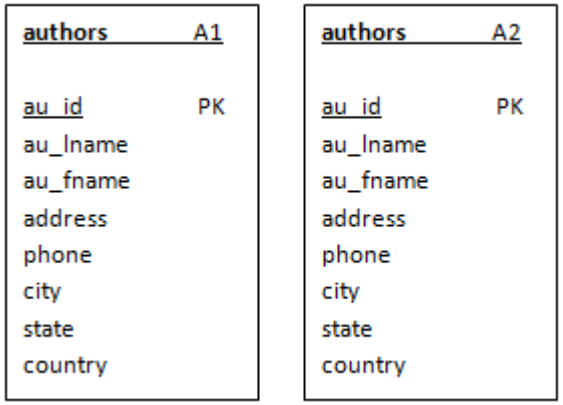


Les paires d'auteurs qui vivent dans la même ville

```
SELECT a1.au_lname, a2.au_lname
FROM authors a1, authors a2
WHERE a1.city=a2.city
```

⇒ 49 résultats

	au_lname character varying(40)	au_lname character varying(40)
1	Bennet	Carson
2	Bennet	Bennet
3	Green	MacFeather
4	Green	Karsen
5	Green	Straight
6	Green	Stringer
7	Green	Green
8	Carson	Carson
9	Carson	Bennet
10	Ringer	Ringer
11	Ringer	Ringer
12	Ringer	Ringer
13	Ringer	Ringer
14	DeFrance	DeFrance
15	Panteley	Panteley
16	McBadden	McBadden
17	Stringer	MacFeather
18	Stringer	Karsen
19	Stringer	Straight
20	Stringer	Stringer
21	Stringer	Green
22	Straight	MacFeather
23	Straight	Karsen
24	Straight	Straight
25	Straight	Stringer

authors	A1
au_id	PK
au_lname	
au_fname	
address	
phone	
city	
state	
country	

authors	A2
au_id	PK
au_lname	
au_fname	
address	
phone	
city	
state	
country	

	au_lname character varying(40)	au_lname character varying(40)
1	Bennet	Carson
2	Green	MacFeather
3	Green	Karsen
4	Green	Straight
5	Green	Stringer
6	Carson	Bennet
7	Ringer	Ringer
8	Ringer	Ringer
9	Stringer	MacFeather
10	Stringer	Karsen
11	Stringer	Straight
12	Stringer	Green
13	Straight	MacFeather
14	Straight	Karsen
15	Straight	Stringer
16	Straight	Green
17	Karsen	MacFeather
18	Karsen	Straight
19	Karsen	Stringer
20	Karsen	Green
21	MacFeather	Karsen
22	MacFeather	Straight
23	MacFeather	Stringer
24	MacFeather	Green
25	Dull	Hunter
26	Hunter	Dull

Les paires d'auteurs qui vivent dans la même ville

SELECT a1.au_lname, a2.au_lname

FROM authors a1, authors a2

WHERE a1.city=a2.city

AND a1.au_id<>a2.au_id

⇒ 26 résultats

Mais Bennet – Carson et Carson – Bennet...

Chaque combinaison apparaît en double !

a1				a2
Green	Oakland		Oakland	Green
Carson	Berkeley		Berkeley	Carson
Straight	Oakland		Oakland	Straight
Bennet	Berkeley		Berkeley	Bennet
Dull	Palo Alto		Palo Alto	Dull
Stringer	Oakland		Oakland	Stringer
MacFeather	Oakland		Oakland	MacFeather
Karsen	Oakland		Oakland	Karsen
Hunter	Palo Alto		Palo Alto	Hunter
Ringer	Salt Lake City		Salt Lake City	Ringer
Ringer	Salt Lake City		Salt Lake City	Ringer

authors	A1
au_id	PK
au_lname	
au_fname	
address	
phone	
city	
state	
country	

authors	A2
au_id	PK
au_lname	
au_fname	
address	
phone	
city	
state	
country	

Les paires d'auteurs qui vivent dans la même ville

```
SELECT a1.au_lname, a2.au_lname
FROM authors a1, authors a2
WHERE a1.city=a2.city
AND a1.au_id<a2.au_id
```

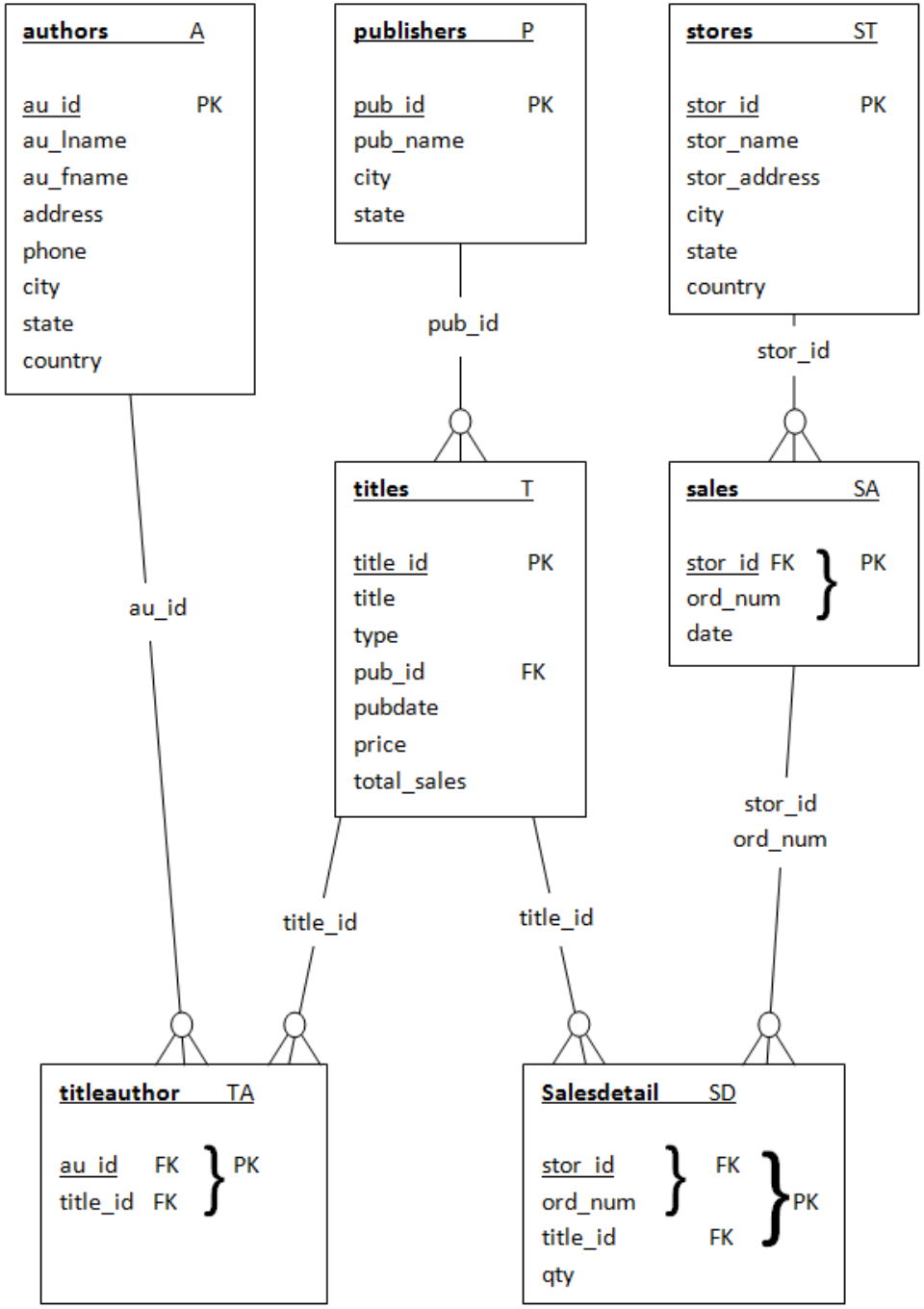
⇒ 13 résultats

	au_lname character varying(40)	au_lname character varying(40)
1	Green	MacFeather
2	Green	Karsen
3	Green	Straight
4	Green	Stringer
5	Carson	Bennet
6	Ringer	Ringer
7	Stringer	MacFeather
8	Stringer	Karsen
9	Straight	MacFeather
10	Straight	Karsen
11	Straight	Stringer
12	MacFeather	Karsen
13	Dull	Hunter

a1				a2
Green	Oakland		Oakland	Green
Carson	Berkeley		Berkeley	Carson
Straight	Oakland		Oakland	Straight
Bennet	Berkeley		Berkeley	Bennet
Dull	Palo Alto		Palo Alto	Dull
Stringer	Oakland		Oakland	Stringer
MacFeather	Oakland		Oakland	MacFeather
Karsen	Oakland		Oakland	Karsen
Hunter	Palo Alto		Palo Alto	Hunter
Ringer	Salt Lake City		Salt Lake City	Ringer
Ringer	Salt Lake City		Salt Lake City	Ringer

Opérateurs d'aggrégation

```
COUNT ( [ ALL | DISTINCT ] expression )|
SUM   ( [ ALL | DISTINCT ] expression )|
MIN   ( [ ALL | DISTINCT ] expression )|
MAX   ( [ ALL | DISTINCT ] expression )|
AVG   ( [ ALL | DISTINCT ] expression )|
COUNT ( * )
```

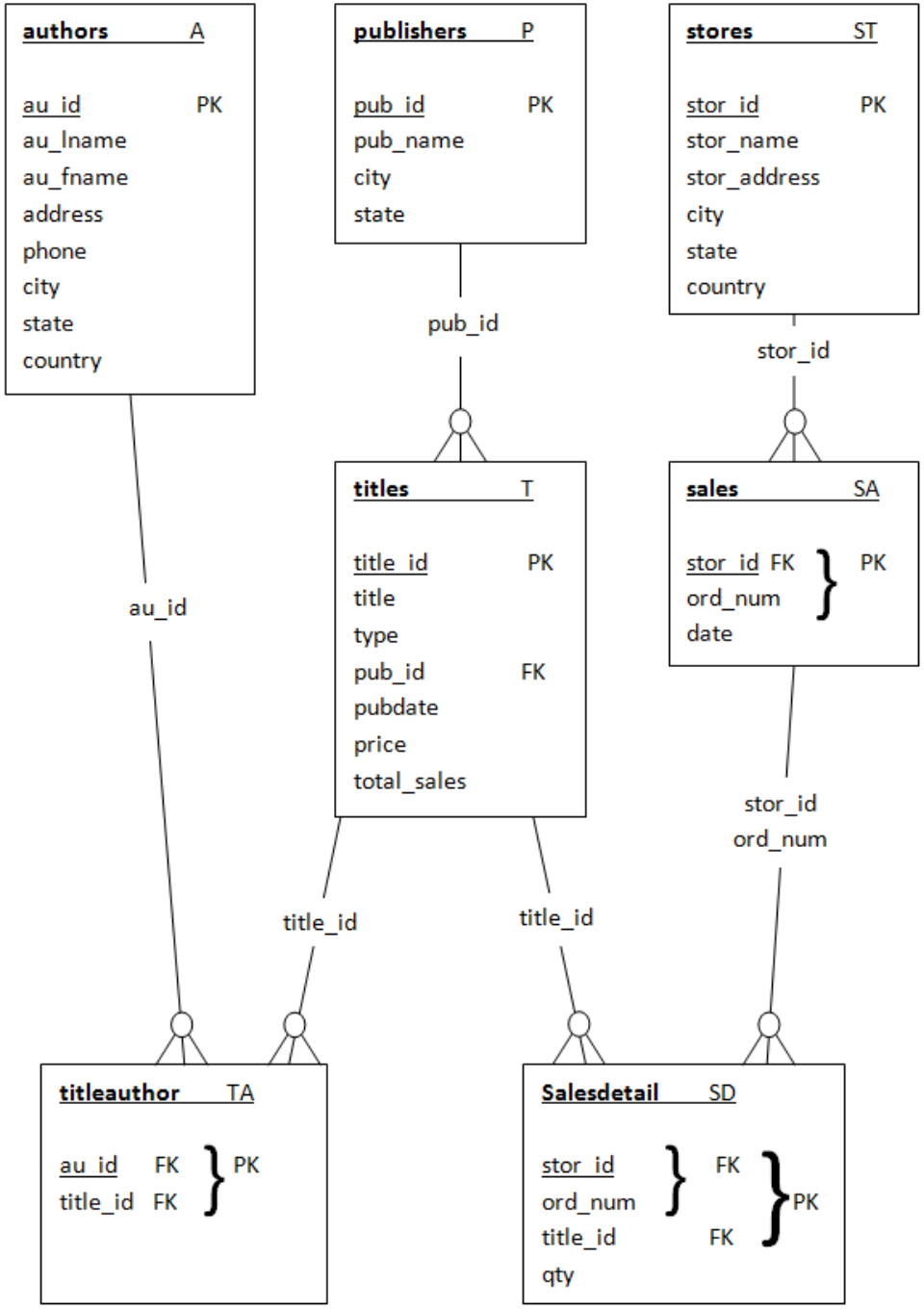


Le nombre d'auteurs

SELECT COUNT(*)

FROM authors

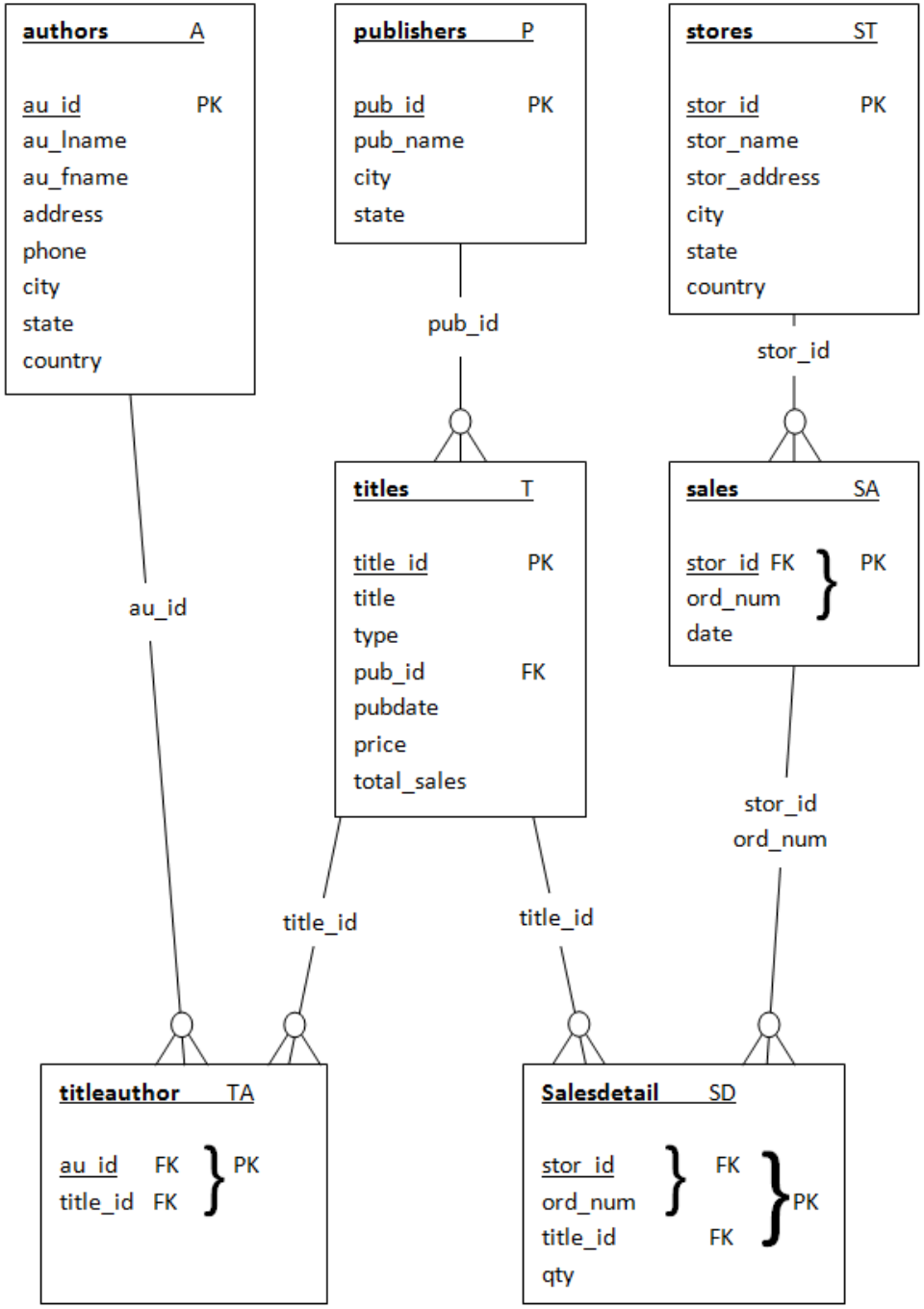
Aggrégation de tous les résultats



Le nombre d'auteurs par livre

```
SELECT COUNT(*), t.title
FROM titleauthor ta,
     titles t
WHERE ta.title_id=t.title_id
GROUP BY t.title_id
```

