Sparsity im Ecommerce

Alexander Weilert, Dragana Jovanovic

Generelles Vorgehen

- ***Vorgehensweise:***
  - Termine und Treffen über Discord
    - Besprechungen
    - Gemeinsame Recherche/Codierungen
    - Absprache ToDos fürs nächstes Treffen
  - Codierung: Try, Error & Retry
  - Recherche: StandUp Meetings
    - Erledigungen
    - Lösungswege

Setting:

Getestet auf:

- Prozessor: AMD Ryzen 7 5800 8-Core Prozessor 3801 MHz
- Grafikkarte: AMD Radeon RX 6800 XT (16GB VRAM)
- Arbeitsspeicher: 32 GB
- Betriebssystem: Windows 10 Pro 64Bit
- Entwicklungsumgebungen:
    - pgAdmin4, Java: 21 (Intellij), Postgresql: 42.7.3

Zeitplanung & Einhaltung

- Aufwand:
  - etwas unterschätzt
  - Arbeitsaufwand: 6h/Person a 2 Tage
- Für jede Phase alles erfolgreich erledigt
  - Vorarbeit + Nacharbeit

Phase 1

generate()

- Die Werte String bzw. Integer wechseln nach jedem Attribut/Spalte
- Jeweils max. 5 Ereignisse von einem Wert in der Tabelle
- Erstellt Views für num_tuples, num_attributes und sparsity

generate (int num_attributes, double sparsity, int num_tuples, String create_table, long time)

generateViews (int num_attributes, String create_table)

Toy-Beispiel

Tabelle H:

	oid	a1	a2	a3	a4	a5
1	1	a	1	a	<null>	a
2	2	a	1	a	<null>	<null>
3	3	<null>	1	b	1	b
4	4	b	<null>	<null>	<null>	<null>
5	5	b	1	b	2	<null>
6	6	c	2	c	2	<null>
7	7	c	2	c	2	<null>
8	8	<null>	3	c	3	d
9	9	<null>	3	d	<null>	<null>
10	10	<null>	<null>	<null>	<null>	d

View toy_bsp_null:

	oid	a1	a2	a3	a4	a5
1	1	a	1	a	<null>	a
2	2	a	1	a	<null>	<null>
3	3	<null>	1	b	1	b
4	4	b	<null>	<null>	<null>	<null>
5	5	b	1	b	2	<null>
6	6	c	2	c	2	<null>
7	7	c	2	c	2	<null>
8	8	<null>	3	c	3	d
9	9	<null>	3	d	<null>	<null>
10	10	<null>	<null>	<null>	<null>	d

Phase 2

Horizontal-to-vertical (H2V)

- *Version 1*: alles in einer Tabelle (Nutzung von varchar)
- *Version 2*: aufgeteilt in int & varchar Tabellen

H2V_String

	oid	key	val
1	1	a5	a
2	3	a5	a
3	4	a1	a
4	5	a1	a
5	6	a1	a
6	7	a1	b
7	7	a5	b
8	8	a5	<null>
9	9	a1	b
10	9	a3	b
11	9	a5	b
12	10	a1	c
13	10	a5	c

H2V_Integer

	oid	key	val
1	1	a2	1
2	2	a2	1
3	3	a2	1
4	3	a4	1
5	4	a4	1
6	5	a2	2
7	10	a2	2

Vertical-to-horizontal (V2H)

- *Version 1*: Hardcodiert aus einzelner Tabelle ausgelesen
- *Version 2*: Mittels join einzelne Werte ausgelesen

	oid	a1	a2	a3	a4	a5
1	1	<null>	1	<null>	<null>	a
2	2	<null>	1	<null>	<null>	<null>
3	3	<null>	1	<null>	1	a
4	4	a	<null>	<null>	1	<null>
5	5	a	2	<null>	<null>	<null>
6	6	a	<null>	<null>	<null>	<null>
7	7	b	<null>	<null>	<null>	b
8	8	<null>	<null>	<null>	<null>	<null>
9	9	b	<null>	b	<null>	b
10	10	c	2	<null>	<null>	c

Optimization

- Materialized View:

```
CREATE MATERIALIZED VIEW mv_v2h
        AS SELECT * FROM v2h_string
                WHERE key = 'a1'
```

