)

2. Si costruisce una nuova relazione con gli attributi di partenza ad **eccezione** di quelli che hanno la **dipendenza transitiva**

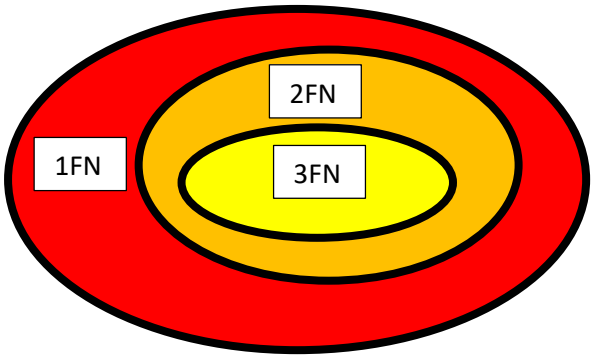
   immatricolazioni: (**targa**, modello, colore, ~~numero_posti~~, anno)

3. Il procedimento viene reiterato finchè tutte le relazioni sono in 3FN, in cui, quindi, non ci sono attributi non-chiave che dipendono da altri attributi non-chiave.

automobili	
Modello	Numero_posti
punto	5
multipla	6
F40	2
X1	5
Clio	5
A3	5

Immatricolazioni			
Targa	modello	colore	anno
AJ321EE	punto	Nero	2015
DS232PE	multipla	Nero	2020
DF435TR	F40	Nero	2017
FF435TG	X1	Rosso	2015
JG938EE	Clio	Blu	2019
GH989RT	X1	Nero	2020
RE434RT	A3	Rosso	2018
RT432WW	punto	blu	2019

Poiché una tabella per essere in 3NF deve essere in 2FN e per essere in 2FN deve essere in 1FN, si possono rappresentare dal punto di vista insiemistico le forme normali nel seguente modo:



Per risolvere gli esercizi di normalizzazione è necessario prima individuare le chiavi candidate, poi:

1. Si determinano le dipendenze funzionali

2. Si verifica la 3NF (ci sono attributi **non chiave** che dipendono da attributi non chiave?), se necessario si normalizza.

3. Si verifica la 2NF (ci sono attributi non chiave che dipendono parzialmente da una chiave candidata? Se non ci sono chiavi **candidate** composte, la 2FN è già verificata), e in caso sia necessario si normalizza

4. Si verifica la 1NF

5. Si verifica la BCNF (spiegata in seguito) e si decide se normalizzare o no

OSSERVAZIONE 1:

Ogni volta che con la procedura di normalizzazione si scompone una tabella è opportuno verificare che tale scomposizione sia corretta. La scomposizione è corretta se si verificano due condizioni:

1. **la dipendenza funzionale degli attributi viene mantenuta**
2. **se non c'è perdita di informazione.**

La dipendenza funzionale viene mantenuta se gli attributi determinanti e quelli determinati, dopo la scomposizione, fanno parte della stessa relazione.

Per verificare che la scomposizione è senza perdita di informazione si esegue un'operazione chiamata **join** fra le due tabelle scomposte e si verifica che la tabella ottenuta è uguale alla tabella di partenza. Il join fra le due tabelle ottenute con la scomposizione deve generare una tabella uguale alla tabella di partenza, altrimenti la scomposizione è "con perdita di informazione" e dunque non è corretta.

Il join si può spiegare così: date due tabelle, il join fra di esse è una tabella che ha le colonne di entrambe le tabelle, le colonne comuni alle due tabelle sono prese solo una volta, le righe sono il prodotto cartesiano delle righe di entrambe le tabelle ma considerando solo quelle in cui gli attributi in comune hanno lo stesso valore.

Esempio1 (magazzino): la composizione è senza perdita di informazione.

Tabelle ottenute dalla normalizzazione:

codice_prodotto	**codice_magazzino**	quantità
545	CA1	800
545	PA2	700
545	VE1	356
100	CA1	245
200	VE1	230
545	PA1	370
100	PA2	350
100	VE1	720

codice_magazzino	indirizzo_magazzino
CA1	Via Tonale, 12
PA2	Via Mazzini, 25
VE1	Calle Corta, 5
PA1	Via Garibaldi, 38

1. Le dipendenze funzionali sono mantenute

   codice prodotto, codice magazzino → quantita

   codice_magazzino → indirizzo_magazzino

2. Join fra le due tabelle (mostrato sotto) restituisce la tabella di partenza, quindi non c'è stata perdita ti informazione.

codice_prodotto	**codice_magazzino**	quantità	indirizzo_magazzino
545	CA1	800	Via Tonale, 12
545	PA2	700	Via Mazzini, 25
545	VE1	356	Calle Corta, 5
100	CA1	245	Via Tonale, 12
200	VE1	230	Calle Corta, 5
545	PA1	370	Via Garibaldi, 38
100	PA2	350	Via Mazzini, 25
100	VE1	720	Calle Corta, 5

Esempio 2 possibile errore di scomposizione di una tabella che può portare a perdita di informazione

Si vogliono rappresentare degli impiegati e i progetti ai quali essi lavorano.

