Indexes

Who needs them anyway?

Access Methods

- Anticipating patterns of access
  - Key lookup
  - Pre-sorted keys
- Query performance vs DML performance
- Brings to fore the physicality of the data
- A cheat

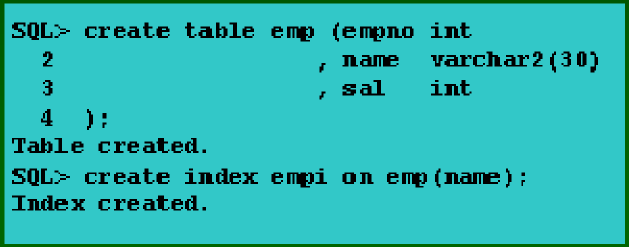
Access Methods

- B*Tree Indexes
- Hashed Clusters
- Bitmap Indexes

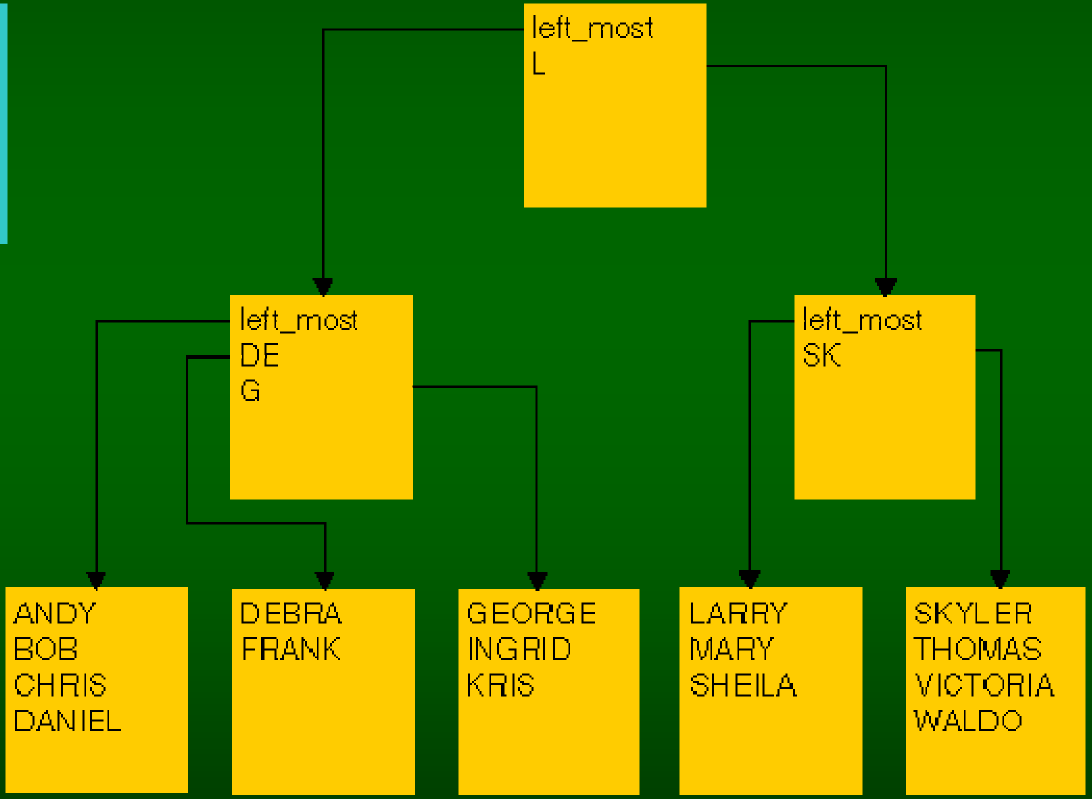
B*Tree Indexes

- Conventional
- Compound Key
- Descending Indexes
- Reverse Key Indexes
- Index Organized Tables

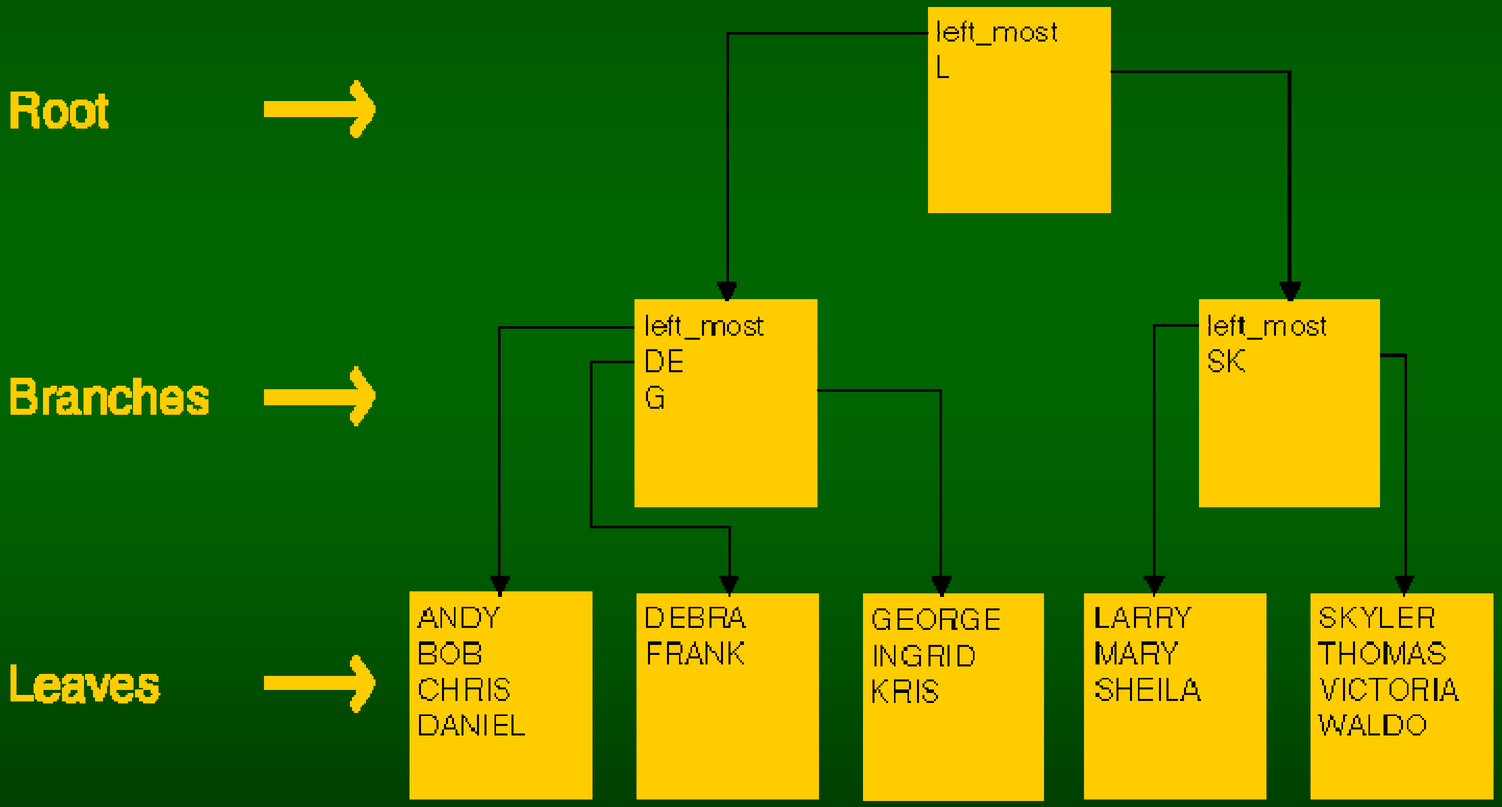
Standard B*Tree



```
SQL> create table emp (empno int
  2                    , name   varchar2(30)
  3                    , sal    int
  4  );
Table created.
SQL> create index empi on emp(name);
Index created.
```