Aggrégation par sous groupe



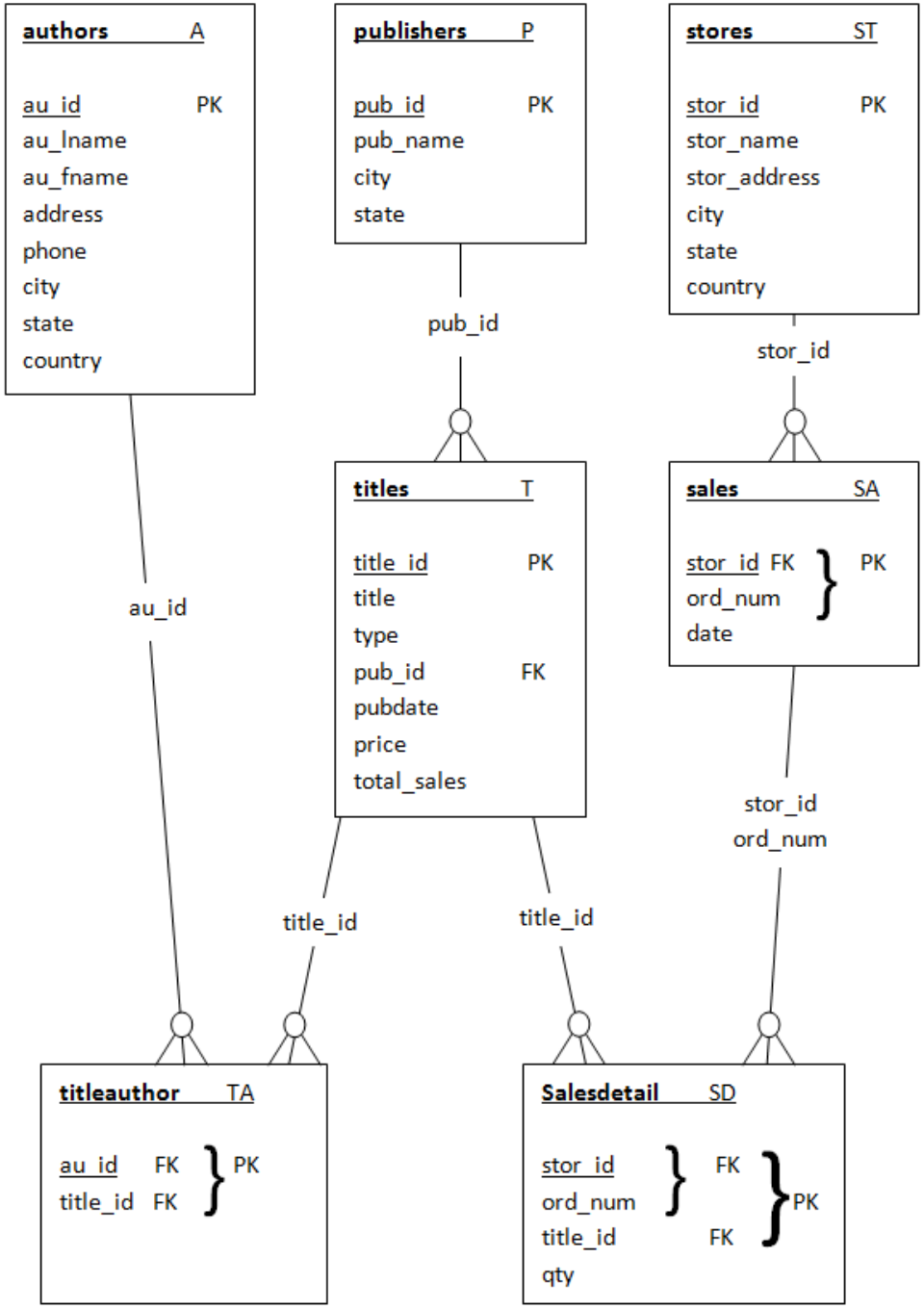
Les auteurs sans téléphone

SELECT *

FROM authors a

WHERE a.phone IS NULL

NULL s'utilise toujours par IS NULL ou IS NOT NULL, jamais par =NULL ou <>NULL



Les auteurs sans téléphone

SELECT *

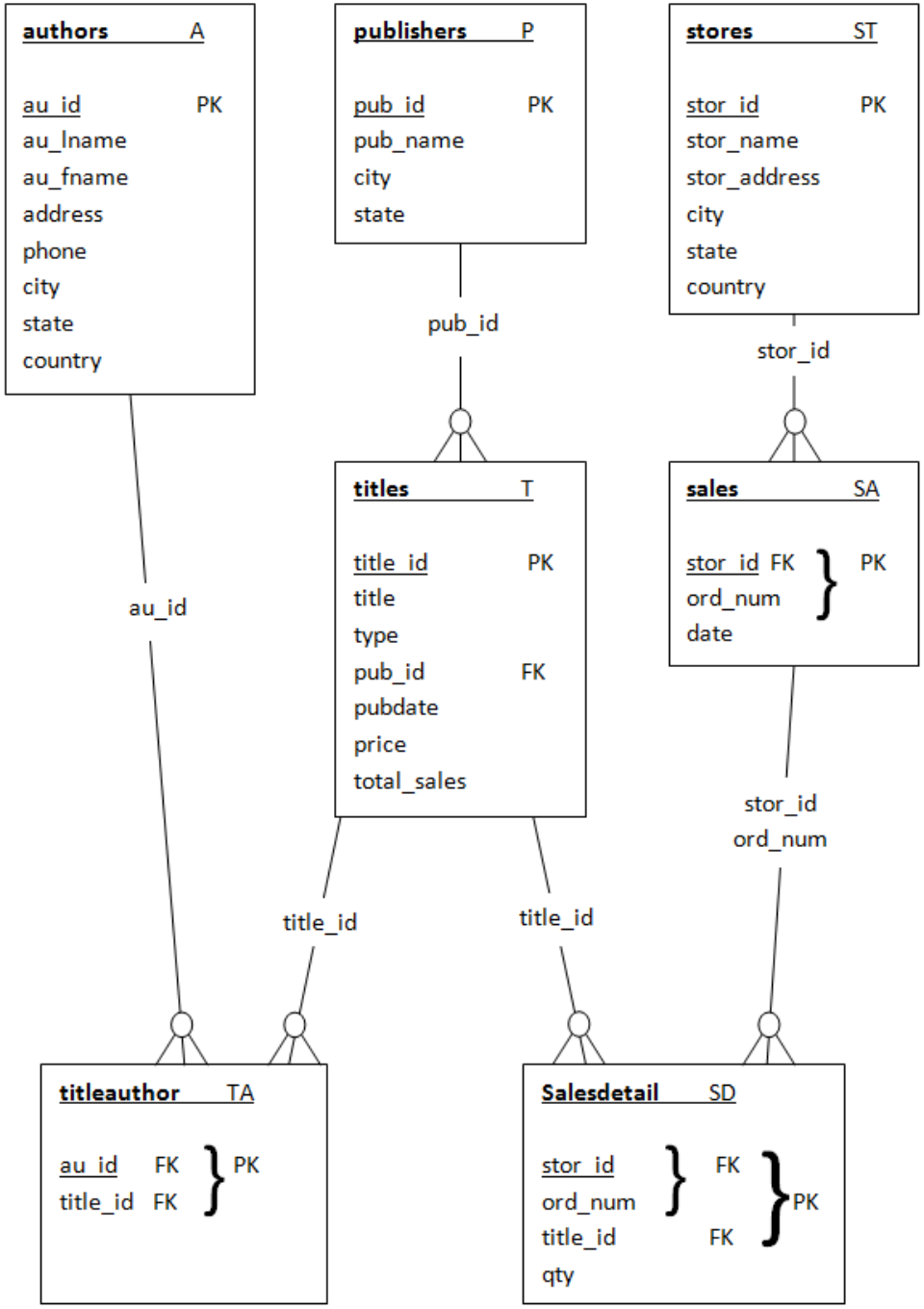
FROM authors a

WHERE a.phone LIKE ''

Parfois on encode un champ textuel vide non pas par NULL mais bien par une chaîne vide.

Dans le doute, il faut demander !

Mais cette requête n'est pas correcte!

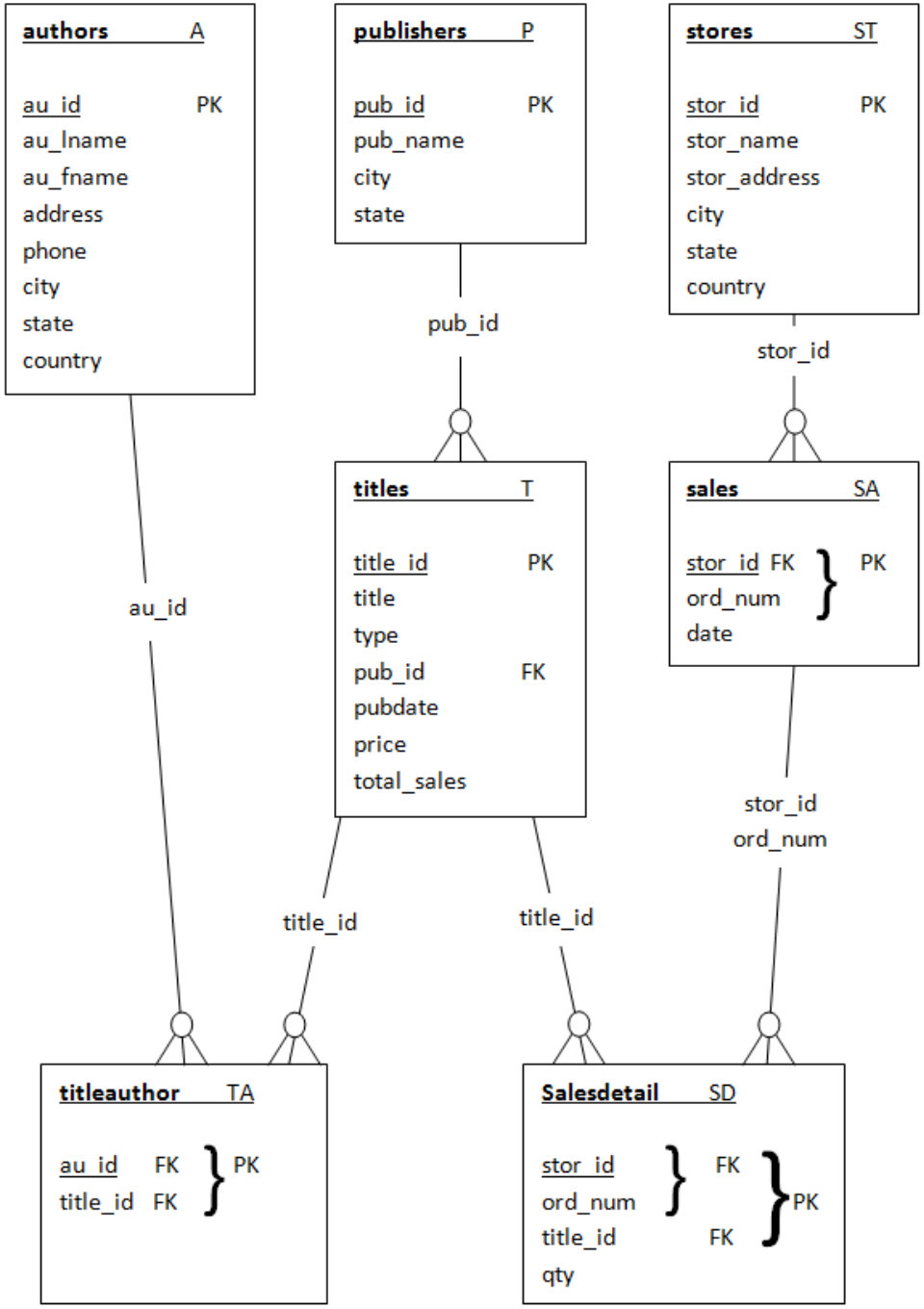


Les auteurs sans téléphone

Property	Value
Name	phone
Position	4
Data type	character(12)

le LIKE s'attend à couvrir tout le champ de taille fixe !

```
SELECT *
FROM authors a
WHERE a.phone LIKE '          '
```



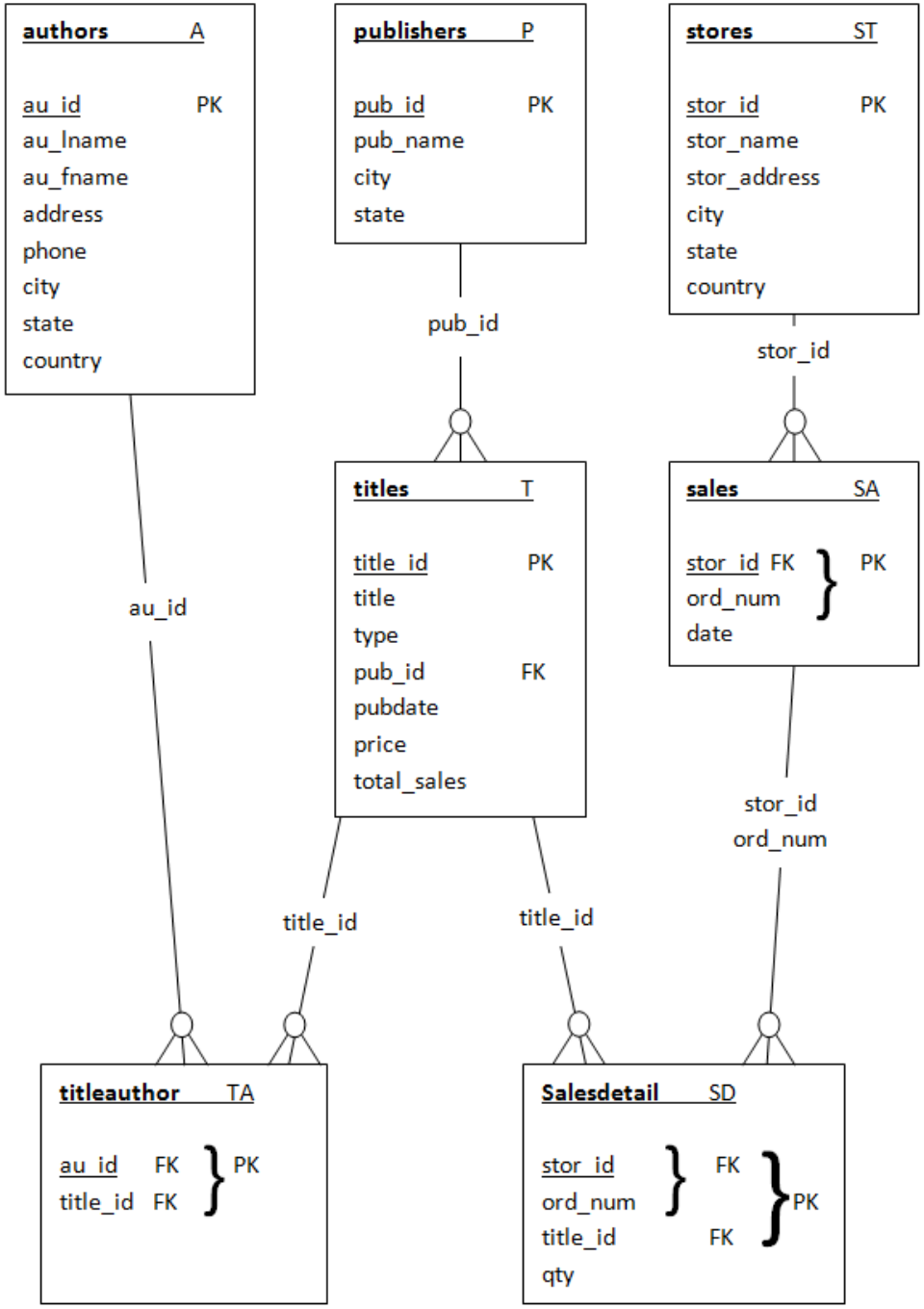
Les auteurs sans téléphone

SELECT *

FROM authors a

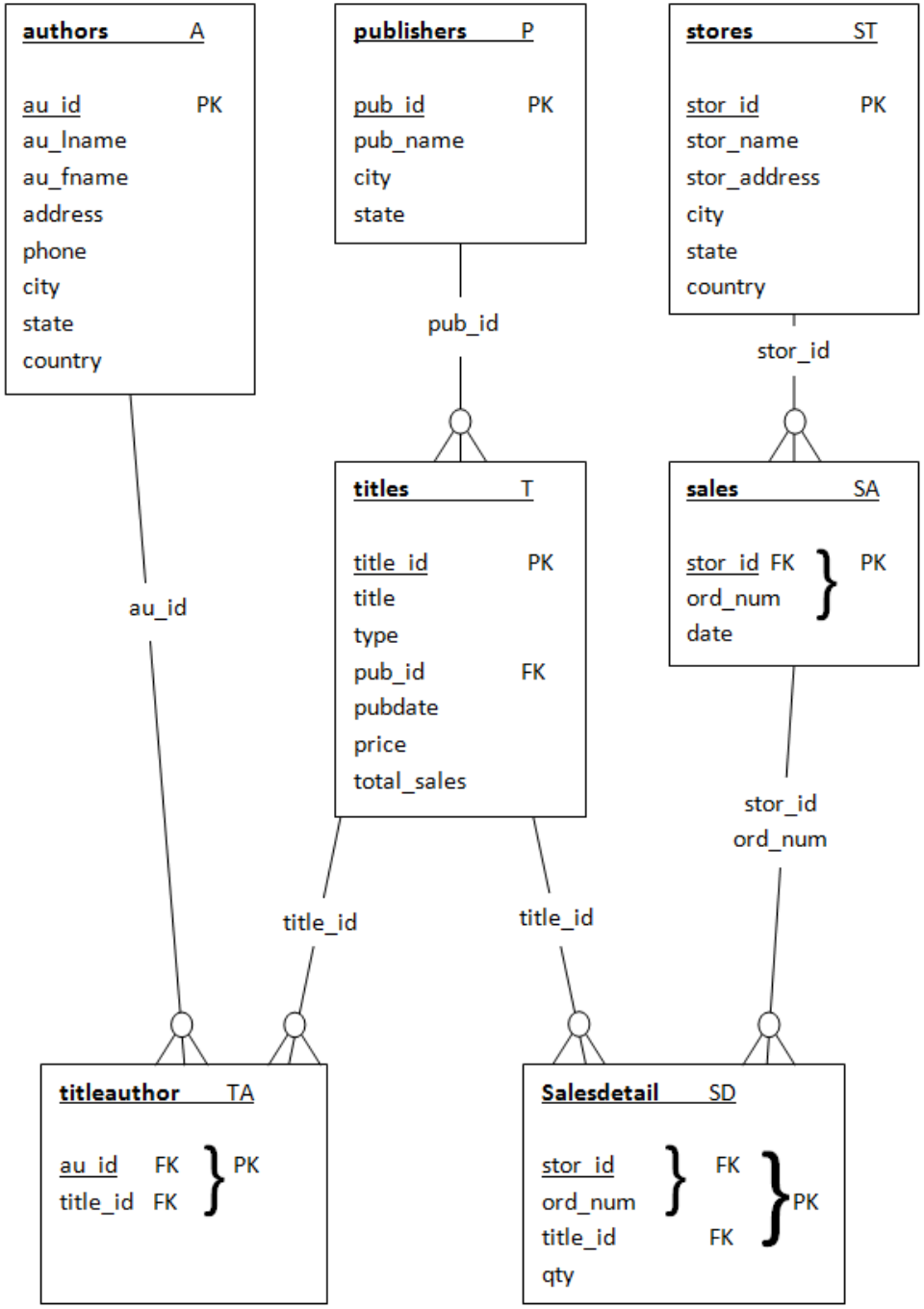
WHERE a.phone = ''

Etrangement le = n'a pas ce problème
et la requête ici est donc correcte.



