deSQLifier (4)

Kia Rahmani
Department of Computer Science
Purdue University, USA

April 4, 2018

1 Syntax of simpSQL

The following is the formal definition of the simpSQL language based on Kartik's document, representing a simple programing language with realistic standard SQL queries.

$$tn \in \texttt{TableName} \qquad f_{id}, f_v \in \texttt{FieldName} \qquad v \in \texttt{Variable}$$

$$\odot \in \{<, \leq, =, >, \geq\} \qquad \oplus \in \{+, -, \times, /\} \qquad \otimes \in \{\wedge, \vee\}$$

$$\begin{array}{lcl}
t & ::= & (tn, \overline{f_{id}}, \overline{f_v}) \\
e_d & ::= & f \mid v \mid e_d \oplus e_d \mid \mathbb{Z} \\
\phi_d & ::= & f \odot e_d \mid f \texttt{ IN } v \mid \neg\phi_d \mid \phi_d \otimes \phi_d \\
e_c & ::= & v \mid \texttt{CHOOSE } v \mid e_c \oplus e_c \mid \texttt{NULL} \mid \mathbb{Z} \\
\phi_c & ::= & v \odot e_c \mid \neg\phi_c \mid \phi_c \otimes \phi_c \\
\phi_j & ::= & f = f \mid \phi_j \wedge \phi_j \\
c & ::= & \texttt{SELECT } \bar{f} \texttt{ AS } v \texttt{ WHERE } \phi_d \mid \texttt{SELECT}_{\phi_j}\ \bar{f} \texttt{ AS } v \texttt{ WHERE } \phi_d \mid \\
 & & \texttt{UPDATE SET } \bar{f} = \bar{e_c} \texttt{ WHERE } \phi_d \mid\ v = e_c \mid c; c \mid \\
 & & \texttt{INSERT VALUES } \bar{f} = \bar{e_c} \mid\ \texttt{DELETE } tn \texttt{ WHERE } \phi_d \mid \\
 & & \texttt{IF } \phi_c \texttt{ THEN } c \texttt{ ELSE } c \mid\ \texttt{FOREACH } v_1 \texttt{ in } v_2 \texttt{ DO } c \texttt{ END}
\end{array}$$

Figure 1: Syntax of simpSQL

2 Syntax of kvSQL

Figure 2 presents the kvSQL language which is used to write generic key-value store backed applications. The language is not very different from SQL; however it replaces tables with (denormalized) objects supporting restricted queries. This model represents the real world restrictions in EC stores such as Cassandra. We will later formally define the translation from simpSQL to kvSQL

$$t \in \texttt{TableName} \qquad f_{id}, f_v \in \texttt{FieldName} \qquad v_{val}, v_{rec} \in \texttt{Variable} \qquad txn \in \texttt{TxnName}$$

$$\odot \in \{<, \leq, =, >, \geq\} \qquad \oplus \in \{+, -, \times, /\} \qquad \otimes \in \{\wedge, \vee\}$$

$$\begin{array}{lcl}
pk & ::= & (\overline{f_{id}}, f_v) \mid (\overline{f_{id}}, f_{id}) \\
obj & ::= & (t, pk, \overline{f_v}) \\
r & ::= & \bar{e} \mid \texttt{CHOOSE}(v_{rec}) \\
e & ::= & \mathbb{Z} \mid \texttt{NULL} \mid r^i \mid v_{val} \mid e \oplus e \\
\phi_{pk} & ::= & pk^1 = \bar{e} \mid pk^2 \odot e \mid \phi_{pk} \otimes \phi_{pk} \\
op & ::= & obj.\texttt{PUT}(r) \mid v_{rec} = obj.\texttt{GET}(\phi_{obj^2}) \mid obj.\texttt{DELETE}(\phi_{obj^2}) \\
\phi_c & ::= & e \odot e \mid r \ \texttt{IN} \ v_{rec} \mid \phi_c \otimes \phi_c \mid \neg\phi_c \\
c & ::= & \{\overline{op}\}_{DC} \mid v_{val} = e \mid v_{rec} = \texttt{FILTER}(v_{rec}) \\
 & & \texttt{IF} \ \phi_c \ \texttt{THEN} \ c \ \texttt{ELSE} \ c \mid c; c \mid \{c\}_{SER} \mid \\
 & & \texttt{FOREACH} \ v \ \texttt{IN} \ v \ \texttt{DO} \ c \ \texttt{END}
\end{array}$$