Ogni impiegato lavora in una sola sede, ogni progetto viene svolto in una sola sede.

Impiegato	**Progetto**	**Sede**
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

FD: impiegato -> sede

progetto -> sede

Entrambe le FD vìolano la 2FN.

Soluzione 1 corretta

Seguendo l'algoritmo di normalizzazione si ottiene la soluzione corretta poiché vengono mantenute le dipendenze funzionali all'interno delle tabelle (scomposizione senza perdita)

T1: {impiegato, sede}

T2: {impiegato, progetto}

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

La congiunzione (join) genera la stessa tabella di partenza→scomposizione senza perdita

Soluzione 2 corretta

Anche la seguente scomposizione mantiene le dipendenze funzionali quindi non genera perdita di informazione (provare da soli a fare il join)

scomposizione (corretta):

T1: {impiegato, progetto}

T2: {progetto, sede}

Soluzione 3 non corretta

La seguente scomposizione invece non è corretta, perde le dipendenze funzionali e quindi genera perdita di informazione scomposizione (non corretta):

T1: {impiegato, sede}

T2: {progetto, sede}

La dipendenza funzionale (impiegato, progetto)→sede viene persa

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Saturno	Milano
Verdi	Venere	Milano
Neri	Giove	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Tuple spurie

La congiunzione (join) genera una tabella diversa da quella di partenza perché sono presenti righe "spurie"→scomposizione con perdita di informazione.

Spuria significa "non autentica", "falsificata". La presenza di righe spurie quindi **è considerata perdita di informazione.**

Come si fa a capire se la scomposizione ottenuta con la procedura di normalizzazione è senza perdita di informazione?

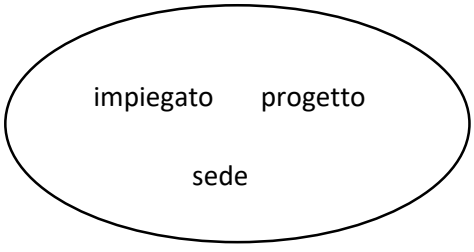
Regola:

Se durante la normalizzazione di una tabella T con un insieme di attributi A, si ottengono due tabelle T1 e T2 rispettivamente con insiemi di attributi A1 e A2, è stato dimostrato che la condizione sufficiente affinchè la scomposizione operata sia "senza perdita" è che l'intersezione fra i due insiemi A1 e A2 sia almeno superchiave per T1 o per T2.

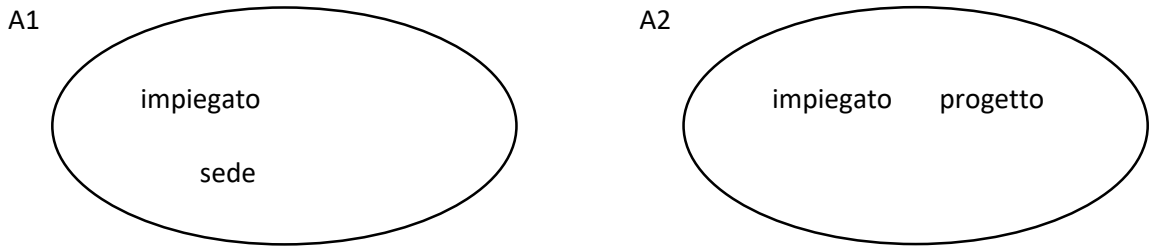
Vediamo l'applicazione di questa regola all'esempio appena considerato:

Consideriamo l'insieme A degli attributi di T

A

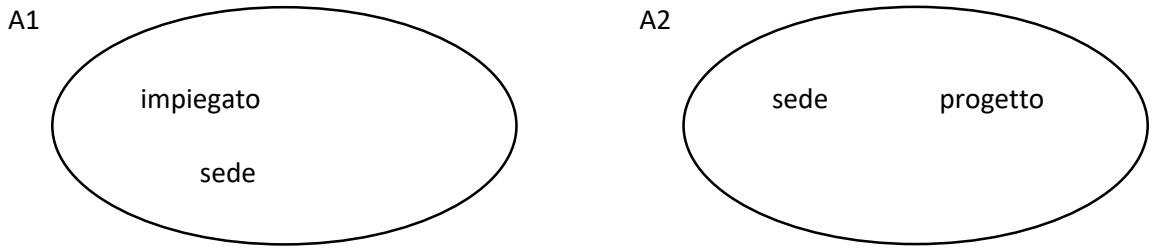


Nella prima scomposizione T1{impiegato, sede} T2{impiegato, progetto}, i due insiemi sono:



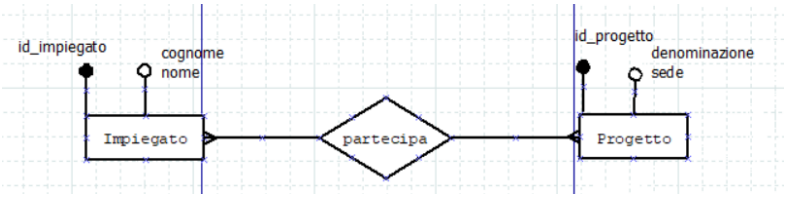
L'intersezione è data da "impiegato" che è superchiave di T1, quindi la scomposizione è senza perdita

Nella seconda scomposizione T1{impiegato, sede} T2{progetto, sede}, i due insiemi sono:



L'intersezione è data da "sede" che è non è superchiave né di T1 né di T2, quindi NON POSSIAMO GARANTIRE CHE la scomposizione sia senza perdita (infatti è con perdita poiché si formano righe spurie, come abbiano visto).

Comunque, se il progetto concettuale e logico fosse stato svolto correttamente, come mostrato con il seguente diagramma E/R, il problema non si sarebbe generato



OSSERVAZIONE 2:

Quando si verifica il rispetto delle FN bisogna ragionare sugli attributi, non sui valori da essi assunti, infatti è possibile che con alcuni dati "casualmente" le FN vengano rispettate, ma ciò che bisogna verificare è che le FN vengano rispettate con TUTTI i possibili valori che possono assumere gli attributi!

ESEMPIO SULLA NORMALIZZAZIONE (P. A48 libro di testo) (Tipico esercizio)

La seguente relazione Incarichi contiene i dati relativi agli impiegati che hanno partecipato a determinati progetti. Ogni impiegato ha un determinato livello di inquadramento nell'azienda ma, in progetti diversi, può svolgere compiti diversi e il ruolo svolto determina il costo orario dell'impiegato.

Incarichi (impiegato, livello, progetto, budget, anno_inizio, ruolo, costo_orario)

impiegato	livello	**progetto**	budget	anno_inizio	ruolo	costo_orario
Rossi	B1	Alfa	40000	2023	tecnico	10
Bianchi	C2	Beta	60000	2024	progettista	25
Bianchi	C2	Delta	60000	2023	progettista	25
Neri	D2	Delta	60000	2023	responsabile	40
Neri	D2	Beta	60000	2024	consulente	30
Neri	D2	Alfa	40000	2023	consulente	30
Verdi	D1	Alfa	40000	2023	responsabile	40
Verdi	D1	Delta	60000	2023	consulente	30
Grigi	C4	Delta	60000	2023	tecnico	10
Grigi	C4	Beta	60000	2024	responsabile	40

La tabella non è ben strutturata, infatti presenta le anomalie prima descritte:

- anomalia di inserimento: non posso aggiungere dati di un impiegato che non partecipa ad alcun progetto
- anomalia di cancellazione: se elimino Neri, Grigi e Bianchi perdo i dati (budget, anno) sul progetto Beta
- anomalia di aggiornamento: se cambia il costo orario del consulente devo modificare la tabella in 3 punti, se cambia quello del progettista in due punti ecc.

1. **La chiave primaria è (impiegato, progetto)**

2. **Individuiamo le dipendenze funzionali e se ci sono vìolazioni della 3FN o della 2FN**

Impiegato → livello	vìola 2FN
Progetto → budget, anno_inizio	vìola 2FN
Ruolo → costo_orario	vìola 3FN

3. **Normalizzo in 3FN**

   Livelli: {**ruolo**, costo_orario}

   Incarichi: {**impiegato**, livello, **progetto**, budget, anno_inizio, ruolo}

4. **Normalizzo in 2FN**

   Ruoli: {**ruolo**, costo_orario}

   Impiegati: {**impiegato**, livello}

   Progetti: {**progetto**, budget, anno_inizio}

   Incarichi: {**impiegato**, **progetto**, ruolo}

5. **Verifico se tutte le tabelle sono in 1FN:**

   Poiché tutte le tabelle hanno la PK e attributi atomici, la 1FN è verificata.

La base di dati è stata quindi normalizzata modificando la relazione di partenza in 4 relazioni.

