SQL

LE LANGAGE SQL ET LES BASES DE DONNÉES

- LES JOINTURES EN SQL
- LES SOUS-REQUÊTES EN SQL
- NOTION D'INDEX

JOINTURES SQL

Les **jointures** en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

Avec le SGBD **MySQL** il existe **2 sortes de jointure**, les **jointures internes** et les **jointures externes (jointures à gauche et jointures à droite)**.

Jointure interne : INNER JOIN

Dans le langage SQL la commande **INNER JOIN** est un type de jointures très commun pour lier plusieurs tables entre-elles. Cette commande retourne les enregistrements lorsqu'il y a **au moins** une ligne dans chaque colonne qui correspond à la condition.

```
SELECT *
FROM table1
INNER JOIN table2
ON table1.id = table2.fk_id
```

JOINTURES SQL …

Jointure interne : INNER JOIN …

La syntaxe suivante peut aussi être rencontrée même si je vous conseille d'utiliser plutôt la syntaxe précédente.

```
SELECT *
FROM table1, table2
WHERE table1.id = table2.fk_id
```

JOINTURES SQL …

Jointure interne : INNER JOIN …

Table utilisateur

id	prenom	nom	email	ville
1	Aimée	Marechal	aime.marechal@example.com	Paris
2	Esmée	Lefort	esmee.lefort@example.com	Lyon
3	Marine	Prevost	m.prevost@example.com	Lille
4	Luc	Rolland	lucrolland@example.com	Marseille

Table commande

utilisateur_id	date_achat	num_facture	prix_total
1	20130123	A00103	203.14
1	20130214	A00104	124.00
2	20130217	A00105	149.45
2	20130221	A00106	235.35
5	20130302	A00107	47.58

Pour afficher **toutes les commandes** associées aux **utilisateurs**, il est possible d'utiliser la requête suivante :