Figure 2: Syntax of kvSQL

3 simpSQL to kvSQL Translation

In this section we will present the complete algorithm to translate arbitrary simpSQL programs to an equivalent kvSQL verison. As an example, we will also apply our procedure on full TPC-C benchmark and derive the kvSQL TPC-C.

3.1 Data Remodeling Rules

//TODO

3.2 Data Remodeling: TPC-C

SimpSQL Table: Warehouse

w_id	w_name	w_address	w_tax	w_ytd

kvSQL Object(s): Warehouse

id := (w_id)
warehouse_by_id := (Warehouse,(id,_),[w_name;w_address;w_tax;w_ytd])

SimpSQL Table: District

d_id	d_w_id	d_info	d_ytd	d_tax	d_next_o_id

kvSQL Object(s): District

id := (d_id,d_w_id)
d_info_by_id := (District,(id,_),[d_info])
d_ytd_by_id := (District,(id,_),[d_ytd])
d_tax_by_id := (District,(id,_),[d_tax])
d_next_o_id_by_id := (District,(id,_),[d_next_o_id])

SimpSQL Table: Customer

c_id	c_d_id	c_w_id	c_name	c_ytd	c_delivery_cnt	c_payment_cnt	c_balance

kvSQL Object(s): Customer

id := (c_id,c_d_id,c_w_id)
c_name+ytd+...._by_id := (Customer,(id,_),[c_name;c_ytd;...])
c_balance_by_id := (Customer,(id,_),[c_balance])
c_ytd+...._by_name := (Customer,(id,c_name),[c_ytd;...])
c_balance_by_name := (Customer,(id,c_name),[c_balance])

SimpSQL Table: Orders

o_id	o_d_id	o_w_id	o_c_id	o_carrier_id	o_entry_d

kvSQL Object(s): Orders

id := (o_id,o_d_id,o_w_id)

order_by_id := (Orders,(id,_),[o_c_id;o_carrier_id;o_entry_d])

o_id+entryD+CarriedID_by_o_c_id := (Orders,(id,o_c_id),[o_id;...])

SimpSQL Table: Item

i_id	i_info

kvSQL Object(s): Item

id := (i_id)

i_info_by_id := (Item,(id,_),[i_info])

SimpSQL Table: OrderLine

ol_o_id	ol_d_id	ol_w_id	ol_number	ol_info

kvSQL Object(s): OrderLine

id := (ol_o_id,ol_d_id,ol_w_id,ol_number)

ol_info_by_id := (OrderLine,(id,_),[ol_info])

ol_number+info_by_ol_o_id := (OrderLine,(id,ol_o_id),[ol_number;ol_info])

SimpSQL Table: Stock

s_i_id	s_w_id	s_quant	s_order_cnt	s_info

kvSQL Object(s): Stock

id := (s_i_id,s_w_id)

s_quant_by_id := (Stock,(id,_),[s_quant])

s_orderCnt_by_id := (Stock,(id,_),[s_order_cnt])

s_info_by_id := (Stock,(id,_),[s_info])

SimpSQL Table: OrderLine JOIN Stock

ol_o_id	ol_d_id	ol_w_id	ol_number	ol_info	s_i_id	s_w_id	s_quant

kvSQL Object(s): OrderLine JOIN Stock

id := (ol_o_id,ol_d_id,ol_w_id,ol_number)

s_quant_by_ol_o_id := (OrderLine $\bowtie$ Stock,(id,ol_o_id),[s_quant])

ol_by_s_i_id := (OrderLine $\bowtie$ Stock,(id,s_i_id),[ol_o_id,...])

