Recoding Slow Correlated Subqueries for Fast Results

Presented at New York Oracle User Group General Meeting March 10, 2005

About the Author

- B.S. Chemistry (1991)
  University of California
  San Diego.
- Focus in Mathematics of Mappings.
- Minor in Fine Art.
- Ten years experience providing enterprise DBA services.
- Managing Partner,
  Stillman Real Consulting, LLC
  http://www.netsrc.us/flash

Special Thanks

- New York Oracle User Group

  Presentation Review Panel.

- Dennis Curran & Gary Hoffman

  EZTime Project

  Emerging Health Information Technologies

- Dr. Yelena Belyaeva-Standen

  Professor of Russian Language

  Dept. of Modern and Classical Languages

  St. Louis University

And….Mom !

Recoding for Fast Results !

- **Application of Mapping Theory to PL/SQL problems.**

- **Generic program model.**

- **Snap-on program elements !**

Requirements of Coding Method

- **Mapping Theory Basics**

- **PL/SQL Basics**

Nothing more is required !

Mapping Theory is Fun…

…and EASY !

- **Gives extended and new vision into coding problems in PL/SQL.**
- **Must understand CLEARLY only the basic concepts of Map Theory.**
- **Map theory allows generic PL/SQL to be used very effectively!**

Advantage of this Coding Method

- **Easy, reliable, small set of PL/SQL program elements !**

- **Minimal knowledge of PL/SQL can yield BIG RESULTS quickly !**

Essential Mapping Theory

- **Domain**
- **Mapping**
- **Target**

Definition of a Domain

- This symbol means: **Domain**
- This symbol denotes appropriate input to the mapping.

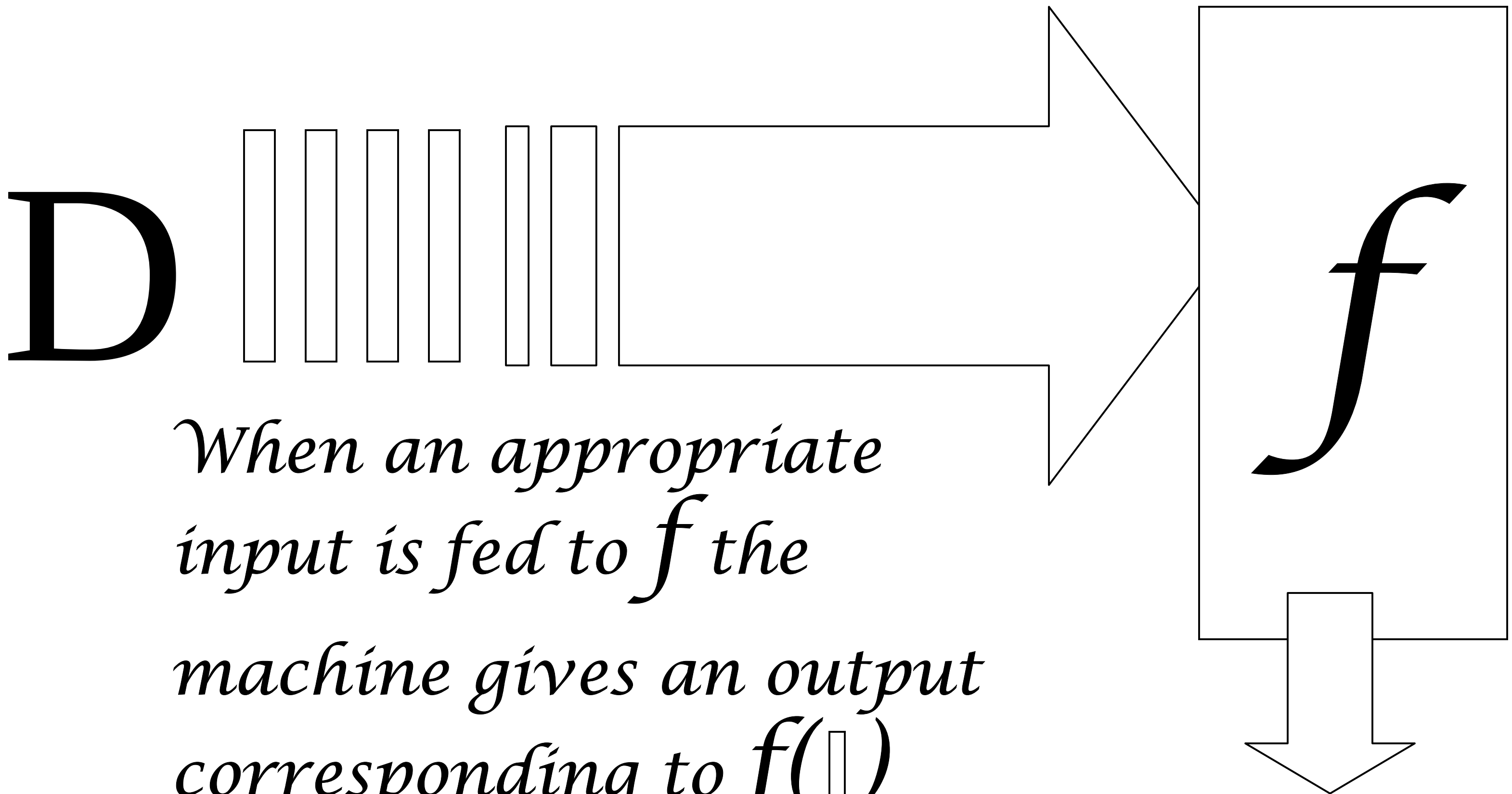
D

Definition of a Mapping

- This symbol means: **Mapping**.
- This symbol denotes the ***process*** of mapping, rather than the end-result of the mapping.
- f can be "imagined as a machine with input and output whose operation mimics the mapping process."

$$f$$

Mapping as a Machine



When an appropriate input is fed to f *the machine gives an output corresponding to* $f(\square)$

We call $f(\square)$ *the image of* $\square$ *under f* $= f(\square)$

Selecting the Domain

- *"Choosing the domain is equivalent to assigning to the elements that belong to that set a* ***coded key*** *that will cause the machine to operate."*

R.F. Wheeler

Definition of the Target

- This symbol means: **Target**.
- This symbol **T** denotes ***any*** set to which ***all*** the images of **D** under mapping f belong.

$$T$$

Mapping Theory and SQL Example 1

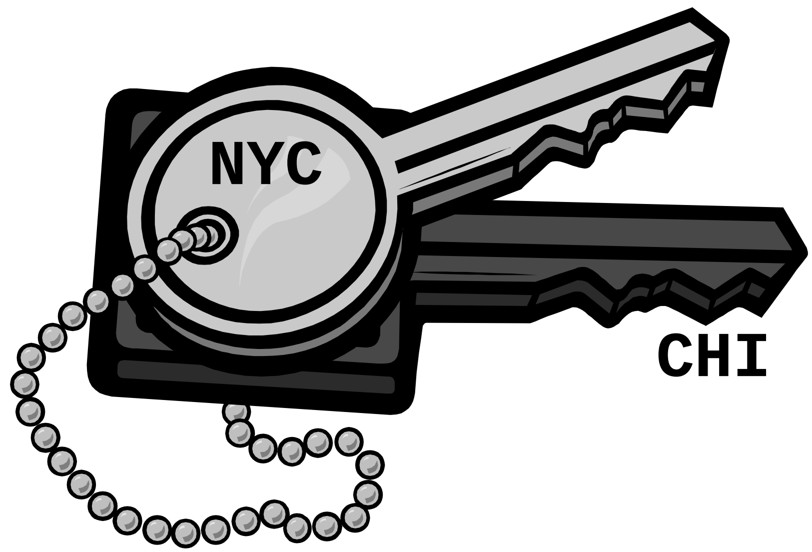
```
Select deptno, deptname, city, employees
from dept
where city = 'NYC'
```

- The select list column names hold values in them that are the domain **D**.
- Values in the select list column CITY are the "***domain keyholes***"
- Where clause prepares the mapping.
- This mapping f accepts values of column CITY.