- Indexing:

```
CREATE INDEX idx_key_h2v
        ON h2v_string (oid)
```

- Details wurden dazu in der Übungsstunde schon geklärt

Benchmark

- Benchmark wurde getestet immer mit Standardwerten:
  - Tuples: 1000, Attributes: 5, Sparsity: 0.5
  - Wobei in der Phase des Benchmarks die Werte:

num_tuples	querryies after 10 s	v total querries	table created in ms	time vertical in ms	time horizontal in ms	total time in ms	table size in kb	index size in kb	v_str table size in kb	v_str index size in kb	v_integer table size in kb	v_integer index size in kb
10000	10000	25243	2181	164	14	2371	720	240	1016	320	688	224
20000	20000	50726	8082	203	15	8312	1400	456	1968	616	1344	440
30000	22092	76089	18798	283	27	19118	2072	672	2928	912	1976	640
40000	22233	101370	33237	380	29	33659	2760	896	3880	1208	2616	848
50000	22203	126349	52827	445	30	53315	3432	1112	4816	1496	3256	1056
100000	21833	253393	248664	924	64	249665	6840	2208	9624	3000	6480	2104

num_attributes	querryies after 10 s	v total querries	table created in ms	time vertical in ms	time horizontal in ms	total time in ms	table size in kb	index size in kb	v_str table size in kb	v_str index size in kb	v_integer table size in kb	v_integer index size in kb
5	1000	2606	77	56	3	148	120	40	152	48	120	40
10	1000	5031	105	58	4	179	136	40	208	64	200	56
50	1000	25179	300	153	18	481	216	40	712	128	712	128
100	1000	50043	541	181	26	760	336	40	1320	208	1336	208
250	1000	124606	1108	429	84	1632	712	40	3176	440	3168	440
500	1000	249712	2293	854	177	3335	1408	40	6336	896	6320	888
750	1000	375966	3538	1354	347	5252	2080	40	9464	1312	9496	1320
1000	1000	500065	5118	1782	553	7472	2728	40	12000	1784	12000	1784
1250	1000	624395	6754	2523	765	10066	3992	40	15000	2120	15000	2128

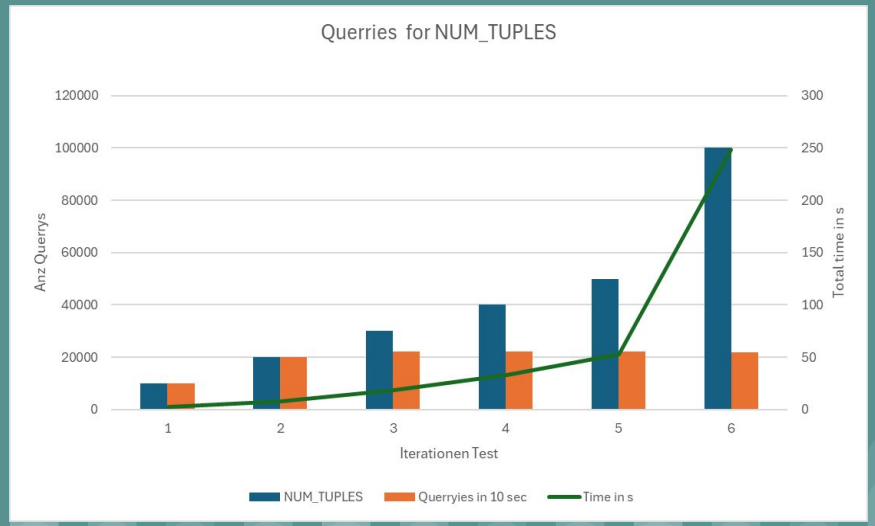
sparsity	querryies after 10 s	v total querries	table created in ms	time vertical in ms	time horizontal in ms	total time in ms	table size in kb	index size in kb	v_str table size in kb	v_str index size in kb	v_integer table size in kb	v_integer index size in kb
0.25	1000	3710	192	44	2	248	120	40	192	56	152	48
0.125	1000	4398	37	46	2	95	128	40	216	64	168	56
0.0625	1000	4705	35	46	2	92	128	40	224	64	176	56
0.03125	1000	4849	38	51	2	101	128	40	224	64	176	56
0.015625	1000	4932	35	47	2	93	128	40	224	64	176	56
0.0078125	1000	4959	38	50	2	101	128	40	232	64	176	56
0.00390625	1000	4971	35	46	2	92	128	40	232	64	176	56
0.001953125	1000	4994	38	50	2	100	128	40	232	64	176	56
9,77E-256	1000	4994	36	46	2	94	128	40	232	64	176	56

