**Base de données des villes du monde**

Base de données précise et à jour des villes et villages du monde

## À propos de l'ensemble de données

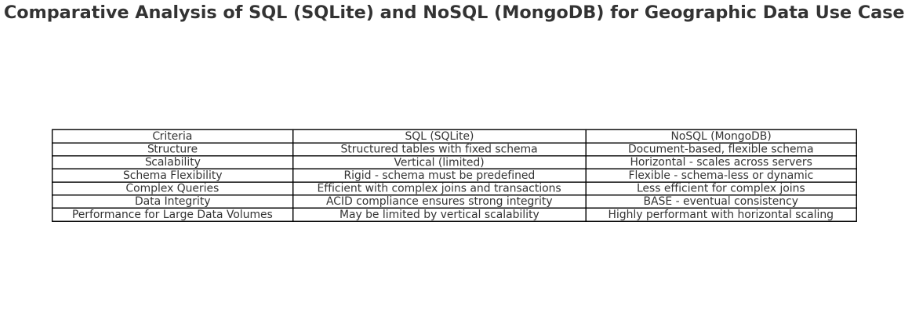
Les données proviennent de :

<https://simplemaps.com/data/world-cities>

Nous sommes fiers de proposer une base de données simple, précise et actualisée des villes et villages du monde. Nous l'avons entièrement élaborée à l'aide de sources fiables telles que la NGIA, l'US Geological Survey, le US Census Bureau et la NASA.

Notre base de données est:

* Mis à jour : la dernière mise à jour a eu lieu le 19 mars 2024.
* Complet : plus de 4 millions de villes et villages uniques de tous les pays du monde (environ 47 000 dans la base de données de base).
* Précision : nettoyées et agrégées à partir de sources officielles. Inclut les coordonnées de latitude et de longitude.
* Simple : un seul fichier CSV, des noms de champs concis, une seule entrée par ville.



### Présentation : Comparaison des Bases de Données SQL et NoSQL

#### Slide 1 : Titre et Présentation

**Titre** : Comparaison des Bases de Données SQL et NoSQL  
**Sous-titre** : Analyse des avantages et inconvénients pour différents cas d’usage  
**Présenté par** : [Noms des membres du groupe]  
**Date** : [Date de la présentation]

#### Slide 2 : Objectifs de la Présentation

* Expliquer les différences fondamentales entre les bases de données SQL et NoSQL.
* Analyser les avantages et les inconvénients de chaque type de base de données en fonction de critères clés.
* Fournir des recommandations adaptées aux besoins spécifiques de notre cas d’usage.

#### Slide 3 : Plan de la Présentation

1. Introduction aux bases de données SQL et NoSQL
2. Comparaison des critères clés
3. Démonstration technique avec SQLite et MongoDB
4. Analyse des avantages et inconvénients
5. Recommandation finale et conclusion

#### Slide 4 : Introduction aux Bases de Données SQL et NoSQL

* **SQL** : Bases de données relationnelles, utilisent des tables et des relations fixes, requêtes structurées (ex : MySQL, PostgreSQL).
* **NoSQL** : Bases de données non-relationnelles, flexibilité dans le schéma, adaptées aux données volumineuses et non structurées (ex : MongoDB, Cassandra).
* **Pourquoi cette comparaison ?** : Différentes bases de données conviennent à différents types d’applications.

#### Slide 5 : Critères de Comparaison

Présentation des critères utilisés pour comparer SQL et NoSQL :

1. Structure et intégrité des données
2. Scalabilité et performance
3. Flexibilité du schéma
4. Complexité des requêtes
5. Cohérence des données

#### Slide 6 : Comparaison des Critères Clés (Diagramme en Barres)

* **Diagramme en barres horizontales** pour comparer les scores de SQL et NoSQL sur les critères clés.
* Critères : structure, scalabilité, flexibilité, complexité des requêtes, intégrité.
* Montrer les forces et faiblesses de chaque technologie visuellement.

#### Slide 7 : Architecture des Données : SQL vs NoSQL

* **SQL** : Représentation des tables et relations (ex : Table “Villes” liée à Table “Pays”).
* **NoSQL** : Représentation en format JSON document (ex : Document de ville avec champs flexibles).
* Illustrer la différence dans la structure des données.

