

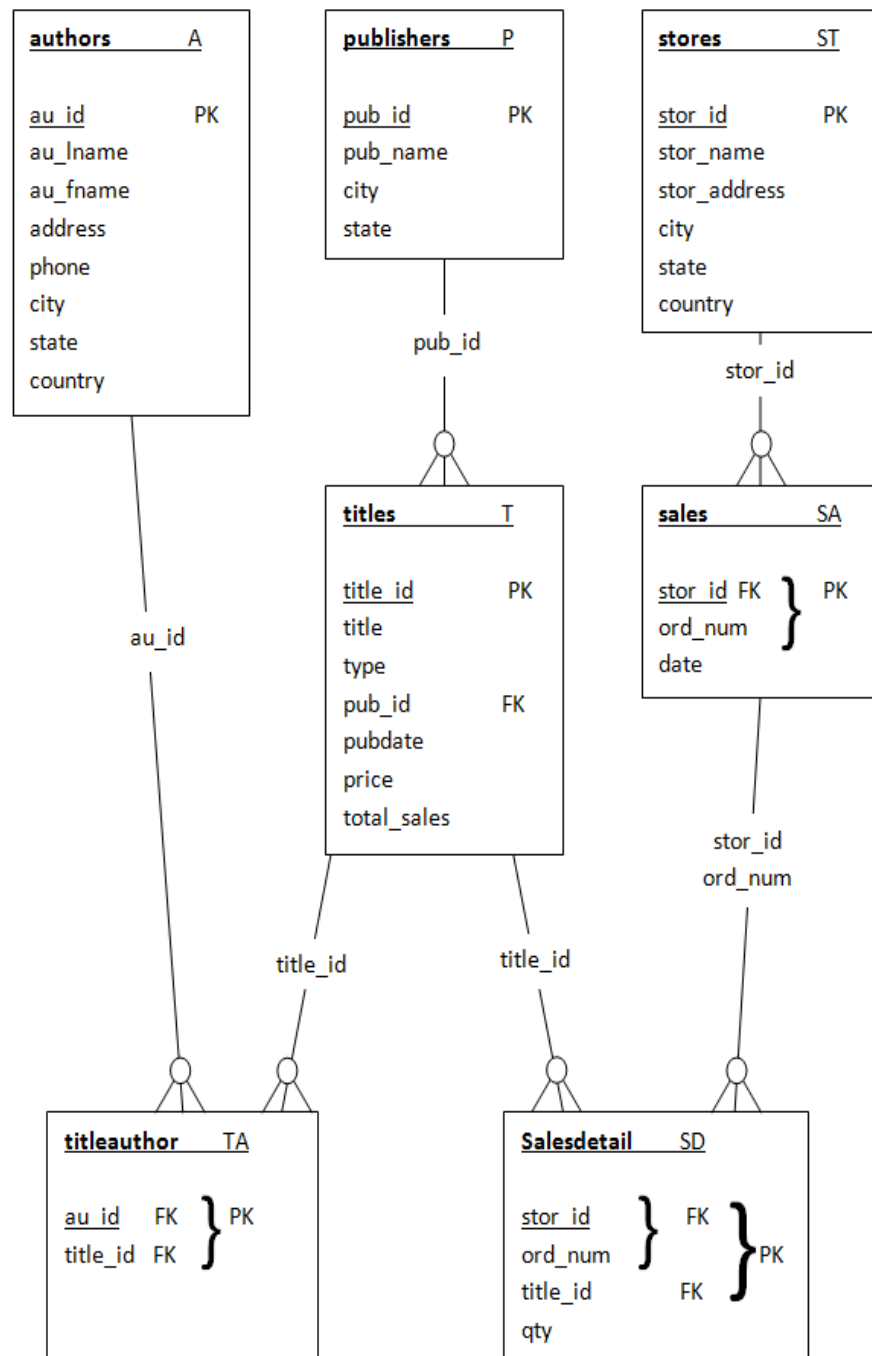
Magasins ayant vendus tous les livres que le magasin Barnum a vendu

Reformulations :

1. Les magasins pour lesquels il n'existe pas de livre vendu par Barnum que ce magasin n'a pas vendu.

Coût: 4408

```
SELECT st1.stor_name
FROM   stores st1
WHERE NOT EXISTS (
    SELECT *
    FROM   salesdetail sd2, stores st2
    WHERE  st2.stor_name LIKE 'Barnum%'
    AND    st2.stor_id=sd2.stor_id
    AND NOT EXISTS (
        SELECT *
        FROM   salesdetail sd3
        WHERE  sd3.stor_id=st1.stor_id
        AND    sd3.title_id=sd2.title_id))
```



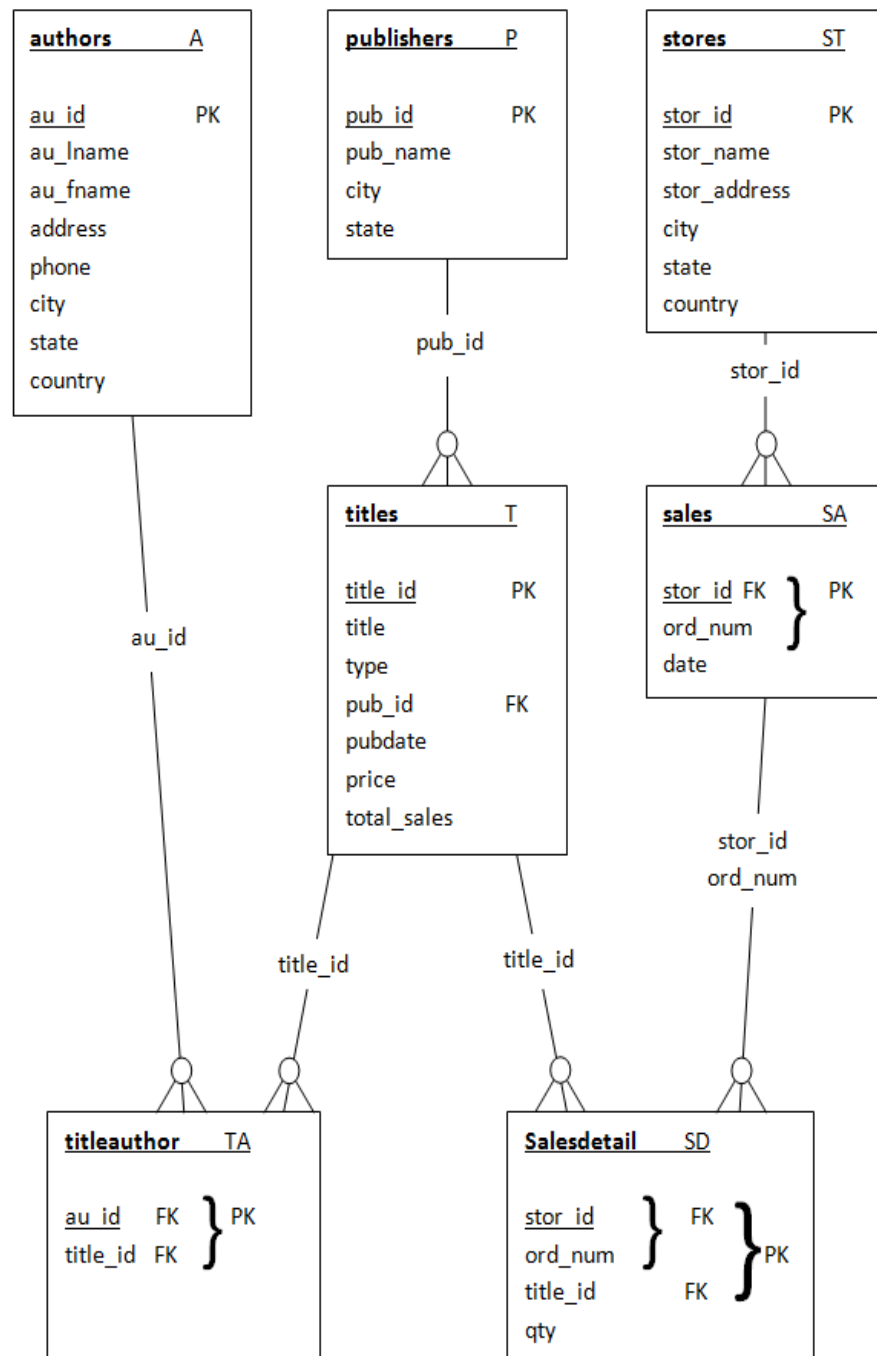
Magasins ayant vendus tous les livres que le magasin Barnum a vendu

