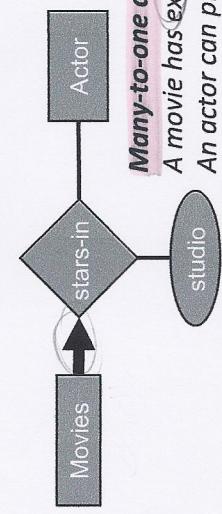
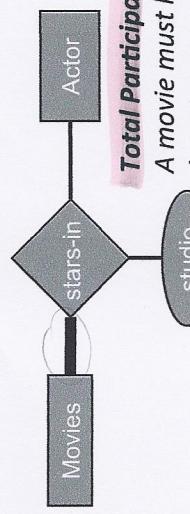
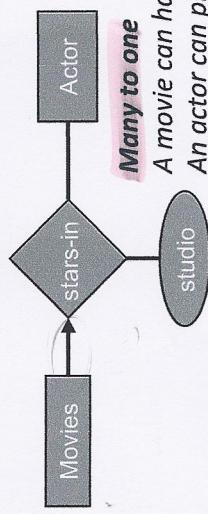
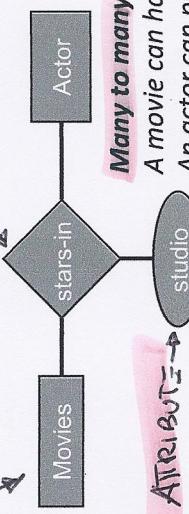
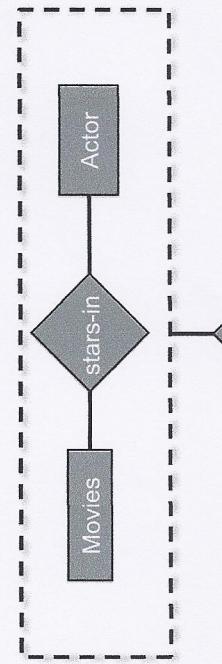
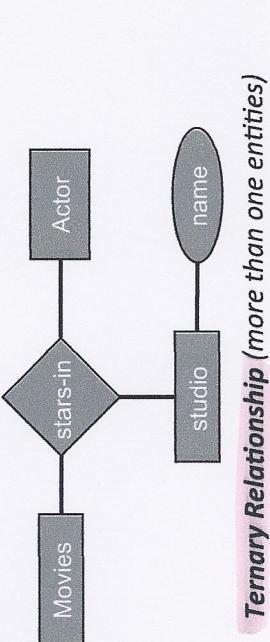
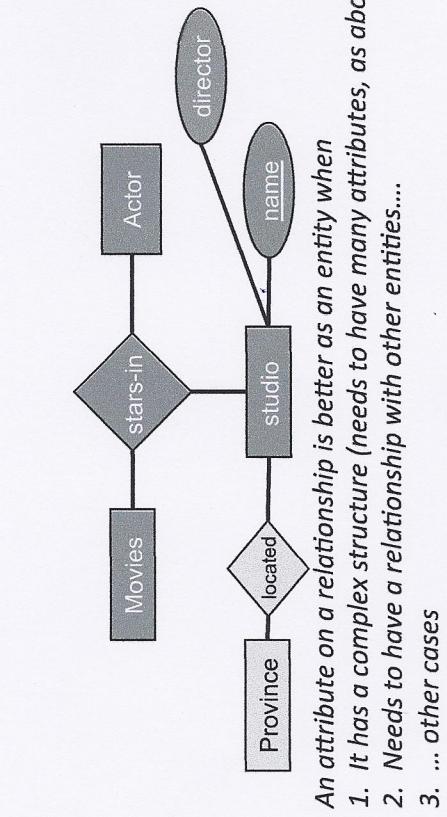


ENTITY

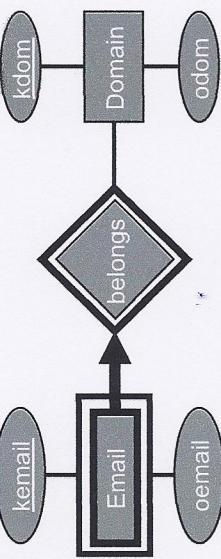


Weak Entity: Cannot exist without the main entity.
But is an entity because we need to connect to it other relationships and also because it does not connect two or more other entities to be a relationship

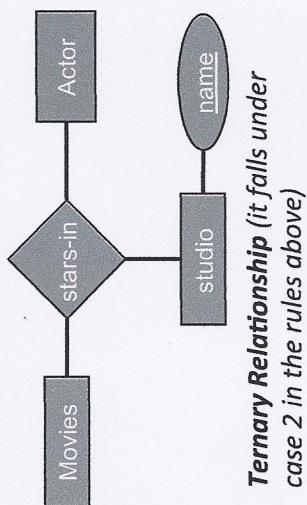


Ternary Relationship (more than one entities)

Aggregation: when entities connected are seen as 1 entity
Used when the starts-in can exist without a studio, or when the stars-in needs to be one to many to with, or when with has its own attributes.

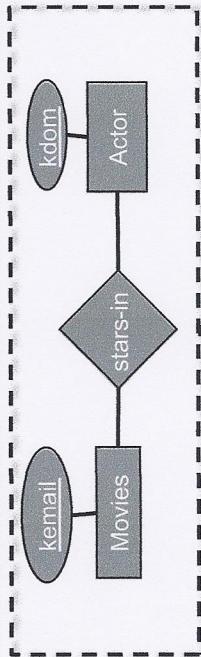


Domain(kdom,odom)
Email(kemail,oemail)

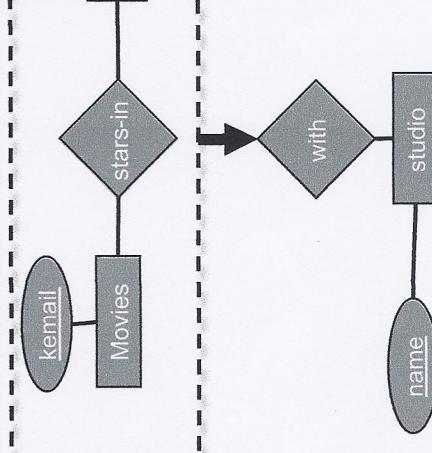


Ternary Relationship (it falls under case 2 in the rules above)

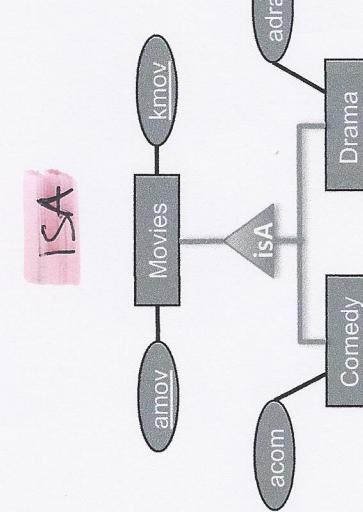
- An ER is translated to a relational as follows:**
1. Each Entity becomes a relation and its attributes columns
 2. Each Many to many Relationship becomes a relation with attributes
 - A. Attributes of the relationship
 - B. The keys of every entity it connects (these attributes become foreign keys to the respective relations derived from the entities.
 - C. The key is all the columns (attributes) created in B above
 3. Each Many to one Relationship becomes a relation as in 2. above but the key is ONLY the key of the Many part.
 4. Each Many to one and total participation relationship does not become a relation by itself, but its attributes and the keys of the "to one" entities are added in the attributes of the "Many" entity. The keys of the "to one" entity has to be foreign key for sure and has to be requested to be NOT NULL.
 5. Special cases (see figures):



with



Starsin(kemail,kdom,name)
Name cannot be null



Option 1:

Movies(kmov,amov)
Actor(name,studio)
Genre(adam,Drama)

Option 2:

Movies(kmov,amov)
Comedy(kmov,amov,acom) *kmov FK to Movies*
Drama(kmov,amov,adram) *kmov FK to Movies*

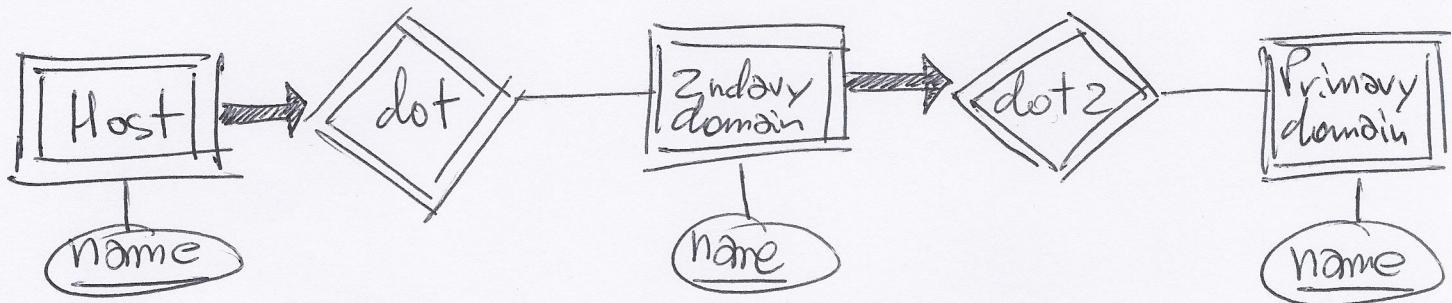
Option 3:
Movies(kmov,amov,acom,adram)

WEAK ENTITY SET

Si consideri un'entità E . È possibile che la chiave di E non provenga completamente dai suoi attributi, ma provenga ~~dalle~~ da una chiave di uno, o più, entità a cui E è collegata.

- L'insieme di queste entità è chiamato **WEAK**

Esempio: IP address consisting of primary domain, sub-domain and host.



- Key of primary domain = its name
- key of secondary domain = its name + name of primary dom.
- Key of host = its name + Key of secondary domain.

NB: Le sole volte che le entity weak sembrano necessarie:

- Quando non si riesce a creare facilmente un ID univoco
- Non c'è nessuna global authority che li assegna.

RELATIONAL DATABASE

- Relational Database : insieme di relazioni
 - Relazioni :
 - Istante : tabella con righe e colonne
 - Schema : specifica il nome della relazione con nome e tipo per ogni colonna.
- Si può pensare ad una relazione come ad un set di righe della tabella che compongono una tupla.

