

PhUSE SDE 06-May-2009



Comparison of different ways using
table lookups on huge tables

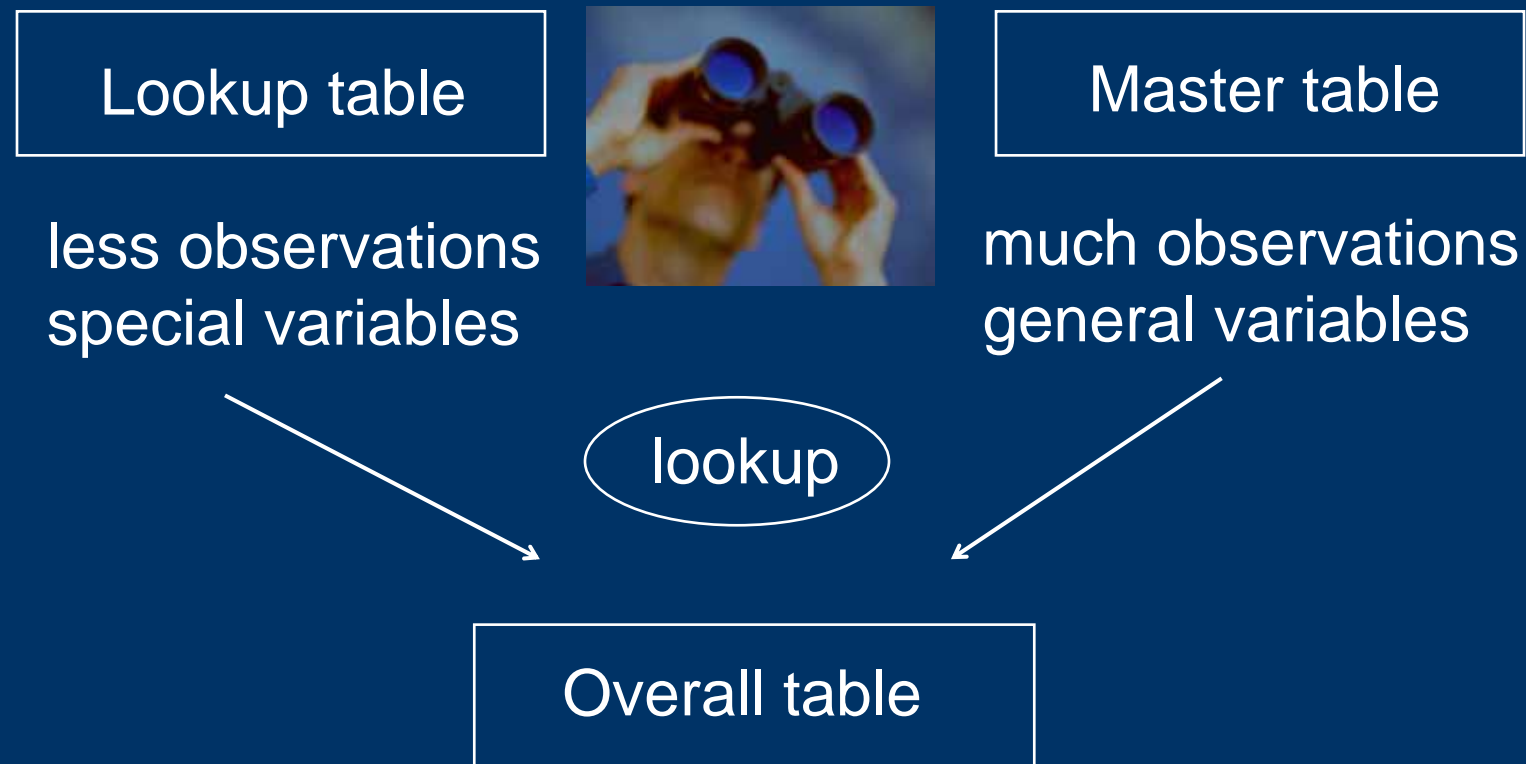
Ralf Minkenbergr

Contents

- Definition
- Classical solutions
- Advanced procedures
- Hash sort
- Comparison and summary