Reformulations :

2. Les magasins tels que le nombre de livres vendus par Barnum et par eux-mêmes est le même que le nombre de livre vendu par Barnum

Coût: 60

```
SELECT st1.stor_name
FROM   stores st1, salesdetail sd1,
       salesdetail sd2, stores st2
WHERE  sd1.stor_id=st1.stor_id
AND    sd1.title_id=sd2.title_id
AND    sd2.stor_id=st2.stor_id
AND    st2.stor_name LIKE 'Barnum%'
GROUP BY st1.stor_id
HAVING COUNT(DISTINCT sd1.title_id)=(
    SELECT COUNT(DISTINCT sd3.title_id)
    FROM   salesdetail sd3, stores st3
    WHERE  sd3.stor_id=st3.stor_id
    AND    st3.stor_name LIKE 'Barnum%')
```



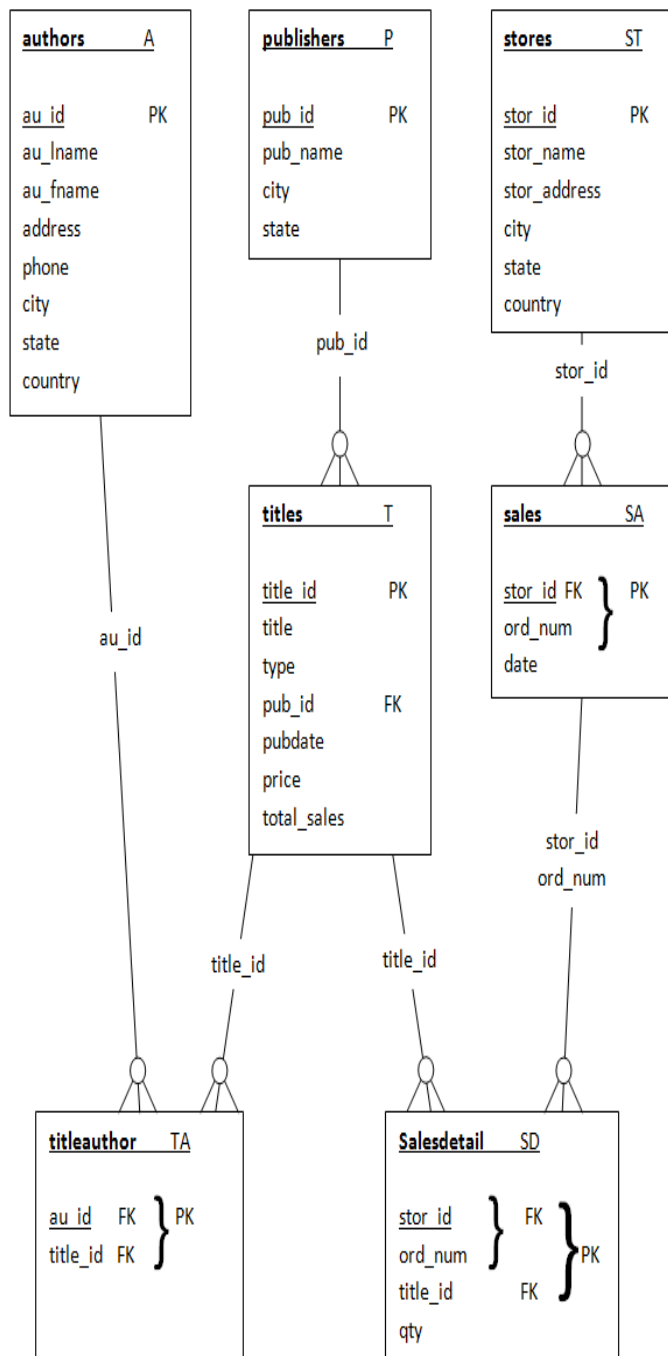
Magasins ayant vendus tous les livres que le magasin Barnum a vendu

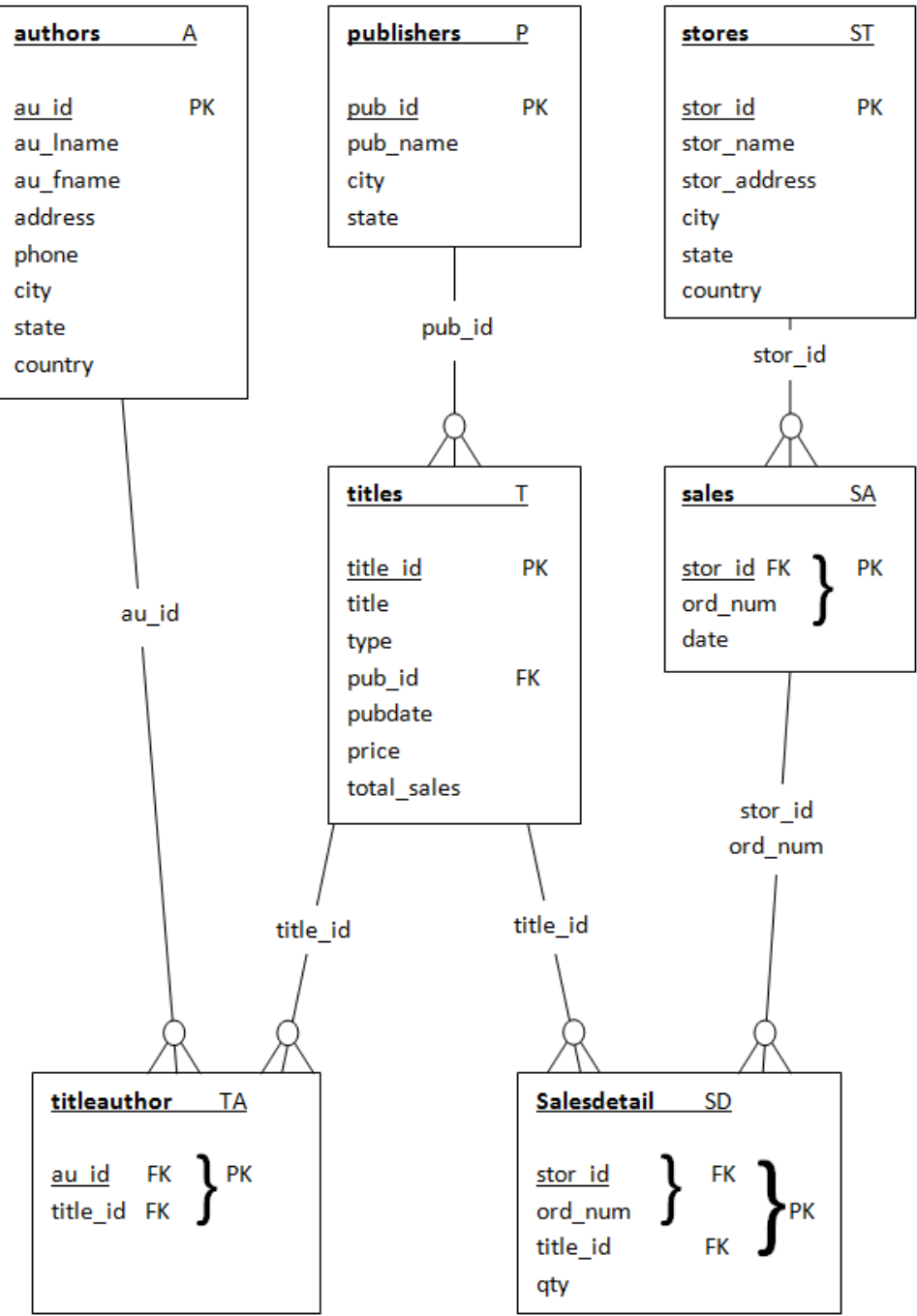
Reformulations :

