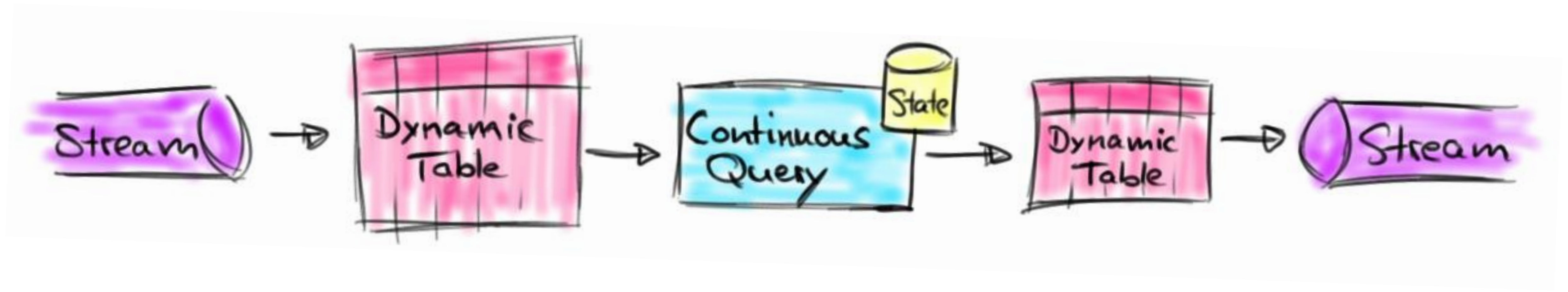
Querying Dynamic Tables with SQL

Flink SQL Training

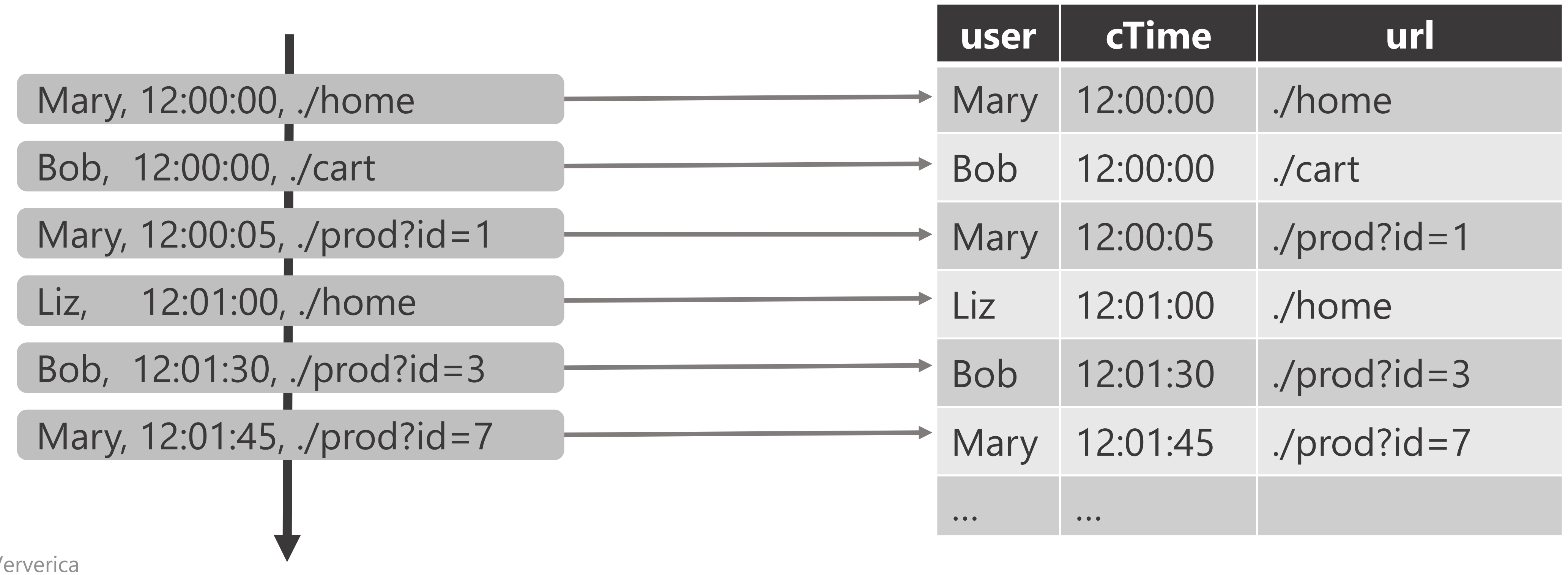
https://github.com/ververica/sql-training