Les auteurs en Californie, Oregon et Michigan

```
SELECT *
FROM authors a
WHERE a.state IN ('CA','OR','MI')
```

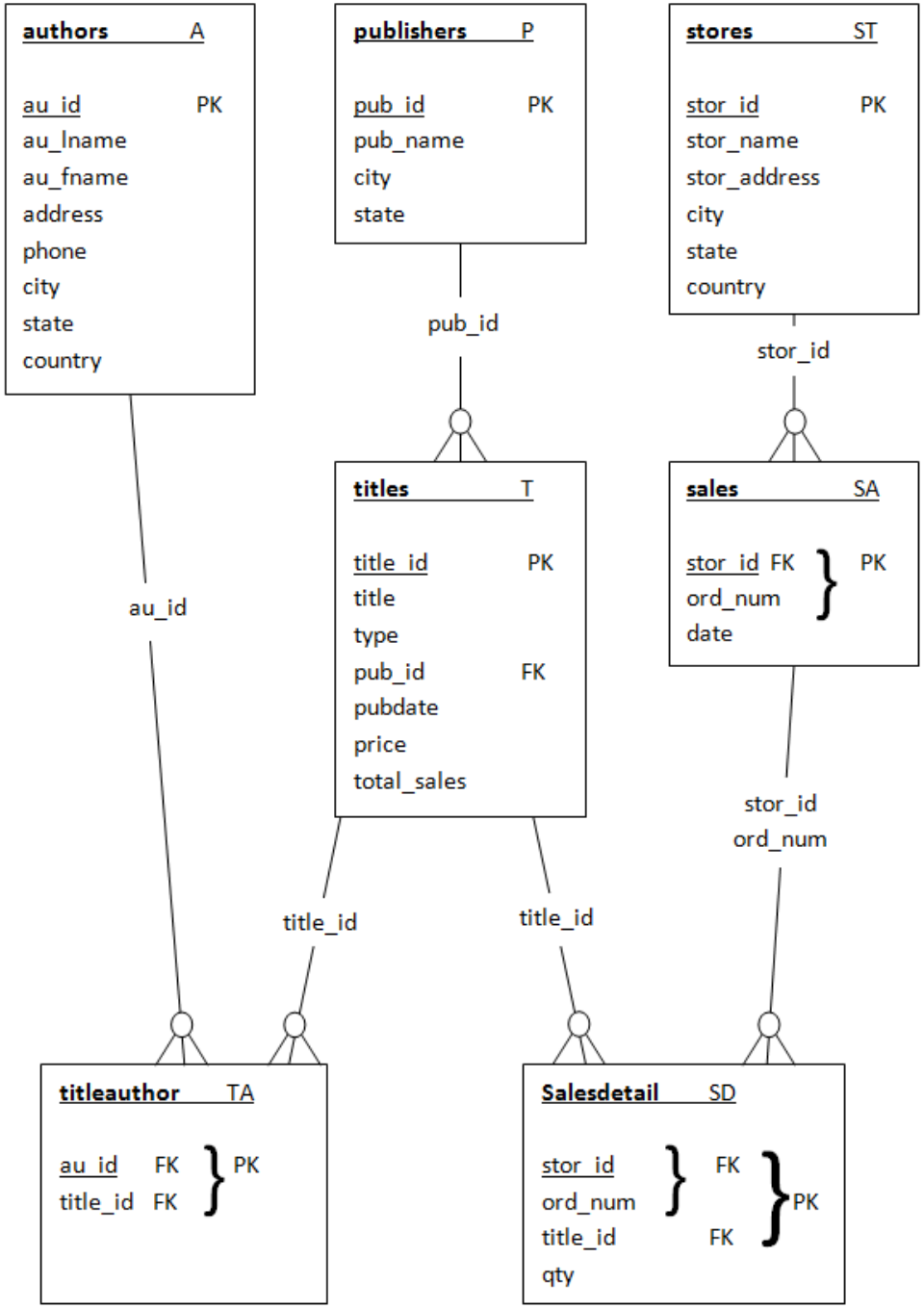


Les ventes en 1991

```
SELECT *
FROM sales s
WHERE s.date>='1991-01-01'
AND s.date<'1992-01-01‘
```

```
SELECT *
FROM sales s
WHERE s.date BETWEEN  '1991-01-01'
                  AND '1991-12-31'
```

```
SELECT *
FROM sales s
WHERE date_part('YEAR',s.date)=1991
```

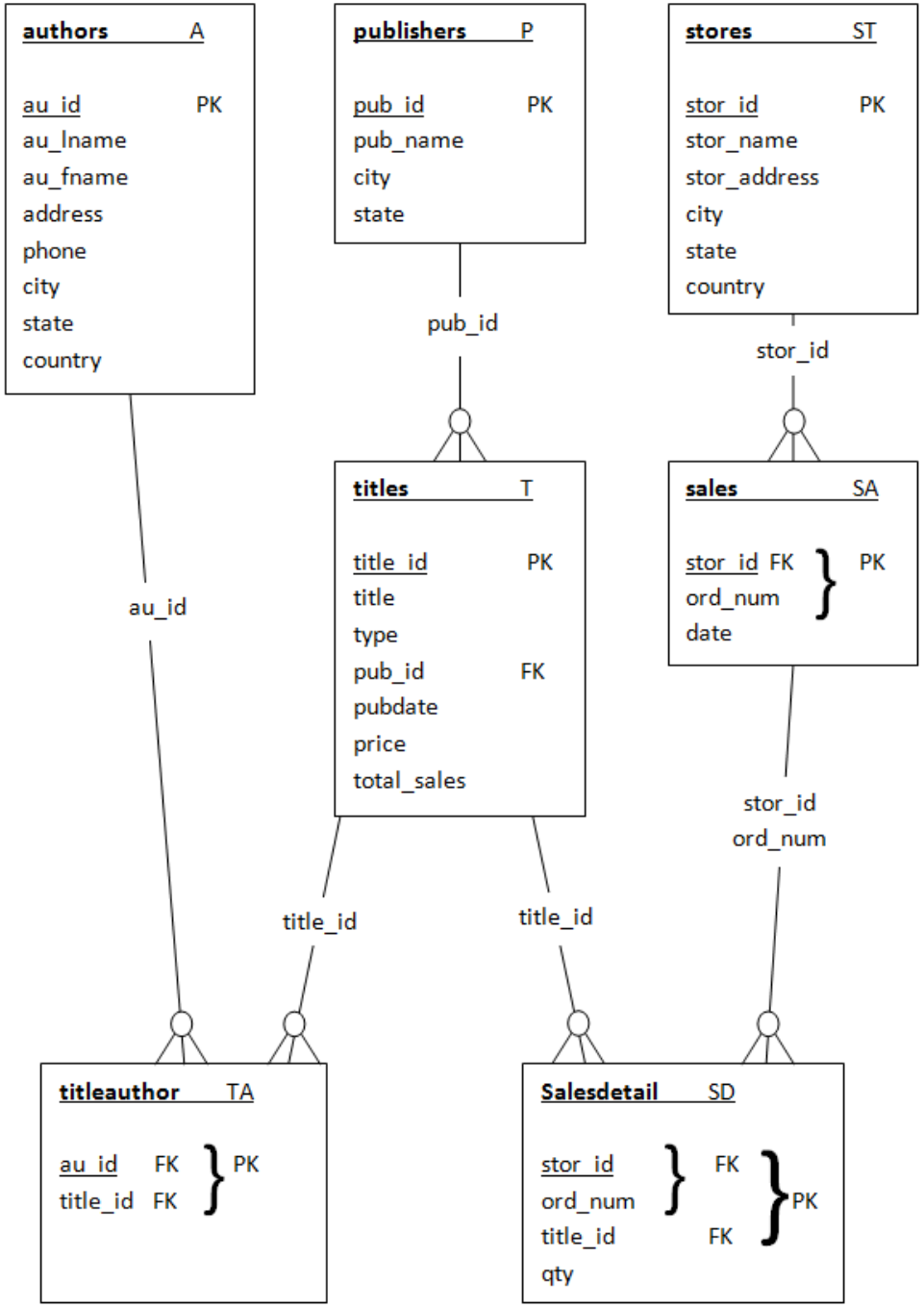


Magasins ayant vendu livre contenant cook ?

```
SELECT DISTINCT
    st.stor_id,st.stor_name
FROM stores st, salesdetail sd, sales
    sa, titles t
WHERE st.stor_id=sa.stor_id
AND sa.stor_id=sd.stor_id
AND sa.ord_num=sd.ord_num
AND sd.title_id=t.title_id
AND t.title SIMILAR TO '%[cC]ook%'
```

Rappel mathématique : la transitivité

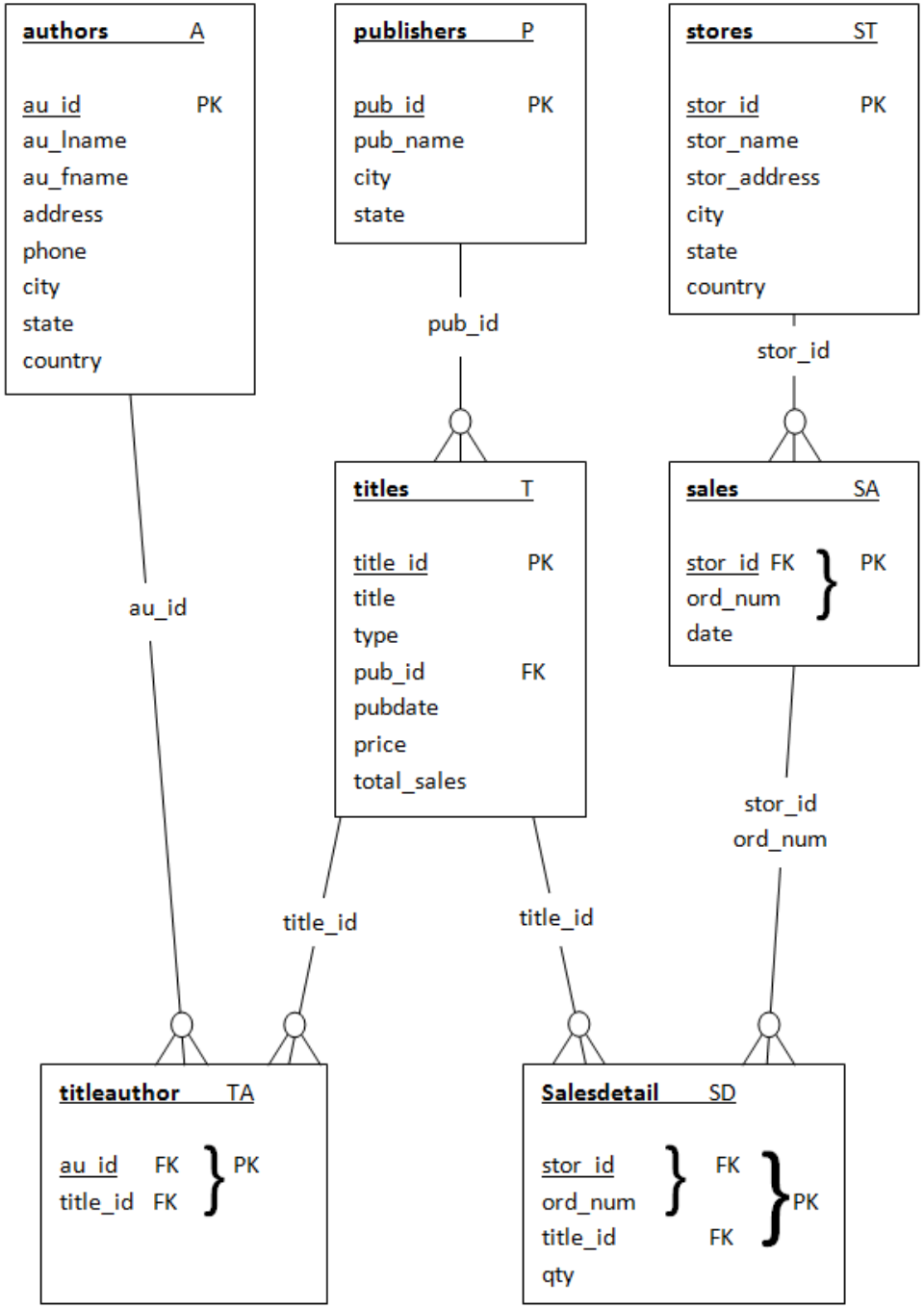
- Si A=B et B=C….
  - Alors A=C !




```
SELECT st.stor_name
FROM stores st, salesdetail sd,
    sales sa, titles t
WHERE st.stor_id=sa.stor_id
AND sa.stor_id=sd.stor_id
AND sa.ord_num=sd.ord_num
AND sd.title_id=t.title_id
AND t.title SIMILAR TO '%[cC]ook%'
```

Simplification…

```
SELECT st.stor_name
FROM stores st, salesdetail sd, titles t
WHERE st.stor_id=sd.stor_id
AND sd.title_id=t.title_id
AND t.title SIMILAR TO '%[cC]ook%'
```

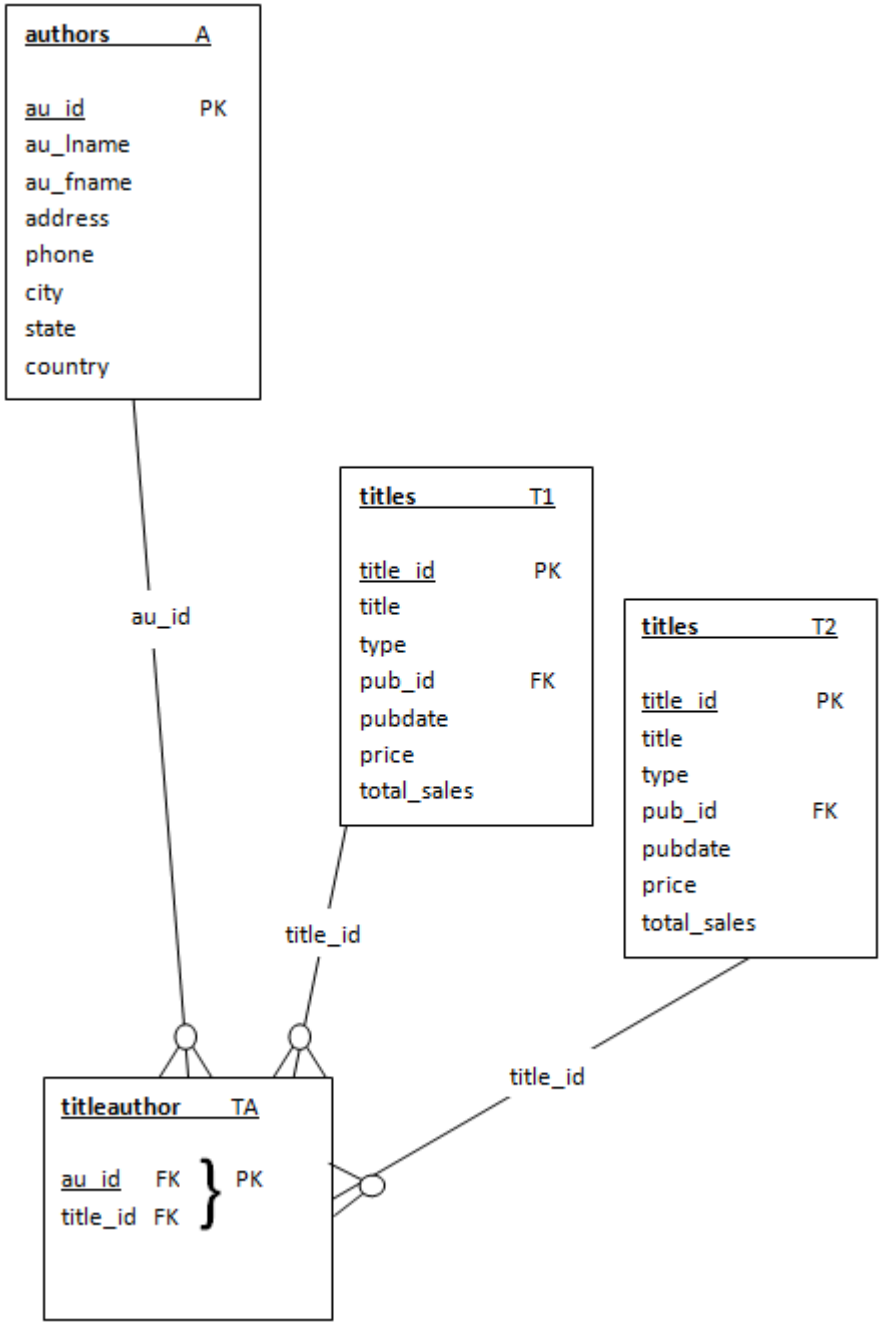


Auteurs n'ayant pas tout publié au même éditeur ?

On ne peut pas répondre à cette question directement sous cette forme.

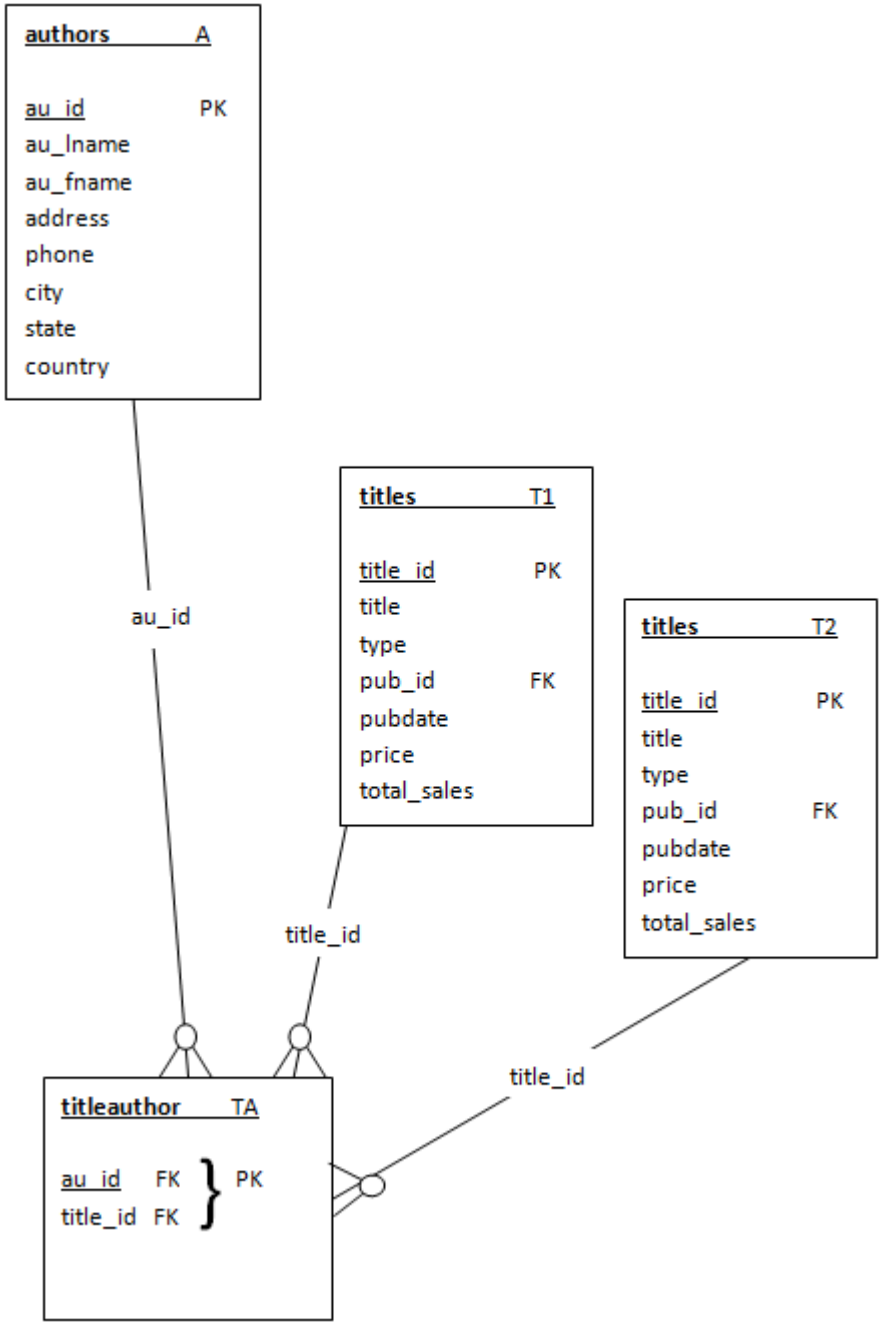
Reformulation :

Auteurs ayant publié au moins deux livres chez deux éditeurs différents.



