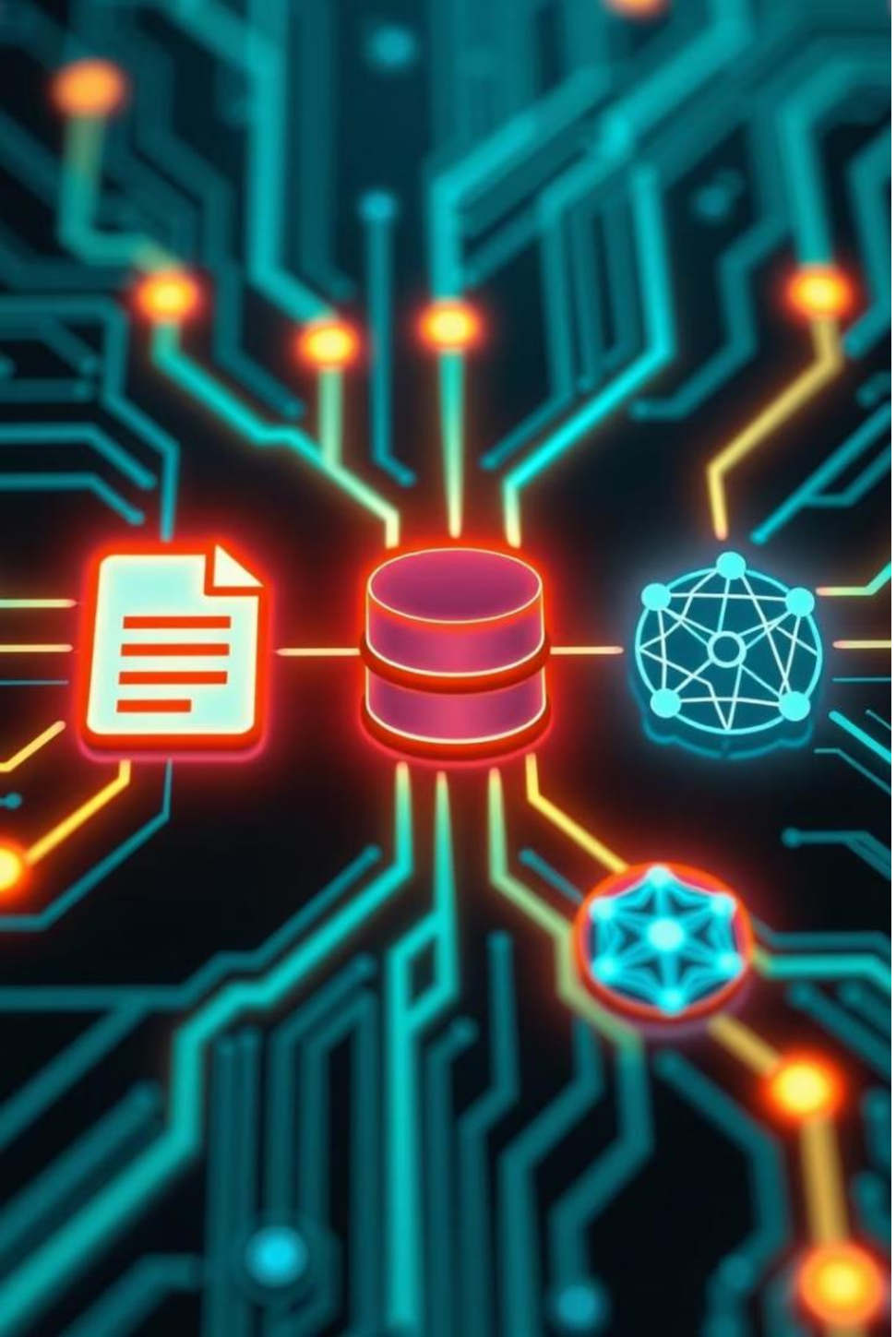


Architecture Polyglotte : MongoDB, PostgreSQL et Neo4J

Projet de gestion des voyages par Yang YANG et Binh Minh NGUYEN.