The *Coded Keys*

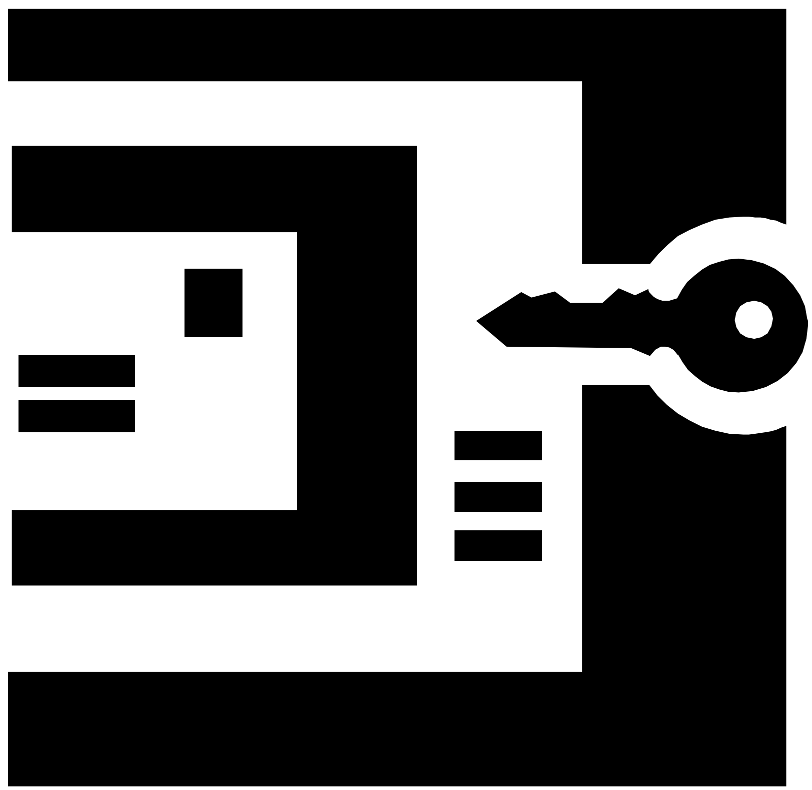
- The coded keys are a set of unique values that are used to map each row □ to its image *f*(□) .
- The set of coded keys is given by the sql:

  select distinct city from dept

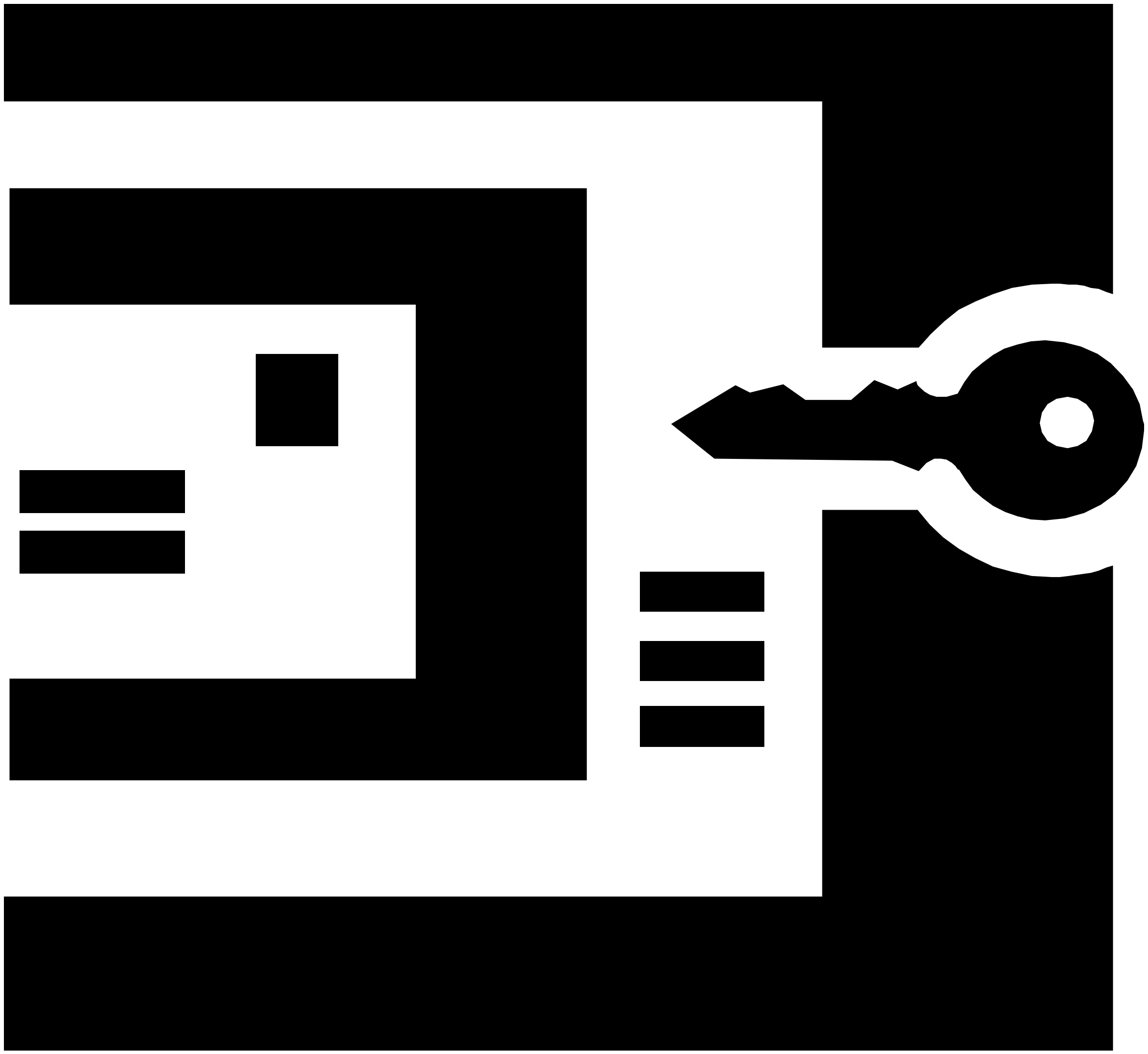


Domain Keyholes

- Each row is a “domain keyhole”.
- Each row in the dept table is tested with one of the coded keys.
- The coded key in this example is NYC.

