

BASI DI DATI

3 livelli: Presentazione - Applicazioni - Dati
Modello relazionale: ogni relazione è composta da attributi che hanno un nome e un dominio, si può rendere come una tabella con intestazione e varie tuple.

$t[A]$ indica il valore che la tupla t ha in corrispondenza dell'attributo A .
Una tupla si indica $\langle A_1:a, A_2:b, \dots \rangle$ dove A_1, A_2 sono i nomi degli attributi e a, b i valori che la tupla assume in corrispondenza.
In una relazione l'ordinamento di righe o colonne non è rilevante.

Schema di Relazione: $R(A_1:D_1, A_2:D_2, \dots)$ A attributi, D domini
Schema di Basi di Dati $R = \{R_1(x_1), \dots, R_n(x_n)\}$

Istanta di relazione su schema $R(x)$ è l'insieme di n -uple.
Istanta di base di dati su schema $R(x)$: insieme di relazioni.
L'algebra relazionale è un linguaggio di interrogazione procedurale.

OPERATORI:

- ridenominazione: $REN_{A_1 \leftarrow B_1, A_2 \leftarrow B_2} \circ REN_{A_1, A_2 \leftarrow B_1, B_2}$
sostituisce A_1, A_2 a B_1, B_2 (solo nome dell'attributo)
- selezione: $SEL_{\text{condizione}}(\text{Operando})$
restituisce un sottoinsieme delle tuple dell'operando che soddisfa la condizione indicata.
- proiezione: $PROJ_{\text{listAttributi}}(\text{Operando})$
restituisce un risultato con solo alcuni attributi dell'operando eliminando eventuali tuple duplicate.
- JOIN: $R_1 \text{ JOIN } R_2$
prendere due tabelle e produce un risultato sull'unione degli attributi degli operandi, con esempi costruite ciascuna a partire da una esemplifica di ognuno degli operandi. Due tuple si fondono per formarne una se hanno lo stesso valore nei campi in comune. Se non ci sono campi in comune ottengono

il prodotto cartesiano. Un Join è completo se ogni n-upla contribuisce al risultato. Un theta-join funziona se due relazioni non hanno attributi in comune, è la selezione su una condizione di un join naturale e si indica: SEL condizione ($R_1 \text{ JOIN } R_2$) o $R_1 \text{ JOIN}_{\text{condizione}} R_2$. Se nella condizione è usato solo l'operatore = si tratta di equi-join. Il modello relazionale non può affrontare informazioni incomplete per cui viene introdotto il valore NULL. Nelle condizioni si può usare ATT IS NULL o ATT IS NOT NULL.

- JOIN esterno estende il join a tuple (eventualmente tagliate) inserendo valori nulli, può essere JOIN LEFT, JOIN RIGHT o JOIN FULL.

- Un vincolo di integrità è una condizione che deve essere soddisfatta da tutte le istanze della base di dati. Ogni vincolo è una funzione booleana che associa ad ogni istanza della base di dati un valore vero o falso (se il vincolo è soddisfatto o no).

• I vincoli possono essere intrarelazionali:

• DI TUPLA: vale per ogni tupla ed è fra gli attributi, se coinvolge un solo attributo si può dire DI DOMINIO.

• DI CHIAVE: una chiave è un insieme non vuoto di attributi che identificano univocamente le n-uple di una relazione. Una superchiave minima (se tolgo un attributo non è più superchiave) è anche detta CHIAVE e ogni schema di relazione ne ha almeno una. Se definisco un vincolo di CHIAVE PRIMARIA vieto che gli attributi della chiave siano null, può essercene una sola.

o interrelazionali (di integrità referenziale):

• DI FOREIGN KEY: per un attributo di una relazione possono esserci solo valori già presenti almeno in un attributo di un'altra relazione (che deve/devono formare una chiave). I valori nulli non violano.

• DI INCLUSIONE: è come foreign key ma gli attributi a cui mi riferiscono non devono essere necessariamente chiavi.