Auteurs avec 2 livres chez 2 éditeurs différents ?

```
SELECT DISTINCT a.*
FROM authors a, titleauthor ta, titles
    t1, titles t2
WHERE t1.title_id<>t2.title_id
AND t1.title_id=ta.title_id
AND t2.title_id=ta.title_id
AND ta.au_id=a.au_id
AND t1.pub_id<>t2.pub_id
```

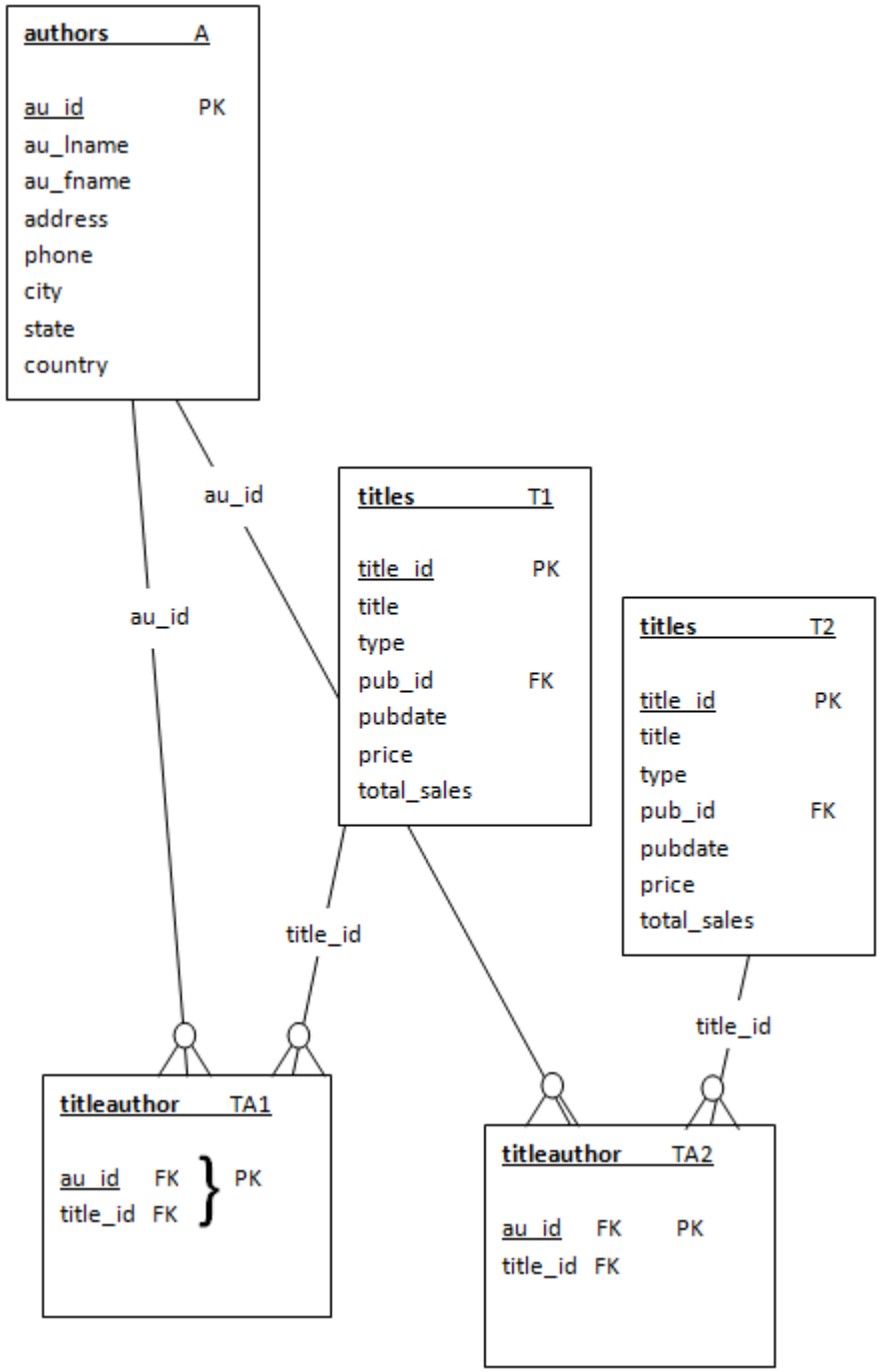


Auteurs avec 2 livres chez 2 éditeurs différents ?

```
SELECT DISTINCT a.*
FROM authors a, titleauthor ta, titles t1, titles t2
WHERE t1.title_id<>t2.title_id
AND t1.title_id=ta.title_id
AND t2.title_id=ta.title_id
AND ta.au_id=a.au_id
AND t1.pub_id<>t2.pub_id
```

Incorrect !

Par transitivité t1.title_id=t2.title_id !



Auteurs avec 2 livres chez 2 éditeurs différents ?

SELECT DISTINCT a.*

FROM authors a, titleauthor ta1, titleauthor ta2, titles t1, titles t2

WHERE **t1.title_id<>t2.title_id**

AND t1.title_id=ta1.title_id

AND t2.title_id=ta2.title_id

AND ta1.au_id=a.au_id

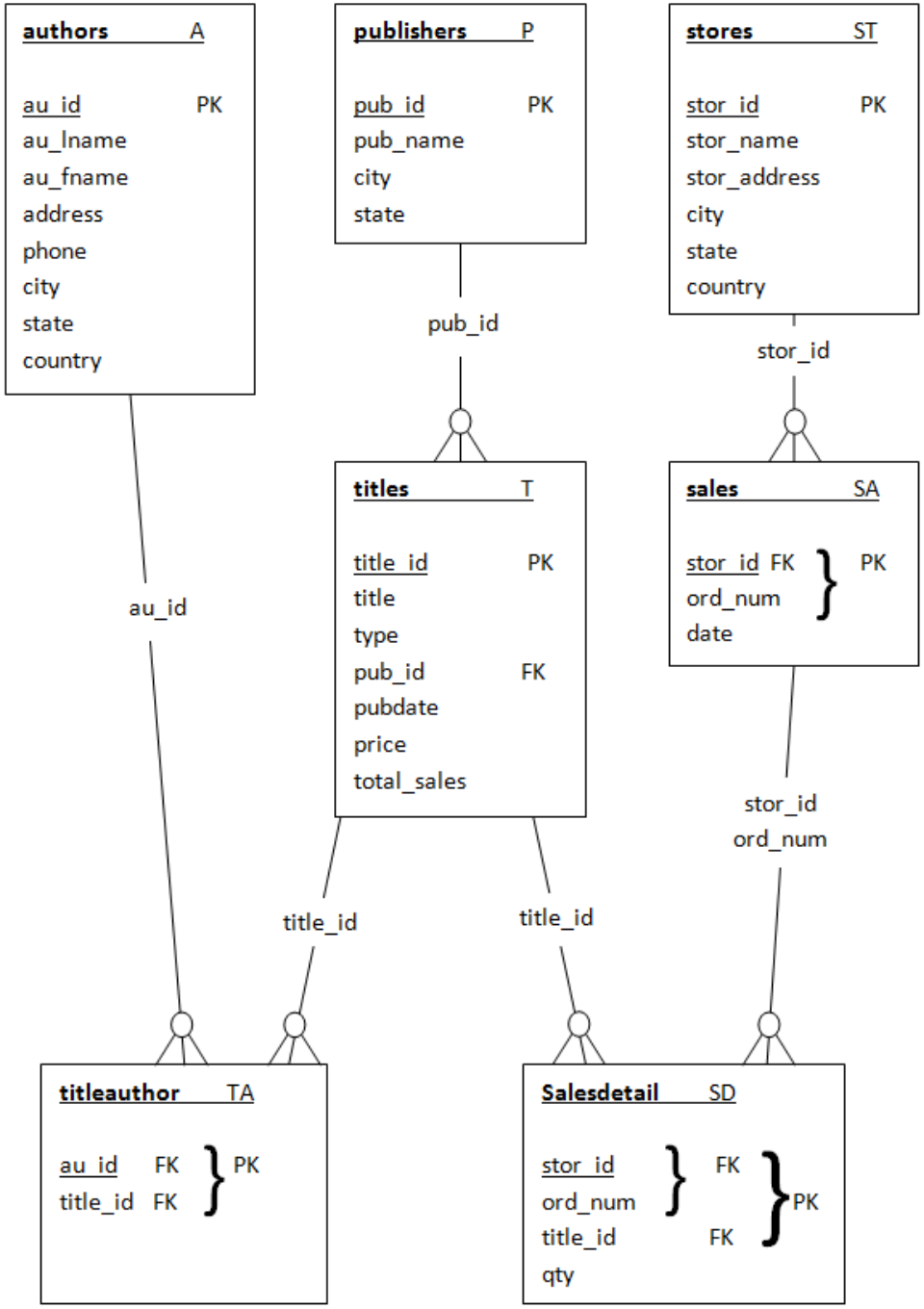
AND ta2.au_id=a.au_id

AND t1.pub_id<>t2.pub_id

CORRECT ! Est-ce que la partie en gras est nécessaire ?

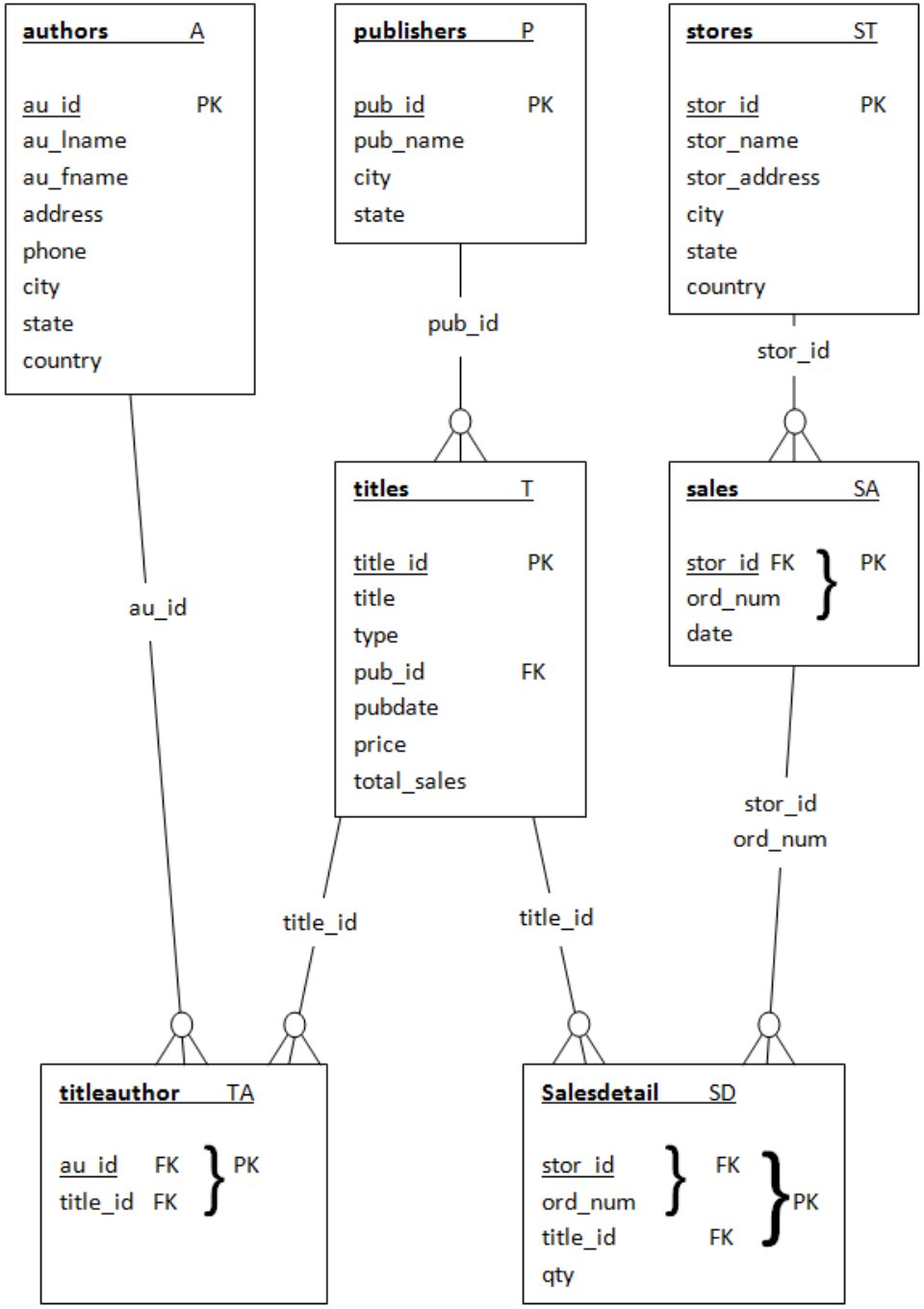
Sous SELECT

- Dans la partie condition :
  - truc = | <> | < | > | <= | >= (SELECT ...)
- où (SELECT ...) est un SELECT retournant un seul tuple d'une seule colonne.
  - On utilisera donc une fonction d'aggrégation globale sur une seule colonne



Livre le plus cher

```
SELECT t.title_id, t.title
FROM titles t
WHERE t.price = ( SELECT MAX(price)
                  FROM titles ) ;
```



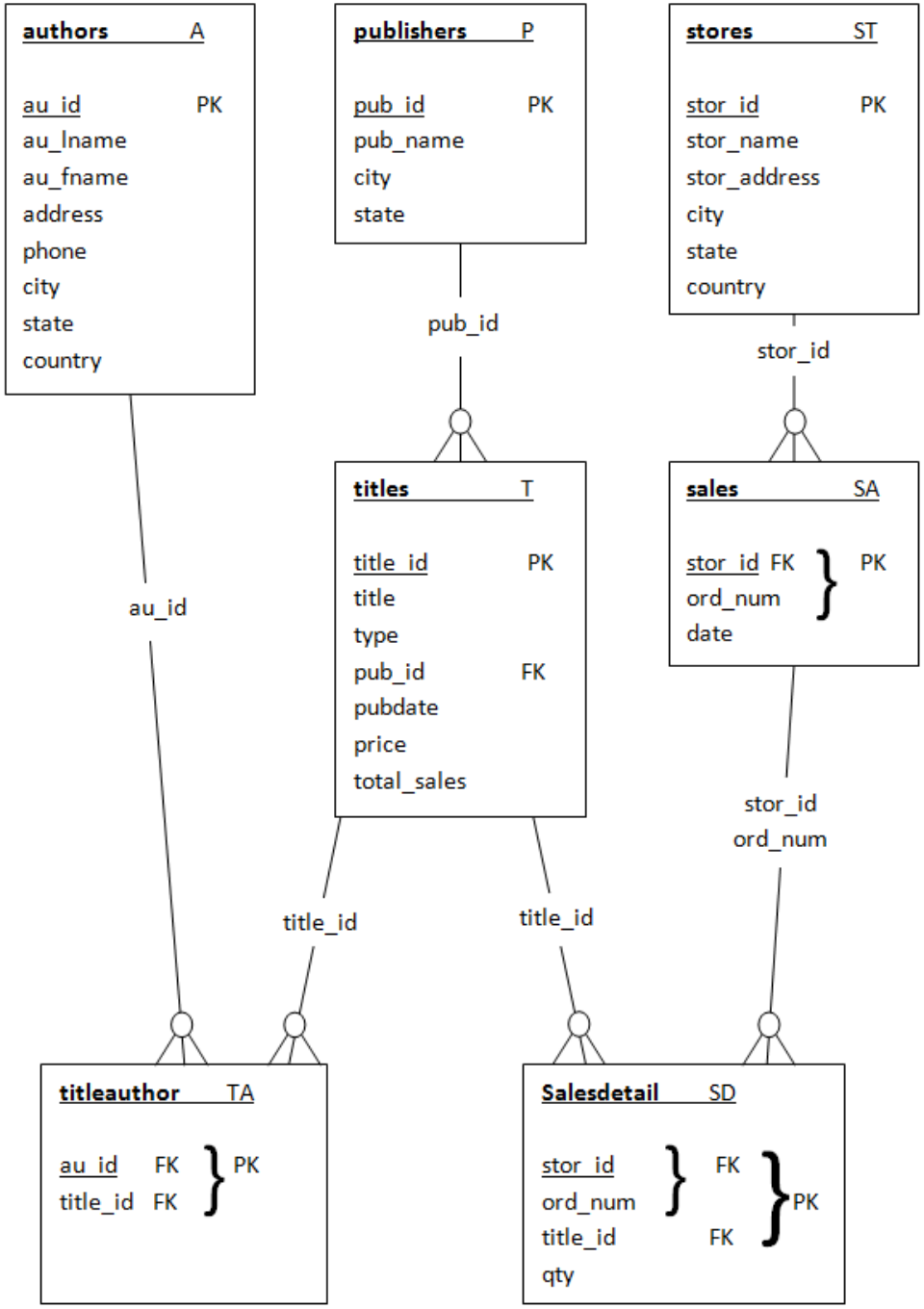
Quels sont les auteurs qui ont écrit exactement 2 livres ?

```
SELECT a.au_lname, a. au_fname
FROM authors a
WHERE 2 = ( SELECT COUNT ( * )
            FROM titleauthor ta
            WHERE ta.au_id = a.au_id ) ;
```

Sous-select fait référence au select principal

Sous SELECT

- Dans la partie condition :
  - truc IN (SELECT ...)
  - truc NOT IN (SELECT ...)
- où (SELECT ...) est un SELECT retournant une seule colonne
  - la condition teste si truc se trouve dans un des tuples du SELECT



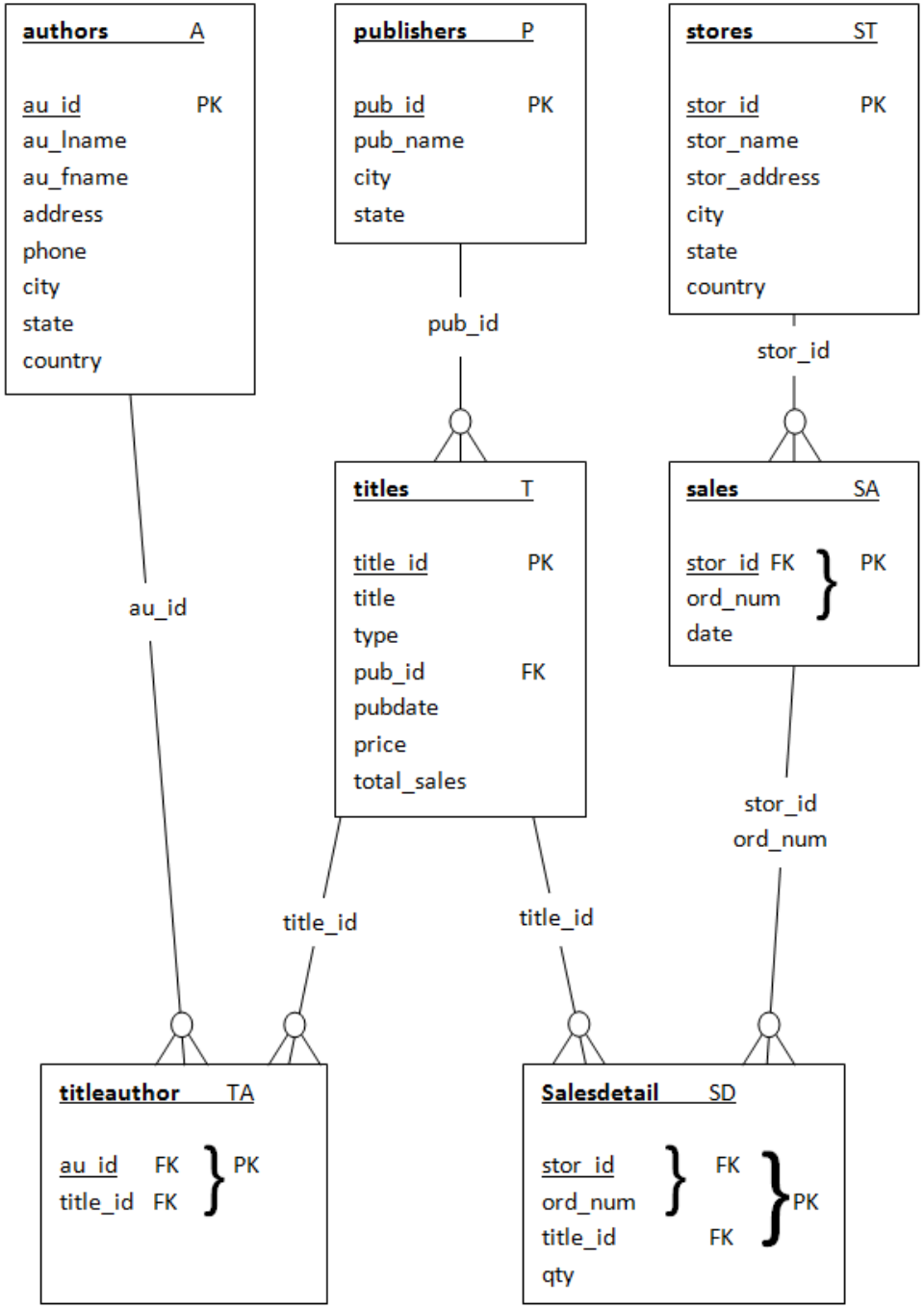
Les auteurs habitant dans le même état qu'un éditeur