3. Les magasins tels que le compte des livres vendus par Barnum mais pas par le magasin est zéro.

Coût: 3276

```
SELECT st.stor_name
FROM stores st
WHERE 0=(SELECT COUNT(*)
        FROM ((SELECT DISTINCT sd.title_id
                FROM salesdetail sd, stores st
                WHERE sd.stor_id=st.stor_id
                AND st.stor_name LIKE 'Barnum%')
        EXCEPT
        (SELECT DISTINCT sd.title_id
        FROM salesdetail sd
        WHERE sd.stor_id=st.stor_id)) AS n)
```

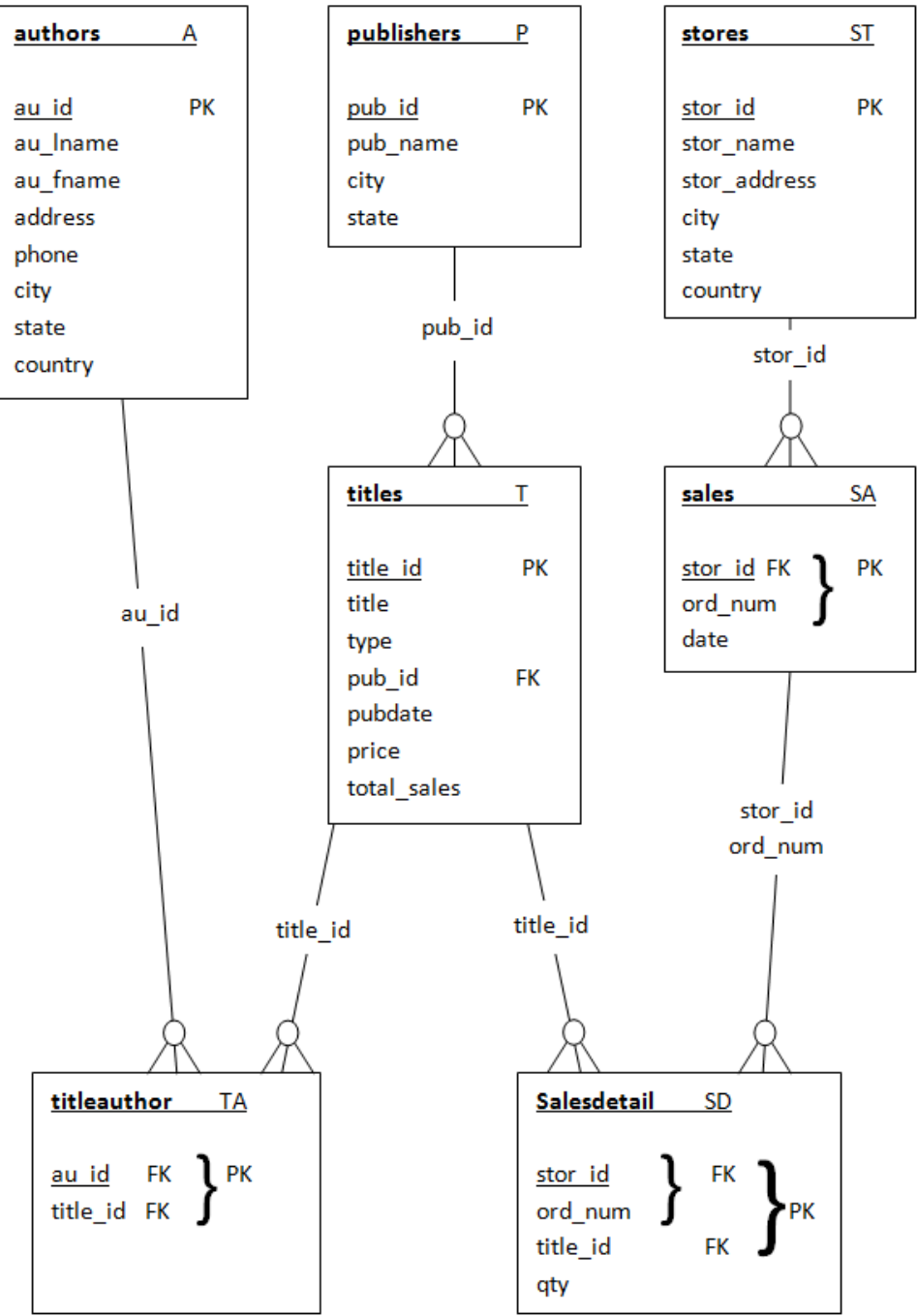




Tous les auteurs et le nombre de livres de plus de 10\$ qu'ils ont écrits, classé par ordre décroissant du nombre de livres

```
SELECT a.au_lname, a.au_fname,
       COUNT(t.title_id)
FROM authors a
LEFT OUTER JOIN titleauthor ta
ON a.au_id = ta.au_id
LEFT OUTER JOIN titles t
ON ta.title_id = t.Title_id
WHERE t.price > 10
GROUP BY a.au_id
ORDER BY COUNT(t.title_id) DESC
```

	au_lname character varying(40)	au_fname character varying(20)	count bigint
1	MacFeather	Stearns	2
2	O Leary	Michael	2
3	del Castillo	Innes	1
4	Yokomoto	Akiko	1
5	Hunter	Sheryl	1
6	Straight	Dick	1
7	Dull	Ann	1
8	Blotchett-Halls	Reginald	1
9	Bennet	Abraham	1
10	Panteley	Sylvia	1
11	White	Johnson	1
12	Green	Marjorie	1
13	Carson	Cheryl	1
14	Gringlesby	Burt	1
15	Karsen	Livia	1