SQL

SELECT attributi FROM tabella WHERE condizione → può essere IS NULL o IS NOT NULL

↓
PROJ attributo (SEL condizione (tabella))

si può omettere

SELECT p.nome AS nome FROM persone AS p → Risdenominazione con AS

SELECT * FROM tabella → non fa proiezione.

SELECT DISTINCT attributi → elimina tuple duplicate.

SELECT numero, numero-2 → mette nella seconda colonna i valori calcolati

SELECT R1.A1, R2.A4 FROM R1, R2 WHERE R1.A2 = R2.A3 è equijoin

SELECT * FROM R1 AS R1, R2 AS R2 WHERE R1.A = R2.B è SELF-JOIN

e si indica in algebra relazionale con $R \text{ JOIN}_A=B_1(R \text{ ENA}_1, B_1 \leftarrow A, B(R))$

Nel caso in cui ho

A	una	Elena
Elena	una	Anna

 e ne voglio solo una tra B1.nome < B2.nome.

SELECT ... FROM T1 JOIN T2 ON condizione → fa join naturale, non serve specificare l'attributo su cui fare il JOIN se ce ne è uno in comune.

LEFT OUTER (o RIGHT, o FULL, o LEFT) JOIN ... ON ... aggiunge le tuple non comprese nel JOIN mettendo NULL dove necessario.

ORDER BY attributo [DESC] → ordina le tuple, posso avere ORDER BY A1, A2, ...

LIMIT n → limita il risultato a n tuple

COUNT(*) conta il numero di tuple e si mette nel select

COUNT(Attributo) Conta * i valori che ha l'attributo (anche duplicati)

COUNT(DISTINCT Attributo) non conta i duplicati.

SUM, AVG, MAX, MIN prendono come parametro un attributo o espressione

→ restituiscono un solo valore per cui una sola tupla, nel select non posso avere attributi che ne restituiscono di più.

SELECT Studente, AVG(Voto) AS MEDIA FROM Esame GROUP BY Studente
raggruppa i voti di ogni studente e ne calcola la media.

GROUP BY può avere anche più attributi.

Se utilizzo GROUP BY posso inserire nella select solo attributi presenti in group by o aggregati.

HAVING condizione si usa solo dopo group by e seleziona alcuni gruppi.

`SELECT ... UNION [ALL] SELECT ...` unisce le due tabella eliminando i duplicati (con ALL li mantiene)

`EXCEPT [ALL]` fa la differenza `INTERSECT [ALL]` fa l'intersezione.

• `CREATE TABLE nome (NomeAtt DOMINIO [vincoli] ... [vincoli generali])`
Posso dichiarare un vincolo con `CONSTRAINT <nome>` per dare nome esplicito.

Vincoli intrarelazionali: `NOT NULL`, superchiave \rightarrow `UNIQUE (att1, att2, ...)`

o anche solo `UNIQUE` dopo il dominio se è solo un attributo, `PRIMARY KEY` ha la stessa sintassi di `UNIQUE` e implica `not null, CHECK()` per vincoli di tuple con la condizione fra parentesi.

Vincoli interrelazionali: `CHECK`, `REFERENCES tabella (attributo)`
Se inserito dopo dominio in un attributo, `FOREIGN KEY (att1, att2)`
`REFERENCES tabella (att1, att2)` Se in fondo alla create.

Se due tabelle hanno references fra di loro devo creare la prima senza references e poi usare `ALTER TABLE nome ADD FOREIGN KEY ...`

• ~~REPLACE~~ `INSERT INTO tabella VALUES (valori), VALUES (valori)`
`INSERT INTO tabella SELECT ...`

• `DELETE FROM tabella WHERE Conditione`

• `UPDATE tabella SET Attributo = <espressione | select | null | default> WHERE cond.`

La `SELECT` può essere dentro una `WHERE` o una `FROM`, posso usare
attributo = `ANY (select ...)` per controllare se il valore è in una lista,
`= ALL (select ...)` per controllare se il valore è uguale a tutti gli elementi,
`exists select` per verificare se è vuota o no. Posso usare anche
`!= ANY` o `!= ALL` o `> ANY`.