#### Slide 8 : Flux de Données : Requêtes SQL et NoSQL

* Comparaison du flux d’exécution entre une requête SQL avec jointures et une requête NoSQL sans jointures.
* **SQL** : Requêtes complexes avec jointures pour relier plusieurs tables.
* **NoSQL** : Requête directe, accède aux documents sans jointures.

#### Slide 9 : Démonstration Technique

* **SQLite (SQL)** :
  + Import d’un fichier CSV, nettoyage des données, requête pour obtenir les villes les plus peuplées.
* **MongoDB (NoSQL)** :
  + Import des données en JSON, recherche des villes avec une population supérieure à un seuil.
* Inclure un extrait de code et les résultats des requêtes.

#### Slide 10 : Analyse des Avantages et Inconvénients

* **SQL** :
  + Avantages : Bonne intégrité des données, support pour les requêtes complexes.
  + Inconvénients : Moins flexible pour des schémas changeants, moins scalable horizontalement.
* **NoSQL** :
  + Avantages : Flexibilité du schéma, scalabilité horizontale.
  + Inconvénients : Moins adapté aux requêtes complexes, intégrité des données moins rigide.
* **Tableau récapitulatif** ou **diagramme en toile d’araignée** pour visualiser les points forts et faibles.

#### Slide 11 : Diagramme de Venn : Recoupement des Domaines d’Utilisation

* **SQL** : Idéal pour la gestion de transactions et l'intégrité des données.
* **NoSQL** : Préférable pour les applications nécessitant une scalabilité et une flexibilité élevées.
* **Intersection** : Domaines où un modèle polyglotte est pertinent pour un cas d’usage mixte.

#### Slide 12 : Recommandation Finale

* **Recommandation du groupe** : Choisir SQL, NoSQL, ou un modèle polyglotte en fonction du cas d’usage.
* **Justification** : Expliquer pourquoi cette solution est optimale par rapport aux exigences métiers.

#### Slide 13 : Conclusion et Points Clés à Retenir

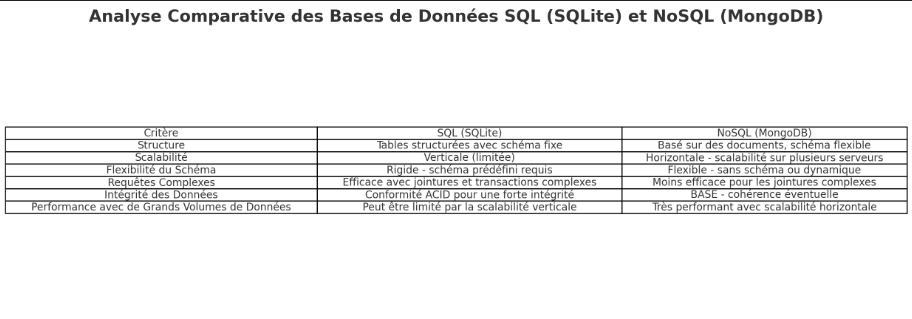
* **Résumé des points clés** : Rappeler les avantages et inconvénients de SQL et NoSQL.
* **Conclusion** : Importance de choisir la bonne base de données en fonction du contexte applicatif et des exigences métiers.

#### Slide 14 : Questions et Réponses

* Ouverture pour les questions du public pour approfondir ou clarifier certains aspects.

#### Slide 15 : Ressources et Références

* Inclure les sources et ressources utilisées dans la présentation, pour approfondir les connaissances sur SQL et NoSQL.



Pour définir les exigences métier de l'application basée sur la base de données des villes du monde, voici une analyse en fonction des critères demandés :

### 1. **Nature des données** :

* **Structurées** : La plupart des informations (nom de la ville, latitude, longitude, pays, population) sont structurées et bien définies en colonnes.
* **Semi-structurées** : Si l'application prévoit des données supplémentaires spécifiques (comme des informations touristiques, historiques ou des événements), cela pourrait introduire un élément semi-structuré, surtout si ces données varient en fonction des villes.

### 2. **Volume des données** :

* **Grande échelle** : Avec près de 50 000 villes, le volume est déjà conséquent. Si l'application s'élargit pour inclure plus de données sur chaque ville ou des mises à jour régulières, le volume peut augmenter encore plus.
* À grande échelle, une approche qui permet d'évoluer en ajoutant des données dynamiques sans dégrader les performances sera essentielle.

