Projektaufgabe 1

Sparsity von E-Commerce Daten (12.5 Punkte)

In der ersten Projektaufgabe geht es um die Realisierung der Konzepte aus Vorlesung „Kapitel 1.2 - Verwaltung von E-Commerce Daten“. Konkret werden Sie den horizontalen und vertikalen Ansatz vergleichen. Hierzu setzten Sie eine Client-Server Infrastruktur um. Der horizontale Ansatz dient als Baseline und Ausgangspunkt. Die Aufgabe gliedert sich in drei Phasen. Die Phasen dienen Ihnen zur Planung, wann welche Deliveries anstehen. Sie sollten mit den Aufgaben der späteren Phasen jedoch frühzeitig beginnen.

Phase 1 – Legen der Basis (2.5 P)

In der ersten Phase legen Sie die Basis für das Gesamtprojekt. Dies umfasst:

- **Aufsetzten der Infrastruktur (0.5 Punkte):** Setzen Sie eine Client-Server Infrastruktur um. Sie sind vollkommen frei in der Wahl der Programmiersprache für den Client und in der Wahl des DBMS. Beachten Sie bei der Wahl des DBMS, dass Sie in Phase 3 ein System mit prozeduraler Erweiterung benötigen. Der Vorschlag für die Realisierung ist daher Java oder Python mit PostgreSQL. Für die Verbindung zwischen Client und Server verwenden Sie Schnittstellen, wie JDBC und keine Objekt-Relationalen Mapper etc.
  Für die Überprüfung dieses Teils zeigen Sie, dass sich der Client mit der Datenbank verbinden und SQL-Befehle ausführen kann.
- **Vorbereiten des Toy Beispiels für Korrektheitstests (1 Punkt):** Wir werden im laufe der VU immer wieder auf Toy-Beispiele für Korrektheitstests zurückgreifen. In dieser Aufgabe verwenden wir die Tabelle H_toy mit folgendem Schema: H_toy(a_1 String, a_2 String, a_3 Integer) mit den Daten aus Tabelle 1.
  a. Schreiben Sie ein Programm, dass die Tabelle in der Datenbank anlegt und mit Daten befüllt.
  b. Überführe, Sie die Tabelle H_toy (von Hand) in die zugehörige vertikale Darstellung V_toy und legen diese (inklusive Daten) in der Datenbank an.
  c. Legen Sie eine Sicht h2v_toy an, die V_toy in H_toy, entsprechend dem in der Vorlesung gegebenen Verfahren überführt.
  d. Legen Sie die beiden Partitionen von V_toy an, so dass die value Spalte den richtigen Datentyp (also Integer oder String) enthält. Vereinigen Sie die beiden Partitionen über ein Sicht zu V_toy_all.

Tabelle 1 Daten des Toy Beispiels. Das Zeichen ⊥ zeigt an, dass der Wert null ist.

oid	a_1	a_2	a_3
1	a	b	⊥
2	⊥	c	2
3	⊥	⊥	3
4	⊥	⊥	⊥

- **Datengenerator für horizontale Daten (1 Punkt):** Realisieren Sie eine Funktion `generate()`, die auf Client-Seite eine Relation `H`, wie in Tabelle 1 gezeigt, in der Datenbank erstellt und mit Daten füllt. Bei jedem Aufruf der Funktion soll die alte Relation gelöscht werden (sofern sie existiert). Der Funktion `generate` sollen folgende Argumente übergeben werden:

a. num_tuples (int) : spezifiziert die Anzahl der Tupel in `H`.
b. sparsity (double) in [0,1] : spezifiziert den (durchschnittlichen) Anteil der Tupel pro Attribut mit dem Attributwert null (⊥)
c. num_attributes (int) > 0 : spezifiziert die Anzahl an Attributen pro Tupel. Maximalwert ist der höchste Wert, den ihr DBMS unterstützt.