Esempio: tabella studente

Sid	Name	Login	Age	GPA
53666	Jones	Jones@cs	18	3.5
53688	Smith	Smith@ecs	18	3.2
53650	Smith	Smith@math	19	3.8

CREATING RELATION IN SQL

- CREATE TABLE Students


```
sid CHAR(20),
name CHAR(20),
login CHAR(10),
age INTEGER,
gpa REAL
```
- DROP TABLE Students
- ALTER TABLE Students


```
ADD COLUMN firstYear INTEGER
```

• `INSERT INTO Students (name, sid, login, age, gpa)`
`VALUES ('Smith', 53688, 'smith@ee', 18, 3.2)`

• `DELETE FROM Students S`

`WHERE S.name = 'Smith'`

INTEGRITY CONSTRAINT (ICs)

- ICs: condizione che deve essere verificata per ogni istanza del database.
 - ↳ vengono create quando lo schema viene creato
 - ↳ vengono controllate alla modifica delle relazioni tra le istanze (`INSERT`, `DELETE`)
- Un'istanza è legale se soddisfa tutte le ICs.

PRIMARY KEY

- Un insieme di colonne dell'istante possono essere considerate key se
 - (1) in quelle colonne tuple distinte non assumono mai lo stesso valore,
 - (2) qualsiasi sottoinsieme di quelle colonne non è una key

Se il punto (2) è falso \Rightarrow SUPERKEY

Se il punto (1) e (2) sono verificati \Rightarrow una delle key è scelta per essere primary key

- Ci possono essere molte colonne candidate ad essere primary key e sono specificate tramite UNIQUE.

ESEMPIO

Per ogni coppia studente - corso può esistere solo un voto

{ CREATE TABLE Enrolled (
 sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2),
 PRIMARY KEY (sid, cid)
)

(bracketo)

Un studente può tenere un solo corso e ricevere un voto per quel corso.

{ CREATE TABLE Enrolled (

 sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2),
 PRIMARY KEY (sid),
 UNIQUE (cid, grade)

Due studenti nello stesso corso non possono ricevere lo stesso voto.

)

FOREIGN KEYS E INTEGRITÀ REFERENZIALE

- FOREIGN Keys: Un insieme di colonne in una relazione che sono usate per referenziare una tupla in un'altra relazione (dove corrispondere ad una chiave primaria nella seconda relazione)

IN SQL

```
CREATE TABLE Enrolled (
    sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY(sid, cid),
    FOREIGN KEY(sid) REFERENCES Students.id )
```

In questo caso solo le tuple presenti nella tabella Students possono seguire corsi

- Se in Enrolled si cerca di inserire una tupla contenente un sid che non riferisce niente (Studente inesistente), la tupla viene rigettata.

- Che cosa succede se viene eliminato/aggiornato l'sid nella tabella Students?

- (1) Non fare nulla \Rightarrow tabelle referenziate inconsistenti
- (2) Propagare la modifica in tutte le tabelle referenziate
- (3) Settare un valore di default in tutte le tabelle referenziate

IN SQL

```
CREATE TABLE Enrolled (
    sid CHAR(20)
    cid CHAR(20)
    grade CHAR(2)
    PRIMARY KEY(sid, cid)
    FOREIGN KEY(sid) REFERENCES Students.id
    { ON DELETE CASCADE
    { ON UPDATE SET DEFAULT })
```

- Sei non specificate \Rightarrow NO ACTION
- CASCADE \Rightarrow propaga l'eliminazione/UPDATE a tutte le tabelle referenziate
- SET NULL
SET DEFAULT \Rightarrow setta a NULL o un valore di DEFAULT tutte le FK referenziate.

RELATIONAL ALGEBRA

- Query Language: permette la manipolazione di dati del database.

Operazioni Base:

- Selection (σ): seleziona un insieme di righe
- Projection (π): seleziona un insieme di colonne
- Cross-Product (\times): prodotto cartesiano tra relazioni
- Set-Difference (-): tuple nella relazione 1 ma non nella 2
- Union (\cup): tuple nella relazione 1 unite a quelle nella 2
- Intersection (\cap): tuple nella relazione 1 e nella 2 nella stesso tempo
- Join (\bowtie): combinazioni di relazioni
- Renaming (ρ): ridenominazione di relazione.

Suppliers (s_id:integer, share:string, address:string)

Parts (pid:integer, pname:string, color:string)

Catalog (s_id:integer, pid:integer; cost:real)

- Find the name of Supplies who supply some red part

$\rho(p, \prod_{pid} \text{Parts} \mid_{color=red})$

$\prod_{share} \left(\left(\prod_{sid} (\text{Catalog} \bowtie p) \right) \bowtie \text{Supplies} \right)$

~~- Find the parts supplied by supplies who supply some red part~~

- Find lids of Supplies that who ~~not~~ supply every part

$\prod_{pid} \left(\text{Catalog} \right) / \prod_{pid} \left(\text{Parts} \right)$

- Find the pids of parts supplied by ~~at least two~~ different suppliers.

$\rho(R1, \text{Catalog}) \cap \rho(R2, \text{Catalog})$

$\prod_{pid} \left(\sigma_{R1.sid \neq R2.sid, R1.pid = R2.pid} R1 \times R2 \right)$

- Find the supplier more of suppliers who supply a red part that costs less than 100\$.

$P(S, \text{T}_{\text{pid}}(\text{Suppliers}))$

$P(C, \text{T}_{\text{pid}}(6^{\text{color-red}}_{\text{Parts}}))$

~~$\text{Ti}_{\text{shape}}(P \times (\text{G} \times S))$~~

~~the price X 100 < 100~~

Esercizio 1 – Navi da battaglia

Sia dato lo schema:

classes (class, type, country, num_guns, bore, displacement)
ships (name, class, launched)
outcomes (ship, battle, result)
battles (name, date)

Esprimere in SQL le seguenti query:

- 1. Elencare nome classe e paese di tutte le classi con almeno 12 cannoni
- 2. Elencare i nomi delle navi varate prima del 1915, cambiando *name* in *shipName*
- 3. Elencare i nomi delle navi che iniziano con 'M'
- 4. Elencare gli affondamenti in battaglia (nome nave e nome battaglia)
- 5. Elencare le navi che hanno lo stesso nome della rispettiva classe
- 6. Elencare i nomi delle navi il cui nome è composto da almeno tre parole
- 7. Elencare nome, stazza e numero cannoni delle navi della battaglia di Guadalcanal
- 8. Elencare le navi con stazza maggiore di 37000 t
- 9. Elencare le navi presenti nel DB (attenzione a navi in Outcomes non in Ships)
- 10. Trovare le battaglie con almeno due navi dello stesso paese
- 11. Trovare i paesi che avevano sia corazzate (bb) che incrociatori (bc)

Esercizio 2 – Prodotti

Sempre a partire dallo schema:

product (model, maker, type)
pc (model, speed, ram, hd, price)
laptop (model, speed, ram, hd, screen, price)
printer (model, color, type, price)

Esprimere in SQL le seguenti query facendo uso di sotto-query:

- 1. Trovare i produttori di laptop veloci almeno 2.0 GHz
- 2. Trovare la stampante / le stampanti con il prezzo più alto
- 3. Trovare i PC più lenti del laptop più veloce
- 4. Elencare il modello / i modelli di prodotto con il prezzo più basso
- 5. Trovare il produttore della stampante a colori con il prezzo più alto
- 6. Trovare produttore/i dei PC più veloci tra quelli con la quantità massima di RAM
- 7. Trovare i produttori di almeno due computer (PC o laptop) veloci almeno 2 GHz
- 8. Trovare il prezzo medio di PC e laptop venduti dal produttore D
- 9. Trovare la dim. media degli HD di PC per ogni produttore di laptop
- 10. Trovare, per ogni prezzo, la velocità media dei PC associati (senza group by)
- 11. Trovare la velocità max dei PC venduti da ciascun produttore (senza group by)
- 12. Trovare il prezzo medio per ciascun produttore e tipologia di prodotto

Esercizio 3 – Prodotti

Sia dato lo schema:

product (model, maker, type)
pc (model, speed, ram, hd, price)
laptop (model, speed, ram, hd, screen, price)
printer (model, color, type, price)

Esprimere in SQL le seguenti query usando funzioni aggregate, GROUP BY e/o HAVING:

- 1.Trovare la dimensione minima, media e massima degli HD dei PC
- 2.Trovare il prezzo medio dei laptop veloci almeno 3.0 GHz
- 3.Trovare il prezzo medio dei PC venduti dal produttore A
- 4.Trovare, per ogni prezzo, la velocità media dei PC associati
- 5.Elencare i produttori che vendono almeno tre diversi modelli di PC
- 6.Trovare la velocità massima dei PC venduti da ciascun produttore
- 7.Trovare, per ogni velocità di PC sopra 2.5 GHz, la dim. media di HD
- 8.Trovare, per ogni produttore, la velocità media dei suoi laptop
- 9.Trovare le velocità di CPU in comune a due o più PC
- 10.Stilare una classifica dei produttori per numero di prodotti diversi venduti
- 11.Elencare per ogni produttore quanti tipi diversi di prodotto (pc/laptop/...) vende
- 12.Trovare il numero di modelli per ciascun produttore e tipologia di prodotto

ESERCIZIO 1 - NAVI DA BATTAGLIA

1. SELECT class, country
FROM ships.classes
WHERE num_guns > 12
2. SELECT name AS shipName
FROM ships.ships
WHERE launched < 1915
3. SELECT name
FROM ships.ships
WHERE name LIKE '%y.%'
4. SELECT ship, battle
FROM ships.outcome
WHERE result = 'sunk'
5. SELECT name
FROM ships.ships
WHERE name = class
6. SELECT name
FROM ships.ships
WHERE name LIKE 'y. y. y.'
7. SELECT s.name, c.num_guns,
c.displacement
FROM ships.outcomes o
JOIN ships.ships s ON
o.ship = s.name
JOIN ships.classes c ON
c.class = s.class
WHERE o.battle = 'Guadalcanal'
8. SELECT s.name
FROM ships.ships s
NATURAL JOIN ships.classes c
WHERE c.displacement > 3700
9. (SELECT s.name
FROM ships.ships)
UNION
(SELECT ships AS name
FROM ships.outcore)
10. SELECT & DISTINCT o1.battle
FROM ships.ships s1,
ships.classes c1,
ships.outcore o1,
ships.ships s2,
ships.classes c2,
ships.outcore o2
WHERE s1.class = c1.class AND
s1.name = o1.ship AND
s2.class = c2.class AND
s2.name = o2.ship AND
s2.name <> s1.name AND
o1.battle = o2.battle.
11. SELECT DISTINCT c1.country
FROM ships.classes c1,
ships.classes c2
WHERE c1.country = c2.country
AND c1.type = 'bb'
AND c2.type = 'bc'

Esercizio 2 - Prodotti

1. SELECT DISTINCT maker

FROM prod.product

WHERE model IN (SELECT model

FROM prod.laptop

WHERE speed > 2.0)