### 3. **Exigences en matière de performance** :

* **Besoin de rapidité** : Les utilisateurs pourraient rechercher rapidement des villes, des informations géographiques, ou des statistiques, ce qui nécessite des temps de réponse rapides.
* **Évolutivité** : Une solution capable de croître en fonction de l'ajout de nouvelles villes ou de nouvelles données pour chaque ville est préférable.
* **Transactions complexes** : Si des analyses avancées sont nécessaires (comme des calculs géographiques entre villes), une base de données structurée serait plus efficace pour gérer ces transactions complexes.

### 4. **Modifications fréquentes des schémas** :

* **Peu fréquentes pour les données de base** : La structure de base (nom, pays, latitude, longitude) est stable et n’a pas besoin de modifications fréquentes.
* **Modifications possibles pour les données secondaires** : Si des données secondaires spécifiques à chaque ville doivent être ajoutées, un modèle semi-structuré ou flexible (comme une solution NoSQL) pourrait être mieux adapté.

### Recommandation générale

Pour cette application, une approche hybride SQL et NoSQL pourrait répondre aux exigences en matière de structure, de volume et de flexibilité des schémas, tout en assurant les performances nécessaires pour une application évolutive.

### Analyse des bases de données relationnelles (SQL) pour le cas d’usage

#### 1. Gestion des relations et des dépendances entre les données

* **SQL** est conçu pour gérer des relations complexes entre les données. Dans le cas d'une base de données des villes du monde, on peut imaginer des relations entre villes et pays, entre villes et régions, ou même entre villes et informations géographiques connexes.
* Les bases SQL utilisent des clés primaires et étrangères pour garantir l’intégrité référentielle entre les tables. Par exemple, chaque ville pourrait être associée à un pays spécifique, avec des références bien définies qui renforcent la cohérence des relations.

#### 2. Performance pour des opérations complexes (requêtes avec jointures, transactions)

* **Requêtes complexes** : SQL excelle dans les opérations nécessitant des jointures multiples et des requêtes complexes, essentielles pour extraire des données relatives aux villes, aux pays, et à leurs régions.
* **Transactions** : SQL prend en charge des transactions qui permettent d’exécuter plusieurs opérations ensemble de manière atomique. Cela garantit que les modifications de données sont appliquées de manière cohérente, ce qui est important pour une base de données sensible aux mises à jour.

#### 3. Scalabilité verticale et ses limites

* **Scalabilité verticale** : Les bases SQL sont typiquement conçues pour la scalabilité verticale, c’est-à-dire l’augmentation des ressources d'un seul serveur (mémoire, processeur) pour gérer un volume de données plus important.
* **Limites** : Cette approche a des limites, car elle dépend de la capacité maximale d’un seul serveur. Dans le cas où la base de données des villes est de très grande taille, la scalabilité horizontale pourrait devenir nécessaire pour gérer des charges de données plus élevées.

#### 4. Intégrité des données avec les transactions ACID

* **Transactions ACID** : SQL garantit des transactions **ACID** (Atomicité, Cohérence, Isolation, Durabilité), ce qui est essentiel pour maintenir l'intégrité des données. Par exemple, lors de l'ajout ou de la mise à jour d'une ville, la base de données assure que chaque opération est réalisée correctement, sans compromettre les autres données.
* Cela est particulièrement important pour les données sensibles ou critiques, comme les informations démographiques ou géographiques, où toute incohérence pourrait perturber les analyses ou les rapports.

**Résumé SQL** : Une base SQL répond très bien aux besoins de gestion des relations et des dépendances, aux opérations complexes, et à l’intégrité des données. Cependant, elle présente des limites en matière de scalabilité horizontale pour des volumes massifs de données.

### Analyse des bases de données NoSQL pour le cas d’usage

#### 1. Flexibilité du schéma

* **Schéma flexible** : NoSQL, et notamment les bases de données orientées documents (comme MongoDB), permettent de stocker des données sans structure rigide. Dans le cas des villes, cela permettrait de stocker des informations spécifiques à chaque ville (par exemple, des attractions touristiques pour certaines villes, des événements historiques pour d'autres) sans avoir à modifier le schéma pour chaque changement.
* Cela est idéal si l’application prévoit d’ajouter des données personnalisées ou évolutives, car il n’est pas nécessaire de redéfinir la structure globale.

