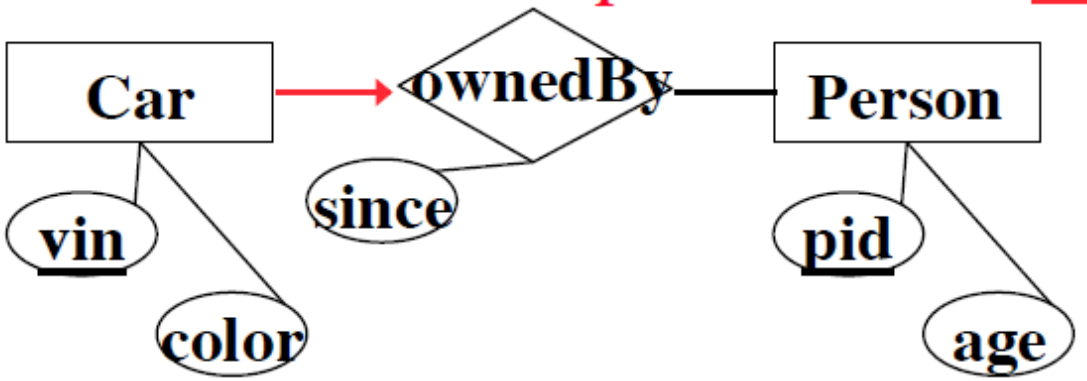
1-N Relationships Revisited e.g.



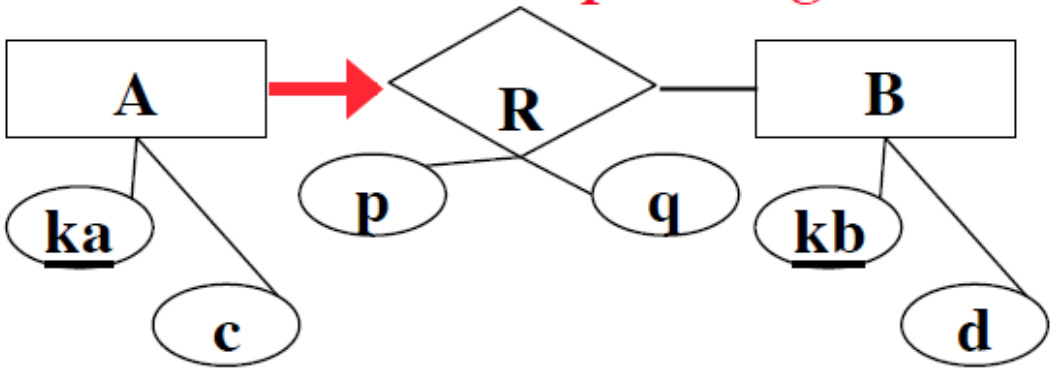
- **Merge** T_ownedBy(vin,pid,since) **into** T_Car(vin, color)

```
table T_Car
        ( vin char(25),
          color,
          pid int references T_Person,
         since date,
        primary key ( vin ) )
```

} *Describes a* **Car**

} *Describes the* **Person** *related to the car by the* **ownedBy** *relation*

1-N Relationship merged: what if total?



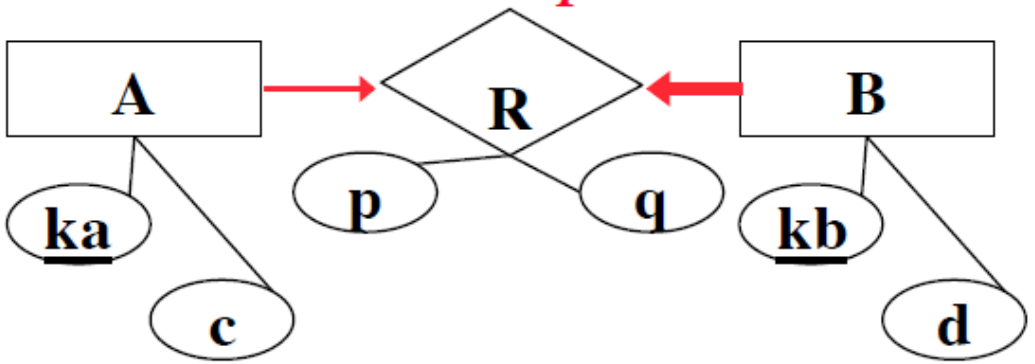
- **Merge** T_R(ka,kb,p,q) **into** T_A(ka, c)