JOINTURES SQL …

Jointure interne : INNER JOIN …

```
SELECT id, prenom, nom, date_achat, num_facture, prix_total
FROM utilisateur
INNER JOIN commande ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	date_achat	num_facture	prix_total
1	Aimée	Marechal	20130123	A00103	203.14
1	Aimée	Marechal	20130214	A00104	124.00
2	Esmée	Lefort	20130217	A00105	149.45
2	Esmée	Lefort	20130221	A00106	235.35

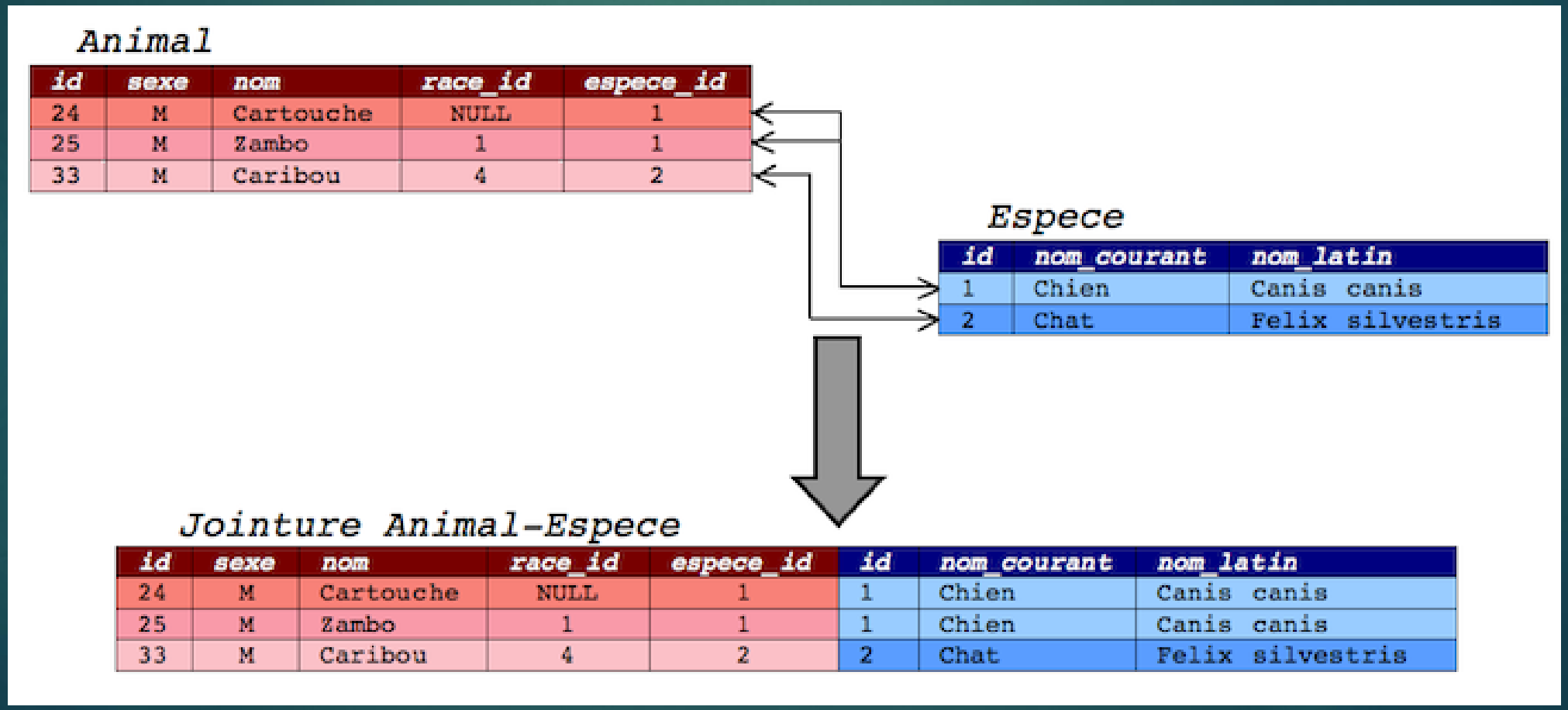
Le résultat de la requête montre la jointure entre les 2 tables. Les utilisateurs 3 et 4 ne sont pas affichés puisqu'il n'y a pas de commandes associés à ces utilisateurs.

Attention : il est important de noter que si un utilisateur à été supprimé, alors on ne verra pas ses commandes dans la liste puisque **INNER JOIN** retourne uniquement les résultats ou la condition est vraie dans les 2 tables.

JOINTURES SQL …

Jointure interne : INNER JOIN …

En fait, lorsque l'on fait une **jointure**, on crée une **table virtuelle** et **temporaire** qui reprend les colonnes des tables liées. Le schéma ci-dessous illustre ce principe :



JOINTURES SQL …

Jointure externe : LEFT OUTER JOIN

Dans le langage SQL, la commande **LEFT OUTER JOIN** est un type de jointure entre 2 tables. Cela permet de lister tous les résultats de **la table de gauche** (left = gauche) **même s'il n'y a pas de correspondance dans la deuxième table**.

Pour lister les enregistrement de **table1**, même s'il n'y a pas de correspondance avec **table2**, il convient d'effectuer une requête SQL utilisant la syntaxe suivante.

```
SELECT *
FROM table1
LEFT OUTER JOIN table2 ON table1.id = table2.fk_id
```

Cette requête est particulièrement intéressante pour récupérer les informations de **table1** tout en récupérant les données associées, même s'il n'y a pas de correspondance avec **table2**. A savoir, s'il n'y a pas de correspondance les colonnes de **table2** vaudront toutes NULL.

JOINTURES SQL …

Jointure externe : LEFT OUTER JOIN …

Table utilisateur

id	prenom	nom	email	ville
1	Aimée	Marechal	aime.marechal@example.com	Paris
2	Esmée	Lefort	esmee.lefort@example.com	Lyon
3	Marine	Prevost	m.prevost@example.com	Lille
4	Luc	Rolland	lucrolland@example.com	Marseille

Table commande

utilisateur_id	date_achat	num_facture	prix_total
1	20130123	A00103	203.14
1	20130214	A00104	124.00
2	20130217	A00105	149.45
2	20130221	A00106	235.35
5	20130302	A00107	47.58

Pour lister tous les **utilisateurs** avec leurs **commandes** et afficher également les **utilisateurs qui n'ont pas effectuées d'achats**, il est possible d'utiliser la requête suivante :

JOINTURES SQL …

Jointure externe : LEFT OUTER JOIN …

