EPL REFERENCE : NAMED WINDOWS AND TABLES

JAVAID MASOOMA

NIANG OUMOUL

SABEUR ANISSA

OVERVIEW

- EPL (Extended Programming Language) is a database management system that allows users to share state between statements using named windows and tables. Named windows are globally-visible data windows that can be inserted-into and deleted-from by multiple statements, and can be decorated to preserve original events. Tables, on the other hand, are globally-visible data structures that hold state and can store aggregation state. They can be used to create and update shared aggregation states, and can be used to query the aggregation state conveniently.

  In summary, named windows are the preferred approach for sharing data windows between statements, while tables may be preferable for rows organized by primary key or holding aggregation state. Statements allow the combined use of both types of data structures. The difference lies in their capabilities and semantics, but both offer stateful capabilities and can be used in various scenarios.

NAMED WINDOW USAGE

- The create window statement is a crucial tool in SQL and EPL for creating named windows. It specifies a window name, one or more data windows, and the type of event to hold in the named window. Two syntaxes are available for creating a named window: modeling after an existing event type or an existing named window, and providing a list of column names and column types. A new named window starts up empty and must be explicitly inserted into by one or more statements. The statement posts events inserted into the named window as new data and deletes or expires out of the data window as the remove stream. The named window contents can be iterated on via the pull API to obtain the current contents of a named window. The syntax for creating a named window by modeling after an existing event type is as follows: [context context_name]

  The window_name assigned to the named window can be any identifier, and the window_spec are one or more data windows that define the expiry policy for removing events from the named window.

TABLE USAGE

- The create table statement is a crucial part of SQL Server, allowing users to create a new table that can be aggregated using into table, on-merge statement, or insert into. The syntax for creating a table includes the table name, column names, types, and primary key columns. The table_name can be any identifier, and the context keyword can be specified. Column types can be non-aggregating or aggregate, with non-aggregating column types being the same as variable types. Primary key keywords can be added after each column type, designating the column as a primary key. The table can hold a numAttempts count aggregation state and a column named active of type boolean, per ipAddress and userId. The create table statement does not provide output to listeners, but the table contents can be iterated on via the pull API to obtain the current contents of a table. All aggregation functions can be used as column types for tables, and type information must be provided when required.

TRIGGERED SELECT : THE ON SELECT CLAUSE

- The on select clause is a one-time, non-continuous query performed on a named window or table every time a triggering event arrives or a triggering pattern matches. The syntax for the on select clause is as follows:

  ```
  on event_type[(filter_criteria)] [as stream_name]
  [insert into insert_into_def]
  select select_list
  from window_or_table_name [as stream_name]
  [where criteria_expression]
  [group by grouping_expression_list]
  [having grouping_search_conditions]
  [order by order_by_expression_list]
  ```

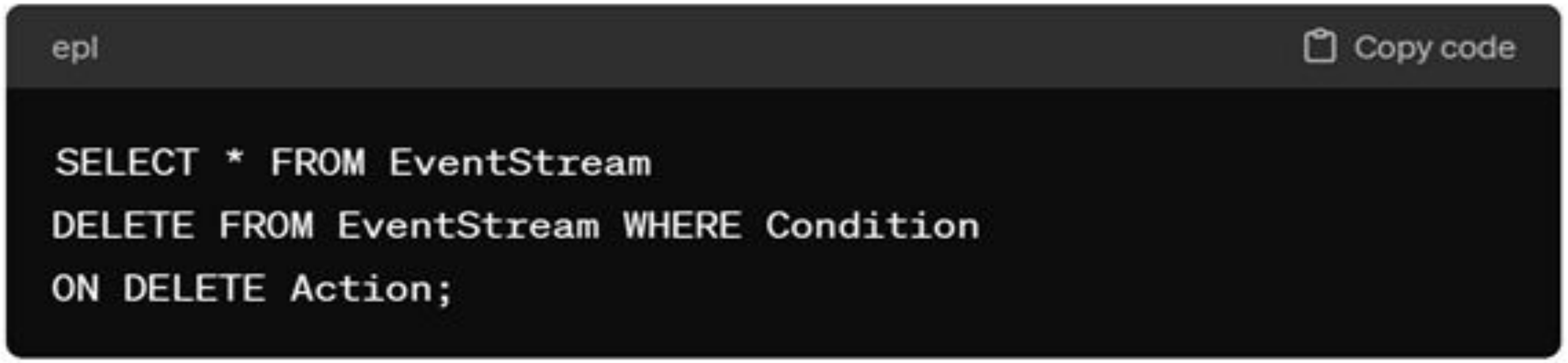
  The select clause is described in the next part, “Choosing Event Properties and Events: The Select Clause”. Merging Streams and Continuous Insertion: The Insert Into Clause works as described in Section 5.10, “Merging Streams and Continuous Insertion: The Insert Into Clause”. The properties of the triggering event or events are available in the select clause and all other clauses.