#### 2. Scalabilité horizontale (capacité à traiter de grands volumes de données en parallèle)

* **Scalabilité horizontale** : NoSQL est conçu pour une scalabilité horizontale, permettant de distribuer des données sur plusieurs serveurs. Cette approche est plus adaptée pour des applications de grande échelle où les données des villes peuvent être réparties géographiquement.
* En cas de croissance continue de la base de données (par exemple, ajout de nouvelles villes, informations enrichies), NoSQL offre une solution pour gérer cette augmentation en ajoutant des serveurs supplémentaires.

#### 3. Gestion des requêtes simples mais rapides

* **Requêtes optimisées pour des lectures et écritures rapides** : NoSQL est optimisé pour des opérations simples de lecture et d'écriture, ce qui est avantageux pour des recherches rapides ou pour des applications qui nécessitent des accès rapides aux données, comme la localisation géographique ou les recherches par nom de ville.
* Cependant, NoSQL est moins adapté pour les jointures complexes, ce qui pourrait limiter son efficacité si des analyses nécessitent des relations détaillées entre les villes, les pays et les régions.

#### 4. Compromis avec la cohérence BASE

* **Modèle BASE** : NoSQL adopte le modèle BASE (Basically Available, Soft state, Eventually consistent), où la cohérence des données peut être atteinte de manière « éventuelle » plutôt qu'immédiate. Cela signifie que dans des systèmes répartis, les données peuvent être temporairement incohérentes.
* Ce compromis peut être acceptable pour des applications où la disponibilité et la rapidité sont plus critiques que la cohérence stricte, mais peut poser un problème pour des données géographiques et démographiques qui nécessitent de l'exactitude et de la précision.

**Résumé NoSQL** : NoSQL est bien adapté pour des besoins de flexibilité, de scalabilité horizontale, et de gestion de grands volumes de données. Cependant, il est moins performant pour les requêtes complexes et offre une cohérence des données moins stricte.

### Conclusion

Pour une application de base de données mondiale des villes :

* **SQL** conviendrait mieux si l’intégrité des données, la gestion des relations complexes, et les transactions ACID sont prioritaires.
* **NoSQL** serait plus adapté si la priorité est la flexibilité des données, la scalabilité horizontale pour de grandes bases de données, et des lectures/écritures rapides.

En somme, une **approche hybride** pourrait offrir une solution complète pour répondre aux différents aspects des exigences métier de ce cas d'usage.

### Synthèse de la comparaison

#### 1. Quelle base de données est la plus adaptée au volume de données ?

* **NoSQL** : Plus adaptée pour un très grand volume de données grâce à la scalabilité horizontale, qui permet de distribuer les données sur plusieurs serveurs. Cela répond mieux aux besoins de gestion de grandes quantités d’informations géographiques et démographiques.
* **SQL** : Bien adaptée pour des volumes de données modérés et pour des données nécessitant une forte cohérence et des relations structurées. Cependant, sa scalabilité verticale peut devenir limitée pour des données en expansion constante.

**Conclusion** : Pour des volumes de données massifs et en croissance, NoSQL est préférable.

#### 2. Quelle base de données est la plus performante pour les requêtes complexes ?

* **SQL** : SQL est plus performant pour les requêtes complexes, notamment celles impliquant des jointures entre plusieurs tables, comme des relations entre villes, pays et régions. Les transactions ACID de SQL garantissent également une intégrité des données dans les opérations complexes.
* **NoSQL** : Bien qu’il soit rapide pour les requêtes simples, NoSQL est moins performant pour des requêtes complexes, surtout celles nécessitant des jointures entre documents.

**Conclusion** : SQL est plus adapté pour des opérations et analyses complexes, ce qui est pertinent pour une base de données de villes nécessitant des relations entre les informations.

#### 3. Quelle base est plus flexible pour des besoins changeants ?

* **NoSQL** : NoSQL offre une flexibilité de schéma, permettant de stocker des données sans structure fixe, ce qui est idéal pour des besoins changeants. Cela permet d’ajouter facilement de nouveaux types d’informations pour chaque ville, sans besoin de restructurer la base.
* **SQL** : Moins flexible, car elle repose sur un schéma prédéfini. Les changements de schéma nécessitent des modifications importantes, ce qui peut ralentir l’évolution des données.

