On the Impedance Mismatch of Vector Data: Vector Databases vs. Relational Databases

Conceptes per a Base de Dades Especialitzades

Laboratori 1

Grup 11

Aina Gomez Pinyol

Rafael Ibañez Rodríguez

Taula de continguts

1. PostgreSQL

1.1. P0: Emmagatzemar dades

En aquesta primera fase del laboratori, l'objectiu és carregar i emmagatzemar un conjunt de textos de la llibreria BookCorpus dins d'una base de dades PostgreSQL, de manera que les frases estiguin organitzades i siguin fàcils de processar. Donada la grandària del *dataset*, cal prendre decisions que minimitzin el risc de bloqueigs o errors i facilitin el treball amb dades en blocs manejables.

Per dur a terme aquest exercici, s'ha utilitzat el dataset *rojagtap/bookcorpus*, una variant de BookCorpus, ja que el dataset original generava errors en descarregar-lo.

1.1.1. Estructura de la taula

Per organitzar les frases dins de PostgreSQL, es va crear una taula anomenada *chunks_db* que està formada per atrubuts. El primer és l'atribut *id*, la clau primària automàtica que garanteix que cada frase tingui un identificador únic i evita conflictes durant les insercions massives. El segon atribut és *chunk_id*, que identifica el bloc al qual pertany cada frase, facilitant la filtració per blocs i la gestió de grans volums de dades. L'últim atribut és *sentence*, on es guarda el text de cada frase.

Aquesta estructura permet adaptar de manera coherent les dades lineals de Python a un model relacional, minimitzant l'efecte de l'*impedance mismatch* i permetent consultes més clares i eficients.

1.1.2. Separació de frases amb NLTK

Abans d'inserir les dades a PostgreSQL, cada text es processa amb la llibreria NLTK per separar-lo en frases. Aquesta decisió és essencial perquè els textos poden contenir puntuació complexa, abreviatures o només propis amb punts que podrien dividir-se incorrectament amb un simple *split*. L'ús de NTLK garanteix que cada frase sigui una unitat autocontinguda, cosa que facilita la gestió de dades dins de la base de dades.

1.1.3. Inserció per blocs

Per evitar saturar el servidor i reduir el nombre de crides SQL, les frases es processen en blocs de 10.000 i s'insereixen dins d'una transacció amb un únic *commit* per bloc. Aquesta decisió assegura que, en cas d'error durant la inserció, només cal revertir el bloc afectat sense comprometre la resta de la base de dades. A més, permet mesurar de manera objectiva el temps que triga cada inserció.

1.1.4. Ús de transaccions

Cada inserció de bloc es realitza dins d'una transacció, la qual cosa garanteix que les operacions siguin atòmiques i que la integritat de les dades es mantingui encara que hi hagi

errors durant el procés. Aquesta decisió és especialment rellevant, ja que evita que la diferència entre la representació de Python i la de PostgreSQL provoquin inconsistències. L'ús de transaccions permet que les operacions siguin més segures i controlades sense augmentar la complexitat del codi.

1.1.5. Conclusió

En conclusió, la primera fase del laboratori ha permès preparar i emmagatzemar de manera eficient un subconjunt del dataset dins de PostgreSQL. Les decisions preses han ajudat a minimitzar l'*impedance mismatch* entre la naturalesa lineal de les dades en Python i l'estructura tabular de la base de dades. Aquest enfocament ha garantit insercions estables i coherents, ha facilitat la gestió de grans volums de dades i ha establert un marc sòlid per a les etapes següents del laboratori.

1.2. P1: Generar embeddings

En aquest segon apartat, l'objectiu principal ja estat generar embeddings per a cadascuna de les frases prèviament emmagatzemades dins de la base de dades. Aquest pas introdueix un nou nivell, ja que els embeddings són vectors de centenars de dimensions i un sistema relacional tradicional no està dissenyat nativament per gestionar aquest tipus d'informació.

