Vocabulaire

- Projection $\pi_{condition}(R)$ → colonnes de `R` satisfaisant `condition` : `select`
- Sélection $\sigma_{condition}(R)$ → tuples (=lignes) de `R` satisfaisant `condition` : `from`
- Produit cartésien $R \times S$ → disjonction des cas : `,`
- Renommage $\rho_{B \leftarrow A}(R)$ → `R.A` devient `R.B` : `as`
- Jointure $S \bowtie_{condition} R$ *ou* jointure naturelle $S \bowtie R$ → produit cartésien suivi d'une con
- Différence $R - S$ → enlève les colonnes `S` de `R` :
  - `not in`
  - `not exists`
  - `except`
- Union $R \cup S$ → ou logique : `or`
- Intersection $R \cap S$ → et logique : `and`
- Division $R \div S$ → *pas très utilisé*

Interaction de l'algèbre relationnelle

- $\pi_A(R \cup S) = \pi_A(R) \cup \pi_A(S)$
- $\sigma_{cond}(R \cup S) = \sigma_{cond}(R) \cup \sigma_{cond}(S)$
- $(R \cup S) \times T = (R \times T) \cup (S \times T)$
- $T \times (R \cup S) = (T \times R) \cup (T \times S)$
- $A \subseteq B \iff$ `not exists (A except B)`

Nom des capitales des pays du continent '`Asia`' avec moins de 10 millions d'habitants :

$\Pi_{name\_city}(city \underset{continent='Asia' \wedge population\_city>10000000 \wedge id=capital}{\bowtie} country)$

```
SELECT name_city FROM city
INNER JOIN country ON
  continent='Asia'
  AND population_city<10000000;
  AND id=capital
```

Nom des pays dans lesquels il existe deux villes différentes ayant exactement le même nombre d'habitants

```
SELECT countryname FROM country
INNER JOIN city AS c1 ON
  c1.countrycode=country.countrycode
INNER JOIN city AS c2 ON
  c2.countrycode=c1.countrycode
  AND c2.population_city=c1.population_city
  AND c2.id!=c1.id;
```

Nom des capitales des pays du continent '`Europe`'

$\Pi_{name\_city}(city \underset{city.id=country.capital \wedge country.continent='Europe'}{\bowtie} country)$

```
SELECT name_city
FROM city
INNER JOIN country ON
  city.id=country.capital
  AND country.continent='Europe';
```

- Dans la requête, pas besoin de `country.countrycode=city.countrycode` car `id` est une clé primaire.

Régions dans lesquelles tous les pays ont une espérance de vie supérieure à 80 ans

```
SELECT region
FROM country AS c1
WHERE region NOT IN(
  SELECT DISTINCT region FROM country AS c2 WHERE
```

Districts de France dans lesquels toutes les villes ont une population supérieure à 500 habitants

```
SELECT district FROM city AS c1
WHERE district NOT IN(
  SELECT DISTINCT district FROM city AS c2
  INNER JOIN country ON
    c2.population_city<=500
    AND name country='France'
```

Régions dans lesquels tous les pays ont une capitale avec plus de 100 000 habi

```
SELECT region FROM country AS c1
WHERE region NOT IN(
  SELECT DISTINCT region FROM country AS c2
  INNER JOIN city ON
    id=capital
    AND population_city<=100000
);
```

Régions dans lesquelles aucun pays n'a un nom qui commence par la lettre 'P'

```
SELECT region FROM country AS c1
WHERE NOT EXISTS(
  SELECT DISTINCT region FROM country AS c2
  WHERE LEFT(c2.name country, 1)='P');
```

Nom des villes qui sont capitale d'un pays et aussi ville d'un autre pays sur le même continent

```
SELECT name_city FROM city AS cit1
INNER JOIN city AS cit2 ON
  cit2.name_city=cit1.name_city
  AND NOT cit2.countrycode=cit1.countrycode
INNER JOIN country AS cou1 ON
  cit1.id=cou1.capital
INNER JOIN country AS cou2 ON
  cit2.countrycode=cou2.countrycode
  AND cou2.continent=cou1.continent;
```

- On supposera qu'on recherche les noms tels qu'ils puissent faire référence à la fois à la capitale d'un pays P1 et à une ville d'un pays P2 (avec P1 ≠ P2 et P1.continent=P2.continent)

Nom des pays dans lesquels la capitale est la ville la plus peuplée

```
SELECT name_country FROM country
WHERE NOT EXISTS(
  SELECT * FROM city AS c1
  WHERE c1.id=capital
  INNER JOIN city AS c2 ON
    c2.population_city>c1.population_city
    AND NOT c2.id=c1.id
);
```

Quelle(s) requête(s) calcule(nt) le nom des villes françaises de plus de 200 000 habitants ?

```
SELECT name_city FROM city WHERE
  population_city>200000
  AND countrycode='FRA';
```

⇒ OK

```
SELECT A.name_city FROM city AS A, country AS B WHERE
  population_city>200000
  AND B.countrycode='FRA';
```

⇒ On va avoir des problèmes à cause du produit cartésien : certaines lignes pourront avoir `B.countrycode='FRA'` et `A.countrycode='ENG'`

Sans utiliser DISTINCT, donnez une requête équivalente en SQL :

```
SELECT DISTINCT Num_Client
FROM COMPTE
WHERE solde < 1000
OR solde > 100000 ;
```

- **AGENCE (*Num_Agence, Nom, Ville, Actif)**
- **CLIENT (*Num_Client, Nom, Prenom, Ville)**
- **COMPTE (*Num_Compte, Num_Agence#, Num_Client#, Solde)**
- **EMPRUNT (*Num_Emprunt, Num_Agence#, Num_Client#, Montant)**

Une première solution exploite le fait que l'opérateur d'union est un opérateur ensembliste et élimine donc les doublons :

```
(SELECT Num_Client
FROM COMPTE
WHERE solde < 1000)
UNION
(SELECT Num_Client
FROM COMPTE
WHERE solde > 100000) ;
```

Une seconde solution détourne le GROUP BY (utilisé d'ordinaire dans le cadre des requêtes d'agrégation)

```
SELECT Num_Client
FROM COMPTE
WHERE solde < 1000
OR solde > 100000
GROUP BY Num_Client ;
```

Les clients n'ayant pas de compte dans la même agence que Liliane Bettencourt. (Tableau résultat : Num_Client).

SELECT Num_Client FROM COMPTE WHERE Num_Agence NOT IN (SELECT Num_Agence FROM COMPTE NATURAL JOIN CLIENT WHERE Client.Nom='Bettencourt' AND Client.Prenom='Liliane') ;

Les agences ayant un actif plus élevé que toutes les agences de Saint-Ouen. (Tableau résultat : Num_Agence).