Root, Branch and Leaves



Root

left_most
L

Branches

left_most
DE
G

left_most
SK

Leaves

ANDY
BOB
CHRIS
DANIEL

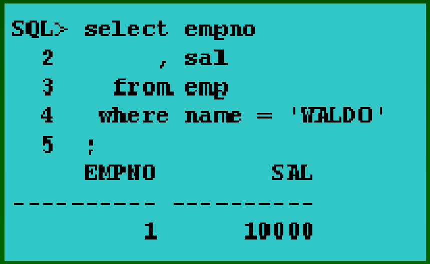
DEBRA
FRANK

GEORGE
INGRID
KRIS

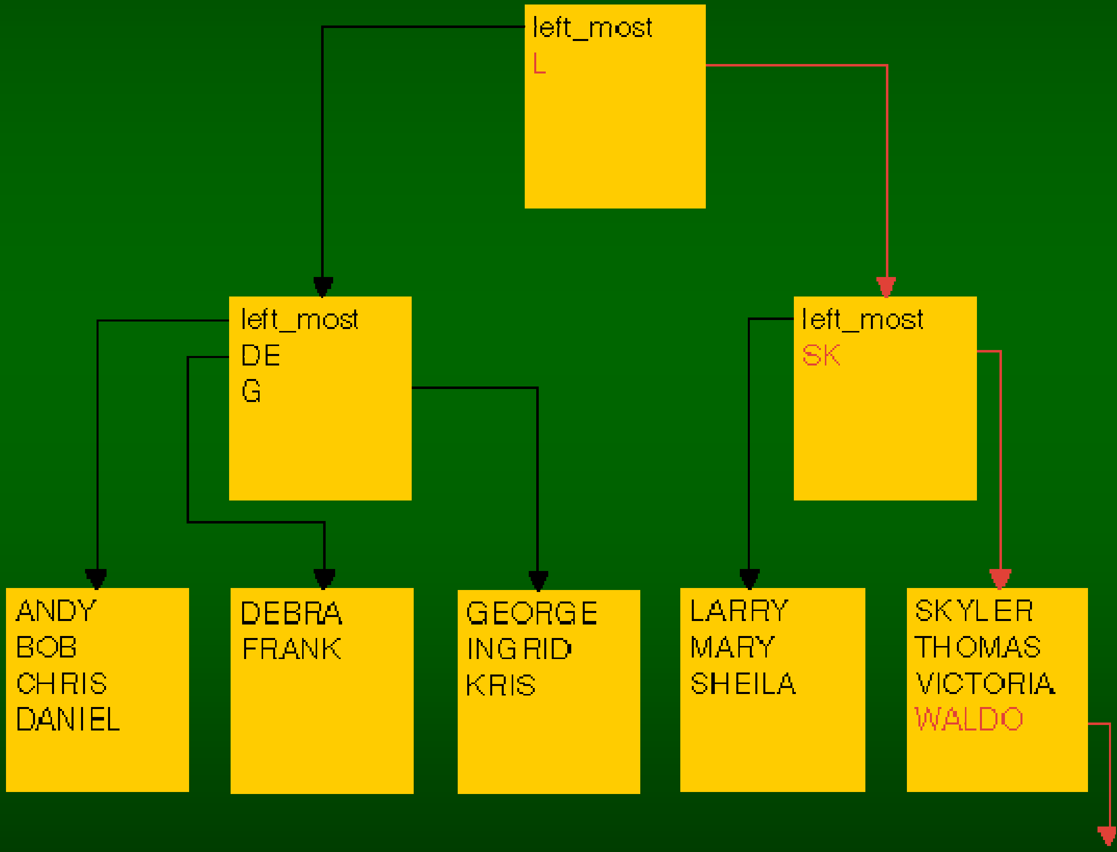
LARRY
MARY
SHEILA

SKYLER
THOMAS
VICTORIA
WALDO

Where’s Waldo?



```
SQL> select empno
  2         , sal
  3     from emp
  4    where name = 'WALDO'
  5  ;
     EMPNO        SAL
---------- ----------
         1      10000
```



left_most
L

left_most
DE
G

left_most
SK

ANDY
BOB
CHRIS
DANIEL

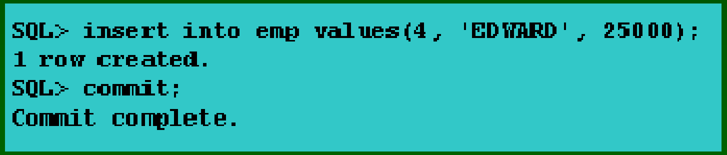
DEBRA
FRANK

GEORGE
INGRID
KRIS

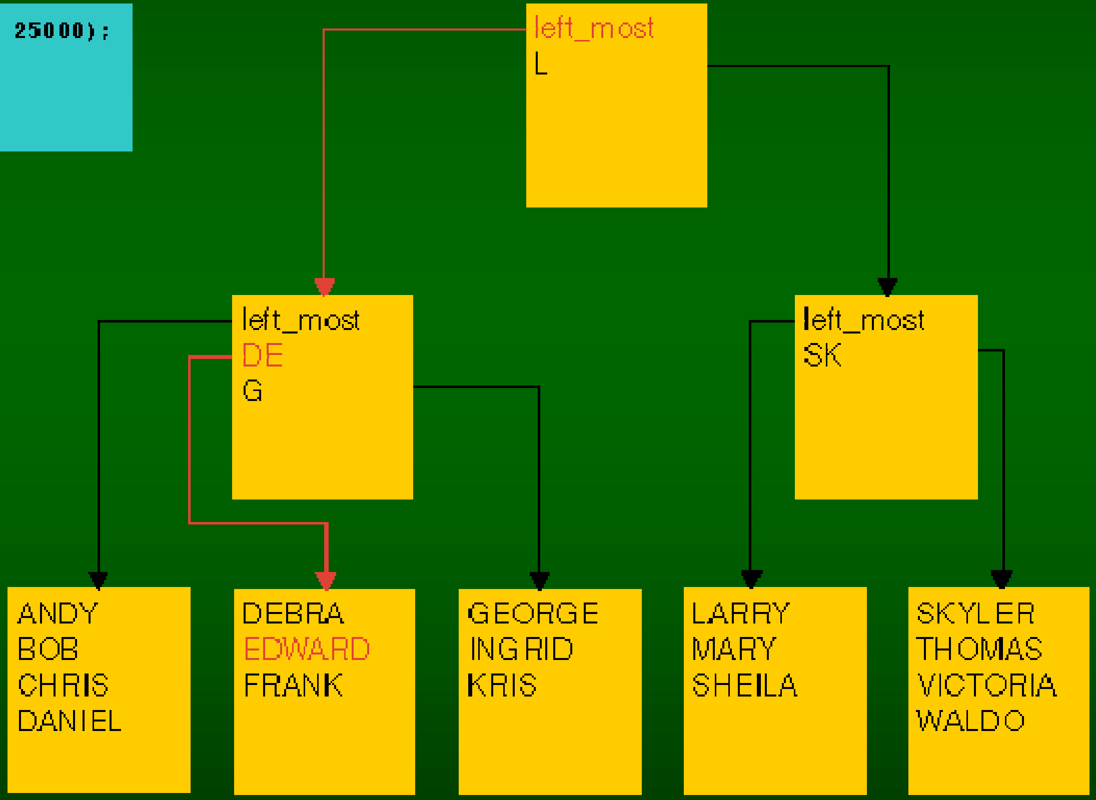
LARRY
MARY
SHEILA

SKYLER
THOMAS
VICTORIA
WALDO

Normal Insertion “Edward”



```
SQL> insert into emp values(4, 'EDWARD', 25000);
1 row created.
SQL> commit;
Commit complete.
```