```
SELECT *
FROM utilisateur
LEFT OUTER JOIN commande ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	date_achat	num_facture	prix_total
1	Aimée	Marechal	20130123	A00103	203.14
1	Aimée	Marechal	20130214	A00104	124.00
2	Esmée	Lefort	20130217	A00105	149.45
2	Esmée	Lefort	20130221	A00106	235.35
3	Marine	Prevost	NULL	NULL	NULL
4	Luc	Rolland	NULL	NULL	NULL

Les dernières lignes montrent des utilisateurs qui n'ont effectuée aucune commande. La ligne retourne la valeur **NULL** pour les colonnes concernant les achats qu'ils n'ont pas effectués.

Filtrer sur la valeur NULL

Attention, la valeur **NULL** n'est pas une chaîne de caractère. Pour filtrer sur ces caractères il faut utiliser la commande **IS NULL** que nous avons déjà rencontré.

JOINTURES SQL …

Jointure externe : RIGHT OUTER JOIN

En SQL, la commande **RIGHT OUTER JOIN** est un type de jointure entre 2 tables qui permet de retourner tous les enregistrements de la table de **droite** (right = droite) **même s'il n'y a pas de correspondance** avec la table de **gauche**.

S'il y a un enregistrement de la table de droite qui ne trouve pas de correspondance dans la table de gauche, alors les colonnes de la table de gauche auront **NULL** pour valeur.

```
SELECT *
FROM table1
RIGHT OUTER JOIN table2 ON table1.id = table2.fk_id
```

Cette syntaxe stipule qu'il faut lister toutes les lignes du tableau **table2** (tableau de droite) et afficher les données associées du tableau **table1** s'il y a une correspondance entre ID de **table1** et FK_ID de **table2**. S'il n'y a pas de correspondance, l'enregistrement de **table2** sera affiché et les colonnes de **table1** vaudront toutes NULL.

JOINTURE SQL …

Jointure externe : RIGHT OUTER JOIN …

Table utilisateur

id	prenom	nom	email	ville	actif
1	Aimée	Marechal	aime.marechal@example.com	Paris	1
2	Esmée	Lefort	esmee.lefort@example.com	Lyon	0
3	Marine	Prevost	m.prevost@example.com	Lille	1
4	Luc	Rolland	lucrolland@example.com	Marseille	1

Table commande

utilisateur_id	date_achat	num_facture	prix_total
1	20130123	A00103	203.14
1	20130214	A00104	124.00
2	20130217	A00105	149.45
3	20130221	A00106	235.35
5	20130302	A00107	47.58

JOINTURES SQL …

Jointure externe : RIGHT OUTER JOIN …

Pour afficher **toutes** les **commandes** avec le nom de l'**utilisateur** correspondant il est habituel d'utiliser INNER JOIN en SQL. Malheureusement, si l'utilisateur a été supprimé de la table, alors ça ne retourne pas l'achat. L'utilisation de **RIGHT OUTER JOIN** permet de retourner tous les achats et d'afficher le nom de l'**utilisateur** s'il existe. Pour cela il convient d'utiliser cette requête :

```
SELECT id, prenom, nom, utilisateur_id, date_achat, num_facture
FROM utilisateur
RIGHT OUTER JOIN commande ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	utilisateur_id	date_achat	num_facture
1	Aimée	Marechal	1	20130123	A00103
1	Aimée	Marechal	1	20130214	A00104
2	Esmée	Lefort	2	20130217	A00105
3	Marine	Prevost	3	20130221	A00106
NULL	NULL	NULL	5	20130302	A00107

Ce résultat montre que la facture A00107 est liée à l'utilisateur numéro 5. Or, cet utilisateur n'existe pas ou n'existe plus. Grâce à **RIGHT OUTER JOIN**, l'achat est tout de même affiché mais les informations liées à l'utilisateur sont remplacé par NULL.

SQL Sous-Requêtes

Dans le langage SQL une **sous-requête** (aussi appelé « requête imbriquée » ou « requête en cascade ») consiste à exécuter une requête à l'intérieur d'une autre requête.

Une requête imbriquée est souvent utilisée au sein d'une clause WHERE ou de HAVING pou remplacer une ou plusieurs constantes.

Requête imbriquée qui retourne un seul résultat