SELECT Num_Agence
FROM Agence
WHERE Actif > ALL
(SELECT Actif
FROM Agence
WHERE Ville='Saint Ouen') ;

Le solde moyen des comptes clients, pour chaque agence dont le solde moyen est supérieur à 10000. (Tableau résultat : Num_Agence, Solde_Moyen).

SELECT AVG(Solde) as Solde_Moyen
FROM Compte
GROUP BY Num_Agence
HAVING AVG(Solde) > 10000 ;

Le nombre de clients de l'agence de nom "Paris-BNF" dont la ville n'est pas renseignée dans la relation CLIENT. (Tableau résultat : Nombre).

SELECT COUNT(DISTINCT num_client) as Nombre
FROM Client, Compte, Agence
WHERE Client.Num_client = Compte.Num_client
AND Agence.Num-Agence=Compte.Num-Agence
AND Agence.Nom='Paris-BNF'
AND Client.Ville IS NULL ;

Attention, la syntaxe .... > MAX (SELECT Actif FROM...) est incorrecte.

Les clients ayant un compte dont le solde est supérieur à la somme totale de tous les actifs des agences de Saint-Ouen. (Tableau résultat : Num_Client).

SELECT Num_Client
FROM Compte
WHERE Solde >
(SELECT SUM(Actif)
FROM Agence WHERE
Ville='Saint-Ouen') ;

Les clients dont la somme du solde de tous les comptes est inférieure à l'actif de chaque agence. (Tableau résultat : Num_Client)

SELECT Num_Client
FROM Compte
GROUP BY Num_Client
HAVING SUM(Solde) <
(SELECT MIN(Actif)
FROM Agence) ;

Attention à la jointure naturelle sur Client et Agence. La jointure sera entre autres faite sur les deux attributs ville, ce qui forcera une contrainte supplémentaire sur les données (si un client possède un compte dans une agence domiciliée dans une ville différente de celle où il habite, alors ce compte n'apparaitra pas).

Attention également à ne pas oublier le mot clef Distinct : nous ne voulons compter chaque client ayant un compte dans l'agence "Paris-BNF" qu'une seule fois, même s'il y possède plusieurs comptes.

les clients ayant un compte dans toutes les agences de Saint-Ouen. (Tableau résultat : Num_Client).

SELECT Num_Client
FROM Client
WHERE NOT EXISTS
(SELECT * FROM Agence
WHERE Ville='Saint-Ouen'
AND NOT EXISTS
(SELECT * FROM Compte
WHERE Client.Num_Client=Compte.Num_Client
AND Compte.Num_Agence=Agence.Num_Agence));

Les clients résidant à Paris, avec un compte dont le solde est supérieur à 10000 et un emprunt dont le montant est inférieur à 100000. (Tableau résultat : Num_Client.)

$$\Pi_{Num\_Client}(\sigma_{ville='Paris'}(Client) \bowtie \sigma_{solde>1000}(Compte) \bowtie \sigma_{montant<100000}(Emprunt))$$

$$\Pi_{Num\_Client}(\sigma_{ville='Paris'}(Client)) \bowtie \sigma_{solde>1000}(Compte)) \bowtie \Pi_{Num\_Client}(\sigma_{montant<100000}(Emprunt))$$

$$\Pi_{Num\_Client}(\sigma_{ville='Paris'}(Client)) \cap \Pi_{Num\_Client}(\sigma_{solde>1000}(Compte)) \cap \Pi_{Num\_Client}(\sigma_{montant<100000}(Emprunt))$$

Les clients n'ayant contracté aucun emprunt. (Tableau résultat : Num_Client.)

$$\Pi_{Num\_Client}(Client) - \Pi_{Num\_Client}(Emprunt)$$

Les clients ayant un compte dans la même agence que Liliane Bettencourt. (Tableau résultat : Num_Client.)

$$\Pi_{Num\_Client}(\Pi_{Num\_Agence}(\sigma_{Prenom='Liliane' \wedge Nom='Bettencourt'}(Client) \bowtie Compte) \bowtie Compte)$$

Soit la table :

ACTIVITE	Intitulé	Titulaire	Heures_Cours	Heures_Tp
	Java	PHE	30	15
	Labo prog	PHE		45
	BD	JLH	30	
	Projet qualité	NHA		30
	Modélisation	JLH	20	10
	Conception	VEN	45	
	Mise en oeuvre	VEN	60	
	Labo gestion	NHA		45

Titulaire	Charge
JLH	60
NHA	
PHE	90
VEN	

Titulaire	Charge
JLH	30
NHA	
PHE	45
VEN	

1. Donnez le résultat des requêtes suivantes :

   (a)
```
SELECT Titulaire, SUM(Heures_Cours) + SUM(Heures_Tp) as Charge
from ACTIVITE
group by Titulaire;
```

   (b)
```
SELECT Titulaire, SUM(Heures_Cours + Heures_Tp) as Charge
from ACTIVITE
group by Titulaire ;
```

1. Le premier et le troisième tuple du résultat auraient été identiques si les valeurs nulles de la table avaient été remplacées par 0. En revanche le calcul aurait été différent pour le second et le quatrième tuple. Au lieu d'obtenir des valeurs nulles, nous aurions obtenu 75 et 105.

Que pensez-vous de ces résultats ? Comment procéderiez-vous s'il vous fallait calculer le nombre d'heures total de chaque enseignant ? Attention, <> NULL n'est pas correct

WITH heures AS

(select TITULAIRE, sum(H_COURS) as CHARGE
from ACTIVITE
where H_COURS IS NOT NULL
group by TITULAIRE
UNION ALL
SELECT TITULAIRE, sum(H_TP) as CHARGE
from ACTIVITE
where H_TP IS NOT NULL
group by TITULAIRE
)

SELECT titulaire, SUM(CHARGE) as charge
FROM heures
GROUP BY titulaire ;

ou bien :

SELECT titulaire, SUM(CHARGE) as charge
FROM

(select TITULAIRE, sum(H_COURS) as CHARGE
from ACTIVITE
where H_COURS IS NOT NULL
group by TITULAIRE
UNION ALL
SELECT TITULAIRE, sum(H_TP) as CHARGE
from ACTIVITE
where H_TP IS NOT NULL
group by TITULAIRE
) as heures

GROUP BY titulaire ;

country

Column	Type
countrycode	character(3)
name_country	text
continent	text
region	text
population_country	integer
lifeexpectancy	real
capital	integer
governmentform	text
gnp	real

language

Column	Type
countrycode	character(3)
language	carchar(20)
isofficial	boolean
percentage	real

city

Column	Type
id	integer
name_city	text
countrycode	character(3)
district	text
population_city	integer

Sans table temporaire

select continent, region
from country c **natural join** language
where isofficial
group by continent, region
having COUNT(distinct language)>=

ALL (**select COUNT**(distinct l2.language)
from country **natural join** language l2
group by l2.continent, l2.region
having l2.continent = c.continent)

Q1. Ecrire une requete SQL qui renvoie, pour chaque continent, la région avec le plus grand nombre de langues officielles parlées dans la région ?

