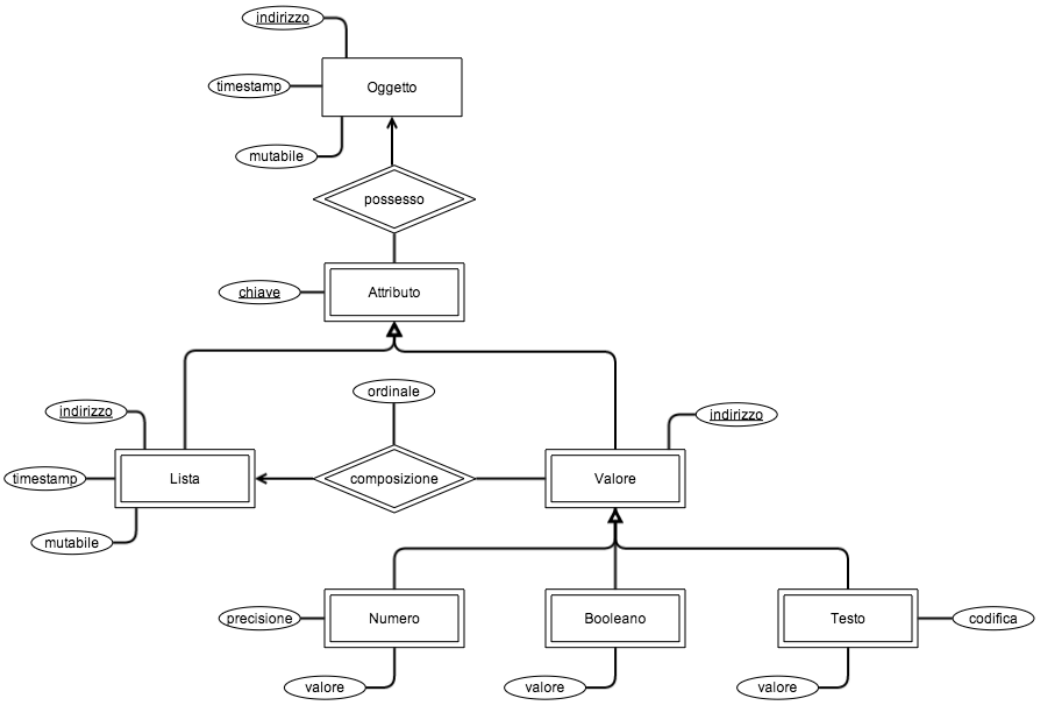
Esercitazione Basi di Dati

Modello Entità Relazione:



Si espliciti ogni assunzione necessaria.

- un oggetto non può avere multiple chiavi con lo stesso nome
- ogni indirizzo di memoria fa riferimento alla stessa memoria
- diverse entità Valore possono contenere lo stesso valore in diversi indirizzi di memoria (es. il valore 15 salvato in due diverse locazioni di memoria per due oggetti diversi)
- i valor di una lista seguono l'ordine di inserimento
- l'ordinato dei valori può non essere continuo

Modello Relazionale:

Si converta il modello E/R in modello relazionale.

```
Oggetto        ( indirizzo , timestamp , mutabile)
Attributo      ( indirizzo_oggetto , chiave , indirizzo )
Lista          ( indirizzo , timestamp , mutabile )
ValoriLista    ( indirizzo_lista , indirizzo_valore , ordinale )
ValoreNumerico ( indirizzo , valore, precisione )
ValoreBooleano ( indirizzo , valore )
ValoreTestuale ( indirizzo , valore, codifica )
```

Le coppie in `(indirizzo_oggetto, indirizzo) in Attributo` devono essere uniche per uno stesso oggetto affinchè un inidirizzo non sia assegnato a multipli attributi.
Gli indirizzi in `Attributo` devono essere unici in modo che oggetti diversi non abbiamo attributi con lo stesso inidirizzo.

Algebra Relazionale:

1. $\pi_{chiave}\ (\sigma_{indirizzo\_oggetto\ =\ '3939'}(Attributo))$
   `-- si assume che l'oggetto 3939 sia effetivamente mutabile`

2. $\pi_{indirizzo\_oggetto,\ chiave}\ ($
   $(\rho_{Oggetto(indirizzo\_oggetto,\ time)}(\sigma_{time<'1382620227'}(\pi_{indirizzo,\ timestamp}(Oggetto))))$
   $\bowtie$
   $(\pi_{indirizzo\_oggetto,\ chiave}\ (Attributo))$
   $)$

3. $\pi_{indirizzo\_oggetto,valore,\ precisione}\ ($
   $(\pi_{indirizzo\_oggetto,\ indirizzo}(\sigma_{indirizzo\_oggetto='3939'\ V\ indirizzo\_oggetto='5939'}\ (Attributo)))$
   $\bowtie$
   $(ValoreNumerico)$
   $)$