La rappresentazione estensionale dei dati è la seguente:

Impiegati	
impiegato	livello
Rossi	B1
Bianchi	C2
Neri	D2
Verdi	D1
Grigi	C4

Incarichi		
impiegato	progetto	ruolo
Rossi	Alfa	tecnico
Bianchi	Beta	progettista
Bianchi	Delta	progettista
Neri	Delta	responsabile
Neri	Beta	consulente
Neri	Alfa	consulente
Verdi	Alfa	responsabile
Verdi	Delta	consulente
Grigi	Delta	tecnico
Grigi	Beta	responsabile

Progetti		
progetto	budget	Anno_inizio
Alfa	40000	2023
Beta	60000	2024
Delta	60000	2023

Ruoli	
ruolo	costo_orario
tecnico	10
progettista	25
consulente	30
responsabile	40

Forma normale di Boyce-Codd (BCNF)

La BCNF consente di ridurre ulteriormente la ridondanza dei dati che potrebbe rimanere in un modello in 3FN. **In alcuni casi però, decompore una tabella in modo che rispetti la BCNF non può avvenire senza la perdita di dipendenze funzionali**. In tali casi il progettista valuta se mantenere una certa ridondanza nei dati ma mantenere le dipendenze funzionali oppure, normalizzando in BCNF, eliminare la ridondanza perdendo la dipendenza funzionale. Negli esercizi ci limiteremo a verificare se una relazione rispetta la BCNF o meno.

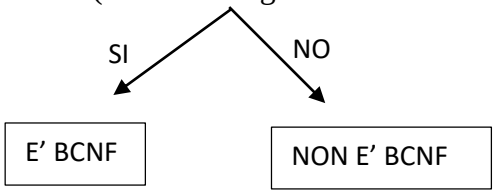
Una relazione è in forma normale di Boyce-Codd (BCNF) se è in 1FN(*) e per ogni dipendenza funzionale A→B dove A è un sottoinsieme di attributi e B è UN attributo (che può essere anche attributo chiave), gli attributi di A formano una superchiave per la relazione di partenza.

In altre parole, per essere BCNF, quando c'è una dipendenza funzionale A→B A deve sempre contenere essere superchiave. Se invece vi è una dipendenza A→B in cui gli attributi di A non sono superchiave, la relazione non è BCNF.

(*) (basta la 1FN, poi la 2FN e 3FN derivano automaticamente dalla definizione di BCNF, infatti è sempre vero che se la relazione è BCNF allora è anche in 3FN mentre il viceversa non è vero)

Per determinare se una relazione è in BCNF si fa così:

1. Si determinano tutte le dipendenze funzionali A → B
2. Si verifica se tutti i determinanti (A) sono superchiavi (ossia contengono una chiave candidata)



Quando una relazione vìola la BCNF, non sempre è opportuno normalizzare. Il motivo è che in alcuni casi, normalizzando si possono perdere le dipendenze funzionali. Il venir meno delle dipendenze funzionali rende molto più complesso il controllo sulla correttezza dei dati inseriti nella base di dati, come vedremo più avanti con un esempio. **Le dipendenze funzionali vengono perse quando gli attributi legati da una dipendenza funzionale "finiscono" in tabelle diverse a causa della procedura di normalizzazione**. Quando ciò avviene è preferibile evitare la normalizzazione, mantenendo le dipendenze funzionali, con lo svantaggio di mantenere una certa ridondanza dei dati.

Esempio di vìolazione BCNF con normalizzazione senza perdita di dipendenze funzionali

In questo esempio l'abbonato è rappresentato solo con il cognome, quindi possono esserci più abbonati con lo stesso cognome che abitano in città ed indirizzi diversi.

Prefisso	**Numero**	**Località**	**Abbonato**	**Indirizzo**
051	457856	Bologna	Rossi	Via Roma 8
059	452332	Modena	Verdi	Via Bari 16
051	987856	Bologna	Bianchi	Via Napoli 77
051	552346	Castenaso	Neri	Piazza Borsa 12
059	387654	Vignola	Mori	Via Piave 65
059	457856	Vignola	Rossi	Via Napoli 77

Dipendenze funzionali:

(prefisso, numero) → località, abbonato, indirizzo in quanto chiave

Località → prefisso (Ad esempio 051 è il prefisso anche di altre località vicino a Bologna)

La tabella è in 1NF, verifichiamo se è in BCNF

Località → prefisso Vìola BCNF poiché località non è superchiave

Normalizzazione:

Prefissi: {**località**, prefisso} (LA PK è località poiché è l'attributo determinante)

Numeri: {**numero**, **località**, abbonato, indirizzo} (Località diventa il nuovo attributo chiave al posto di prefisso)

Le seguenti tabelle sono in BCNF perché per ogni dipendenza funzionale, il determinante è superchiave.