Continuous Queries in Flink



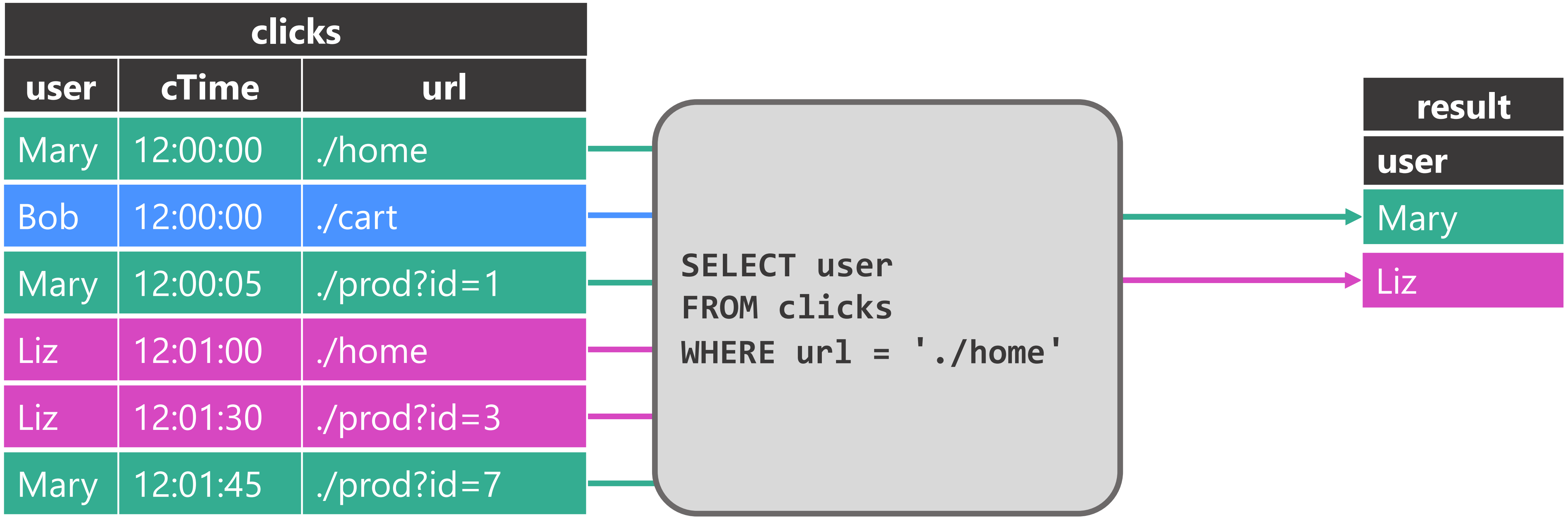
Stream → Dynamic Table: INSERT

- Append mode
  - Stream records are appended to table
  - Table grows as more data arrives


Mary, 12:00:00, ./home
Bob, 12:00:00, ./cart
Mary, 12:00:05, ./prod?id=1
Liz, 12:01:00, ./home
Bob, 12:01:30, ./prod?id=3
Mary, 12:01:45, ./prod?id=7

user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:00:05	./prod?id=1
Liz	12:01:00	./home
Bob	12:01:30	./prod?id=3
Mary	12:01:45	./prod?id=7
…	…	