SELECT DISTINCT maker

FROM prod.product p, prod.product p2

WHERE p.model = p2.model AND p2.speed > 2.0

2. SELECT model

FROM prod.printer

WHERE price = (SELECT MAX(price)

FROM prod.printer)

(SELECT model FROM prod.printer)

EXCEPT

(SELECT p1.model

FROM prod.printer p1, prod.printer p2

WHERE p1.price < p2.price)

3. SELECT DISTINCT model

FROM prod.pc

WHERE speed < (SELECT MAX(speed)

FROM prod.laptop)

4. SELECT model

```
FROM( (SELECT model, price FROM prod.pc)
UNION
(SELECT model, price FROM prod.laptop)
UNION
(SELECT model, price FROM prod.printer)
) AS ModelPrice
```

WHERE price < ALL (SELECT price FROM prod.pc)
AND price < ALL (SELECT price FROM prod.laptop)
AND price < ALL (SELECT price FROM prod.printer)

5. SELECT maker

```
FROM prod.product p JOIN prod.printer s
ON p.model = s.model
```

WHERE color = TRUE AND price = (SELECT MAX(price)
FROM prod.printer
WHERE color = TRUE)

> ALL (SELECT price
FROM prod.printer
WHERE color = TRUE)

6. SELECT maker

```
FROM prod.product p
NATURAL JOIN prod.pc p1
LEFT OUTER JOIN prod.pc p2
ON p1.ram < p2.ram OR (p1.ram = p2.ram AND
p1.speed < p2.speed)
```

WHERE p2.model IS NULL

7. SELECT maker
FROM prod.product NATURAL JOIN
(SELECT model FROM prod.pc
WHERE speed > 2.0)
UNION
(SELECT model
FROM prod.laptop
WHERE speed > 2.0)
GROUP BY maker
HAVING COUNT(*) > 2

8. SELECT AVG(price) AS avg-price
FROM prod.product NATURAL JOIN
(SELECT model, price FROM prod.pc)
UNION
(SELECT model, price FROM prod.laptop)
WHERE maker = 'D'

9. SELECT maker, AVG(hd) AS avg-hd
FROM prod.product NATURAL JOIN prod.pc
WHERE maker IN (SELECT maker
FROM prod.product
WHERE type = 'laptop')
GROUP BY maker

10. SELECT DISTINCT p1.price,
SELECT AVG(speed) AS avg-speed
FROM prod.pc p2
WHERE p2.price = p1.Price) AS avg-speed
FROM prod.pc p1

ESERCIZIO 3 - PRODOTTI

1. SELECT MIN(hd) AS min-hd, AVG(hd) AS avg-hd, MAX(hd) AS max-hd
FROM prod.pc
2. SELECT AVG(price) AS avg-price
FROM prod.laptop
WHERE speed > 3.0
3. SELECT AVG(price) AS avg-price
FROM prod.pc NATURAL JOIN prod.product
WHERE maker = 'A'
4. SELECT price, AVG(speed) AS avg-speed
FROM prod.pc
GROUP BY price
5. SELECT maker
FROM prod.product
WHERE type = 'pc'
GROUP BY maker
HAVING COUNT(*) >= 3
6. SELECT maker, MAX(speed)
AS max-speed
FROM prod.product
NATURAL JOIN prod.pc
GROUP BY maker
7. SELECT speed, AVG(hd) AS avg_hd
FROM prod.pc
WHERE speed > 2.5
GROUP BY speed.

8. SELECT maker, AVG(speed) AS avg_speed
FROM Prod.product NATURAL JOIN Prod.laptop
GROUP BY maker
9. SELECT speed
FROM Prod.pc
GROUP BY speed
HAVING COUNT(*) > 2
10. SELECT maker, COUNT(*)
FROM Prod.product
GROUP BY maker
ORDER BY COUNT(*) DESC,
maker ASC
11. SELECT maker, COUNT(DISTINCT type)
FROM Prod.product
GROUP BY maker
12. SELECT maker, type, COUNT(*) AS num_models
FROM Prod.product
GROUP BY maker, type

DEDUCTIVE DATABASE

In SQL-92 standard non si possono scrivere ed eseguire query come:

"Qual è il totale dei componenti e il costo per produrre la macchina ZX600?"

DATALOG

Una query si può leggere in due modi:

- (1) Se una tupla esiste nella tabella indicata nel From che soddisfa la clausola nel Where, Allora la tupla scelta appartiene alla risposta.
- (2) La tupla sta nella risposta se esiste nella tabella indicata dal From che soddisfa la clausola Where.

$(\forall t_0, t_1, t_2) \exists t_3 \text{ work} -: (t_0, t_1, t_2, t_3) g_{123}$
 $(\forall t_0, s_1, t_2) \exists t_3 \text{ work} -: (t_0, s_1, t_2, t_3) g_{123}$
 $(t_0, s_1, t_2) g_{123}, t_3$

stato good

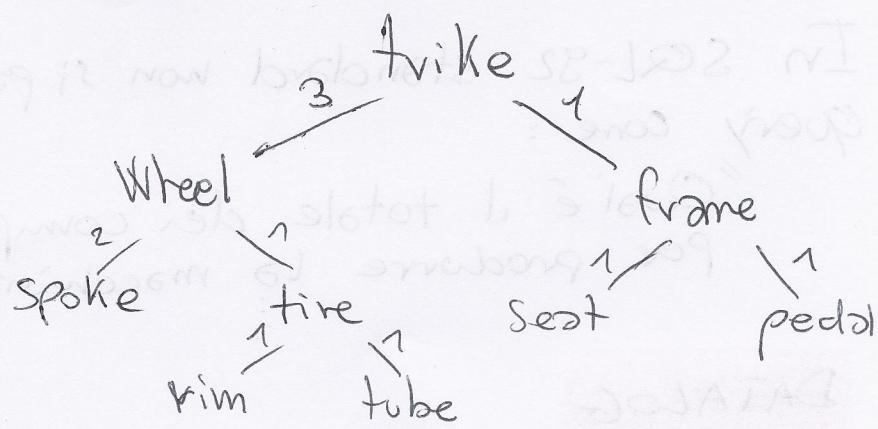
stato bad
avvertimenti

stato good

ESEMPIO

Assembly Instance

Part	SubPart	Qty
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1



→ Trovare tutte le componenti di un triciclo (trike).

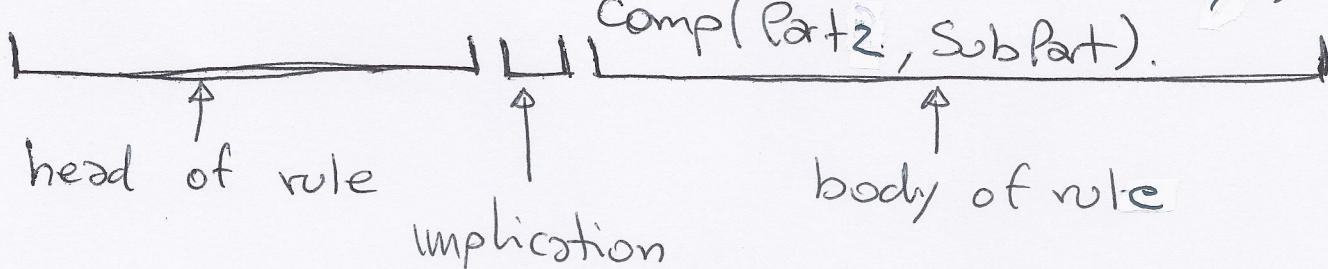
Per fare questo intuitivamente si può Joinare AssemblyInstance con se stessa per dedurre che il triciclo è formato da spoke, tire, seat e pedal.

Per trovare a loro volta i loro componenti, dobbiamo continuare a fare Join con se stessa per scendere ancora di un livello nell'albero.

Questo processo termina quando si raggiungono le foglie.

Che Datalog query fa esattamente questo:

$\text{Comp}(\text{Part}, \text{SubPart}) :- \text{Assembly}(\text{Part}, \text{SubPart}, \text{Qty}).$
 $\text{Comp}(\text{Part}, \text{SubPart}) :- \text{Assembly}(\text{Part}, \text{Part2}, \text{Qty}),$
 $\quad \text{Comp}(\text{Part2}, \text{SubPart}).$



La seconda si può leggere come segue

"Per tutti i valori di Part, SubPart e Qty,
Se esiste una tupla (Part, Part2, Qty) in Assembly
E una tupla (Part2, SubPart) in Comp,
Allora ci deve essere una tupla (Part, SubPart) in Comp"

IN SQL

WITH RECURSIVE Comp(Part, subPart) AS

(SELECT A1.part , A2.subpart FROM Assembly A1)

UNION

(SELECT A2.Part, C1.SubPart

FROM Assembly A2, Comp C1

WHERE A2.subpart = C1.part)

SELECT * FROM Comp C1

SCHEMA REFINEMENT AND NORMAL FORM

Dipendenza funzionale (FD): dipendenza tra gli attributi.

$X \rightarrow Y$, X un insieme di attributi
 Y un insieme di attributi

O più formalmente:

$t_1 \in r, t_2 \in r, \Pi_X(t_1) = \Pi_X(t_2)$ implies
 $\Pi_Y(t_1) = \Pi_Y(t_2)$.

Notazione: $ABCD$ (insieme di attributi) si decomponga in
 $AB \Leftarrow BCD$, $ACD \Leftarrow ABD$.

Le FDs sono correlate con lo schema del DB, non con i dati inseriti all'interno.

Le Primary Key sono un caso speciale di FDs, perché se conosco la primary key, so ricavare tutti gli altri attributi della tabella.

ESEMPIO

Hourly-Emps (ssn, name, lot, rating, hrly-wages, hrs-worked)

- set of attributes $\{S, N, L, R, W, H\}$

- ssn is the key: $S \rightarrow SNLRWH$

- il rating determina hrly-wages: $R \rightarrow W$

Nell'esempio precedente c'è un attributo che non dipende direttamente dalla chiave primaria: $R \rightarrow W$.
Quindi si possono creare le seguenti problematiche

- (1) Update anomalies: se si aggiorna un attributo (R nell'esempio) si devono aggiornare anche tutte le altre tuple con quel valore.

T_1	A	R	W
	A	8	10
	B	8	10
	C	5	5

↓

- Se voglio aggiornare la tupla $(B, 8, 10)$ in $(B, 8, 15)$ devo aggiornare anche $(A, 8, 10)$.

T_1	A	R
	A	8
	B	8
	C	5

T_2	R	W
	8	10
	5	5

- spezzandole invece posso aggiornare una tupla di T_2 senza aggiornare tutte quelle collegate

- (2) Insertion Anomaly: stesso problema dell'update

- (3) Deletion Anomaly: si perdono informazioni.