Avec table temporaire

Select region, continent, COUNT(distinct language)
from country **natural join** language
where isofficial
group by region, continent ;

WITH Langues AS

(**select** region, continent,
COUNT(distinct language) **as** nbe
from country **natural join** language
where isofficial
group by region, continent)

select L1.continent, L1.region
from Langues L1
where L1.nbe=

(select MAX(L.nbe) from Langues L
where L.continent=L1.continent) ;

Q2. Pays avec moins de 10 villes et tels que la capitale a une population > 30% de la population du pays ? + pays avec 10 villes ou plus, et la capitale a >30% de la population des dix villes les plus peuplées, réunies ?

select name_country as nom
from country **inner join** city **on** country.capital=city.id
where 10*city.population_city> 3* country.population_country
union
select nom **from**

(**select** c2.name_country as nom, c2.countrycode,
SUM(c2.population_city) as pop_villes
from country **natural join** city c2 **group by** c2.countrycode
having COUNT(distinct c2.id)>=10
and 3*pop_villes < 10* (**select** c3.population_city **from** city c3 **where** c3.id=c2.capital)
ORDER BY c2.population_city **DESC LIMIT 10**) as foo

Q3. Donner pour chaque région, le pays avec le plus haut gnp par habitant ?

select distinct region, name_country
from country c1 **where not exists**

(**select * from** country c2
where c2.region=c1.region
and c2.gnp/c2.population_country > c1.gnp / c1.population_country) ;

ou bien :

WITH AUX **AS**

(**select** region, countrycode,
name_country as nom,
gnp/population_country as gnp_hab
from country
where population_country>0)

select region, countrycode
from AUX
where aux.gnp_hab=

(**select MAX**(a2.gnp_hab)
from AUX a2
where a2.region=aux.region) ;

Q3.b Donner les régions qui maximisent l'écart (dans la région) entre lifeexpectancy la plus haute et la plus basse ?

WITH table(region, ecart) AS

(**select** region, MAX(lifeexpectancy)- **MIN**(lifeexpectancy)
from country **group by** region)
select distinct region **from** table
where ecart=(select MAX(ecart) from table) ;

ou alors :

select region from country
group by region
having MAX(lifeexpectancy)-**MIN**(lifeexpectancy)>= **ALL**(**select MAX**(lifeexpectancy)-**MIN**(lifeexpectancy)
from country **group by** region) ;

Q4 - Régions où il n'existe qu'une seule forme de gouvernement

Requête avec op. d'agrégat :

select region from country group by region
having COUNT(distinct governmentform)=1 ;

Requete sans COUNT et sans ALL :

select distinct c1.region from country c1
where not exists (select * from country c2 where c2.region=c1.region
and c2.governmentform <> c1.governmentform) ;

Remarque : les deux solutions ci-dessus renvoient aussi les régions où il existe une seule forme de gouvernement, et aussi des pays dans la région avec un NULL comme governmentform.

Schema : Films(titre, realisateur, duree) Seances(idseance, cinema, titre, jour, heure_debut)

Question : quels sont les cinemas où plus de la moitié des films diffusés, sont des films de Varda, et tels qu'au moins un tiers des films réalisés par Varda, soient diffusés dans le cinema

```
select cinema from Seances S natural join Films
where S.realisateur='Varda'
group by cinema
HAVING 2*COUNT(distinct S.titre)>

(select COUNT(distinct S2.titre) from Seances
S2 group by S2.cinema having S2.cinema=S.cinema)
AND 3*COUNT(distinct S.titre)>(select COUNT(distinct F.titre) from Films F

where
F.realisateur='Varda') ;
```

- Trouver les films qui partagent le même réalisateur et au moins un acteur avec Chinatown :
  $\pi_{cinema}(\sigma_{realisateur='Polanski'}(\sigma_{Film.titre=Seance.titre}(Film \times Seance)))$

```
select titre from Film where (realisateur, acteur) in
    (select realisateur, acteur from Film where titre='Chinatown');
```

$\Leftrightarrow \pi_{cinema}(\sigma_{realisateur='Polanski' \wedge Film.titre=Seance.titre}(Film \times Seance)))$

```
select F1.titre from Film F1, Film F2
where F2.titre='Chinatown' and F1.realisateur=F2.realisateur and F1.acteur=F2.acteur;
```

- Trouver des acteurs qui ont joué dans des films de Kubrick ou de Polanski :
  $\pi_{acteur}(\sigma_{realisateur='Kubrick'}(Film)) \cup \pi_{acteur}(\sigma_{realisateur='Polanski'}(Film))$

```
select acteur from Film where realisateur='Kubrick'
UNION select acteur from Film where realisateur='Polanski';
```

```
select acteur from Film where realisateur='Kubrick' or realisateur='Polanski';
```

- Trouver tous les acteurs qui sont aussi réalisateurs :
  - $\pi_{personne}(\rho_{personne \leftarrow acteur}(Film)) \cap \rho_{personne \leftarrow realisateur}(Film))$ $\Leftrightarrow$

```
select acteur as personne from Film intersect
   (select realisateur as personne from Film);
```

```
select acteur as personne from Film in
    (select realisateur as personne from Film);
```

- Les films dans lesquels Deneuve ne joue pas :
  - $\pi_{titre}(Film) - \pi_{titre}(\sigma_{acteur='Deneuve'}(Film))$

```
select titre from Film where titre not in
    (select titre from Film where acteur='Deneuve');
```

- Les films dans lesquels ne joue que Deneuve :
  - $\pi_{titre}(Film) - \pi_{titre}(\sigma_{acteur \neq 'Deneuve'}(Film))$

```
select titre from Film where titre not in
    (select titre from Film where acteur<>'Deneuve');
```

- Les films dans lesquels Deneuve n'est pas la seule personne à jouer :
  - $\pi_{titre}(Film) - \sigma_{acteur='Deneuve'}(Film)$

```
select titre from Film where acteur not in    (select acteur from Film where acteur='Deneuve');
```

$\Leftrightarrow$

$\pi_{titre}(\sigma_{acteur \neq 'Deneuve'}(Film))$

```
select titre from Film where acteur<>'Deneuve';
```

- Les films qui passent dans un cinéma :
  - $\pi_{titre}(Film \bowtie Seance)$

```
select titre from Film where exists    (select * from Seance where Film.titre=Seance.titre);
```