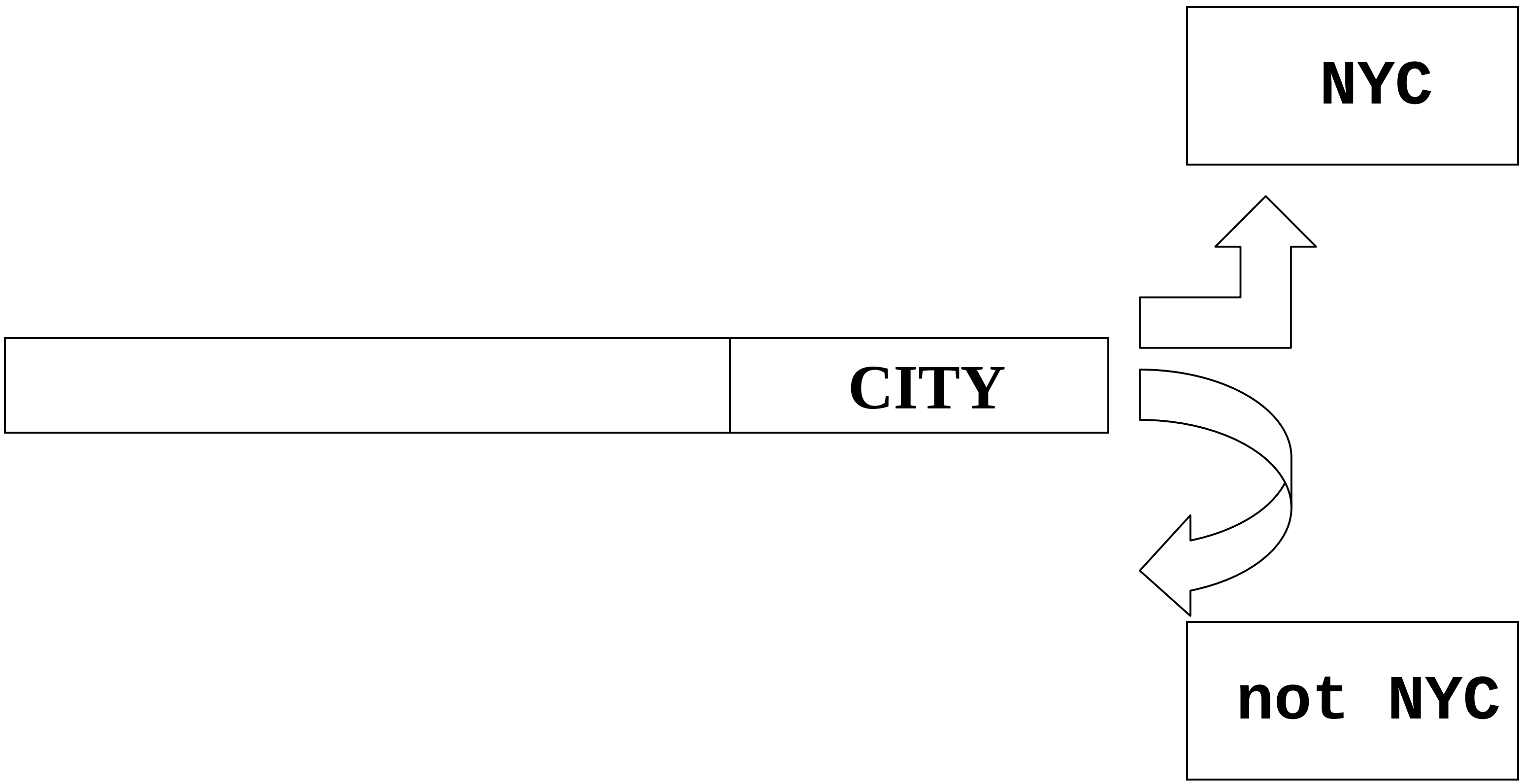


Mapping Rows by Unique Pseudokey



- **This row fits the unique pseudokey.**
- **The pseudokey gains entry to the domain keyholes.**
- **Other columns in select list "go along for ride"**

Technical Detail of Mapping



Features of this Simple Mapping

- Each row in the domain has one and only one image under the mapping f.
- The mapping f maps the rows to their unique images in the target **T.**
- The images of the rows are a 2 value set !

  $\{$ (**NYC**), (**not NYC**) $\}$

- The image of a row □ from the domain **D** under mapping f **is <u>not</u>** the select list of the query !

More on Images

- The target of the mapping is a set of two values {NYC, not NYC} or {ON, OFF}.
- The values of the select list of the cursor FOR Loop are part of the image but they are not visible to us at the time of the mapping.
- The result set {ON, OFF} and the result set (select list of the cursor FOR Loop) are ***orthogonal sets.***
- Let's talk briefly about ***orthogonal sets*** **…**

Orthogonal Sets

- We map the rows to one box that has NYC off, and one box that has NYC on.
- We cannot "see" the box so the attribute "NYC box YES" or "NYC box NO" must be ***attached*** to the row itself.
- When we look at the rows for the purpose of mapping, we only look at the END of the row, where we see only the {ON, OFF} attribute showing {NYC, not NYC}.

The Image We Want

- We generally want to know the values of the select list columns in the cursor FOR loop for rows which mapped to NYC = ON.
- We accomplish in mapping theory by rotating the row 90 degrees to reveal the select list attributes of the row.

Orthogonal Sets

CITY | emp | dept | sal

NYC | emp | dept | sal

More Features of Simple Mapping

- The select list column names of the query are a group of attributes which can accompany the domain keyhole column of CITY.
- Different select list column names are possible for the domain keyhole column of CITY.
- For a given value assigned to the keyhole column of CITY, there can be many rows keyed to that value of keyhole column CITY.
- For example, there are many rows which have keyhole column CITY value ‘NYC’.

Important to Understand

- Values of CITY are ultimately reduced by the computing system to a switch that is ON or OFF.
- This is all that computers can do:

Test if a switch is on or off.

- When values of CITY are tested:

```
ON  = NYC
OFF = not NYC
```

Reminder

- The images of the rows □ in the domain **D** under the mapping f are a set of only 2 values !

$$\{(\mathbf{NYC}),\quad (\mathbf{not\ NYC})\}$$

ON OFF

Alternate Images

- The data we want, the select list columns from the cursor FOR loop are available to us.
- We find rows that have the “NYC” set to “ON” then conceptually, we rotate the row through a right angle (90 degrees) to see the image of the row in terms of the cursor FOR Loop select list columns !

Another Way of Looking at IT

- The rows mapped by the PL/SQL program to the desired result set can be viewed from different angles, perspectives, viewpoints.
- When we map on a key, we take the row, visualize it as a long rod, and slide it into a rack with it's round end facing us. Each end is marked ON or OFF.
- Rows marked ON are removed from the rack at read time and the values of the columns along the length of the bar are read to the user.