INFER FDs

Date una serie di FDs, si possono inferire altre FDs con gli assiomi di Armstrong.

- Reflexivity: controlla se una FD è trivial oppure no.

$ABC \rightarrow BC$ è trivial perché la parte destra delle FDs è un sottinsieme della parte sinistra.

- Augmentation: se $X \rightarrow Y$ allora $XZ \rightarrow YZ, XZ$
ma non è vero che $AB \rightarrow CB \Rightarrow A \rightarrow B$ (No!)

- Transitivity: se $X \rightarrow Y$ e $Y \rightarrow Z \Rightarrow X \rightarrow Z$

ESEMPI

$$JP \rightarrow C$$

$$C \rightarrow CDJDpq \Rightarrow JP \rightarrow CDJDpq$$

$$SD \rightarrow P \Rightarrow SDJ \rightarrow JP$$

Dato uno schema relazionale e delle FDs, si
possono calcolare altre FDs per chiusura per scoprire
se tutti gli attributi possono dipendere da un insieme
detto chiave.

ESEMPIO

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

$A \rightarrow B$

$B \rightarrow C$

$CD \rightarrow E$

Closure $B^+ = \{B, C\}$?

Closure $BD^+ = \{B, D, C, E\}$

Closure $A^+ = \{A, B, C\}$

Closure $AD^+ = \{A, D, B, C, E\}$

Nello schema R sono indicate alcune FDs ma che
nessuna fa emergere la chiave dello schema;
quindi calcolando la chiusura di AD si scopre
che è la chiave primaria, in quanto con AD posso
identificare univocamente tutti gli altri
attributi.

BCNF

Le forme normali hanno il compito di controllare la consistenza tra gli attributi di una tabella, evitando i problemi (1), (2), e (3) delle pagine precedenti.

BCNF \Rightarrow buon design del DB

Dato uno schema relazionale e delle FDs, si può dire se è un buon design o meno.

- $R(x, y, A)$
 $x \rightarrow A$

non è un BCNF perché non riesco a trovare un attributo che mi identifichi univocamente tutti gli altri.

- $R(cf, name, dob, car)$
 $cf \rightarrow name, dob$

Non è in BCNF, perché conoscendo il codice fiscale (cf) di una persona riesco a inferire il nome e data di nascita, ma non la sua macchina.

\Rightarrow Uno schema relazionale è in BCNF se riesco a inferire almeno una chiave per quella relazione.

NB: Se non sono presenti alcuna FDs, allora tutti gli attributi sono chiave, perché non ci sono vincoli sugli attributi da rispettare.

ESEMPIO

Calcolare tutte le chiavi del seguente schema:

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

$A \rightarrow B$

$BC \rightarrow E$

$ED \rightarrow A$

Con le FDs date non si riesce a trovare una chiave, quindi si prova con tutte le possibili combinazioni di attributi ~~per calcolare la chiusura~~.

Si inizia con un attributo singolo fino alla dimostrazione dello schema -1.

$$A^+ = B$$

$$B^+ = B$$

$$C^+ = C$$

$$D^+ = D$$

$$E^+ = E$$

$$AB^+ = AB$$

$$AC^+ = ACBE$$

$$AD^+ = ADB$$

$$AE^+ = AEB$$

$$BC^+ = BCE$$

$$BD^+ = BD$$

$$BE^+ = BE$$

$$CD^+ = CD$$

$$DE^+ = DEAB$$

$$ABE^+ = ABCE$$

$$BCD^+ = BCDEA$$

$$CDE^+ = CDEAB$$

$$DEA^+ = DEAB$$

$$EAB^+ = EAB$$

$$CDA^+ = CDABE$$

Anche se abbiamo trovato delle primary key con 3 attributi, dobbiamo controllare anche quelli con 4. Ma sapendo che saranno chiavi tutte le chiusure con 4 attributi che iniziano/contengono una chiave primaria scoperta dalle chiusure con 3 attributi.

TRICK: un attributo è sicuramente chiave se non compare nella parte destra di ogni FDs; ciò significa che quel attributo non può essere inferito da nessuna FDs, quindi deve far parte della chiave.

Nell'esempio precedente:

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

$A \rightarrow B$

$BC \rightarrow E$

$ED \rightarrow A$

Come si può notare CeD non compaiono nella parte destra di nessuna FDs quindi inizio a calcolare la chiusura partendo da quelle:

$$CDA^+ = CADABC$$

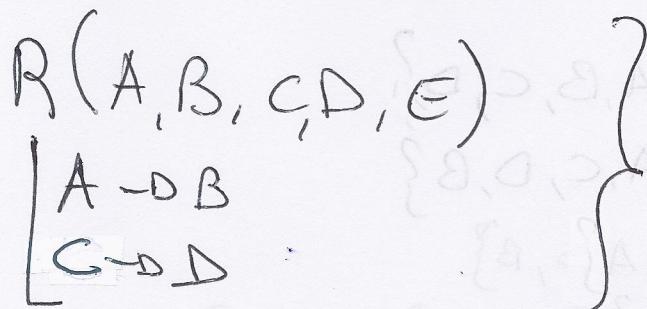
$$CDB^+ = CDBGA$$

$$CDE^+ = CDEAB$$

ho trovato le chiavi con 3 attributi senza necessità di calcolarmi quelle precedenti.

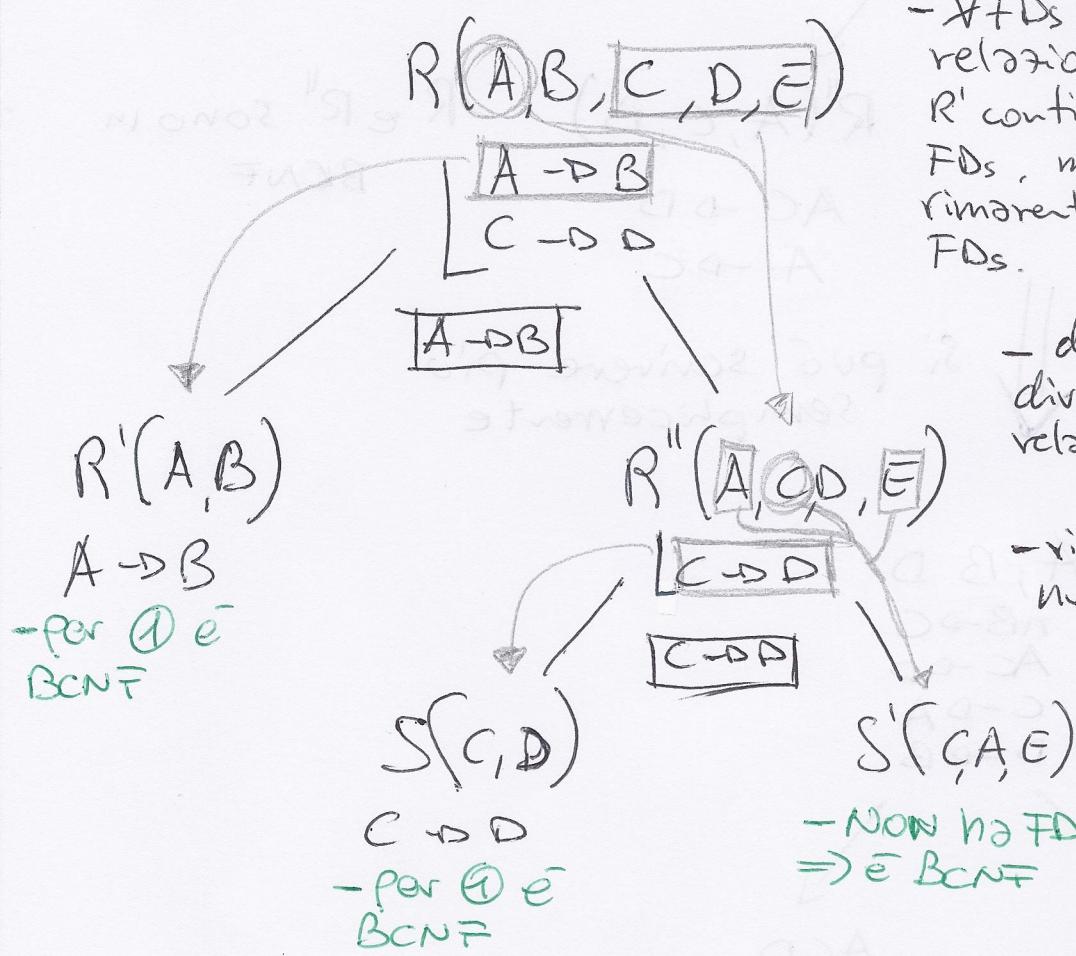
ESERCIZIO

Dato il seguente schema e le sue FDs dire se è in BCNF



- ① HFDs controllare (per chiusura) se la parte sinistra è una chiave
 $A^+ = \{A, B\}$ $C^+ = \{C, D\}$
 \Rightarrow non sono chiavi
 \Rightarrow NO BCNF.

② Se non è in BCNF si decomponere la relazione in base alle FDs.



- HFDs non chiave si spezza la relazione in due R' e R'' dove R' contiene tutti gli attributi della FDs, mentre R'' contiene i rimanenti più la parte sx delle FDs.

- dopo aver spezzato si dividono le FDs sulle nuove relazioni così create.

- ripetere ① e ② sulle nuove relazioni

Dato lo schema relazionale e le sue FDs dire se è in BCNF.

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

$$\begin{array}{l} AB \rightarrow DC \rightarrow AB^+ = \{A, B, C, D\} \\ AC \rightarrow D \rightarrow AC^+ = \{A, C, D, B\} \\ C \rightarrow A \rightarrow C^+ = \{C, A, D, B\} \\ D \rightarrow B \rightarrow D^+ = \{B\} \end{array}$$

→ D non è una chiave.