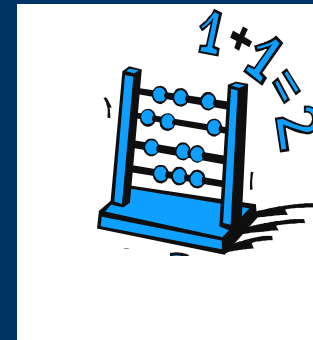


Table lookup



Classical methods

- DATA-Step with MERGE
- Creating of indexes
- PROC SQL



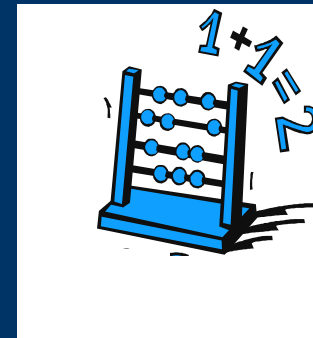
DATA step with MERGE

```
PROC SORT DATA=large; BY no; RUN;
PROC SORT DATA=small; BY no; RUN;
DATA look;
  MERGE small(IN=a) large;
  BY no;
  IF a;
RUN;
```

master table

lookup table

overall table

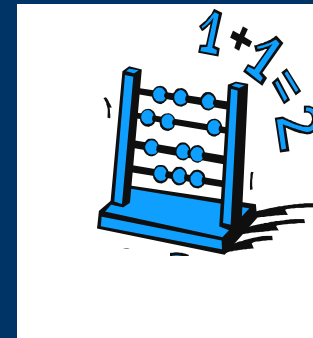


DATA step with MERGE and index

```
PROC DATASETS;  
  MODIFY large; master table  
INDEX CREATE no / UNIQUE;  
QUIT;
```

```
PROC SORT DATA=small; lookup table BY no; RUN;
```

```
DATA look; overall table  
  MERGE small(IN=a) large;  
  BY no;  
  IF a;  
RUN;
```



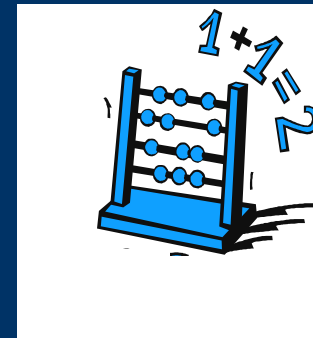
PROC SQL

```
PROC SQL;
  CREATE TABLE look AS
    SELECT large.*
    FROM small, large
    WHERE small.no = large.no;
QUIT;
```

overall table

lookup table

master table



Methods in SAS

- DATA-Step with MERGE
- Creating of indexes
- PROC SQL

classical solutions

- FORMAT procedure with PUT statement
- SET statement with KEY option
- CALL EXECUTE

advanced methods

- Hash sort → new in V9

FORMAT procedure and PUT statement

```
DATA fmtin; lookup table  
  SET small (RENAME = (no = start));  
  RETAIN label 'SMALL' fmtname 'NO';  
RUN;
```

```
PROC FORMAT CNTLIN=
```

```
RUN;
```

```
DATA look; overall table  
  SET large; master table  
  WHERE PUT (no, no.) 'SMALL';  
RUN;
```



SET statement with KEY option

```
DATA look; overall table  
SET small; lookup table  
SET large KEY=no; master table  
RUN;
```



 **Important:** table `large` must have an index `no`!

CALL EXECUTE

```
DATA _null_ ; lookup table
  SET small END=last;
  IF _n_ EQ 1 THEN DO;
    CALL EXECUTE( 'DATA look;' );
    CALL EXECUTE( 'SET large;' ); overall table
    CALL EXECUTE( 'WHERE no IN (' ); master table
  END;
  CALL EXECUTE(no);
  IF last THEN DO;
    CALL EXECUTE(');');
    CALL EXECUTE('RUN;');
  END;
RUN;
```



Hashing



(long) key value

Hash

integer out of small set of integers

perfect hash function: • very costly to determine
• to recalculate for additional values

=> non perfect hash function

Effective hashing

- Which part of all possible integers should be really **seized** with numbers after hashing ?
- If this portion of seized numbers has been set the **optimal range** has to be determined.
- The proceedings for **collisions** have to be defined.