Querying a Dynamic Table



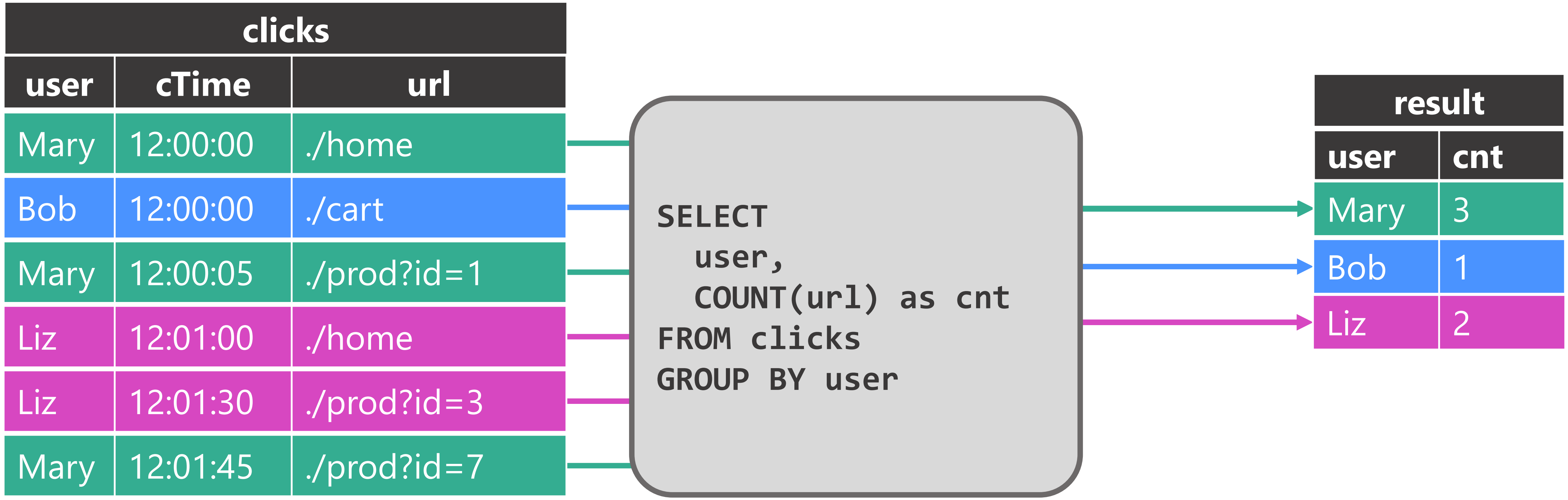
clicks		
user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:00:05	./prod?id=1
Liz	12:01:00	./home
Liz	12:01:30	./prod?id=3
Mary	12:01:45	./prod?id=7

```
SELECT user
FROM clicks
WHERE url = './home'
```

result
user
Mary
Liz

Rows of result table are appended.

Querying a Dynamic Table



clicks		
user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:00:05	./prod?id=1
Liz	12:01:00	./home
Liz	12:01:30	./prod?id=3
Mary	12:01:45	./prod?id=7