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

$R'(D, B)$
 $D \rightarrow DB$

$R''(A, C, D)$

$AC \rightarrow D$
 $C \rightarrow DA$

$R' \text{ e } R''$ sono in
BCNF

↓
Si può scrivere più
semplicemente

ABCD

$AB \rightarrow DC$

$AC \rightarrow DD$

$C \rightarrow DA$

$D \rightarrow B$

DB
 $D \rightarrow DB$

ACD
 $AC \rightarrow DD$
 $C \rightarrow DA$

	Scan	Equality ($A=5$)	Range ($A>7$)	Insert 1 tuple (that has $A = 9$)	Delete tuples $A=6$
Heap File	B	B [c1]	B	1+1	$(1+1)*B$ [c2]
Sorted File	B	$\log B + 0.5B$ [c3]	$\log B + 0.5B$	$\log B + 0.5B*(1+1)$	$\log B + (N_c / R)$ [c4]
Clustered B+ tree	B	$L_t + N_c/R$	$L_t + N_c/R$	$L_t + N_c/R + 1$	$L_t + N_c/R * (1+1)$
Unclustered B+tree	B	$L_t + N_c$	$L_t + N_c$	$L_t + 1+1$	$L_t + N_c * (1+1)$
Hash (Unclustered)	B	$L_h + N_c$	---	$L_h + 1+1$	$L_h + N_c * (1+1)$

Assume:
 • Relation S with an attribute A

- Cost of reading or writing a page is 1
- Every page on the disk if for 1 relation only (cannot have tuples from more than 1 relation)
 - It is ok if a page does not have the max number of tuples it can fit.
 - Every time we divide 2 integers we round up (e.g., 3.7 becomes 4)
- P: Size of one page
- T: Size of one tuple
- B: # pages of S
- R: # tuples that fit in a page
- N: # tuples of S
- N_c : # tuples of S satisfying condition c
- L_h : Lookup Cost of Hash Index on A
- L_t : Lookup Cost of a B+tree index on A
- $|A|$: Cardinality of A, i.e., number of distinct values of A
- Assume that the cost of updating Hash or a B+tree is 0

It applies that:

- $N=R*B$
- $N_{A=6} = N / |A|$
- $N_{A>7} = \text{depends on the case. Need to find it...}$
 - $L_h = 1.2$ (typically. Has to be told to you)
 - $L_t = \log |A|$ (assuming dense index) or we can assume 4 as average but it has to be said to consider it.
 - N_c is computed either assuming uniform distribution or using a given selectivity factor.
 - There is NO clustered Hash index
- [c1] if A is a key, the avg cost would be $0.5B$
- [c2] We assume that every page has at least one tuple of $A=6$. If A was a key, the cost would have been: $0.5B + 1$
- [c3] If A was a key, the cost would be $\log B$
- [c4] We assumed that we have to read until we find the A=7 to link it to the A=5.

How long does it take to sort a relation with N pages on the disk ?

Using: Two way merge sort: $2^*N*(\log_2 N+1)$ – Requires 3 buffers

Using: External Merge Sort: $2^*N*(1+\log_{b-1} \text{RoundUp}(N/B))$ – Requires B buffers

What is the cost of R JOIN S (if R has M pages and S has N pages) ?

Simple Nested Loops Join: $M+N*M$

Sort Merge Join: Sort R + Sort S + M + N

Hash Join: $3(M+N)$ if partition Fits in Memory

Block Nested Loops: $M+N*\text{Roundup}(M/B-2)$

Index Nested Loops Join: $M + \# \text{ tuples in } M * \text{ Cost of retrieving matching tuples}$

B: requires Buffers

Cost Retr. Matching Tuples: Do a lookup to Find Leaf ($\log N$ in B+ trees, 1.2 in Hash) + Read Tuples of the pointer (or pointers)

Read Tuples: # tuple reads if unclustered, # tuples/#tuples per page in cluster

Index Only Operator: Is an operator (select, join, etc. that does not require you to access the actual tuple but only from the index you can find what you want. For example, select S.A from S, T where S.B=T.C with an index on T.C, is index-only because you only need a lookup on the Index and not to actually read the tuples.

Query Plans: A query plan is a tree of relational operators that achieve a query. The optimizer is the one that decides the final order of the operators.

Very often it is highly based on the way that the user expressed the query . An important task is to identify for every node of the tree (meaning for every operator) the cost and also the tuples that it creates. The cost depends on the tuples that they arrive from the operators under it, and the method chosen to implement it.

Note that for intermediate operators, if the data is too large they have to be stored on the disk. Data stored on the disk have no indexes.

ESERCIZI COSTO OPERAZIONALE

Exec(ename, title, dname, address)

- 10K pages : la tabella Exec occupa 10K su disco
- 20byte : ogni tupla occupa 20 byte
- 1K page : ogni pagina occupa 1K di spazio
- red factor 10% : percentuale di trovare un attributo uguale ad un valore.

Volutare la seguente query:

```
SELECT E.title, E.ename
FROM Exec E
WHERE title = 'CEO'
```

nel caso:

- ① Clustered B+tree su title
- ② Unclustered B+tree su title
- ③ Clustered B+tree su ename

1) Essendo che l'indice B+tree è su un attributo che compare nella clausola WHERE, allora mi aiuta nella ricerca delle tuple.

$$\text{COSTO TOT} = L_B + \frac{N_c}{R}$$

$$L_B: \text{fixed} = 1.2$$

$$N_c = \# \text{tuple che soddisfano clausola WHERE} = \# \text{tuple della tabella Exec} \cdot 0.1 =$$

$$= \# \text{tuple di Exec che trovano in una pagina} \cdot 10k \cdot 0.1 = \frac{1k}{20 \text{ bytes}} \cdot 10k \cdot 0.1$$

$$R = \# \text{tuple che trovano in una pagina} = \frac{1k}{20 \text{ bytes}}$$

2) Stesso ragionamento come ①

$$\text{COSTO TOT} = L_B + N_c$$

$$N_c = \# \text{tuple che soddisfano clausola WHERE} = \left[\left(\frac{1k}{20 \text{ bytes}} \right) \cdot 10k \right] \cdot 0.1$$

3) In questo caso l'indice è sull'attributo ename, il che non mi aiuta a trovare il title='CFD'; quindi dovrò scorrere tutte le tuple.

$$\text{COSTO TOT} = \text{Scan} = B = 10K$$

ESERCIZI COSTO OPERAZIONALE

Sia dato il seguente schema:

Emp(eid:int, sal:int, age:real, did:int)

Dep (did:int, projid:int, budget:real)

Project(projid:int, code:int, report:varchar(20))

Quanto occupa una tupla di ogni tabella su disco?

Emp: int + int + real + int = 4·3 + 8 byte

Dep: int + int + real = 4·2 + 8 byte

Project: int + int + varchar(20) = 4·2 + 20 byte

Ottimizzare le seguenti query:

① SELECT eid FROM Emp

② SELECT DISTINCT eid FROM Emp

③ INSERT INTO Dep(x,y,z) ...

④ INSERT INTO Emp(5,...,A) ...

⑤ DELETE FROM Dep WHERE budget < 5

- ① Selezionare tutte le chiavi dalla tabella Emp.
- In questo caso non avrò bisogno di nessun indice, perché non sto cercando una tupla in particolare, ma dovrò comunque leggerle tutte.
- COSTO TOT: Scan = # pagine di Emp.
- ② In questo caso non è una semplice Scan, ma la clausola DISTINCT, per eliminare i duplicati, prima ordina tutte le tuple, poi elimina i duplicati dall'output; quindi il costo della query è dato dall'ordinamento e dal costo di lettura di tutte le tuple
- COSTO TOT: $N \log N + N$
- ③ Prima si deve controllare che la tupla da inserire non sia già presente, quindi creando un hash index sull'attributo chiave si ottimizzano tutte le equality search. Quindi il costo è dato dal trovare la pagina, leggere tutte le tuple e riscrivere tutte le tuple:
- COSTO TOT: $L_H + 1_{read} + 1_{write}$
- ④ Si deve controllare prima se "s" non esiste in Emp e "A" sia presente in Dept. Quindi per ottimizzare gli equality search, la cosa migliore è un hash index su Dept.did e Emp.eid.
- ⑤ Essendo che si tratta di una delete si deve tener conto delle tuple associate (Foreign Key). Quindi per eliminare le tuple con budget < 5 ~~sare~~ un B⁺tree su budget e un hash index su Emp.did, perché per ogni tupla eliminata in Dep si devono eliminare tutte le tuple in Emp con lo stesso did

1. (4 punti) Given the following CREATE statements for tables :

```
CREATE TABLE Employee (
    cf CHAR(14) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hired_on TIMESTAMP,
    email VARCHAR(150) UNIQUE )
```

```
CREATE TABLE Project (
    number INT NOT NULL,
    resp CHAR(14),
    budget REAL,
    delivery DATE NOT NULL,
    PRIMARY KEY (number, resp),
    FOREIGN KEY (resp) REFERENCES Manager )
```

```
CREATE TABLE Manager (
    cf CHAR(14) PRIMARY KEY,
    promoted DATE NOT NULL,
    wage REAL,
    FOREIGN KEY (cf) REFERENCES Employee )
```

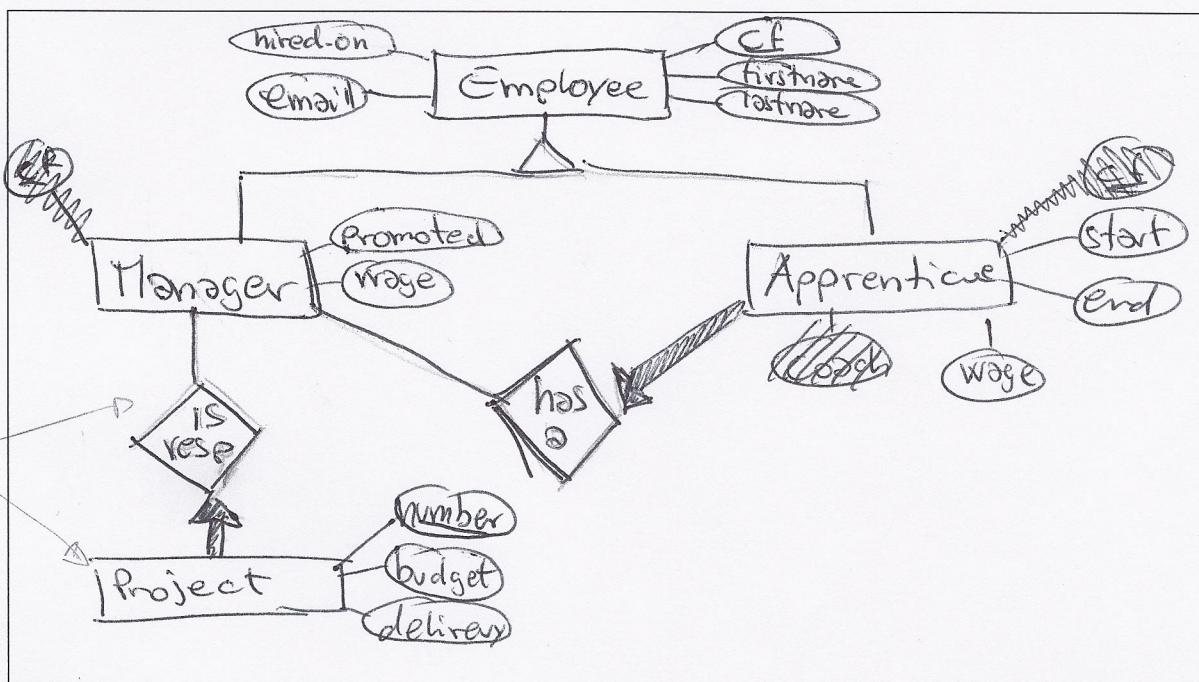
```
CREATE TABLE Apprentice (
    cf CHAR(14) PRIMARY KEY,
    starts DATE NOT NULL,
    ends DATE NOT NULL,
    wage REAL,
    coach CHAR(14),
    FOREIGN KEY (cf) REFERENCES Employee,
    FOREIGN KEY (coach) REFERENCES Manager )
```

(a) Design the ER schema that produces the above definitions

(b) Can a Manager be also registered as an Apprentice ? YES NO UNDECIDABLE

Non-standard notation (i.e, different from what present in the book and/or in the slides) will be considered a mistake.