Se uso la `select` nel `FROM` posso indicarla con un nome e aggiungere
~~del~~ ~~del~~ `WITH nome AS (select ...)` prima della query.

Una VISTA è una tabella virtuale \rightarrow `CREATE VIEW nome AS select...`

viene calcolata solo se usata con il suo alias e non è mai memorizzata.

Una TRANSAZIONE è un insieme di operazioni da considerare indivisibile, corretto anche in presenza di concorrenza e con effetti definitivi. Proprietà ACID: Atomicità, Consistenza, Isolamento e Durabilità. Ogni comando è anche una transazione.

BEGIN: specifica inizio transazione

COMMIT (o END): rende permanenti le operazioni eseguite.

ROLLBACK: annulla l'effetto di tutte le operazioni dal begin.

- Posso specificare comportamenti ai vincoli di foreign key da seguire se la tupla a cui mi riferisco viene eliminata o modificata.
FK (attributi) REF tabella (Attributi) ON DELETE ... / ON UPDATE ...

dove al posto di ... posso mettere: no action (non fa eliminare il padre), restrict (errore immediato anche in transazione), cascade (elimina le tuple che referenziano quella eliminata), set default (si sostituisce il valore con quello di default), set null (si sostituisce con NULL).

- Un vincolo può essere definito con la specifica DEFERRABLE. In questo caso all'inizio di una transazione posso usare il comando SET CONSTRAINTS nome DEFERRED (o ALL al posto di nome) per far sì che i vincoli deferred siano controllati solo alla fine della trans.

- CREATE FUNCTION nome (arg₁ arg-type, ...)

RETURNS { return-type | TABLE (col₁ col-type, ...) | TRIGGER } AS

\$\$ DECLARE dichiarazioni variabili BEGIN istruzioni END; \$\$ plpgsql
se il return-type è VOID la chiama procedura, altrimenti nelle istruzioni avrò RETURN ...

Possiamo creare una funzione ed associarla a un TRIGGER che la eseguirà in automatico quando viene eseguita una operazione specifica.

CREATE TRIGGER nome BEFORE INSERT/UPDATE/DELETE ON tabella FOR EACH ROW EXECUTE PROCEDURE nomefunzione(); dovendo restituire un TRIGGER.

Con INSERT posso usare NEW, con UPDATE OLD e NEW, con DELETE solo OLD
all'interno della funzione per riferirmi alla tupla su cui opero.

- Strutture comuni: IF.. THEN.. ELSE.. END IF

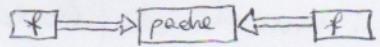
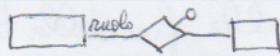
FOR r IN select... LOOP ... END LOOP

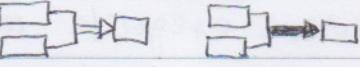
PROGETTAZIONE di una BASE di DATI.

Requisiti → PROGETTAZIONE CONCETTUALE → schema concettuale → PROGETTAZIONE LOGICA → schema logico → PROGETTAZIONE FISICA → schema fisico. Lo schema concettuale è formulato nel modello Entità-Relazione (ER), lo schema logico nel modello relazionale con vincoli di integrità.

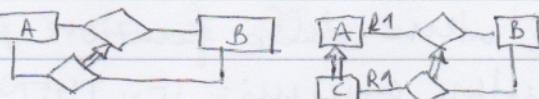
Il modello ER è un modello di dati che prevede una distinzione fra livello intenzionale (schema concettuale) ed estensionale (istanze).