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

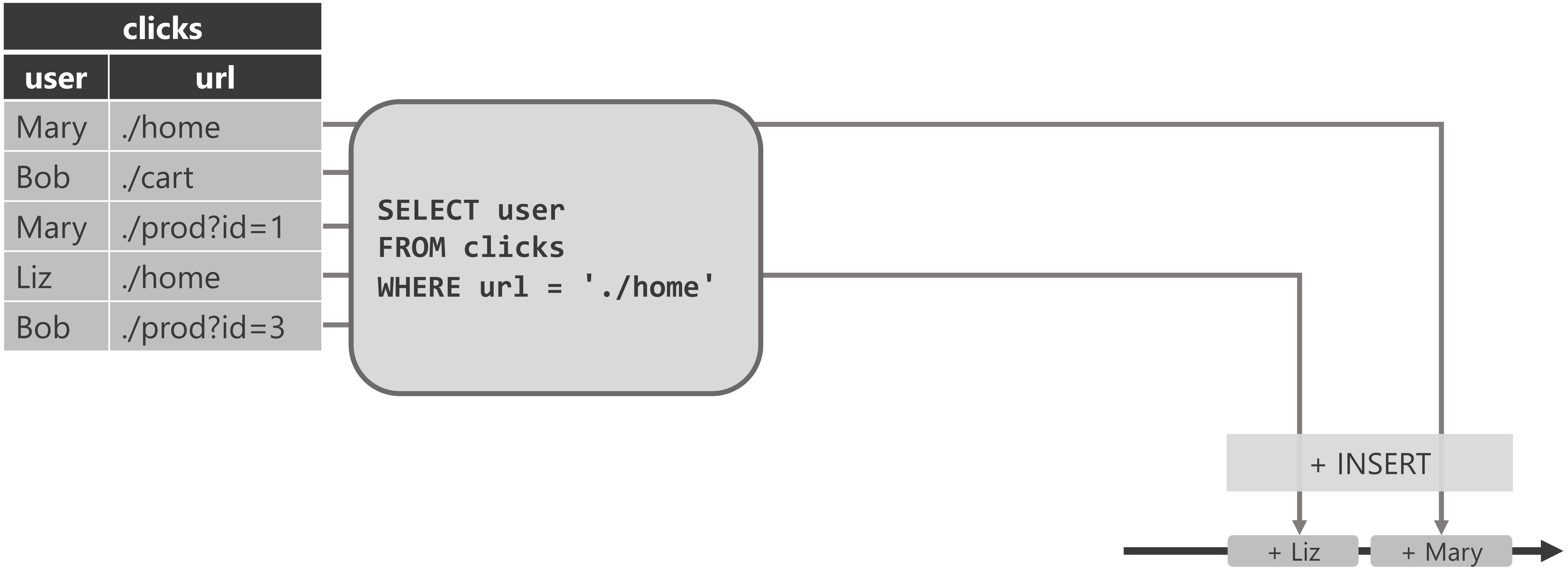
result	
user	cnt
Mary	3
Bob	1
Liz	2

Rows of result table are updated.

Dynamic Table → Stream

- Converting a dynamic table into a stream
- Dynamic tables might update or delete existing rows
- Updates must be encoded in outgoing stream

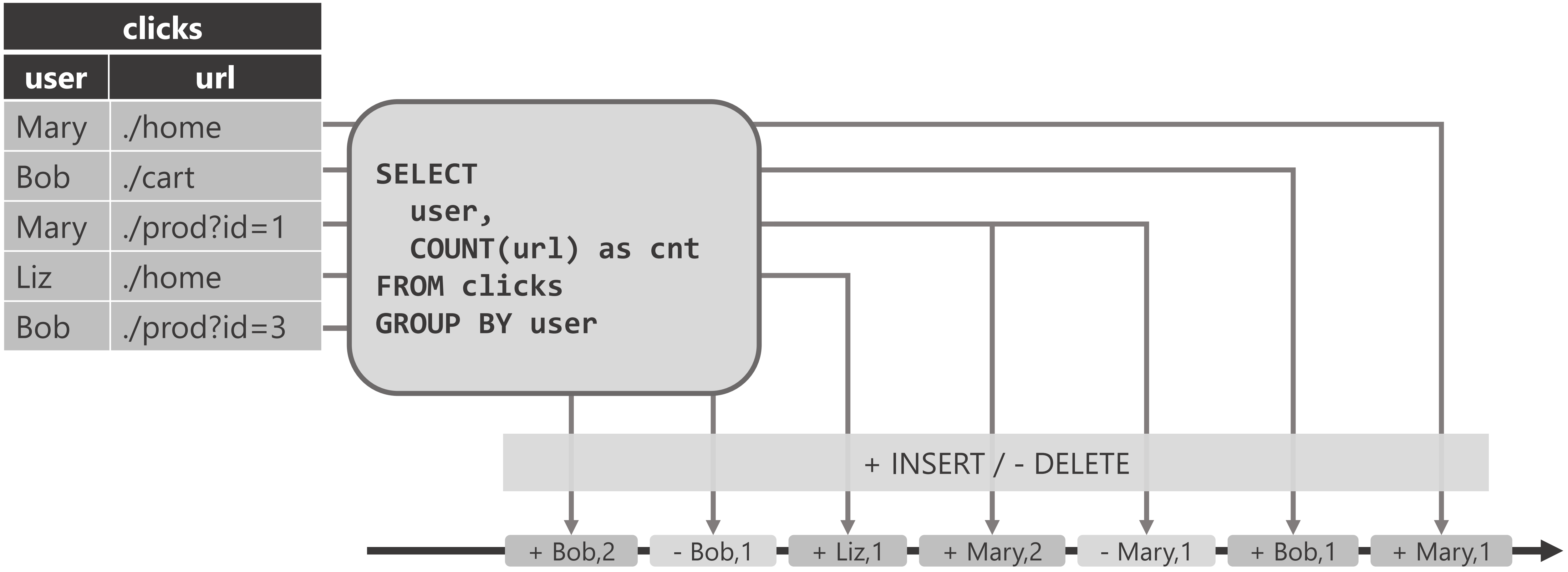
Dynamic Table → Stream: INSERT-only



clicks	
user	**url**
Mary	./home
Bob	./cart
Mary	./prod?id=1
Liz	./home
Bob	./prod?id=3

```
SELECT user
FROM clicks
WHERE url = './home'
```