4. $\pi_{indirizzo,valore}\ ($
   $(\pi_{indirizzo}\ (ValoriLista)$
   $\bowtie$
   $(\pi_{indirizzo,\ valore}(ValoreNumerico)$
   $U\ \pi_{indirizzo,\ valore}(ValoreBooleano)$
   $U\ \pi_{indirizzo,\ valore}(ValoreTestuale))$
   $)$

5. $\pi_{indirizzo\_oggetto}\ ($
   $(\pi_{indirizzo\_oggetto,\ indirizzo}\ (Attributo))$
   $\bowtie$
   $(\pi_{indirizzo}\ (\sigma_{mutabile\ =\ True}(Lista)))$
   $)$

6. $\pi_{indirizzo}\ (Oggetto)\ -\ ($
   $\pi_{indirizzo\_oggetto}\ ($
   $(\pi_{indirizzo\_oggetto,\ indirizzo}\ (Attributo))$
   $\bowtie$
   $(\pi_{indirizzo}\ (Lista))$
   $)$

7. $\pi_{timestamp}(Oggetto) - ($
$\pi_{O1.timestamp}($
$\rho_{O1}(\pi_{indirizzo\_oggetto,\ timestamp}(Oggetto))$
$\bowtie_{O1.timestmap > Ot.timestamp}$
$\rho_{Ot}(\pi_{indirizzo\_oggetto,\ timestamp}(Oggetto))$
$)$
$\cap$
$\pi_{O2.timestamp}($
$\rho_{O2}(\pi_{indirizzo\_oggetto,\ timestamp}(Oggetto))$
$\bowtie_{O2.timestmap < Ot.timestamp}$
$\rho_{Ot}(\pi_{indirizzo\_oggetto,\ timestamp}(Oggetto))$
$)$
$)$

8. $\pi_{indirizzo\_oggetto}($
$(\pi_{indirizzo\_oggetto,\ indirizzo}(Attributo))$
$\bowtie (\pi_{indirizzo}(Lista))$
$) -$
$\pi_{indirizzo\_oggetto}(\pi_{indirizzo\_oggetto,\ indirizzo}($
$(\pi_{indirizzo\_oggetto,\ indirizzo}(Attributo))$
$\bowtie (\pi_{indirizzo}(Lista))$
$)$
$\bowtie_{indirizzo\_oggetto = indirizzo\_oggetto2 \wedge indirizzo > indirizzo2}$
$\rho_{(indirizzo\_oggetto2,\ indirizzo2)}($
$\pi_{indirizzo\_oggetto,\ indirizzo}($
$(\pi_{indirizzo\_oggetto,\ indirizzo}(Attributo))$
$\bowtie (\pi_{indirizzo}(Lista))$
$)$
$)$
$)$

SQL:

```
-- Oggetto        ( indirizzo , timestamp , mutabile)
CREATE TABLE oggetto (
  indirizzo  BIGINT PRIMARY KEY,
  timestamp  TIMESTAMP,
  mutabile   BOOLEAN
);

-- Attributo      ( indirizzo_oggetto , chiave , indirizzo  )
CREATE TABLE attributo (
  indirizzo_oggetto  BIGINT
             REFERENCES oggetto(indirizzo)
             ON DELETE CASCADE
             ON UPDATE CASCADE,
  chiave     VARCHAR(255),
  indirizzo BIGINT,
  PRIMARY KEY (indirizzo_oggetto, chiave),
  UNIQUE (indirizzo_oggetto, indirizzo),
  UNIQUE (indirizzo) – ci assicuriamo indirizzi unici
);

-- a livello di performance è più spesso conveniente avere
-- primary key su oggetti di tipo intero/numerico
-- quindi alternativamente:
  PRIMARY KEY (indirizzo_oggetto, indirizzo),
  UNIQUE (indirizzo_oggetto, chiave),
  UNIQUE (indirizzo)



-- Lista          ( indirizzo , timestamp , mutabile )
CREATE TABLE lista (
  indirizzo  BIGINT PRIMARY KEY
             REFERENCES attributo(indirizzo)
             ON DELETE CASCADE
             ON UPDATE CASCADE,
  timestamp  TIMESTAMP,
  mutabile   BOOLEAN
);



-- ValoriLista    ( indirizzo_lista , indirizzo_valore , ordinale )
CREATE TABLE valori_lista (
  indirizzo_lista  BIGINT
             REFERENCES lista(indirizzo)
             ON DELETE CASCADE
             ON UPDATE CASCADE,
  indirizzo_valore BIGINT,
  ordinale   INT,
  PRIMARY KEY (indirizzo_lista, indirizzo_valore),
  UNIQUE (indirizzo_lista, ordinale),
  UNIQUE (indirizzo_valore)
);
```

```
-- ValoreNumerico ( indirizzo , valore, precisione )
CREATE TABLE valore_numerico (
  indirizzo  BIGINT PRIMARY KEY,
  valore     BIGINT,
  precisione   INT
);


-- ValoreBooleano ( indirizzo , valore )
CREATE TABLE valore_booleano (
  indirizzo  BIGINT PRIMARY KEY,
  valore     BOOLEAN
);

-- ValoreTestuale ( indirizzo , valore, codifica )
CREATE TABLE valore_testuale (
  indirizzo  BIGINT PRIMARY KEY,
  valore     TEXT,
  codifica    VARCHAR(255)             -- oppure un ENUM
);


-- SI evidenzino i problemi inerenti a questa implementazione
è possibile risolverli senza introdurre altri problemi?
```