SELECT DISTINCT a.au_id, a.au_lname, a.au_fname

FROM authors a, publishers p

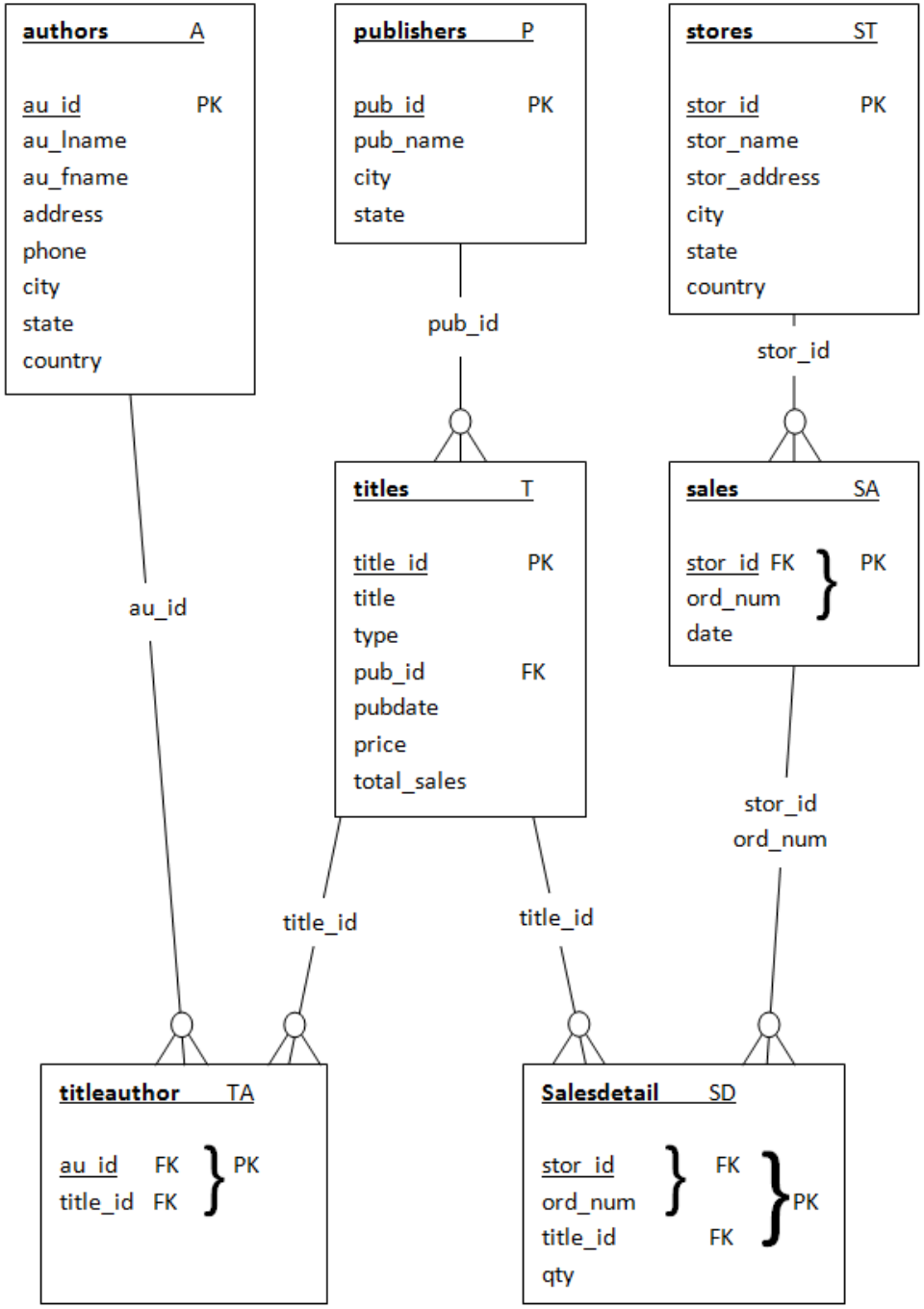
WHERE p.state=a.state

Attention produit cartésien et doublons !



Les auteurs habitant dans le même état qu'un éditeur

```
SELECT a.au_id, a.au_lname,
    a.au_fname
FROM authors a
WHERE a.state IN
    ( SELECT p.state
        FROM publishers p) ;
```



Les auteurs habitant dans le même état qu'un éditeur

SELECT p.state

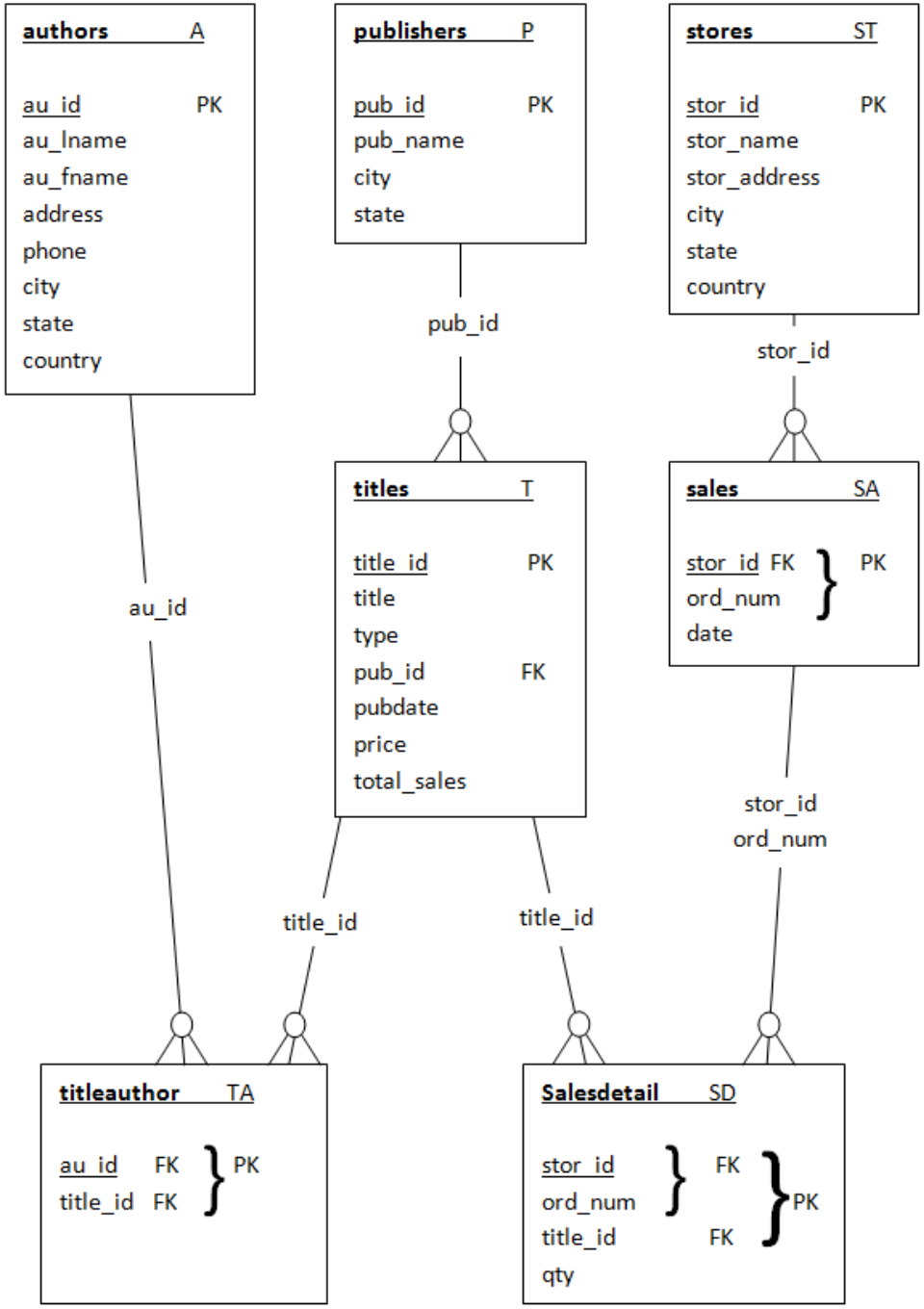
FROM publishers p;

⇒ MA, DC, CA

SELECT a.au_id, a.au_lname, a.au_fname

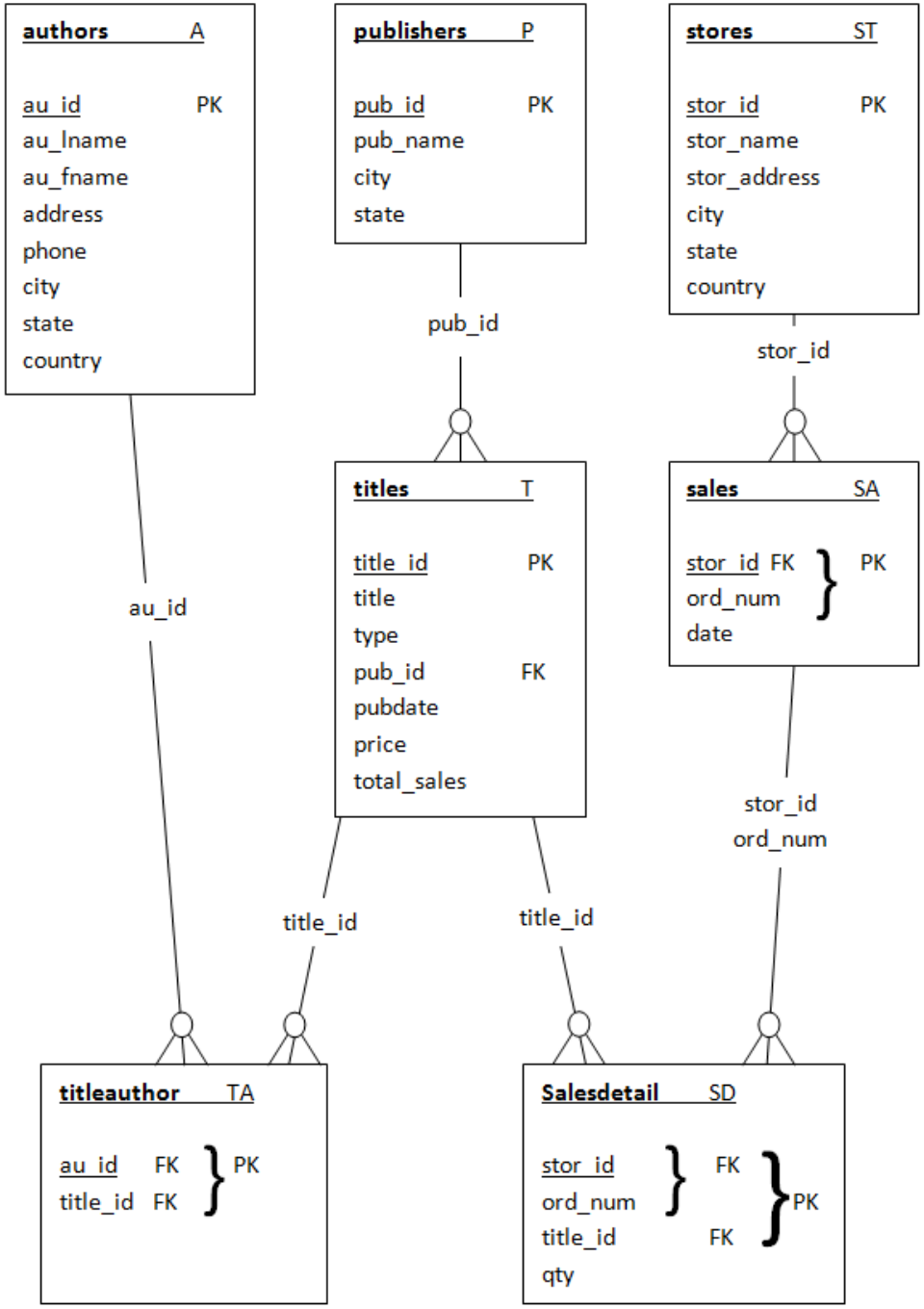
FROM authors a

WHERE a.state IN ('MA','DC','CA');



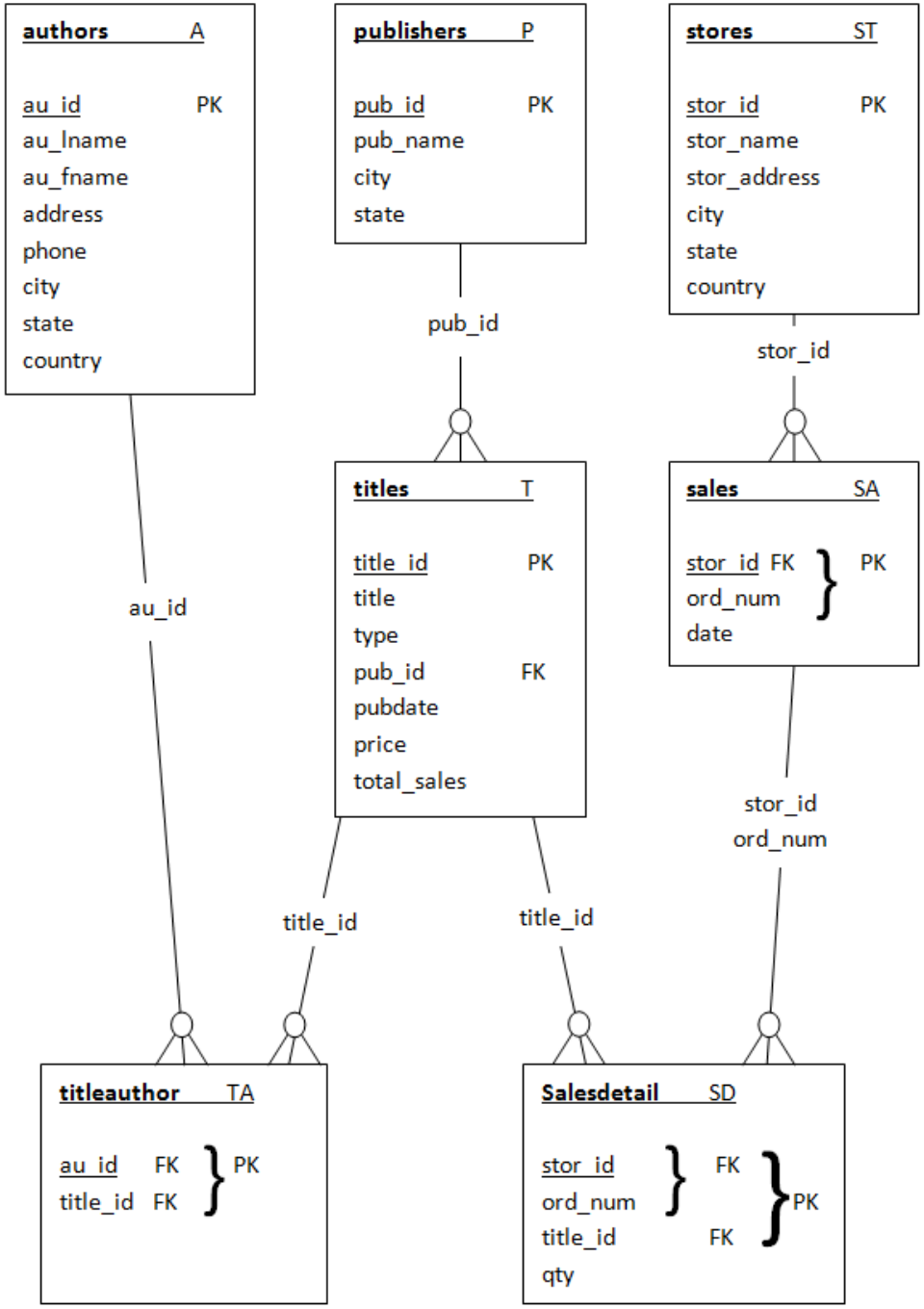
Les auteurs habitant dans le même état qu'un éditeur ayant publié un livre de business

```
SELECT DISTINCT a.au_id,
    a.au_lname, a.au_fname
FROM authors a, publishers p,
    titles t
WHERE p.pub_id = t.pub_id
  AND a.state = p.state
  AND t.type = 'business' ;
```



Les auteurs habitant dans le même état qu'un éditeur ayant publié un livre de business

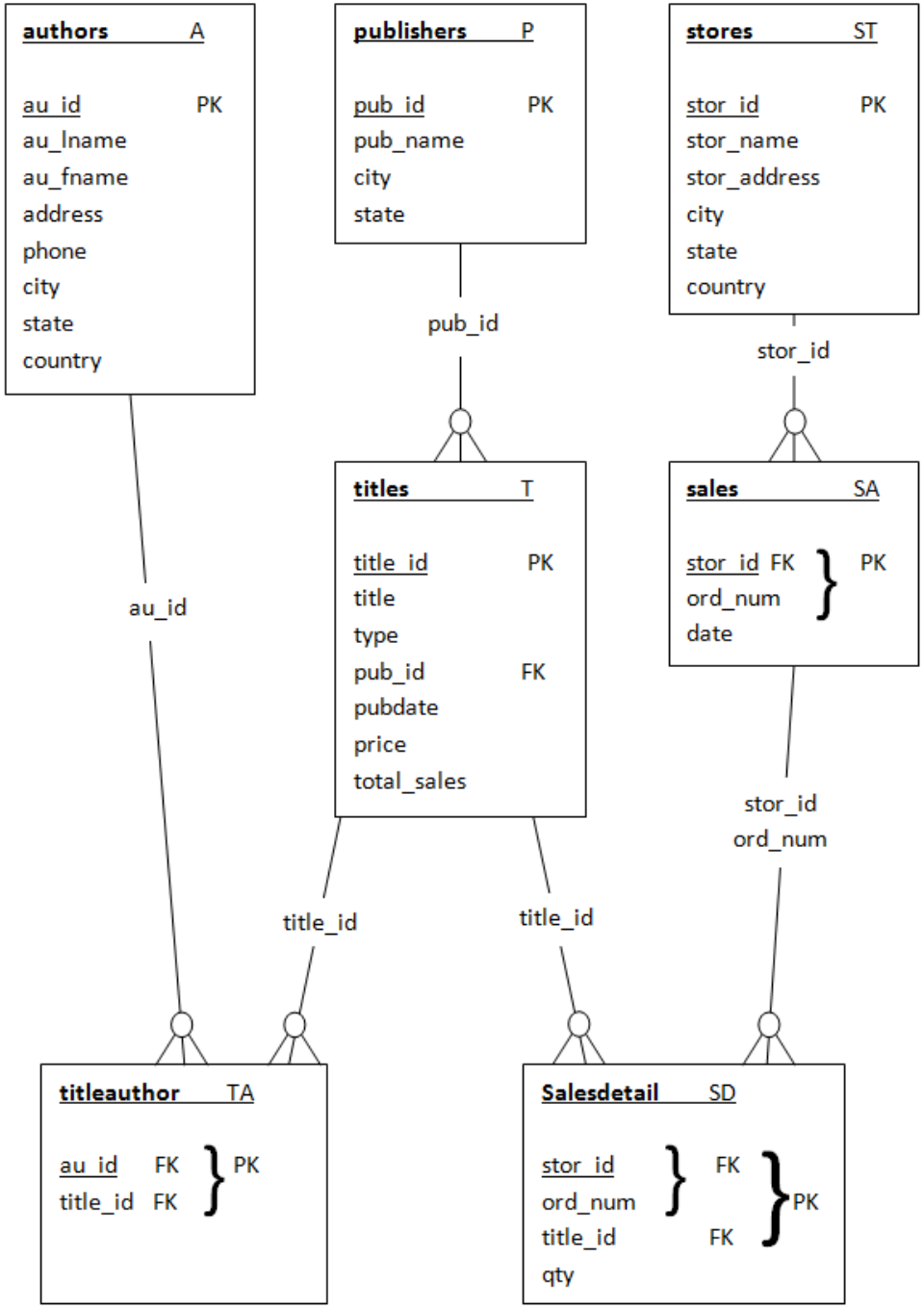
```
SELECT a.au_id, a.au_lname,
    a.au_fname
FROM authors a
WHERE a.state IN ( SELECT p.state
        FROM publishers p, titles t
        WHERE p.pub_id = t.pub_id
         AND t.type = 'business' ) ;
```