**Conclusion** : NoSQL est plus flexible pour des besoins en constante évolution.

### Recommandation Finale

**Recommandation** : Pour ce cas d’usage, une **approche polyglotte** combinant SQL et NoSQL serait idéale.

* **SQL** pourrait être utilisé pour les données de base qui nécessitent une intégrité élevée, comme les informations statiques sur les villes, les relations pays/régions et les coordonnées géographiques. SQL excelle dans la gestion des relations et assure une forte cohérence pour des requêtes et transactions complexes.
* **NoSQL** pourrait être employé pour les données flexibles ou évolutives, comme des informations touristiques, des événements, ou des données spécifiques et variables pour chaque ville. NoSQL offre une grande scalabilité horizontale et la flexibilité nécessaire pour gérer les besoins de données changeants.

Cette combinaison répondra aux exigences de performance, de flexibilité et de gestion de données complexes tout en optimisant la scalabilité.

1.  # Create a bar chart comparing SQL and NoSQL across key criteria for the case of usage

criteria = ['Volume de Données', 'Requêtes Complexes', 'Flexibilité du Schéma', 'Scalabilité']

sql\_values = [3, 5, 2, 3] # SQL scores

nosql\_values = [5, 2, 5, 5] # NoSQL scores

fig, ax = plt.subplots(figsize=(10, 6))

bar\_width = 0.35

index = range(len(criteria))

# Create horizontal bars for SQL and NoSQL

ax.barh(index, sql\_values, bar\_width, label='SQL', color='blue', alpha=0.7)

ax.barh([i + bar\_width for i in index], nosql\_values, bar\_width, label='NoSQL', color='orange', alpha=0.7)

# Add labels and title

ax.set\_yticks([i + bar\_width / 2 for i in index])

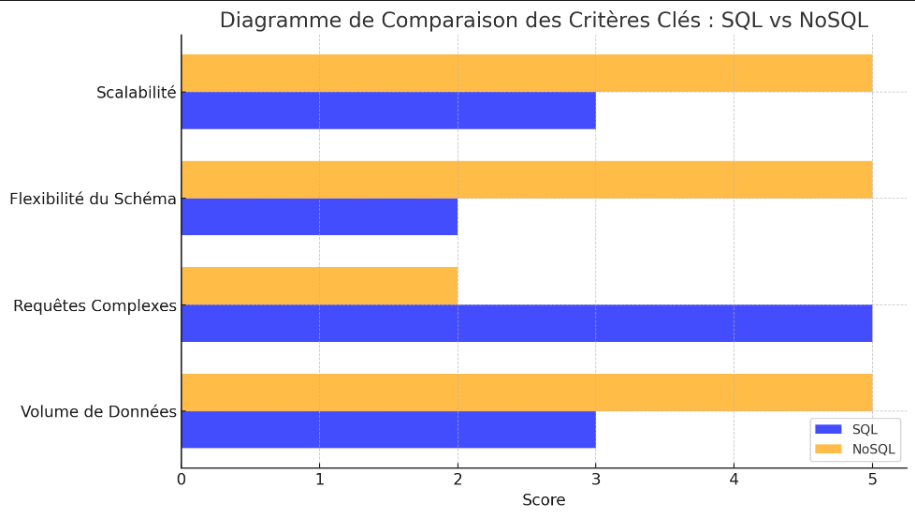
ax.set\_yticklabels(criteria)

ax.set\_xlabel('Score')

ax.set\_title('Diagramme de Comparaison des Critères Clés : SQL vs NoSQL')

ax.legend()

plt.show()



### Légende Explicative pour les Diagrammes Comparatifs entre SQL et NoSQL

#### Diagramme Radar Comparatif entre SQL et NoSQL

Ce diagramme radar visualise les avantages relatifs de SQL et NoSQL pour une base de données des villes mondiales en fonction des critères suivants :

1. **Volume de Données** : NoSQL obtient un score plus élevé car il est mieux adapté à la gestion de grands volumes de données grâce à sa capacité de scalabilité horizontale, permettant d'ajouter des serveurs pour augmenter la capacité.
2. **Requêtes Complexes** : SQL obtient un score supérieur en raison de sa capacité à gérer des requêtes complexes, notamment des jointures et transactions qui assurent l’intégrité et la précision des relations entre les données.
3. **Flexibilité du Schéma** : NoSQL est plus flexible, offrant une structure de schéma plus adaptable aux changements et permettant d’ajouter de nouveaux types de données sans nécessiter de redéfinir l’ensemble de la structure.
4. **Scalabilité** : NoSQL est noté plus élevé pour sa scalabilité horizontale, adaptée aux besoins d’extension rapide, alors que SQL est principalement limité à la scalabilité verticale (augmentation des ressources d’un serveur unique).