⇄

```
select Film.titre from Film, Seance where Film.titre=Seance.titre;
```

- Les films qui ne passent dans aucun cinéma :
  - $\pi_{titre}(Film) - \pi_{titre}(Seance \bowtie Film)$

```
select Film.titre from Film where not exists
    (select * from Seance where Film.titre=Seance.titre);
```

⇄

- $\pi_{titre}(Film) - \pi_{titre}(Seance)$

```
select titre from Film where titre not in
   (select titre from Seance);
```

- Les réalisateurs dont les films passent dans tous les cinémas (au moins un film dans chaque cinéma) :
  - $\pi_{cinema,realisateur}(Seance \bowtie Film) \div \pi_{cinema}(Seance)$

```
select F.realisateur from Film F where not exists
    (select S.cinema from Seance S except
        (select S1.cinema from Seance S1, Film F1 where S1.titre=F1.titre
 and F1.realisateur=F.realisateur));
```

- Trouver les réalisateurs dont les films passent dans tous les cinémas :
  - $\pi_{realisateur}(F) - \pi_{realisateur}((\pi_{cinema}(S) \times \pi_{realisateur}(F)) - \pi_{cinema,realisateur}(F \bowtie S))$

```
select F1.realisateur from Film F1 where not exists
    (select S.cinema from Seance S where not exists
        (select F2.realisateur from Film F2 where F2.titre=S.titre and F1.realisateur=F2.realisateur
));
```

Les cinémas qui ne passent aucun film de Polanski

$\pi_{cinema}(Seance) - \pi_{cinema}(\sigma_{realisateur<>'Polanski'}(Film) \bowtie Seance)$

select cinema **from** Seance **where not exists**

(**select** * **from** Film **where** Seance.titre=Film.titre **and** realisateur='Polanski');

Trouver tous les acteurs qui ne sont pas des réalisateurs :

select acteur as personne **from** Film **except**

(**select** realisateur as personne **from** Film);

select acteur as personne **from** Film **not in**

(**select** realisateur as personne **from** Film);

Types d'attributs SQL courants

- `bool` : boolean
- `int` : entier signé sur 4 octets
- `real` : réel
- `numeric(precision, echelle)` : décimaux où `echelle` est le nombre de chiffres après la virgule, et `precision` le nombre de chiffres totaux
- `text` : chaîne de caractères
- `char(n)` : chaîne d'au plus `n` caractères
- `serial` : entier à incrémentation automatique (pratique pour les identifiants internes)
- `date` : date au format année-mois-jour
- `time` : heure au format heure-minute-seconde
- `timestamp` : combine date et heure
- `year` : stocke une année

Contraintes génériques :

- contraintes sur une seule table : `check`

```
create table Article(
    id integer primary key,
    prix numeric not null,
    prix_solde numeric,
    check(prix>prix_solde)
);
```

Remarque : il est possible de nommer une contrainte avec `constraint`

```
create table Produits(
    prod_nb integer primary key,
    prix numeric constraint prix_positif check(prix>0)
);
```

- contraintes sur plusieurs tables : `assertions SQL`

Contraintes de clé

- Une table a toujours *au moins* une clé primaire, qui est forcément `not null`

```
create table Film(titre varchar(30) primary key);create table Batiment(
    ville varchar(50),
    rue varchar(100),
    numero integer,
    #etages integer,
    primary key(ville, rue, numero)
);
```

- Les autres sont des clés candidates (ou secondaires), et ne sont pas forcément `not null`

```
create table Personne(
    nss varchar(20) primary key,
    nom varchar(50) not null,
    prenom varchar(50) not null,
    naissance integer,
    unique(nom, prenom, naissance)
);
```

Contrainte `not null`

```
create table Film(titre varchar(30) not null);
```

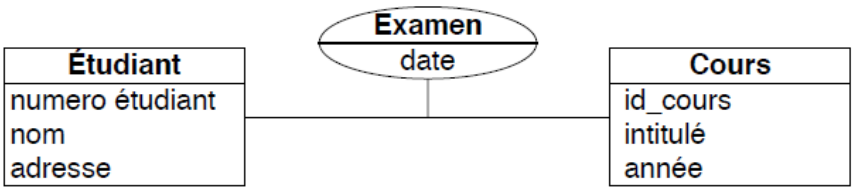
Contraintes d'identification (clés)

- Une superclé d'une entité est un ensemble d'un ou plusieurs attributs tel qu'il n'existe pas deux instances avec les mêmes valeurs pour ces attributs
- Une clé (ou clé candidate) d'une entité est une superclé minimale
- Plusieurs clés candidates peuvent exister pour une entité, mais pour chaque entité, seulement une est choisie comme clé primaire

Entités faibles

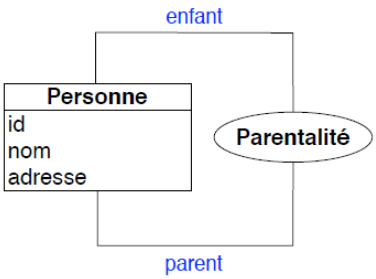
Une entité faible est identifiée par

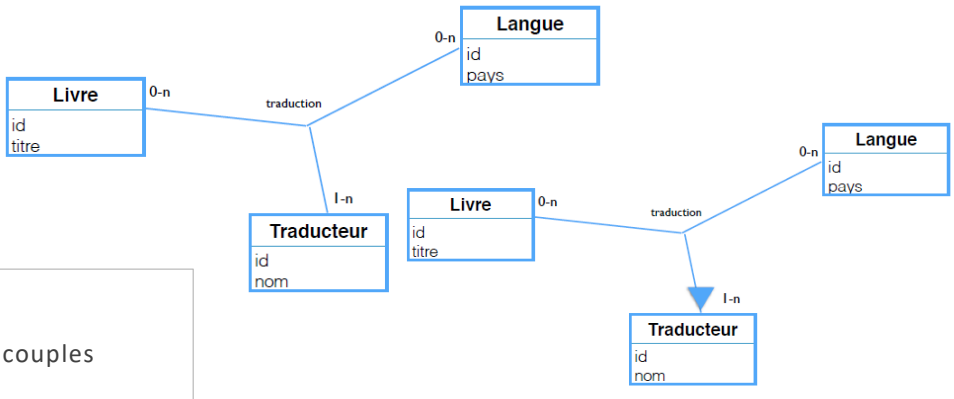
- un ensemble d'attributs internes appelés discriminant - un ensemble d'entités appelées entités identifiantes
- Chaque entité identifiante doit être reliée à l'entité faible par une association binaire appelée



- Les clés étrangères servent à relier des tables, et sont **clés primaires** de leur table