Les auteurs habitant dans le même état qu'un éditeur ayant publié un livre de business

```
SELECT a.au_id, a.au_lname, a.au_fname
FROM authors a
WHERE a.state IN (
    SELECT p.state
    FROM publishers p
    WHERE p.pub_id IN (
        SELECT t.pub_id
        FROM titles t
        WHERE t.type = 'business' )) ;
```

IN et jointures sont interchangeables



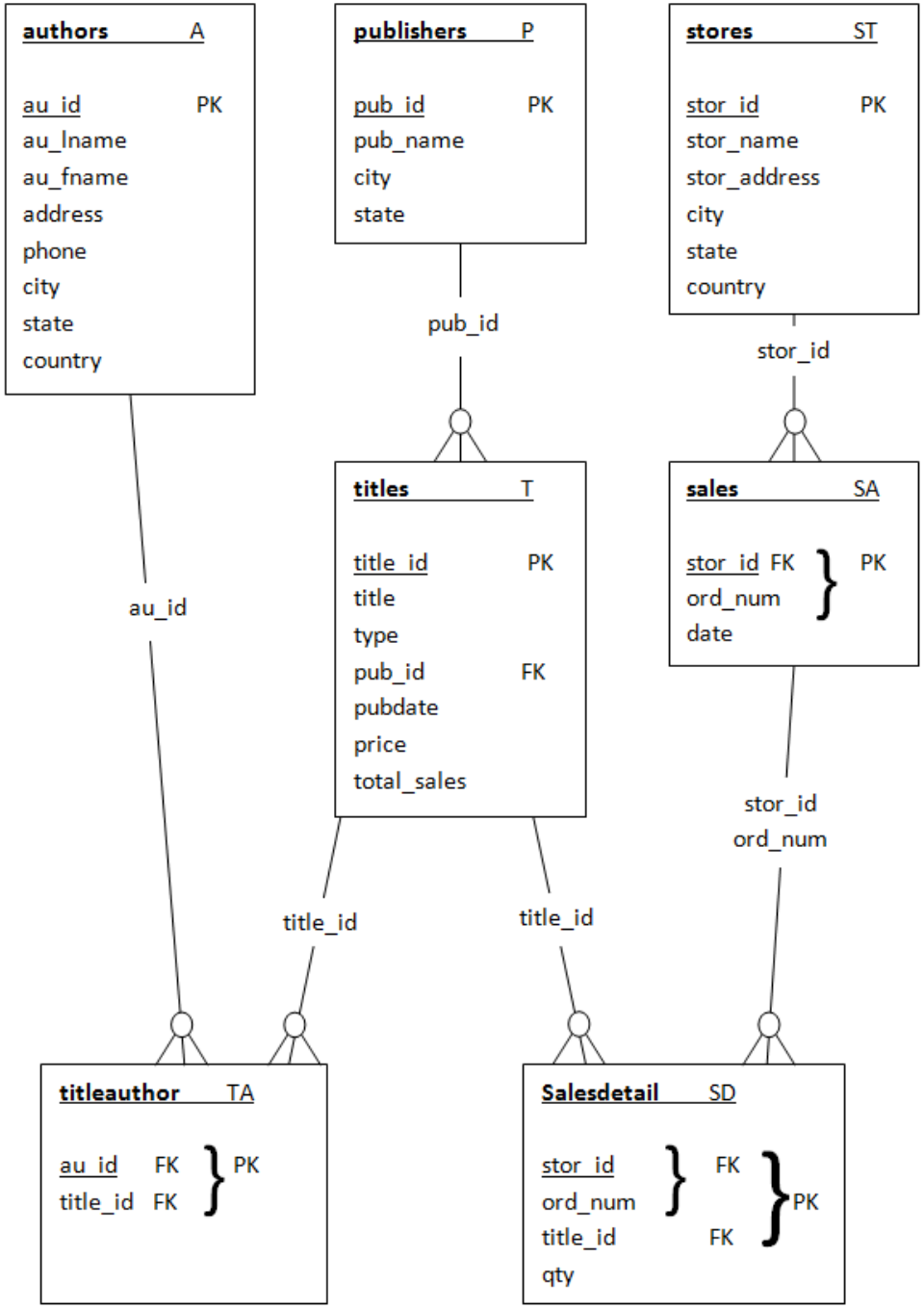
Les auteurs habitant dans un état sans éditeur

```
SELECT a.au_id, a.au_lname,
    a.au_fname
FROM authors a
WHERE a.state NOT IN (
            SELECT p.state
            FROM publishers p) ;
```

Impossible avec des jointures !

Sous SELECT

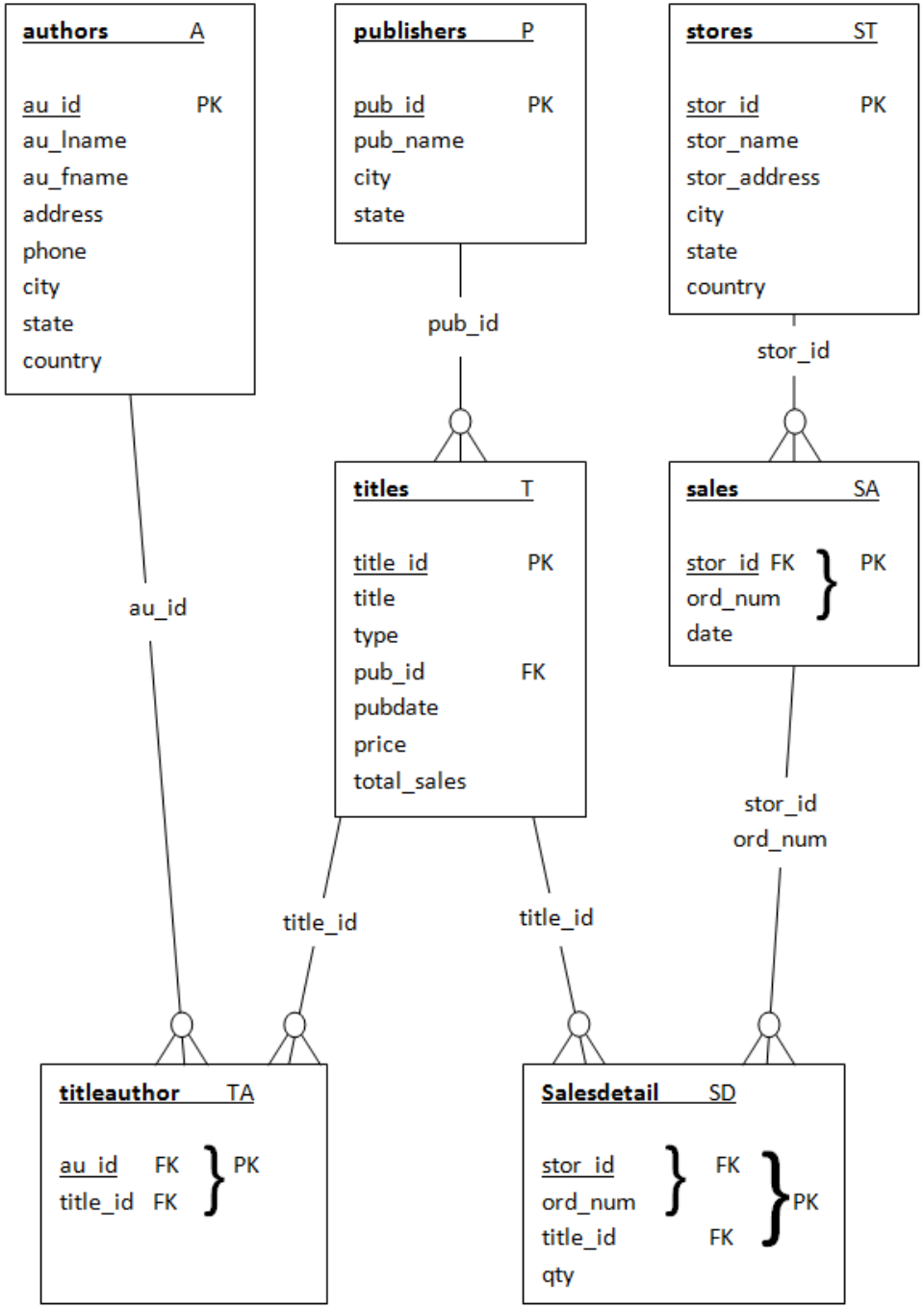
- Dans la partie condition :
  - truc = | <> | > | < | <= | >= ANY (SELECT ...)
  - truc = | <> | > | < | <= | >= ALL (SELECT ...)
- où (SELECT ...) est un SELECT retournant une seule colonne
  - la condition teste si l'opérateur est vrai pour au moins un (ANY) ou tous (ALL) les éléments du sous-select.



Les auteurs habitant dans le même état qu'un éditeur

```
SELECT a.au_id, a.au_lname,
    a.au_fname
FROM authors a
WHERE a.state = ANY (
        SELECT p.state
        FROM publishers p) ;
```

IN d'un sous-select est équivalent à = ANY



Livre le plus cher

```
SELECT t1.title_id, t1.title
FROM titles t1
WHERE t1.price =
        ( SELECT MAX(t2.price)
        FROM titles t2) ;
```

Résultat : PC1035

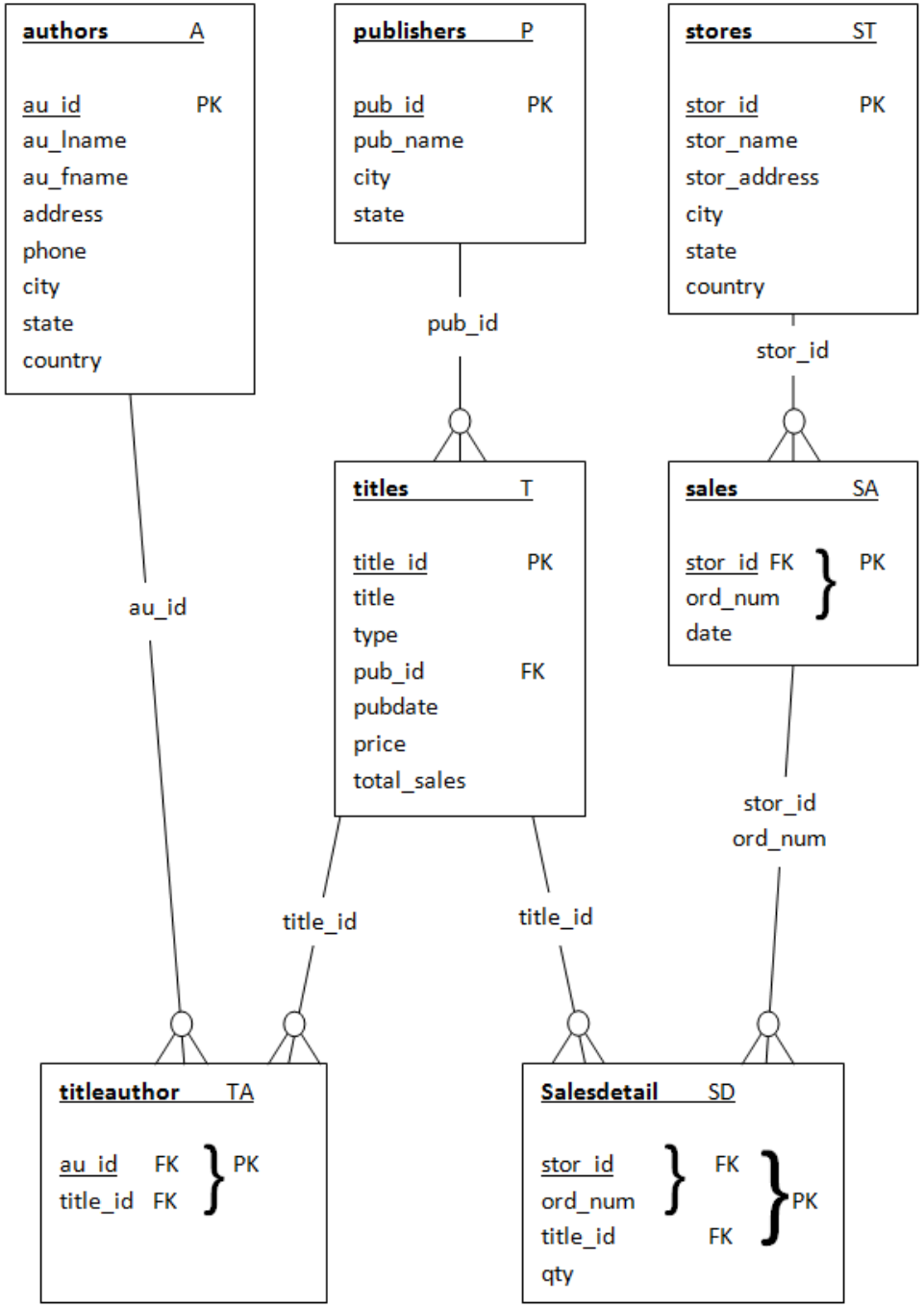
```
SELECT t1.title_id, t1.title
FROM titles t1
WHERE t1.price >=
        ALL ( SELECT t2.price
                FROM titles t2) ;
```

Résultat : ∅

Pourquoi ? null ...

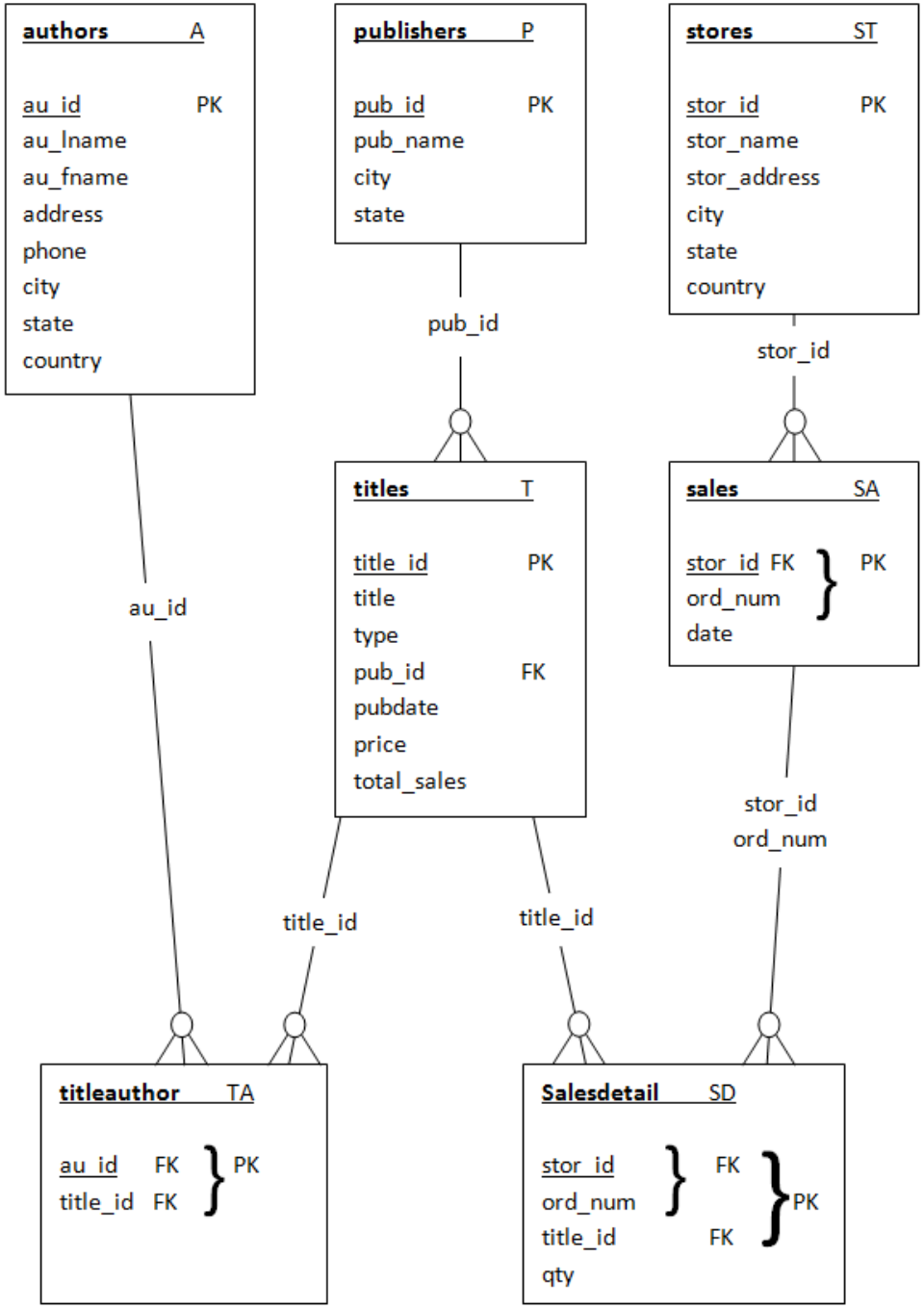
Sous SELECT

- Dans la partie condition :
  - EXISTS (SELECT ...)
  - NOT EXISTS (SELECT ...)
- où (SELECT ...) est un SELECT sans condition particulière
  - la condition vérifie si le sous-select est vide ou pas.
  - comme la présence d'un tuple est suffisante, inutile de SELECT des colonnes en particulier : SELECT *



