SQL
et
Bases de données

Cours 5

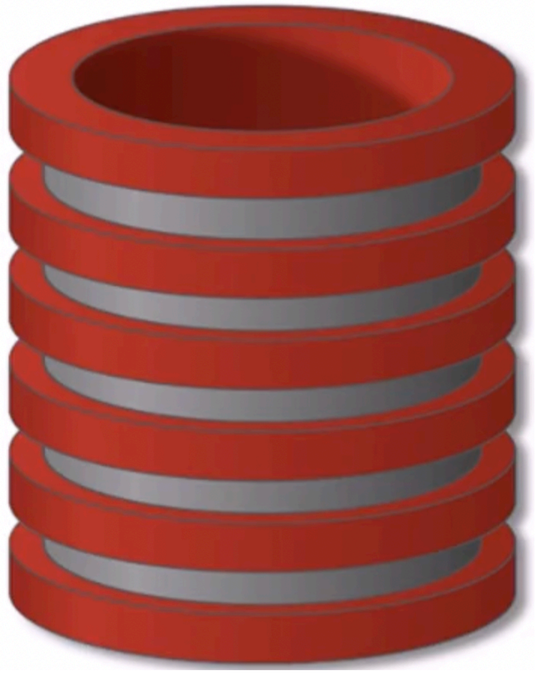
Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/lp-sql

Plan

- jokers et filtrage
- join et contraintes
- indexation



- deux bons tutoriels

  http://www.w3schools.com/sql/default.asp

  http://www.programiz.com/sql

client

cID	nom	actif	ville
1	Tom	23000	Bordeaux
2	Jean-Jacques	38000	Paris
3	Martin	51000	Nice
4	Kiki	54000	Pekin
5	Iteki	84000	Tokyo
6	Bob	6100	Nice
7	Albert	12000	Bordeaux
8	Manu	8150	Paris
9	Valou	10300	Bordeaux
10	Joe	32500	Nice
11	Helmut	8150	Paris
12	Martine	11200	Bordeaux
13	Marina	9150	Nice
14	Masha	10290	Nice
15	Julia	32000	Paris
17	Bob	38000	Nice

produit

pID	pNom	pCat	pVille	prix
1	clio	auto	Paris	13000
2	audi	auto	Paris	45000
3	tesla	auto	Pekin	70000
4	tesla	auto	Nice	40000
5	yamaha	moto	Tokyo	8000
6	kawasaki	moto	Tokyo	8000
7	megamo	velo	Paris	3240
8	shimano	velo	Paris	1900
9	btwin	velo	Nice	990
10	triban	velo	Nice	690
11	peugeot	velo	Paris	750
12	bertin	eVelo	Paris	1190
13	trek	eVelo	Bordeaux	1390
14	trek	eVelo	Paris	1350

devis

cID	pNom	dCat	commande
2	peugeot	velo	0
7	NULL	velo	0
6	trek	eVelo	0
8	NULL	auto	0
8	NULL	velo	0
8	NULL	eVelo	0
9	honda	auto	0
11	NULL	auto	0
4	honda	moto	0
5	NULL	moto	0
8	triban	velo	0
13	NULL	velo	0
11	yaris	auto	0
12	NULL	velo	0
1	NULL	eVelo	0

Solutions des exercices

- voir la fin des transparents du cours 4

- lister les catégories de produits avec des devis dont les actifs des acheteurs sont inférieurs à la moyenne des actifs

  [« les les catégories de produits achetés par les plus pauvres »]

```
select dCat, actifCatMoyen
from
  (select dCat, avg(actif) as actifCatMoyen
   from
     (select distinct dCat, actif, client.cID
     from devis, client
     where client.cID = devis.cID
     order by dCat, client.cID)
     group by dCat)
   where actifCatMoyen <
     (select avg(actif) from client);
```

dCat	actifCatMoyen
auto	19225.0
eVelo	22416.6666666667
velo	31700.0

Jokers

- on peut tester partiellement une valeur avec `LIKE`

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

[caractères spéciaux: ne pas utiliser `_` et `%` pour un caractère quelconque ou une série de tels caractères]

- exemple des clients dont le nom commence par la lettre M:

```
select * from client
where nom like 'M%';
```

JOIN

- JOIN pour relier deux tables (déjà fait avec WHERE)

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

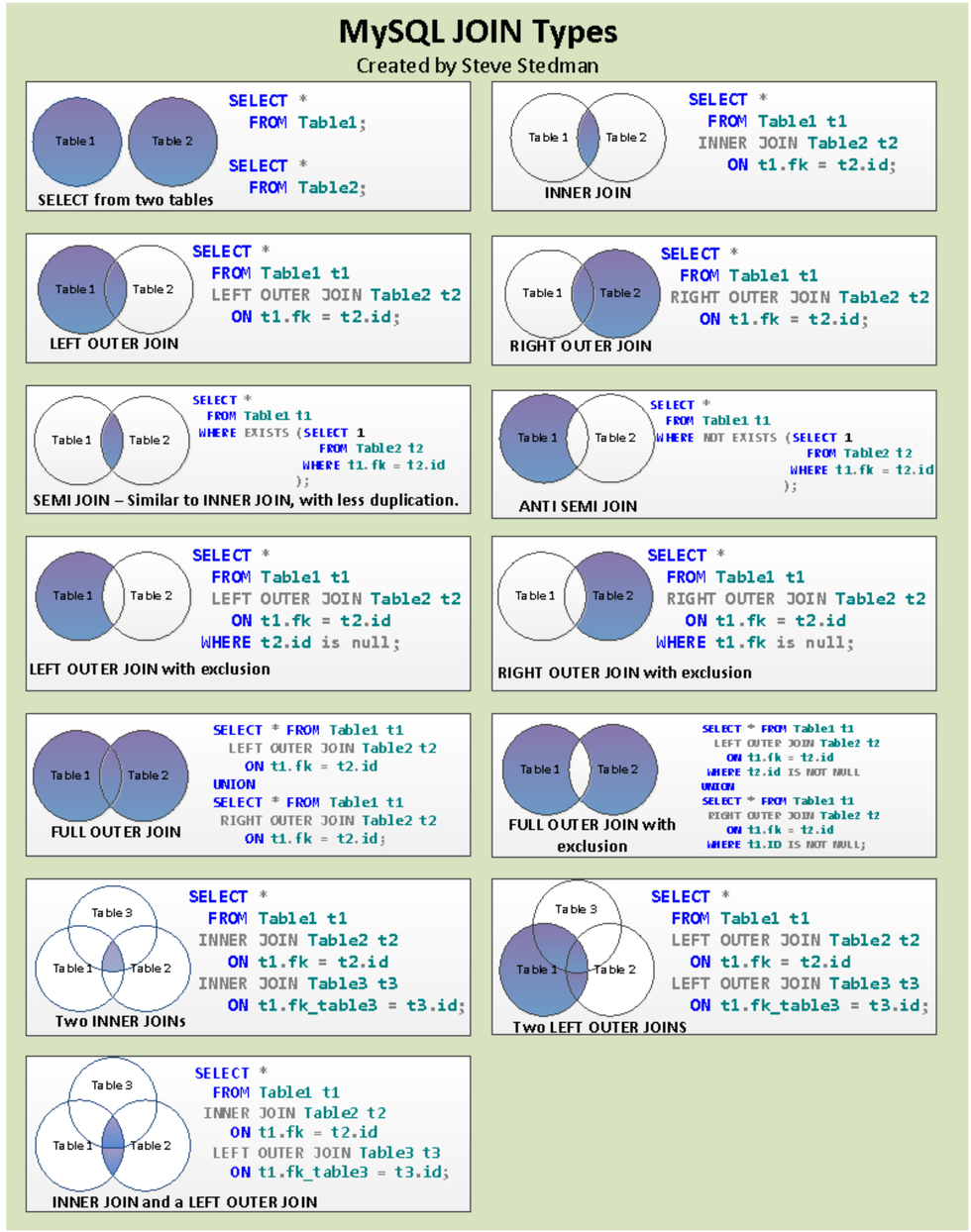
- exemple:

```
select *
from devis
inner join client on
devis.cID = client.cID;
```