Tous les auteurs et le nombre de livres de plus de 10\$ qu'ils ont écrits, classé par ordre décroissant du nombre de livres

```
SELECT a.au_lname, a.au_fname,
       COUNT(t.title_id)
FROM authors a
LEFT OUTER JOIN titleauthor ta
ON a.au_id = ta.au_id
LEFT OUTER JOIN titles t
ON ta.title_id = t.Title_id
AND t.price > 10
GROUP BY a.au_id
ORDER BY COUNT(t.title_id) DESC
```

	au_lname character varying(40)	au_fname character varying(20)	count bigint
1	O Leary	Michael	2
2	MacFeather	Stearns	2
3	Blotchet-Halls	Reginald	1
4	Ringer	Albert	1
5	Ringer	Anne	1
6	Yokomoto	Akiko	1
7	Dull	Ann	1
8	Bennet	Abraham	1
9	Panteley	Sylvia	1
10	White	Johnson	1
11	Green	Marjorie	1
12	Karsen	Livia	1
13	Gringlesby	Burt	1
14	Carson	Cheryl	1
15	del Castillo	Innes	1
16	Hunter	Sheryl	1
17	Straight	Dick	1
18	McBadden	Heather	0
19	Stringer	Dirk	0

Jusqu'ici...

- On a reçu une base de données
 - Schéma déjà construit (la structure des tables + relations + ...)
 - Données déjà entrées
- On a seulement fait des consultations

Créer sa propre base de données

- Consiste à créer les tables
 - Avec leurs relations
 - Leurs contraintes d'intégrité
 - c-a-d les tables n'acceptent les données que si les contraintes d'intégrité sont respectées

CREATE TABLE

```
CREATE TABLE titleauthor (  
  au_id character varying(11)  
  NOT NULL  
  REFERENCES authors(au_id),  
  title_id character varying(6)  
  NOT NULL  
  REFERENCES titles (title_id),  
  CONSTRAINT ta_pkey  
  PRIMARY KEY  
  (au_id, title_id)
```

nom de la table

nom colonne & type

contrainte intégrité

relation

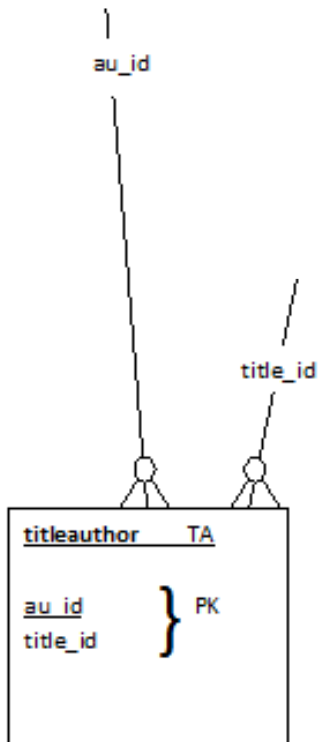
nom colonne & type

contrainte intégrité

relation

contrainte de table

clef primaire



CREATE TABLE

```
CREATE TABLE nom_table ( [  
    { nom_colonne type_donnees  
        [ DEFAULT default_expr ] [ contrainte_colonne [...] ]  
    | contrainte_table }  
    [, ... ]  
    ] )
```

CREATE TABLE

```
CREATE TABLE nom_table ( [  
    { nom_colonne type_donnees  
        [ DEFAULT default_expr ] [ contrainte_colonne [...] ]  
    | contrainte_table }  
    [, ... ]  
    ] )
```

CREATE TABLE

- Types colonne :
 - character [(n)]
 - character varying [(n)]
 - smallint
 - integer
 - numeric [(p, s)]
 - p=nombre total de chiffres significatifs, s=après la virgule
 - double precision
 - timestamp
 - confer annexe syllabus

Example

```
CREATE TABLE utilisateurs (  
    u_id          INTEGER,  
    prenom        CHARACTER VARYING (50),  
    nom           CHARACTER VARYING (50),  
    email         CHARACTER VARYING (50),  
    naissance     TIMESTAMP,  
    taille        NUMERIC (5,2),  
    theme_id      INTEGER  
)
```

CREATE TABLE

```
CREATE TABLE nom_table ( [  
    { nom_colonne type_donnees  
        [ DEFAULT default_expr ] [ contrainte_colonne [...] ]  
    | contrainte_table }  
    [, ... ]  
]
```

CREATE TABLE

- Contraintes colonnes
 - NOT NULL
 - NULL
 - UNIQUE
 - PRIMARY KEY
 - CHECK (expression)
 - REFERENCES table (colonne)

Example

```
CREATE TABLE utilisateurs (  
    u_id          INTEGER PRIMARY KEY,  
    prenom        CHARACTER VARYING (50) NOT NULL,  
    nom           CHARACTER VARYING (50) NOT NULL,  
    email         CHARACTER VARYING (50) UNIQUE,  
    naissance     TIMESTAMP NOT NULL,  
    taille        NUMERIC (5,2) CHECK (taille>0),  
    theme_id      INTEGER REFERENCES themes (theme_id)  
)
```

CREATE TABLE

```
CREATE TABLE nom_table ( [  
    { nom_colonne type_donnees  
        [ DEFAULT default_expr ] [ contrainte_colonne [...] ]  
    | contrainte_table }  
    [, ... ]  
]
```

CREATE TABLE

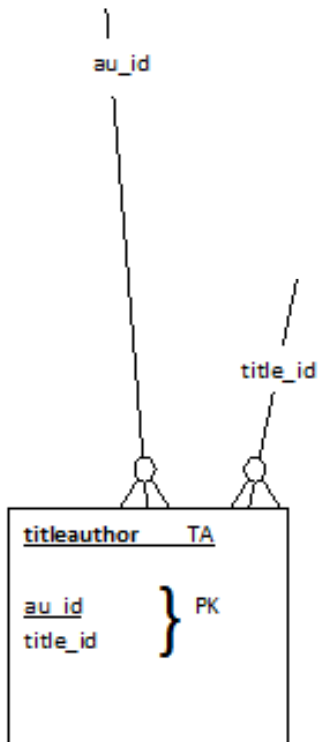
- Contraintes table
 - UNIQUE (colonne1, colonne2, ...)
 - PRIMARY KEY (colonne1, colonne2, ...)
 - CHECK (expression)
 - FOREIGN KEY (colonneA1, colonneA2, ...) REFERENCES table (colonneB1, colonneB2, ...)

Example

```
CREATE TABLE themes (  
    theme_id      INTEGER PRIMARY KEY,  
    couleurFond   CHARACTER (10),  
    couleurAvant  CHARACTER (10),  
    CHECK (couleurFond<>couleurAvant),  
    UNIQUE (couleurFond,couleurAvant)  
)
```

Autre exemple

```
CREATE TABLE titleauthor (  
    au_id CHARACTER VARYING (11) NOT NULL  
        REFERENCES authors(au_id),  
    title_id character varying(6) NOT NULL  
        REFERENCES titles (title_id),  
    CONSTRAINT ta_pkey PRIMARY KEY (au_id, title_id)  
)
```



Contraintes d'intégrité

- Vérification effectuée automatiquement.
 - Garantie ultime de l'intégrité des données.
 - Travail pour le développeur :
 - Une seule fois réfléchir correctement !
 - Inutile de perdre son temps à valider ensuite.