```
SELECT *
FROM `table`
WHERE `nom_colonne` = (  SELECT `valeur`  FROM `table2`  LIMIT 1 )
```

Cet exemple montre une requête interne (celle sur « **table2**″) qui renvoi une seule valeur. La requête externe quant à elle, va chercher les résultats de « table » et filtre les résultats à partir de la valeur retournée par la requête interne.

A noter : il est possible d'utiliser n'importe quel opérateur d'égalité tel que =, >, <, >=, <= ou <>.

SQL Sous-Requêtes …

Requête imbriquée qui retourne une colonne

Une requête imbriquée peut également retournée une **colonne entière**. Dès lors, la requête externe peut utiliser la commande **IN** pour filtrer les lignes qui possèdent une des valeurs retournées par la requête interne. L'exemple ci-dessous met en évidence un tel cas de figure :

```
SELECT *
FROM `table`
WHERE `nom_colonne` IN (  SELECT `colonne`  FROM `table2`
    WHERE `cle_etrangere` = 36
  )
```

SQL Sous-Requêtes ...

Table question

q_id	q_date_ajout	q_titre	q_contenu
1	20130324 12:54:32	Comment réparer un ordinateur?	Bonjour, j'ai mon ordinateur cassé, comment puis-je procéder pour le réparer?
2	20130326 19:27:41	Comment changer un pneu?	Quel est la meilleur méthode pour changer un pneu facilement ?
3	20130418 20:09:56	Que faire si un appareil est cassé?	Est-il préférable de réparer les appareils électriques ou d'en acheter de nouveaux?
4	20130422 17:14:27	Comment faire nettoyer un clavier d'ordinateur?	Bonjour, sous mon clavier d'ordinateur il y a beaucoup de poussière, comment faut il procéder pour le nettoyer? Merci.

Table reponse

r_id	r_fk_question_id	r_date_ajout	r_contenu
1	1	20130327 07:44:32	Bonjour. Pouvez vous expliquer ce qui ne fonctionne pas avec votre ordinateur? Merci.
2	1	20130328 19:27:11	Bonsoir, le plus simple consiste à faire appel à un professionnel pour réparer un ordinateur. Cordialement,
3	2	20130509 22:10:09	Des conseils son disponible sur internet sur ce sujet.
4	3	20130524 09:47:12	Bonjour. Ça dépend de vous, de votre budget et de vos préférence visàvis de l'écologie. Cordialement,

SQL Sous-Requêtes …

Requête imbriquée qui retourne un seul résultat

Avec une telle application, il est peut-être utile de connaître la question liée à la dernière réponse ajoutée sur l'application. Cela peut être effectué via la requête SQL suivante :