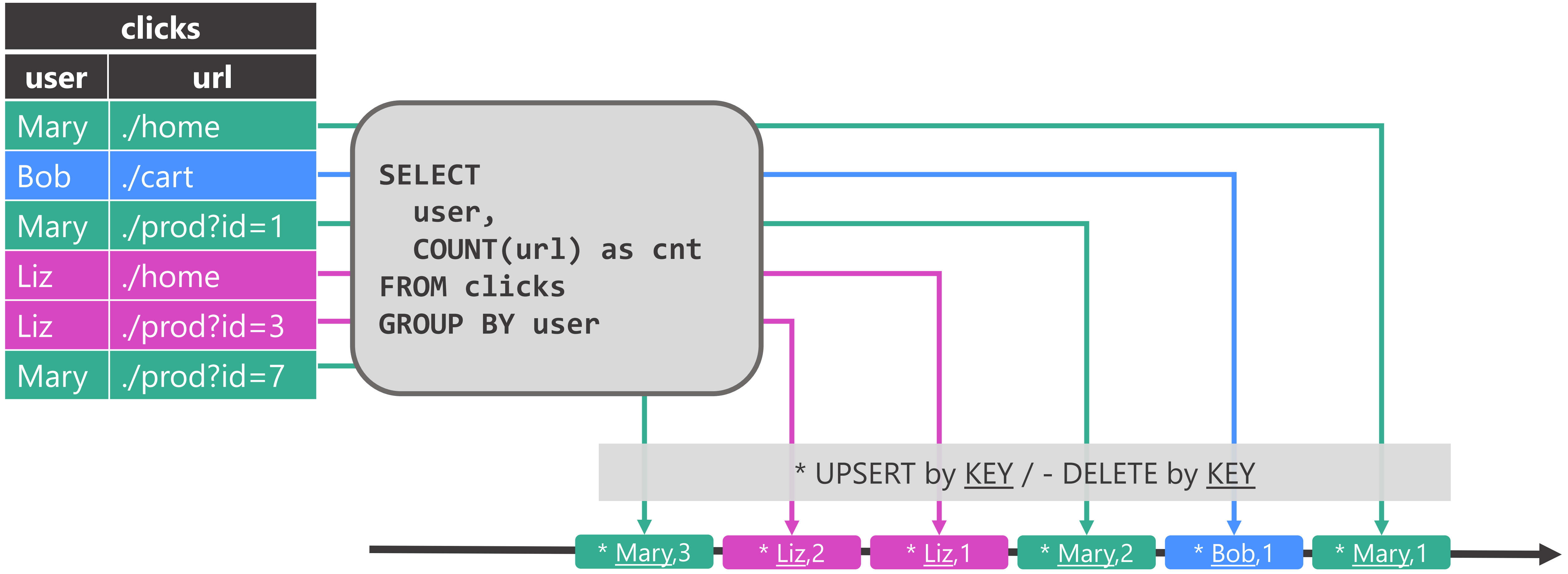
Dynamic Table → Stream: INSERT+DELETE



clicks	
user	**url**
Mary	./home
Bob	./cart
Mary	./prod?id=1
Liz	./home
Bob	./prod?id=3

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

Dynamic Table → Stream: UPSERT+DELETE



clicks	
user	**url**
Mary	./home
Bob	./cart
Mary	./prod?id=1
Liz	./home
Liz	./prod?id=3
Mary	./prod?id=7

