**Stored programs pseudocode**

**parse\_event Procedure**

1. Initialize variables for event attributes (e.g., event type, account ID, event ID, etc.)
2. Determine the maximum data feed ID from **SM\_data\_feed** table.
3. Iterate over each row in the **SM\_data\_feed** table using a while loop.
   * For each iteration, retrieve event-related data (e.g., SMUID, event ID, event category codes, event time, hashtags, content) for the current **data\_feed\_id**.
   * Parse the event category codes (**evcatcodes**) to extract two category values, adjusting for out-of-range values.
   * Format **event\_time** to a proper **DATETIME** format or set a default value if null.
   * Use a **CASE** statement to decide which upsert procedure to call based on the event category (**evcat1**):
     + If **evcat1** is 0 or 4, call **UpsertSocialEvent**.
     + If **evcat1** is between 1 and 3, call **UpsertDeviceLog**.
     + Otherwise, call **UpsertActions**.
   * Increment the index to move to the next row.

**UpsertSocialEvent Procedure**

1. Check if an entry with the given **event\_id** exists in the **social\_event** table.
2. If the entry exists, update the existing record with the new data.
3. If the entry does not exist, insert a new record into the **social\_event** table with the provided data.

**UpsertDeviceLog Procedure**

1. Check if an entry with the given **event\_id** exists in the **device\_event\_log** table.
2. If the entry exists, update the existing record with the new data.
3. If the entry does not exist, insert a new record into the **device\_event\_log** table with the provided data.

**UpsertActions Procedure**

1. Check if an entry with the given **event\_id** exists in the **social\_action** table.
2. If the entry exists, update the existing record with the new data.
3. If the entry does not exist, insert a new record into the **social\_action** table with the provided data.

**process\_hashtag Procedure**

1. Determine the maximum data feed ID from **SM\_data\_feed** table.
2. Iterate over each row in the **SM\_data\_feed** table using a while loop.
   * For each iteration, retrieve the event ID and hashtags for the current **data\_feed\_id**.
   * Split the hashtags string into individual tags.
   * For each tag, call **UpsertHashtags** to either update an existing tag or insert a new one, linking it with the event.
   * Increment the index to move to the next row.

**UpsertHashtags Procedure**

1. Check if the provided hashtag already exists in the **hashtags** table.
2. If the hashtag exists, retrieve its ID.
3. If the hashtag does not exist, insert it into the **hashtags** table and retrieve the new ID.
4. Check if an association between the event and the hashtag already exists in the **event\_tag** link table.
5. If the association does not exist, insert a new record into the **event\_tag** table to link the event with the hashtag.

**process\_ip\_address Procedure**

1. Declare a variable to hold the IP address (**v\_ip\_address**).
2. Retrieve an IP address from the **SM\_data\_feed** table and store it in **v\_ip\_address**.
3. Check if **v\_ip\_address** is null.
   * If yes, set **v\_ip\_address** to the default value of '0.0.0.0'.
4. Remove all non-numeric characters from **v\_ip\_address**, except for periods.
5. Insert the processed **v\_ip\_address** into the **ip\_address\_table**.

**process\_location Procedure**

1. Declare variables for person ID, account ID, location, and location ID.
2. Retrieve **person\_id** and **account\_id** from the **social\_account** table where **SMUID** matches **p\_SMUID**.
3. Retrieve **location** from the **SM\_data\_feed** table where **SMUID** matches **p\_SMUID**.
4. If **v\_location** is not null:
   * Check if the location already exists in the **address** table for the given **person\_id** and **location**.
   * If the location does not exist (**v\_location\_id** is null), insert a new record into the **address** table with **person\_id**, **account\_id**, and **location**.

**process\_SMUID Procedure**

1. Declare a variable for the social account ID (**SM\_social\_account\_id**).
2. Retrieve **social\_account\_id** from the **social\_account** table where **SMUID** matches **p\_SMUID**.
3. If **SM\_social\_account\_id** is null:
   * Insert a new record into the **social\_account** table with **SMUID**.
   * Set **SM\_social\_account\_id** to the last inserted ID.

**add\_device Procedure**

1. Declare variables for account ID and device ID.
2. Call **processSMUID** with **p\_SMUID** to process the social media user ID and retrieve the account ID.
3. Set **v\_account\_id** to the output of **processSMUID**.
4. If **p\_device** is not null:
   * Retrieve **device\_id** from the **device\_make** table where **make\_name** matches **p\_device**.
   * If **v\_device\_id** is null (the device make does not exist):
     + Insert a new make into **device\_make** with **p\_device**.
     + Insert a new model into **device\_model** with "UNKNOWN" as the model name and the last inserted make ID as **make\_id**.

Insert a new device record into **device** with **p\_ip\_address**, the last inserted model ID as **model\_id**, and **v\_account\_id**.