Simple Correlated Subquery Example 2

```
SELECT P.ENAME, P.DEPTNO, P.SAL
FROM   EMP P
WHERE
SAL > (SELECT AVG(SAL)
       FROM EMP A
       WHERE P.DEPTNO = A.DEPTNO);
ORDER BY DEPTNO, SAL;
```

Mapping Theory and SQL Example

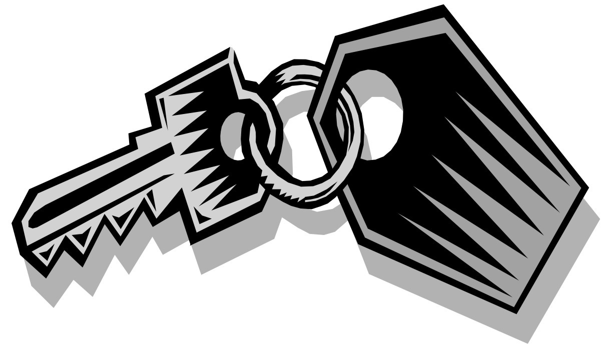
- The select list column names hold values in them that are the domain **D**.
- In this example the "***coded key***" is ***actually two keys*** of DEPTNO and AVG(SAL).
- Where clause prepares the mapping.
- Each mapping f accepts values of each part of the compound key which represent pairs of UNIQUE values **(** DEPTNO, AVG(SAL) **)**.

The *Coded Keys*

- The coded keys are a set of unique values that are used to map each row □ to its image *f*(□)

- The set of coded keys is given by the sql:

  select deptno, avg(sal)
  from dept
  group by deptno



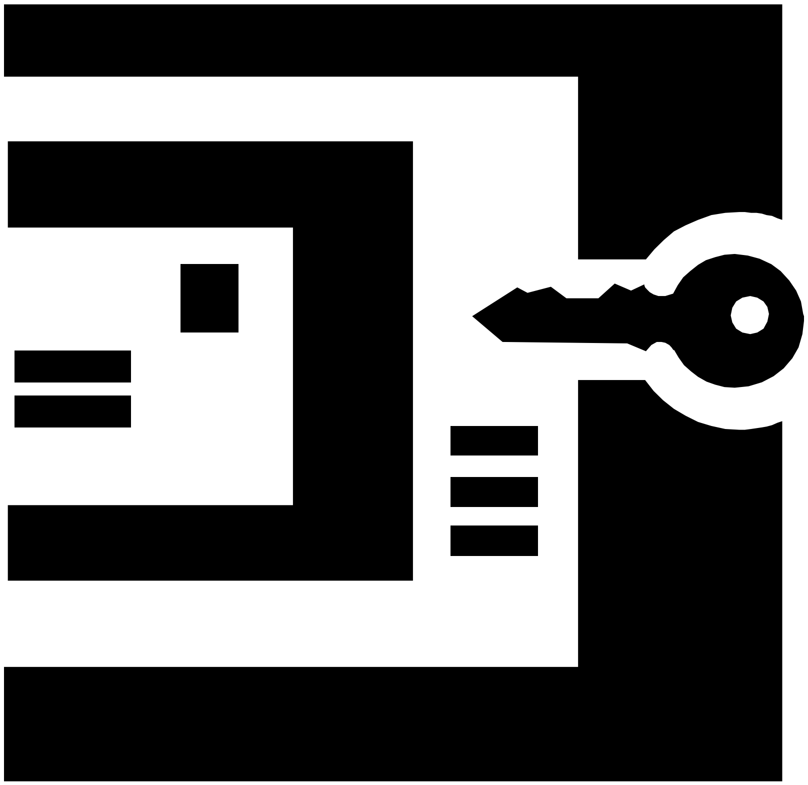
New Features of this Example

- This key graphic is made up of “two keys” because two keys are required in this case for the mapping values to be UNIQUE.

Domain Keyholes

- Each row is again a “domain keyhole”.
- Each row in the emp table is tested with each of the coded keys.
- The coded keys in this example are made from

(DEPTNO, AVG(SAL) **)**.

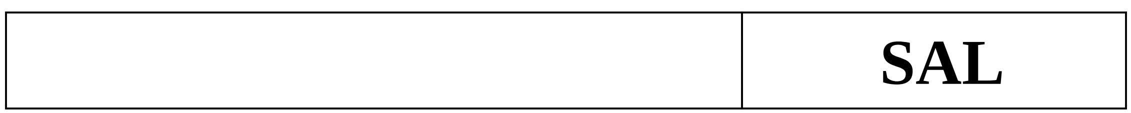


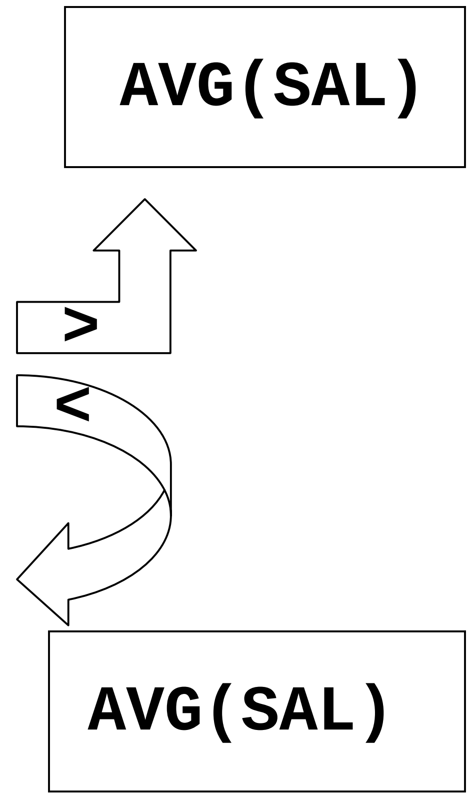
New Features of Example 2

- There is a *SET* of coded keys this time, where before there was only ONE key.
- This is because AVG(SAL) alone is not sufficient to map values of SAL to the {(ON),(OFF)} set.
- That is why this is often called a correlated subquery “mimic” of a “group by”.

Technical Detail of Mapping

Another function computes AVG(SAL) grouped by ***each distinct*** *deptno*





We want employees that have

SAL > AVG(SAL)

for ***each distinct*** *deptno.*

Features of this Simple Mapping

- Each row in the domain has one and only one image under the mapping f.
- The mapping f maps the rows to their images **T** the target**.**
- The images of the rows are a *SET* of sets !

Additional Features of Example 2

- The image of a row □ from the domain **D** under mapping f **is <u>not</u>** the select list of the query.

Reminder

- The images of the rows □ in the domain **D** under the mapping f are a SET of targets T where each target T is a set of only 2 values !