1.2.1. Estructura de la taula

Per resoldre aquest problema, s'ha creat una taula addicional anomenada *embedding_table*, on cada registre conté l'identificador de la frase original *id_sentence*, l'identificador del bloc *chunk_id* i el vector resultant *embedding*. La clau primària de la taula és *sentence_id*, mentre que s'ha definit una clau forana que assegura la integritat referencial amb la taula de frases. Pel que fa l'atribut *embedding*, el tipus de dada escollit ha estat *FLOAT8*, ja que permet emmagatzemar un llistat de valors reals de manera directa i és la solució més propera dins de les opcions que ofereix PostgreSQL.

1.2.2. Estratègia d'inserció

Un altre aspecte rellevant ha estat la manera d'inserir les dades. En aquest cas, s'ha optat per fer un *commit* després de cada inserció. Tot i que aquesta estratègia és poc eficient des del punt de vista del rendiment, ja que implica un gran nombre de transaccions petites, facilita el control i la traçabilitat dels errors, assegurant que cada embedding queda guardat immediatament sense dependre d'altres insercions.

1.2.3. Generació dels embeddings

Pel que fa a la generació dels embeddings, s'ha utilitzat el model *all-MiniLM-L6-v2*, un transformador lleuger que ofereix un bon equilibri entre qualitat i temps de càlcul. Tot i això, s'ha de tenir en compte que el temps de processament no és sempre estable, ja que depèn tant de la longitud de la frase com de la càrrega del sistema en el moment de l'execució.

1.2.4. Conclusió

En aquest apartat s'ha desenvolupat un script que genera *embeddings* per a cada frase i els desa en PostgreSQL, mantenint la coherència amb la taula original mitjançant claus foranes. Tot i utilitzar commits després de cada inserció, una estratègia senzilla, però poc eficient, això garanteix simplicitat en la gestió. El procés posa de manifest les limitacions de treballar amb vectors en bases de dades relacionals i obre la porta a futures millores en rendiment.

1.3. P2: Càlcul de les frases similars

Per aquesta part hem seleccionat 10 frases de forma aleatòria de la base de dades i hem calculat les dues frases més similars a cadascuna d'elles segons dues mètriques: *Cosine Similarity* i *Euclidean Distance*.

1.3.1. Estructura de dades i càrrega en memòria

Una decisió clau va ser carregar tots els embeddings a memòria com a diccionari *{sentence_id: (chunk_id, embedding)}*. Aquesta estructura permet accedir ràpidament a qualsevol embedding sense haver de fer múltiples consultes SQL, reduint així la càrrega sobre la base de dades i minimitzant el cost de temps associat a múltiples transaccions. Aquest enfocament ajuda a mitigar l'*impedance mismatch*, ja que es passa de la representació interna de la base de dades a una estructura Python nativa que es pot manipular directament per càlculs matemàtics.

1.3.2. Càlcul de similitud

Per calcular la similitud entre embeddings, vam triar *Cosine Similarity* i *Euclidean Distance*, dues mètriques complementàries que proporcionen perspectives diferents sobre la proximitat semàntica entre frases. Els embeddinfs es van reorganitzar amb *reshape* per adaptar-los a les funcions de *sklearn*, mostrant com la transformació de dades és necessària per ajustar la representació de la base de dades a la que requereix el processament Python.

1.3.3. Conclusió

En resum, les decisions principals preses per aquest apartat van ser: carregar tots els embeddings a memòria per reduir consultes SQL, utilitzar estructures nadiues de Python per al càlcul de similitud, i separar la lògica de càlcul de la lògica de la base de dades per minimitzar l'impacte de l'impedance mismatch. Aquest enfocament permet obtenir resultats correctes i ràpids tot mantenint el control i la flexibilitat del processament vectorial.

1.4. Respostes de PQ1

Els temps d'inserció del text mostren certa variabilitat, amb valors mínims de 0.01432s i màxims de 0.62377s, mentre que els embeddings són molt més estables, amb una mitjana de 0.01262s. Tot i que cada inserció d'embeddins és ràpida, el temps total necessari per generar-los arriba a uns 633 segons.