SimpSQL Table: NewOrder

ol_o_id	ol_d_id	ol_w_id

kvSQL Object(s): NewOrder

id := (no_o_id,no_d_id,no_w_id)

no_by_no_d_id := (NewOrder,(id,no_d_id),[])

SimpSQL Table: History

h_id	h_info

kvSQL Object(s): History

id := (h_id)
h_info_by_id := (Item,(id,_),[h_info])

3.3 Program Rewriting Rules

//TODO

3.4 Program Rewriting: TPC-C

New Order :

```
NewOrder(wh_id,dist_id,cust_id,item_list)  :=
   SELECT $w\_tax$ AS $wx$ WHERE $w\_id = wh\_id$
   SELECT $(d\_tax, d\_next\_o\_id)$ AS $dtx$ WHERE $d\_id = dist\_id \wedge d\_w\_id = wh\_id$
   UPDATE SET $d\_next\_o\_id = dtx^2 + 1$ WHERE $d\_id = dist\_id \wedge d\_w\_id = wh\_id$
   SELECT $(c\_discount, ...)$ AS $cx$ WHERE $c\_id = cust\_id \wedge c\_d\_id = dist\_id \wedge ...$
   INSERT VALUES $(o\_id, o\_c\_id, ...) = (dtx^2, cust\_id, ...)$
   INSERT VALUES $(no\_o\_id, no\_d\_id, no\_w\_id) = (dtx^2, dist\_id, wh\_id)$
   FOREACH $i$ IN $item\_list$ DO
      SELECT $i\_info$ AS $ix$ WHERE $i\_id = i$
      SELECT $(s\_quant, s\_orderCnt, ...)$ AS $sx$ WHERE $s\_i\_id = i \wedge s\_w\_id = wh\_id$
      IF $sx^1 - i^{quant} < 10$
      THEN  $sqx = sx^1 - i^{quant} + 91$
      ELSE  $sqx = sx^1 - i^{quant}$
      UPDATE SET $(s\_orderCnt, s\_quant, ...) = (sx^2 + 1, sqx, ...)$ WHERE $s\_i\_id = i \wedge ...$
      INSERT VALUES $(..., ol\_number, ...) = (..., unique, ...)$
   END
```

Listing 1: simpSQL

```
# some non−interesting updates are eliminated
NewOrder(wh_id,dist_id,cust_id,item_list,ol_quant)  :=  {
  wx= (warehouse_by_id).GET (id=wh_id) #Retrieve warehouse by PK
  dtx= (d_tax_by_id).GET (id=dist_id) #Retrieve d_tax by PK
  #Update d_next_o_id by PK:
  dnoix= (d_next_o_id_by_id).GET (id=dist_id)
  (d_next_o_id_by_id).PUT(dnoix[d_next_o_id ↦d_next_o_id+1]);
  cx= (c_info_by_id).GET (id=(cust_id,...)) #Retrieve customer by PK
  #Enter new rows into Order and NewOrder objects (3 Objects):
  (order_by_id).PUT(...); #new row is created from known values
  (o_info_by_o_c_id).PUT(...); #structure of the new row should match the
     denormalized object
  (no_by_d_id).PUT(...);

  FOREACH item_id IN item_list DO
     ix= (item_info_by_id).GET (id=item_id)
     #Retrieve Stock information by PK (from 3 objects):
     socx= (s_orderCnt_by_id).GET (id=(item_id,...))
     sqx= (s_quant_by_id).GET (id=(item_id,...))
     six= (s_info_by_id).GET (id=(item_id,...))
     IF  (sqx - ol_quant < 10)
        (s_quant_by_id).PUT(sqx[s_quant↦(s_quant-ol_quant+91)]);
        olx= (ol_by_s_id).GET (s_id=(item_id)) #All OL using this stock
        FOREACH o_id IN ol_x DO
           (s_quant_by_ol_o_id).PUT(...,sqx[s_quant ↦(s_quant - ol_quant+91)],...);
        END;
     ELSE
        (s_quant_by_id).PUT(sqx[s_quant ↦s_quant - ol_quant]);
        olx= (ol_by_s_id).GET (s_id=(item_id)) #All OL using this stock
        FOREACH o_id IN ol_x DO #update the denormalized join object
           (s_quant_by_ol_o_id).PUT(...,sqx[s_quant ↦(s_quant - ol_quant)],...);
        END;

     #Enter a new order line (4 objects):
     (ol_info_by_id).PUT(...); #insert a new row from known values
     (ol_number + info_by_ol_o_id).PUT(...);#same values; dnrmlz'd object
     (s_quant_by_ol_o_id).PUT(...); #known values; insert in join object
     (ol_by_s_id).PUT(...); #insert in the denormalized join object
  END;

}SER
```