- ENTITÀ: è una classe di oggetti di interesse per l'applicazione, ha nome univoco e proprietà. Un attributo di entità è una sua proprietà e associa un valore appartenente a un dominio. Un attributo composto ha come dominio un tipo record.
- RELAZIONE: si definisce fra 2 o più entità e rappresenta un legame. È un sottoinsieme del prodotto cartesiano fra le istanze delle entità. Anche le relazioni possono avere attributi MA non possono avere una relazione fra due ^{stesse} entità due volte con diversi valori di attributo. Bisogna indicare i ruoli, se non espliciti si sottointendono i nomi delle entità.



- ISA: istanza di una entità figlia è anche istanza dell'entità padre, una entità può avere al massimo un padre. Ogni figlia ha gli attributi del padre e può partecipare alle relazioni in cui è coinvolto il padre. Una entità può avere più figlie, le quali possono avere istanze in comune.
- GENERALIZZAZIONE: indica che i sottousceni dell'entità padre sono disgiunti e può essere completa o no (se l'unione delle figlie è uguale alle istanze del padre).  La generalizzazione è completa se la freccia è riempita.

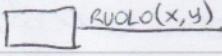
- ISA e GENERALIZZAZIONI fra relazioni sono consentite solo se hanno lo stesso grado (numero di entità coinvolte), gli stessi ruoli e ogni ruolo è lo stesso per entità padre e figlia.

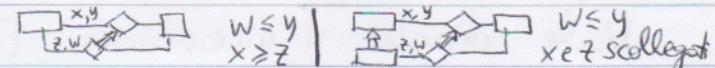


(due attributi con lo stesso nome sono la stessa funzione).

• VINCOLI DI INTEGRITÀ (cardinalità, identificazione ed estenu):

- VINCOLI DI CARDINALITÀ: si associa al ruolo in una relazione e impone un limite minimo e massimo di istanze della relazione a cui può partecipare ogni istanza dell'entità nel ruolo.

 $x \in \text{intero} \geq 0, y \in \text{intero} \geq x$ o "n" per indicare infinito.

La mancanza del vincolo equivale a $(0, n)$. Una sottosentita eredita la cardinalità. Nel caso ci sia una sottorelazione la cardinalità massima è ereditata ~~ma~~ ma si può impostare più stringente, la minima è scorrrelata nel caso in cui si tratta di entità e sottosentita o è \geq quella della relazione padre. 

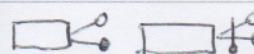
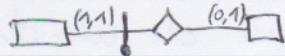
NEL CASO DI VINCOLI DI CARDINALITÀ SU ATTRIBUTI si intende di default $(1,1)$.

se la cardinalità massima non è 1 l'attributo è multivalore, se la cardinalità minima è 0 la funzione non è più totale (optionalità).

Ogni istanza di entità ha almeno n e massimo m attributi diversi e compare fra n e m volte nella relazione.

• VINCOLI DI IDENTIFICAZIONE di entità: un identificatore di una entità è un insieme di proprietà (attributi e relazioni) che identifica univocamente le istanze dell'entità. Un vincolo di identificazione per una entità E definisce un identificatore per E. Posso definire un numero arbitrario di vincoli di identificazione sulla mia entità:

• Può essere interno (solo su attributi) o esterno (comprende solo o anche ruoli di relazioni). Tutti gli attributi e ruoli che concorrono ad un identificatore di entità devono avere cardinalità $(1,1)$. Una entità con identificatore esterno viene detta debole.

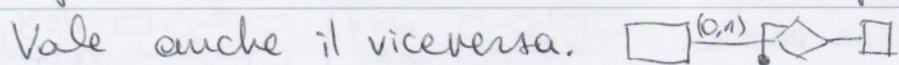
 Se in una relazione ho una entità con cardinalità $(1,1)$ e l'altra con cardinalità massima 1 allora posso dedurre che la prima ha un identificatore di entità sul ruolo della relazione. 

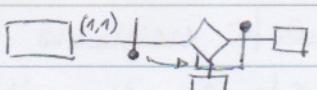
Indico un identificatore come non essenziale se posso ~~non~~ escludere una proprietà e farla rimanere identificatore, lo indico essenziale altrimenti.