1. Ogni chiave di attributo dell'oggetto mutabile con indirizzo `3939`

```
SELECT chiave
FROM pjson.attributo
WHERE indirizzo_oggetto = 3939;
```

2. Le chiavi di ogni attributo degli oggetti creati prima del `1382620227`

```
SELECT a.inidirizzo_oggetto, a.chiave
FROM pjson.oggetto o  JOIN pjson.attributo a
     ON o.indirizzo = a.indirizzo_oggetto
WHERE o.timestamp < to_timestamp(1382620227);
```

3. I valori e la precisione di tutti gli attributi numerici degli oggetti in `3939` e`5939`

```
SELECT v.valore, v.precisione
FROM pjson.attributo a JOIN  pjson.valore_numerico v
     ON a.indirizzo = v.indirizzo
       AND a.indirizzo_oggetto IN (3939, 5939);
```

4. Gli indirizzi e i valori che appartengono a una lista.

```
SELECT l.indirizzo_lista, l.indirizzo_valore, v.val
FROM  valori_lista l JOIN (
  (SELECT n.indirizzo, n.valore || '' as val FROM valore_numerico n)
  UNION
  (SELECT b.indirizzo, b.valore || '' as val FROM valore_booleano b)
  UNION
  (SELECT t.indirizzo, t.valore as val FROM valore_testuale  t)
 ) AS v ON l.indirizzo_valore = v.indirizzo;
```

5. Gli indirizzi degli oggetti che contengono valori di tipo lista mutabile.

```
SELECT a.indirizzo_oggetto
FROM pjson.attributo a
     JOIN  pjson.lista v
     ON a.indirizzo = v.indirizzo
       AND v.mutabile = true;
```

6. Tutti gli indirizzi di oggetti semplici

```
SELECT oggetto.indirizzo
FROM pjson.oggetto
EXCEPT(
      SELECT a.indirizzo_oggetto
      FROM pjson.attributo a
      JOIN  pjson.lista v
            ON a.indirizzo = v.indirizzo
      );
```

7. Il timestamp dell'oggetto creato per ultimo e per primo.

```
SELECT o.indirizzo, o.timestamp
FROM pjson.oggetto o
     JOIN (
        SELECT MIN(o1.timestamp) as timestamp FROM pjson.oggetto o1
        UNION
        SELECT MAX(o2.timestamp) as timestamp FROM pjson.oggetto o2
      ) AS times
      ON o.timestamp = times.timestamp
```

8. Gli oggetti che hanno un solo attributo lista.

```
SELECT a.indirizzo_oggetto
FROM pjson.attributo a
     JOIN  pjson.lista v
     ON a.indirizzo = v.indirizzo
GROUP BY a.indirizzo_oggetto
HAVING COUNT(a.indirizzo) = 1
```

1. Per ogni oggetto il numero di attributi testuali con codifica UTF-8 2.

```
SELECT a.indirizzo_oggetto, COUNT(v.indirizzo)
FROM pjson.attributo a
     JOIN  pjson.valore_testuale v
     ON a.indirizzo = v.indirizzo AND v.codifica = 'UTF-8'
GROUP BY a.indirizzo_oggetto;
```

2. Il numero medio di attributi numerici per ogni oggetto

```
SELECT AVG(g.num)
FROM (SELECT a.indirizzo_oggetto, COUNT(v.indirizzo) as num
        FROM pjson.attributo a
        JOIN pjson.valore_numerico v
            ON a.indirizzo = v.indirizzo
      GROUP BY a.indirizzo_oggetto) AS g;
```

3. La lista contenente il numero maggiore di Booleani con valore False

```
SELECT a.indirizzo_oggetto , COUNT(v.indirizzo) as num
        FROM pjson.attributo a
        JOIN pjson.valore_booleano v
            ON a.indirizzo = v.indirizzo AND v.valore = False
      GROUP BY a.indirizzo_oggetto
      HAVING COUNT(v.indirizzo) > ALL
                  (SELECT COUNT(v.indirizzo) as num
                     FROM pjson.attributo a
                     JOIN pjson.valore_booleano v
                       ON a.indirizzo = v.indirizzo
                          AND v.valore = False
                GROUP BY a.indirizzo_oggetto);

-- esistono diverse alternative per ottenere lo stesso risultato.
```

Provate a creare le tabelle nel vostro database Postres, inserite diversi valori di prova e verificate l'esecuzione delle query.

Si accettano correzioni e soluzioni alternative.

Gli studenti che hanno collaborato alla buona riuscita dell'esercitazione venendo alla lavagna sono invitati a mandare un email all'esercitatore.