- JOIN peut être INNER, OUTER, LEFT, RIGHT selon la présence de valeurs NULL

JOIN

- JOIN sous diverses formes avec MySQL (certaines fonctionnalités ne marchent pas en sqlite)



Contraintes

- contraintes ajoutées à la création (ou après `ALTER`)

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

- exemples de contraintes:

`NOT NULL` (attribut toujours non null)
`UNIQUE` (toutes les valeurs sont uniques
`PRIMARY KEY` (non null et unique)
`....`

Indexation

- les index améliorent l’efficacité des requêtes
- ce sont des données persistantes qui sont stockées avec la base de données
- plein de problèmes intéressants pour les implémenter

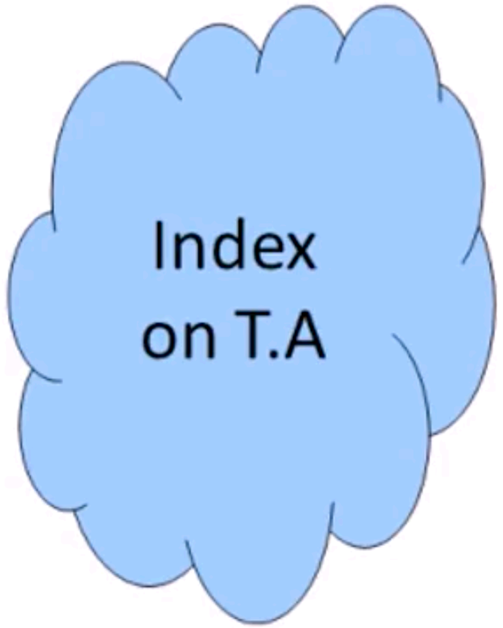
Indexation

T

	A	B	C
1	cat	2	...
2	dog	5	...
3	cow	1	...
4	dog	9	...
5	cat	2	...
6	cat	8	...
7	cow	6	...
	...	...	...

- l’utilisateur ne voit pas l’index.
- C’est le système sous-jacent qui interroge l’index

Indexation

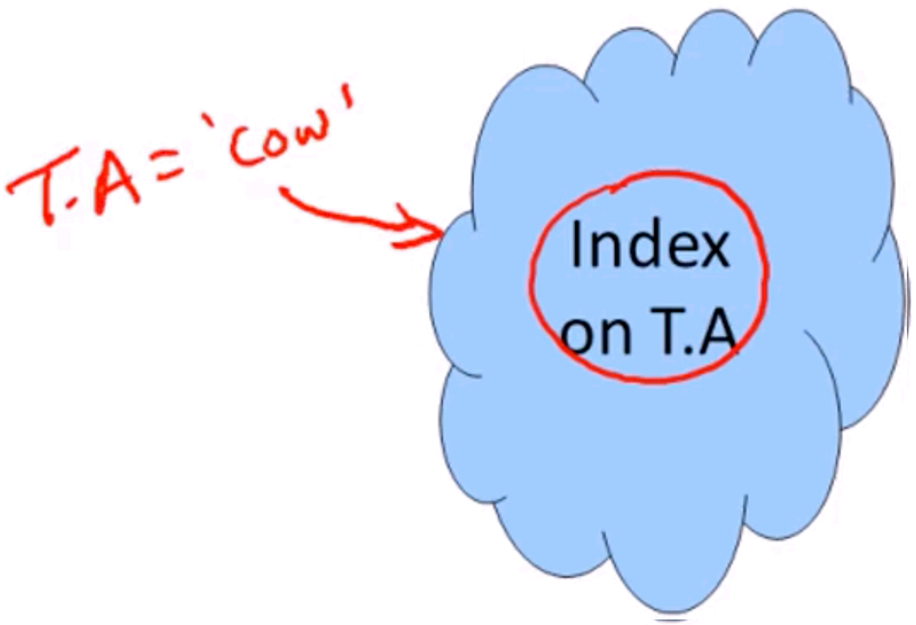


T

	A	B	C
1	cat	2	…
2	dog	5	…
3	cow	1	…
4	dog	9	…
5	cat	2	…
6	cat	8	…
7	cow	6	…
	…	…	…

- l’utilisateur ne voit pas l’index.
- C’est le système sous-jacent qui interroge l’index