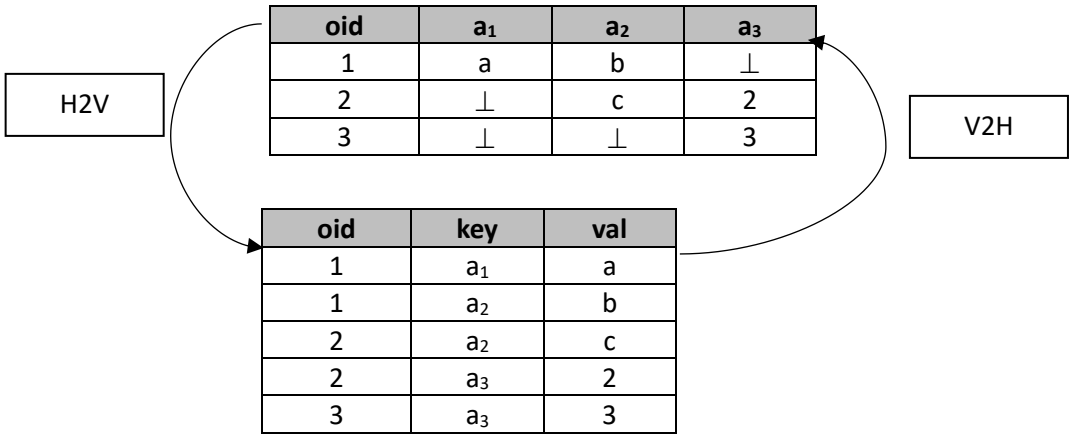
Für alle Argumente ist es zulässig nur eine Auswahl an vordefinierten Werten zuzulassen. Zusätzlich sollen pro Attributwert die Anzahl der Tupel mit gleichem Attributwert beschränkt werden. Vorschlag hierfür ist, jeden Attributwert 5-mal vorkommen zu lassen. Es sollen mindestens zwei unterschiedliche Datentypen (z.B. String & Integer) unterstützt werden. Sie können Attributnamen und Werte generisch halten (z.B. wie in Tabelle 1).

Für die Überprüfung der Aufgabe gilt: Sofern optisch (`Select * from H`) die Daten richtig aussehen: 0.5 P. Für die restlichen Punkte: Legen Sie Sichten im DBMS an, anhand derer Sie zeigen können, dass die restlichen Anforderungen erfüllt sind.

Phase 2 – Gros der Implementierung (5 P)

In dieser Phase realisieren Sie die Operatoren V2H sowie H2V, zeigen ihre Korrektheit, und optimieren die Performance des V2H Operators. Zusätzlich definieren und implementieren Sie den Benchmark anhand dessen V2H & H2V vergleichen werden.

- **Operator Implementierung (3 P):** Implementieren Sie die Operatoren H2V und V2H, wie in der Vorlesung beschrieben. Sie zeigen die Korrektheit der Implementierung mit Hilfe des Toy Beispiels aus der vorhergehenden Phase (1P). D.h. für den ersten Punkt muss es erstmal nur auf dieses Beispiel funktionieren. Für die vollständige Korrektheit der Operatoren muss zusätzlich folgendes gelten:



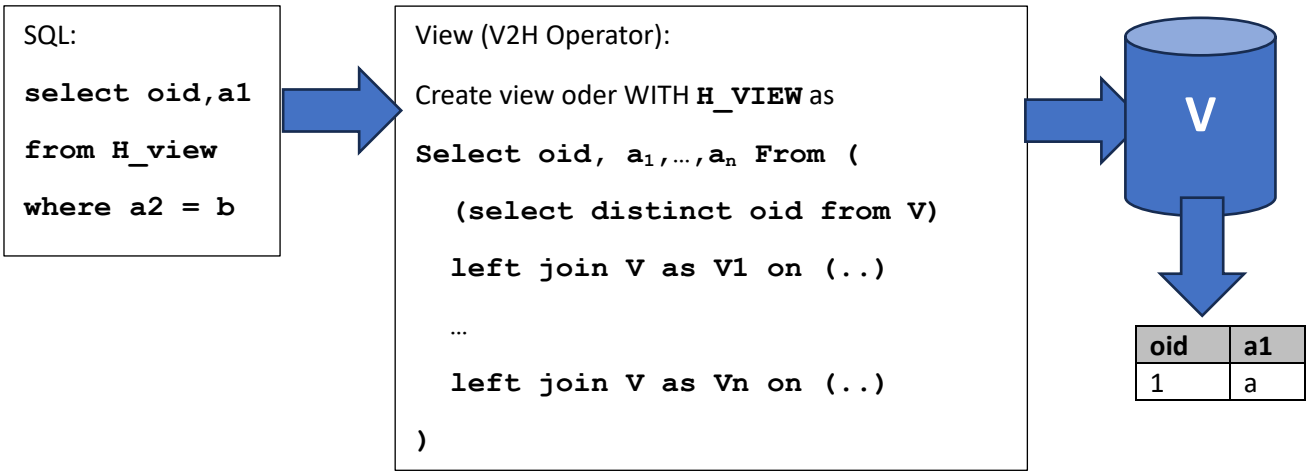
oid	a_1	a_2	a_3
1	a	b	⊥
2	⊥	c	2
3	⊥	⊥	3

oid	key	val
1	a_1	a
1	a_2	b
2	a_2	c
2	a_3	2
3	a_3	3

a. **Horizontal-to-vertical (H2V) (1P):** Dieser Operator soll einmalig angewendet werden, um horizontal gespeicherte Daten in die vertikale Repräsentation zu überführen. Input ist der Name einer beliebigen Relation die mit generate() erschaffen werden kann. Dies bedeutet insbesondere, dass weder Anzahl der Attribute noch die jeweiligen Datentypen bekannt sind. Diese müssen aus dem **DB-Katalog** ausgelesen werden. Zusätzlich müssen attributtypspezifische vertikale Fragmente unterstütz werden.
Es gibt bei dieser Aufgabe keine Vorgabe, wie (und wo) der Operator realisiert wird. Vorschläge sind: Generierung und (einmalige) Ausführung eines SQL-Befehls auf Client Seite. Alternativ können Sie auch eine View auf der vertikalen Tabelle `V` erstellen, in

die sie die Daten einfügen. Bzgl., des letzten Vorschlags ist es möglich, dass nicht alle DBMS updatable views auf self-joins unterstützen.

b. **Vertical-to-horizontal (V2H) (1P):** Grundsätzlich besteht die Idee darin, dass eine Anwendung nicht wissen muss, ob die Daten in horizontal oder vertikal gespeichert werden. Daher müssen Anfrageergebnisse aus der vertikalen wieder in die horizontale Repräsentation überführt werden können. Der Unterschied zum H2V besteht als darin, dass VH2 nach jeder Anfrage auf dem vertikalen Ergebnis ausgeführt werden muss.
Wir gehen davon aus, dass die anfragende SQL auf der horizontalen Darstellung gestellt ist. Also z.B. **nicht** `Select oid, a1 from V where key = a2 and val b`, sondern **`Select oid, a1 from H_VIEW where a2 = b`**. Schematisch ist der Ablauf also folgender:



Hinweis: Die Performanz des Operators kann stark von der Anzahl der Attribute n abhängen. Begrenzen Sie n sinnvoll, so dass die Benchmarks im Folgenden in absehbarer Zeit durchgeführt werden können.

- **Definition des Benchmarks (1.5 Punkte):** Wir sind daran interessiert zu bestimmen für welche Kombination aus Datensatzgröße |H|, Anzahl an Attributen |A|, und Sparsity-Werten S, welche der beiden Repräsentationen vorteilhaft ist. Definieren Sie sinnvoll gewählte Werte für H, A, und S, so dass alle Kombinationen |H| * |A| * |S| für Sie testbar sind. Halten Sie dabei folgende Hinweise ein: (1 P)
  - **a.** Die Datensatzgröße sollte exponentiell steigen und auch im kleinsten Fall > 1000 sein.
  - b. Die Anzahl der Attribute soll äquidistant steigen. Fangen Sie mit kleinen Zahlen z.B. 5 Attributen an.
  - c. Uns interessieren vor allem sparse Datensätze, so dass sich $s = 1-2^{-i}$ mit i >1 anbietet.
  - d. Als Anfragen verwenden wir
    - i. `Select * from H_VIEW where oid = ???` → `Resultatsgröße = 1, mit oid uniform aus [1, max(oid)]`
    - ii. `Select oid from H_VIEW where ai = ???` → `Resultatsgröße ca 5, mit i uniform aus [1,|A|] und ai uniform aus dom(ai)`
  - e. Wir zählen die Anzahl an Queries pro Zeiteinheit. Wählen sie eine sinnvolle Zeiteinheit.

Bestimmen Sie vor jedem Benchmark zusätzlich den benötigten Speicherbedarf für V und H inklusive etwaiger Indexe (0.5 P).

- Optimierung der Anfrageperformanz auf der vertikalen Darstellung (1P): Für die Optimierung können Sie von folgenden Punkten ausgehen.
    a. Es werden keine Daten aus H (bzw. V) eingefügt oder gelöscht
    b. Es werden nur die Anfrage i und ii ausgeführt, in dieser Form.
    c. Am Ende muss beim Client ein korrektes horizontales Ergebnis ankommen.
    d. Indexe dürfen genutzt werden.
    e. Die Operatoren dürfen geändert werden
    f. Es kann Logik an die DB ausgelagert werden, um den Kommunikationsoverhead zu minimieren.

  Nutzen Sie mindestens 2 der Punkte d-f.

Phase 3 – Messen, Auswerten und finale Optimierung

Dies ist die finale Phase des Projekts.