  Statements that use tables and named windows work the same, with examples provided using the OrdersNamedWindow named window and the SecuritySummaryTable table.

UPDATING DATA : THE ON UPDATE CLAUSE

- The "On Delete" clause in EPL is used to specify the behavior that should occur when an event matching certain criteria is deleted from the event stream. It allows developers to define actions or processes to execute when events are removed from the system. This can be useful for managing resources, updating state, or triggering further events based on the deletion of specific events.

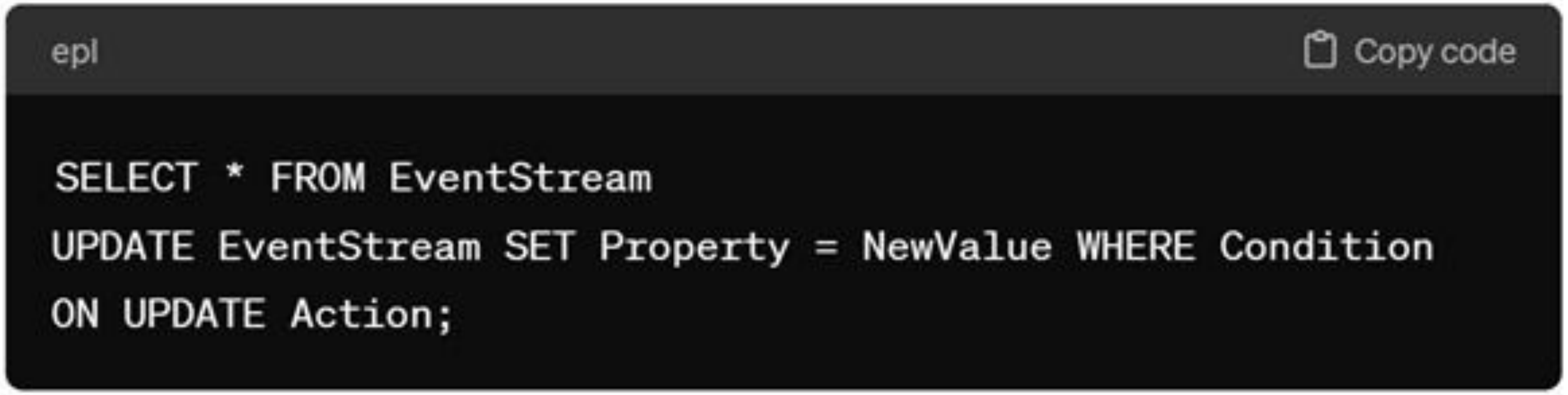
- SELECT * FROM EventStream: This is the query to select events from the event stream.
- DELETE FROM EventStream WHERE Condition: This specifies the condition under which events will be deleted from the stream. Only events matching this condition will trigger the "On Delete" action.
- ON DELETE Action: This is the action or set of actions to be performed when an event is deleted from the stream based on the specified condition



```
SELECT * FROM EventStream
DELETE FROM EventStream WHERE Condition
ON DELETE Action;
```

- "On Update" clause is used to define actions or processing logic that should occur when an event in the event stream is updated or modified.