left_most
L

left_most
DE
G

left_most
SK

ANDY
BOB
CHRIS
DANIEL

DEBRA
EDWARD
FRANK

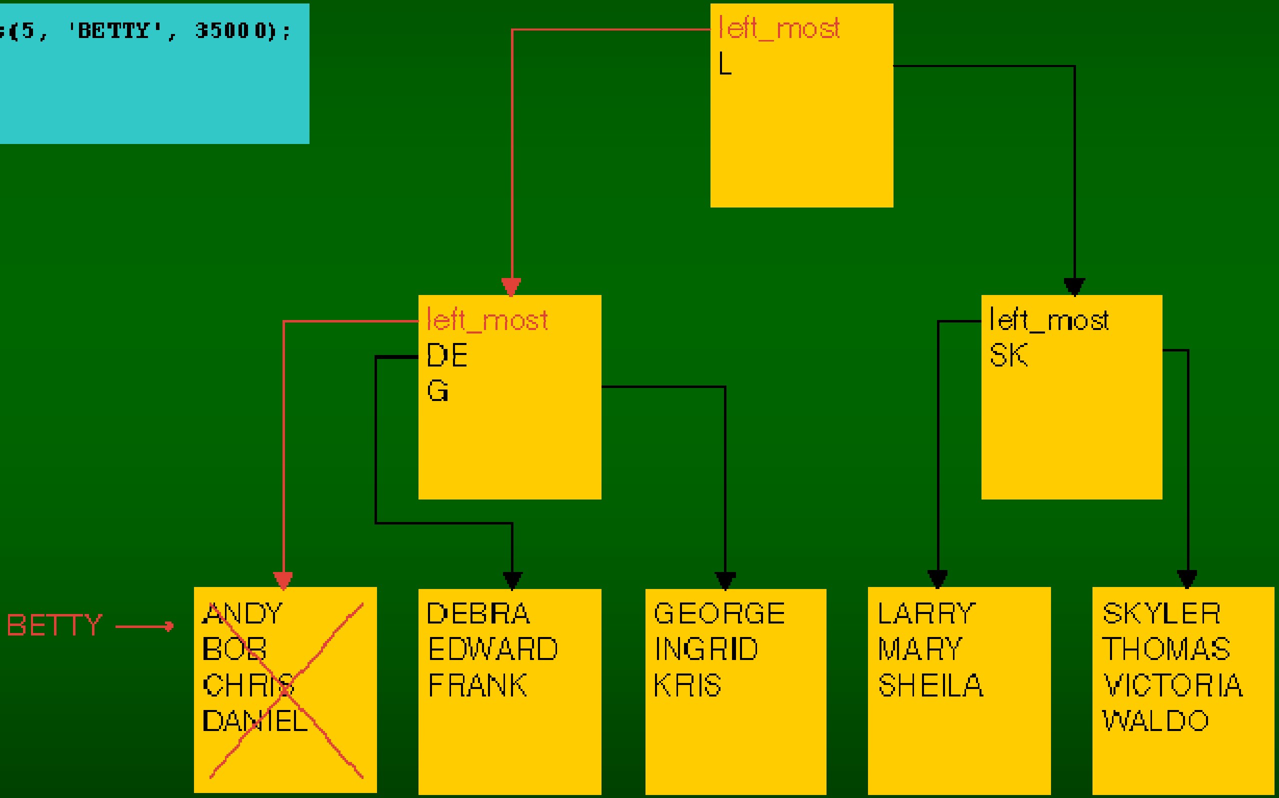
GEORGE
INGRID
KRIS

LARRY
MARY
SHEILA

SKYLER
THOMAS
VICTORIA
WALDO

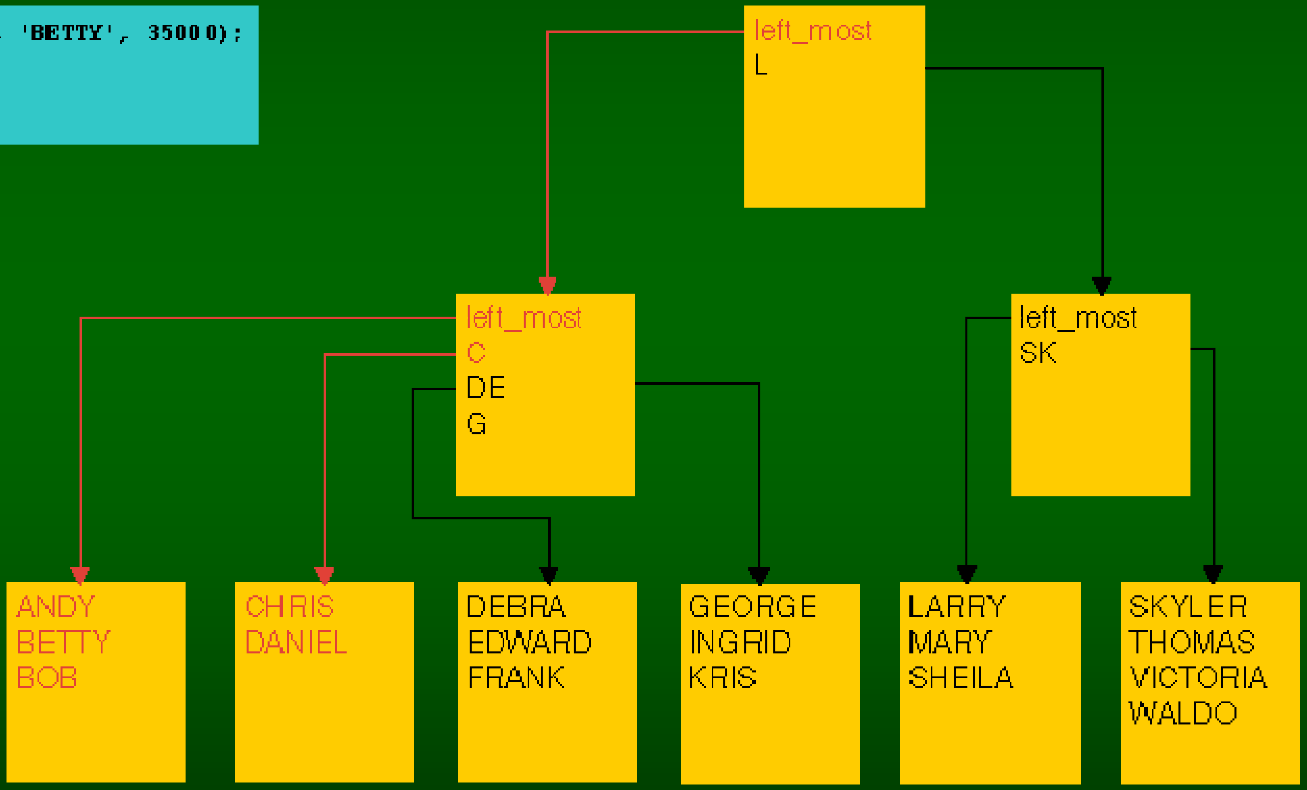
What about Betty?

```
SQL> insert into emp values(5, 'BETTY', 35000);
1 row created.
SQL> commit;
Commit complete.
```



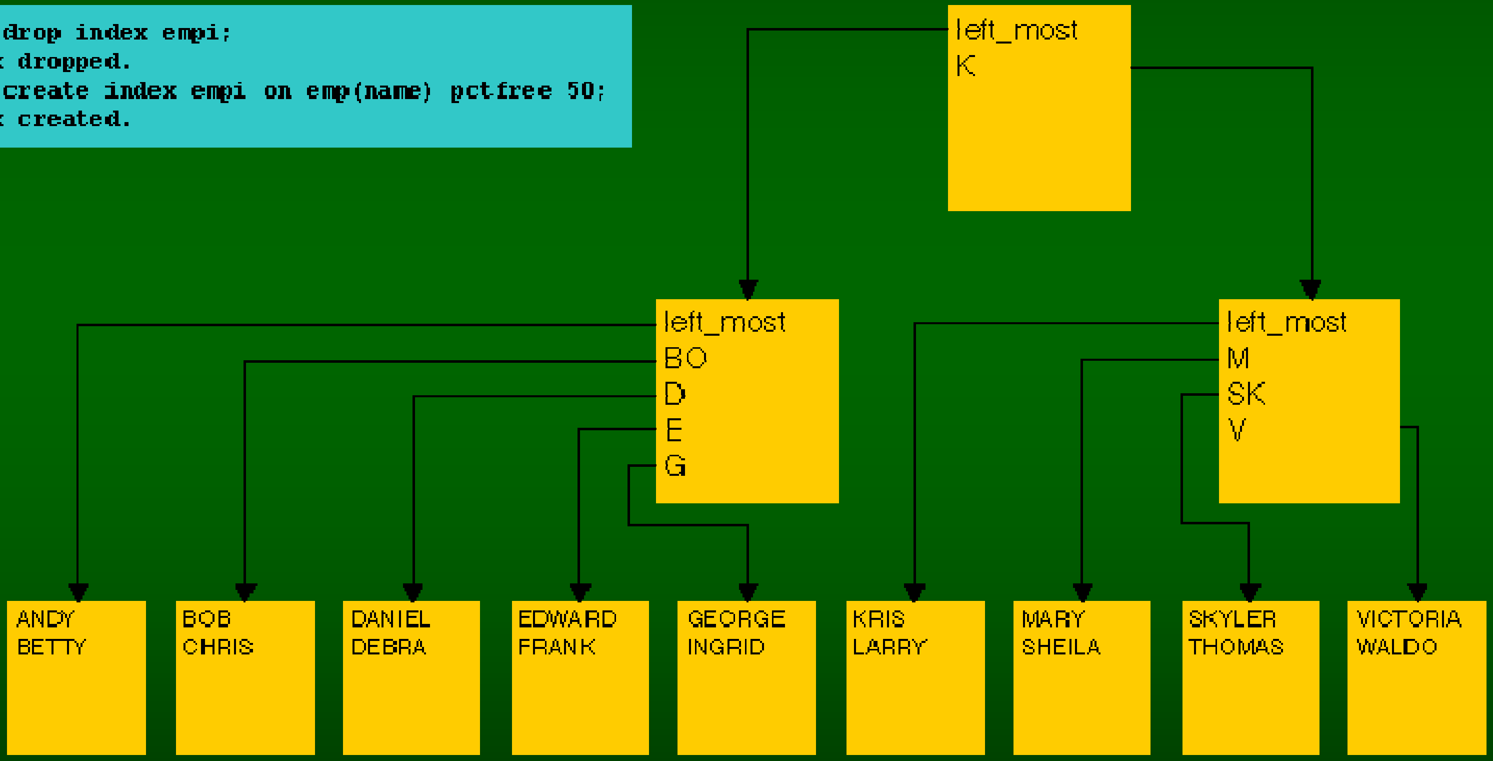
Leaf block split

```
SQL> insert into emp values(5, 'BETTY', 35000);
1 row created.
SQL> commit;
Commit complete.
```

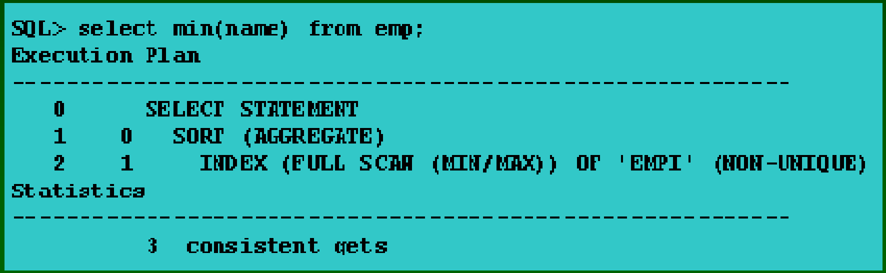


Periodic rebuilds with pct_free

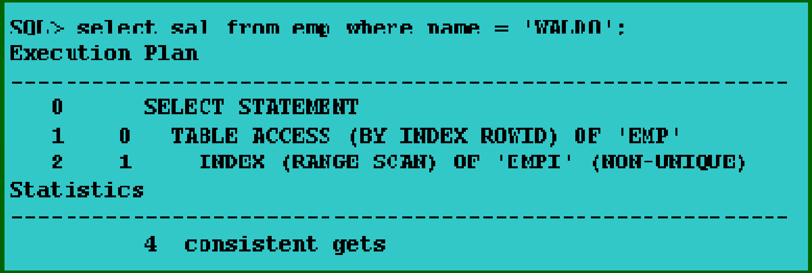
```
SQL> drop index empi;
Index dropped.
SQL> create index empi on emp(name) pctfree 50;
Index created.
```