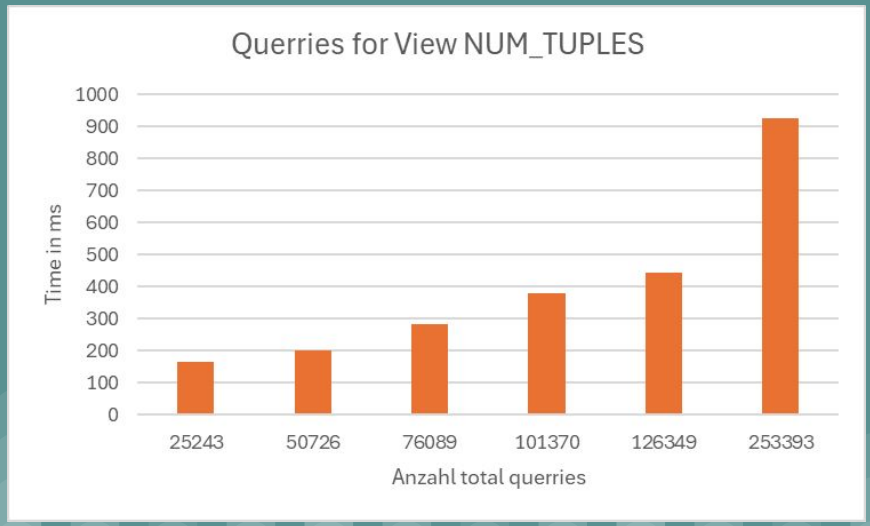
Phase 3

Graph: Tuples

- Testdaten:
  - Attribute: 5
  - Sparsity: 0.5
  - Tuples: Mit jeder Iteration steigend

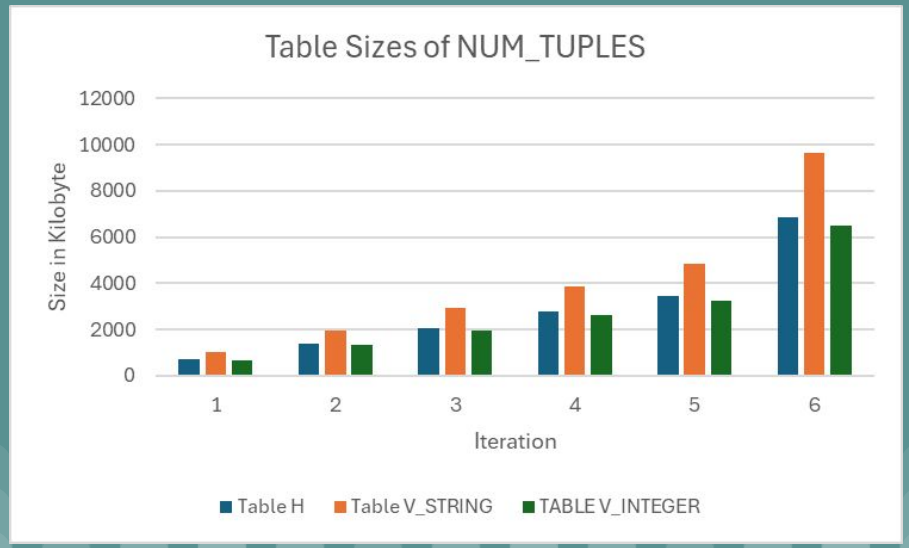
- Wie zu sehen:
  - Steigende Tupel = exponentiell steigende Laufzeit
  - Für Anzahl Queries für V gilt das gleiche





