# Etude comparative des bases de données

Base de données relationnel : Le modèle de données relationnel a été défini en 1970 par l’informaticien anglais Edgar F. Codd, travaillant pour la société IBM. Depuis les années 1990, le modèle de base de données relationnel est utilisé dans la plupart des bases de données et ce jusqu’à aujourd’hui. Une base de données relationnel est comparable à une collection de tableur connectés entre eux, permettant une organisation simple et précise. Chaque tableur est appelé « table ». Ces tables stockent les données, et peuvent être représenter sous la forme de colonnes, ou attributs, qui indiquent le type de donnée, et lignes, ou tuples, qui contiennent les valeurs correspondant aux différents types de données. Par exemple, dans une catégorie « personnes » (= table) qui contient des int (= colonne/type de donnée), soit l’âge de chaque personne (= ligne). Dans une même table, il peut y avoir plusieurs colonnes et bien sûr, dans chaque colonne il peut y avoir plusieurs lignes. Dans une base de données relationnel, chaque table a un unique id pour chaque ligne, appelé clé primaire (ou « primary key » en anglais). Pour relier les tables entre elles, on peut utiliser une clé étrangère (ou « foreign key » en anglais), qui n’est autre que la clé primaire d’une autre table. C’est-à-dire que dans 2 tables différentes, on peut relier/combiner les données si 2 colonnes ont le même nom et sont composées du même type de données, renvoyant au fait que la colonne appelé primary key est généralement retrouvée dans une autre table sauf sous la forme d’une foreign key car l’autre table a sa propre primary key. 2 tables pourraient avoir 2 primary key identiques mais dans ce cas, on pourrait littéralement fusionner les 2 tables en une seule puisque le contenu de chaque ligne de chaque colonne se basent sur les valeurs. Dans le cas ou il n’y a aucune colonne similaire entre 2 tables, l’architecture/l’organisation de la base de données doit être repensée, par exemple en ajoutant une 3ème table qui comporterait 1 colonne de chacune des tables 2 façon à pouvoir les connecter entre elles (tel un pointeur). Concernant les liens entre les tables, il y a plusieurs façons de les faire en utilisant SQL, en fonction notamment du résultat attendu :

* Inner join : créé une nouvelle table qui va rassembler les colonnes des tables reliées. Puis copie les valeurs des lignes correspondant aux valeurs de la colonne primary key de la 1ère table et de la colonne foreign key de la 2ème table dont les valeurs sont égales.
* Left join : similaire à l’inner join mais avec organisation un peu différente et surtout, si les valeurs d’une ligne dans la colonne primary key n’a pas de match dans les lignes dans la colonne foreign key d’une autre table, les valeurs seront mises quand même dans la nouvelle table mais avec des valeurs NULL pour les lignes qui n’ont pas de match avec les lignes de l’autre table.
* Right join : similaire à Left join. Utilisé s’il y a une table intermédiaire : considère ainsi la primary key de la table qui est normalement « issue » d’une autre table (puisque les tables ont une sorte d’ordre d’importance, notamment parce que la primary key de la table « principale » se retrouve sous la forme de foreign key chez les autres tables « secondaires »). En remodelant l’organisation des tables, faire un Right join devient inutile et alors remplacé par un Left join
* Full join : utilisé s’il n’y a pas de de foreign key, soit si une colonne (qui n’est pas une primary key), est similaire à une colonne (qui n’est pas une primary key) d’une autre table, ou a du moins le même type de donnée. A part ça, les données sont rassemblées de la même façon qu’un inner join (sauf donc qu’on ne se base pas sur une primary key) et/ou qu’un left join, si les valeurs n’ont pas de match dans l’autre table
* Union join : créé une table dans laquelle une colonne rassemblera les données d’une colonne de 2 tables (ou plus bien sûr) différentes (les valeurs égales sont rassemblées en une, sauf si on utilise un Union All)
* Cross join : créé une table avec une colonne et fait quelque chose comme ça :

Table 1 colonne « name » : A, B

Table 2 colonne « name » : C, D

Nouvelle table colonne « name » : A-C, A-D, B-C, B-D

Avantage modèle relationnel :

* Facile à prendre en main. C’est un des premiers modèles de base de données donc depuis sa sortie, son utilisation a eu le temps de se standardiser, ce qui signifie plus de documentation ainsi que d’outils à disposition. Le langage utilisé, SQL (Structured Query Language) a aussi eu le temps de prendre forme et devenir lui aussi simple d’utilisation.
* Organisation. Grâce aux contraintes que représente le modèle (clé primaire/étrangère, tables, colonnes, lignes), les données sont bien organisées
* Puissance du SQL. SQL permet différentes opérations complexes : requêtes, jointures (liens entre les tables) …
* Robustesse : propriété ACID. Tous les changements au niveau des données sont exécutés comme s’il n’étaient qu’une opération : ACID désigne chacune de ces étapes : Atomicity 🡪 tout ou rien ; Consistency 🡪 données valides ; Isolation 🡪 aucune interférence ; Durability 🡪 données récupérables. Si l’une de ces étapes ne fonctionne pas, alors la transaction est annulée. Cela permet d’avoir une base de données uniquement composée de données sûres et vérifiées.