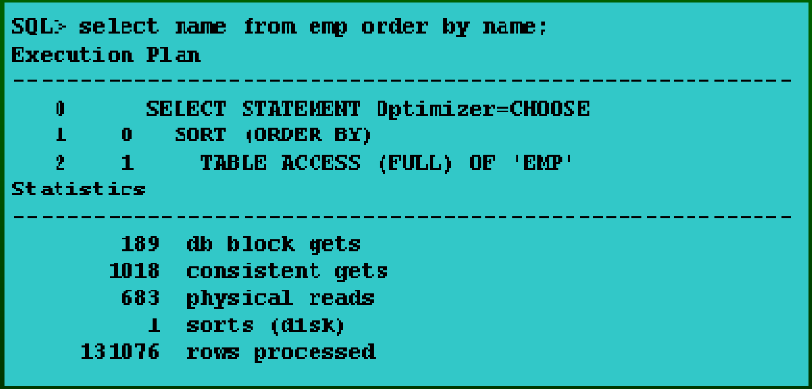
Queries which benefit



```
SQL> select min(name) from emp;
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT
   1    0   SORT (AGGREGATE)
   2    1     INDEX (FULL SCAN (MIN/MAX)) OF 'EMPI' (NON-UNIQUE)
Statistics
----------------------------------------------------------
          3  consistent gets
```



```
SQL> select sal from emp where name = 'WALDO';
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'EMP'
   2    1     INDEX (RANGE SCAN) OF 'EMPI' (NON-UNIQUE)
Statistics
----------------------------------------------------------
          4  consistent gets
```

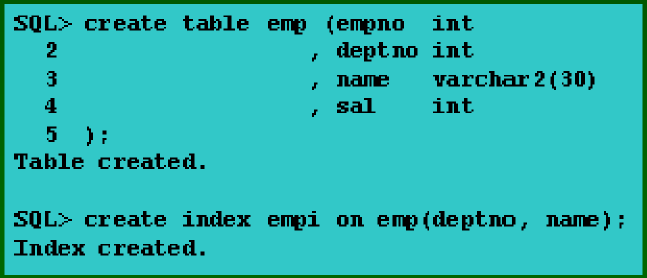


```
SQL> select name from emp order by name;
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (ORDER BY)
   2    1     TABLE ACCESS (FULL) OF 'EMP'
Statistics
----------------------------------------------------------
        189  db block gets
       1018  consistent gets
        683  physical reads
          1  sorts (disk)
     131076  rows processed
```

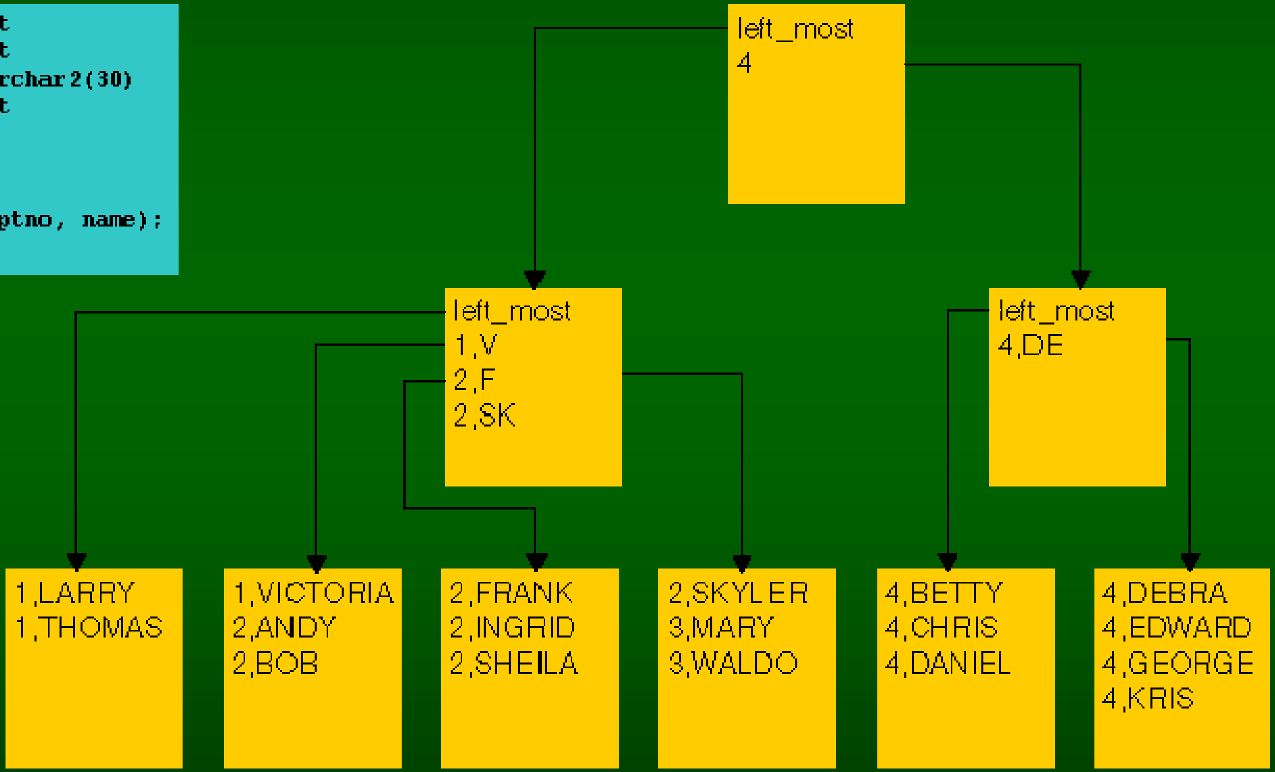
Compound Key

- Concatenation of two or more keys
- More cheating

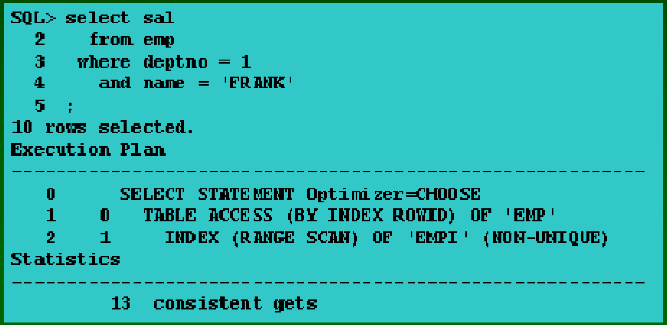
Dept, ename



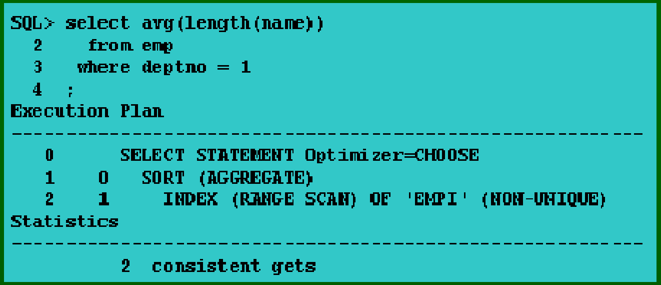
```
SQL> create table emp (empno  int
  2                  , deptno int
  3                  , name   varchar2(30)
  4                  , sal    int
  5  );
Table created.

SQL> create index empi on emp(deptno, name);
Index created.
```



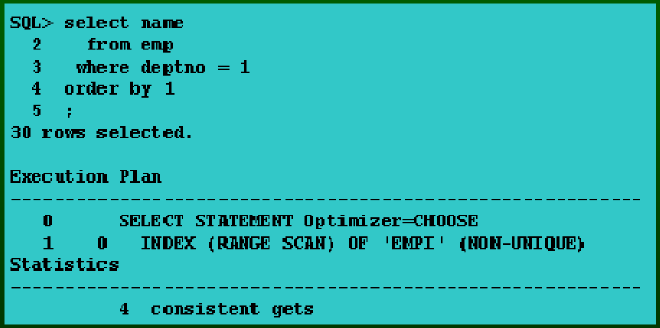
Queries which benefit



```
SQL> select sal
  2    from emp
  3   where deptno = 1
  4     and name = 'FRANK'
  5  ;
10 rows selected.
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'EMP'
   2    1     INDEX (RANGE SCAN) OF 'EMPI' (NON-UNIQUE)
Statistics
----------------------------------------------------------
         13  consistent gets
```



```
SQL> select avg(length(name))
  2    from emp
  3   where deptno = 1
  4  ;
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (AGGREGATE)
   2    1     INDEX (RANGE SCAN) OF 'EMPI' (NON-UNIQUE)
Statistics
----------------------------------------------------------
          2  consistent gets
```

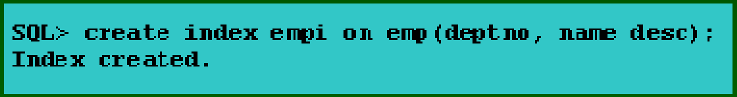


```
SQL> select name
  2    from emp
  3   where deptno = 1
  4  order by 1
  5  ;
30 rows selected.

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   INDEX (RANGE SCAN) OF 'EMPI' (NON-UNIQUE)
Statistics
----------------------------------------------------------
          4  consistent gets
```