{(sales dept),(not sales dept)}

ON OFF

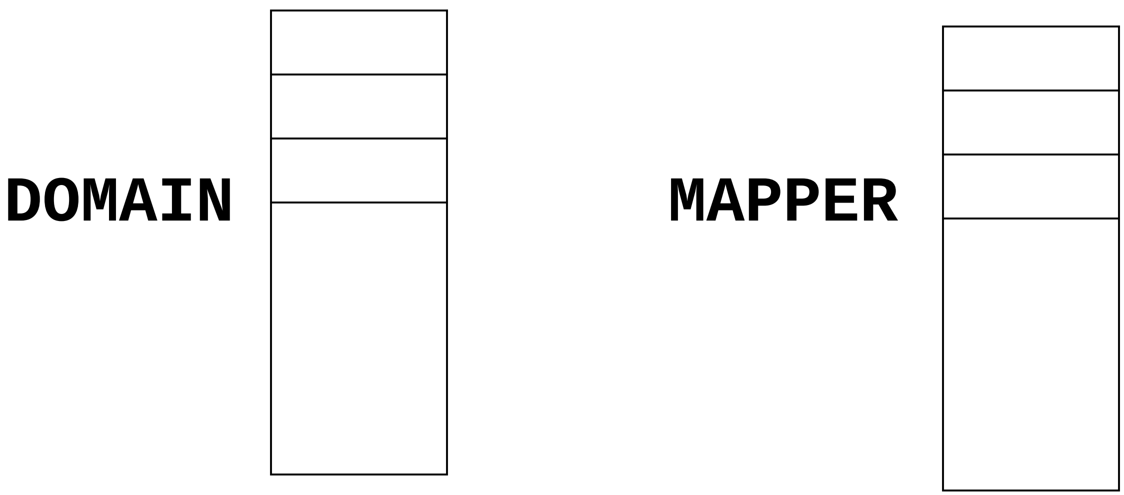
{(> AVG(SAL)sales), (< AVG(SAL)sales)}

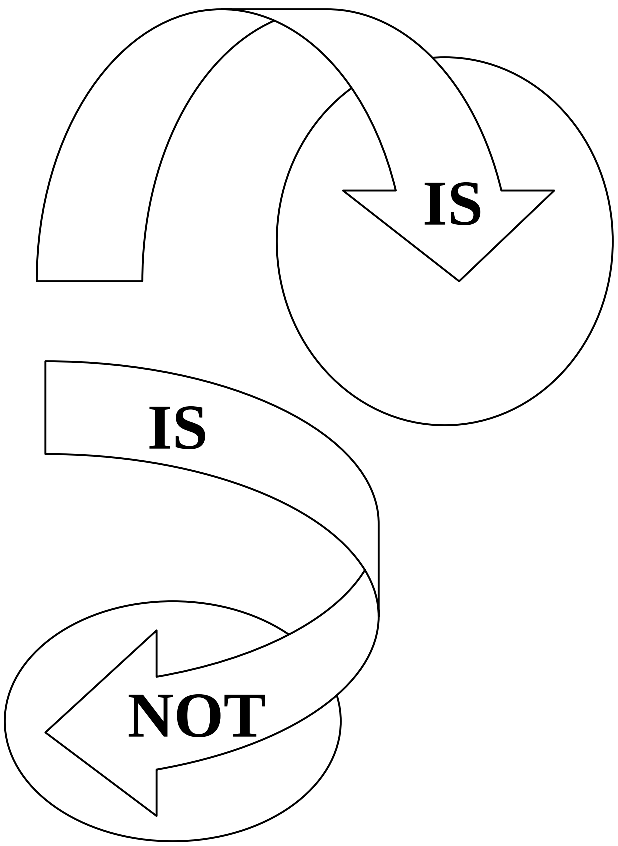
ON OFF

- For each deptno there is a target T that consists of the two value set.

Caveat

- Mapping of a row in the domain D requires one or more unique values in a "mapping set" to which domain keyhole values are compared.





- Cursor k1 is the "mapping set".

Let’s Start to Tie this Theory to PL/SQL

- PL/SQL must obey mapping theory
- If we can determine how our various program steps tie back to map theory, we can see where our mapping steps are efficient and where they are not.
- In this presentation we only work with a small set of PL/SQL constructs.
- These few constructs can do wonders !

Our PL/SQL “Snap-on” Toolset

- **Cursors**
- **Cursor FOR loop**
- **Cursor FOR loop WHERE clauses**

Note to PL/SQL Expert Coders

- More advanced PL/SQL coders can use additional, more sophisticated elements of PL/SQL, subject to the following requirements:

- MUST clearly understand which map theory element corresponds to your add-on tool (domain, mapping, target).

- Add-on tools correspond to one and ONLY one of the three map theory concepts.

Names of our “Snap-on” Tools

- **Cursors: k1, c1**
- **Cursor FOR loop**
- **Cursor FOR loop WHERE clauses**

Each snap-on tool corresponds one and only one element of function theory…

Snap-On Tool:
Cursor k1

- **Specifies the mapping** (f)

- **Nickname: “Unique Pseudokey”**

Snap-on Tool:
Cursor c1

- **Part of the specification of the Domain**

- **Nickname: "Domain Builder"**

Snap-on Tool
WHERE clause(s) of cursor FOR loop

- **Part of the mapping** (f)
- **Tests each domain lock with unique pseudokey(s)**

- **Nickname: "Image Preprocessor"**

The Mapping Theory

- In the Snap-On tool slides we saw the Domain, Mapping, and Image.

- Those are the only fundamental mapping concepts that are needed.

- Domain elements are mapped to their images by the mapping.

Mapping Theory

- **The domain is specified first.**
- **The mapping maps domain elements to their images in the target.**
- **Each domain element has ONE AND ONLY ONE image in the target.**
- **We can represent the process graphically.**

Creating a table for the target T

- **Output images (rows) reside in a special result table.**
- **We call "QRS_TABLE_NAME" the "Query Result Set" which is a table created to hold our image results of the mapping.**

The Cursor FOR Loop WHERE clauses :

- **Deliver rows for cursor k1 which meet all other criteria and are ready to be mapped by the Unique Pseudokey.**

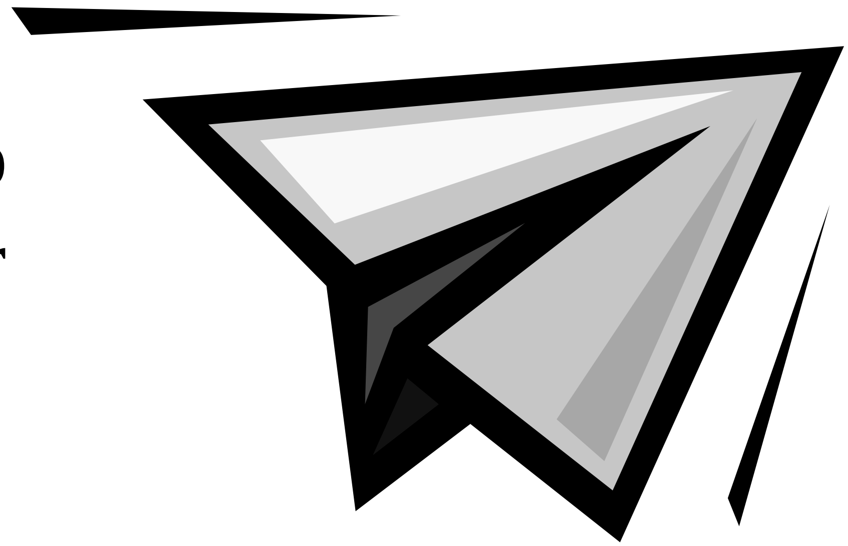
However, it is very worthwhile to think about the Cursor FOR Loop WHERE clauses more…

Pondering the Cursor FOR Loop WHERE Clause Conceptually

- Let’s think some more about the Cursor FOR Loop WHERE Clause
- It’s a hot spot and there’s action there !