Inconvénients du modèle :

* Rigidité. Les bases relationnelles nécessitent un schéma strict. Donc modifier de larges quantités de données régulièrement peut s’avérer difficile
* Analyse. Faire des analyses massives de données nécessite de passer par beaucoup de liens entre les tables, rendant l’opération peu optimisée, par rapport à des modèles comme les bases colonnes ou orientées graphes.
* Inadéquation avec certains types données. Le modèle n’est pas très adapté dans le cas où les données sont des graphes (ex : réseaux sociaux), des données non structurés (ex : images) ou des vecteurs (ex : données d’ia)

Une base de données orientée graphes est une base de données orientée objets qui utilise la théorie des graphes, soit des modèles abstraits de dessins de réseaux reliant des objets entre eux. Ce type de base de données permet de stocker les données tout en étant capable d’avoir un lien avec un élément voisin, qui peut être accéder en l’occurrence grâce à un pointeur. C’est un peu comme en C, quand on a une structure qui contient des pointeurs vers d’autres structures. Ou de façon plus visuelle, c’est un peu comme le tableau d’un détective qui rassemble différents indices, que ce soit des images, des documents, des objets etc.. et qui les relie par des fils rouges pour représenter les connections entre les éléments.   
Il y a plusieurs types de liens entre les objets/nœuds existant, par exemple, les liens peuvent avoir une direction (ex : élève ---> professeur, soit l’élève de tel professeur), ou une propriété (ex : élève -en 2020-> professeur, soit l’élève de tel professeur en 2020).

Le type de requêtes ou d’interactions est différent de celles généralement effectuées avec SQL. Pour les bases de données orientées graphes ont leurs propres langages de requêtes, à savoir Gremlin, Sparql, Cypher et autres.

Exemple d’utilisation de Cypher :

MATCH (w:Worker)-[:WORKS\_FOR]->(c:Company {c.Name:"Smals"})  
RETURN w.Name

Une image contenant texte, diagramme, cercle, Police

Description générée automatiquement

Avantages modèle orienté graphe :

* Efficacité et qualité de la gestion des relations entre les éléments/nœuds. Ce modèle gère extrêmement bien les relations complexes. Prenons l’exemple du système « amis d’amis » dans les réseaux sociaux. Si on devait utiliser une clé primaire/étrangère dans une base de données relationnelle, ça voudrait dire qu’il faudrait créer, dans la table qu’on peut appeler « utilisateur » une colonne pour l’identifiant de chaque amis par exemple, puis créer une table qui contiendrait tout les identifiants des amis de chacun des amis ainsi qu’une colonne avec les identifiants des amis de base (pour avoir une clé étrangère). Et c’est sans compter les informations stockées pour chaque utilisateur « amis ». Alors que dans une base de données orientée graphes, il suffit de faire un lien/pointeur entre l’utilisateur et ses amis, puis pour chaque ami, entre l’ami et l’ami de l’ami. Ce qui rend le schéma très flexible, apte à ajouter ou supprimer des éléments, ainsi que viable dans des domaines où les données sont difficiles à structurer à l’avance comme dans des systèmes d’information évolutifs ou des analyses contextuelles
* Requêtes intuitives. Les langages de requêtes comme Cypher ou Gremlin permettent de poser des questions complexes de manière naturelle en parcourant les graphes, par exemple, demander à trouver le chemin le plus court entre 2 nœuds. En SQL, il faudrait passer par différentes jointures, ce qui est plus complexe à utiliser.
* Performances stables. Même avec des volumes importants de relations, la capacité à gérer les données sera constante. Par exemple, dans une analyse de réseau, le temps de réponse dépend souvent du nombre de relations proches plutôt que de la taille totale de la base.

Inconvénients modèle orienté graphes :

* Modèle récent. Cela signifie que les outils utilisés, comme les langages de requête (Cypher, Gremlin…), ne sont pas encore standardisés, ce qui entraîne des manques de documentations, d’optimisation ou encore d’adaptabilité, puisque le manque de normes comme le SQL (pour les bases relationnelles) peut rendre difficile l'interopérabilité ou la migration de données entre différents systèmes. Il y a également peu d’experts qualifiés, donc peu de gens pour faire avancer les choses rapidement, en plus du coût qu’ils représentent à engager étant donné leur rareté. Il faut encore attendre un certain temps avant de pouvoir exploiter le plein potentiel des bases de données orientées graphes.
* Agrégation globale des données. Quand il s’agit de faire des calculs impliquant l’ensemble des données, ce modèle présente plus de difficultés que le modèle relationnel. Après tout, si on compare en C, il est plus facile de compter les strings dans une array, plutôt que dans une liste chaînée…
* Complexité. Le manque d’outils, de normes, et le principe de fonctionnement du modèle le rendent difficile d’utilisation. Pour reprendre l’exemple entre array et liste chaînée en C, une liste chaînée (pour représenter la base de données graphes) est évidemment plus difficile à gérer qu’une array (pour représenter la base de données relationnelle). Ainsi, ce modèle n’est pas idéal pour des applications nécessitant de nombreuses transactions simples ou des requêtes bien structurées.

Sources :

<https://www.youtube.com/watch?v=Yh4CrPHVBdE>

<https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es_relationnelle>

<https://www.smalsresearch.be/graph-db-vs-rdbms/>

https://fr.wikipedia.org/wiki/Base\_de\_donn%C3%A9es\_orient%C3%A9e\_graphe

chatgpt.com