Indexation

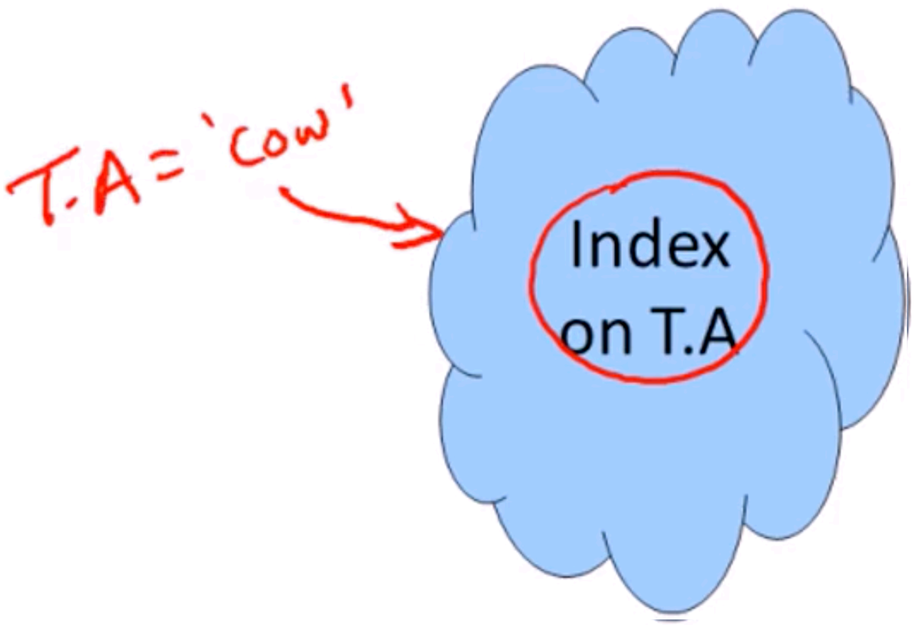


T

	A	B	C
1	cat	2	...
2	dog	5	...
3	cow	1	...
4	dog	9	...
5	cat	2	...
6	cat	8	...
7	cow	6	...
	...	...	...

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index

Indexation

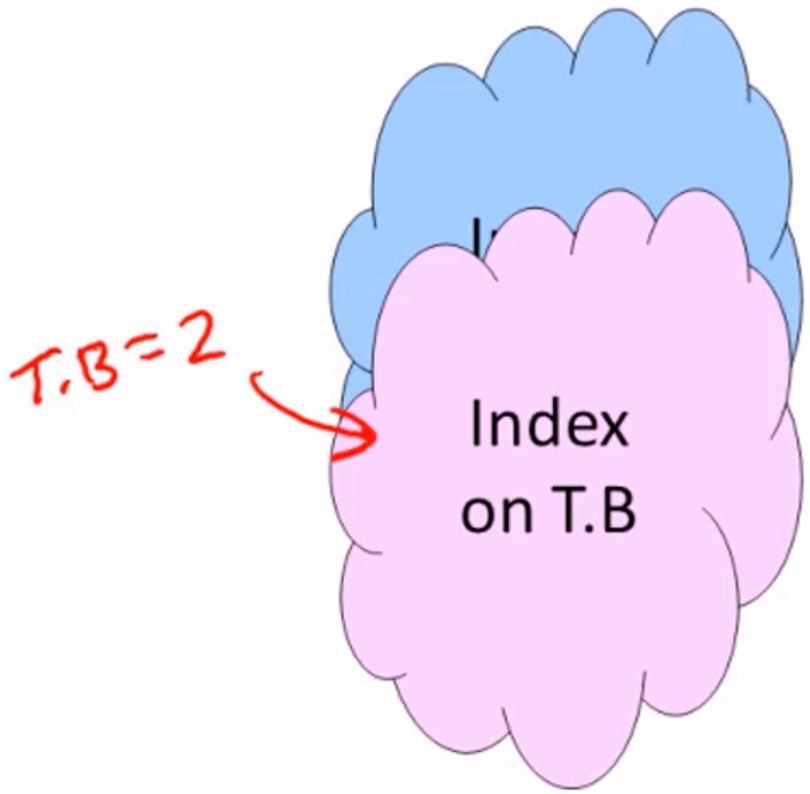


T

	A	B	C
1	cat	2	…
2	dog	5	…
3	cow	1	…
4	dog	9	…
5	cat	2	…
6	cat	8	…
7	cow	6	…
	…	…	…

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index et répond les lignes **3** et **7**

Indexation

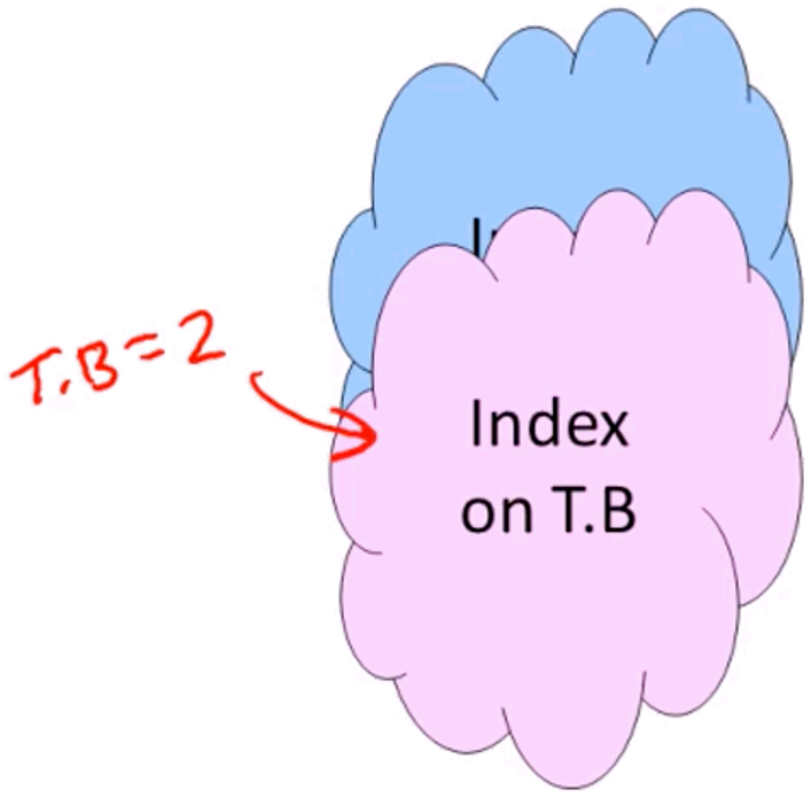


T

	A	B	C
1	cat	2	…
2	dog	5	…
3	cow	1	…
4	dog	9	…
5	cat	2	…
6	cat	8	…
7	cow	6	…
	…	…	…

