Global sums for currently active sessions

Status: *Draft*
Intent:

- transaction sums per customer session (simple using session-windowed aggregation)
- global transaction sums for all *currently active* customer sessions

```
builder
  .stream(/*key serde*/, /*transaction serde*/, "transaciton-topic")

  .groupByKey(/*key serde*/, /*transaction serde*/)

  .aggregate(
    () -> /*empty aggregate*/,
    aggregator(),
    merger(),
    SessionWindows.with(SESSION_TIMEOUT_MS).until(SESSION_TIMEOUT_MS*2),
    /* aggregate serde */,
    txPerCustomerSumStore() // this store can be queried for per customer session data  )

  .toStream()

  .filter(((key, value) -> value != null)) // tombstones only come when a session is merged into a bigger
session, so ignore them

 // the below map/groupByKey/reduce operations are to only propagate updates to the latest session per customer
to downstream

  .map((windowedCustomerId, agg) -> // this moves timestamp from the windowed key into the value
                                    // so that we can group by customerId only and reduce to the later value
    new KeyValue<>(
      windowedCustomerId.key(), // just customerId
      new WindowedAggsImpl( // this is just like a tuple2 but with nicely named accessors: timestamp() and
aggs()
        windowedCustomerId.window().end(),
        agg
      )
    )
  )
  .groupByKey( /*key serde*/, /*windowed aggs serde*/ ) // key is just customerId
  .reduce( // take later session value and ignore any older - downstream only cares about current sessions
    (val, agg) -> val.timestamp() > agg.timestamp() ? val : agg,
    TimeWindows.of(SESSION_TIMEOUT_MS).advanceBy(SESSION_TIMOUT_DELAY_TOLERANCE_MS),
    "latest-session-windowed"
  )

  .groupBy((windowedCustomerId, timeAndAggs) -> // calculate totals with maximum granularity, which is per-
partition
    new KeyValue<>(
      new Windowed<>(
        windowedCustomerId.key().hashCode() % PARTITION_COUNT_FOR_TOTALS,  // KIP-159 would come in handy here,
to access partition number instead
        windowedCustomerId.window() // will use this in the interactive queries to pick the oldest not-yet-
expired window
      ),
      timeAndAggs.aggs()
    ),
    new SessionKeySerde<>(Serdes.Integer()),
    /* aggregate serde */
  )

  .reduce(
    (val, agg) -> agg.add(val),
    (val, agg) -> agg.subtract(val),
    txTotalsStore() // this store can be queried to get totals per partition for all active sessions
  );

builder.globalTable(
  new SessionKeySerde<>(Serdes.Integer()),
  /* aggregate serde */,
  changelogTopicForStore(TRANSACTION_TOTALS), "totals");
// this global table puts per partition totals on every node, so that they can be easily summed for global
totals, picking the oldest not-yet-expired window
```

```
TODO: put in StreamParitioners (with KTable.through variants added in KAFKA-5045) to avoid re-partitioning where
I know it's unnecessary.
```

The idea behind the % PARTITION_COUNT_FOR_TOTALS bit is that I want to first do summation with max parallelism and minimize the work needed downstream. So I calculate a per-partition sum first to limit the updates that the totals topic will receive and the summing work done by the interactive queries on the global store.