Listing 2: kvSQL

Payment :

```
Payment(wh_id,dist_id,cust_id,cust_name,amnt) :=
  SELECT $(w\_ytd)$ AS $wx$ WHERE $w\_id = wh\_id$
  UPDATE SET $w\_ytd = wx^1 + 1$ WHERE $w\_id = wh\_id$
  SELECT $(d\_ytd)$ AS $dx$ WHERE $d\_id = dist\_id \wedge w\_id = wh\_id$
  UPDATE SET $d\_ytd = dx^1 + 1$ WHERE $d\_id = dist\_id \wedge w\_id = wh\_id$
  IF $cust\_id = $ NULL  THEN
    SELECT $(c\_id, c\_balance, c\_ytd\_payment...)$ AS $cx1$ WHERE $c\_name = cust\_name \wedge ...$
    $cx =$ CHOOSE $cx1$
    UPDATE SET $(c\_balance, c\_ytd\_payment, ...) = (cx^2 - amnt, cx^3 + amnt, ...)$
      WHERE $c\_id = cx^1)$
  ELSE
    SELECT $(c\_balance, c\_ytd\_payment...)$ AS $cx$ WHERE $c\_id = cust\_id \wedge ...$
    UPDATE SET $(c\_balance, c\_ytd\_payment, ...) = (cx^2 - amnt, cx^3 + amnt, ...)$
      WHERE $c\_id = cust\_id)$
  INSERT VALUES $(h\_id, h\_info) = (unique, ...)$
```

Listing 3: simpSQL

```
Payment(wh_id,dist_id,cust_id,cust_name,amnt) := {
  wx= (warehouse_by_id).GET (id=wh_id) #Retrieve warehouse by PK
  (warehouse_by_id).PUT(wx[w_ytd ↦w_ytd+1]); #Update the ytd of the wrhs
  dx= (d_ytd_by_id).GET (id=dist_id) #Retrieve d_ytd by PK
  (d_ytd_by_id).PUT(dx[d_ytd ↦d_ytd+1]); #Update the ytd of the district

  # Retrive customer info (except c_balance):
  IF (cust_id = NULL) #Retrieve by id or name?
  THEN cx1= (c_info_by_name).GET (c_name=cust_name);
       cx = CHOOSE cx1 # pick the middle customer;
  ELSE cx= (c_info_by_id).GET (id=(cust_id,...)) #Retrieve customers by PK
  (c_info_by_id).PUT(cx
               [c_ytd_payment↦c_ytd_payment+amnt]
               [c_payment_cnt↦c_payment_cnt+1]);

  # Retrive and update customer's balance:
  IF (cust_id = NULL) #Retrieve by id or name?
  THEN cbx1= (c_balance_by_name).GET (c_name=cust_name);
       cbx = CHOOSE cbx1 # pick the middle customer;
       #Update both customer objects:
       (c_balance_by_id).PUT(cbx [c_balance↦c_balance-amnt]);
       (c_balance_by_name).PUT(cbx [c_balance↦c_balance-amnt]);
  ELSE cbx= (c_balance_by_id).GET (id=(cust_id,...))#Retrieve customers by PK
       #Retrieve the same customer's info
       cix= (c_info_by_id).GET (id=(cust_id,...))#Retrieve customer by PK
       # Update both objects:
       (c_balance_by_id).PUT(cbx [c_balance↦c_balance-amnt]);
```

```
            $(c\_balance\_by\_name)$.PUT((cix.name,cust_id,cbx.c_balance-amnt));
    $(h\_info\_by\_id)$.PUT(wh_id,dist_id,...);
}$_{SER}$
```