Els temps de consulta per calcular les dues frases més similars per a 10 frases de prova són relativament constants, amb una mitjana de 0.41306s i una desviació de 0,02338s. No s'observen diferències significatives entre les dues mètriques utilitzades i la manera de calcular la similitud produeix resultats fiables i uniformes per a totes les frases analitzades.

Per optimitzar les operacions sense usar Pgvector, es podrien aplicar *batch commits* per reduir el nombre de transaccions o crear índexs sobre les claus de les taules. Aquestes tècniques permetrien millorar la velocitat d'inserció i de consulta, minimitzant l'impacte de l'*impedance mismatch* inherent en l'ús de vectors en una base de dades relacional.

2. Chroma

2.1. C0: Carregant les dades

Per a aquest apartat, la primera decisió va ser utilitzar el mateix subconjunt de dades que s'havia carregat prèviament a PostgreSQL. Les frases es van desar en un fitxer de text pla, que posteriorment es va llegir línia per línia en Python. Aquesta aproximació permet un flux de dades senzill i evita errors d'incompatibilitat de formats, ja que Chroma pot processar directament textos i embeddings.

La connexió amb Chroma es va establir a través d'un client persistent, indicant un directori dins de la carpeta del projecte Ubuntu. Aquesta decisió va permetre que totes les dades carregades es guardessin localment i que es pogués reprendre el procés sense haver de recarregar les frases cada cop que s'executa el *script*. A cada frase se li va generar l'embedding utilitzant el model preentrenat all-MiniLM-L6-v2 de SentenceTransformers abans d'afegir-la a la col·lecció de Chroma. D'aquesta manera, la inserció de les dades textuals i dels vectors es fa en un sol pas, minimitzant el nombre de crides i simplificant el codi.

2.2. C1: Generar embeddings

Per aquest apartat es va crear un script en Python que connecta amb la col·lecció de Chroma on prèviament es van carregar les frases. S'ha utilitzat el model SentenceTransformer per generar els embeddings de totes les frases i després s'han actualitzat a la col·lecció. Aquestes actualitzacions s'han fet de manera individual per cada embedding tot i que Chroma permet inserir per lots. Aquesta decisió permet analitzar la distribució dels temps de manera més detallada i detectar possibles anomalies o casos atípics en l'actualització de vectors.

Per obtenir mesurament més precís dels intervals curts, s'ha optat per utilitzar la funció *time.perf_counter()* en lloc de *time.time()*. Aquesta elecció garanteix que les estadístiques de temps recollides siguin més fiables i comparables amb altres processos similars.

Els embeddings es generen abans d'entrar al bucle d'actualització, evitant recalcular-los en cada iteració. Aquesta estratègia redueix el temps total d'execució i simplifica el bucle principal, cosa que permet centrar-se només en l'actualització de la col·lecció amb els vectors ja calculats.

2.3. C2: Càlcul de similitud

Per aquest apartat es van seleccionar aleatòriament deu frases de la col·lecció de Chroma ja carregada, de manera que es pogués replicar la mateixa avaluació feta amb PostgreSQL. Cada frase seleccionada es va comparar amb la resta d'elements de la col·lecció per trobar les dues més similars segons dues mètriques de distància, la similitud cosinus i la distància euclidiana. Aquesta aproximació permet analitzar com Chroma gestiona les consultes de

similitud de manera directa sobre els embeddings i proporciona una comparació amb l'estratègia anterior en bases de dades relacionals.

Les decisions principals en la implementació van ser generar un array amb tots els embeddings abans del bucle per evitar recalcular-los durant cada consulta, cosa que redueix significativament el temps total d'execució. També es va assegurar que la frase de consulta no es comparés amb ella mateixa, assignant valors extrems segons la mètrica corresponent, per obtenir les dues frases més similars.

La cronometria de cada consulta es va fer individualment, amb la qual cosa es va poder calcular el temps mínim, màxim, mitjà i la desviació estàndard de totes les consultes, cosa que permet avaluar l'estabilitat de les operacions i detectar possibles anomalies. Aquest enfocament facilita entendre com el rendiment de Chroma es comporta amb consultes sobre embeddings, evidenciant la influència de la vectorització i la cerca per similitud en el temps de resposta. També permet establir una comparació directa amb PostgreSQL, mostrant com els sistemes especialitzats en vectors poden gestionar les operacions de manera més eficient o diferent respecte a una base de dades relacional convencional.