Notre Stack de Bases de Données Polyglotte

PostgreSQL

Version 17.5, gérée via pgAdmin 4. Base de données relationnelle robuste.

MongoDB

Client mongosh 2.5.1, serveur mongod v8.0.9. Outils Compass et Atlas pour le cloud.

Neo4j

Version Enterprise 5.24.0, gérée via Neo4j Desktop. Base de données orientée graphe.

Vue d'ensemble des BDD Polyglotte

info-api (MongoDB)

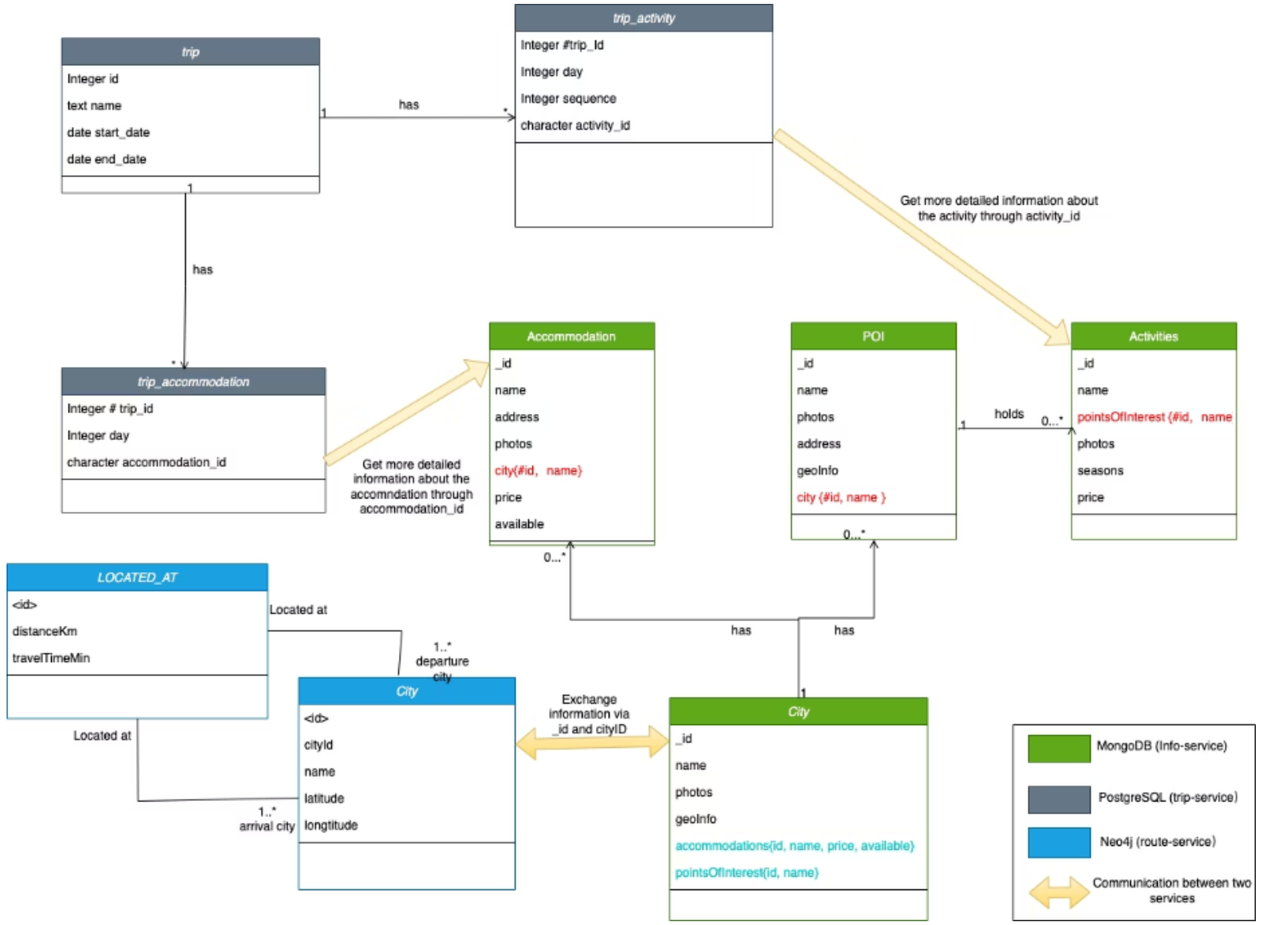
Gère les villes, hébergements, POI et activités. Idéal pour les données hiérarchiques et semi-structurées grâce à son modèle documentaire flexible.