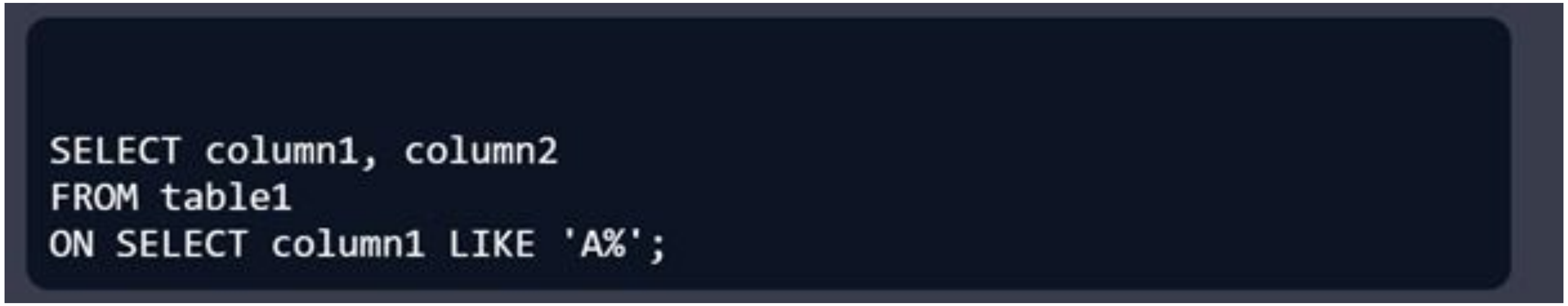
- SELECT * FROM EventStream: This is the query to select events from the event stream.
- UPDATE EventStream SET Property = NewValue WHERE Condition: This specifies the condition under which events will be updated in the stream. Only events matching this condition will trigger the "On Update" action, and the specified property will be updated with the new value.
- ON UPDATE Action: This is the action or set of actions to be performed when an event is updated in the stream based on the specified condition.



```epl
SELECT * FROM EventStream
UPDATE EventStream SET Property = NewValue WHERE Condition
ON UPDATE Action;
```

- The "On select" clause in an EPL (Event Processing Language) statement is used to define a set of conditions that must be met for an event to be selected for processing. When an event enters the system, it is evaluated against the conditions specified in the "On select" clause. If the event meets all the conditions, it is selected for further processing

- In this example, the query is selecting column1 and column2 from table1 where column1 starts with the letter 'A'. The ON SELECT clause is used to filter the rows before they are returned in the result set.



```
SELECT column1, column2
FROM table1
ON SELECT column1 LIKE 'A%';
```

TRIGGERED UPSERT

- What is the use of MERGE statement in SQL Server?

Merge statement introduced allows us to perform Inserts, Updates and Deletes in one statement. This means we no longer have to use multiple statements for performing Insert, Update and Delete.