Keeping Harry Happy

- We can think about someone named Harry who likes to toss paper airplanes through the air and see how well they fly…



Let’s Look at Cursor c1 “Domain Builder”

- Pieces of paper are required to make paper airplanes.
- Harry can fold airplanes much faster if he has entire sheets of rectangular paper, not just odd-shapes.
- If the sheets are in shreds and he has to tape them together to make a rectangle it slows him down A LOT
- We use cursor(s) c1 to get square pieces of paper ready for Harry. We use temp tables because Harry needs real sheets of paper ready at hand, not “virtual” sheets of paper

Tip

- Use cursor c1 to perform a complex join ONCE and then store the results to a TEMP table for processing in the cursor FOR loop.

Let’s Look at Cursor FOR Loop Where Clauses

- Once the pieces of paper have been prepared by cursor c1, Harry has to fold them into airplanes with his hands.
- Harry’s hands are his Cursor FOR Loop WHERE clause(s).
- The cursor FOR Loop WHERE clause(s) select and fold up into airplanes the pieces of paper that Harry will waft with his right hand.

Cursor FOR Loop Where Clauses Continued…

- In PL/SQL the Cursor FOR Loop WHERE clause(s) identify all rows that meet the mapping criteria.

- All rows which meet the Cursor FOR Loop WHERE clause criteria MUST be mapped by the Unique Pseudokey to ONE and ONLY one image in the target (the paper airplane cannot land “in two places at once”).

Let’s Look at Cursor k1 “Unique Pseudokey”

- Once Harry’s hands have folded the paper, Harry is ready to toss them to the target.
- Harry takes each folded plane and wafts it toward the target.
- The unique pseudokey determines where that plane will land.   The unique pseudokeys in Harry’s case are random factors: air currents, lift and drag coefficients, Harry’s toss motion, airplane folding …

"Unique Pseudokey" continued…

- In PL/SQL of course, the factors that determine where the rows that meet the criteria of the Cursor FOR Loop WHERE clauses will map (where the airplanes will land) are not random.
- In PL/SQL Harry's life is far more boring. The elements map according to a set of rules and numbers. Harry's job is more like that of an accountant, and less like flying airplanes…
- Harry has to compare a bunch of numbers and sift through the rows to figure out where they go; where they MAP to !

Understanding Cursor k1 Better

- The “Unique Pseudokey” is that value or combination of values that determine where a row is going to land in the target.

Get the Picture ?

- That is why the Unique Pseudokey is the specification of the mapping.
- The Cursor FOR Loop WHERE clauses select the rows…
- …And the Unique Pseudokey maps them to the target !

Remember

- Any PL/SQL snap-on tool that you want to use should play a clear role in the Theory.
- First figure out if it’s part of domain preparation, or if it’s part of mapping.
- Think of Harry and what makes him happiest and you will have *faster running* code than you had before !!

Big Results can be Achieved !

Harry helped me to see how to recode a batch job that took

34 HOURS

into a PL/SQL process that completed in

12 SECONDS !

Harry's Intuitive Coding (continued…)

- When a correlated subquery has a complicated join in the outer query, a good starting point is to use cursor c1 to do that join and then insert those rows into a temporary table.
- The temporary table is initialized inside the PL/SQL stored procedure and is accessed in the cursor FOR Loop.

Ways We Think About Coding

- Do you ever feel like you're in a maze when you are trying to recode ?
- People talk a lot about things like "parses", "bind variables", "wait times" and other rather technical outlooks on coding.
- These are great, very valid ideas that do work, but these ideas are not very intuitive.
- Not very accessible for DBA's like me who don't code all day long for a living, only when called upon now and then to code !

Tips

- Build cursor k1, the Unique Pseudokey, from the column(s) of the outer query that are used as the correlated select criteria in the inner correlated subquery.
- When building a Unique Pseudokey for a correlated subquery that acts as a “group by” you must include the group by column as part of the specification of the Unique Pseudokey.

Coding a “Unique Pseudokey”

- Create or replace procedure procname is
  cursor k1 is select unique pseudokey from…
- cursor c1 is select domain columns from…

The Answers to these Questions…

- Is something that you must figure out. The required things that you must determine in your recoding effort are the following:
- The function “where clauses”
- The function “unique pseudokey”
- The domain specification
- The result set

Specifying the Mapping

- **The mapping is the the unique pseudokey (cursor k1) !**

Additional Domain Specification

- **CURSOR c1 is used build domain preprocessing tables (if needed) to simplify the work of the cursor FOR Loop.**
- **CURSOR c1 is OPTIONAL**
- **CURSOR c1 can be very powerful when you have a domain built from views that are themselves complex joins of multiples tables !**