DRAW THE ANSWER BELOW:



IsResp e Project sono weak perché una parte della chiave di Project è presente in Manager (vedi*)

2. (3 punti) Given the following tables, and with only the tuples present below

S(A, B, C)

A	B	C
1	a	a
2	a	a
3	a	c
4	a	e

T(D, E, F)

D	E	F
a	e	1
a	c	2
a	a	NULL
b	a	3

Nelle domande chicole
Se può essere qualcosa
con lo schema e i dati
a disposizione.

Taking into consideration only the tuples showed above, without any further assumption, answer the following
[+1 point for correct answers, -1 points for wrong answers, 0 points if answer left blank]

(a) Can S(B,C) be foreign key with reference to T(D,E)? YES NO UNDECIDIBLE

(b) Can (A,B) be a primary key for S? YES NO UNDECIDIBLE

(c) Can (D,E,F) be a primary key for T? YES NO UNDECIDABLE

3. (4 punti) Consider the following schema instances (structure and tuples):

S(A, B)

A	B
1	10
2	20
3	30

T(B, C, D)

B	C	D
1	10	NULL
2	20	f
3	NULL	w

U(D, E)

D	E
NULL	f
o	w
f	w
m	NULL

Show the results of the following queries:

WRITE YOUR ANSWER BELOW:

(a) SELECT T.D, SUM(T.B)+1
FROM S JOIN T ON S.B >= T.C+10
GROUP BY T.D

(b) SELECT T.D, U.D, U.E
FROM T JOIN U ON T.D=U.E

T.D SUM(T.B)+1		
F	3	
N	3	
X		

T.D	U.D	U.E
NULL	NULL	f
f	o	w
w	f	w
f	n	t
w	o	w
w	f	w

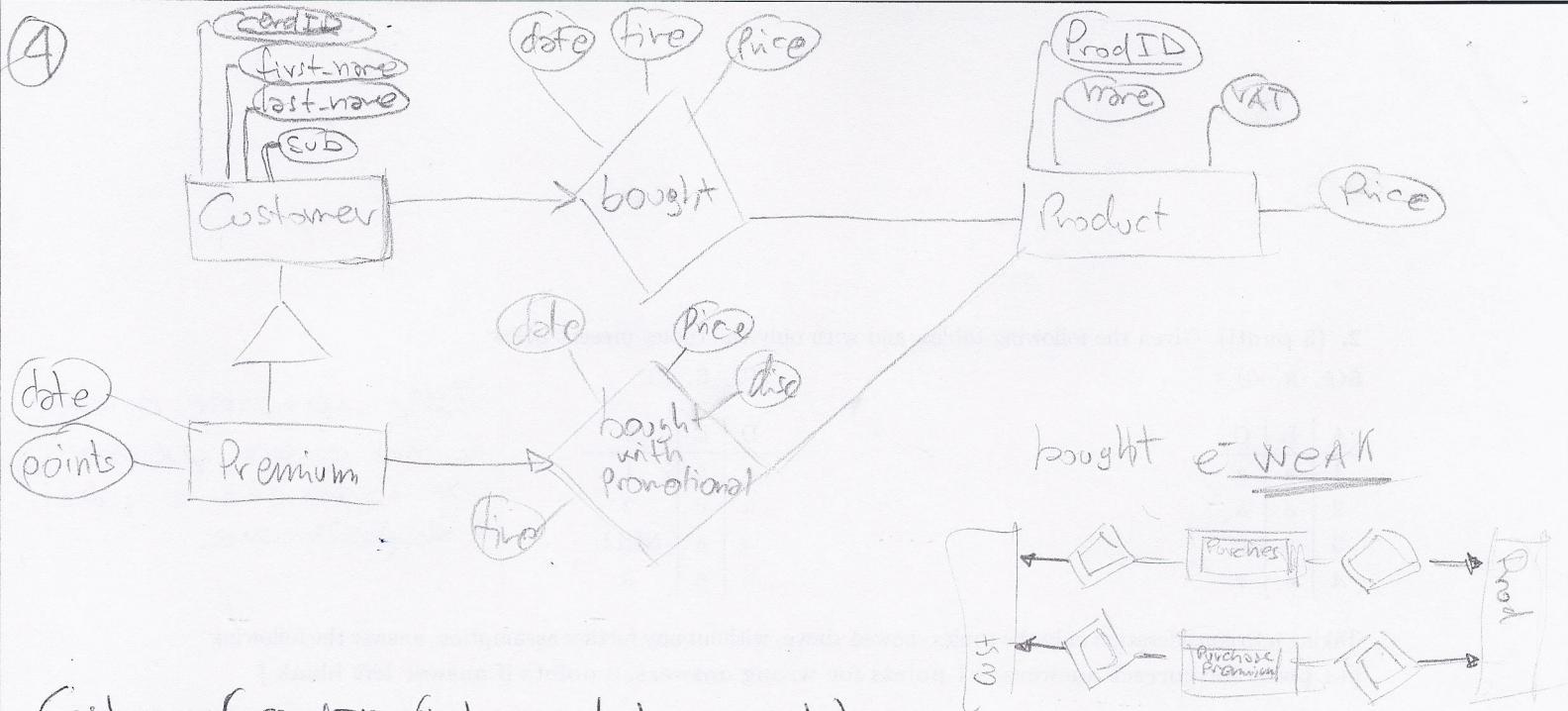
NULL CON GROUP BY:

fanno un gruppo a parte

NULL NELLE JOIN:

- 3 - NULL = NULL => FALSE
- NULL ≠ NULL => FALSE
- NULL = 1 => FALSE

NULL comparato con qualcosa
dava come risultato => FALSE.



Customer (cardID, first-name, last-name, sub)

Bought (cardID, ProdID, date, time, price)

ProdID Foreign Keys to Product. ProdID

CardID Foreign Keys to Customer. CardID

Product (ProdID, name, VAT, price)

Premium (cardID, date, points)

CardID Foreign Key to Customer. cardID

BoughtWithPromotion (cardID, date, time, price, discount, ProdID)

CardID Foreign Key to ~~Product. cardID~~ Customers. cardID

ProdID " " " ~~Customers. ProdID~~ Product. ProdID

5a) - Join Review with Review with different user

$\rho(R_1, \text{Review} \setminus_{\text{User} \neq \text{User}} \text{Review})$

- Join Review with R₁

$\rho(R_2, R_1 \setminus_{\text{User} \neq \text{User}} \text{Review})$

- Subtract to all Review

$\rho(R_3, \text{Review} - R_2)$

$\Pi \text{code}, \text{supplier}, \text{brand} (R_3)$

5b) - Find Supplier without discontinued products

$\rho(S_1, \text{G}_{\text{discontinued}(\text{Supplier})} = \text{false})$

- Subtract to all Supplier

$(\text{Supplier}) - S_1$

4. (5 punti) Assume you need to design a database for monitoring purchases of customers subscribers of a fidelity card for a grocery chain. The system manages the customers, with their fidelity card account, products that the chain sells, and all the customer's purchases.

Customers are identified by a unique cardID number, then you need to store the first and last name, and the subscription date for the card. The system separates customers with a Premium subscription. For each of them, the system stores the date when the Premium subscription was first registered, and a number of Premium points that the customer acquires in time.

Products are characterized by their name, the VAT number of the producer, the price, and they are identified by their serial code "ProdID".

For each customer, the system keeps track of the list of products bought; each purchase is stored separately. For each purchase, the system also keeps track of date and time of the purchase, amount of products purchased, and price of the single item (*note*: the price may vary over time).

A purchase is identified by the customer, the product and the date and time of the purchase. Only customers with a Premium subscription can buy products with a promotional price. For purchases with a promotional price, the original price is still saved in the system, but additionally the system stores the discount percentage and the number of premium points acquired by the customer with such purchase.

Provide the following:

- the ER diagram for the database to respect all above conditions.
- the relational schema of the database (mark also foreign keys in the schema)

Non-standard notation (i.e, different from what present in the book and/or in the slides) will be considered a mistake.

5. (10 punti) A different database stores information about products and reviews, with the following schema:

~~Product (code, supplier, name_prod, brand, price, address, discontinued)~~
~~Review (code, supplier, user, date, name_prod, email, text, rating, brand, price)~~
~~H I L N O P Q R S~~

The product is identified by the code and the *id* of the supplier. Of the product the database stores the name, the brand, the price, the address of the supplier and a boolean flag to mark it as discontinued.

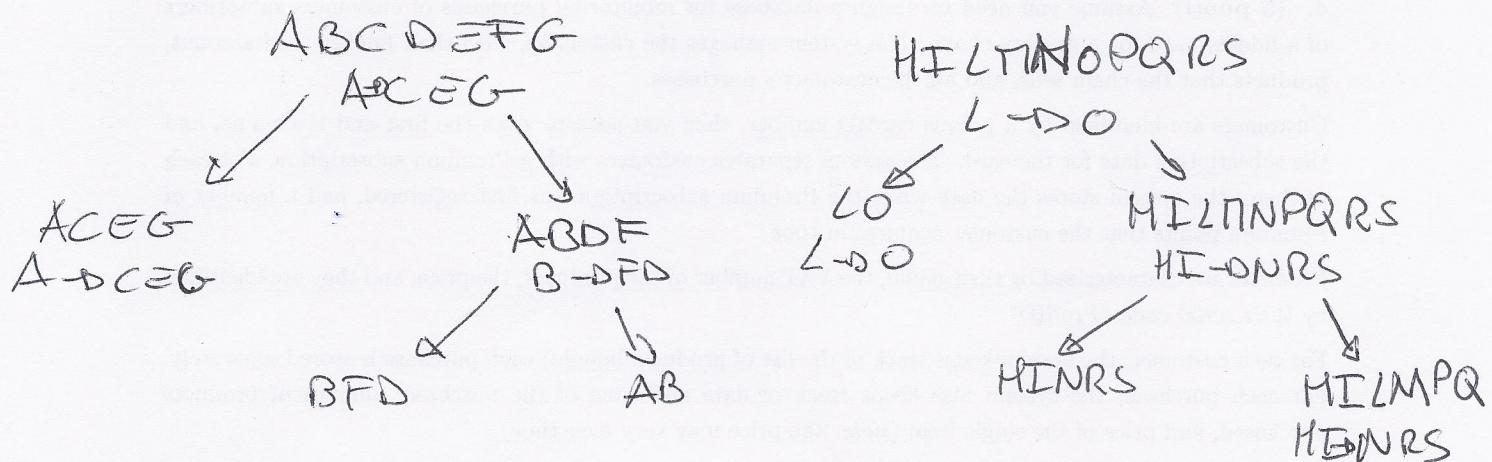
The review refers to a single product, contains the id of the user who wrote it, and her email address. It also stores the date in which it has been submitted, the name of the product, the text of the actual review, the rating given to the product, the brand of the product and the price.