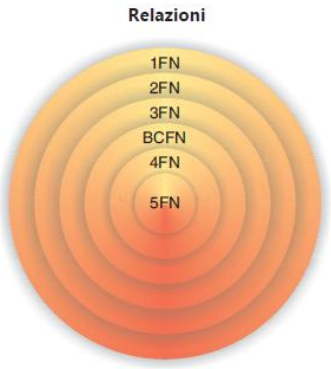
Prefisso	**Località**
051	Bologna
059	Modena
051	Castenaso
059	Vignola

Numero	**Località**	**Indirizzo**	**Abbonato**
457856	Bologna	Via Roma 8	Rossi
452332	Modena	Via Bari 16	Verdi
987856	Bologna	Via Napoli 77	Bianchi
552346	Castenaso	Piazza Borsa 12	Neri
387654	Vignola	Via Piave 65	Mori
457856	Vignola	Via Napoli 77	Rossi

Oltre a quelle viste esistono ulteriori forme normali (4FN e 5FN) ma che vengono raramente utilizzate perché hanno il vantaggio di ridurre la ridondanza dei dati ma possono provocare problemi dovute alla perdita di dipendenze funzionali e quindi di informazioni.

Le normalizzazioni fino alla 3FN invece non provocano mai perdita di informazione.

Ogni forma normale implica che sia rispettata la forma normale precedente come si può rappresentare nel seguente modo:



Comunque, la regola d'oro, è che mantenendo ben separate le entità nella progettazione del digramma E/R, le tabelle risultanti risultano, nella maggior parte dei casi, già normalizzate!

ESERCIZI DA FARE: LIBRO P. A85 n°54 (voli), 55 (ordini), 56 (palestra), 57 (vendite articoli) 58 (computer)

54 Con riferimento alla tabella *Voli*, riportata in basso e relativa ai voli nazionali giornalieri di una certa compagnia aerea

Voli(ora_volo, tratta, partenza, arrivo, aereo, posti)

A. individuare in essa le dipendenze funzionali;
B. proporne una riduzione di 3NF;
C. verificare se la 3NF individuata è anche in BCNF.

55 Con riferimento alla tabella *Ordini*, riportata in basso e relativa a ordini di prodotti effettuati da clienti a una certa azienda

Ordini(cliente, indirizzo, articolo, descrizione, prezzo, n_ordine, quantita)

sapendo che gli ordini sono numerati progressivamente per cliente e che in uno stesso ordine possono essere ordinati solo articoli diversi in quantità specificata:

A. individuare in essa le dipendenze funzionali;
B. proporne una riduzione di 3NF;
C. verificare se la 3NF individuata è anche in BCNF.

56 Una palestra ospita diversi corsi appartenenti a diverse tipologie di arti marziali (karate, judo, ju-jitsu, ecc.). Ogni corso ha una sigla, che lo identifica, un insegnante e alcuni allievi. Un insegnante offre in generale più corsi, anche di diverse tipologie, e anche un allievo può essere iscritto a più corsi. Di ogni insegnante interessano il nome (che lo identifica) e l'indirizzo. Di ogni allievo interessano il nome (che lo identifica) e il numero di telefono. Per ogni allievo interessa sapere, per ogni corso che frequenta, quanto ha già versato finora. La palestra gestisce attualmente i dati con un'unica tabella *Gestione dati*, strutturata come quella riportata in basso:

A. individuare in essa le dipendenze funzionali;
B. proporne una riduzione di 3NF;
C. verificare se la 3NF individuata è anche in BCNF.

Voli

ora_volo	tratta	partenza	arrivo	aereo	posti
07.00	T1	Milano	Napoli	A01	90
07.00	T2	Roma	Milano	A05	80
12.30	T1	Milano	Napoli	A02	70
10.00	T2	Roma	Milano	A01	90
10.00	T3	Roma	Napoli	A07	85
11.30	T1	Milano	Napoli	A05	80

Ordini

cliente	indirizzo	articolo	descrizione	prezzo	n_ordine	quantita
C1	I1	A1	D1	15	1	50
C1	I1	A2	D2	10	1	50
C1	I1	A1	D1	15	2	100
C1	I1	A3	D3	5	3	50
C2	I2	A2	D2	10	1	60
C2	I2	A4	D4	20	2	100
C3	I3	A1	D1	15	1	60

Gestione dati

Allievo	Telefono	Corso	TipoCorso	Saldo	Insegnante	Indirizzo
Matteo	303180	C1	Ju-jitsu	250.00	Giovanni	Via Roma
Pietro	304140	C1	Ju-jitsu	200.00	Giovanni	Via Roma
Matteo	303130	C4	Karate	300.00	Marco	Via Bini
Andrea	303530	C3	Judo	300.00	Luca	Via Roma
Pietro	304140	C7	Judo	250.00	Giovanni	Via Roma