trip-api (PostgreSQL)

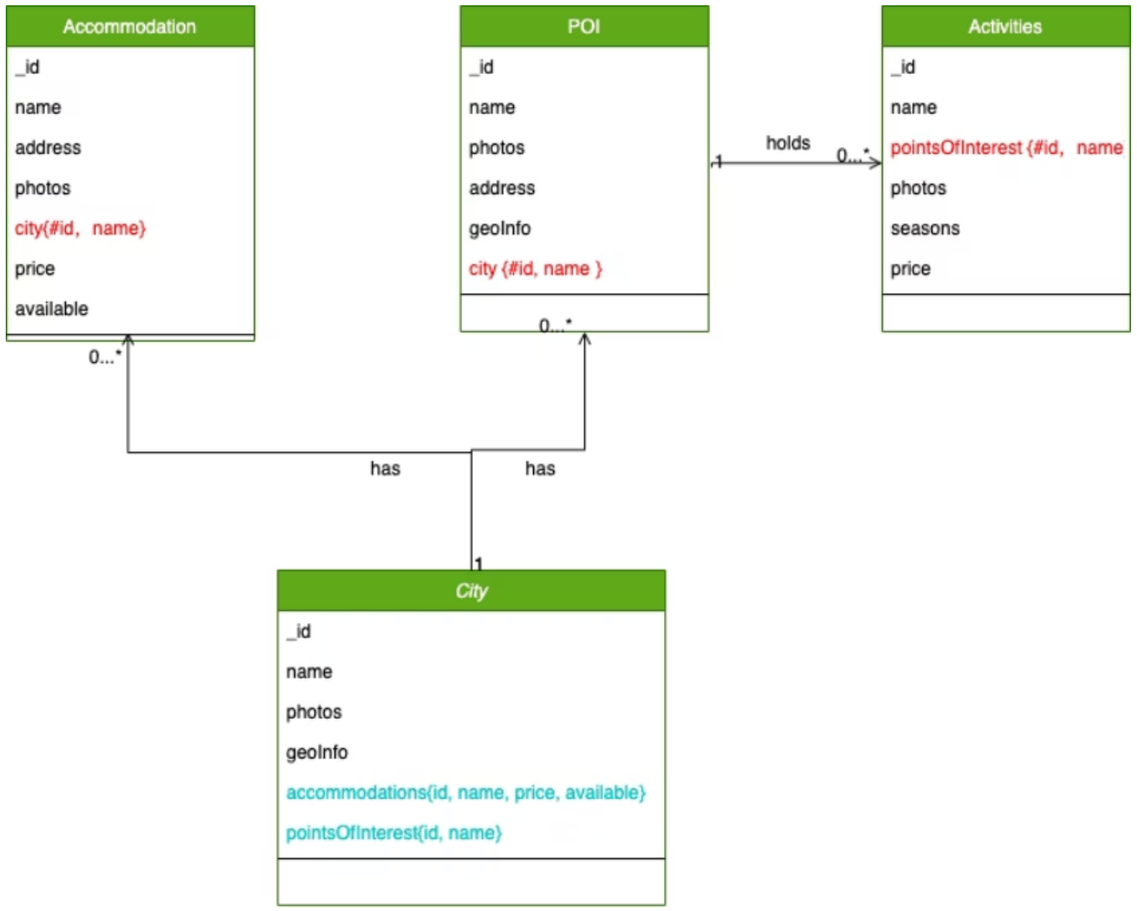
Crée et gère les itinéraires journaliers. Parfait pour les contraintes relationnelles strictes et les transactions ACID requises par les itinéraires.

route-api (Neo4j)

Modélise le réseau routier comme un graphe. Calcule les distances, durées et itinéraires optimisés en exploitant les algorithmes de graphe.