Two distinct suppliers can use the same code for any two products even if they are different. A supplier sells only products of a single brand, and has a single address. A product has a single name and price. A user only owns a single email.

- Provide in **RELATIONAL ALGEBRA** the query to return code, supplier and brand of products with exactly 2 reviews.
- Provide in **RELATIONAL ALGEBRA** the query to return suppliers with all discontinued products.

6) Product(A,B,C,D,E,F,G)
 $A \rightarrow DC\bar{E}G$
 $B \rightarrow FD$

Review(H,I,L,M,N,O,P,Q,R,S)
 $C \rightarrow O$
 $H \rightarrow D \bar{E}NRS$



5c) - Find discontinued product with Review

$P(P_1, (\exists_{\text{discontinued} = \text{true}} \text{Product})) \bowtie_{\text{Code Supplier}} \text{Review}$

- Subtract to all product

(Product) - P₁

SELECT code, supplier FROM Product
 WHERE discontinued = true AND NOT IN
 (SELECT DISTINCT code, supplier FROM Review)

5d) SELECT supplier, AVG(NUM)
 FROM (SELECT supplier, code, COUNT(user, date) AS NUM
 FROM Review
 GROUP BY (supplier, code)) AS Number
 GROUP BY supplier.

5e) WITH Rating AS (
 SELECT supplier, AVG(rating) AS rate
 FROM Review
 GROUP BY supplier
)
 SELECT supplier, rate FROM Rating
 WHERE rate = (SELECT MAX(rate) FROM Rating)

- (c) Provide in **RELATIONAL ALGEBRA** and **SQL** the query to find discontinued products without any review.
- (d) Provide in **SQL** the query to find for all suppliers the average number of reviews for products.
- (e) Provide in **SQL** the query to find for suppliers with the highest average rating for their products.

6. (2 punti) Given the Relational Schema of Exercise 5:

Product (code, supplier, name_prod, brand, price, address, discontinued)

Review (code, supplier, user, date, name_prod, email, text, rating, brand, price)
with the same assumptions described above

- (a) Find all the functional dependencies
- (b) Compute the BCNF decomposition

7. (2 punti) Given the following relations $R(A, B, C, D, E)$ e $S(F, G, H)$ and the following set of functional dependencies: $AB \rightarrow C$, $CD \rightarrow E$, $A \rightarrow E$, $FG \rightarrow H$. State if any of the following satisfied the above dependencies and as such can be an instance of the schema.

[+1 point for correct answers, -1 points for wrong answers, 0 points if answer left blank]

I.

A	B	C	D	E
1	4	1	3	6
2	4	2	4	7
2	4	2	5	7
2	4	2	6	2

F	G	H
3	2	1
4	5	5

YES NO

II.

A	B	C	D	E
3	5	1		
2	5	1		
1	5	1		

YES NO

8. (3 punti) Answer the following questions

[+1 point for correct answers, -1 points for wrong answers, 0 points if answer left blank]

✓ (a) If R and S are relations with only common attribute a , then it holds that $\sigma_{R.a=3}(R \times S) \equiv \sigma_{a=3}(R) \times \sigma_{a=3}(S)$.

YES NO UNDECIDABLE.

✓ (b) If the attribute A for the relation R is Foreign Key referencing Y in the relation S , then it holds that A in R can contain NULL values.

YES NO UNDECIDABLE.

✗ (c) The \cap operator in relational Algebra can be equivalently expressed by using \bowtie and π operators

YES NO UNDECIDABLE.

\cap (intersezione) si può ottenere con
join (\bowtie) e selezionando (π)₅
solo le colonne in comune

join
Projection
of columns

Rispondi alle seguenti domande indicando la risposta corretta / Respond to the following questions indicating the correct answer.

Per eseguire la query $\sigma_{A>4}(R)$ è più conveniente usare una lettura completa della tabella al posto di un indice hash sull'attributo A di R, assumendo che R sia *clustered*. **Vero o Falso?**

To execute the query $\sigma_{A>4}(R)$ it is better to use a table scan on R than to use a hash index on the attribute A of R given the fact that R is clustered. **True or False?**

In una Tabella ci può essere solo un attributo clustered.

È possibile avere due indici diversi su attributi distinti della stessa tabella nel caso in cui essi siano entrambi *clustered*. **Vero o Falso?**

It is possible to have two indexes (on different attributes) on the same relation provided of course that they are both clustered. **True or False?**

Durante l'ottimizzazione delle query di tipo selezione-proiezione-join, la migliore strategia in assoluto è quella di anticipare le selezioni il prima possibile in modo da avere il minor numero di tuple possibile e quindi far sì che l'esecuzione della query sia la più veloce. **Vero o Falso?**

During query optimization of select-project-join queries the best strategy is to push the selections as early as possible so that we make sure that we have less tuples to deal with and the overall query will execute faster. **True or False?**

Falso perché la condizione R.B=S.B non deve essere rispettata.

Considera le relazioni R[A,B] e S[B,C], e assumi che ci sia almeno una tupla in R che ha un valore di R.B maggiore di 10. Di conseguenza, la query "SELECT * FROM R, S WHERE R.B=S.B AND R.B>10" ritorna almeno una tupla. **Vero o Falso?**

Consider relations R[A,B] and S[B,C], and assume that there is at least one tuple in R that has a B-value which is greater than 10. Then, the query "SELECT * FROM R, S WHERE R.B=S.B AND R.B>10" returns at least one tuple. **True or False?**

Cardinalità: come cardinalità insiemistica, numero di tuple.

La cardinalità del risultato della join di due tabelle dipende dall'ordine ($R \bowtie_C S$ oppure $S \bowtie_C R$) in cui il join viene eseguito. **Vero o Falso?**

The cardinality of the results of the join of two tables depends on the order the join is performed ($R \bowtie_C S$ or $S \bowtie_C R$). **True or False?**

Si possono creare due indici sullo stesso attributo a patto che siano unclustered.

Non si possono creare due indici sullo stesso attributo poiché il database non è in grado di disporre le tuple in modo tale da soddisfare le esigenze di entrambi gli indici. **Vero o Falso?**
You cannot create two indexes on the same attribute because the database does not know how to arrange the tuples to participate in both indexes. **True or False?**

Considera la query $\sigma_{A=6 \text{ AND } B=4}(R)$. Se il fattore di riduzione per $A=6$ è di 0.2 e per $B=4$ è di 0.5, si può considerare come fattore di riduzione della selezione 0.1, assumendo non vi sia nessuna dipendenza funzionale tra A e B, e vice versa. **Vero o Falso?**

Consider the relation $\sigma_{A=6 \text{ AND } B=4}(R)$. If the reduce factor for $A=6$ is 0.2 and for $B=4$ is 0.5, the reduce factor of the selection can be assumed to be 0.1 assuming that there is no functional dependency between A and B and vice versa. **True or False?**

Essendo che non c'è nessuna dipendenza funzionale fra A e B, si può considerare come eventi indipendenti; quindi l'evento che si verifichi $A=6$ e $B=6$ si può assumere con un fattore di riduzione di $0.2 \cdot 0.5 = 0.1$.

T

F

T

F

F

F

T

53

Non è un ottimizzazione perché nella seconda considera anche i NULL

Un'ottimizzazione intelligente della query `SELECT * FROM R WHERE A>5 OR A≤5`" è di riscriverla come la query equivalente "SELECT * from R" (assumendo naturalmente che A sia un attributo esistente in R). **Vero o Falso?**

A smart optimization of the query `SELECT * FROM R WHERE A>5 AND A≤5` is to replace it with the equivalent query `SELECT * from R` (assuming of course that A is an attribute of R) **True or False?**

Considera lo schema seguente / Consider the following schema

Employee [Eid, EName, Salary, WorksFor]

Dept [Did, DName, Budget, Mgr]

Emp.WorksFor è una chiave esportata riferita a Dept.Did / Emp.WorksFor is a foreign key referencing to Dept.Did

Dept.Mgr è una chiave esportata riferita a Emp.Eid / Dept.Mgr is a foreign key referencing to Emp.Eid

Assumi di avere le seguenti scelte alternative di indici:

Assume different alternatives on what indexes have been created on that schema.

- A. clustered B-tree su Emp[Eid] e clustered B-tree su Dept[Did]
- B. clustered B-tree su Emp[WorksFor] e clustered B-tree su Dept[Did]
- C. clustered hash su Emp[Eid] e clustered hash su Dept[Mgr]
- D. clustered B-tree su Emp[Salary] e clustered B-tree su Dept[Did]
- E. clustered B-tree su Emp[Salary] e clustered B-tree su Dept[Budget]
- F. (nessun indice) / No index at all.

NON PUÒ
ESISTERE
UNA
CLUSTERED
HASH INDEX!

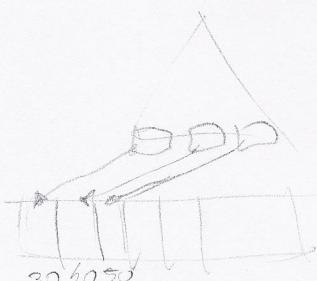
Per ognuna delle seguenti query o aggiornamento, scegli la migliore alternativa tra quelle riportate precedentemente. Se due o più scelte consentono le stesse performance per una query o aggiornamento, riporta tutte le opzioni. Puoi usare ogni singola scelta per più query o aggiornamenti. Assumi che nessun altro indice sia presente ad eccezione di quelli presenti all'interno della singola opzione.