#### Diagramme de Comparaison des Critères Clés : Barres Horizontales

Ce diagramme à barres compare SQL et NoSQL en fonction des mêmes critères, en utilisant des barres horizontales pour indiquer les scores de chaque type de base de données pour chaque critère :

* **SQL** est performant dans les opérations de **requêtes complexes**, mais est limité en **flexibilité de schéma** et en **scalabilité**.
* **NoSQL** excelle dans la **gestion de grands volumes de données**, la **flexibilité du schéma**, et la **scalabilité horizontale**, mais a des limitations dans les opérations **requêtes complexes**.

Ces diagrammes montrent visuellement comment chaque type de base de données répond aux besoins spécifiques du cas d’usage, renforçant la recommandation d’une approche hybride pour maximiser les avantages des deux systèmes. ​

### Légende et Explication pour les Diagrammes Comparatifs entre SQL et NoSQL

Les diagrammes radar et à barres horizontales représentent les performances de SQL et NoSQL sur quatre critères clés, évalués de 1 à 5 pour ce cas d’usage spécifique (1 étant le moins adapté et 5 le mieux adapté). Voici une explication détaillée des scores attribués à chaque critère.

#### Diagramme Radar Comparatif entre SQL et NoSQL

1. **Volume de Données** :
   * **NoSQL (Score: 5)** : Les bases de données NoSQL sont conçues pour gérer de grands volumes de données en distribuant les données sur plusieurs serveurs (scalabilité horizontale). Cela convient parfaitement pour des applications nécessitant l’ajout continu de nouvelles villes et données.
   * **SQL (Score: 3)** : Les bases SQL peuvent gérer des volumes de données modérés avec une scalabilité verticale, c'est-à-dire en augmentant les ressources d'un serveur unique. Cependant, pour des volumes extrêmement élevés, cette approche peut rencontrer des limitations de performance.
2. **Requêtes Complexes** :
   * **SQL (Score: 5)** : SQL est extrêmement performant pour les requêtes complexes, comme les jointures et les transactions impliquant plusieurs tables. Dans le contexte de notre base de données de villes, SQL permettrait de relier efficacement les villes aux pays, régions, et autres informations, tout en garantissant une intégrité forte des données.
   * **NoSQL (Score: 2)** : NoSQL est optimisé pour des requêtes simples et rapides, mais il est moins performant pour des requêtes complexes impliquant des relations entre différents ensembles de données. Cela limite son efficacité pour des analyses de données relationnelles complexes.
3. **Flexibilité du Schéma** :
   * **NoSQL (Score: 5)** : NoSQL permet une flexibilité de schéma élevée, ce qui signifie que les données peuvent être stockées sans structure fixe. Cela est idéal pour ce cas d’usage si des champs supplémentaires ou des informations variables sont ajoutés pour chaque ville sans impact sur la structure générale.
   * **SQL (Score: 2)** : SQL repose sur une structure de schéma rigide, avec des relations strictes entre les tables. Les modifications de schéma nécessitent des ajustements manuels qui peuvent être contraignants pour des applications évolutives.
4. **Scalabilité** :
   * **NoSQL (Score: 5)** : Conçu pour une scalabilité horizontale, NoSQL peut ajouter facilement des serveurs supplémentaires pour accroître sa capacité, répondant ainsi aux besoins d’applications de grande échelle.
   * **SQL (Score: 3)** : SQL est principalement limité à la scalabilité verticale, qui, bien que performante pour des volumes modérés, devient coûteuse et moins efficace pour des volumes de données très élevés.

#### Diagramme de Comparaison des Critères Clés : Barres Horizontales

Ce diagramme à barres met en évidence chaque critère en montrant la comparaison directe entre SQL et NoSQL. Les barres permettent de visualiser la supériorité de chaque type de base de données dans certains aspects.