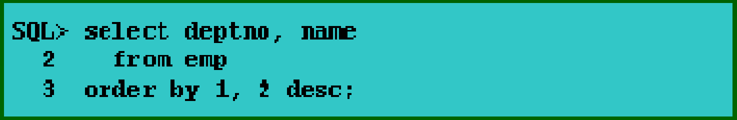
Descending

- Stores keys in reverse order
- Oracle can already scan indexes backwards so why this?
- Again anticipation of sort

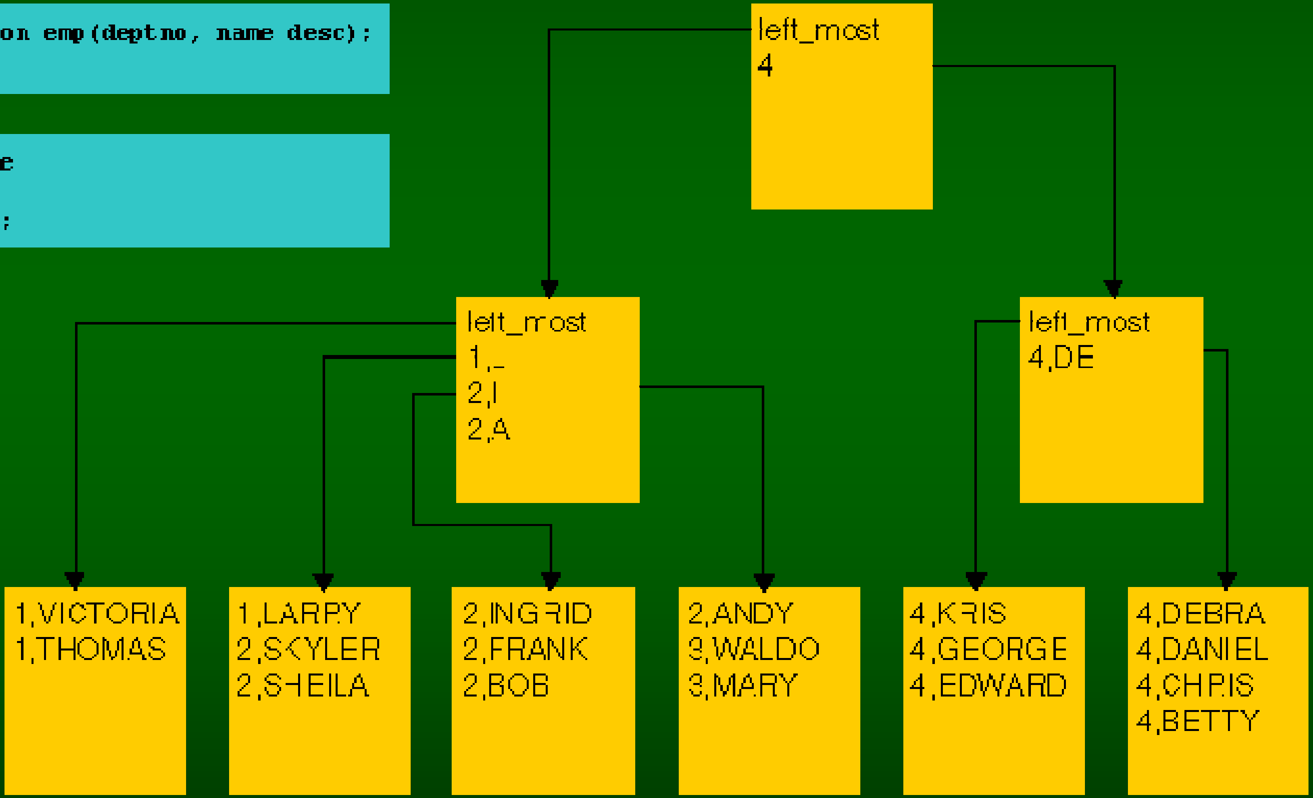
Descending



```
SQL> create index empi on emp(deptno, name desc);
Index created.
```



```
SQL> select deptno, name
  2     from emp
  3  order by 1, 2 desc;
```



left_most
4

left_most
1,_
2,I
2,A

left_most
4,DE

1,VICTORIA
1,THOMAS

1,LARRY
2,SKYLER
2,SHEILA

2,INGRID
2,FRANK
2,BOB

2,ANDY
3,WALDO
3,MARY

4,KRIS
4,GEORGE
4,EDWARD

4,DEBRA
4,DANIEL
4,CHRIS
4,BETTY

Functional Indexes

- Index on f(x)
- Oracle built in functions
- User defined functions
- Whopper of a cheat

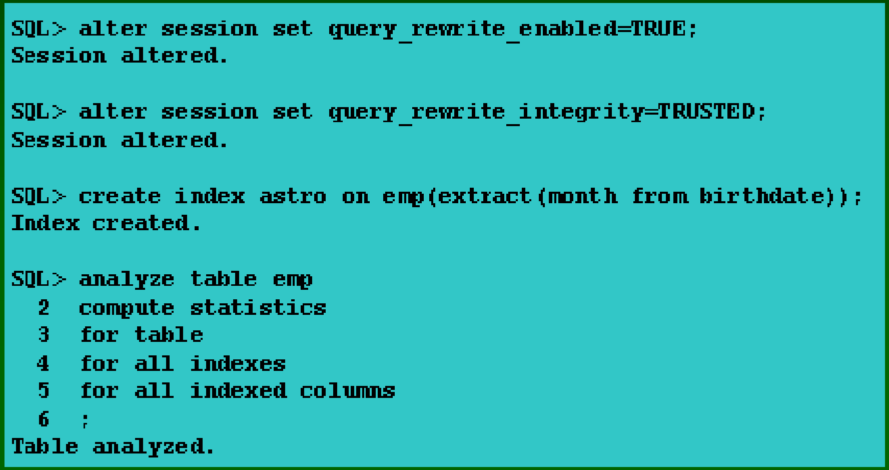
Oracle Functions

- UPPER()
- TRIM()
- SUBSTR()
- MONTHS_BETWEEN()
- LENGTH()
- DECODE()

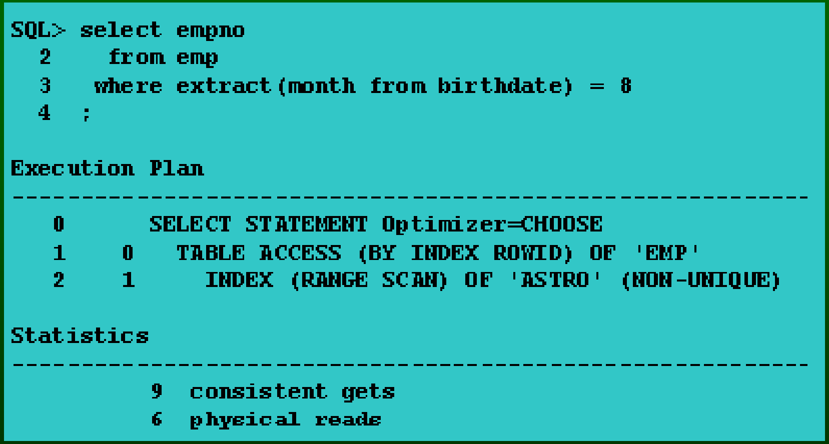
User Functions

- student_rank(SAT, GPA, is_residentp)
- credit_score(income, years_on_job,…)
- astro(birth_date)

Functional Indexes



```
SQL> alter session set query_rewrite_enabled=TRUE;
Session altered.

SQL> alter session set query_rewrite_integrity=TRUSTED;
Session altered.

SQL> create index astro on emp(extract(month from birthdate));
Index created.

SQL> analyze table emp
  2  compute statistics
  3  for table
  4  for all indexes
  5  for all indexed columns
  6  ;
Table analyzed.
```



```
SQL> select empno
  2    from emp
  3   where extract(month from birthdate) = 8
  4  ;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'EMP'
   2    1     INDEX (RANGE SCAN) OF 'ASTRO' (NON-UNIQUE)

Statistics
----------------------------------------------------------
          9  consistent gets
          6  physical reads
```

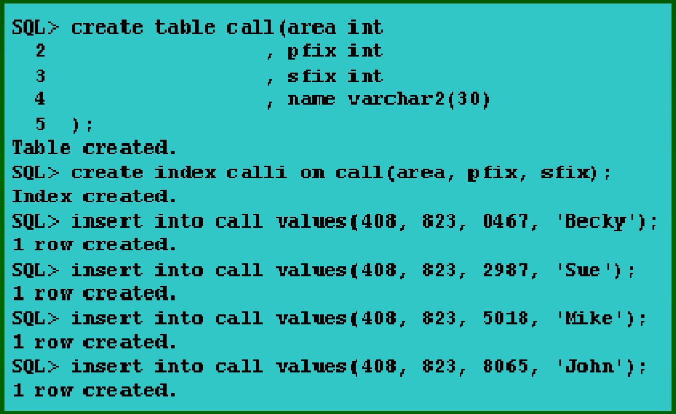
Issues

- Query_rewrite_enabled=TRUE
- Query_rewrite_integrity=TRUSTED
- Substr(varchar2)
- “deterministic” for user defined functions
- DML more expensive