Un **attribut d'association** décrit une propriété des instances de l'association



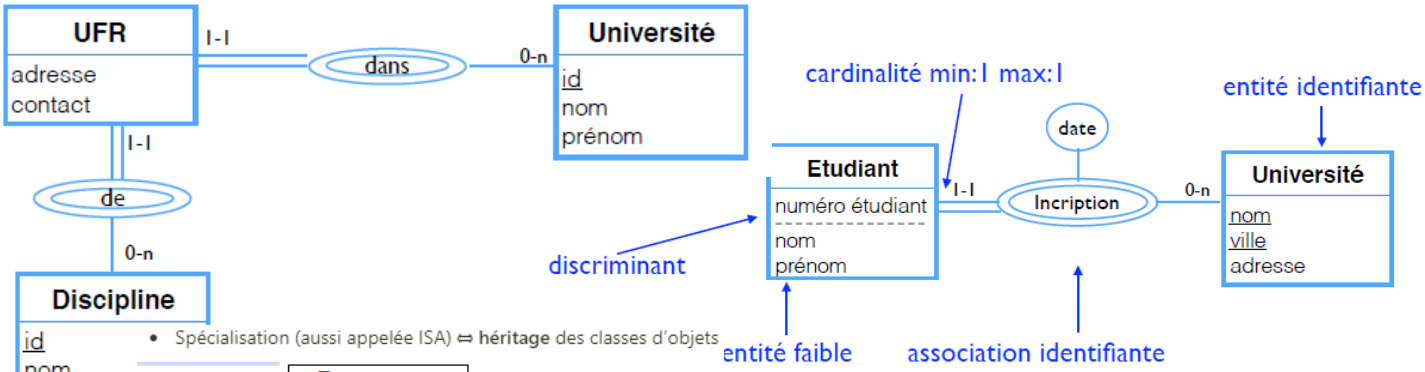


Cas n-aires

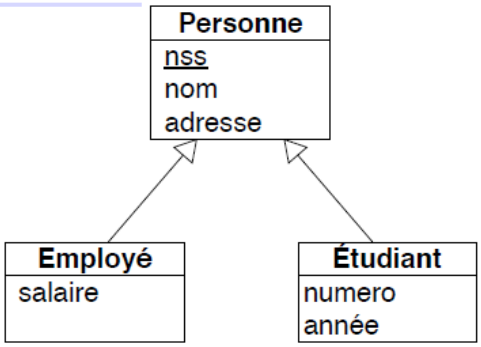
- Un livre est associé à 0 ou plusieurs couples langue-traducteur
- Une langue est assoié à 0 ou plusieurs couples livre-traducteur
- Un traducteur est associé à 1 ou plusieurs couples livre-langue

Cas n-aires avec intégrité fonctionnelle

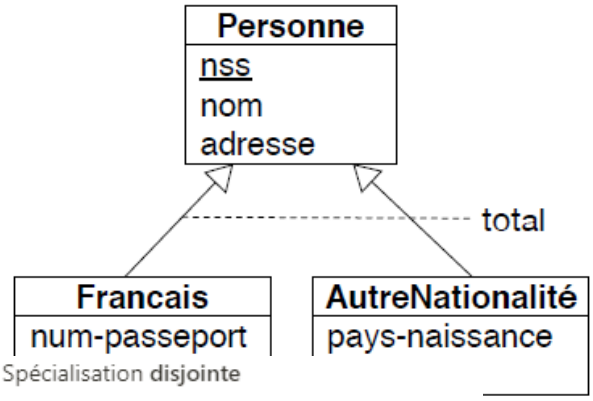
- Avec la flèche en plus, un couple livre-langue correspond au plus à 1 traducteur



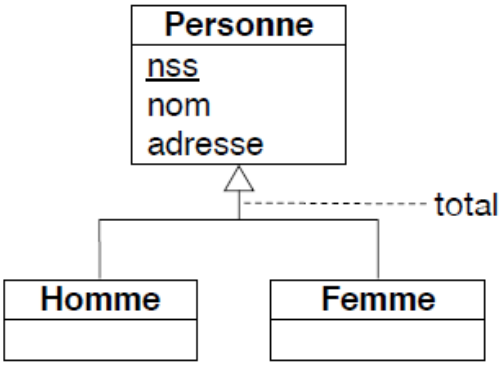
- Spécialisation (aussi appelée ISA) ⇔ **héritage** des classes d'objets

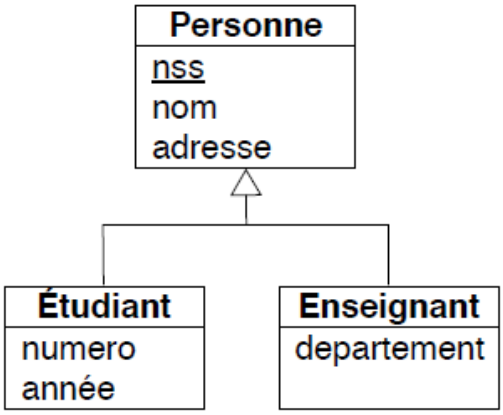


- Spécialisation **disjointe totale**
  - L'union des instances renvoie à la classe mère



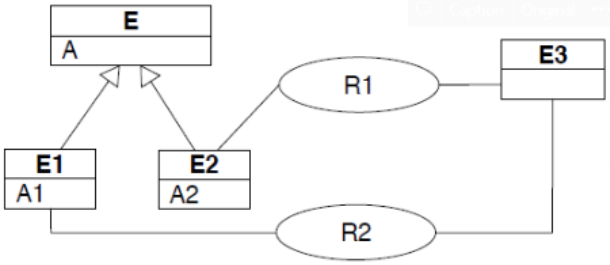
- Spécialisation **disjointe**
  - L'intersection des instances est **vide**



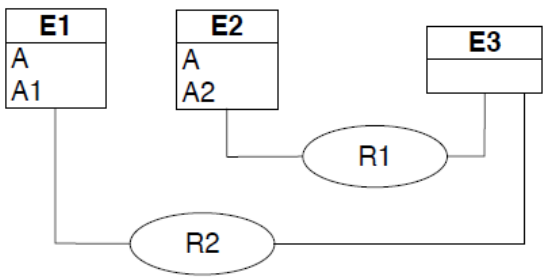


éliminer l'entité mère

- Possible uniquement si la **spécialisation est totale**
- A préférer si l'entité mère
- a peu d'attributs
- participe à peu d'associations
- est peu accédée
- Implique le dédoublement des associations de la mère