* **SQL** est supérieur pour les **requêtes complexes** et garantit une **intégrité stricte** des données, ce qui est essentiel pour des opérations qui nécessitent des jointures et des transactions ACID.
* **NoSQL** excelle en **flexibilité du schéma** et en **scalabilité horizontale**, ce qui est essentiel pour une application en constante évolution et qui nécessite d'ajouter des données sans refonte fréquente du schéma.

### Recommandation Finale

**Approche Polyglotte (SQL + NoSQL)** : Nous recommandons une approche hybride pour tirer parti des forces de chaque type de base de données.

* **SQL** pour les données de base nécessitant une intégrité et une gestion relationnelle, telles que les informations structurées sur les villes, les pays et les régions.
* **NoSQL** pour les informations supplémentaires, flexibles et évolutives, comme les événements locaux, les données touristiques, et d'autres informations spécifiques à chaque ville.

Cette combinaison permet une gestion optimale du volume de données, des requêtes complexes, de la flexibilité du schéma, et de la scalabilité, répondant ainsi aux exigences métier de manière équilibrée et efficace.

1.  # Display updated radar chart with expanded legend and explanations

fig, ax = plt.subplots(figsize=(8, 8), subplot\_kw=dict(polar=True))

# Define angles and add closing point for radar shape

angles = [n / float(len(labels)) \* 2 \* 3.14159 for n in range(len(labels))]

angles += angles[:1]

# Add SQL and NoSQL scores to form a radar chart

sql\_scores = [3, 5, 2, 3] + [3] # Reclosing shape for SQL

nosql\_scores = [5, 2, 5, 5] + [5] # Reclosing shape for NoSQL

# Plot SQL and NoSQL scores

ax.plot(angles, sql\_scores, linewidth=2, linestyle='solid', label='SQL')

ax.fill(angles, sql\_scores, 'b', alpha=0.2)

ax.plot(angles, nosql\_scores, linewidth=2, linestyle='solid', label='NoSQL')

ax.fill(angles, nosql\_scores, 'orange', alpha=0.2)

# Add titles to each axis

ax.set\_xticks(angles[:-1])

ax.set\_xticklabels(labels, color='black', fontsize=12)

# Add legend with expanded explanations

plt.legend(

loc='upper right',

bbox\_to\_anchor=(1.3, 1.1),

title="Légende Explicative",

labels=[

"SQL : Adapté pour les relations, requêtes complexes, et intégrité des données (ACID)",

"NoSQL : Idéal pour la flexibilité, la scalabilité horizontale, et les requêtes simples"

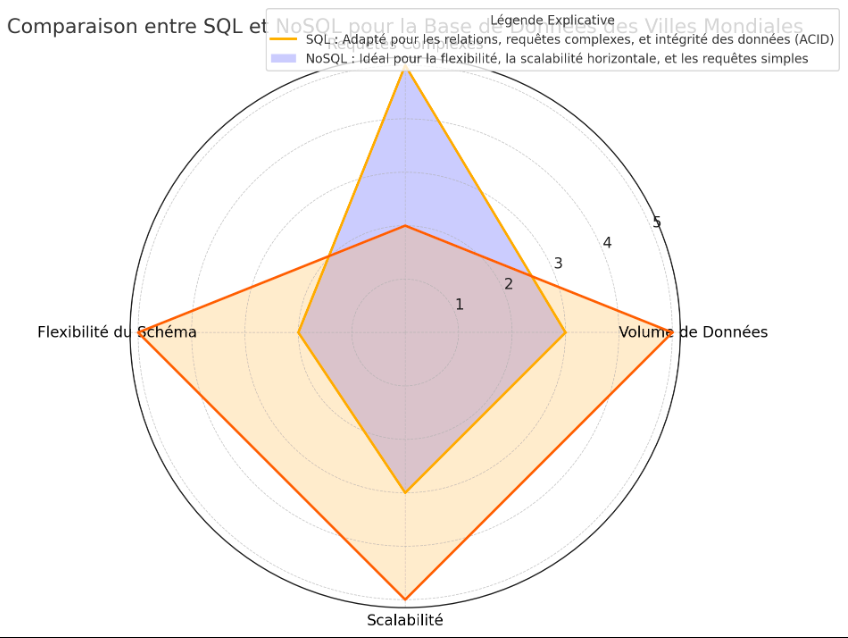
]