Soluzioni

ES 54

1. Chiavi candidate (e primaria):

   {ora_volo, tratta} PK

   {ora_volo, aereo}

2. Dipendenze funzionali:

   Tratta → partenza, arrivo vìola 2FN

   (anche partenza, arrivo → Tratta)

   Aereo → posti vìola 2FN

3. Risolvo la seconda violazione (porto in 2FN):

   Aerei:{**aereo**, posti}

   Voli:{**ora_volo, tratta,** partenza, arrivo, aereo}

4. Risolvo la prima vìolazione (porto in 2FN)

   Tratte:{**tratta**, partenza, arrivo}

   Aerei:{**aereo**, posti}

   Voli:{**ora_volo, tratta,** aereo}

5. Verifico se è rispettata la BCNF

   Dipendenze funzionali di Tratta

   Tratta → partenza, arrivo OK perché tratta è chiave (candidata)

   Partenza, arrivo → tratta Ok perché Partenza, arrivo è chiave (candidata)

   Dipendenze funzionali di Aerei

   aereo → posti OK perché aereo è chiave (candidata)

   Dipendenze funzionali di Voli

   Ora_volo, tratta → aereo OK perché ora_volo e tratta sono attributi chiave

Il DB Schema finale rispetta le prime tre FN e la BCNF.

ES. 55 (ordini)

1. Chiave candidata e primaria: {cliente, articolo, n_ordine}
2. Dipendenze funzionali:

   cliente → indirzzo vìola 2FN

   articolo → descrizione, prezzo vìola 2FN

3. Risolvo la prima vìolazione:

   clienti: {**cliente**, indirizzo}

   Ordini: {**cliente**, **articolo**, descrizione, prezzo, **n_ordine**, quantità}

4. Risolvo la seconda vìolazione:

   clienti: {**cliente**, indirizzo}

   Articoli: {**articolo**, descrizione, prezzo}

   Ordini: {**cliente**, **articolo**, **n_ordine**, quantità}

5. Verifico BCNF

   Poiche le dipendenze funzionali presente hanno come determinante solo chiavi, la BCNF è rispettata.

Il DB Schema finale rispetta le prime tre FN e la BCNF.

ES 56 (palestra):

Osservazione: Un numero di telefono non identifica l'allievo, due allievi potrebbero avere lo stesso numero di telefono, se abitano insieme oppure se sono bambini, potrebbe essere il numero di telefono di un genitore.

1. Chiave candidata e primaria (allievo, corso)
2. Dipendenze funzionali:

   allievo → telefono vìola 2FN

   corso → insegnante, tipo_corso vìola 2 FN

   insegnante → indirizzo vìola 3FN

3. Risolvo la terza vìolazione (porto in 3FN)

   Insegnanti: {**insegnante**, indirizzo}

   Iscrizioni: {**allievo**, telefono, **corso**, tipo_corso, saldo, insegnante}

4. Risolvo le prime due vìolazioni (porto in 2FN)

   Insegnanti: {**insegnante**, indirizzo}

   Allievi:{**allievo**, telefono}

   Corsi: {**corso**, tipo_corso, insegnante}

   Iscrizioni: {**allievo**, **corso**, saldo}

5. Verifica BCNF

   L'unica che presente dubbi è la tabella Corsi. Verifichiamo se ci sono dipendenze oltre a quelle della chiave primaria:

   tipo_corso → corso ? NO perché posso avere più corsi di judo

   tipo_corso → insegnante ? NO perché posso avere due insegnanti di judo

   insegnante → corso ? NO perché un insegnante può insegnare più corsi

   insegnante → tipo_corso ? NO perché un insegnante può insegnare più tipi di corsi, ad esempio karate e judo

   Poiché vi sono solo dipendenze funzionali da attributi chiave, la BCNF è rispettata

ES 58 (computer)

58 Sono dati i seguenti schemi relazionali che rappresentano la gestione dei laboratori scolastici di informatica:

Computer(idComputer, marca, modello, fornitore, telefono_fornitore)

Installazione(idComputer, idSoftware, descrizione_software, data_installazione)

Supponendo che:

- computer di un certo modello siano identici e identificati dal loro codice di inventario;
- i computer di una stessa marca abbiano tutti lo stesso fornitore;
- uno stesso software sia in genere installato su più computer;

A. individuare le dipendenze funzionali;

B. verificare se lo schema è in BCNF, altrimenti lo si decomponga.

Computer: {**idComputer**, marca, modello, fornitore, telefono_fornitore}

Installazione: {**idComputer, id,Software**, descrizione,_software, data_installazione}