- VINCOLI DI IDENTIFICAZIONE di relazione: hanno lo stesso significato di quelli di entità. È implicito l'identificatore formato da tutti i ruoli che partecipano alla relazione. Possono includere sia ruoli che attributi della relazione.

Judica che non esistono due istanze della relazione con gli stessi valori per le proprietà che compongono l'identificatore. Esistono dei vincoli derivati di identificazione di relazione:

- se un'entità partecipa ad una relazione con cardinalità massima uguale ad 1, il suo ruolo è identificatore della relazione.

Vale anche il viceversa. 

- se un'entità ha una relazione e il ruolo costituisce un identificatore per l'entità, gli altri ruoli della relazione saranno identificatori per la relazione. 

- se ho una relazione ternaria ~~ma~~ in cui una entità ha cardinalità massima 1 e un'altra ha cardinalità (1,1) avrò sulla seconda un identificatore di entità con il ruolo.

- VINCOLI ESTERNI: sono vincoli non esprimibili con il modello ER che vengono inseriti come testo.

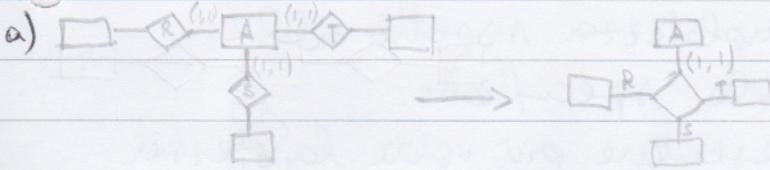
→ Oltre al diagramma ER, lo schema concettuale comprende il dizionario dei dati con 4 tavole: entità, relazioni, attributi e vincoli esterni.

La progettazione concettuale comprende 4 attività:

- Strutturazione e organizzazione dei requisiti raccolti
- Costruzione di un glossario schematico (optional)
- costruzione schema concettuale (diagramma e dizionario dei dati)
- controllo di qualità

Nella costruzione di un diagramma e dizionario dei dati (modello ER) bisogna scegliere il costrutto adatto a ogni concetto, dei pattern di modellazione e operazioni di trasformazioni dello schema (equivalenti).

Pattern:



Penso accorpate relazioni binarie in una relazione n-aria che avrà come ruoli i nomi delle relazioni originali. Si può fare se ha senso concettualmente.

b)
 Rappresenta un evento V che si verifica al massimo una volta con le stesse istanze delle due entità.

c)
 Rappresenta un evento V che si può verificare più volte con le stesse istanze di E1 ed E2. Aggiungendo un identificatore di entità su V che comprende le due relazioni e l'attributo sto intendendo che l'evento non capiterà mai due volte con quelle stesse caratteristiche.

d)
 Rappresentazione storizzata di entità, ci permette di tenere traccia nel tempo di alcune proprietà dell'entità. Le proprietà A₁, ..., A_n non variano nel tempo mentre le proprietà B₁, ..., B_m sono quelle che possono variare e di cui vogliamo tenere traccia. Le proprietà B₁, ..., B_m possono essere anche relazioni.

Trasformazioni:

a)
 Trasforma l'attributo semplice di una entità in una relazione se E è l'unica entità con attributo A, se sono di più dovranno collegarle alla stessa F.

b)
 Trasforma l'attributo composto di una entità in una nuova entità e le connette con una relazione.

c)
 Trasforma una relazione in una entità, nel caso in cui ci sia una sotto-relazione sulle stesse entità la aggiungo come sotto-entità di R.

d) Trasforma l'attributo di una relazione in una relazione e avviene eseguendo prima c) e poi a).