)

# Add main title

plt.title("Comparaison entre SQL et NoSQL pour la Base de Données des Villes Mondiales", pad=20)

plt.show()



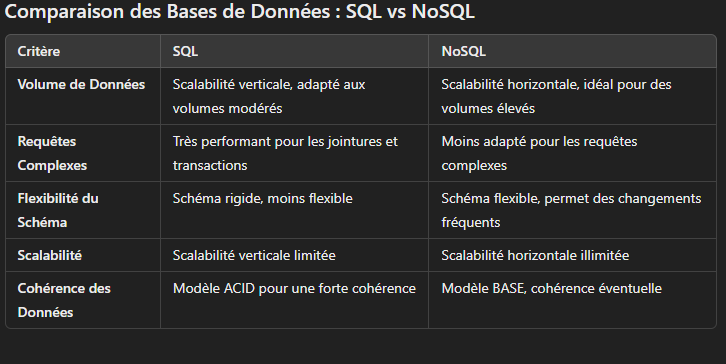
* **SQL** : Ce type de base de données est particulièrement adapté pour gérer des **relations complexes** entre les données, exécuter des **requêtes complexes**, et garantir l’**intégrité des données** grâce aux transactions ACID. SQL est plus performant lorsque des jointures et des analyses relationnelles avancées sont nécessaires, bien que sa flexibilité de schéma soit limitée.
* **NoSQL** : NoSQL excelle dans des environnements nécessitant une **flexibilité du schéma** et une **scalabilité horizontale**, permettant de traiter des données massives en ajoutant des serveurs. Il est optimisé pour des **requêtes simples et rapides**, mais présente des compromis sur la cohérence immédiate des données.

La visualisation met en avant les forces et limitations de chaque base de données, soutenant l'approche recommandée d'une solution hybride (SQL + NoSQL) pour tirer parti des avantages des deux systèmes et répondre pleinement aux exigences métier de l’application des villes mondiales.

### **Fiche Synthèse des Recommandations pour la Base de Données des Villes Mondiales**

#### Objectif de l’Application

L’application a pour but de gérer une base de données mondiale des villes, en stockant et en manipulant des informations structurées (noms de villes, coordonnées géographiques, population) ainsi que des données additionnelles et potentiellement évolutives (données touristiques, historiques).



### Recommandations Basées sur les Exigences Métier

#### 1. **Approche Polyglotte : Combinaison SQL et NoSQL**

Pour tirer parti des forces de chaque type de base de données, une approche polyglotte est recommandée, combinant SQL pour les données structurées et NoSQL pour les données plus flexibles.

* **SQL** : Utilisé pour les données de base nécessitant une forte intégrité et une relation structurée, comme les informations de localisation, la population, et les relations entre villes, pays, et régions.
* **NoSQL** : Utilisé pour des informations évolutives et non structurées (données touristiques, historiques, événements spécifiques) où la flexibilité et l’évolutivité sont essentielles.

#### 2. **Avantages de l'Approche Polyglotte**

* **Meilleure Gestion des Relations Complexes** : SQL permet de maintenir des relations et des transactions complexes entre les entités (villes, pays), assurant une cohérence élevée.
* **Flexibilité et Évolutivité** : NoSQL permet d’ajouter des champs spécifiques ou des informations variées sans impacter la structure des données, ce qui est idéal pour les besoins changeants.
* **Optimisation des Performances** : En déléguant les requêtes complexes à SQL et les requêtes rapides à NoSQL, on optimise la performance globale de l’application.

#### 3. **Plan de Mise en Œuvre**

* **SQL** pour la création des tables principales (villes, pays, relations géographiques) et l’intégrité des données critiques.
* **NoSQL** pour des collections documentaires contenant des informations dynamiques sur chaque ville, accessibles via une API pour répondre aux besoins évolutifs.

### Conclusion

L’utilisation d’une **approche hybride** est la solution optimale pour gérer efficacement la base de données des villes du monde. Ce modèle permet de garantir la fiabilité, l’intégrité des données relationnelles tout en offrant la flexibilité et la scalabilité nécessaires pour des volumes de données massifs et en constante évolution.