Hashing (example)

key values

185
971
400
260
922
970
543
532
050
067

Hashing



hash values

04
10
11
01
13
09
11
13
12
03

Collisions

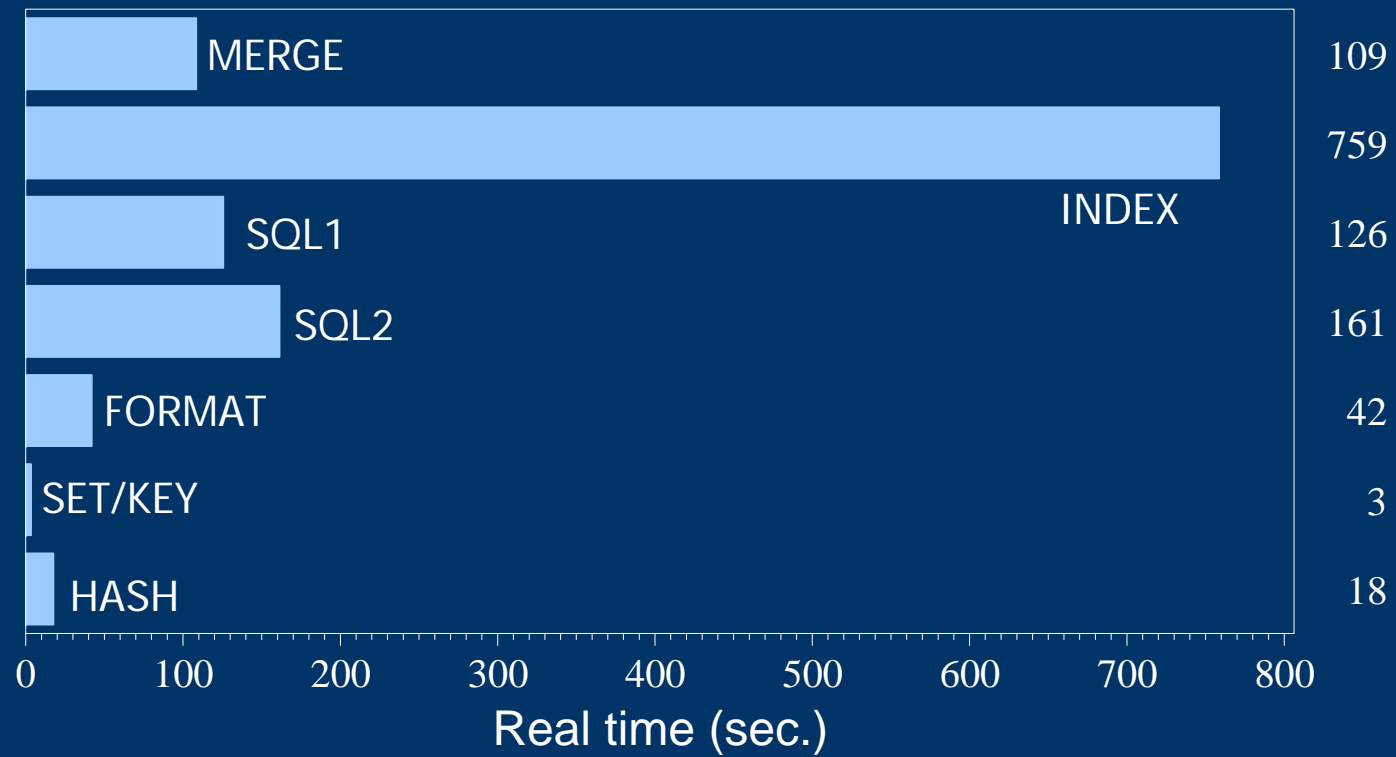


Hashing: Division by a prime (e.g. 13),
residual as result (modulo)

Hashing in SAS

```
DATA look; overall table
  SET small POINT=_n_;
  DECLARE Hash ha(dataset: 'work.small'); lookup table
  ha.DefineKey('no');
  ha.DefineDone();
  DO UNTIL (last); master table
    SET large END=last;
    IF ha.find() EQ 0 THEN OUTPUT;
  END;
  STOP;
RUN;
```

Comparison



Summary

- **Numerous** methods to perform table lookups in SAS
- **Classical** methods need (substantial) **more** time and memory
- With index in master table: **SET/KEY** method very efficient
- **Hashing** fastest method without an index