Listing 4: kvSQL

Order Status :

```
OrderStatus(cust_id,cust_name) :=
   IF $cust\_id = $ NULL THEN
     SELECT $(c\_id, c\_info, ...)$ AS $cx1$ WHERE $c\_name = cust\_name \land ...$
     $cx = $ CHOOSE $cx1$
   ELSE SELECT $(c\_id, c\_info, ...)$ AS $cx$ WHERE $c\_id = cust\_id \land ...$
     SELECT $(o\_id, ...)$ AS $ox1$ WHERE $o\_c\_id = cx^1 \land o\_d\_id = dist\_id \land ...$
     $ox = $ CHOOSE $ox1$
   SELECT $(ol\_info, ...)$ AS $olx$ WHERE $ol\_o\_id = ox^1 \land ol\_d\_id = dist\_id \land ...$
   print olx
```

Listing 5: simpSQL

```
OrderStatus(cust_id,cust_name) := {
  IF (cust_id = NULL) #Retrieve by id or name?
  THEN cx1= $(c\_info\_by\_name)$.GET (c_name=cust_name);
         cx = CHOOSE $cx1$ # pick the middle customer;
  ELSE cx= $(c\_info\_by\_id)$.GET (id=(cust_id,...)) #Retrieve customers by PK
  ox1= $(o\_info\_by\_o\_c\_id)$.GET(o_c_id=cx.id); #Retrieve orders by non-PK
  ox = CHOOSE ox1 ; # pick the largest order o_id
  olx= $(ol\_info\_by\_ol\_o\_id)$.GET(ol_o_id=ox.o_id); #Retrieve OrdLn by non-PK
  print olx
}$_{SER}$
```

Listing 6: kvSQL

Stock Level :

```
StockLevel(dist_id,wh_id,thrshld) :=
   SELECT $(d\_next\_o\_id)$ AS $dnox$ WHERE $d\_id = dist\_id \land d\_w\_id = wh\_id$
   SELECT$_{ol\_i\_id \bowtie s\_i\_id}$ $(s\_info)$ AS $sx$
      WHERE $(ol\_o\_id < dnox^1) \land (ol\_o\_id > dnox^1 - 20) \land (s\_quant < thrshld) \land ...$
   print sx
```

Listing 7: simpSQL

Stock Level :

```
StockLevel(dist_id,wh_id) := {
  #Retrieve d_next_o_id by PK:
  dnox= ($d\_next\_o\_id\_by\_id$).GET (id=(wh_id,dist_id))
  sqx1 = ($s\_quant\_by\_ol\_o\_id$).GET(ol_o_id=dnox.next_o_id)
  sqx = FILTER sqx1 #Filter by w_id and d_id and by s_quant
  print sqx
}$_{SER}$
```