(Le scelte si devono basare solo sulle informazioni qui riportate, nessun'altra informazione è data)

For each query/update, pick the best index design from those above. If two or more designs support the same performance for the given query/update, indicate all these designs. You may use a design multiple times (or not at all). Assume no indexes are present except the ones indicated in each design.

(You have no other information apart from the one presented here)

- Trova gli Eid degli impiegati (Employee) il cui salario è superiore del budget del dipartimento (Dept) con Did = 1 | Find the Eid's of employees whose salary is greater than the budget of the department with Did = 1
 - Risposta/Answer: B
- Inserisci la tupla di un impiegato (Employee) | Insert an Employee tuple
 - Risposta/Answer: A
- Rimuovi tutti gli impiegati (Employees) del dipartimento con Did = 1 | Delete all employees in the department with Did = 1
 - Risposta/Answer: B



Considera il seguente schema:

- Writer[Wid:Integer, Name:VarChar, Age:Integer]
- Paper[Pid:Integer, PubDate:Date, Abstract:VarChar]

1K bytes page
 $L_B = 5$
 $L_H = 1,2$

Assumi che ogni tupla scrittore (Writer) occupi 50 bytes e che ogni tupla articolo (Paper) occupi 200 bytes. Ci sono 4,000 tuple per Writer e 10,000 per Paper. Inoltre una pagina sul disco occupa 1000 bytes. Puoi assumere (se necessario) che il costo per leggere la foglia di un indice B+tree index richieda 4 letture da disco, e che un indice hash richieda (in media) 1,2 accessi al disco.

Assume that every tuple Writer takes 50 bytes, and that every tuple Paper takes 200bytes. There are 4,000 tuples in Writer and 10,000 in Paper. Moreover, a page on disk takes 1000bytes. You can assume that the cost to read a leaf node in the B+Tree takes 4 reads to disk, and that an hash index takes 1,2 (average) accesses to disk.

Considera le seguenti due query: (q1) "trova tutti i writers con age=30" e (q2) "trova tutti gli articoli pubblicati con PubDate=09/06/2002." Assumi che il numero di tuple qualificate (ovvero il numero di tuple che soddisfano una query) sia lo stesso per entrambe le query. Assumi di creare un indice B+tree su `age` e un indice B+tree su PubDate per velocizzare queste query. Quali dei due indici è più conveniente avere come clustered B+tree invece di non-clustered? Assumi che gli svantaggi per un indice clustered siano esattamente gli stessi per entrambe le relazioni. Concentrati solo sui benefici offerti dagli indici clustered. Spiega il motivo e giustifica la tua risposta.

Consider the following two queries: (q1) "find all writers with age=30" and (q2) "find all papers published on 09/06/2002." Assume the number of qualifying tuples (that is, the number of tuples in the result) is the same for both queries. Assume that you want to create a B+ tree index on age and a B+ tree index on PubDate to speed up these queries. Which of the two choices will be more beneficial to do as a clustered B+tree index as opposed to unclustered? You can assume that the problems that clustered indexed have are equally bad for both relations. Concentrate only on the benefit that the clustered index can offer. Explain why. Justify your answer.

Risposta/Answer: clustered B+tree su PubDate perché clustered ordina le tuple su disco ed essendo ~~ma~~ i valori che non cambiano nel tempo (il contrario di age aggiornato ogni anno).

Digione
sotto vander
delle tuple
↓
più tuple filter
in una pagina
migliore sarà
il rendimento
con clustered

Considera la query "trova tutti i writers con age>30." Supponi che la relazione Writer sia clustered e che ci sia un indice B+tree non-clustered sull'attributo Age. Assumi che il numero di tuple qualificate sia N. Per quali valori di N una lettura completa della tabella Writer è più conveniente rispetto a utilizzare un indice? Oltre alla risposta finale illustra anche i calcoli che hanno portato ad essa.

Consider the query "find all writers with age>30." Suppose that the Writer relation is clustered and that there is an un-clustered B-tree index on attribute Age. Let the number of qualifying tuples be N. For what values of N is a table-scan of the Writer relation cheaper than using the index? Illustrate apart from the final answer, how you reached that conclusion.

Risposta/Answer:



$\text{Scan } B =$ $B < L_B + N_c$ $N_c \geq B - L_B$	Scan on Writer: $\begin{cases} 4000 \\ 2000 \\ 50 \end{cases} = 200$ cost on B+tree $4 + N$ $200 < 4 + N$
---	---

$K \cap L$

Considera le relazioni $R[A,B,C]$ e $S[B,D,E]$, e assumi che R abbia K tuple che occupino M pagine, e S abbia L tuple che occupino N pagine. Assumi ci sia un indice hash per la coppia (B,D) di S . L'attributo B è chiave per S . La migliore strategia fra le seguenti per il join $R \bowtie_B S$ ha costo:

Consider relations $R[A,B,C]$ and $S[B,D,E]$, and assume that R has K tuples occupying M pages, and S has L tuples occupying N pages. There is a 2-attribute hash index on the attribute pair (B,D) of the relation S . B is the key for S . The cost of the best strategy we can do for the join $R \bowtie_B S$ is:

$$A : M + K$$

$$B : M + K * (\log N + 1)$$

$$C : M + K * N$$

$$D : M + K * 1,2 * N$$

$$E : M + K * (1,2 + 1) \text{ or this?}$$

$$F : M + K * 2 * N * \log N$$

hash function non può essere utilizzata
 perché join perderebbe join è su B
 mentre l'index è su (B,D) .
 Se hash index solo su $B \Rightarrow$ risposta Θ

$$\text{Costo tot di } R = N_R \cdot t_R$$

$$\text{tipe in 1 pag} = \frac{N_R \cdot t_R}{750}$$

$$4 \cdot T.c = 4 \text{ byte}$$

$$T.c \text{ red fact } 0.25$$

Considera le 3 tabelle R, S e T

Una pagina ha dimensione 750 byte

R ha $N_R = 10000$ tuple, ciascuna di dimensione $t_R = 50$ byte

S ha $N_S = 500$ tuple, ciascuna di dimensione $t_S = 350$ byte

T has $N_T = 100K$ tuples each of which is $t_T = 1000$ bytes long

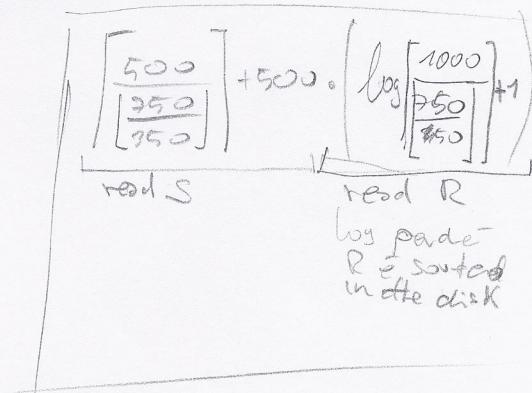
c è un attributo di T di tipo integer

Il fattore di selettività dell'attributo $T.c$ è 25%

L'attributo a è chiave primaria per R e b è una chiave esportata di S che si riferisce a $R.a$

Il costo di accesso di un B+tree è 4 e quello di un indice Hash 1.2

Non ci sono attributi che possono essere NULL



Consider 3 relations R, S and T

A page is of 750 bytes

R has $N_R = 10000$ tuples each of which is $t_R = 50$ bytes long

S has $N_S = 500$ tuples each of which is $t_S = 350$ bytes long

T has $N_T = 100K$ tuples each of which is $t_T = 1000$ bytes long

c is an attribute of T and is an integer

The selectivity factor of the attribute $T.c$ is 25%.

Attribute a is key for R and b is a FK of S referencing $R.a$

The lookup cost of a B+tree is 4 and of a Hash is 1.2

There are no attributes that can be NULL

Si assuma che R e S sono ordinate su disco e che non esista alcun indice, scrivi la formula che calcola il costo di trovare $R \bowtie_{R.a=S.b} S$ e ritornare il risultato a schermo per l'utente.

Assuming that R and S are ordered on the disk and no index exists, write down the formula that computes the cost of finding $R \bowtie_{R.a=S.b} S$ and printing the results on the screen of the user.

$$\cancel{\text{tuple } R} * \cancel{\text{tuple } S} + \cancel{\text{tuple } S}$$

$$t_R \cdot t_S + t_S$$

Si assume che R ha un indice B+tree sull'attributo a, considerando il piano di esecuzione sotto riportato. Per ogni nodo nel piano di esecuzione, **calcola il costo per eseguire l'operazione** (alla voce C1, C2, C3, C4), e **stima il numero di tuple prodotte** (alla voce T1, T2, T3, T4). Non eseguire alcun calcolo, limitarsi a riportare la formula necessaria. La formula può contenere solo numeri, operatori (+, -, *, /) e variabili tra C1, C2, C3, C4, e T1, T2, T3, T4.

Assuming that R has a B+tree index on attribute a, and the query execution tree below. For every node of the tree, **compute the Cost of implementing the operation** (noted as C1, C2, C3, C4) and **the tuples produced** (noted as T1, T2, T3, T4). Do not make any calculations. Simply write down the formula. The formula can have only numbers, operations (like + - * /), and variables from the C1, C2, C3,... and from the T1, T2, T3, ... etc.

		T4:
	Π_a	C4:
	T3:	
	\bowtie	C3:
	T1: T2:	
Index Nested Loop Join	a=b	C1:
R[a,x,y]	S[b,u,v]	
$C1 = \frac{Mr \cdot fr}{750} + fr \cdot \frac{Ns \cdot ts}{750}$		$\text{RSAR parte } \rightarrow R(A) \text{ ha B+tree}$
T1 =		$\left[\begin{array}{c} 500 \\ 750 \\ 350 \end{array} \right] + 500 \cdot (4+1)$
C2 = B		$\text{SAR} \rightarrow \text{ogni v. d'una S mappa con v. solo}$
T2 = $N_f \cdot 0.25$	X	$\text{v. v. in R} \Rightarrow \text{n. tuple in S}$
C3 =		500
T3 =		$2 \cdot 100 \text{ tuple paralel: una tuple}$
C4 =		$\text{occupa 2 byte (750 byte per page, e tuple}$
0.5		occupa 1000 byte)
T4 =		\checkmark
		$\left[\begin{array}{c} 500 \\ 750 \\ 350 \end{array} \right] + 0.25 \cdot 100k \text{ or } \left[\begin{array}{c} T_1 \\ 750 \\ 350 \end{array} \right] + \frac{T_2}{0.5}$
		CARTESIAN PRODUCT
		$T_1 \cdot T_2$
		O perché una selezione di una colonna non costa nulla.
		T_3 perché non deve fare operazioni aggiuntive per stamparla > schemma.