=> Il est très intéressant d'utiliser les contraintes d'intégrité.

Suppression d'une TABLE

`DROP TABLE table`

- Attention, à ne pas violer les contraintes d'intégrité quand on drop une table
- En général il y a un ordre pour dropper les tables

`DROP TABLE table CASCADE`

- Supprime la table + toutes ces choses qui empêchent de supprimer cette table
- Potentiellement peut effacer la BD complète

Modification d'une TABLE

ALTER TABLE nom [*] action [, ...]

- action peut être :

ADD [COLUMN] colonne type [contrainte_colonne [...]]

DROP [COLUMN] colonne

ALTER [COLUMN] colonne [SET DATA] TYPE type [USING expression]

ALTER [COLUMN] colonne SET DEFAULT expression

ALTER [COLUMN] colonne DROP DEFAULT

ALTER [COLUMN] colonne { SET | DROP } NOT NULL

ADD contrainte_table

DROP CONSTRAINT nom_contrainte

ALTER TABLE nom [*] RENAME [COLUMN] colonne TO nouvelle_colonne

ALTER TABLE nom RENAME TO nouveau_nom

Transformer un problème réel en un schéma de BD

- Si on fait n'importe quoi on passe à côté des avantages du modèle relationnel
 - query impossible à écrire
 - performances médiocres de la base de données
 - problèmes de redondance et de cohérence
 - problème de mise à jour (réécritures)
- On utilise une technique appelée normalisation

Première forme normale

Produit (PK)	Fournisseur
téléviseur	VIDEO SA, HITEK LTD

- Comment connaître tous les produits de HITEK LTD ?
 - Recherche au sein du champ texte fournisseur
 - lent
 - facilite les erreurs de frappe

Première forme normale

- Première forme normale
 - Chaque attribut d'un tuple contient une valeur atomique (non composée)

Produit (PK)	Fournisseur (PK)
téléviseur	VIDEO SA
téléviseur	HITEK LTD

Seconde forme normale

Produit (PK)	Fournisseur (PK)	Adresse fournisseur
téléviseur	VIDEO SA	12 rue du cherche-midi
écran plat	VIDEO SA	12 rue du cherche-midi
téléviseur	HITEK LTD	25 Bond Street

- Que se passe-t'il quand VIDEO SA change d'adresse ?
 - Multiples mises à jour
 - Risque d'incohérence des données

Seconde forme normale

- Seconde forme normale
 - Première forme normale
 - Chaque attribut qui n'appartient pas à la clef ne dépend pas uniquement d'une partie de la clef

Produit (PK)	Fournisseur (PK & FK)
téléviseur	VIDEO SA
écran plat	VIDEO SA
téléviseur	HITEK LTD

Fournisseur (PK)	Adresse fournisseur
VIDEO SA	12 rue du cherche-midi
HITEK LTD	25 Bond Street

Troisième forme normale

Fournisseur (PK)	Adresse fournisseur	Ville	Pays
VIDEO SA	12 rue du cherche-midi	PARIS	FRANCE
HITEK LTD	25 Bond Street	LONDON	ENGLAND

- Que se passe-t'il quand la Belgique annexe la France ?
 - Multiples mises à jour
 - Risque d'incohérence des données

Troisième forme normale

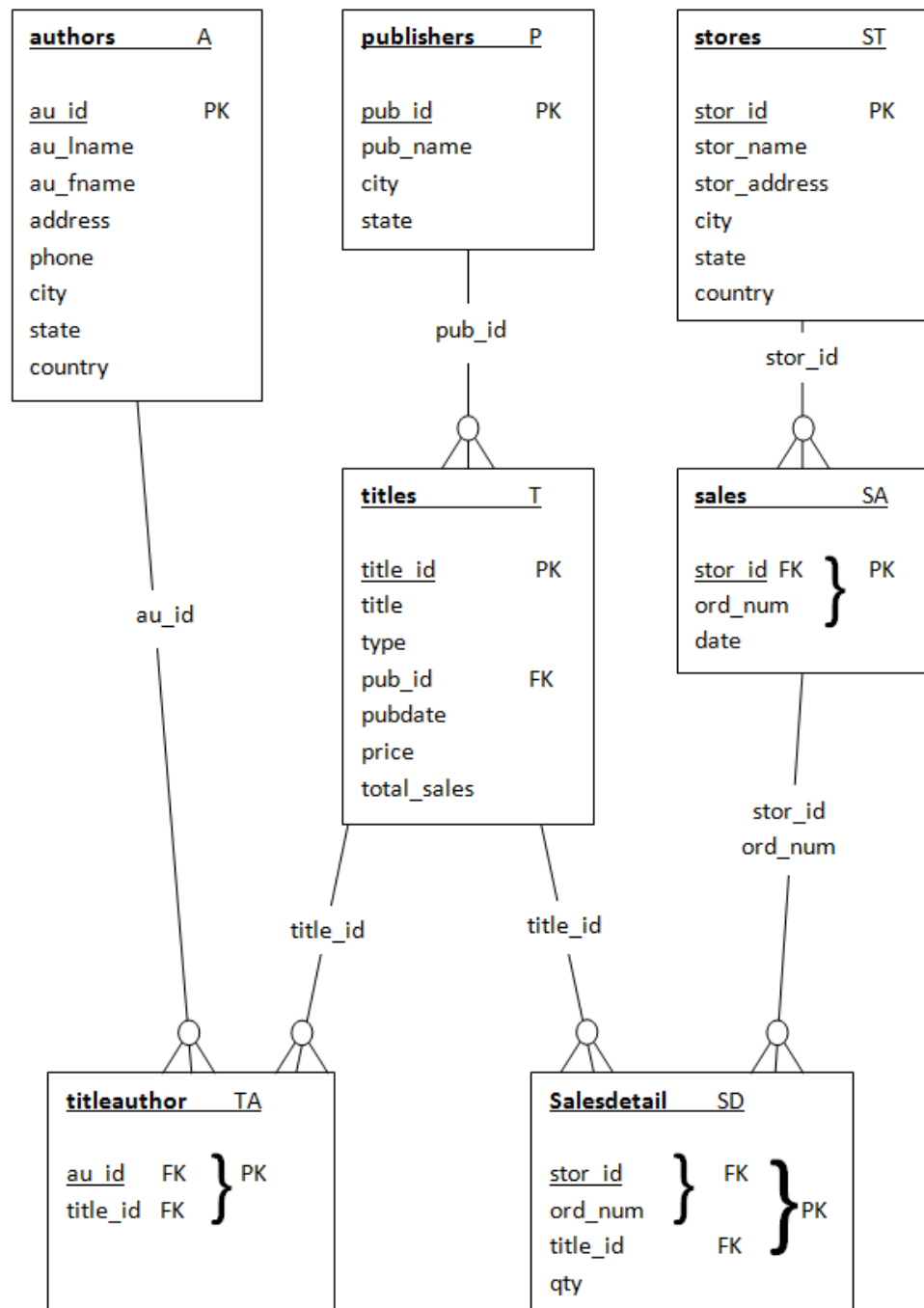
- Troisième forme normale
 - Deuxième forme normale
 - Chaque attribut qui n'appartient pas à la clef ne dépend pas d'attributs n'appartenant pas non plus à la clef

Fournisseur (PK)	Adresse fournisseur	Ville (FK)
VIDEO SA	12 rue du cherche-midi	PARIS
HITEK LTD	25 Bond Street	LONDON

Ville (PK)	Pays
PARIS	FRANCE
LONDON	ENGLAND

Normalisation

Qu'est-ce qui n'est pas normalisé ?