```
SELECT *
FROM `question`  WHERE q_id = (
    SELECT r_fk_question_id  FROM `reponse`
    ORDER BY r_date_ajout DESC  LIMIT 1
  )
```

q_id	q_date_ajout	q_titre	q_contenu
3	20130418 20:09:56	Que faire si un appareil est cassé?	Est-il préférable de réparer les appareils électriques ou d'en acheter de nouveaux?

SQL Sous-Requêtes …

Ce résultat démontre que la question liée à la dernière réponse sur le forum est bien trouvée à partir de ce résultat.

Requête imbriquée qui retourne une colonne

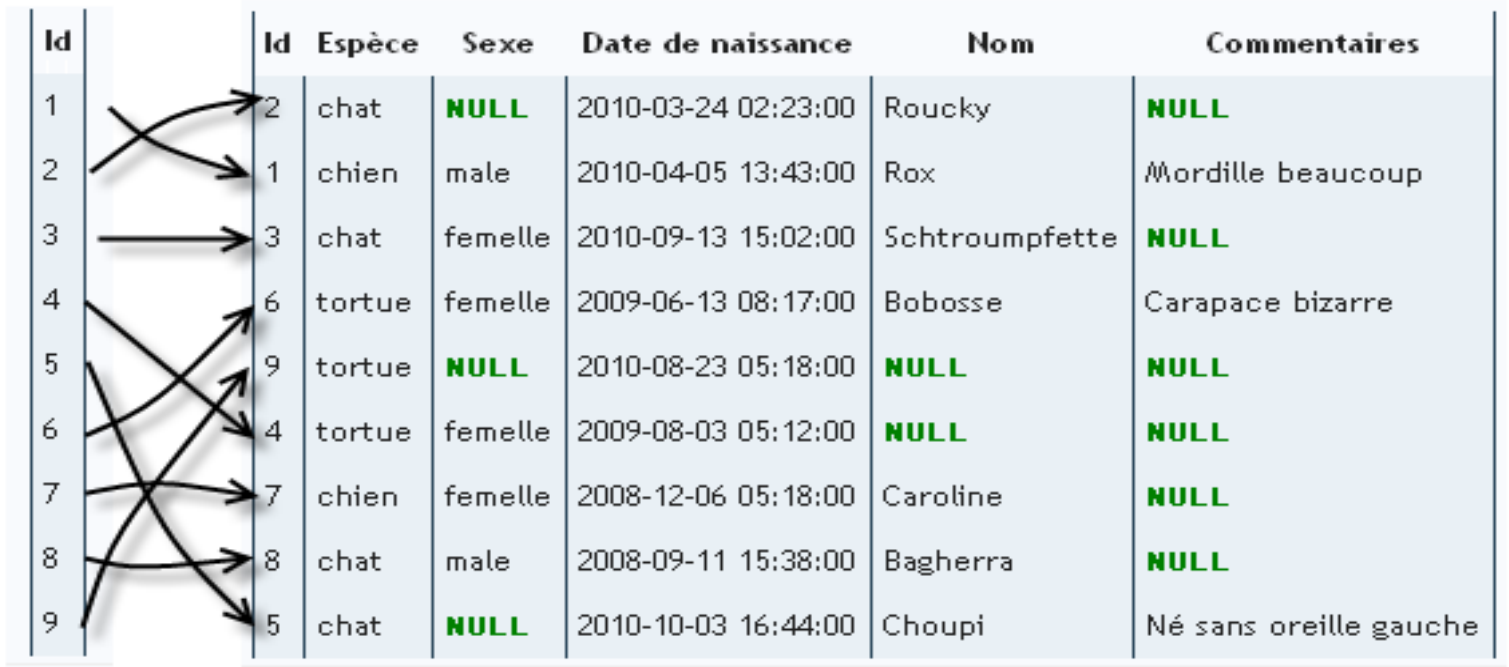
Imaginons maintenant que l'ont souhaite obtenir les questions liées à toutes les réponses comprises entre 2 dates. Ces questions peuvent être récupérée par la requête SQL suivante :