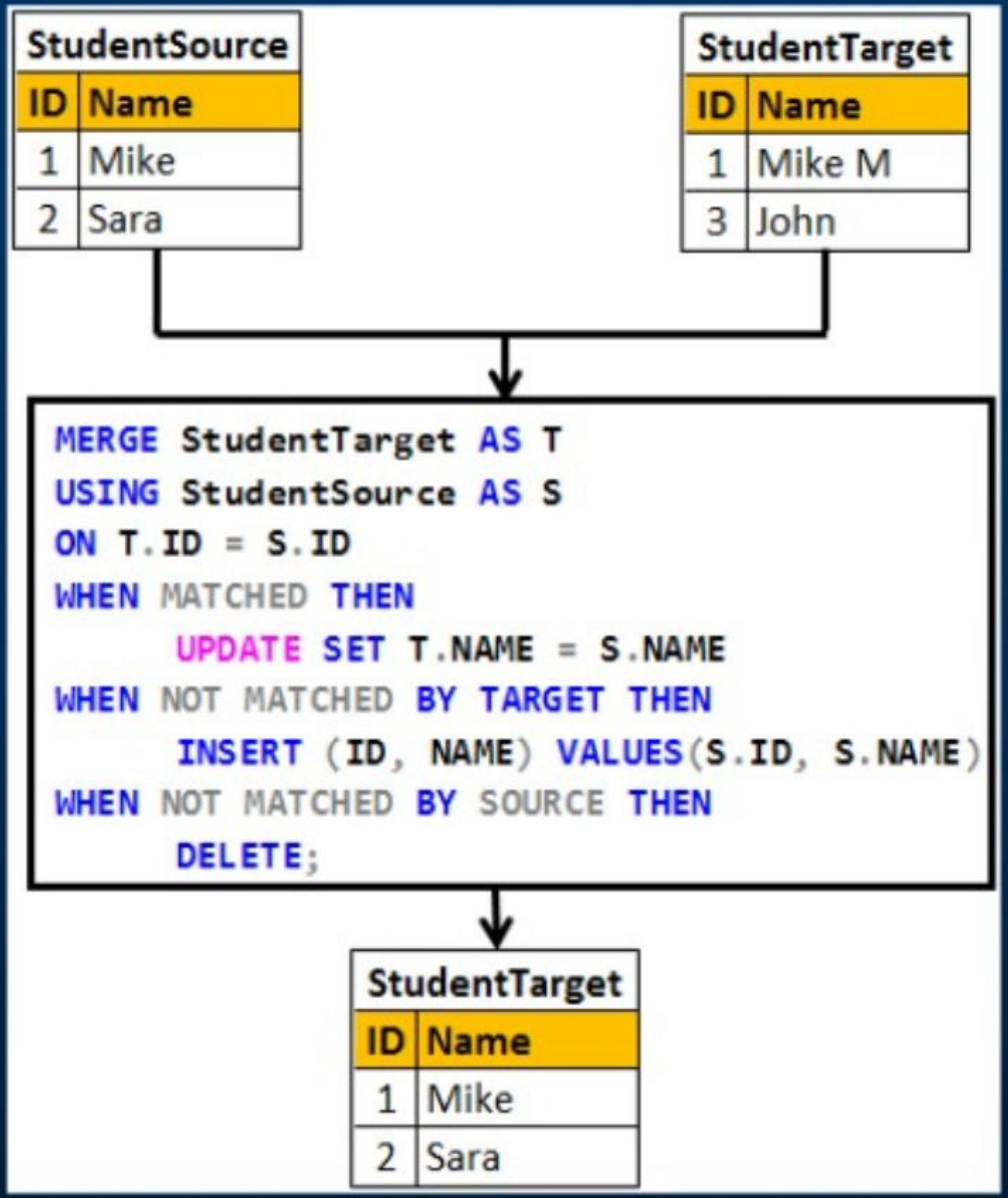
With merge statement we require 2 tables

1. Source Table - Contains the changes that needs to be applied to the target table
2. Target Table - The table that require changes (Inserts, Updates and Deletes)

The merge statement joins the target table to the source table by using a common column in both the tables.

Based on how the rows match up as a result of the join, we can then perform insert, update, and delete on the target table.

```
MERGE [TARGET] AS T
USING [SOURCE] AS S
   ON [JOIN_CONDITIONS]
 WHEN MATCHED THEN
      [UPDATE STATEMENT]
 WHEN NOT MATCHED BY TARGET THEN
      [INSERT STATEMENT]
 WHEN NOT MATCHED BY SOURCE THEN
      [DELETE STATEMENT]
```