- l’utilisateur ne voit pas l’index.
- C’est le système sous-jacent qui interroge l’index

Indexation

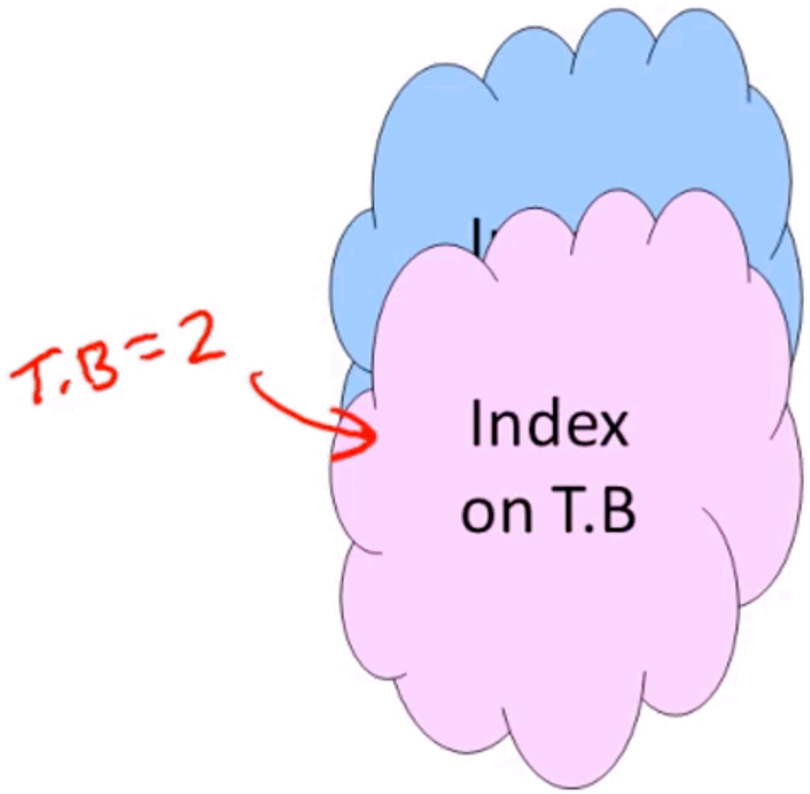


T

	A	B	C
1	cat	2	…
2	dog	5	…
3	cow	1	…
4	dog	9	…
5	cat	2	…
6	cat	8	…
7	cow	6	…
	…	…	…

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index et répond les lignes **1** et **5**

Indexation

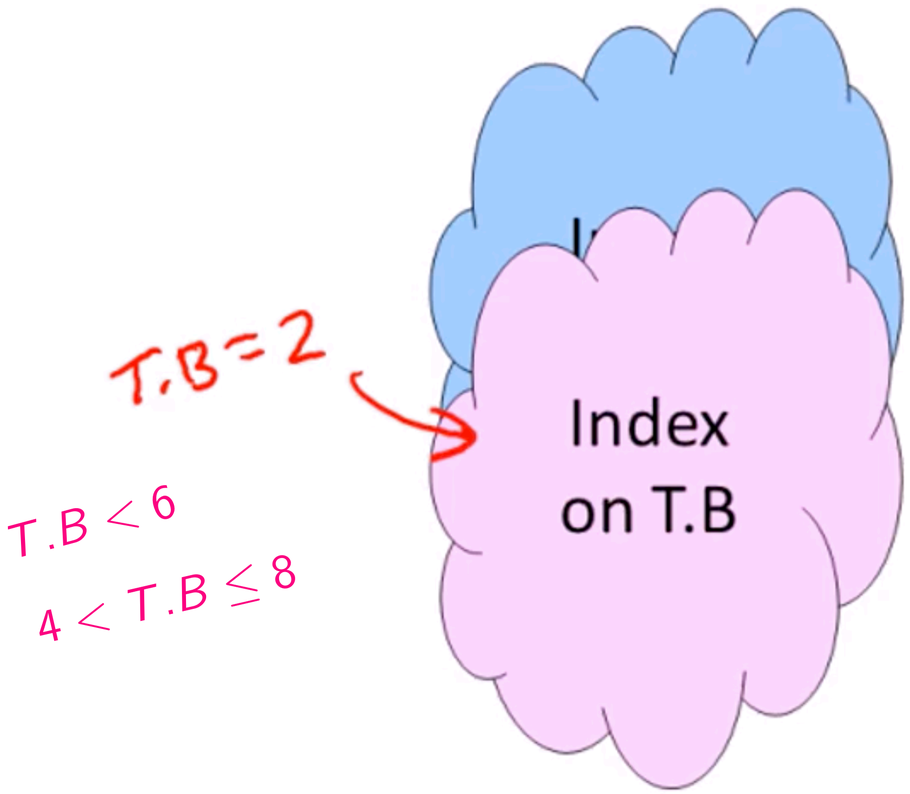


T

	A	B	C
1	cat	2	…
2	dog	5	…
3	cow	1	…
4	dog	9	…
5	cat	2	…
6	cat	8	…
7	cow	6	…
	…	…	…

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index et répond les lignes **1** et **5**
- on peut poser des questions plus compliqués avec inégalités