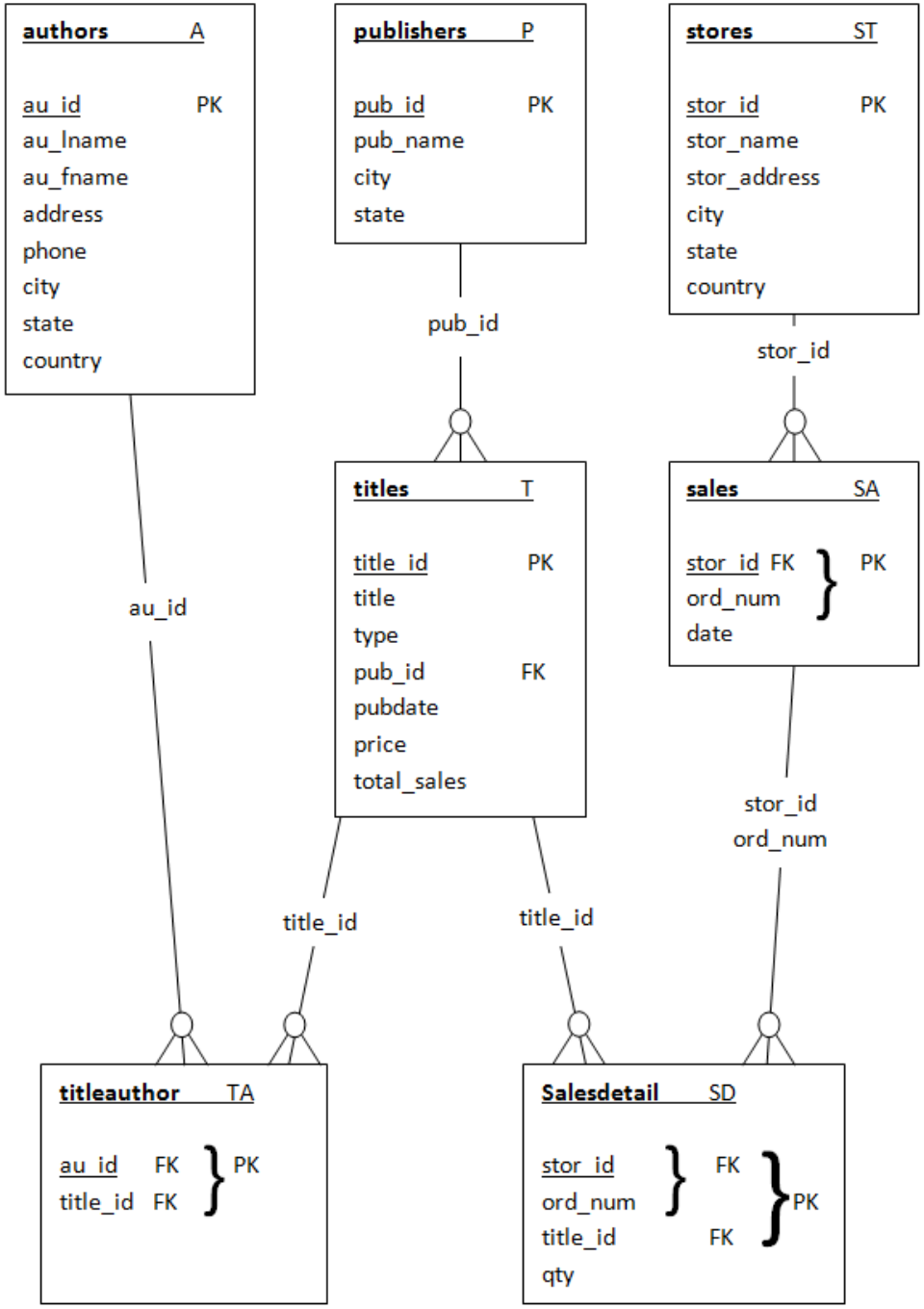
Les auteurs habitant dans le même état qu'un éditeur

```
SELECT a.au_lname,
    a.au_fname
FROM authors a
WHERE EXISTS (
    SELECT *
    FROM publishers p
    WHERE p.state = a.state)
```



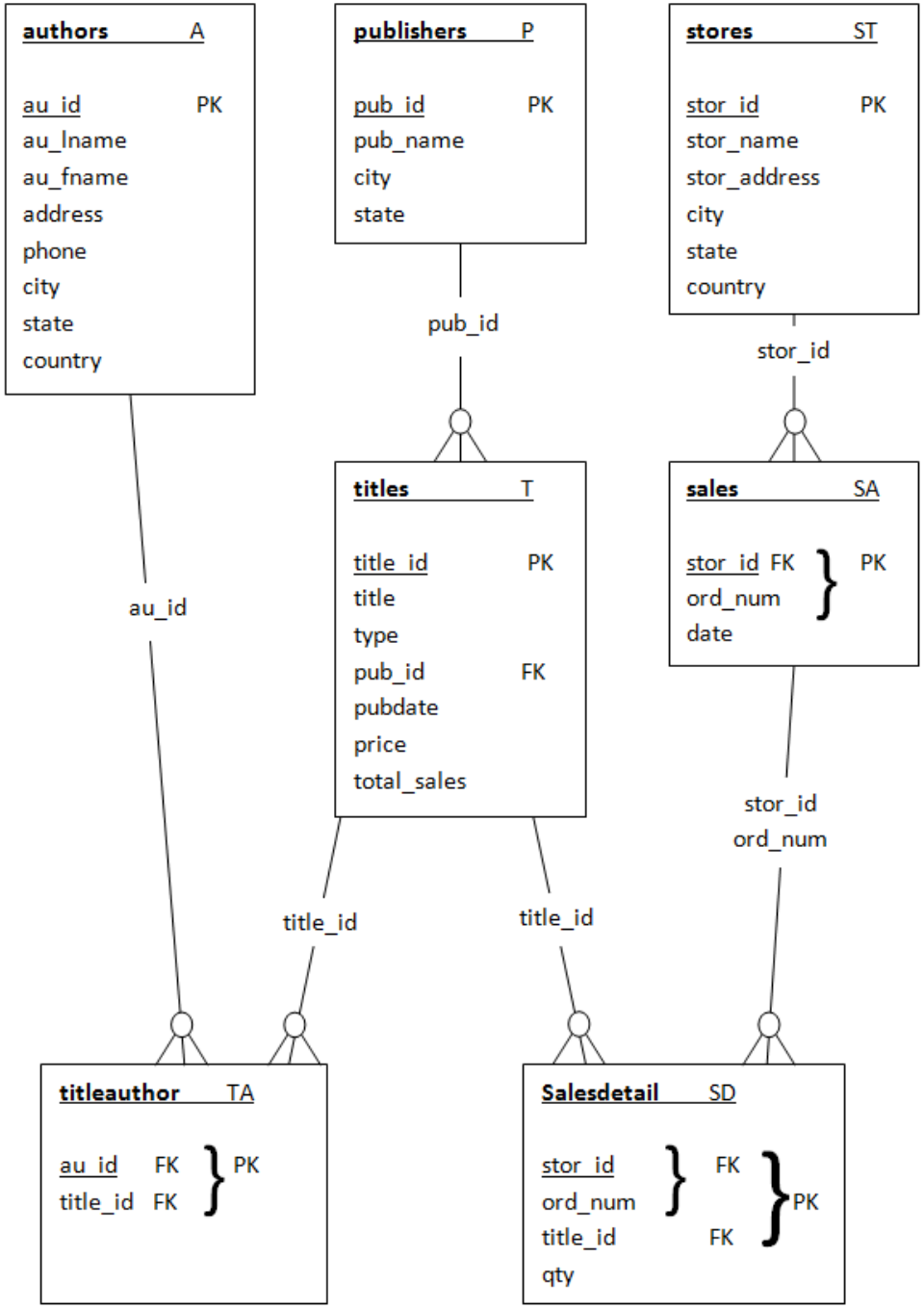
Les auteurs habitant dans un état sans éditeur

```
SELECT a.au_lname,
    a.au_fname
FROM authors a
WHERE NOT EXISTS (
    SELECT *
    FROM publishers p
    WHERE p.state = a.state)
```



Les magasins qui vendent tous les livres édités par Algodata ?

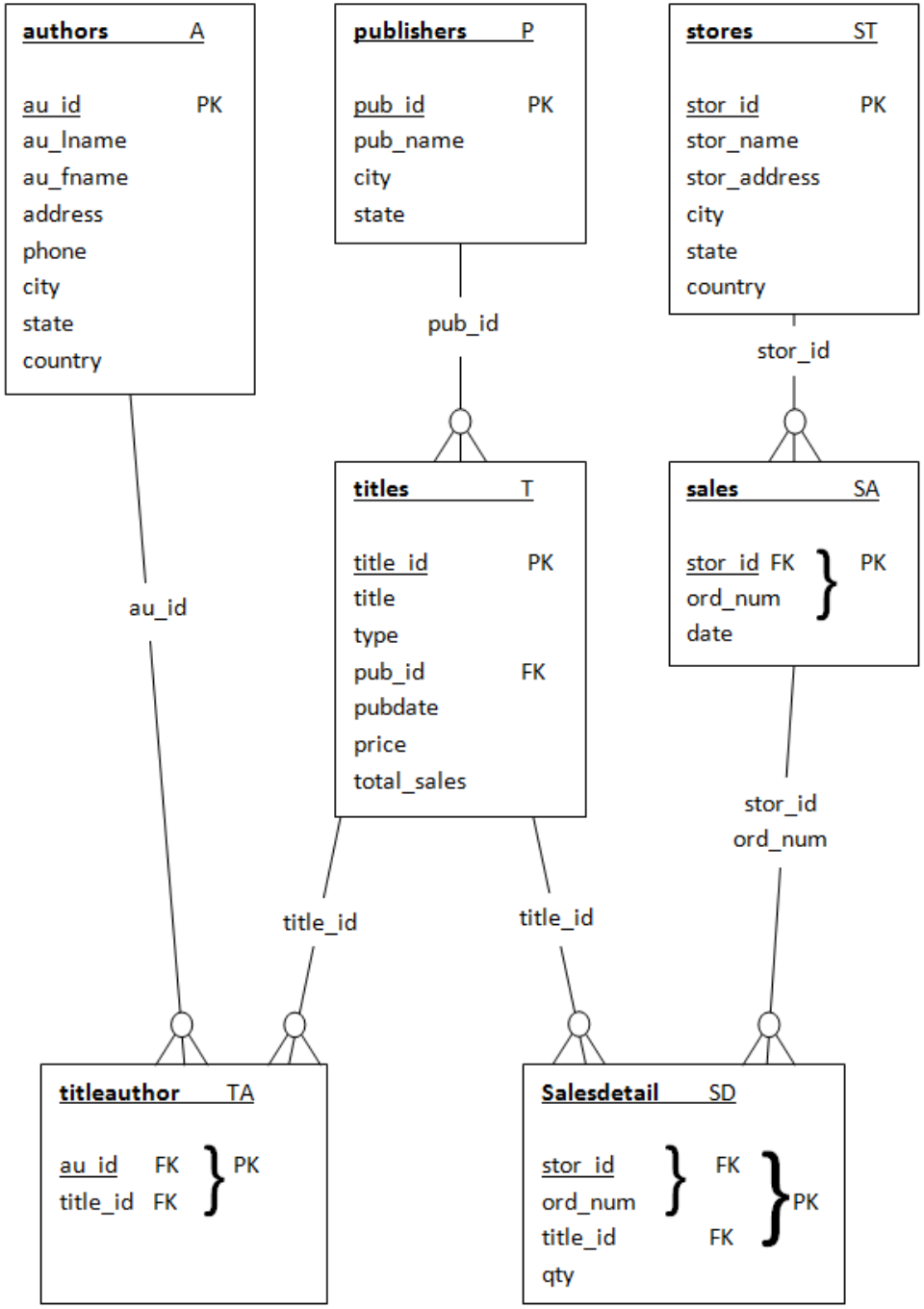
On ne peut pas traduire directement en SQL !



Les magasins qui vendent tous les livres édités par Algodata ?

Reformulation :

Les magasins tels qu'il n'existe pas de livre édité par Algodata qui n'y soit pas vendu

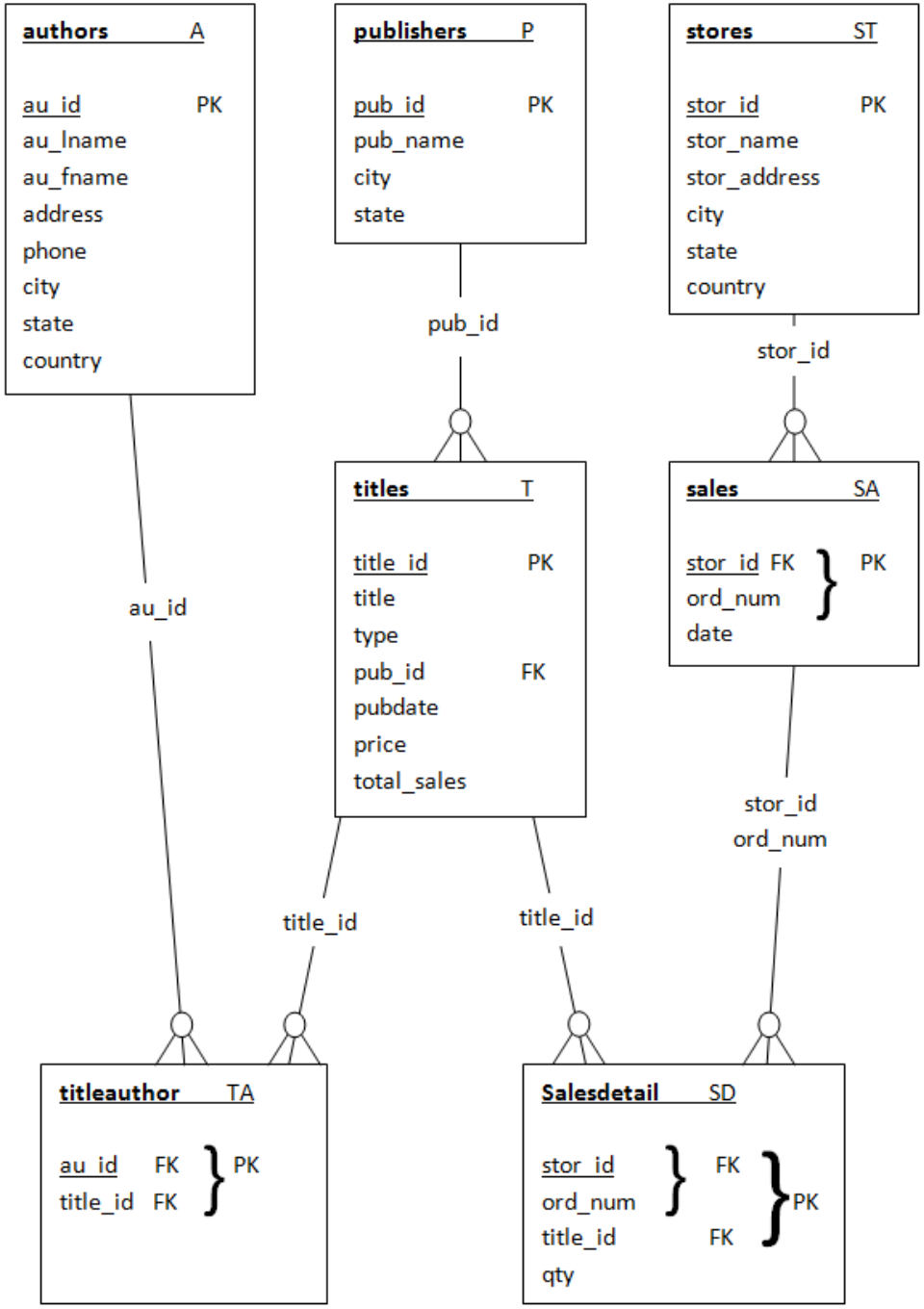


Les magasins tels qu'il n'existe pas de livre édité par Algodata qui n'y soit pas vendu

```
SELECT st.stor_id, st.stor_name
FROM stores st
WHERE NOT EXISTS (
    SELECT *
    FROM titles t, publishers p
    WHERE t.pub_id = p.pub_id
    AND p.pub_name LIKE 'Algodata%'
    AND NOT EXISTS (
        SELECT *
        FROM salesdetail sd
        WHERE sd.title_id = t.title_id
        AND sd.stor_id = st.stor_id ));
```

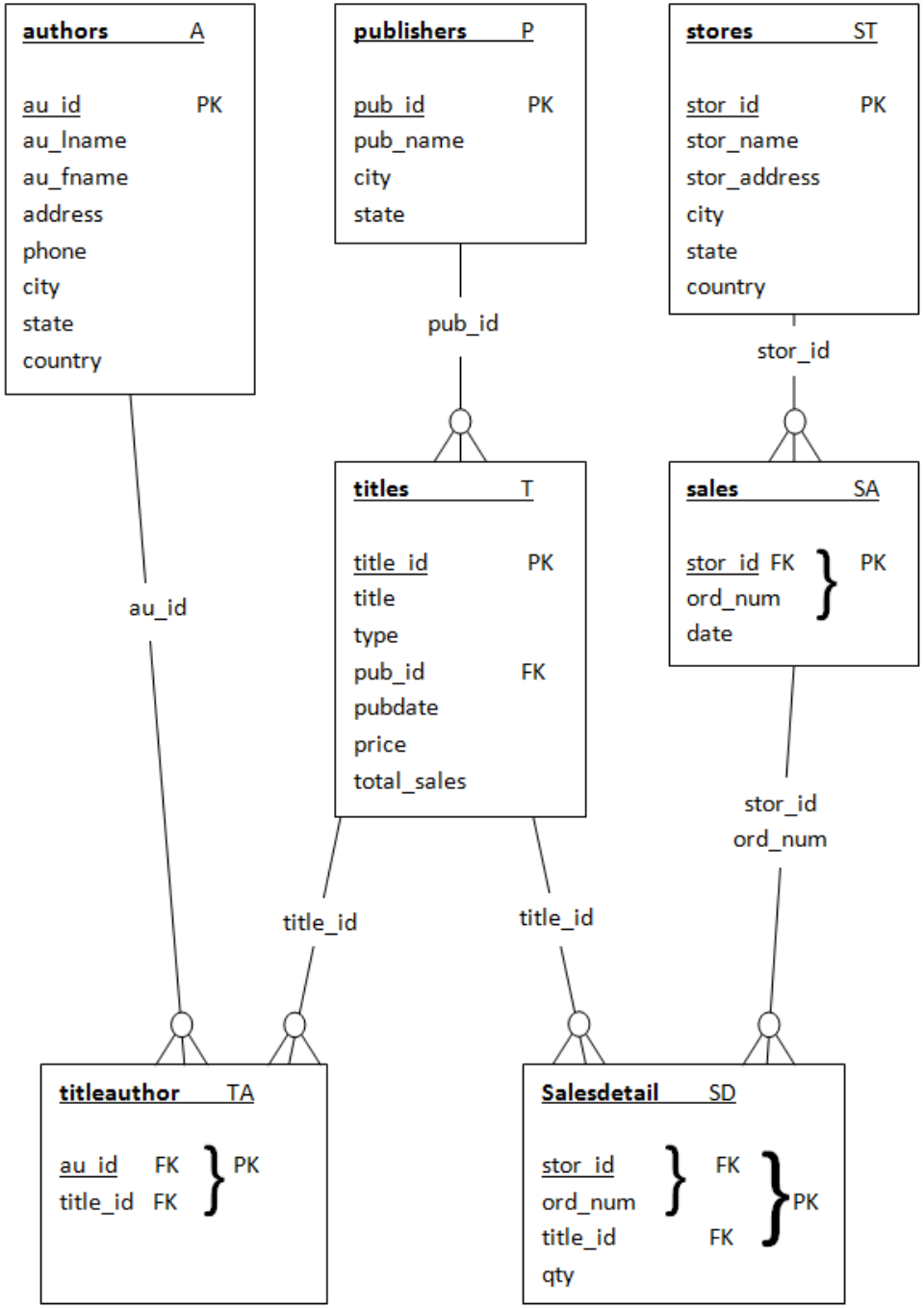
Union, Intersection, Difference

- (SELECT ...) UNION (SELECT ...)
  - Les deux selects doivent produire le même nombre de colonnes de types compatibles.
  - Effectue l'union ensembliste
- (SELECT ...) INTERSECT (SELECT ...)
  - Les deux selects doivent produire le même nombre de colonnes de types compatibles.
  - Effectue l'intersection ensembliste
- (SELECT ...) EXCEPT (SELECT ...)
  - Les deux selects doivent produire le même nombre de colonnes de types compatibles.
  - Effectue la différence