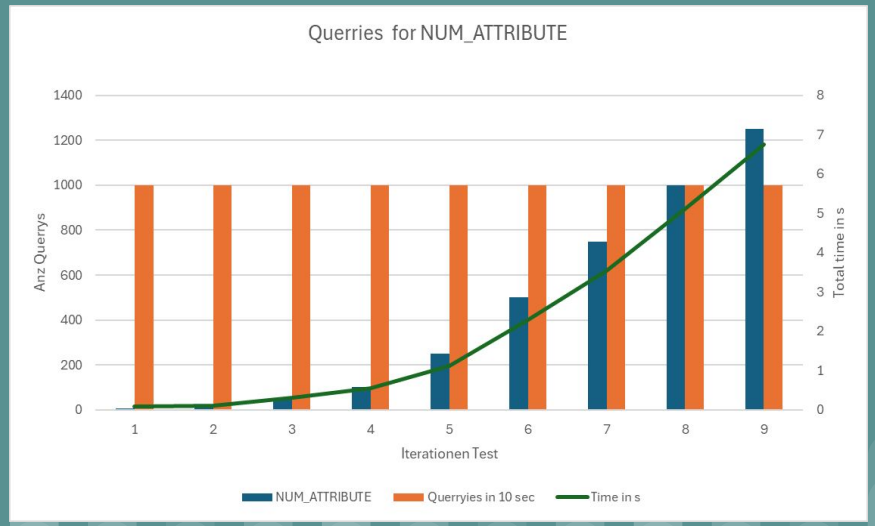
Graph: Tuples

- Testdaten:
  - Attribute: 5
  - Sparsity: 0.5
  - Tuples: Mit jeder Iteration steigend
- Wie zu sehen:
  - Wie mehr Tupel verwendet werden, desto mehr speicher wird benötigt

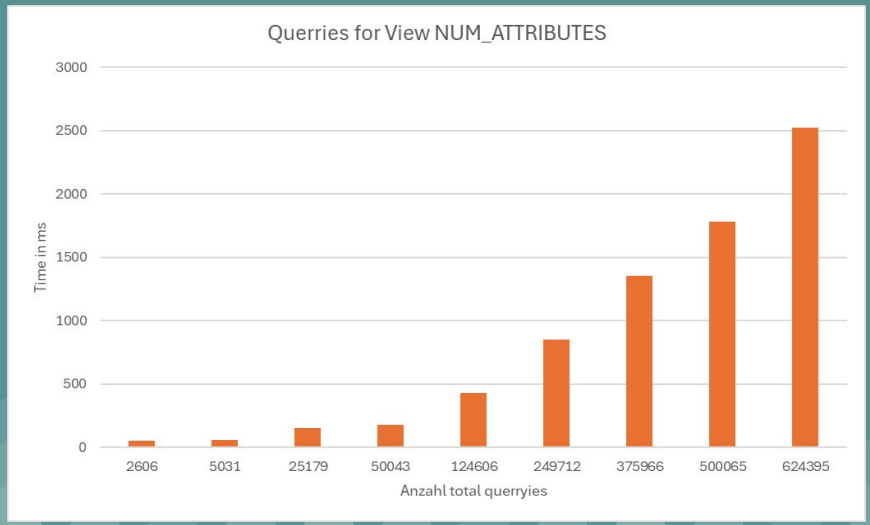


Graph: Attribute

- Testdaten:
  - Attribute: Mit jeder Iteration steigend
  - Sparsity: 0.5
  - Tuples: 1000

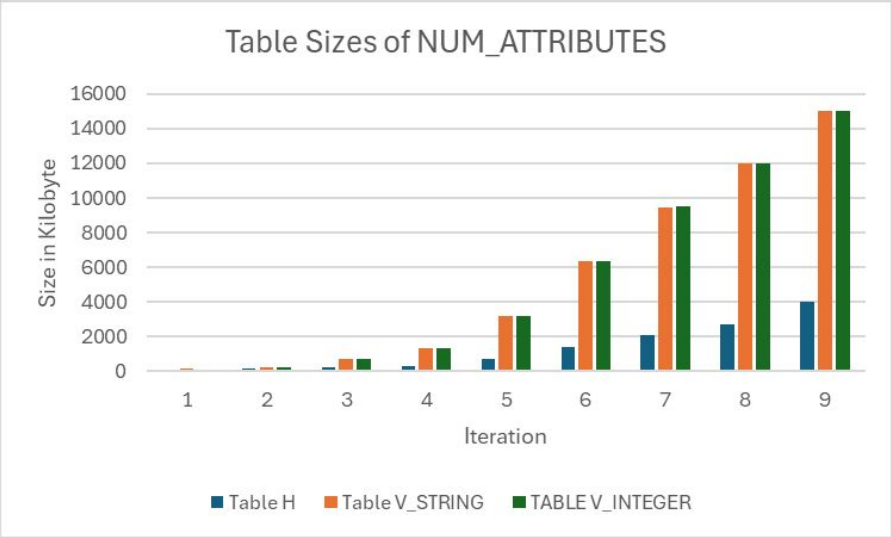


- Wie zu sehen:
  - Steigende Attribute = Steigende Laufzeit,
    - Jedoch trotzdem effizient
  - Anzahl der Queries für V deutlich höher als bei den Tupel