Dipendenze funzionali descritte

IdComputer → marca, modello

telefono_fornitore → fornitore vìola BCNF

Marca → fornitore vìola BCNF

idSoftware → descrizione_software

Scomposizione proposta:

fornitori: {**telefono_fornitore** , fornitore, marca}

Computer: {**id_computer**, modello, telefono_fornitore}

Software: {**id_software**, descrizione}

Installazioni: {**id_computer**, **id_software**, data_installazione}

4. CAP 4: TRANSAZIONI

I DBMS possono essere monoutente o multiutente. Se sono **multiutente** più applicazioni possono accedere ai dati contemporaneamente, se sono **monoutente** può accedere ai dati solamente una applicazione per volta. Con il termine utente si indica quindi un'applicazione sw che accede ai dati di un database gestito da un DBMS. Ad esempio il sito web di una compagnia aerea (tipo Ryanair) a cui accedono contemporaneamente più siti web (www.edreams.it, www.volaregratis.com, ecc..). Oppure il sistema informatico di una banca a cui accedono i sistemi informatici di altre banche per le operazioni di pagamento o accreditamento.

La maggior parte dei DBMS è ormai multiutente e per garantire questa modalità di accesso il BMS deve rispettare determinati requisiti per evitare i seguenti problemi.

Problema 1.

Accesso multiutente solo per **l'interrogazione di** dati (non per la loro modifica). Il problema è semplicemente di tipo prestazionale, ossia se molti utenti accedono ai dati per avere informazioni è necessario rendere veloce il tempo di elaborazione. Questo problema si risolve semplicemente scegliendo un **hardware opportuno** (ad esempio esecuzione su più server, anche centinaia, in maniera trasparente per l'utente) su cui installare il DBMS e grazie al fatto che i DBMS attuali **implementano delle sofisticate tecniche di** ottimizzazione delle interrogazioni.

Problema 2.

Accesso concorrente in operazioni che portano alla trasformazione dei dati (inserimento, cancellazione, modifica). In questo caso il DBMS deve occuparsi di evitare le tipiche problematiche dell'accesso concorrente che potrebbero causare dati errati che vìolino i vincoli di integrità.

In questo caso i problemi possono essere tipicamente di due tipi:

- Due utenti accedono contemporaneamente allo stesso dato con l'intenzione di aggiornarlo. Come abbiamo visto in TPS parlando di programmazione concorrente, una delle due operazioni potrebbe andare "persa" generando un errore nella base di dati:

  Esempio1: Conto corrente (visto l'anno scorso in TPS):

  l'applicazione di una banca (utente A) svolge un'operazione di accredito di un conto corrente mentre un operazione con POS in un negozio (utente B) svolge contemporaneamente un'operazione di prelievo sullo stesso conto corrente. Una delle operazioni potrebbe "andare persa" a causa della "non atomicità" delle operazioni a basso livello.

Per rendere ancora più evidente il problema immaginiamoci il conto corrente di un grosso supermercato, in cui le operazioni contemporanee di accredito (pagamenti con POS alle casse) e addebito (ad esempio pagamenti dei fornitori eseguiti dagli impiegati) possono essere molto frequenti.

Esempio2: prenotazione di un posto su un volo aereo:

Se due (o più) utenti tentano contemporaneamente di prenotare lo stesso posto, solamente una operazione deve poter andare a buon fine.

Esempio3: sito di ecommerce:

Se due (o più) utenti tentano contemporaneamente di acquistare un prodotto del quale è rimasto un solo esemplare, una sola operazione deve concludersi con il pagamento e l'effettivo acquisto del prodotto.

- Un'applicazione software deve effettuare modifiche multiple, ma logicamente correlate fra loro, tanto che tutte devono essere annullate se anche solo una di esse non ha esito positivo.

  Esempio:

  Una semplice operazione di pagamento che avviene fra due conti correnti di una stessa banca, quindi sullo stesso database, deve portare alla modifica sia del saldo sul conto corrente da cui viene effettuato il pagamento, sia del saldo sul conto corrente su cui viene effettuato il pagamento.

Per scongiurare errori dovuti a queste problematiche è necessario che l'elaborazione dei dati da parte del DBMS avvenga in maniera atomica. Questo è ciò avviene con le **transazioni**.

Una **transazione** è una **sequenza di operazioni** eseguite su una base di dati che, se ha successo, trasforma lo stato della base di dati da uno stato **consistente "Si" (stato iniziale)** ad un altro **stato consistente "Sn" (stato nuovo).**

Consistente significa "i cui dati non violano i vincoli di integrità".

La transazione può comprendere una o piu' modifiche dei dati e, se anche una sola delle operazioni della transazione non ha successo, la base di dati viene riportata nello stato iniziale Si.