Indexation

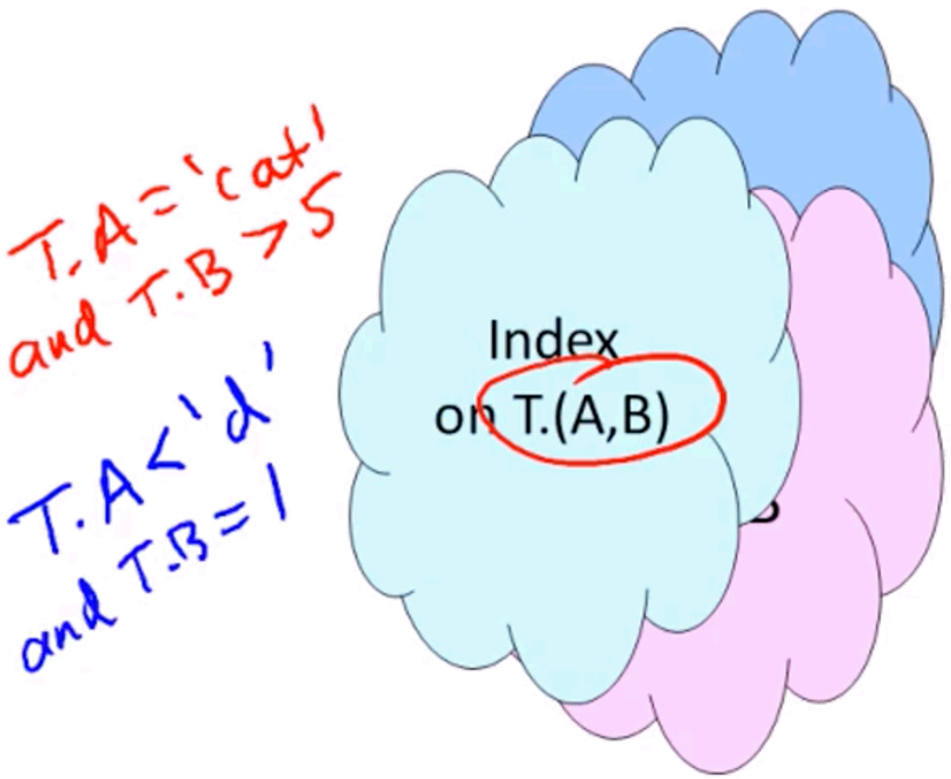


T

	A	B	C
1	cat	2	...
2	dog	5	...
3	cow	1	...
4	dog	9	...
5	cat	2	...
6	cat	8	...
7	cow	6	...
	...	...	...

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index et répond les lignes **1** et **5**
- on peut poser des questions plus compliqués avec inégalités

Indexation

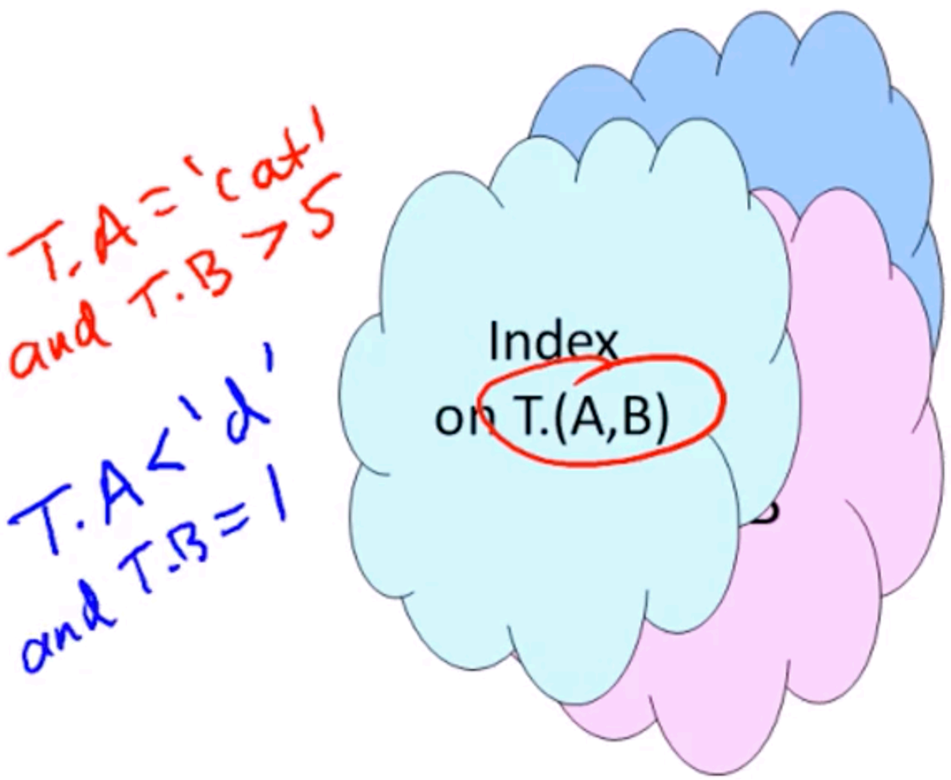


T

	A	B	C
1	cat	2	...
2	dog	5	...
3	cow	1	...
4	dog	9	...
5	cat	2	...
6	cat	8	...
7	cow	6	...
	...	...	...

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index

Indexation

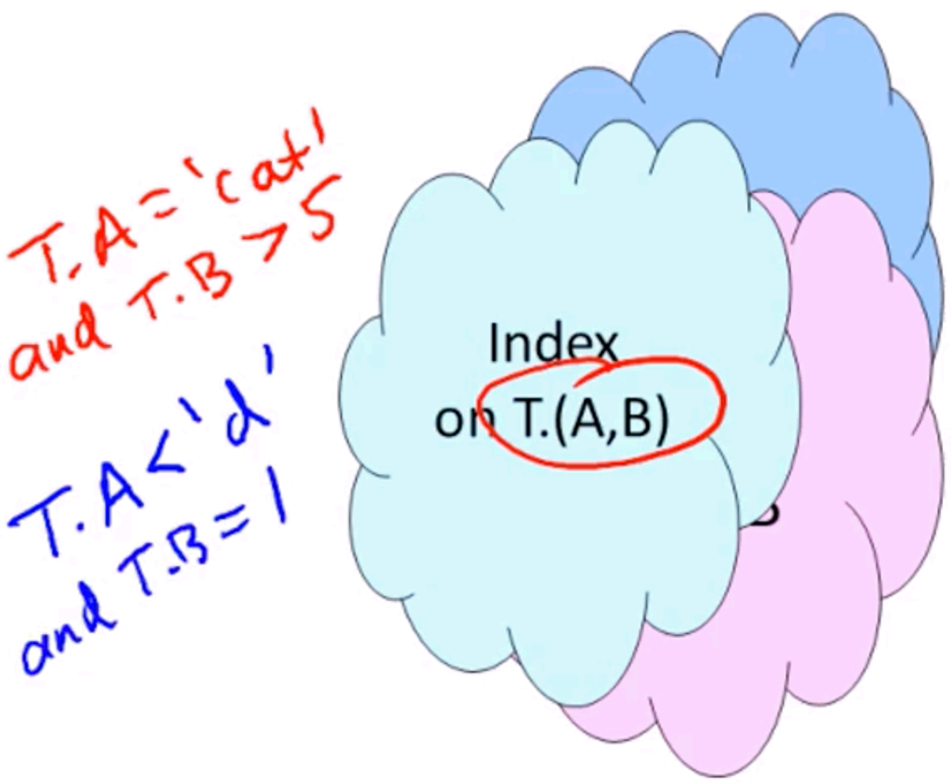


T

	A	B	C
1	cat	2	...
2	dog	5	...
3	cow	1	...
4	dog	9	...
5	cat	2	...
6	cat	8	...
7	cow	6	...
	...	...	...