- **Ausführung und Auswertung des Benchmarks (1 P):** Führen Sie Ihren benchmark auf H und der optimierten Variante von V aus Phase 2 aus. Stellen Sie die Ergebnisse grafisch dar und interpretieren Sie es (als Teil des Vortrags).
- **Erstellung einer Anfragespezifischen API (1 P):** In der ersten Implementierung des V2H Operators wird das gesamte Tupel konzeptionell wieder zusammengesetzt und dann die Anfrage ausgeführt. Es ist zu erwarten, dass DBMSe unterschiedlich gut darin sind nur den relevanten Teil des Tupels zusammenzusetzen, bzw. zu erkennen, dass die restlichen left Joins nicht ausgeführt werden müssen. Daher vereinfachen wir nun die Operatoren und Implementieren sie direkt in dem DBMS. Wir nutzen also deren prozedurale Erweiterungen[1]. Konkret heißt das:
    a. Definieren Sie pro Anfragetyp aus Phase 2 eine Funktion im DBMS, welche als Input nur die Argumente (also keine SQL) enthalten. Ein Aufruf von Client-Seite sieht dann z.B. so aus:
        i. `Select * from q_i(42)`
        ii. `Select * from q_ii('ai', 3)`
    b. Das zweite Argument für q_ii sollte vom richtigen Typ sein. Im obigen Fall also Integer. Abkürzungen über typ-spezifische Partitionen von V sind erlaubt.
    c. Rückgabewert ist eine horizontale Tabelle (nicht nur ein Wert).
    d. Für i) ist ein Schema ausreichend, dass nur nicht null Werte enthält (aber nicht erforderlich)
    e. Vergleichen Sie die Anfragepläne der Implementierung aus Phase mit denen der API. Fallen Unterschiede auf (Teil des Vortrags)? Nutzen Sie hierfür im PostgreSQL[2] `Explain Analyze` [SQL]
- **Benchmark Erweiterung (1P):** Führen Sie den Benchmark auf der API durch. Sie können hierzu auch eine weitere Prozedur oder Funktion schreiben, so dass keine Kommunikation mit dem Client mehr nötig ist (0.5P). Vergleichen Sie das Resultat mit der Version aus Phase 2 (0.5 P).
- **Finale Optimierung der vertikalen Darstellung:** Nach Fertigstellung der API, überlegen Sie welche weiteren Optimierungsmöglichkeiten für V bestehen. Testen Sie mindestens 2 davon (jeweils 0.5 P). Negative Resultate sind ok.

[1] https://www.postgresql.org/docs/current/sql-createfunction.html
[2] https://www.postgresql.org/docs/current/sql-explain.html

- **Vorstellung Ihres Projektes:** Stellen Sie ihr Projekt in Form eines 8-10 minütigen Vortrags inklusive Live-Demo der wesentlichen Bestandteile vor.
    a. Generelles Vorgehen und Setting
    b. Zeitplanung & deren Einhaltung
    c. Was lief gut, was nicht?
    d. Kernergebnis & Fazit

  Geben Sie zusätzlich den Code, sowie die Vorstellungspräsentation über Blackboard ab.

Bewertungsmatrix

Bis wann	Was	#Punkte	Σ Phase
Ende Phase 1	Datengenerator für horizontale Daten	0.5	
	Parameterisierbarkeit des Datengenerators bzgl. #Datensätze, Sparsity-Faktor (in %), Anzahl Attribute	1	
	Wahl Programmiersprache / DBMS & Aufsetzen der Infrastruktur	0.5	
	Wahl des Toy-Beispiels & Definition der Korrektheitstests (auf Papier)	0.5	2.5
Ende Phase 2	Demonstration der Korrektheit der Operatoren V2H & H2V am Toy-Beispiel in der Infrastruktur	1	
	V2H Operator Implementation	1	
	H2V Operator Implementation	1	
	Optimierung der Vertikalen Darstellung	1	
	Definition des Benchmarks	0.5	
	Speicherverbrauchsbestimmung	0.5	5
Ende Phase 3	Ausführung und Auswertung des Benchmarks	1	
	Erstellung der API	1	
	Erweiterung des Benchmarks auf die API	1	
	Finale Optimierung der Vertikalen Darstellung	1	
	Vorstellung des Projektes & Fazit	1	5

Sie erhalten Sie oben angegebenen Punkte, wenn zum Ende der Phase vollständig erfüllt sind, sonst erhalten Sie Anteilig Punkte. Kleinere Nacharbreiten und Korrekturen nach Phase 1 und 2 werden individuell vereinbart, so dass die Gesamt-Punkte dennoch erreicht werden können. Nacharbeiten nach Phase 3 sind nicht zulässig.