Normalisation d'un problème réel

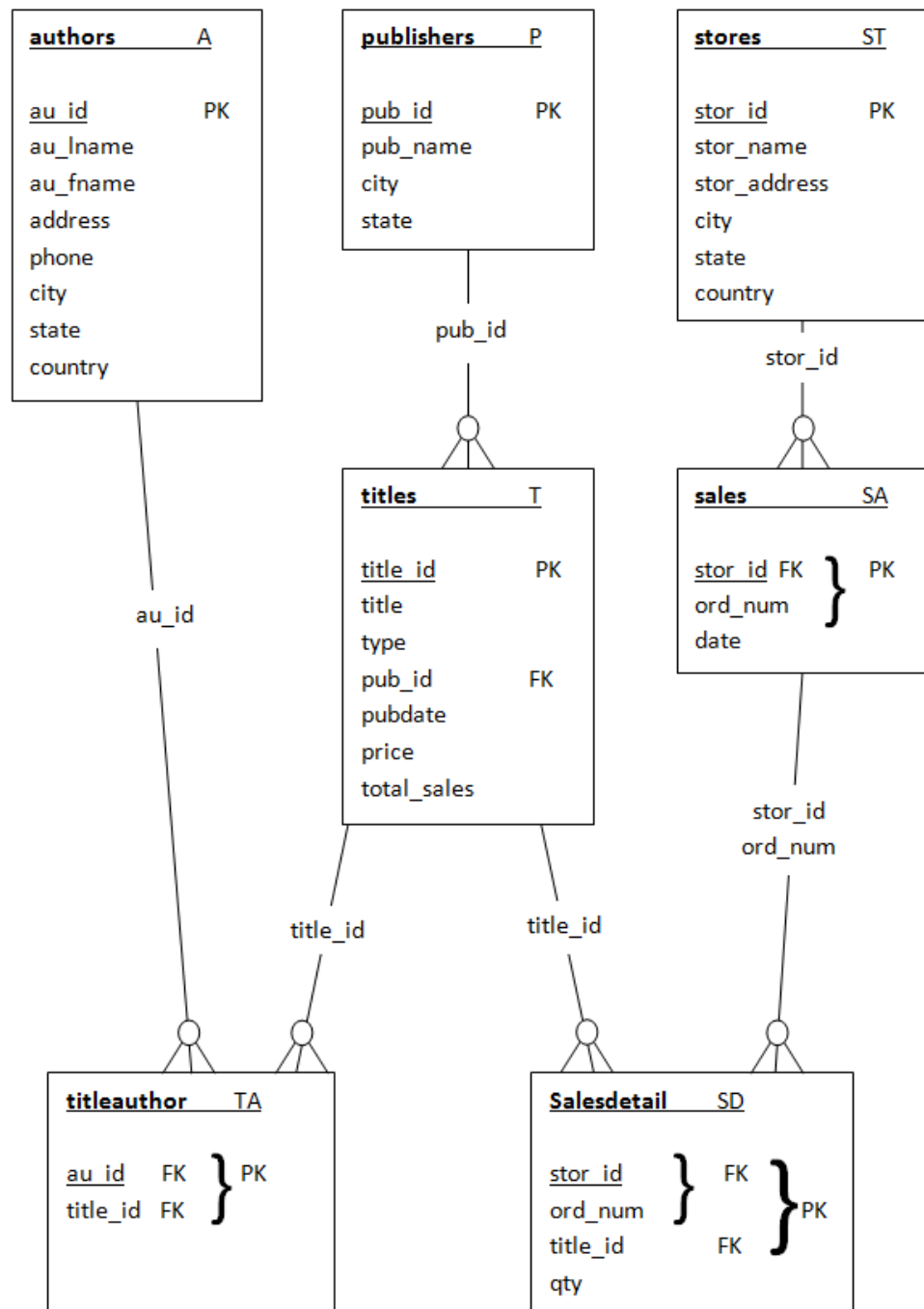
1. Identifier les champs
2. Les regrouper en tuples
3. Normaliser en 1^{ère} forme
4. Normaliser en 2^{ème} forme
5. Normaliser en 3^{ème} forme
6. (on peut ensuite normaliser en Boyce-Codd, 4^{ème} FN, 5^{ème} FN et FN domaine clef)

Dénormalisation

- Normalisation = remplacer la redondance par des relations.
- Relations = jointures.
- Jointures = perte de performance.
- On dénormalisera donc pour des raisons de performance.

Qu'est-ce qui n'est pas normalisé ?

Gain de performance ?



Maintenant que l'on peut créer une BD, on peut y mettre des données

- INSERT INTO table [(colonne [, ...])]
 { DEFAULT VALUES |
 VALUES ({ expression | DEFAULT } [, ...]) }
- UPDATE table
 SET colonne = { expression | DEFAULT } [, ...]
 [WHERE condition]
- DELETE FROM table [WHERE condition]

Rappel table themes

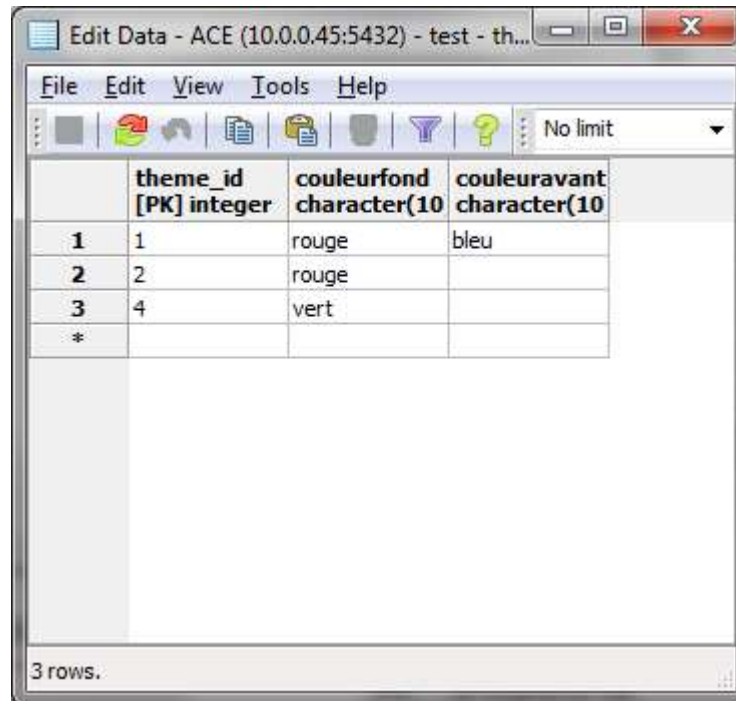
```
CREATE TABLE themes (  
    theme_id      INTEGER PRIMARY KEY,  
    couleurFond   CHARACTER (10),  
    couleurAvant  CHARACTER (10),  
    CHECK (couleurFond<>couleurAvant),  
    UNIQUE (couleurFond,couleurAvant)  
)
```