2.4. Respostes de CQ1

Pel que fa als temps d'inserció de textos i embeddings, s'ha observat que són bastant estables. Els temps mínim, màxim, mitjà i la desviació estàndard calculats per cada actualització de document amb el seu embedding indiquen que, tot i petites variacions, no hi ha anomalies significatives ni pics de temps inesperats. Aquesta estabilitat es deu en part al fet que la vectorització es realitza abans de l'actualització i cada embedding s'envia directament a la col·lecció de Chroma, cosa que permet obtenir mesuraments consistents per a cada frase.

Quant als temps de consulta per calcular les similituds, també es poden considerar estables. Les mètriques utilitzades, similitud cosinus i distància euclidiana, mostren resultats coherents entre les diferents frases de prova, tot i que la distància euclidiana tendeix a requerir lleugerament més temps per computar-se sobre tots els embeddings a causa del càlcul de la norma quadrada per a cada vector. Chroma permet mesurar el temps de la consulta en conjunt, però no separa directament la creació de l'embedding de la inserció del text, ja que normalment s'integra dins de l'operació de add o update.

Per millorar el rendiment de les operacions en Chroma, es podrien explorar tècniques com la inserció per lots, en què diversos documents i embeddings es carreguen simultàniament, reduint el temps de crida individual. També es podria considerar l'ús d'estructures d'indexació internes de Chroma com arbres aproximats de veïns més propers (ANN) o altres índexs vectorials que accelerin les consultes de similitud, especialment quan la col·lecció creix significativament en nombre de vectors. Aquestes estratègies permetrien reduir considerablement tant el temps d'inserció com el de consulta sense perdre precisió en els resultats.

3. Comparació entre PostgreSQL i Chroma

Quan es compara PostgreSQL amb Chroma des de la perspectiva de l'*impedance mismatch*, es poden observar diferències clares en la forma en què cada sistema tracta les dades. PostgreSQL és una base de dades relacional tradicional, dissenyada per a dades estructurades, on els vectors s'han d'emmagatzemar com a *arrays* de *floats* dins de taules. Això comporta que cada operació d'inserció o consulta requereixi adaptacions específiques, com ara convertir els *embeddings* a *arrays* i escriure consultes SQL que facin càlculs de similitud externament. Aquest enfocament pot resultar més laboriós i menys directe quan es treballa amb grans volums de vectors, però ofereix una gran robustesa, control de transaccions i integritat referencial de les dades.

En canvi, Chroma és un sistema especialitzat en vectors i embeddings, dissenyat per a la cerca semàntica i la recuperació de veïns més propers. La gestió dels vectors és nativa, i les operacions com la cerca de similitud cosinus o la distància euclidiana estan optimitzades, la qual cosa facilita el desenvolupament i millora el rendiment en comparació amb un entorn SQL tradicional. La inserció i actualització d'embeddings és més natural i el temps de consulta tendeix a ser més consistent per a grans col·leccions de vectors. No obstant això, aquest enfocament sacrifica algunes funcionalitats típiques de bases de dades relacionals, com transaccions complexes, *joins* i integritat estricta de dades.

Si s'inclogués l'ús de Pgvector a PostgreSQL, es reduiria significativament l'*impedance mismatch*, ja que els vectors podrien ser gestionats de manera més natural dins de la base de dades i els índexs vectorials accelerarien les consultes de similitud. Això permetria apropar el comportament de PostgreSQL al d'un sistema especialitzat com Chroma, conservant alhora els avantatges relacionals clàssics, tot i que la configuració i manteniment serien encara més complexos que amb Chroma.

Annex A: Enllaç al repositori de github

El codi i els scripts desenvolupats per aquest treball es poden consultar al següent repositori: https://github.com/moonchildx/cbde-lab1