```
SELECT *
FROM `question`  WHERE q_id IN (
    SELECT r_fk_question_id  FROM `reponse`
    WHERE r_date_ajout BETWEEN '2013-01-01' AND '2013-12-31'
  )
```

q_id	q_date_ajout	q_titre	q_contenu
1	20130324 12:54:32	Comment réparer un ordinateur?	Bonjour, j'ai mon ordinateur de cassé, comment puis-je procéder pour le réparer?
2	20130326 19:27:41	Comment changer un pneu?	Quel est la meilleur méthode pour changer un pneu facilement ?
3	20130418 20:09:56	Que faire si un appareil est cassé?	Est-il préférable de réparer les appareils électriques ou d'en acheter de nouveaux?

SQL La notion d'Index

Définition : **Structure de données** qui reprend la **liste ordonnée** des valeurs auxquelles il se rapporte.



Id
1
2
3
4
5
6
7
8
9

Id	Espèce	Sexe	Date de naissance	Nom	Commentaires
2	chat	NULL	2010-03-24 02:23:00	Roucky	NULL
1	chien	male	2010-04-05 13:43:00	Rox	Mordille beaucoup
3	chat	femelle	2010-09-13 15:02:00	Schtroumpfette	NULL
6	tortue	femelle	2009-06-13 08:17:00	Bobosse	Carapace bizarre
9	tortue	NULL	2010-08-23 05:18:00	NULL	NULL
4	tortue	femelle	2009-08-03 05:12:00	NULL	NULL
7	chien	femelle	2008-12-06 05:18:00	Caroline	NULL
8	chat	male	2008-09-11 15:38:00	Bagherra	NULL
5	chat	NULL	2010-10-03 16:44:00	Choupi	Né sans oreille gauche

Fixons un **index** sur l'***id*** de la table ***Animal***

l'index sur **l'*id*** est **trié** simplement par **ordre croissant**

Cela permet de grandement **accélérer** toute recherche faite sur cet ***id***

SQL La notion d'Index ...

En effet, si nous voulons récupérer toutes les lignes dont l'id est **inférieur** ou égal à **5**.

Sans index, **MySQL** doit parcourir toutes les lignes une à une.
Par contre, grâce à l'index, dès qu'il tombe sur la ligne dont l'id est 6, il sait qu'il peut s'arrêter, puisque toutes les lignes suivantes auront un id supérieur ou égal à 6. Dans cet exemple, on ne gagne que quelques lignes, mais imaginez une table contenant des millions de lignes. Le gain de temps peut être assez considérable.

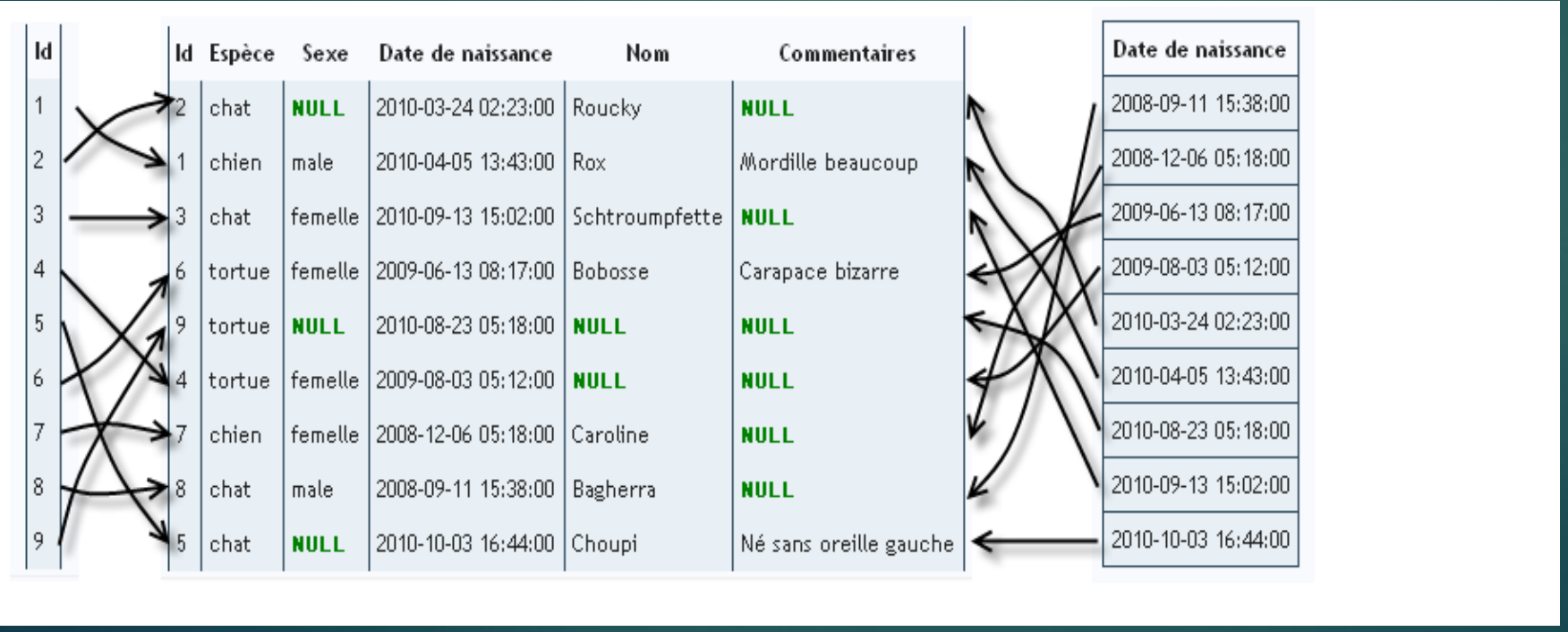
Par ailleurs, avec les id triés par ordre croissant, pour rechercher un id particulier, MySQL n'est pas obligé de simplement parcourir les données ligne par ligne. Il peut utiliser des **algorithmes** de recherche puissants (comme la recherche **dichotomique** que nous avons déjà rencontré en algorithmie), toujours afin **d'accélérer la recherche**.

SQL La notion d'Index …

Mais pourquoi ne pas simplement trier la table complète sur la base de la colonne **id** ?