Elementary Recoding Example

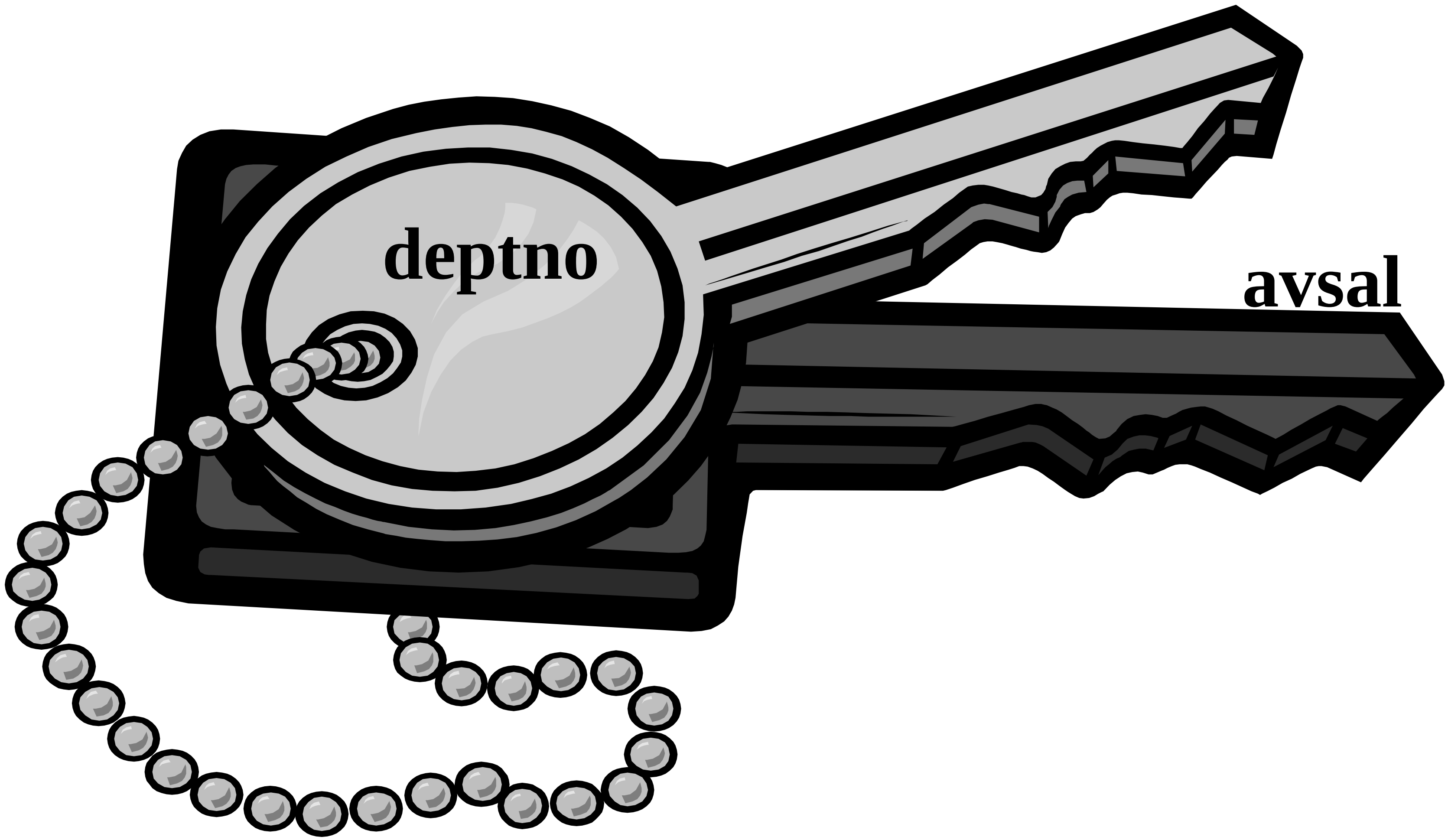
```
SELECT P.ENAME, P.DEPTNO, P.SAL
FROM   EMP P
WHERE
SAL > (SELECT AVG(SAL)
       FROM EMP A
       WHERE P.DEPTNO = A.DEPTNO);
ORDER BY DEPTNO, SAL;
```

Identify the PL/SQL Specs…

- **We have to translate this original code into the PL/SQL block**
- **The domain is …**
- **The function is…**
- **The images are…**
- **Sometimes it's not easy to see at first, but it's worth the effort !**

The Unique Pseudokey for the AVG(SAL) Problem

This is cursor k1. It is part of the specification of the function!



Images are Often Easiest to ID…

- The orthogonal images are rows of (ename, deptno, sal)
- The domain is clearly EMP
- The mapping is all rows where

  SAL > AVG(SAL) for that department.

Now build the PL/SQL !

The Great Race !

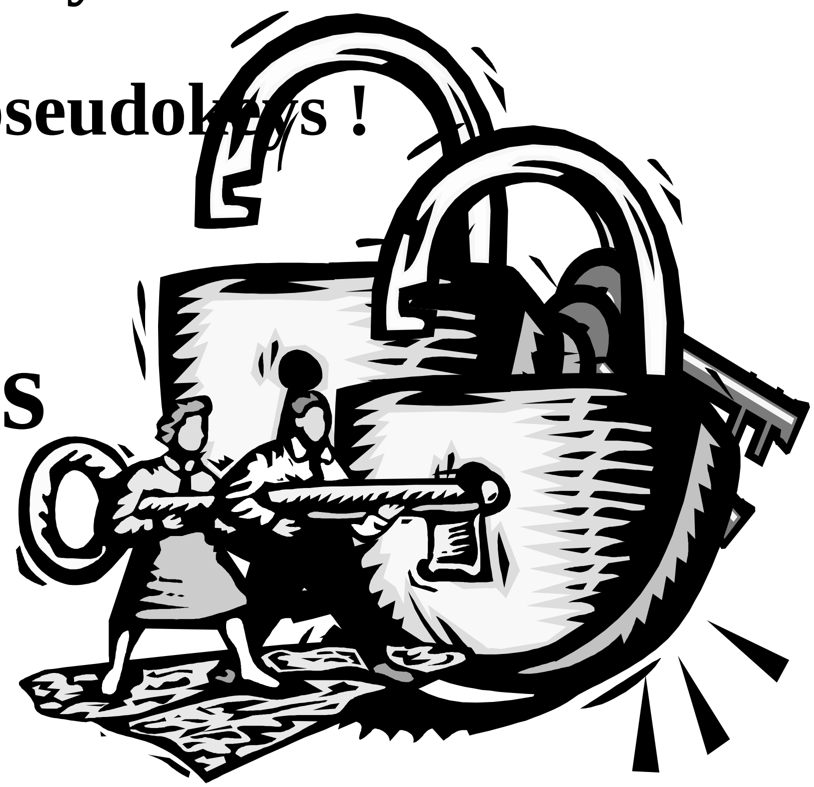
Did you see the episode of “The Great Race” where they had a key and they had to find the lock on the long bar to which hundreds of locks had been attached in which the key would fit and open the lock ? Our PL/SQL construct does just this…

Reliable Coding Tips

- Do not nest a cursor for loop inside of any domain pre-processing cursors.
- Stick with the basic PL/SQL structure for reliable results when using the methods described here.
- CLOSE all domain preprocessing cursors BEFORE opening the Cursor FOR Loop !
- Advanced coders will probably see ways to create domain preprocessing cursors that “feed” planes to Harry rather than fill a box.

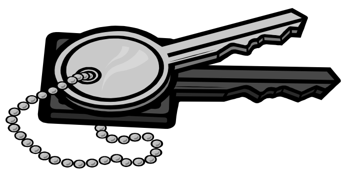
- **Which rows match DEPTNO ?**
- **Which rows also have SAL > AVG(SAL) ?**
- **Cursor c1 rows: the “keyholes”!**
- **Cursor k1 rows: the pseudokeys !**

Cursor k1 FOR loop finds where keys fit keyholes !

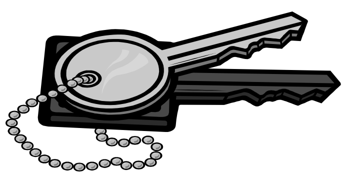


Generic PL/SQL : Section 1

```
CREATE OR REPLACE PROCEDURE
            SCHEMA.PROCNAME IS
```



```
            CURSOR k1 is ...;
                CURSOR c1 is ...;
  DOMAIN_REC1  c1%ROWTYPE;
```



```
            KEY_REk1  k1%ROWTYPE;
                  QRS_REC QRS%ROWTYPE;
```

Generic PL/SQL : Section 2

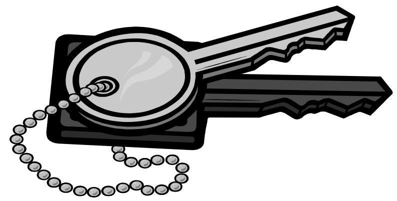
```
BEGIN
initialize domain preprocess table
EXECUTE IMMEDIATE ‘truncate…’;
initialize target table
EXECUTE IMMEDIATE ‘truncate…’;
```

TIP: Use EXECUTE IMMEDIATE *not* DELETE FROM !