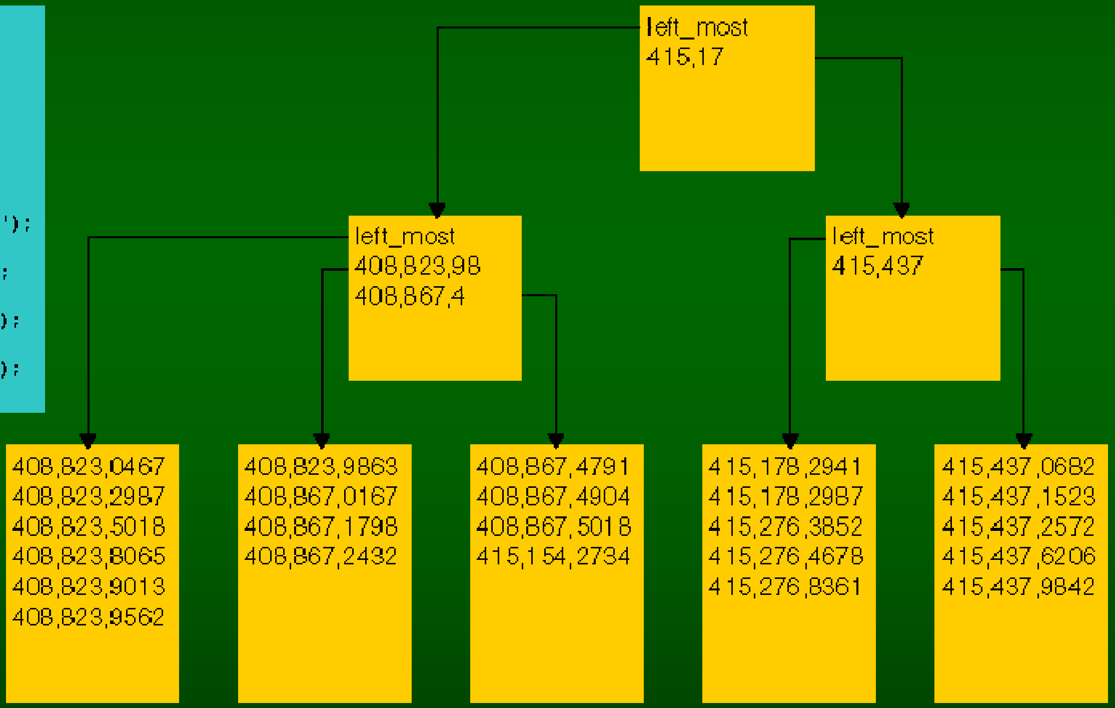
Compressed Indexes

- Saves space in concatenated indexes with highly repetitive data
- Particularly good for large keys

Call List (Uncompressed)



```
SQL> create table call(area int
  2                    , pfix int
  3                    , sfix int
  4                    , name varchar2(30)
  5  );
Table created.
SQL> create index calli on call(area, pfix, sfix);
Index created.
SQL> insert into call values(408, 823, 0467, 'Becky');
1 row created.
SQL> insert into call values(408, 823, 2987, 'Sue');
1 row created.
SQL> insert into call values(408, 823, 5018, 'Mike');
1 row created.
SQL> insert into call values(408, 823, 8065, 'John');
1 row created.
```



left_most
415,17

left_most
408,823,98
408,867,4

left_most
415,437

408,823,0467
408,823,2987
408,823,5018
408,823,8065
408,823,9013
408,823,9562

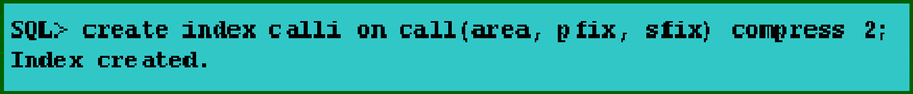
408,823,9863
408,867,0167
408,867,1798
408,867,2432

408,867,4791
408,867,4904
408,867,5018
415,154,2734

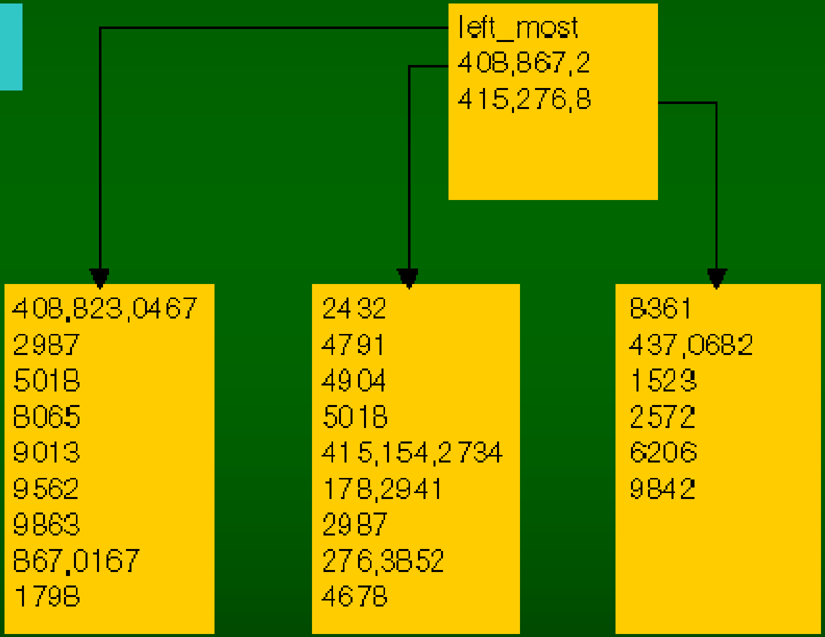
415,178,2941
415,178,2987
415,276,3852
415,276,4678
415,276,8361

415,437,0682
415,437,1523
415,437,2572
415,437,6206
415,437,9842

Call List (Compressed)



```
SQL> create index calli on call(area, pfix, sfix) compress 2;
Index created.
```



left_most
408,867,2
415,276,8

408,823,0467
2987
5018
8065
9013
9562
9863
867,0167
1798

2432
4791
4904
5018
415,154,2734
178,2941
2987
276,3852
4678

8361
437,0682
1523
2572
6206
9842

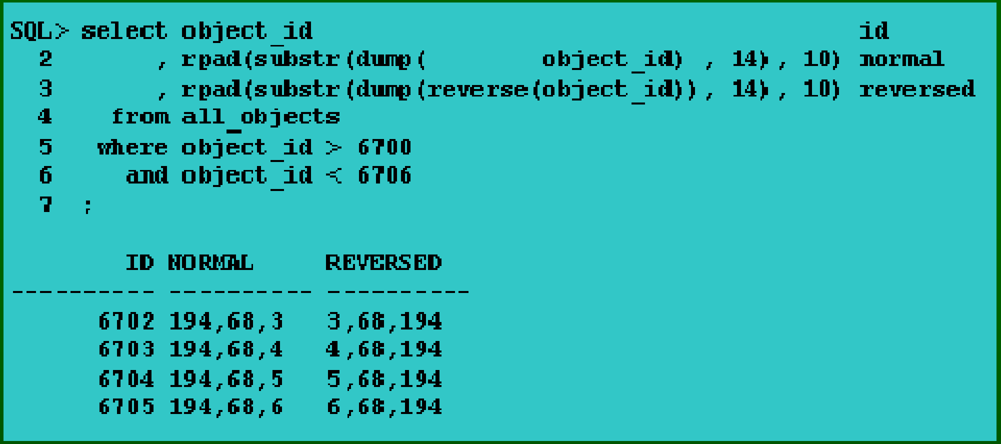
Issues with Compression

- Saves space
- Potentially reduces physical I/Os
- Increased CPU usage

Reverse Key

- Purpose built for parallel server
- Reverses Oracle representation of key
- Generates pseudo-randomness

Reverse Key



```
SQL> select object_id                                              id
  2        , rpad(substr(dump(        object_id)  , 14) , 10) normal
  3        , rpad(substr(dump(reverse(object_id)) , 14) , 10) reversed
  4     from all_objects
  5    where object_id > 6700
  6      and object_id < 6706
  7  ;

        ID NORMAL     REVERSED
---------- ---------- ----------
      6702 194,68,3   3,68,194
      6703 194,68,4   4,68,194
      6704 194,68,5   5,68,194
      6705 194,68,6   6,68,194
```

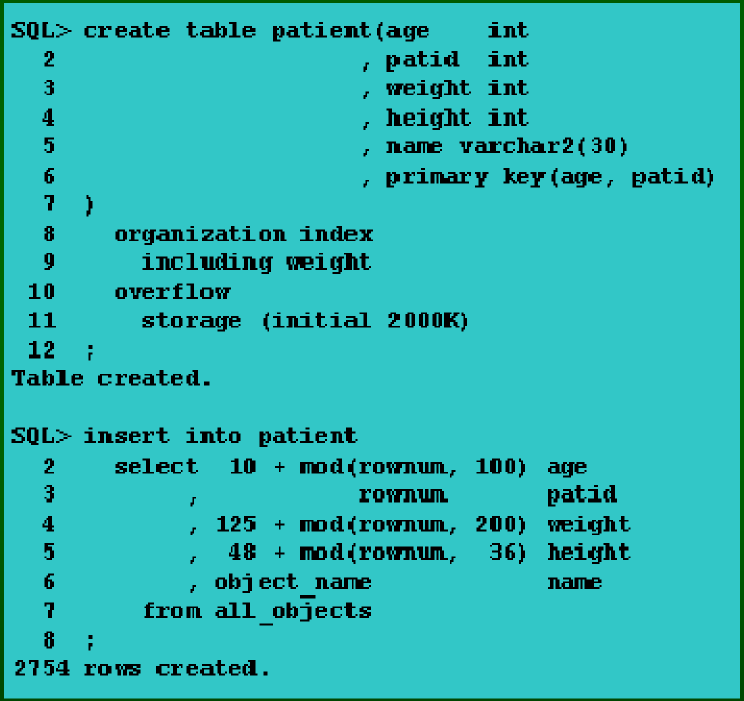
Issues with Reverse Keys

- Index can be used for equality (x = 7)
- But not for range scans (x > 7)