Pourquoi créer et stocker une structure spécialement pour l'index ?

Tout simplement parce qu'il peut y avoir **plusieurs index** sur une **même table**, et que l'ordre des lignes, pour chacun de ces index, n'est pas nécessairement le même. Par exemple, nous pouvons créer un **second index** pour notre table **Animal**, sur la colonne **date_naissance**.



Id	Id	Espèce	Sexe	Date de naissance	Nom	Commentaires
1	2	chat	NULL	2010-03-24 02:23:00	Roucky	NULL
2	1	chien	male	2010-04-05 13:43:00	Rox	Mordille beaucoup
3	3	chat	femelle	2010-09-13 15:02:00	Schtroumpfette	NULL
4	6	tortue	femelle	2009-06-13 08:17:00	Bobosse	Carapace bizarre
5	9	tortue	NULL	2010-08-23 05:18:00	NULL	NULL
6	4	tortue	femelle	2009-08-03 05:12:00	NULL	NULL
7	7	chien	femelle	2008-12-06 05:18:00	Caroline	NULL
8	8	chat	male	2008-09-11 15:38:00	Bagherra	NULL
9	5	chat	NULL	2010-10-03 16:44:00	Choupi	Né sans oreille gauche

Date de naissance
2008-09-11 15:38:00
2008-12-06 05:18:00
2009-06-13 08:17:00
2009-08-03 05:12:00
2010-03-24 02:23:00
2010-04-05 13:43:00
2010-08-23 05:18:00
2010-09-13 15:02:00
2010-10-03 16:44:00

Comme vous pouvez le constater, **l'ordre** n'est pas du tout le même.

SQL La notion d'Index ...

Intérêt des index

Vous devriez avoir compris maintenant que tout l'intérêt des index est **d'accélérer les requêtes** qui utilisent des **colonnes indexées** comme critères de recherche.
Par conséquent, si vous savez que, dans votre application, vous ferez énormément de recherches sur la **colonne X**, ajoutez donc un **index** sur cette **colonne**.

Les index permettent aussi d'assurer **l'intégrité des données** de la base de données. Pour cela, il existe plusieurs types d'index différents, et deux types de "clés". Grâce aux **clés** et aux **index**, vous pouvez par exemple avoir la garantie que tous les clients auxquels vous faites référence dans la table **Commande** existent bien dans la table **Client**.

Désavantages

Si tout ce que fait un **index**, c'est **accélérer les requêtes** utilisant les critères de recherche correspondants, autant en mettre partout et en profiter à chaque requête. Seulement, les index ont **deux inconvénients**.

- Ils prennent de la **place en mémoire**.
- Ils **ralentissent les requêtes d'insertion**, **modification** et **suppression**, puisqu'à chaque fois, il faut remettre l'index à jour en plus de la table.

Par conséquent, **ajoutez** un **index** lorsque c'est **vraiment utile**.