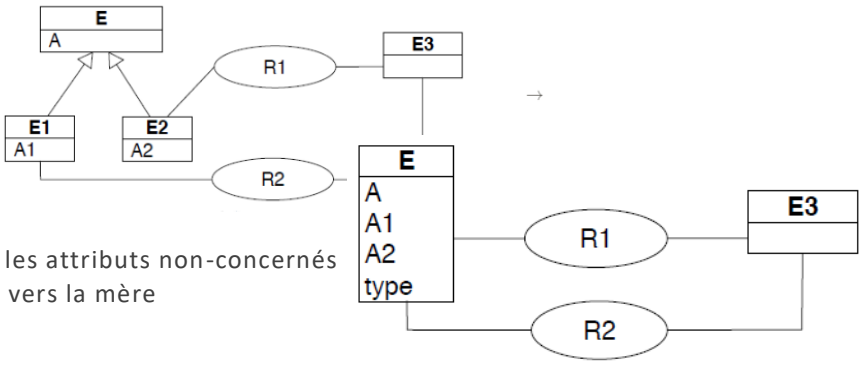


→



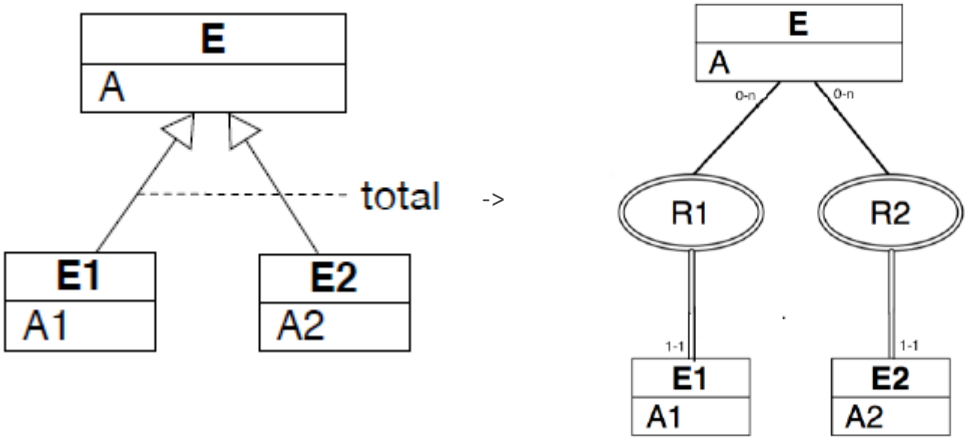
Eliminer les entités filles

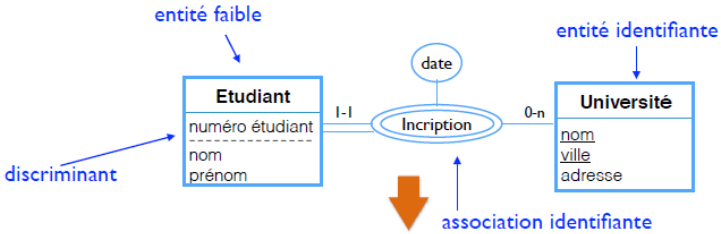
- A préférer si les entités filles
  - ont peu d'attributs
  - participent à peu d'associations
  - sont peu accédées
- Implique des valeurs non-significatives pour les attributs non-concernés
- Implique le report des associations des filles vers la mère



Maintenir toutes les entités, et simuler la spécialisation avec des associations

- Si spécialisation totale, chaque instance de E participe à R1 ou à R2
- Si spécialisation disjointe, aucune instance de E ne participe à la fois à R1 et à R2



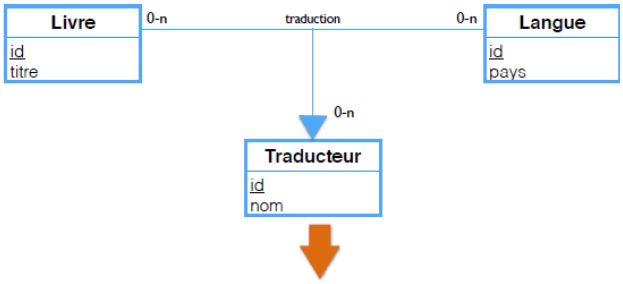


Étudiant (num-etu, nom-univ, ville-univ, nom, adresse, date-inscription)

+ Contrainte de clef étrangère :

Étudiant [nom-univ, ville-univ] ⊆ Université[nom, ville]

Association identifiante un à plusieurs



Traduction (id-livre, id-langue, id-traducteur)

+ contraintes :

Traduction[id-livre] ⊆ Livre[id]

Traduction[id-langue] ⊆ Langue[id]

Traduction[id-traducteur] ⊆ Traducteur[id]

Association n-aires

Conservation des doublons

- Union → `union all` `sql select * from S union all select * from T;`
- Intersection → `intersect all`
- Différence → `except all`

Fonctions et opérateurs prédéfinis

- `abs(num)` : valeur absolue
- `ceiling(num)` : partie entière supérieure
- `floor(num)` : partie entière inférieure
- `position(sub in string)` : position de `sub` dans `string`
- `length(str)` : longueur de `str`
- `substring(chaine [from int] [to int])`
- `str1 || str2` : concaténation des chaînes `sql select 'réalisateur : ' || nom from Artistes;`
- `now()` : date et heure courantes
- `current_date` : date courante
- `current_time` : heure courante

Comparaisons avec `like`

- `_` représente n'importe quelle lettre
- `%` représente n'importe quelle sous-chaîne (dont ∅)

Requêtes d'agrégat simple

- `min` et `max` s'appliquent aussi aux chaînes de caractères (ordre lexico)
- `sum` additionne tous les éléments d'une colonne
- `sum distinct` additionne tous les éléments distincts d'une colonne
- `count` compte les éléments d'une colonne
- `count distinct` compte les éléments distincts d'une colonne

Valeur `NULL` et opérations

- Toute opération arithmétique mettant en jeu un `NULL` à pour résultat `NULL`
  - `1+NULL` → `NULL`
  - `1-NULL` → `NULL`
  - `1*NULL` → `NULL`
  - `1/NULL` → `NULL`
  - `NULL/0` → `NULL`
- Les opérateurs d'agrégation ignorent les `NULL`
  - mais exception pour `count(*)`
- Le calcul d'un agrégat sur un multi-ensemble vide retourne `NULL` (ou `0` pour `count`)
- Dans les opérations ensemblistes, les `NULL` sont traités comme n'importe quelle autre valeur
- Le résultat des requêtes avec conditions de sélection (`where R.A=S.A`) est vide

Eviter les pièges de la valeur `NULL`

- Vérifier si la valeur est nulle ou pas avec `is null`, et non `=null`
- `coalesce` retourne le premier de ses arguments non `NULL`

```
select nom, (prix-coalesce(reduction, 0)) as prix_netfrom Article;
```