Listing 8: kvSQL

Delivery :

```
Delivery(dist_id,wh_id, carr_num, curr_time) :=
   SELECT ($no\_o\_id$) AS $nox1$ WHERE $no\_d\_id = dist\_id \wedge no\_w\_id = wh\_id$
   $nox$ = CHOOSE ($nox1$)
   DELETE $NewOrder$ WHERE $(no\_o\_id = nox^1) \wedge (no\_d\_id = dist\_id) \wedge (no\_w\_id = wh\_id)$
   SELECT ($o\_c\_id$) AS $ocx$ WHERE $(o\_id = nox^1) \wedge (o\_d\_id = dist\_id) \wedge \ldots$
   UPDATE SET $(o\_carrier\_id) = (carr\_num)$ WHERE $(o\_id = nox^1) \wedge (o\_d\_id = dist\_id) \wedge \ldots$
   SELECT $ol\_amount$ AS $olx$ WHERE $ol\_o\_id = nox^1 \wedge ol\_d\_id = dist\_id \wedge \ldots$
   UPDATE SET $ol\_delivery\_d = curr\_time$ WHERE $ol\_o\_id = nox^1 \wedge ol\_d\_id = dist\_id \wedge \ldots$
   $sumx$ = CHOOSE ($olx$)
   SELECT $(c\_balance, c\_deliveryCnt)$ AS $cx$ WHERE $o\_id = ocx^1 \wedge c\_d\_id = dist\_id \wedge \ldots$
   UPDATE SET $(c\_balance, c\_deliveryCnt) = (cx^1 + sumx, cx^2 + 1)$ WHERE $o\_id = ocx^1 \wedge \ldots$
```

Listing 9: simpSQL

```
Delivery(dist_id, carr_num, curr_time) := {
  nox2= ($no\_by\_d\_id$).GET(no_d_id=dist_id); #Retrive by partial key
  nox1 = FILTER(nox2); #Filter records by W_id
  nox = CHOOSE(nox1); #Pick the record with the lowest o_id
  ($no\_by\_d\_id$).DELETE(id=(nox.o_id,...)); #Delete by PK
  ox = ($order\_by\_id$).GET(id=(nox.o_id,...)); #Retrive order by PK
  ($order\_by\_id$).PUT(ox[o_carier_id ↦carr_num]); #Update the carrier ID
  ($o\_id + \ldots\_by\_o\_c\_id$).PUT(ox'[o_carier_id ↦carr_num]); #Update the carrier ID
  # ox' only includes interesting columns from ox
  olx1=($ol\_info\_by\_ol\_o\_id$).GET(ol_o_id=nox.o_id);
  olx = FILTER olx1 ; #Filter by w_id and d_id
  s = 0;
  FOREACH olr IN olx DO
      ($ol\_info\_by\_ol\_o\_id$).PUT(olr[ol_info↦curr_time]);
      ($ol\_info\_by\_id$).PUT(olr[ol_info↦curr_time]);
      $s = s+$ olr.ol_info
  END;
  cx← (c_info_by_id).GET (id=ox.c_id) #Retrive customer by PK
  #Update c_info_by_id and c_balance_by_id:
  (c_info_by_id).PUT(cx[c_delivery_cnt ↦ c_delivery_cnt + 1]);#update deliveryCnt
  (c_info_by_id).PUT(cx[c_balance ↦ c_balance - s]); #update delivery cnt
  #Update c_info_by_name and c_balance_by_name:
  (c_info_by_name).PUT(cx'[c_delvry_cnt↦c_delvry_cnt+1]);#update deliveryCnt
  (c_info_by_name).PUT(cx'[c_balance ↦ c_balance - s]); #update delivery cnt
}SER
```

Listing 10: kvSQL

3.5 Soundness of the Translation

//TODO

4 Optimization

We create a more optimized version of the initial kvSQL program, either by incrementally strengthening the weakest isolation level (using Kartik's analysis and program patches) or by weakening the SER version (how?). //TODO

4.1 Example: Optimized TPC-C

//TODO

4.2 Soundness of the Optimizer

//TODO