- Controllo di qualità: prevede il controllo di tre fattori
- correttezza: adeguatezza e completezza rispetto alle regole del modello ER e alle specifiche.
 - minimosità: evitare di rappresentare più volte la stessa proprietà e documentare nei vincoli esterni eventuali ridondanze che si ~~sviluppano~~ sviluppavano. vuole mantenere. Le ridondanze possono essere intenzionali e si risolvono "semplificando" lo schema, se ho una generalizzazione^{completa} in cui le due sottointenti hanno lo stesso attributo posso spostarlo sull'entità padre. Le ridondanze estensionali possono essere di due tipi, nel primo posso calcolare il valore di un attributo da altre proprietà; nel secondo posso ottenere una relazione passando da altre relazioni. In ogni caso bisogna documentare fra i vincoli esterni le ridondanze estensionali.
 - leggibilità: produrre un diagramma con pochi incroci e rappresentare le proprietà con vincoli esplicativi e non esterni se possibile.

La progettazione logica prende in input uno schema concettuale e restituisce lo schema logico usando il modello relazionale con alcuni vincoli. Deve esprimere tutti i concetti espresi dallo schema concettuale senza ignorare nulla. È necessario considerare le prestazioni utilizzando un modello di costo che tiene conto del tempo di esecuzione e dello spazio di memoria. Serve quindi conoscere il volume dei dati e le caratteristiche delle operazioni che verranno eseguite.

FASI della PROGETTAZIONE LOGICA:

1. ristrutturazione schema ER: eliminazione dei costituti non direttamente traducibili, scelta degli identificatori principali di entità e relazioni. Ottengo uno schema ER ristrutturato.

2. traduzione diretta: è eseguita automaticamente senza scelte del progettista, lo schema logico/relazionale prodotto non contiene ridondanze se non volute. Ottengo uno schema logico.

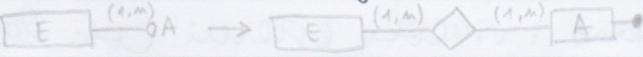
3. ristrutturazione dello schema relazionale: richiede scelte del progettista per ottimizzare le prestazioni.

1 → ristrutturazione dello schema ER in 8 fasi

a) si rendono esplicativi alcuni vincoli rilevanti a discrezione del progettista.

b) analisi delle ridondanze: si valuta se mantenere ridondanze estensionali aggiungendone eventualmente altre per efficienza.

c) eliminazione attributi multivalore (1,n): vengono sostituiti da entità legate con una relazione.



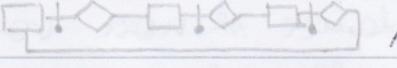
d) eliminazione attributi composti che vengono spostati direttamente sull'entità con eventuale vincolo nel caso cardinalità sia (0,1) che dice che devono essere presenti tutti o nessuno. Otrimenti trasformazione b.

e) eliminazione ISA e generalizzazioni:

The diagram shows the transformation of an ISA relationship. On the left, there are two entity boxes, E and F, with lines labeled '0..1' and '0..1' respectively, pointing to a diamond-shaped box labeled 'ISA-E-F'. This diamond box has two lines labeled '(0..1)' pointing to it from E and F. An arrow points from this configuration to the right. On the right, the entity boxes E and F are shown again, but now they are connected by a line labeled '(1..1)' (total participation), indicating that every instance of F is also an instance of E. Below this, there is another entity box labeled 'E' with a line labeled '0..1' pointing to it.

Ora E ed F sono disgiunti. Se invece avessi una generalizzazione dove nei aggiungere un'altra relazione analogia ad F con il vincolo che nessuna istanza di F partecipa ad entrambe le relazioni (se non completa) o che ogni istanza di F partecipa ad una oppure all'altra relazione ma non ad entrambe (se completa). Nel caso in cui ho due sottoentità con lo stesso attributo devo inserire il vincolo esterno per cui avranno lo stesso valore se legate alla stessa istanza dell'entità padre. Per eliminare una ISA fra relazioni aggiungo un vincolo esterno per cui ogni istanza di R1 è anche di R2.

f) scelta degli identificatori principali di entità e relazioni:
per ogni entità è necessario individuare un identificatore principale, in assenza di identificatori se ne aggiunge uno artificiale (come un codice). Si indica \rightarrow ma non è necessario indicare il principale se è l'unico. Si preferiscono identificatori essenziali, formati da pochi elementi, usati più spesso per l'accesso nelle operazioni, preferisco quelli interni e fra gli esterni quelli derivati da una eliminazione di ISA.