- `case` de même

```
select nom, (prix-case
    when reduction is null then 0
    else reduction
    end) as prix_net
from Article;
```

Jointures externes

- `natural left outer join` : lors de la jointure, les tuples qui n'ont pas de correspondant sont complétés avec `NULL` à droite (on garde la valeur de gauche)
- `natural right outer join` : idem, mais on garde la valeur de droite, et complète celle de gauche par `NULL`
- `natural full outer join` garde tous les tuples possibles (union des résultats de `right` et `left`)

AND	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

OR	t	f	u
t	t	t	t
f	t	f	u
u	t	u	u

	NOT
t	f
f	t
u	u

Vues

- Une vue fournit un mécanisme pour cacher ou restructurer les données accessibles à certains utilisateurs

```
create view nom_vue as <requete>;select * from nom_vue;
```

- La mise à jour des vues n'est possible que sur des vues simples, donc pas
  - de jointure
  - d'opération ensembliste
  - d'agrégation
  - de requête récursive
  - → màj ambiguës

Tables temporaires

```
with nom_table as <requete>select * from nom_table;
```

Requêtes récursives

- Forme générale de la requête du `as` :
  - terme non récursif
  - `union` ou `union all`
  - **terme récursif** pouvant contenir une référence au résultat de la requête elle même
  - `limit` pour ne pas avoir de résultats infinis avec `union all`

```
with recursive R(attribut_1, ..., attribut_n) as(
    requete_de_base(sans R)
    union [all]
    requete_recursive(avec R)
)
requete_avec_R(et autres tables)
limit 1000;
```

  - Remarque : ne pas utiliser `not in`, `except` et `not exists` dans la partie récursive

- Trouver tous les prérequis d'un cours

```
with recursive c_prereq(id_cours, id_prereq) as(
    select id_cours, id_prereq from prereq
    union
    select prereq.prereq_id, c_prereq.id_cours
    from prereq, c_prereq
    where prereq.id_cours=c_prereq.id_prereq
)
select * from c_prereq;
```

- Calculer la somme des 100 premiers entiers

```
with recursive t(n) as(
    values(1)
    union
    select n+1 from t where n<100
)
select sum(n) from t;
```

1. Calculez
   — les zones d'arrêt qui ne sont pas de location_type = 1
   — les arrêts avec location_type = 1 dont le stop_id ne commence pas par "StopArea"
   — les arrêts dont le stop_id commence pas "StopArea" qui ne sont pas des zones d'arrêts (0 résultats pour les trois requêtes)

   (On en déduit qu'il y a trois façons alternatives de reconnaître les zones d'arrêts :
   — les arrêts qui sont parent d'un autre arrêt ;
   — les arrêts qui ont location_type = 1 ;
   — les arrêts dont le stop_id commence par "StopArea").

```
\! echo requete 1
select stop_id from stops where parent_station is null and location_type <> 1;
select stop_id from stops where location_type =1 and stop_id not like 'StopArea%';
select stop_id from stops where stop_id like 'StopArea%' and parent_station is not null
```

2. Quelles sont les zones d'arrêts présents dans la table stop_times ?

```
\! echo requete 2
select stop_times.stop_id
from stop_times join stops on (stop_times.stop_id = stops.stop_id)
where parent_station is null;
```

3. Quels sont les stop_id des vrais arrêts (c'est-à-dire des arrêts qui ne représentent pas des zones d'arrêts) ?

```
\! echo requete 3
select stop_id from stops where parent_station is not null; //null pour les zones d'arrets
```

5. Quels sont les arrêts de transport en commun qui se trouvent dans la zone de la "GARE DE BAGNEUX" ? Pour chaque arrêt, renvoyer son identifiant et son nom. Les résultats doivent être triés par nom

```
\! echo requete 5
CREATE INDEX stops_stop_name_idx ON stops USING btree (stop_name);
CREATE INDEX stops_parent_station_idx ON stops USING btree (parent_station);
-- evitent les jointures couteuses

select s.stop_id,s.stop_name
from stops s, stops s0
where s0.stop_name='GARE DE BAGNEUX' and
s.parent_station=s0.stop_id
order by s.stop_name;
```

6. Quels sont les lignes de transport en commun (c-à-d les routes) qui ont un arrêt dans la zone de la "GARE DE BAGNEUX" ? Pour chaque ligne, renvoyer le nom long de la ligne. Les résultats doivent être triés.

```
\! echo requete 6
CREATE INDEX stop_times_stop_id_idx ON stop_times USING btree (stop_id);
-- pour aider la jointure entre s et t
CREATE INDEX trips_route_id_idx ON trips USING btree (route_id);
-- pour aider la jointure entre u et r

select distinct r.route_long_name
from stops s0, stops s, stop_times t,  trips u, routes r
where s0.stop_name='GARE DE BAGNEUX' and s.parent_station=s0.stop_id
and s.stop_id=t.stop_id and
u.trip_id=t.trip_id
and r.route_id=u.route_id
order by route_long_name;
```

7. Quels sont les arrêts accessibles en prenant un transport en commun dans la zone de la "GARE DE BAGNEUX", sans changement ? Renvoyer à la fois le nom de l'arrêt et le nom long de la ligne de transport permettant d'y accéder.

   Les résultats doivent être triés par nom de ligne, puis par nom d'arrêt. Ne pas oublier que pour qu'un arrêt s2 soit accessible depuis un arrêt s1 par une ligne de transport en commun, s2 doit suivre s1 dans le trajet de cette ligne (numéro de séquence plus élevé).

```
\! echo requete 7
CREATE INDEX stop_times_trip_id_idx ON stop_times USING btree (trip_id);
--pour aider le self join entre t1 et t2

select distinct s2.stop_name, r2.route_long_name
from stops s0, stops s1, stop_times t1, stop_times t2,  stops s2, trips u2, routes r2
where s0.stop_name='GARE DE BAGNEUX' and s1.parent_station=s0.stop_id and
s1.stop_id=t1.stop_id and
t1.trip_id=t2.trip_id and s2.stop_id=t2.stop_id
and t1.stop_sequence<t2.stop_sequence
and u2.trip_id=t2.trip_id
and r2.route_id=u2.route_id
order by route_long_name, s2.stop_name;
```

```
\! echo requete 8
select distinct s2.stop_name, r2.route_long_name
from stops s0, stops s1, stop_times t1, stop_times t2,  stops s2, trips u2, routes r2
where s0.stop_name='GARE DE BAGNEUX' and s1.parent_station=s0.stop_id and
s1.stop_id=t1.stop_id and
t1.trip_id=t2.trip_id and s2.stop_id=t2.stop_id
and t1.stop_sequence<t2.stop_sequence
and u2.trip_id=t2.trip_id
and r2.route_id=u2.route_id
and route_type = 2
order by route_long_name, s2.stop_name;
```