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index
- on peut poser des questions encore plus compliqués

Indexation



T

	A	B	C
1	cat	2	…
2	dog	5	…
3	cow	1	…
4	dog	9	…
5	cat	2	…
6	cat	8	…
7	cow	6	…
	…	…	…

- l'utilisateur ne voit pas l'index.
- C'est le système sous-jacent qui interroge l'index
- on peut poser des questions encore plus compliqués

et répond les lignes **6** ou **3**

Indexation

- sans indexation, une requête peut devoir scanner tous les n-uplets d'une table.
  Elle prend alors un temps linéaire en la taille de la table $O(n)$

par exemple : $n = 100000$

- avec indexation, on obtient un gain d'efficacité très substantiel (un ordre de magnitude)

par exemple : $\log n = 16,6$

- on peut réaliser les indexs avec arbres B (ou B+) .
  Ce sont des arbres équilibrés qui fournissent une accélération logarithmique $O(\log n)$
  Pour requêtes avec égalités et inégalités

par exemple : 1

- on peut réaliser les indexs avec des tables de hachage
  Les réponses sont alors en temps quasi constant $O(1)$
  Pour requêtes avec égalités

- on verra plus tard ces structures de données pour réaliser des indexs

Indexation

- si requête simple

```
SELECT nom
FROM client
WHERE cID = 20813;
```

- sans index, on doit scanner toute la table client pour trouver le bon n-uplet

- avec index sur `cID`, on trouve rapidement le `nom` du n-uplet tel que `cID = 20813`

- si l'attribut est déclaré comme `PRIMARY KEY`, tous les systèmes de base de données créent automatiquement un index

Indexation

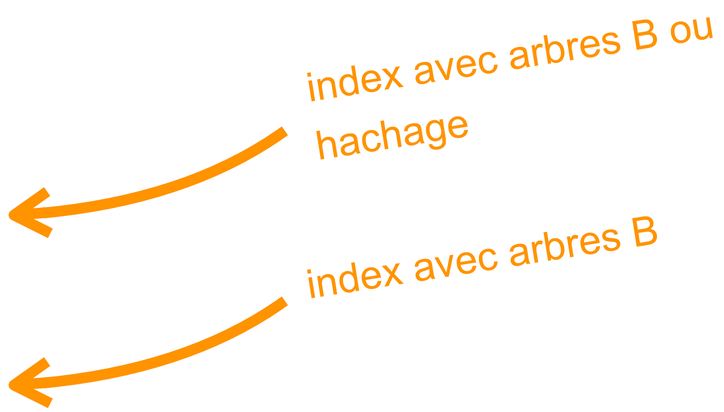
- si requête composée

```
SELECT cID
FROM client
WHERE nom = ‘Tom’ AND actif > 20000;
```

- avec index sur `nom`, on trouve rapidement les n-uplets tels que `nom = ‘Tom’`
  et on teste pour chacun si `actif > 20000`

- avec index sur `actif`, on trouve rapidement les n-uplets tels que `actif > 20000`
  et on teste pour chacun si `nom = ‘Tom’`

- avec index sur `(nom, actif)`, on trouve rapidement les n-uplets tels que
  `nom = ‘Tom’ et actif > 20000`



Indexation

- si requête avec jointure entre 2 tables

```
SELECT cID
FROM client, devis
WHERE client.cID = devis.cID
```

- avec index sur les `cID` de la table `devis`, on scanne les n-uplets de la table `client` et on teste rapidement si la table des devis contient ce `cID`

- avec index sur les `cID` de la table `client`, on scanne les n-uplets de la table `devis` et on teste rapidement si la table `client` contient ce `cID`

- on peut souvent coupler ces 2 index souvent fournis par ordre croissant et faire alors un tri par fusion

- pour des requêtes plus compliquées, les bons systèmes de bases de données recherchent la bonne utilisation des index pour optimiser les requêtes

planification des requêtes et optimisation

Indexation

Problèmes :

- place supplémentaire (peut doubler la place? grave?)
- temps à la création — beaucoup de temps, mais on ne le fait qu'une fois
- modification ralentie car modifier l'index (*trade off* entre fréquence des modifs et le temps)

Indexation

Avantages / Inconvénients :

- bénéfice selon la taille des tables
- dépend de la distribution des valeurs
- fréquence des requêtes ⟷ fréquence des modifications

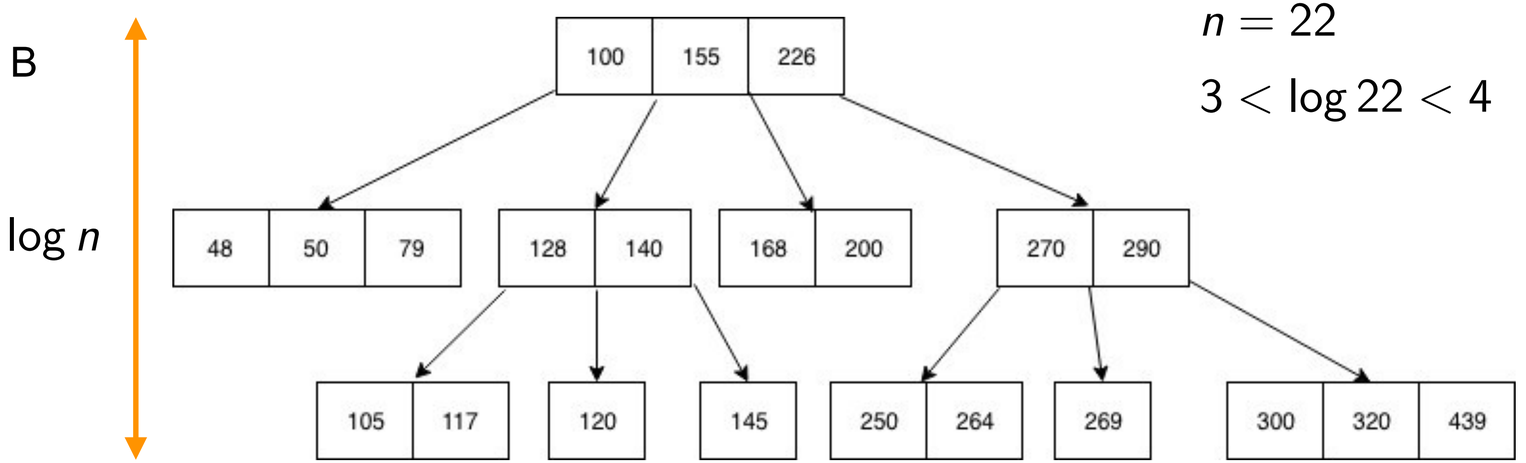
Indexation

- les systèmes de BD modernes fournissent des conseils (*design advisor*)
  - selon les statistiques et la charge (*workload*)
  - fournissent la liste des indexs recommandés