Bisogna inoltre eliminare eventuali cicli di identificazione esterna.  non va bene perché ognuno usa l'altro per identificarsi.  non va bene neanche se in una relazione ho altre entità o se un identificatore prende anche attributi.

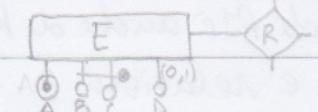
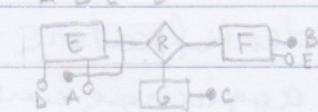
Per ogni relazione che non partecipa ad un identificatore principale di entità è necessario scegliere l'identificatore principale. Se non ne ha allora si può ~~utilizzare esplicitamente~~ utilizzare quello implicito; se invece ne ha ma preferisco usare quello implicito allora dovrò renderlo esplicito.

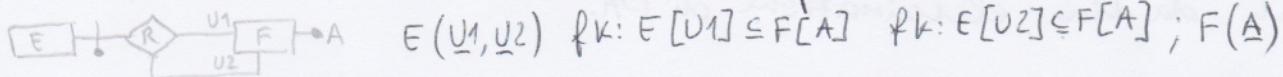
- g) Si riportano i vincoli esterni con le eventuali aggiunte fatte
- h) Si riformulano operazioni e specifiche sul carico applicativo in termini dello schema ristrutturato.

2 → Traduzione diretta composta da 4 attività, ovvero traduzione di entità e relazioni dello schema ER in relazioni dello schema logico, traduzione dei vincoli esterni e riformulazione di operazioni e specifiche sul carico applicativo.

Notazione: NomeRel (attchiave1, attchiave2, attnull*, attnonnull*)
 vincoli: [foreign key: Rel[att1, att2] ⊆ altrarel[att3, att4]]
 [inclusione: Rel[att1] ⊆ altrarel[att2]]
 [chiave: attnonnull]

Di base ogni entità e relazione diventerà una relazione, i identificatori privati saranno chiavi, i ruoli delle relazioni saranno foreign key e gli attributi saranno attributi della relazione.

- a)  diventa $E(A^*, B, C, D^*)$ chiave: B,C
 traduzione di entità senza accorpamento (idee: princ. interno)
- b)  diventa $E(A, F, G, D)$ foreign key: $E[F] \subseteq F[B]$ fk: $E[G] \subseteq G[C]$; $G(C)$; $F(B, E)$; traduzione con accorpamento
- c) se ho un attributo sulla relazione lo inserisco nella entità che l'ha accorpato.
- d) se una relazione lega la stessa entità con due ruoli (e non è quella che ha accorpato) aggiungerò a quella che ha accorpato i due ruoli come attributi al posto del nome dell'entità.