MongoDB : Structure



Collections Principales

Accommodation, POI, Activities et City sont nos quatre collections principales. MongoDB utilise des références au lieu de clés étrangères dans SQL relationnel.

Embedding Partiel

Nous embeddons des champs fréquemment lus dans d'autres collections. Par exemple, l'ID et le nom des POI dans City, ou le nom de la ville dans POI et Accommodation.

Problème de Consistance

Les modifications peuvent rendre les champs embeddés obsolètes. Cela crée des incohérences si non géré correctement.

MongoDB : Triggers Atlas Cloud

Pour résoudre les problèmes d'incohérence, nous utilisons des triggers MongoDB Atlas. Ces triggers assurent la synchronisation automatique des collections.

Synchronisation Automatique

Les triggers sont déclenchés par insertion, mise à jour ou suppression.

Mise à Jour des Embeds

Ils mettent à jour les champs embeddés dans les collections liées.

Cohérence des Données

Ils garantissent que les données restent cohérentes à travers tous les documents.

Nous avons six triggers spécifiques. Ils assurent que nos références et embeddings restent à jour.**1.SyncAccommodationToCity 2.SyncPOIToCity 3.SyncCityNameToChildren**

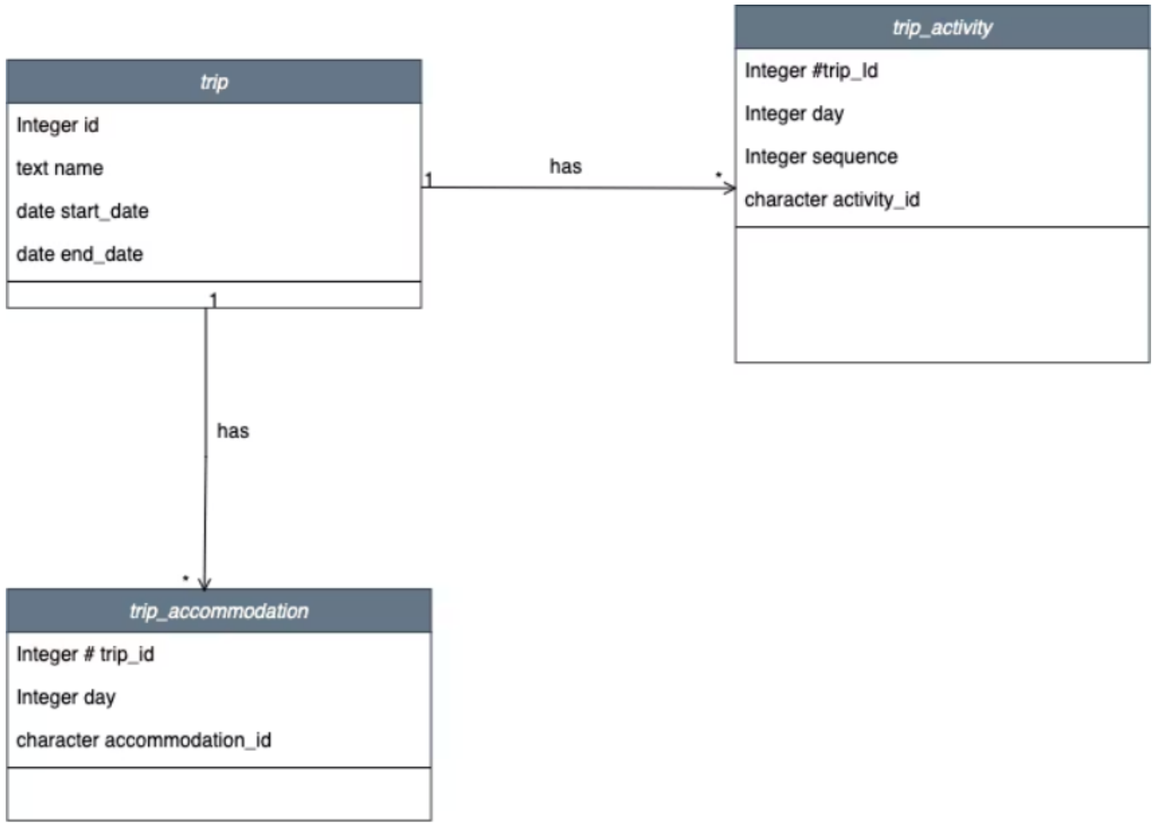
4.SyncPOIToActivity 5.cascadeDeleteCityChildren 6.cascadeDeletePOIChildren