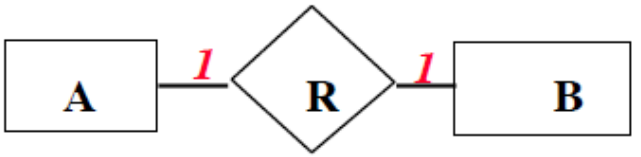
```
table T_A
        ( ka
          c
          R_kb  references T_B  not null
          p
          q
        primary key ( ka )
```

The "total" constraint can now be captured in one table!

Note that if B participated "total", there was still no way to capture this in the the T_A table, merged with R. Need a general assertion.

*1-1 Relationships **Revisited***



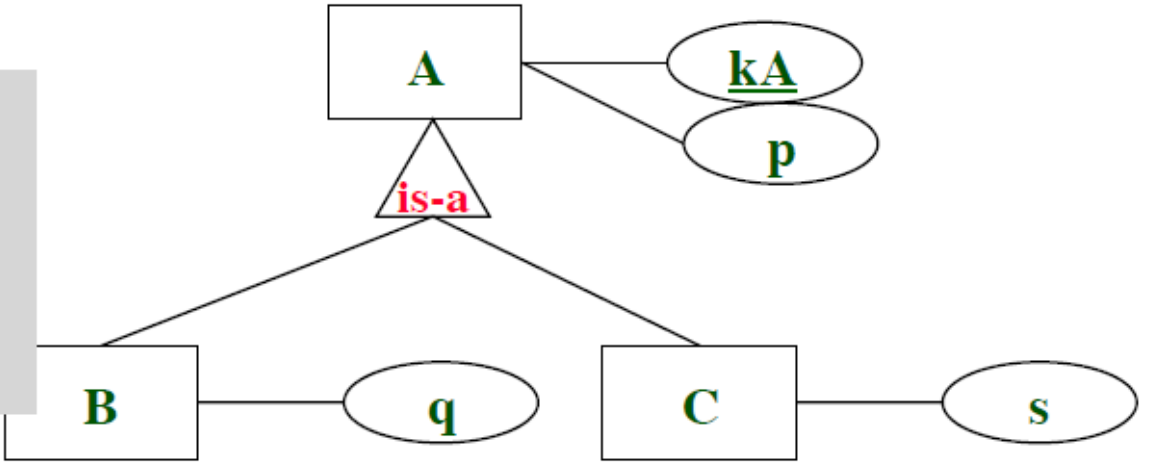


- ***Can merge T_R into T_A or T_B ?***
- ***Choose "total" end, so* NOT NULL *constraint enforces it***

```
table T_B
        ( kb                          } Describes a B
          d
          R_ka references T_A   not null
          p
          q
        primary key ( kb )
```

Mapping Subclasses - 2nd way (merge up)

```
table T_A(
      kA primary key
      p
      q
      s   )
```



- Merge T_B (and/or T_C) into T_A: ***note that merge conditions hold***!
- Note that objects in B but not in C will have null *s*; and objects in A but not B nor C will have null *q* and *s*.
- *Potentially serious violation of "know membership" rule:*
  - » **How can you tell if an object is in subclass B? Only if q value is not null. So this design only applicable when each subclass has some non-null attribute that can act as a discriminator.**
  - » ***Otherwise need a boolean flag column to indicate the subclass:**