Exemples d'INSERT

- `INSERT INTO themes VALUES (1, 'rouge', 'bleu')`
- `INSERT INTO themes VALUES (2, 'rouge', DEFAULT)`
- `INSERT INTO themes DEFAULT VALUES`
- `INSERT INTO themes (couleurfond, theme_id) VALUES ('vert', 4)`

Exemples d'INSERT

- `INSERT INTO themes VALUES (1, 'rouge', 'bleu')`
- `INSERT INTO themes VALUES (2, 'rouge', DEFAULT)`
- ~~`INSERT INTO themes DEFAULT VALUES`~~
- `INSERT INTO themes (couleurfond, theme_id) VALUES ('vert', 4)`



The screenshot shows a window titled "Edit Data - ACE (10.0.0.45:5432) - test - th...". The window contains a table with the following columns: **theme_id [PK] integer**, **couleurfond character(10)**, and **couleuravant character(10)**. The table has 3 rows of data. The first row has values 1, rouge, and bleu. The second row has values 2, rouge, and an empty cell. The third row has values 4, vert, and an empty cell. The status bar at the bottom indicates "3 rows."

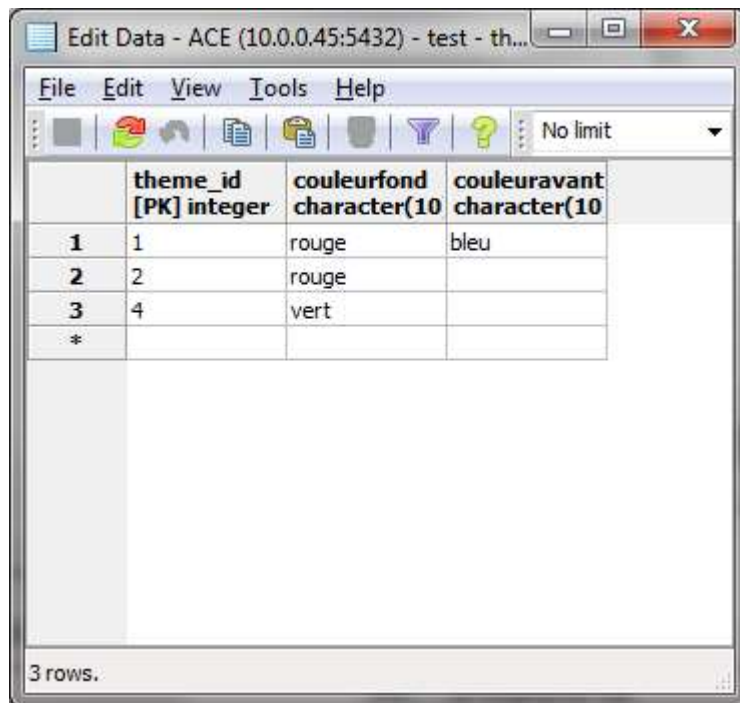
	theme_id [PK] integer	couleurfond character(10)	couleuravant character(10)
1	1	rouge	bleu
2	2	rouge	
3	4	vert	
*			

Exemple d'UPDATE

UPDATE themes

SET couleuravant='jaune'

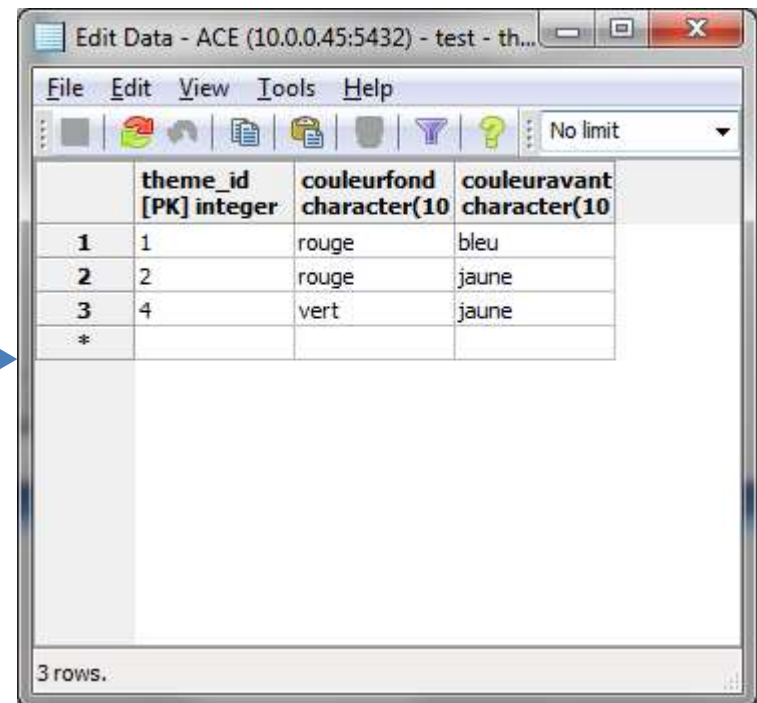
WHERE couleuravant IS NULL



Edit Data - ACE (10.0.0.45:5432) - test - th...

	theme_id [PK] integer	couleurfond character(10)	couleuravant character(10)
1	1	rouge	bleu
2	2	rouge	
3	4	vert	
*			

3 rows.



Edit Data - ACE (10.0.0.45:5432) - test - th...

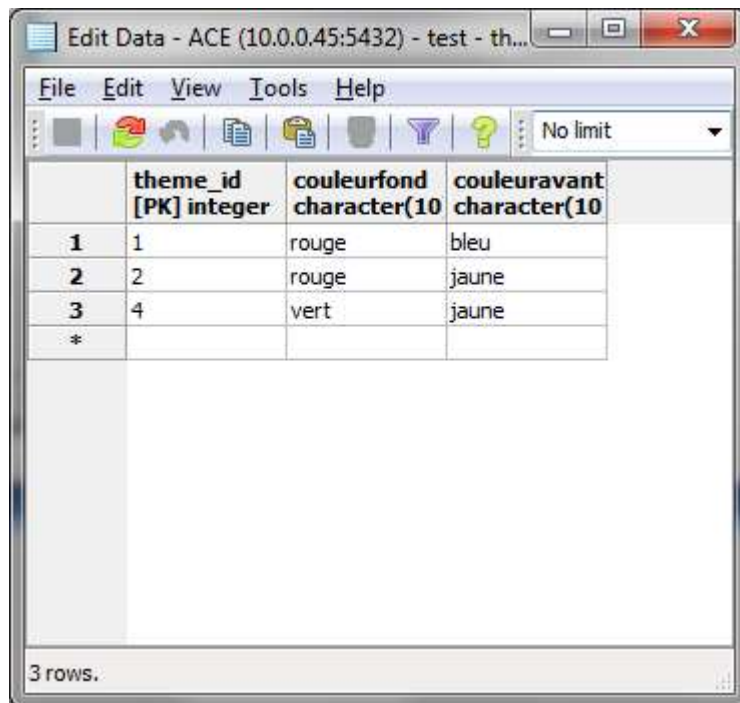
	theme_id [PK] integer	couleurfond character(10)	couleuravant character(10)
1	1	rouge	bleu
2	2	rouge	jaune
3	4	vert	jaune
*			

3 rows.