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

Operators and State

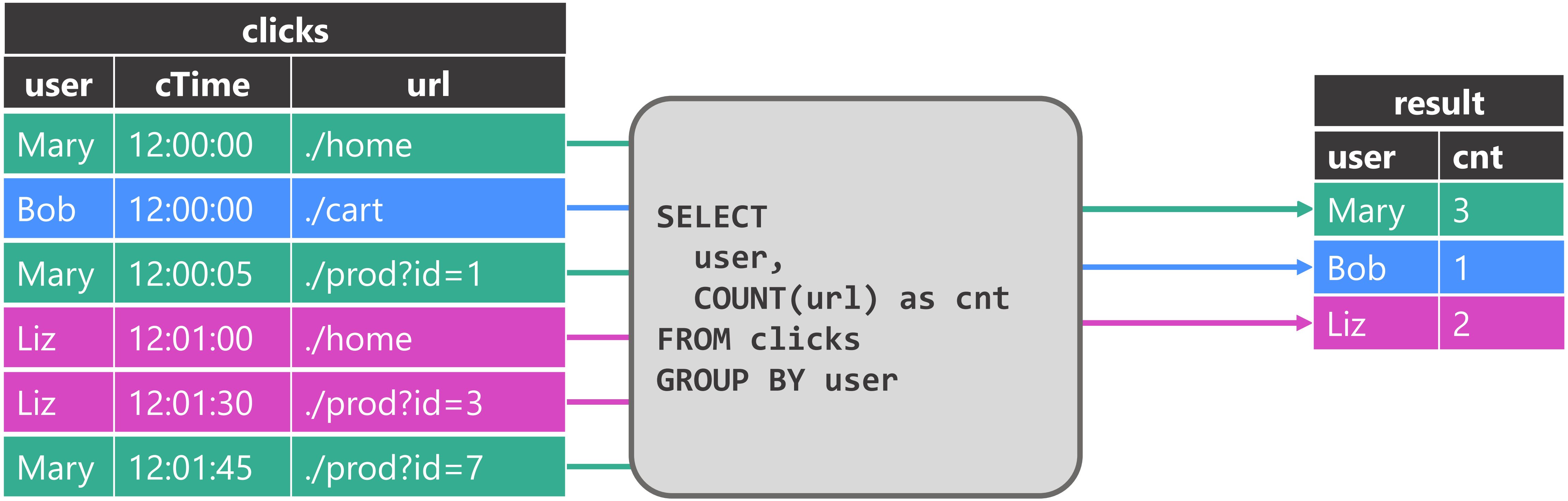
Operator Types

- Stateless operators
  - Filter
  - Projection

- Materializing operators
  - Aggregation
  - Joins

- Temporal operators (discussed in later sessions)
  - Window aggregation (GROUP BY, OVER)
  - Time-based joins (Interval join, Temporal-table join)
  - Pattern matching (MATCH_RECOGNIZE)

Materializing Aggregation



clicks

user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:00:05	./prod?id=1
Liz	12:01:00	./home
Liz	12:01:30	./prod?id=3
Mary	12:01:45	./prod?id=7

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

result

user	cnt
Mary	3
Bob	1
Liz	2

Rows of result table are updated.

Materializing Aggregation

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

- The aggregation needs to maintain a count for every user forever.
  - Every user could click at any point in time
- The aggregation state is growing with every new user
  - For some aggregation functions, state grows with every new input row

Stateful Operators

- Materializing operators
  - Computations are not bound by temporal condition and never complete
  - Input and output records can be updated or deleted
  - Hold records and/or results forever in state
  - State can grow over time (depending on query and data)
- Temporal operators
  - Associate records based on a temporal condition
  - Only accept new records. Previously added records can not be updated or deleted.
  - Hold records and/or results in state until a computation is complete
  - Automatically clean up state as soon as records and results are no longer needed

Managing State Size of Materializing Operators

- Query state might grow indefinitely
  - Depends on query and input tables
- Slowly growing state can be addressed by scaling the query
  - `SELECT user, COUNT(*) FROM logins GROUP BY user;`
- State can be automatically pruned
  - `SELECT session, COUNT(*) FROM clicks GROUP BY session;`
  - Rows and persisted results can be removed after an idle timeout

Idle State Clean Up

- Configure Flink to automatically remove state that was not accessed for x time
  - The query result is not updated when state is removed

- Query result remains consistent if removed state is not needed again

- Query result becomes inconsistent if query needs to access state that was removed!

- Trade the accuracy of the result for size of state

Summary

Summary

- Streams are interpreted as changelog for a Table
  - Flink 1.10 supports INSERT-only stream to table conversion
  - Flink 1.11 will support full changelog conversion

- SQL queries on dynamic tables yield another dynamic table
  - Input and query determines whether resulting dynamic table is append-only or updating

- Dynamic tables can be converted back into streams
  - INSERT-only, INSERT+DELETE, UPSERT+DELETE

Summary

- Use regular SQL to run queries on dynamic tables
  - No need to learn special syntax or semantics
  - Writing and executing queries is easy, BUT…
- Pay attention to the state requirements of your query
  - Depending on query and input data, the state might grow indefinitely
  - Enable idle state pruning to trade off query accuracy and state consumption

Hands On Exercises

Querying Dynamic Tables with SQL

Continue with the hands-on exercises in
“Querying Dynamic Tables with SQL”

https://github.com/ververica/sql-training/wiki/Querying-Dynamic-Tables-with-SQL

We are here to help!

www.ververica.com

@VervericaData