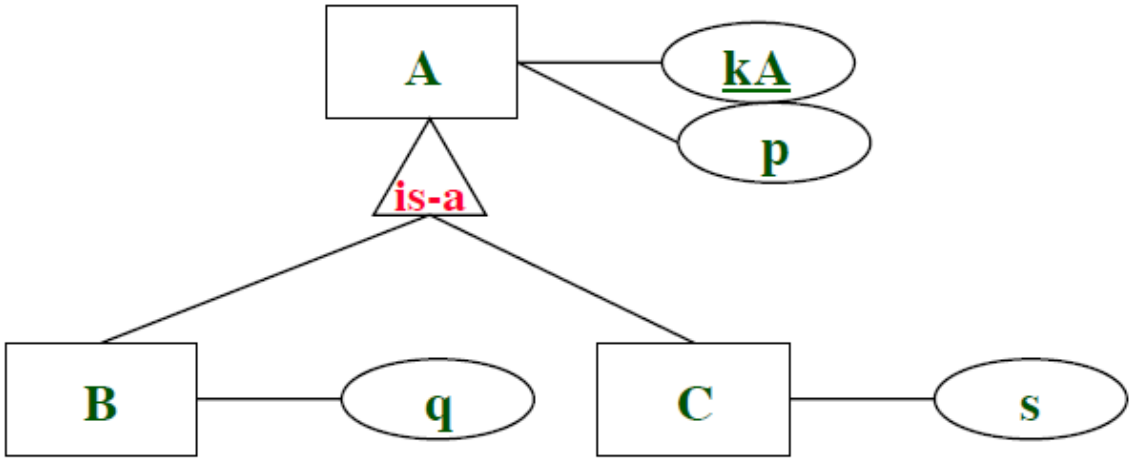
```
inB? boolean,
inC? boolean,
```

What about total and disjoint constraints?

*Mapping Subclasses: 3rd way (collapse down - not a 'merge'

```
table T_B(
      kA primary key
      p
      q )
```

```
table T_C(
      kA primary key
      p
      s  )
```



- This design is used when subclasses **cover total** superclass
  - » **otherwise, to represent membership in class A alone is "fake": need to mark "non-Bs" in T_B and "non-Cs" in T_C (e.g., assume *q* and *s* are not null for B and C)**
- and are **disjoint**
  - » **otherwise A information, such as *kA*, *p* is repeated in both tables for some object <-- against no duplication principle (see later)**

DB table design principles that motivate merging tables:

From database point of view: storage efficiency, making queries easier to state, making integrity checking faster, etc.

(*i*) **Repetition** of data is bad if it is **unnecessary**

e.g. enrolledIn(studntId, courseId, studntAddress, grade)

because

- **need to verify consistency** ***(do all occurrences of studntId=4454 have the same address?)***
- **need to maintain consistency** (***when studntID 4454 changes address, all tuples need to be sought and changed)***
- **wasted space**

The above example involved repeating non-key information *(studntAddress)*. Note that key info cannot be repeated by definition of "key".

DB Table design principles (cont'd)

(*ii*) Having fewer tables is better because queries require fewer joins, which are

- » **easier to write for the user**
- » **less expensive to evaluate for the DBMS**

(*iii*) Having tables with too many nulls is undesirable

As usual, there are trade-offs.

Merge rule is a consequence of DB design principles

If you have two tables of the form T(K,X) and S(K2,Y) , *Y not null*, where K2 is a foreign key referencing T(K) then you can merge S into T to get instead a single table TS(K,X,Y). Column Y will have NULLs for all keys in T but not S.

- This is good by principle (***ii***): avoid joins.
- By def'n of key, for every K' value there is only one Y, and for every K there is only one X, so when you combine them you get only 1 row, and principle "(***i) no repetition***" is preserved
- If T('ann' ,45) has no matching S('ann' ,...), we need to use null in final table: T('ann' ,45,NULL). This may be a problem with "(iii) not too many nulls", if there are very many columns in S, and very many T's that are not in S.