Exemple de DELETE

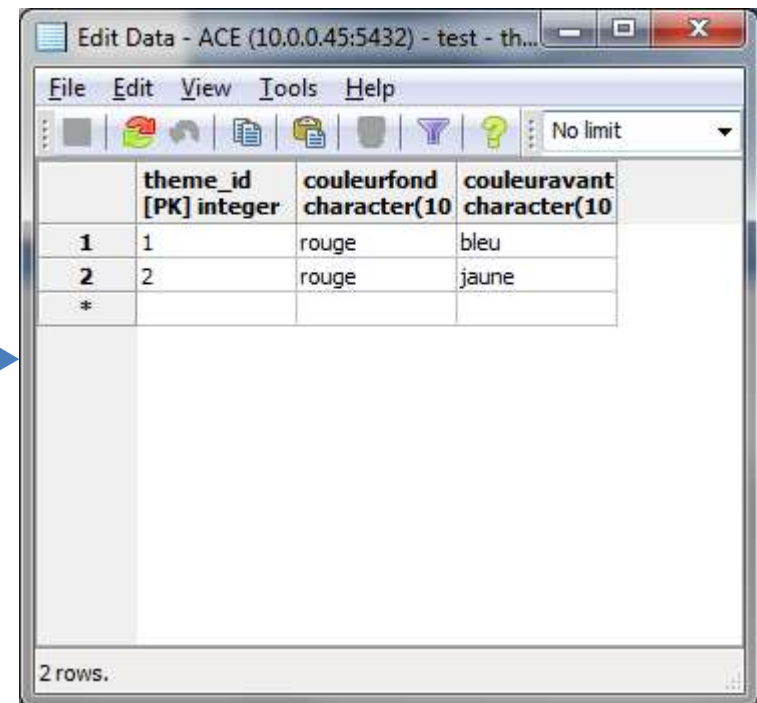
```
DELETE FROM themes  
WHERE couleurfond='vert'
```

Attention aux contraintes d'intégrité !



	theme_id [PK] integer	couleurfond character(10)	couleuravant character(10)
1	1	rouge	bleu
2	2	rouge	jaune
3	4	vert	jaune
*			

3 rows.



	theme_id [PK] integer	couleurfond character(10)	couleuravant character(10)
1	1	rouge	bleu
2	2	rouge	jaune
*			

2 rows.

Exemples de DELETE

Pour effacer un tuple en particulier : PK

```
DELETE FROM themes  
WHERE theme_id=1
```

Pour effacer tous les tuples :

```
DELETE FROM themes
```

SQL procédural

- But : permettre de programmer un comportement directement au niveau de la base de données
- Sur les bases de données modernes
 - Nombreux choix de langages
- Historiquement
 - Extension des instructions SQL
 - Cas de PostgreSQL : PL/pgSQL

Procédural ≠ Orienté Objet

- SQL procédural date des années 1970
 - OO commence réellement dans les années 1990
 - Pas de notion d'objet ni de classe
 - Notion de procédure \approx méthode statique dans une unique classe
 - Beaucoup de lourdeur
- Etat
 - Pas de variable globale
 - On a les tables par contre
 - Les procédures sont globales (stockées dans une table système)

CREATE FUNCTION

- Permet de déclarer une fonction
 - avec un nom
 - avec des paramètres
 - avec une valeur de retour éventuelle
- Invocation d'une fonction
`SELECT nom_procédure (arg1, arg2, ...)`

Example function

```
CREATE FUNCTION fib (integer) RETURNS integer AS $$
DECLARE
    fib_for ALIAS FOR $1;
BEGIN
    IF fib_for < 2 THEN
        RETURN fib_for;
    END IF;
    RETURN fib(fib_for - 2) + fib(fib_for - 1);
END;
$$ LANGUAGE plpgsql;

SELECT fib(8)
```

CREATE FUNCTION

Uniquement les types

```
CREATE FUNCTION nomFonction(type1, type2,..., typeX)
  RETURNS typeOut AS $$
  DECLARE
    nomParam1 ALIAS FOR $1;
    nomParam2 ALIAS FOR $2;
    ...      nomParamX ALIAS FOR $X;
    nomVar1 typeVar1;
    nomVar2 typeVar2;
    ...      nomVarY typeVarY;
  BEGIN
    corpsDeclare ;
  END ;
  $$ LANGUAGE plpgsql;
```

Paramètres

Variables locales

\$\$ sert de délimiteur entre SQL et PL/pgSQL

Affectation

variable := expression ;

Commentaire

-- tout ce qui suit -- est ignoré jusqu'à la fin de la
ligne

IF

- IF condition THEN ... ELSE ... END IF;

FOR

```
FOR variable_locale IN instruction_select  
LOOP  
    instructions  
END LOOP
```

- `variable_locale` doit être déclarée et typée dans la partie `DÉCLARE` de la procédure
 - Le type `RECORD` permet d'itérer sur des tuples et d'accéder aux champs par `variable_locale.champ`

Exemple FOR et IF

```
CREATE FUNCTION compteSalesDetailQtyMin10() RETURNS INTEGER AS $$  
DECLARE  
    i INTEGER := 0;  
    record RECORD;  
BEGIN  
    FOR record IN SELECT * FROM Salesdetail LOOP  
        IF record.qty>=10 THEN  
            i := i + record.qty;  
        END IF;  
    END LOOP;  
    RETURN i;  
END;  
$$ LANGUAGE plpgsql;
```

FOR

- La procédure précédente est équivalente à
`SELECT SUM(qty) FROM Salesdetail WHERE qty>=10;`
- **Remarque importante** : il n'y a aucune valeur ajoutée à implémenter soi-même ce qui devrait en fait être une requête SQL. Le code ne sera jamais aussi performant que la requête équivalente. Cela prend toujours plus de temps d'écrire une implémentation plutôt que d'écrire la requête, et le risque d'erreur est plus élevé. Dans le cadre de ce cours, ceci est donc considéré comme une faute et sanctionné comme tel.

WHILE

WHILE condition LOOP

instructions

END LOOP

Exceptions

```
CREATE FUNCTION nomFonction(type1, type2,..., typeX)
  RETURNS typeOut AS $$
  DECLARE
    nomParam1 ALIAS FOR $1 ;
    nomParam2 ALIAS FOR $2 ;
    ...      nomParamX ALIAS FOR $X ;
    nomVar1 typeVar1;
    nomVar2 typeVar2;
    ...      nomVarY typeVarY;
  BEGIN
    corpsDeclare ;
  EXCEPTION
  WHEN condition [ OR condition ... ] THEN
    instructions_gestion_erreurs
  [ ... ]
  END ;
$$ LANGUAGE plpgsql;
```

Exemple Exception

```
BEGIN
  y := x / 0;
EXCEPTION
  WHEN division_by_zero THEN -- ignore l'erreur
END;
```

- Lever une exception
 - RAISE EXCEPTION condition_name
 - Confer documentation PostgreSQL pour la liste des condition_name possibles
<http://www.postgresql.org/docs/9.3/static/errcodes-appendix.html>