Graph: Attribute

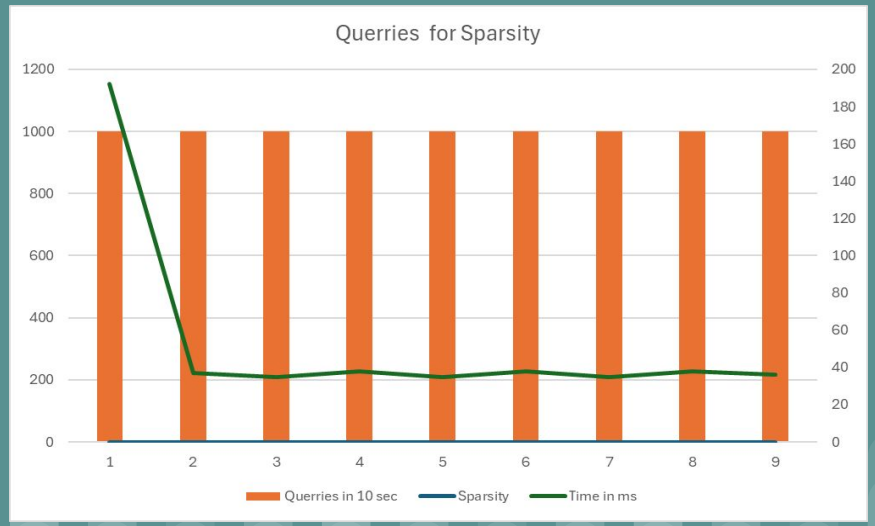
- Testdaten:
  - Attribute: Mit jeder Iteration steigend
  - Sparsity: 0.5
  - Tuples: 1000
- Wie zu sehen:
  - Mit einer erhöhten Anzahl von Attributen wird deutlich mehr Speicher in der View benötigt, als bei einer höheren Anzahl von Tupel

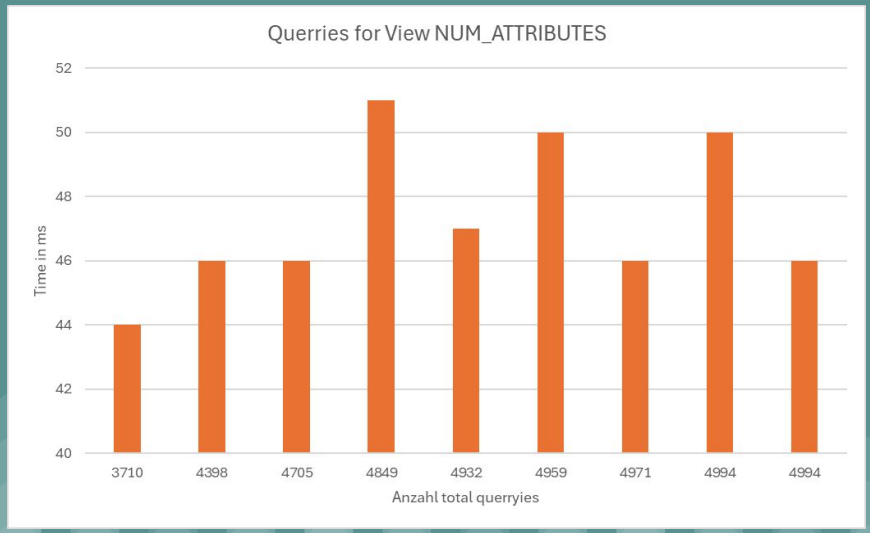


Graph: Sparsity

- Testdaten:
  - Attribute: 5
  - Sparsity: Mit jeder Iteration sinkend
  - Tuples: 1000