Come verrà illustrato più dettagliatamente quando si studierà il linguaggio SQL, generalmente i DBMS di default funzionano in modalità AUTO-COMMIT, ciò significa che ogni comando

SQL è implicitamente seguito da un COMIT. In questo modo ogni comando SQL genera una transazione. In alcuni casi è necessario che una transazione contenga più comandi SQL, in questo caso si può disattivare l'opzione AUTO-COMMIT in modo che il COMMIT, quindi la transazione, deve essere ESPLICITAMENTE gestito dal programmatore con un apposito comando.

Ogni transazione (quindi la modifica di uno o più dati) è formata dalle seguenti fasi operative:

1. BOT: begin of transation.
2. Si eseguono le operazioni di modifica dei dati (comandi SQL: INSERT, UPDATE, DELETE) Queste operazioni avvengono all'interno della transazione e sono memorizzate in un file di log delle transazioni
3. Se il DBMS riscontra qualche vìolazione dell'integrità dei dati, esegue un'istruzione detta di **abort**, per abortire la transazione. L'annullamento delle modifiche apportate è detto "**rollback**" e riporta allo stato iniziale (prima del BOT) la base di dati.
4. Se tutte le operazioni vengono eseguite senza riscontrare vìolazioni della base di dati, si esegue un'istruzione detta di "**commit**", per confermare la transazione.

Per implementare un transazione, tipicamente si usa un'apposita area d' appoggio del disco fisso in cui vengono copiati i dati originali appena prima di essere modificati. Quando viene eseguito un commit, la copia dei dati viene eliminata. Quando viene eseguito un rollback, i dati originali vengono ripristinati. Pertanto, un commit è più efficiente di un rollback (poiché quest'ultimo deve copiare i dati una volta in più).

Un DBMS che implementa le transazioni garantisce le quattro proprietà note con l'acronimo **ACID** (Atomicity, Consistency, Isolation, Durability):

Atomicity (atomicità): le operazioni di cui è composta la transazione devono avvenire in maniera atomica, quindi "tutte o nessuna". Non sono ammesse esecuzioni parziali.

Consistency (**consistenza o coerenza, dipende da come viene tradotto il termine inglese coonsistency**): Lo stato della base di dati a seguito di una transazione deve essere consistente ossia i dati non devono violare i vincoli di integrità.

Isolation (isolamento): ogni transazione deve avvenire in maniera indipendente dalle altre transazioni in esecuzione contemporaneamente. L'eventuale fallimento di una transazione non deve interferire su una transazione in esecuzione. **Il risultato di più transazioni concorrenti deve essere lo stesso che si otterrebbe se le transazioni non avvenissero contemporaneamente**. Esempio: alle casse del

supermercato avvengono contemporaneamente due pagamenti con il POS, ogni pagamento è una transazione, il saldo iniziale sul conto corrente prima delle due transazioni è di 1000 €,

- La transazione T1 inizia un accredito di 100 € portando il saldo a 1100 € ma non ha ancora eseguito il commit.
- La transazione T2 legge il saldo (1100 €) e accredita un pagamento di 50 € portando il saldo a 1150 € ed esegue il commit
- La transazione T1 non riesce ad eseguire il commit e quindi tale transazione viene abortita.
- Il saldo risulta sbagliato (1150 € anziché 1050 €).

Il principio di isolation garantisce che questo non avvenga.

Durability (durabilità o persistenza): una volta eseguito il commit, i dati devono essere scritti in maniera permanente nella base di dati.

Ma chi si occupa dell'esecuzione delle transazioni e che i requisiti ACID vengano rispettati?

Se ne occupa il DMBS (questo è uno dei numerosi vantaggi dell'utilizzo di un DBMS per la gestione di un database). In particolare il **componente software** del DBMS che si occupa della gestione delle transazioni si chiama **Transaction Processing** ed è a sua volta suddiviso in due moduli**:**

- il **CCM** (**Concurrency Control Management**) che si occupa di garantire l'atomicità e l'isolamento. (per garantire l'isolamento il CCM utilizza opportuni lock su alcune risorse del database) Questo modulo si occupa anche di risolvere eventuali problemi di **deadlock** facendo abortire una o più transazioni quando si verificano le operazioni di stallo (ad esempio: la transazione A non può modificare la tabella T1 perché è bloccata dalla transazione B, B non può modificare la tabella T2 perché è bloccata dalla transazione A).
- il **LRM** (**Local Recovery Management**) che si occupa di garantire la consistenza e la persistenza (o durabilità). Questo modulo si occupa di memorizzare su opportuni file di *log* ogni singola operazione svolta sulla base di dati e di **ripristinare** la base di dati in uno stato consistente nel caso di violazioni dei vincoli di integrità o malfunzionamento (ad esempio un *crash* durante la scrittura dei dati).