- les systèmes de BD modernes essaient donc plusieurs indexations possibles et conseillent la meilleure

Indexation

• index de *n* éléments réalisé avec un arbre B

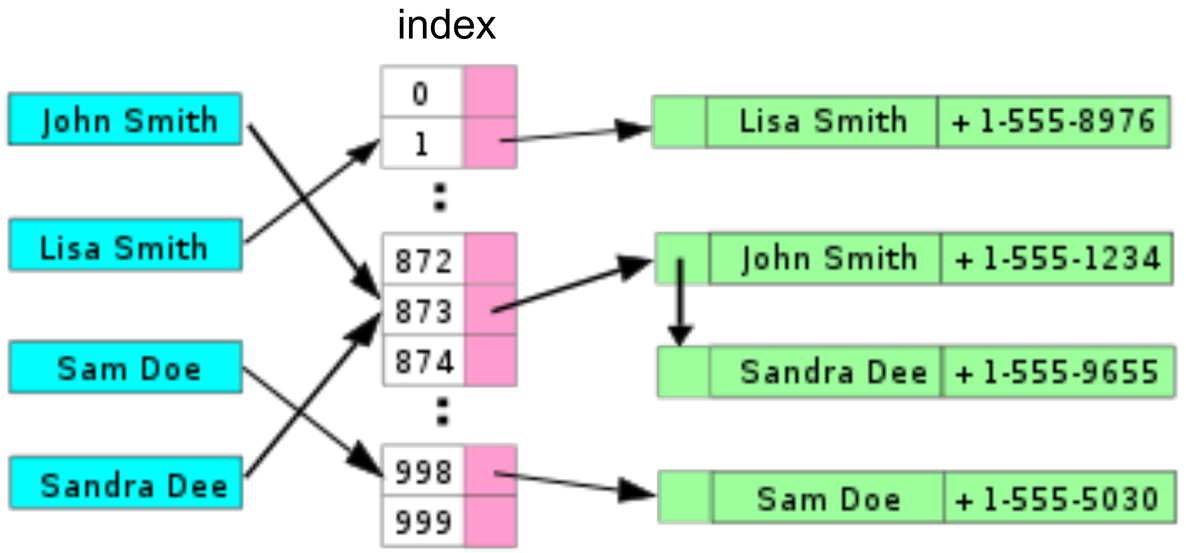


$n = 22$

$3 < \log 22 < 4$

• index de *n* éléments réalisé avec une table de hachage

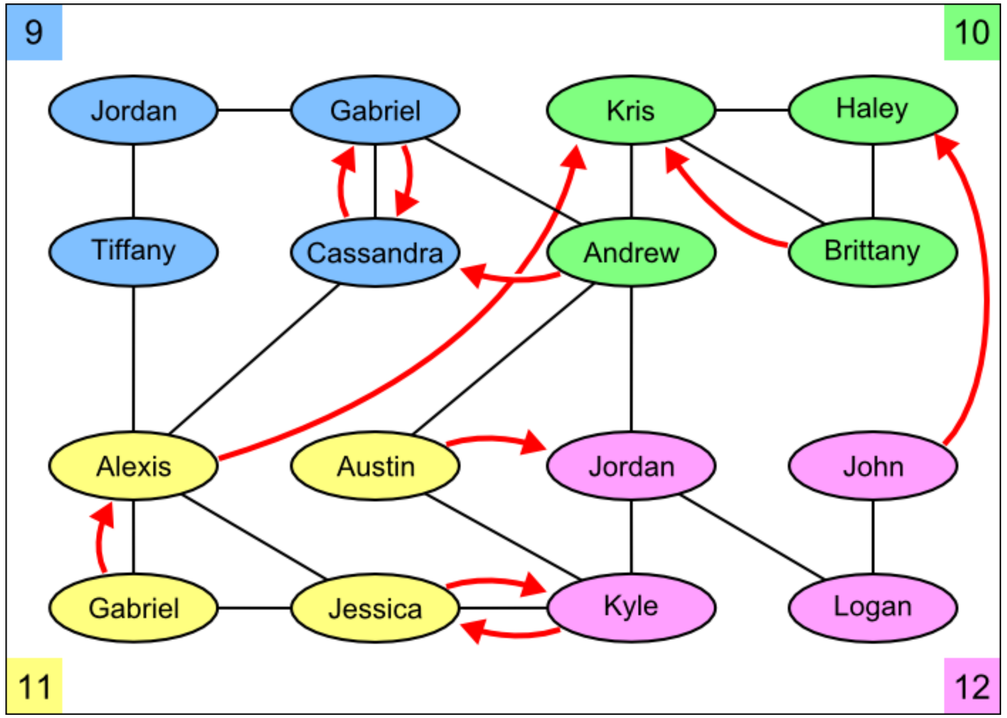
ici la fonction de hachage prend ses valeurs entre 0 et 999



Exercice

```
drop table if exists Ecolier;
drop table if exists estAmiDe;
drop table if exists Aime;

create table Ecolier(ID int, nom text, age int);
create table estAmiDe(ID1 int, ID2 int);
create table Aime(ID1 int, ID2 int);
```




```
insert into Ecolier values (1510, 'Jordan', 9);
insert into Ecolier values (1689, 'Gabriel', 9);
insert into Ecolier values (1381, 'Tiffany', 9);
insert into Ecolier values (1709, 'Cassandra', 9);
insert into Ecolier values (1101, 'Haley', 10);
insert into Ecolier values (1782, 'Andrew', 10);
insert into Ecolier values (1468, 'Kris', 10);
insert into Ecolier values (1641, 'Brittany', 10);
insert into Ecolier values (1247, 'Alexis', 11);
insert into Ecolier values (1316, 'Austin', 11);
insert into Ecolier values (1911, 'Gabriel', 11);
insert into Ecolier values (1501, 'Jessica', 11);
insert into Ecolier values (1304, 'Jordan', 12);
insert into Ecolier values (1025, 'John', 12);
insert into Ecolier values (1934, 'Kyle', 12);
insert into Ecolier values (1661, 'Logan', 12);
```