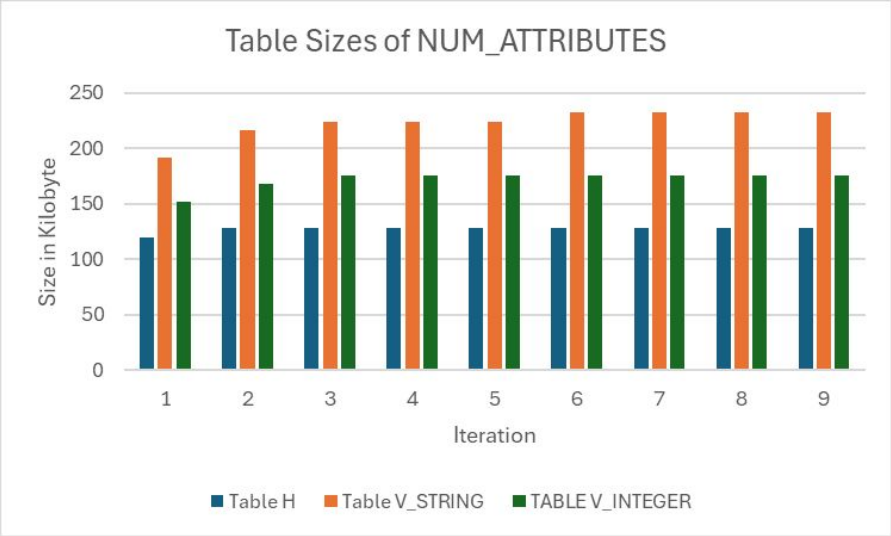
- Wie zu sehen:
  - Zeit bleibt immer gleich, außer man hat besitzt viele NULL Objekte





Graph: Sparsity

- Testdaten:
  - Attribute: 5
  - Sparsity: Mit jeder Iteration sinkend
  - Tuples: 1000
- Wie zu sehen:
  - Speicherverbrauch bleibt auch fast identisch
  - Logischerweise :
    - mehr NULL-Objekte = weniger Speicher



API

- Language Support vom integrierten plpgsql
- Logik aus Anfragen aus Phase 2 in Funktionen umgeschrieben
- Funktion q_i (int)
    - Durchläuft alle Spalten einer View mit der “oid” einer beliebigen Zahl
    - Gibt alle Werte aus, die die gesuchte OID besitzt
- Funktion q_ii (varchar, int)
    - Funktionsaufruf:
        - Varchar = a2, a4, ...
        - int = beliebige Zahl
    - Selektiert alle OIDs, die die Werte “VARCHAR” und “INT” besitzen,

Benchmark API

- Funktion *benchmark()* erstellt
  - Berechnet in ms wie lange die jeweiligen Funktionen brauchen
  - Gleiche Werte für Funktionen q_i, q_ii und Abfragen aus Phase 2 benutzt
- Vergleich zu Phase 2 Anfragen:
  - Planungs-Zeit kürzer
    - Datenbank kann schneller auf die Abfragen reagieren
  - Erhöhte Anzahl an Queries
- Funktionen q_i & q_ii:
  - “Explain Analyze” funktioniert nicht richtig, keine Ausgabe des Execution Plans

Optimization Part 2

- Funktion q_i:
  - Statt doppelten Selektionen könnte man Joins nutzen
  - Reduzierung der Anzahl an Abfragen
  - Vermeiden von unnötigen Loops auf Selektionen, da diese viel Performance verbrauchen
- Funktion q_ii:
  - Vermeiden von unnötigen Loops auf Selektionen, da diese viel Performance verbrauchen
    - Dadurch konnte zwar die Planning Time, um die Hälfte reduziert werden
    - Aber: Execution hat doppelt so lange gedauert

Was lief gut, was nicht?

- ***Erfolge:***
  - Kommunikation & Teamwork
  - Vorbereitung auf den Übungsteil der UV
  - Implementierung möglichst dynamisch gehalten
- ***Herausforderungen:***
  - Vereinzelte Schwierigkeiten beim Aufgabenverständnis

Kernergebnis & Fazit

- Selbstständiges Entwickeln mittels Datenbanken unter einer neuen Entwicklungsumgebung (Java)
  - Manchmal nicht so einfach wie man vermutet
- Verständnis von Optimierungen und dem Aufbau einer SQL-Abfrage
- Viele neue Möglichkeiten gelernt Daten ...
  - ... zu verwenden
  - ... zu verwerten
  - ... zu löschen
  - ... oder zu speichern
- Allgemeine Begeisterung von der Anzahl an Möglichkeiten Queries zu bearbeiten und zu verwenden

Danke für eure Aufmerksamkeit!