8. Refaire la requête précédente, en sélectionnant cette fois ci uniquement les arrêts accessibles en train (exclure bus, métro, tram etc). Se rappeler que le type de transport en commun est décrit par l'attribut route_type de la table routes.

```
-- StopArea:8775868 = GARE DE BAGNEUX
WITH RECURSIVE accessible(stop_area,n_trains) AS (
VALUES('StopArea:8775868',0)
UNION
SELECT s2.parent_station, n_trains + 1
FROM accessible, stops s1, stops s2,
stop_times t1, stop_times t2, trips u2, routes r2
where
s1.parent_station=accessible.stop_area and
s1.stop_id=t1.stop_id and t1.trip_id=t2.trip_id and s2.stop_id=t2.stop_id
and t1.stop_sequence < t2.stop_sequence
and u2.trip_id=t2.trip_id and r2.route_id=u2.route_id and route_type=2
and  n_trains < 2
)
SELECT stop_name, n_trains
from accessible,stops WHERE accessible.stop_area=stops.stop_id
ORDER BY n_trains ASC, stop_name;
```

9. Quelles sont les zones d'arrêt accessibles en prenant au plus deux trains successifs (c-à-dire au plus un changement) depuis la zone de la "GARE DE BAGNEUX" ? Remarquer que les lignes empruntées doivent être uniquement des trains (on ne s'intéresse pas aux autres modes de transport). Remarquer également qu'un changement peut avoir lieu entre deux arrêts quelconques qui se trouvent dans la même zone (i.e. qui ont la même station parent).

Pour cette requête vous pouvez utiliser l'information que la "GARE DE BAGNEUX" en tant que zone d'arrêt porte le stop_id "StopArea:8775868". Renvoyer pour chaque résultat le nom de la zone d'arrêt ainsi que le nombre de trains successifs qui permettent d'y accéder depuis la zone de la "GARE DE BAGNEUX" (0, 1 ou 2). Les résultats doivent être triés par nombre de trains, puis par nom de zone.

10. Refaire la requête précédente, mais renvoyer cette fois-ci, pour chaque zone d'arrêt dans le résultat, son nom ainsi que le nombre minimum de trains successifs qui permettent d'y accéder. Les résultats doivent être triés par nombre minimum de trains, puis par nom de zone.

```
\! echo requete 10
WITH RECURSIVE accessible(stop_area,n_trains) AS (
VALUES('StopArea:8775868',0)
UNION
SELECT s2.parent_station, n_trains + 1
FROM accessible, stops s1, stops s2,
stop_times t1, stop_times t2, trips u2, routes r2
where
s1.parent_station=accessible.stop_area and
s1.stop_id=t1.stop_id and t1.trip_id=t2.trip_id and s2.stop_id=t2.stop_id
and t1.stop_sequence < t2.stop_sequence
and u2.trip_id=t2.trip_id and r2.route_id=u2.route_id and route_type=2
and  n_trains < 2
)
SELECT stop_name, min(n_trains) as n
from accessible,stops WHERE accessible.stop_area=stops.stop_id
group by stop_area, stop_name
ORDER BY n ASC, stop_name;
```

```
\! echo requete 11

WITH RECURSIVE accessible(stop_area,n_trains, seq) AS (
VALUES('StopArea:8775868',0, '')
UNION
SELECT s2.parent_station, n_trains + 1, seq ||
|| case seq when '' then '' else ' ' end || route_long_name
FROM accessible, stops s1, stops s2,
stop_times t1, stop_times t2, trips u2, routes r2
where
s1.parent_station=accessible.stop_area and
s1.stop_id=t1.stop_id and t1.trip_id=t2.trip_id and s2.stop_id=t2.stop_id
and t1.stop_sequence < t2.stop_sequence
and u2.trip_id=t2.trip_id and r2.route_id=u2.route_id and route_type=2
and  n_trains < 2
)
SELECT stop_name, seq
from accessible A,stops
WHERE A.stop_area=stops.stop_id
and n_trains = (select min(n_trains) from accessible where stop_area = A.stop_area)
ORDER BY seq, stop_name;
```

11. Refaire la requête précédente mais renvoyer cette fois-ci, pour chaque zone d'arrêt dans le résultat, son nom ainsi que les séquences possibles de lignes de train utilisables pour y accéder (utilisant un nombre minimum de trains). On pourra représenter une séquence de lignes de train par la concaténation des noms longs de la ligne, séparés par des espaces. Attention à ne pas commencer une séquence par un espace. Les résultats doivent être triés par séquence de trains, ensuite par nom de gare.

12. Refaire la requête précédente mais renvoyer cette fois-ci, pour chaque zone d'arrêt dans le résultat, la concaténation de toutes les séquences possibles de lignes de train utilisables pour y accéder (utilisant un nombre minimum de trains).
Utiliser la fonction d'agrégation array_agg pour agréger les séquences possibles pour un même arrêt. Les résultats doivent être triés par nom de gare.

```
SELECT stop_name, array_agg(seq)
from accessible A,stops
WHERE A.stop_area=stops.stop_id
and n_trains = (select min(n_trains) from accessible where stop_area = A.stop_area)
group by stop_area, stop_name
ORDER BY stop_name;
```

```
\! echo requete 13

CREATE INDEX routes_route_type_idx ON routes USING btree (route_type); -- amelioration moderée
WITH RECURSIVE accessible(stop_area,distance) AS (
VALUES('StopArea:8775868',0)
UNION
SELECT s2.parent_station, distance + t2.stop_sequence-t1.stop_sequence
FROM accessible, stops s1, stops s2, stop_times t1, stop_times t2, trips u2, routes r2
where
s1.stop_id=t1.stop_id and t1.trip_id=t2.trip_id and s2.stop_id=t2.stop_id
and s1.parent_station=stop_area
and t1.stop_sequence < t2.stop_sequence
and u2.trip_id=t2.trip_id and r2.route_id=u2.route_id
and (route_type = 0 or route_type = 1 or route_type = 2 or route_type = 7)
and distance + t2.stop_sequence-t1.stop_sequence <= 3 )
```

13. Quels sont les zones d'arrêt accessibles depuis la zone de la GARE DE BAGNEUX en effectuant au plus 3 arrêts en tout (destination incluse, origine exclue) ?

On peut utiliser uniquement des transports en commun sur rail, et les changements de ligne sont autorisés (un changement de ligne compte comme un arrêt).

```
SELECT
stop_name,min(distance)
from accessible,stops
WHERE accessible.stop_area=stops.stop_id
GROUP BY stop_id, stop_name
ORDER BY min(distance) ASC, stop_name;
```