Index Organized Tables

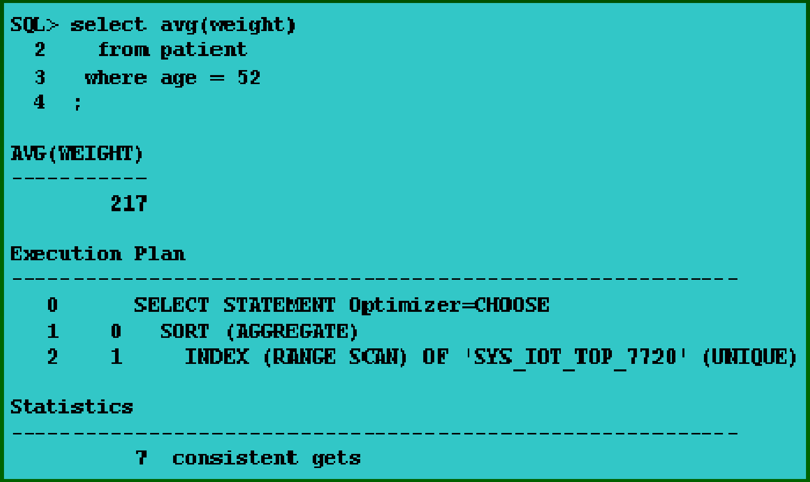
- B*Tree without the table
- Table becomes superfluous
- Co-location of rows with similar or identical key values
- Great for large lookup tables

Patient Table

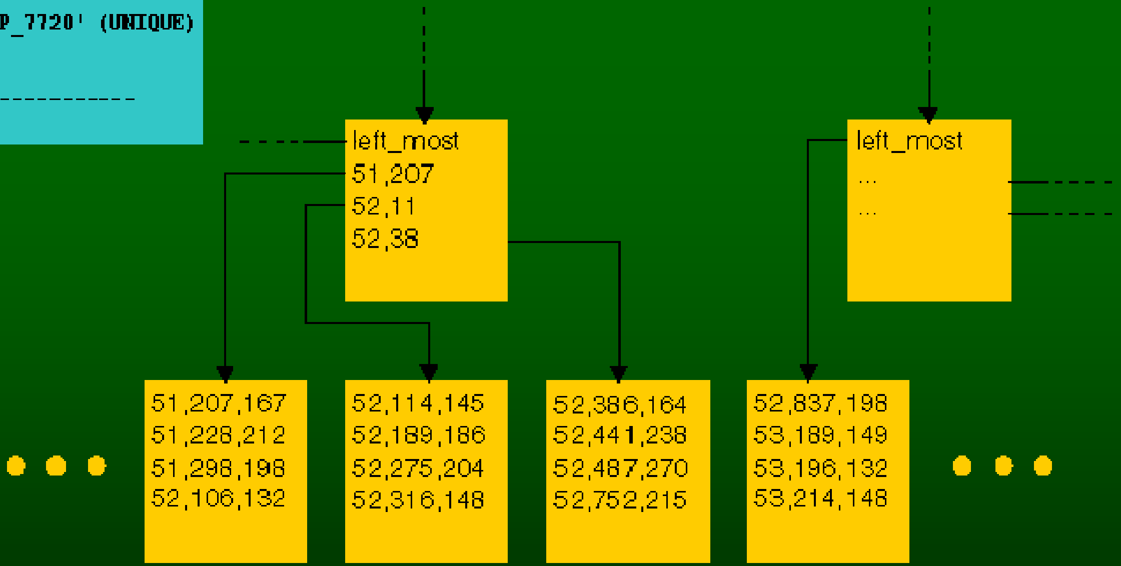


```
SQL> create table patient(age    int
  2                       , patid  int
  3                       , weight int
  4                       , height int
  5                       , name varchar2(30)
  6                       , primary key(age, patid)
  7  )
  8    organization index
  9      including weight
 10    overflow
 11      storage (initial 2000K)
 12  ;
Table created.

SQL> insert into patient
  2    select  10 + mod(rownum, 100)  age
  3         ,           rownum        patid
  4         , 125 + mod(rownum, 200)  weight
  5         ,  48 + mod(rownum,  36)  height
  6         , object_name             name
  7      from all_objects
  8  ;
2754 rows created.
```

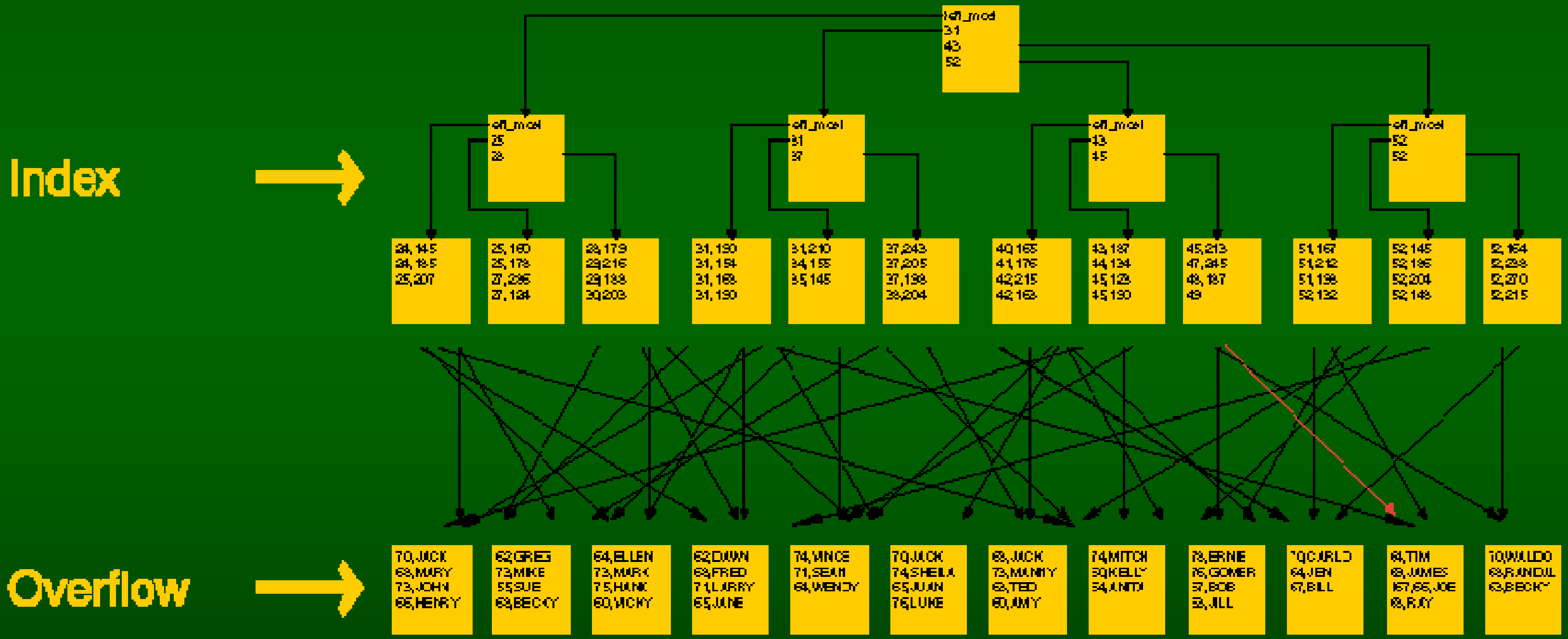
Patient Table



```
SQL> select avg(weight)
  2     from patient
  3    where age = 52
  4   ;

AVG(WEIGHT)
-----------
        217

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (AGGREGATE)
   2    1     INDEX (RANGE SCAN) OF 'SYS_IOT_TOP_7720' (UNIQUE)

Statistics
----------------------------------------------------------
          7  consistent gets
```