Generic PL/SQL : Section 3

```
OPEN c1;
   LOOP
   FETCH c1 INTO REC1;
   EXIT WHEN c1%NOTFOUND;
     populate preprocess
     domain table
   COMMIT;
   END LOOP;
   CLOSE c1;
```

Generic PL/SQL : Section 4



```
OPEN k1;
 LOOP
 FETCH k1 into REk1;
 EXIT WHEN k1%NOTFOUND;
 FOR QRS_REC IN
 map domain rows
        using f(where, cursor k1)...
```

Generic PL/SQL : Section 5

```
LOOP
         insert mapped values into
                     target table
COMMIT;
   END LOOP;
   END LOOP;
   CLOSE k1;
END;
```

Types of Unique Pseudokey

- There is the “key value” such as a column that is already unique, such as “employee number” or “empno”

- There is the compound type unique pseudokey, which sorts values of the domain into target sets, typically values “above” a certain value and values “below” a certain value.

Unique Pseudokey Types

- A sorting unique pseudokey (typically a compound key) usually acts as a FILTER or SORTER (think of the AVG(SAL) example…we needed DEPTNO too to make the key useful).

- A primary unique pseudokey (such as empno, a “natural” unique key because it is intrinsically unique with no further specification) usually acts as a unique IDENTIFIER or POINTER. It FOCUSES the function on the specific domain element that we wish to image under the function.

Another correlated subquery recoding challenge…

```
SELECT O.PARTNUM, SUM(O.QUANTITY*P.PRICE),
COUNT(PARTNUM)

FROM ORDERS O, PART P

WHERE P.PARTNUM = O.PARTNUM

GROUP BY O.PARTNUM

HAVING SUM(O.QUANTITY*P.PRICE) >

(SELECT AVG(O1.QUANTITY*P1.PRICE)

FROM PART P1, ORDERS O1

WHERE P1.PARTNUM = O1.PARTNUM

AND P1.PARTNUM = O.PARTNUM)
```

Partnum is a natural unique pseudokey. It's a pointer type pseudokey, not a filter…

We've seen our correlated subqueries in the select list of the outer query, and also in the where clause of the outer query, but now we have a correlated subquery in the "having" clause of the group by.

This example is interesting because the having clause is a SORTER type filter while the partnum as mentioned above is a POINTER type unique pseudokey. What do we get when we recast this "east meets west" type situation in our PL/SQL construct ?

Let's see…

Example: Correlated Subquery Mimics a “Group By”

```
SELECT   O.PARTNUM,
         SUM(O.QUANTITY*P.PRICE) as "SUM_ORDERS",
         COUNT(O.PARTNUM)
FROM     ORDERS O, PART P
WHERE    P.PARTNUM = O.PARTNUM
GROUP BY O.PARTNUM
HAVING   SUM(O.QUANTITY*P.PRICE) >
              (SELECT AVG(OO.QUANTITY*PP.PRICE)
               FROM  PART PP, ORDERS OO
               WHERE PP.PARTNUM = OO.PARTNUM
               AND   PP.PARTNUM = O.PARTNUM);
```

Rewrite in PL/SQL as follows…

Create Supporting Tables

```
DROP TABLE schema.temp_table_1B;
CREATE TABLE schema.temp_table_1B
(PARTNUM     NUMBER(20),
 SUM_ORDERS  NUMBER(20),
 NUM_ORDERS  NUMBER(20))
tablespace TOOLS;
```

```
DROP TABLE schema.QRS_ORD_RECS;
CREATE TABLE schema.QRS_ORD_RECS
(PARTNUM     NUMBER(20),
 SUM_ORDERS  NUMBER(20),
 NUM_ORDERS  NUMBER(20))
tablespace TOOLS;
```

- Create the tables to do the domain pre-processing and
- Create the table to hold the images under the mapping.

Cursor k1
“compound unique pseudokey”

```
CREATE OR REPLACE PROCEDURE
  schema.OPS_QRS_WTT IS
   CURSOR k1 IS
       SELECT   o.partnum,
  AVG(o.quantity*p.price) as "AVG_ORDERS"
       FROM     ORDERS o, PART p
       WHERE    o.partnum = p.partnum
       GROUP by o.partnum;
```

Cursor c1
Domain Pre-processing

```
CURSOR c1 IS
      SELECT   O.PARTNUM,
 SUM(O.QUANTITY*P.PRICE) as "SUM_ORDERS",
 COUNT(O.PARTNUM) as "NUM_ORDERS"
      FROM     ORDERS O, PART P
      WHERE    P.PARTNUM = O.PARTNUM
      GROUP BY O.PARTNUM;
```

Cursor c1 pre-calculates the sum() and count() so that the domain is pre-processed…

Prepare the Pipeline…

```
ORD_REC1 c1%ROWTYPE;
ORD_REk1 k1%ROWTYPE;
QRS_REC schema.QRS_ORD_RECS%ROWTYPE;
BEGIN
   EXECUTE IMMEDIATE 'truncate table
  schema.temp_table_1B';
   EXECUTE IMMEDIATE 'truncate table
  schema.qrs_ord_recs';
```

The new thing to realize about these old friends is that they will transmit domain elements to their images under the mapping…

Cursor c1…
“Domain Builder”

```
Open c1;
LOOP
FETCH c1 INTO ORD_REC1;
EXIT WHEN c1%NOTFOUND;
insert into schema.temp_table_1B
values
 (ord_rec1.PARTNUM,
  ord_rec1.SUM_ORDERS,
  ord_rec1.NUM_ORDERS);
commit;
END LOOP;
   close c1;
```

Cursor k1
“Mapper” aka “Unique Pseudokey” aka “Function”

```
open k1;
  LOOP
      FETCH k1 into ORD_REk1;
      EXIT WHEN k1%NOTFOUND;
      FOR QRS_REC IN
     (select PARTNUM, SUM_ORDERS, NUM_ORDERS
      from schema.temp_table_1B
      where  SUM_ORDERS > ORD_REk1.AVG_ORDERS
      and        PARTNUM = ORD_REk1.PARTNUM)
  LOOP
  insert into schema.QRS_ORD_RECS
  values (QRS_REC.PARTNUM,
          QRS_REC.SUM_ORDERS,  QRS_REC.NUM_ORDERS);
  COMMIT;
  END LOOP;
```

The Basic Strategy

- **Simplify any complex domains by preprocessing of domain and generous use of temporary tables to hold intermediate results.**
- **Use the same programming construct over and over.**
- **Performance gains are dramatic and often as number of joins in domains are reduced by use of intermediate processing tables**

References

- Rethinking Mathematical Concepts, R. F. Wheeler Ellis Horwood, John Wiley & Sons, US Publishers, 1981.

Probably the best book of which I am aware that clearly explains mapping theory on a level that is both theoretically sound and practically useful.