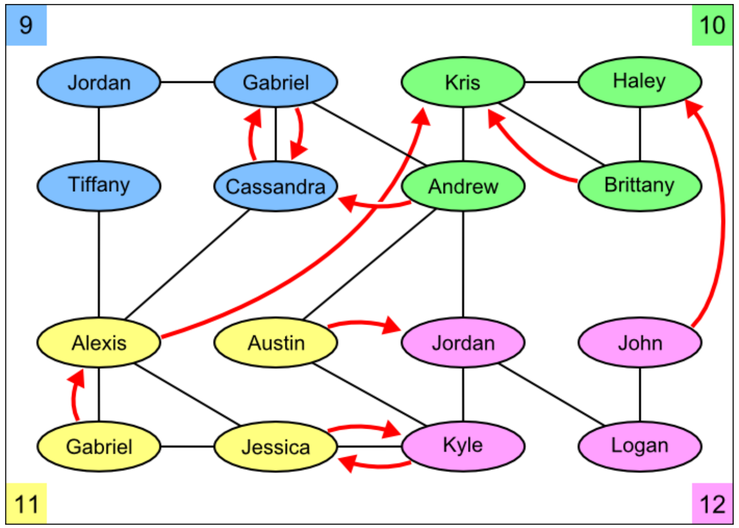
```
insert into estAmiDe values (1510, 1381);
insert into estAmiDe values (1510, 1689);
insert into estAmiDe values (1689, 1709);
insert into estAmiDe values (1381, 1247);
insert into estAmiDe values (1709, 1247);
insert into estAmiDe values (1689, 1782);
insert into estAmiDe values (1782, 1468);
insert into estAmiDe values (1782, 1316);
insert into estAmiDe values (1782, 1304);
insert into estAmiDe values (1468, 1101);
insert into estAmiDe values (1468, 1641);
insert into estAmiDe values (1101, 1641);
insert into estAmiDe values (1247, 1911);
insert into estAmiDe values (1247, 1501);
insert into estAmiDe values (1911, 1501);
insert into estAmiDe values (1501, 1934);
insert into estAmiDe values (1316, 1934);
insert into estAmiDe values (1934, 1304);
insert into estAmiDe values (1304, 1661);
insert into estAmiDe values (1661, 1025);
insert into estAmiDe select ID2, ID1
                        from estAmiDe;
```

```
insert into Aime values(1689, 1709);
insert into Aime values(1709, 1689);
insert into Aime values(1782, 1709);
insert into Aime values(1911, 1247);
insert into Aime values(1247, 1468);
insert into Aime values(1641, 1468);
insert into Aime values(1316, 1304);
insert into Aime values(1501, 1934);
insert into Aime values(1934, 1501);
insert into Aime values(1025, 1101);
```

on peut la charger en http://jeanjacqueslevy.net/lp-sql/ecole.db

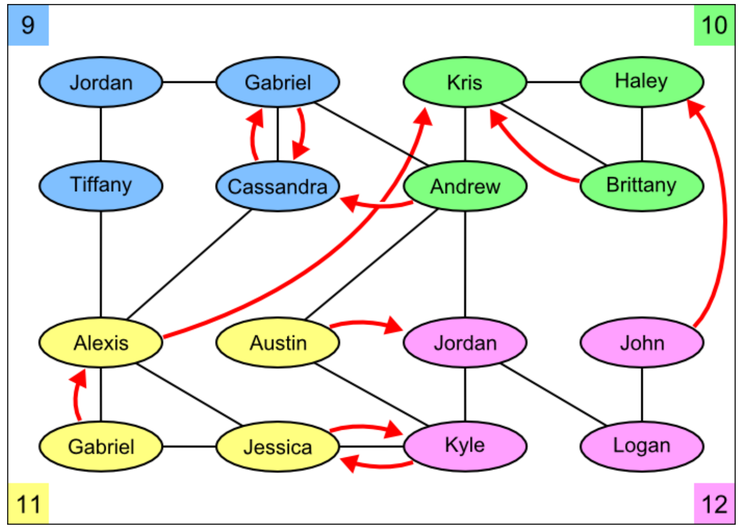
Exercices

- **Q1** Trouver les amis de Gabriel.
- **Q2** Trouver les noms des écoliers qui aiment un écolier de 2 ans ou plus jeune qu'eux.
- **Q3** Imprimer le nom et l'âge de toute paire d'écoliers qui s'aiment mutuellement.
- **Q4** Imprimer (trié par âge, puis par nom) les écoliers qui n'apparaissent pas dans la table Aime.
- **Q5** Imprimer les paires d'écoliers A et B tels que A aime B dont on ne sait rien sur ceux qu'aime B.
- **Q6** Imprimer les noms et âges de ceux qui n'ont pas d'amis du même âge. Imprimer les triés par âge
- **Q7** Pour tout écolier A qui aime un écolier B mais qui ne sont pas amis, trouver un ami commun C (qui pourrait les mettre en relation). Imprimer les noms de A, B, C et leurs âges.
- **Q8** Trouver la différence entre le nombre d'écoliers et le nombre de noms différents.
- **Q9** Trouver le nom et l'âge de tous les écoliers qui sont aimés par un autre écolier.



Exercices

- **Q1** Pour tout écolier A qui aime B, mais B aime un écolier différent C, imprimer les noms et âges de A, B, C
- **Q2** Trouver les écoliers dont les amis ont tous des âges différents d'eux.
- **Q3** Quel est l'âge moyen des amis par écolier ?
- **Q4** Trouver le nombre d'écoliers amis de Cassandra ou amis d'amis de Cassandra.
- **Q5** Trouver le nom et l'âge de l'écolier avec le plus grand nombre d'amis.
- **Q6** Supprimer les plus vieux écoliers de la table des écoliers
- **Q7** Supprimer les paires d'écoliers A et B où A aime B mais B n'aime pas A..
- **Q8** Ajouter les paires A et C où A aime B et B aime C.
  (Les amis de mes amis sont mes amis — Attention aux répétitions !)



Prochains cours

- indexation (arbres de recherche équilibrés, hachage)
- introduction au requêtes concurrentes
- transactions
- correspondance avec la logique du 1er ordre
- interface avec Python, HTML et Javascript