MongoDB : Validators pour l'Intégrité

Collection	But du Validator	Champs Clés et Contraintes
city	Assurer que chaque ville inclut des POI et hébergements dénormalisés.	id (objectId) · name (string) · photos (tableau d'URL HTTP/HTTPS) · geoInfo (lat∈[-90,90], lon∈[-180,180]) · accommodations (liste d'objets avec id, nom, prix ≥0, dispo booléen) · pointsOfInterest (liste d'objets avec id et nom)
pointOfInterest	Vérifier les POI détaillés, avec infos géographiques et référence à la ville.	_id, name, photos (URL) · address (string) · geoInfo (lat/lon valides) · city (objet avec id et nom)
activity	Garantir que chaque activité est liée à un POI, avec saisons et tarification.	_id, name · pointOfInterest (id + nom) · photos (URL) · seasons (un ou plusieurs mois parmi 12 valeurs autorisées) · price (objet { adult ≥ 0, child ≥ 0 })
accommodation	Garantir que chaque hébergement possède les informations détaillées et correctement formatées.	_id (ObjectId) · name (string) · address (string) · photos (array d'URL HTTP/HTTPS) · city (objet avec id ObjectId, nom string) · price (int/double/decimal ≥ 0) · available (booléen)

Les validators sont appliqués à la création des collections. Ils imposent des règles strictes sur la structure des documents.

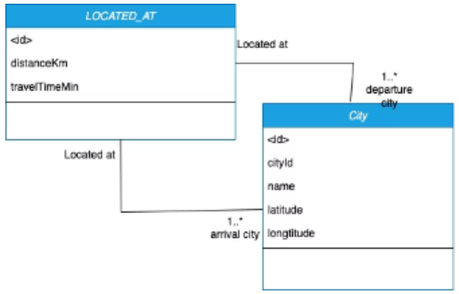
PostgreSQL : Cohérence et Intégrité

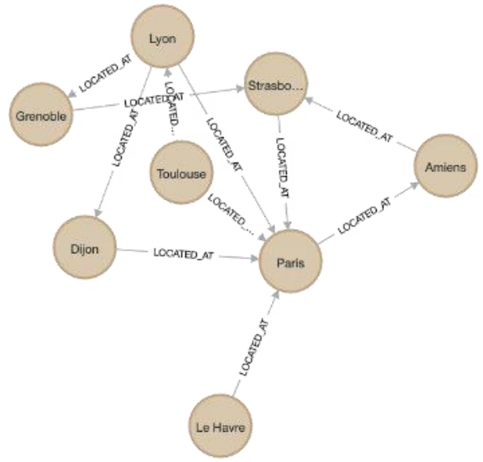


- Nous utilisons des triggers pour garantir la cohérence des données entre les tables après les mises à jour.
- Des opérations en cascade (suppression/mise à jour) sont en place pour assurer l'intégrité référentielle.
- Les contraintes CHECK et UNIQUE assurent la validité des données, comme les dates de fin postérieures aux dates de début.

Les contraintes SQL sont moins puissantes pour les formats complexes. Nous effectuons ces validations au niveau de l'application avec Spring Boot.

Neo4j : Graphes pour les Itinéraires Optimisés





Nœuds Ville

Chaque ville est un nœud City, relié par une relation LOCATED_AT.

Données Géospatiales

Le type géospatial permet de calculer les distances et d'estimer les temps de trajet. (point/point.distance)

Chemin Optimal

La fonction shortestPath() trouve le chemin minimal entre les villes.

Conclusion