- minima pari a 1,
- e) la cardinalità (1,n) diventerà un vincolo di inclusione: $E(G) \text{ fk: } E[G] \subseteq G[A]$; $G(A)$ inclusione: $G[A] \subseteq E[G]$
- f) $E(G) \text{ fk: } E[G] \subseteq G[A]$; $G(A) \text{ fk: } G[A] \subseteq E[G]$
poiché abbiano in G cardinalità minima=1 e massima=1
- g) se oltre a G avessi un'altra entità coinvolta nella relazione R con cardinalità (1,1) potrei scegliere anche l'altra come chiave primaria. Quella che non è chiave primaria deve essere chiave.
- h) nella traduzione di una ex ISA l'entità poiché non ha vincoli, la figlia ha foreign key.
- i) traduzione relazioni non accorpate. Le faccio diventare relazioni con vincoli di foreign key in base all'identificatore principale.
- j) $E(A); R(E,G) \text{ fk: } R[E] \subseteq E[A] \text{ fk: } R[G] \subseteq G[B];$
se card. (1,n) \rightarrow $G(B)$ inclusione: $G[B] \subseteq R[G]$.
- k) $E(A) \text{ fk: } E[A] \subseteq R[E]$ sarebbe inclusione per card. min 1 ma E è chiave in R quindi fk
 $G(B)$ inclusione: $G[B] \subseteq R[B]$; $R(E,G) \text{ fk: } R[E] \subseteq E[A] \text{ fk: } R[G] \subseteq G[B]$ chiave: E
- l) Indico l'identificatore di E su A ed R con un vincolo esterno poiché in E non avrò riferimenti ad R. Scrivo: nel join fra E ed R sull'attributo B, gli attributi A ed F formano una chiave 3 → Ristrutturazione dello schema logico per migliorarne l'efficienza. Per ogni relazione posso effettuare decomposizioni verticali o orizzontali per facilitare l'accesso o miste per evitare valori nulli. Posso effettuare anche accorpamenti per facilitare l'accesso (evitando join) ed eliminare relazioni o attributi inutili.

- a) decomposizione verticale R $\rightarrow R_1 [K, \alpha] \quad R_2 [K, \beta]$ $\text{fk: } R_1[K] \subseteq R_2[K]$
 $\text{fk: } R_2[K] \subseteq R_1[K]$
- b) decomposizione orizzontale R $\rightarrow R_1 [K, \alpha] \quad R_2 [K, \beta]$ $R_1[K] \cap R_2[K] = \emptyset$
- c) decomposizione mista R $\rightarrow R_1 [K, \alpha] \quad R_2 [K, \beta]$ $R_1[K] \cap R_2[K] = \emptyset$

le due tabelle R1 e R2 sono fortemente accoppiate

- d) accorpamento 1

R1	K1	X
----	----	---

R2	K2	B
----	----	---

 fk: R1[K1] ⊑ R2[K2]
 fk: R2[K2] ⊑ R1[K1] →

R	X	B
---	---	---
- e) accorpamento 2

R1	K1	X
----	----	---

R2	K2	F*
----	----	----

 fk: R2[K2] ⊑ R1[K1] →

R	X	F	B*
---	---	---	----

le due tabelle sono debolmente accoppiate. F è booleano e vale true nelle tuple corrispondenti a R2. Se B fosse not null allora F sarebbe ridondante perché un valore null in R[B] indicherebbe che la tupla viene da R1, ~~ma non~~ un valore diverso da null che viene da R2.

- f) accorpamento 3 è analogo al 2) ma ha una relazione principale (come R1) e tante secondarie (come R2) con foreign key sulla prima. Ogni tabella secondaria è debolmente accoppiata con la prima. Si può accoppare aggiungendo un attributo T che assume come valore il numero della relazione da cui proviene la tupla (ed è analogo a F).

- g) accorpamento 4

R1	K1	A
----	----	---

R2	K2	B
----	----	---

 fk: R1[A] ⊑ R2[K2] →

le tabelle R1 e R2 sono lasciamente accoppiate e diventa

- | | | | |
|----|----|---|---|
| R1 | K1 | A | B |
|----|----|---|---|

R2	K2	B
----	----	---

 fk: R1[A,B] ⊑ R2[K2,B]

- h) accorpamento per eliminare relazioni inutili

- | | | |
|----|----|---|
| R1 | K1 | A |
|----|----|---|

R2	B
----	---

 fk: R1[A] ⊑ R2[B]
inclusione: R2[B] ⊑ R1[A] →

R1	K1	A
----	----	---

- i) accorpamento per eliminare attributi inutili

- | | | | | |
|----|----|---|---|---|
| R1 | K1 | A | B | X |
|----|----|---|---|---|

 vincolo di tupla: per ogni tupla t di R1, t[A] = t[B] →

R1	K1	A	X
----	----	---	---