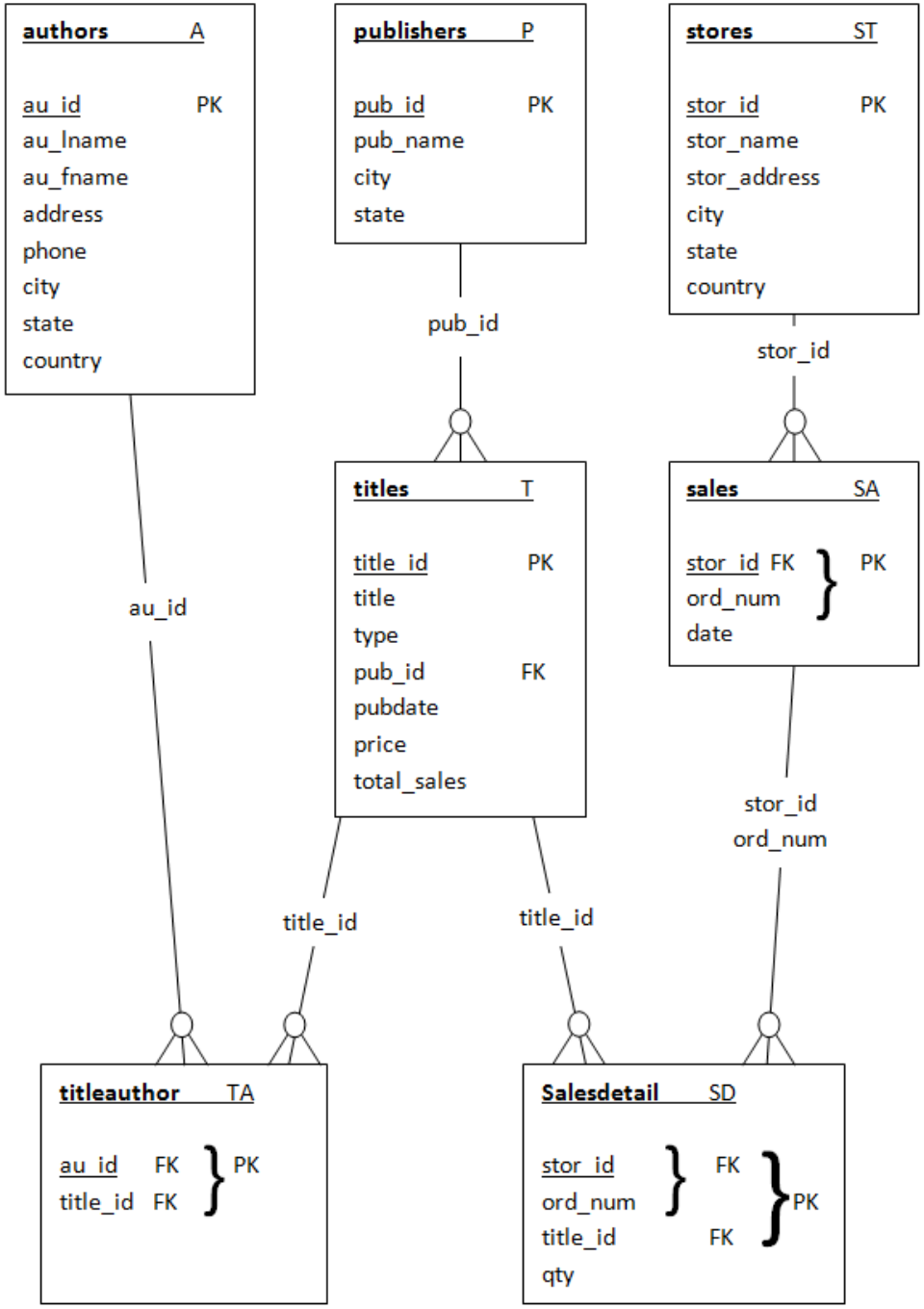
Les villes dans lesquelles il y a un auteur et/ou un éditeur ?

```
(SELECT DISTINCT a.city
FROM authors a)
UNION
(SELECT DISTINCT p.city
FROM publishers p) ;
```



Les villes dans lesquelles il y a un auteur et un éditeur ?

(SELECT DISTINCT a.city
FROM authors a)
INTERSECT
(SELECT DISTINCT p.city
FROM publishers p) ;



Les villes dans lesquelles il y a un auteur mais pas d'éditeur ?

```
(SELECT DISTINCT a.city
FROM authors a)
EXCEPT
(SELECT DISTINCT p.city
FROM publishers p) ;
```

Outer Join

Dans la partie FROM d'un SELECT

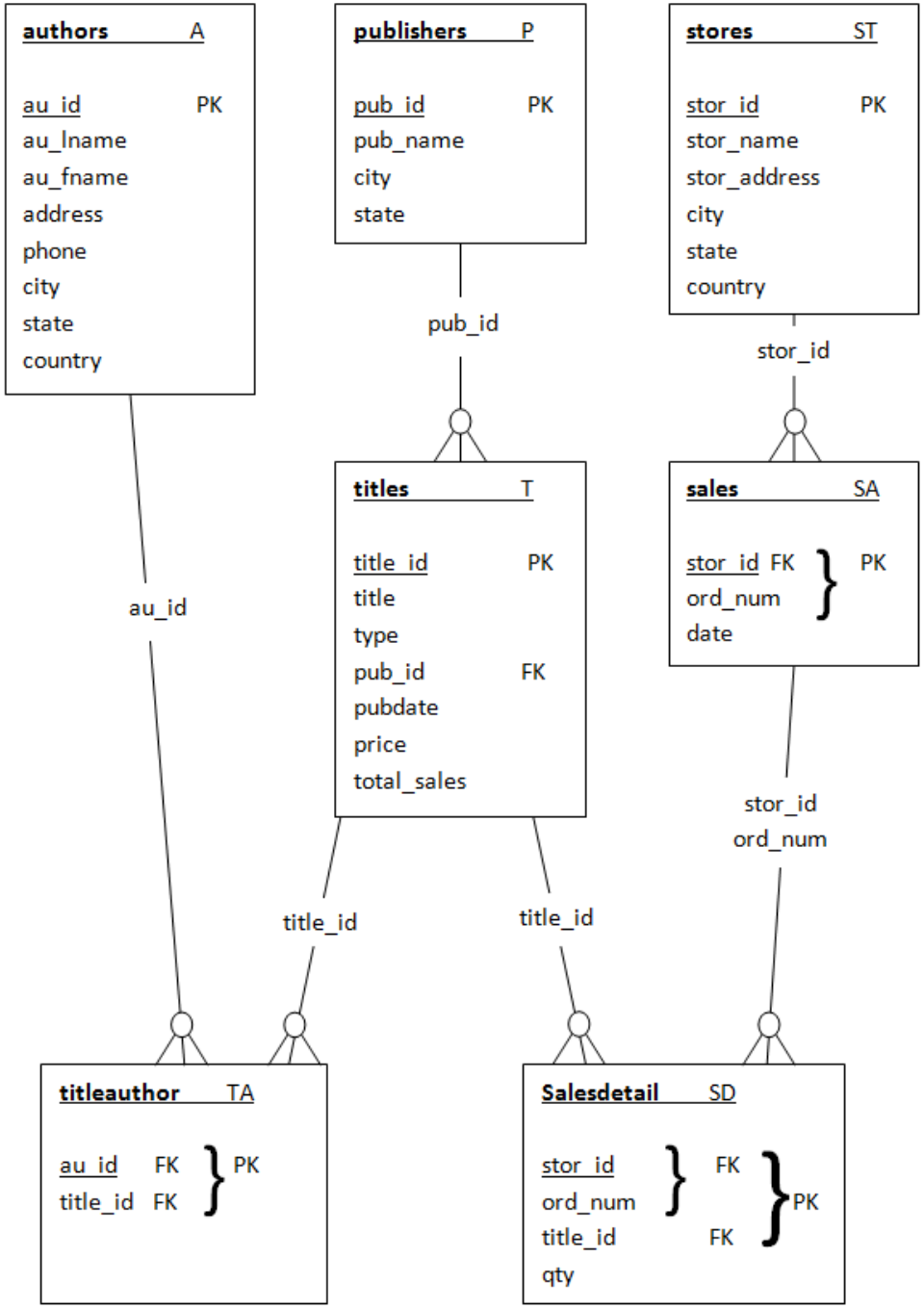
- tableGauche INNER JOIN tableDroite ON condition

  => FROM tableGauche, tableDroite WHERE condition
- tableGauche LEFT OUTER JOIN tableDroite ON condition

  retourne tous les champs de tableGauche, joint avec tableDroite quand c'est possible, null au sinon
- tableGauche RIGHT OUTER JOIN tableDroite ON champ

  retourne tous les champs de tableDroite, joint avec tableGauche quand c'est possible, null au sinon
- tableGauche FULL OUTER JOIN tableDroite ON champ

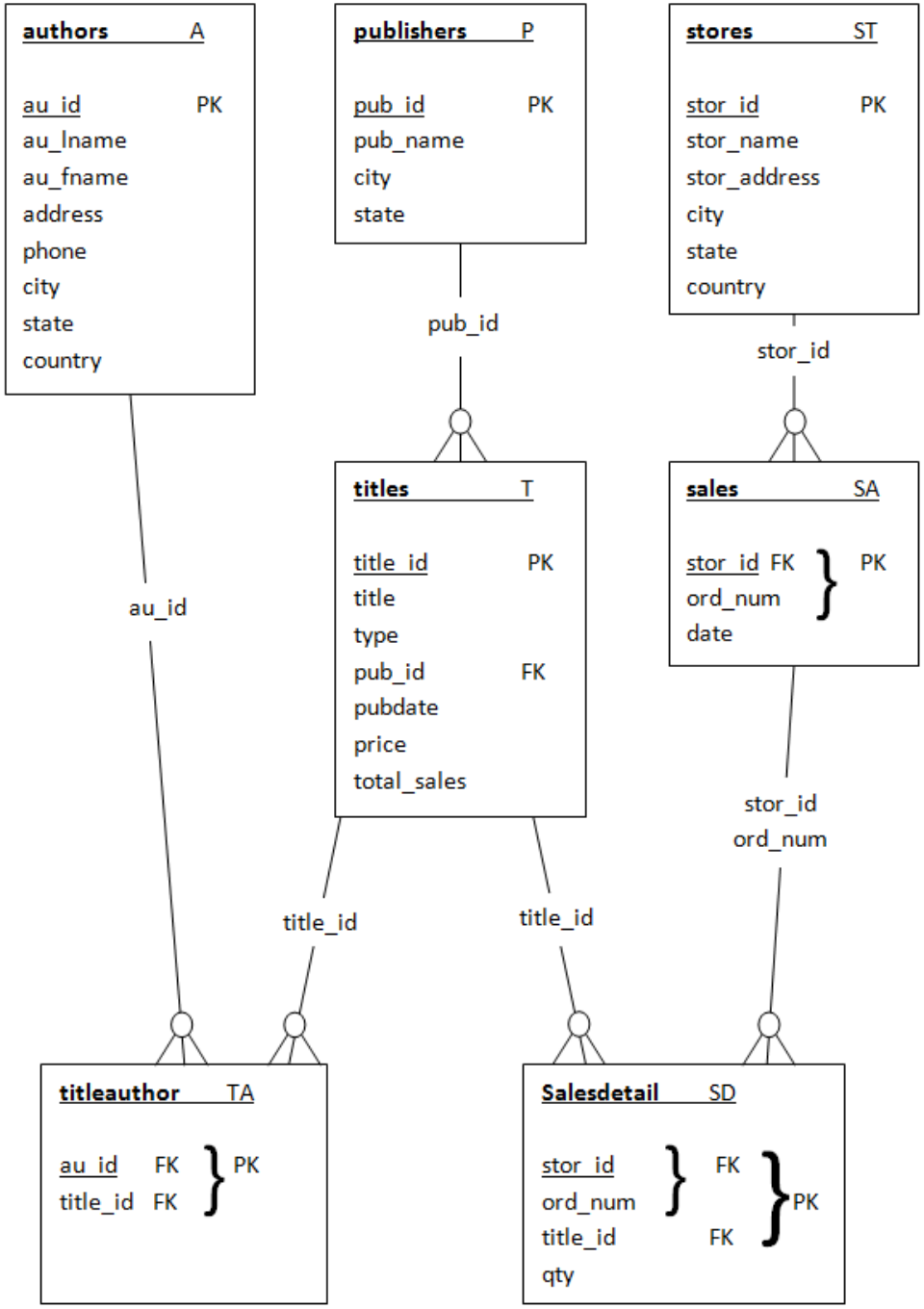
  retourne tous les champs de tableDroite et de tableGauche, joint avec l'autre table quand c'est possible, null au sinon



Tous les livres avec leurs quantités vendues

```
SELECT t.title, SUM(sd.qty)
FROM titles t, salesdetail sd
WHERE t.title_id=sd.title_id
GROUP BY t.title_id;
```

	title character varying(80)	sum bigint
1	Sushi, Anyone?	4095
2	Cooking with Computers: Surreptitious Balance Sheets	3876
3	Computer Phobic and Non-Phobic Individuals: Behavior Variations	375
4	The Gourmet Microwave	22246
5	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean	375
6	Emotional Security: A New Algorithm	3336
7	Fifty Years in Buckingham Palace Kitchens	15096
8	You Can Combat Computer Stress!	15722
9	Straight Talk About Computers	4095
10	Prolonged Data Deprivation: Four Case Studies	4072
11	But Is It User Friendly?	8780
12	Secrets of Silicon Valley	2095
13	The Busy Executive's Database Guide	4095
14	Silicon Valley Gastronomic Treats	2032
15	Life Without Fear	111
16	Is Anger the Enemy?	2045



Tous les livres avec leurs quantités vendues

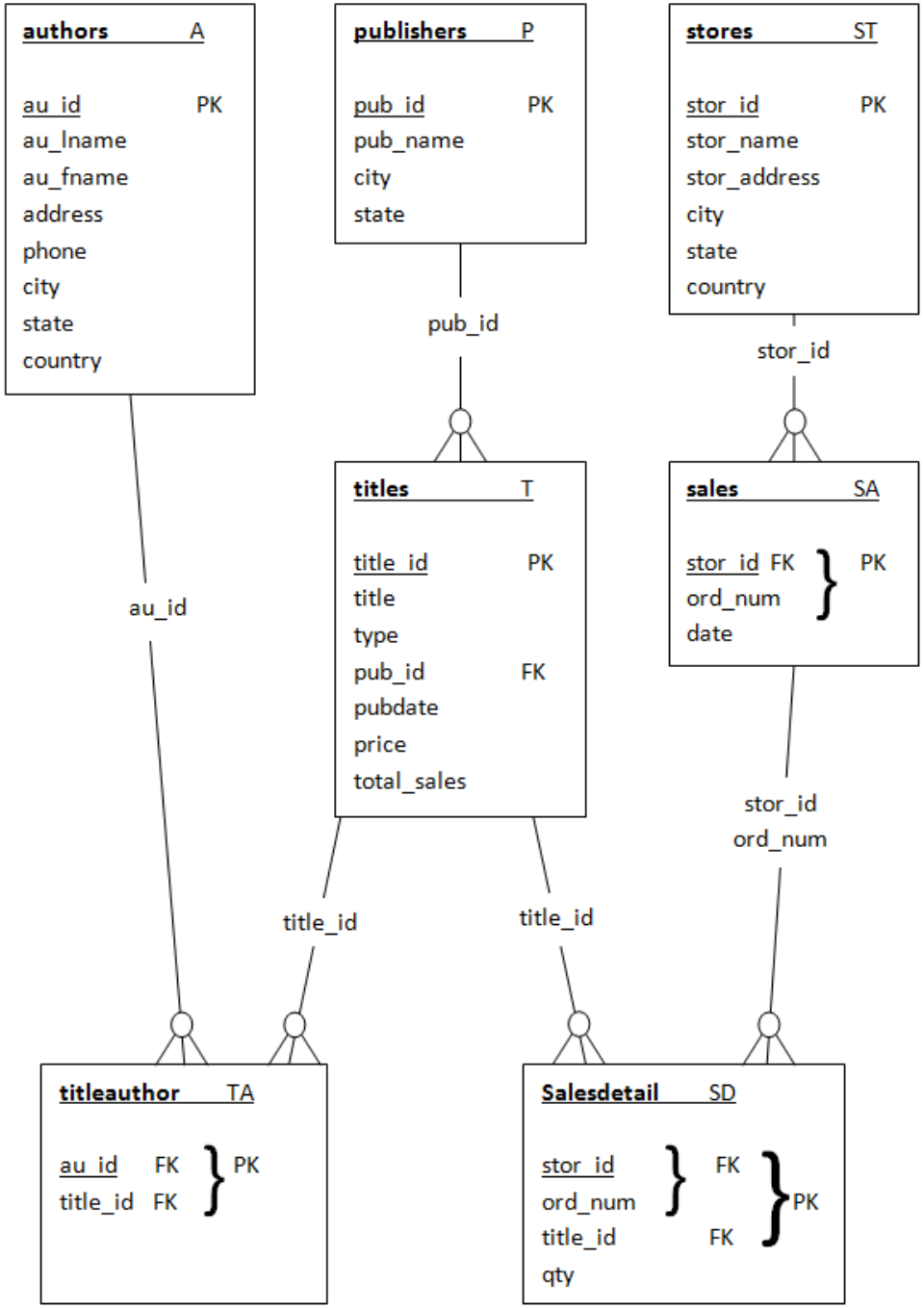
SELECT count(t.title_id)

FROM titles t;

Retourne 18 !

Où sont les 2 livres manquants ?

Pas de vente et donc pas de jointure...



Tous les livres avec leurs quantités vendues

```
SELECT t.title, SUM(sd.qty)
FROM titles t LEFT OUTER JOIN salesdetail sd
ON t.title_id=sd.title_id
GROUP BY t.title_id;
```

	title character varying(80)	sum bigint
1	The Psychology of Computer Cooking	
2	Sushi, Anyone?	4095
3	Cooking with Computers: Surreptitious Balance Sheets	3876
4	Computer Phobic and Non-Phobic Individuals: Behavior Variations	375
5	The Gourmet Microwave	22246
6	Emotional Security: A New Algorithm	3336
7	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean	375
8	Fifty Years in Buckingham Palace Kitchens	15096
9	You Can Combat Computer Stress!	15722
10	Straight Talk About Computers	4095
11	Secrets of Silicon Valley	2095
12	Prolonged Data Deprivation: Four Case Studies	4072
13	But Is It User Friendly?	8780
14	The Busy Executive's Database Guide	4095
15	Net Etiquette	
16	Silicon Valley Gastronomic Treats	2032
17	Life Without Fear	111
18	Is Anger the Enemy?	2045