StudentSource

ID	Name
1	Mike
2	Sara

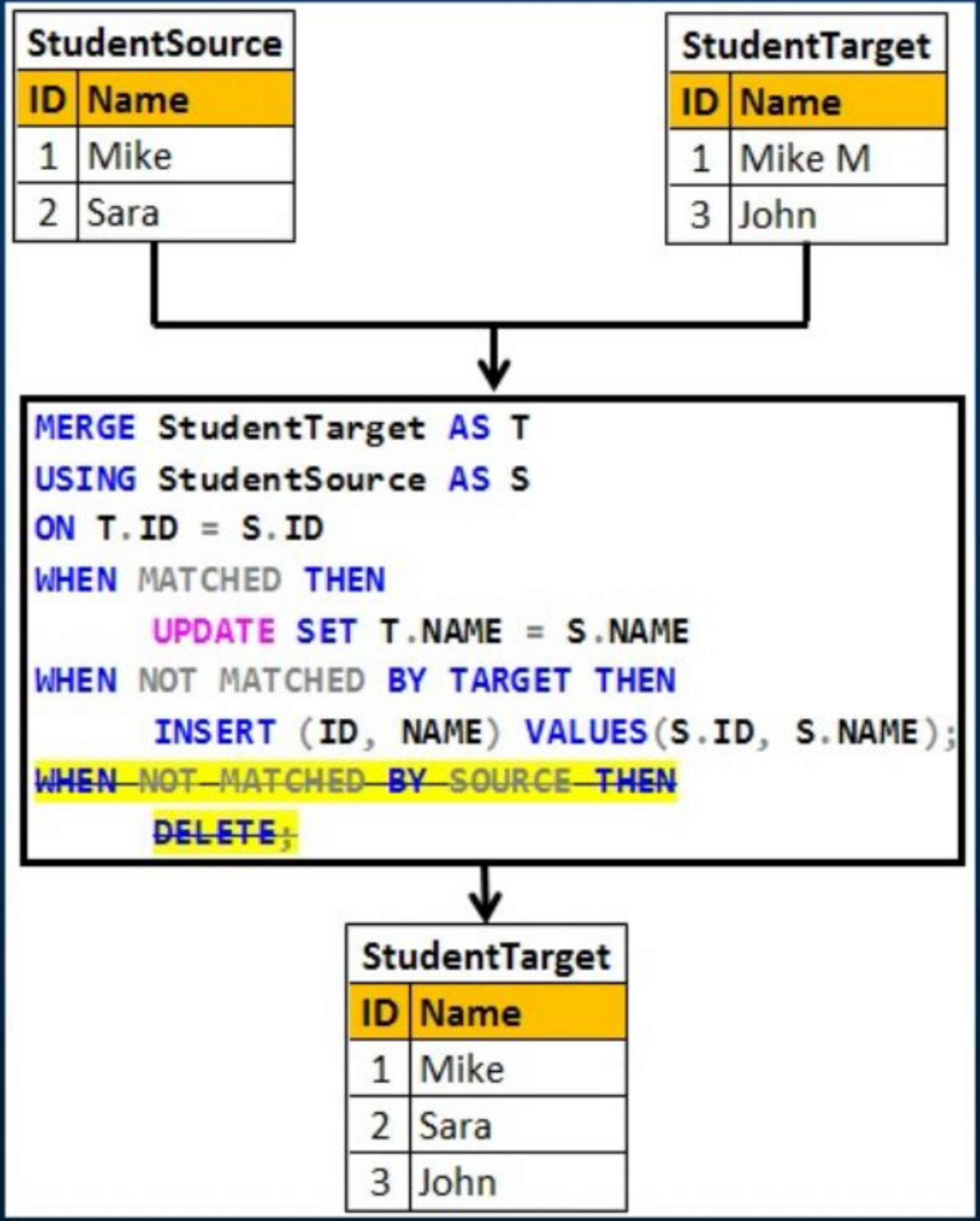
StudentTarget

ID	Name
1	Mike M
3	John

```
MERGE StudentTarget AS T
USING StudentSource AS S
ON T.ID = S.ID
WHEN MATCHED THEN
     UPDATE SET T.NAME = S.NAME
WHEN NOT MATCHED BY TARGET THEN
     INSERT (ID, NAME) VALUES(S.ID, S.NAME)
WHEN NOT MATCHED BY SOURCE THEN
     DELETE;
```

StudentTarget

ID	Name
1	Mike
2	Sara



StudentSource	
ID	Name
1	Mike
2	Sara

StudentTarget	
ID	Name
1	Mike M
3	John

```
MERGE StudentTarget AS T
USING StudentSource AS S
ON T.ID = S.ID
WHEN MATCHED THEN
      UPDATE SET T.NAME = S.NAME
WHEN NOT MATCHED BY TARGET THEN
      INSERT (ID, NAME) VALUES(S.ID, S.NAME);
~~WHEN NOT MATCHED BY SOURCE THEN~~
      ~~DELETE;~~
```

StudentTarget	
ID	Name
1	Mike
2	Sara
3	John