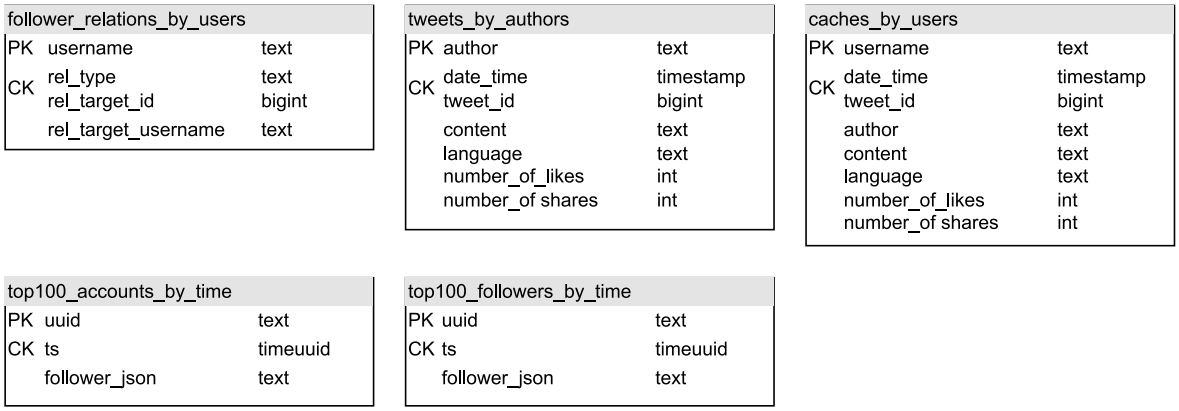
Datenmodellierung

Zunächst ist es wichtig zu erwähnen, dass in C* die Datenmodellierung nicht die klassischen Konzepte der Datenmodellierung relationaler Datenbanken verfolgt werden. Statt sich anhand eines ER-Diagramms die benötigten Entitäten und deren Beziehungen zu überlegen, verfolgt man hier den Ansatz, als erstes die Queries zu entwerfen. Dies hat unter anderem den Hintergrund, dass in C* gewisse Operationen, welche in diversen relationalen Datenbanken als selbstversändlich angesehen werden, schlichtweg nicht unterstützt werden. Schwierige Operationen sind bspw. **GROUP BY** und **ORDER BY**, welche nur unter erheblichen Einschränkungen, im Vergleich zu relationalen Datenbanken, verwendet werden können.

Dies hat letztendlich zur Folge, dass für so gut wie jede Abfrage eine eigene Tabelle entworfen wird, welche die Daten in einem optimalen Schema bereitstellt.

Auf Basis dieser Überlegungen, hat sich das Team letztendlich für das in der folgenden Abbildung dargestellte Datenmodell entschieden:



Im Wesentlichen werden die Daten in den Tabellen **follower_relations_by_users** und **tweets_by_authors** gespeichert.

follower_relations_by_users:

- Den Partition-Key stellt hier der **username** dar, welcher an die Bedingung geknüpft ist, unique zu sein
- Der Clustering-Key **rel_type** gliedert die gespeicherten Follower Relationen in **"follower"** und **"follows"** Relationen
- Durch die doppelte Speicherung einer Relation, jeweils einmal als **"follows"** beim folgenden und als **"follower"** beim verfolgten Account, ermöglicht eine schnelle Abfrage vin bspw. allen verfolgten oder folgenden Accounts eines bestimmten Nutzers
- Da in C* jede Kombination aus Partikion-Key und Clustering-Key unique sein muss, damit ein **INSERT** nicht zu einem **UPDATE** wird, wurde für den Clustering-Key ebenfalls die Id des Ziels der Relation als Attribut mit aufgenommen (**rel_target_id**)

- Darüber hinaus wird für jede gespeicherte Relation der **rel_target_username** mit gespeichert, welche anhand eines Skripts (**/cleaning_scripts/follower_relations.py**) generiert wurden

tweets_by_authors

- Den Partition-Key stellt hier der Name des Authors (**author**) dar, für welchen nun eine beliebige Menge an Tweets gespeichert werden kann (begrenzt lediglich von den systembedingten Limitierungen von C*)
- Hiermit wird sichergestellt, dass die Tweets desselben Authors auch immer auf der gleichen Maschine gespeichert werden, was eine Bulk Abfrage erleichtert/beschleunigt
- Als Clustering-Keys werden hier die **date_time** zu der der Tweet abgesetzt wurde, sowie eine eindeutige **tweet_id** gespeichert
- darüber hinaus beinhaltet jeder Datensatz weitere Informationen über den jeweiligen Tweet, wie bspw. den Inhalt (**content**)

Die Tabelle **caches_by_users** dient der Speicherung der nutzerspezifischen Caches für die Startseite der Anwendung. Ein Cache für einen Nutzer kann hierbei über das Skript **/runtime_scripts/init_cache_for_user.py** initialisiert werden. Hierbei werden für den jeweiligen **username** (Partition-Key) die Tweets der verfolgten Accounts aus der Tabelle **tweets_by_authors** extrahiert und in chronologische Reihenfolge in dessen Cache gespeichert. Eine Abfrage auf dieser Tabelle ist um einiges schneller, als die erneute Extraktion der relevanten Tweets, bei jedem Abruf der Startseite. Beim Einfügen eines neuen Tweets in **tweets_by_authors** anhand des Skripts **/runtime_scripts/insert_new_tweets.py** wird dieser zudem ebenfalls in den Cache eines jeden Nutzers eingefügt, welcher als Follower des Authors in der Tabelle **follower_relations_by_users** gespeichert ist.

Die Tabellen **top100_accounts_by_time** und **top100_followers_by_time** dienen als Time-Series-Tables für die Umsetzung der Queries 2 und 3.