Overflow Table



Index

ofl_mod
31
43
52

ofl_mod
25
28

ofl_mod
31
37

ofl_mod
43
45

ofl_mod
52
52

24, 145
24, 185
25,207

25, 160
25, 178
27, 236
27, 124

28, 179
29,216
29, 188
30,203

31, 190
31, 154
31, 168
31, 190

31,210
34, 155
35,145

37,243
37,205
37,198
38,204

40,165
41,176
42,215
42, 168

43, 187
44, 134
45,128
45,130

45,213
47,245
48, 187
49

51, 167
51,212
51, 198
52, 132

52, 145
52, 195
52,204
52,148

52, 164
52,238
52,270
52,215

Overflow

70,JACK
68,MARY
73,JOHN
66,HENRY

62,GREG
73,MIKE
55,SUE
63,BECKY

64,ELLEN
73,MARK
75,HANK
60,VICKY

62,DAWN
63,FRED
71,LARRY
65,JUNE

74,VINCE
71,SEAN
64,WENDY

70,JACK
74,SHEILA
65,JUAN
76,LUKE

68,JACK
73,MANNY
62,TED
60,AMY

74,MITCH
50,KELLY
54,ANITA

78,ERNIE
76,GOMER
57,BOB
53,JILL

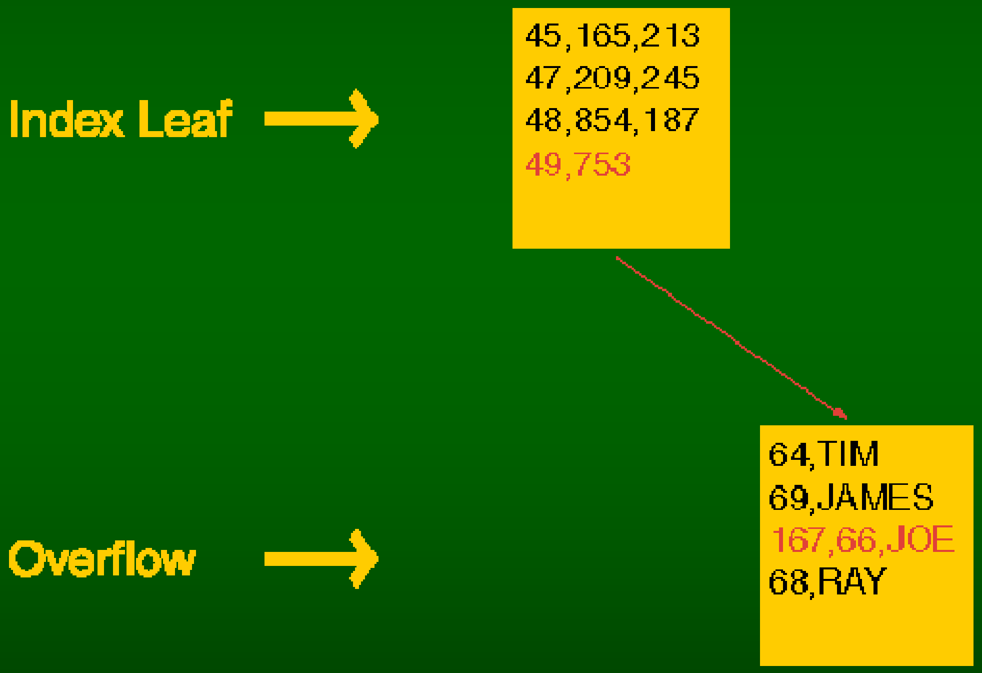
70,CARLO
64,JEN
67,BILL

64,TIM
68,JAMES
67,66,JOE
68,RAY

70,WALDO
68,RANDAL
63,BECKY

TERLINGUA SOFTWARE

Full Table Scan Problem



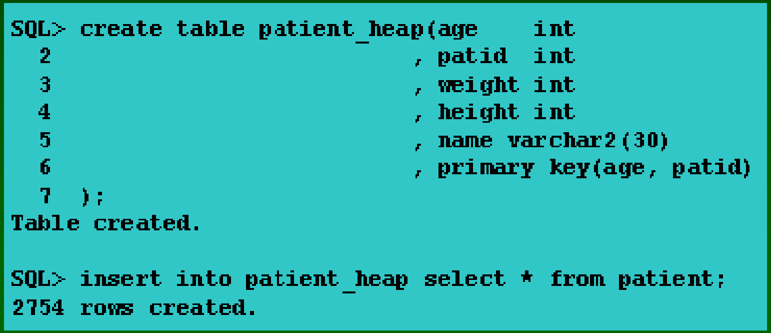
Index Leaf →

45,165,213
47,209,245
48,854,187
49,753

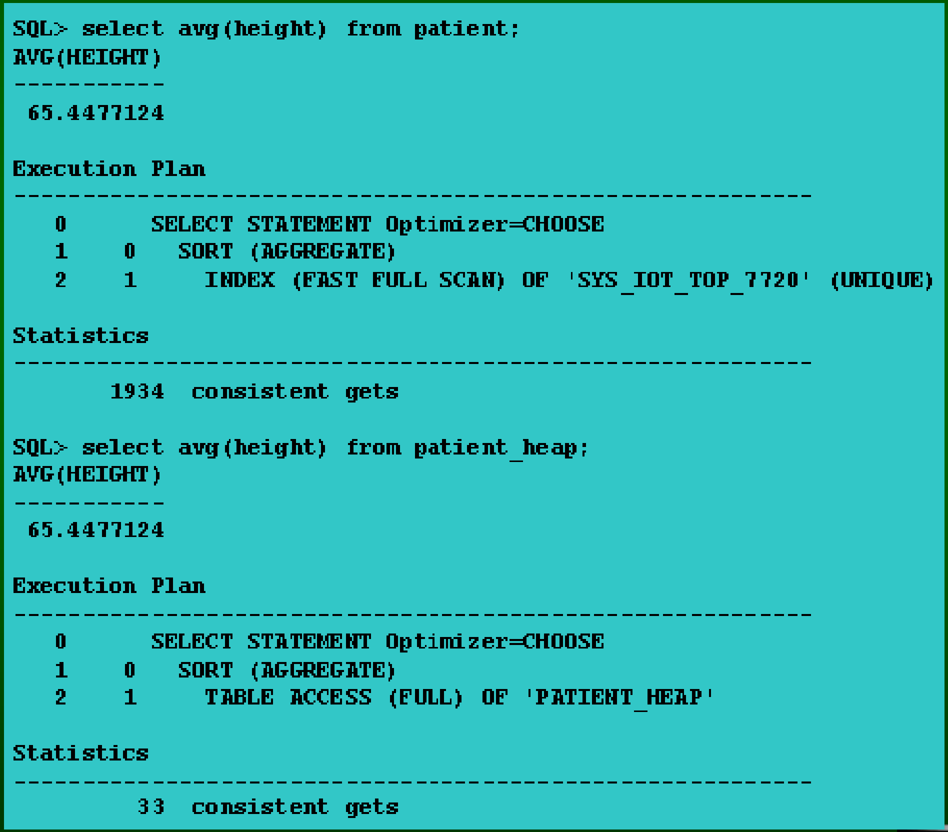
Overflow →

64,TIM
69,JAMES
167,66,JOE
68,RAY

Full scans IOT vs. heap



```
SQL> create table patient_heap(age    int
  2                           , patid  int
  3                           , weight int
  4                           , height int
  5                           , name varchar2(30)
  6                           , primary key(age, patid)
  7  );
Table created.

SQL> insert into patient_heap select * from patient;
2754 rows created.
```

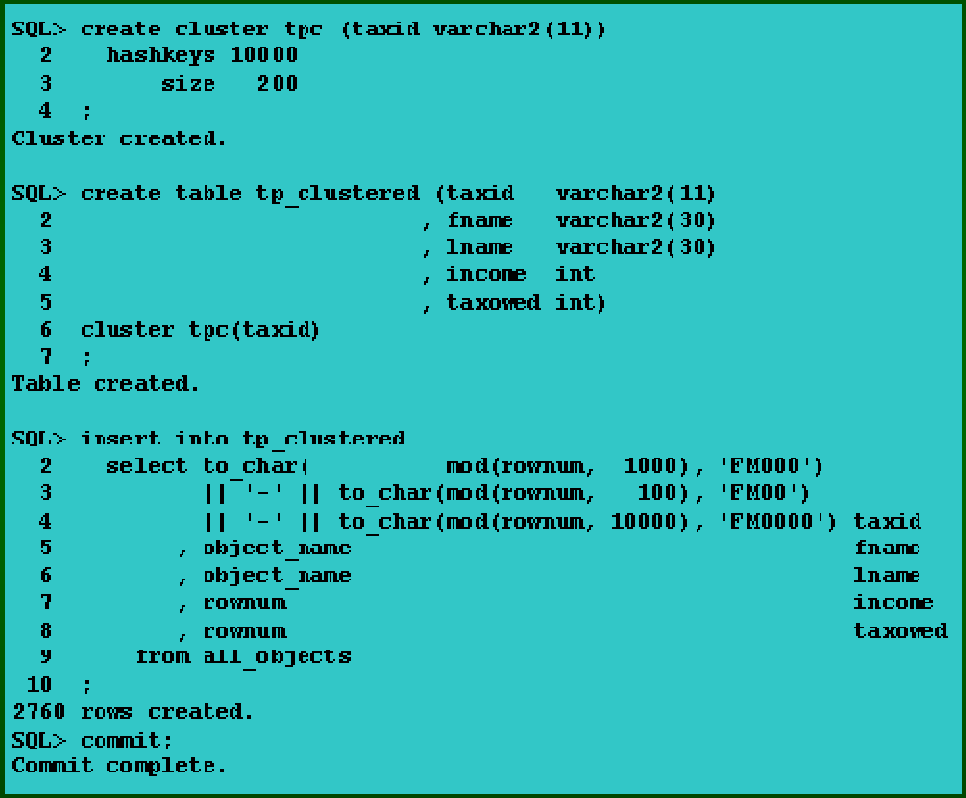


```
SQL> select avg(height) from patient;
AVG(HEIGHT)
-----------
 65.4477124

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (AGGREGATE)
   2    1     INDEX (FAST FULL SCAN) OF 'SYS_IOT_TOP_7720' (UNIQUE)

Statistics
----------------------------------------------------------
       1934  consistent gets

SQL> select avg(height) from patient_heap;
AVG(HEIGHT)
-----------
 65.4477124

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (AGGREGATE)
   2    1     TABLE ACCESS (FULL) OF 'PATIENT_HEAP'

Statistics
----------------------------------------------------------
         33  consistent gets
```

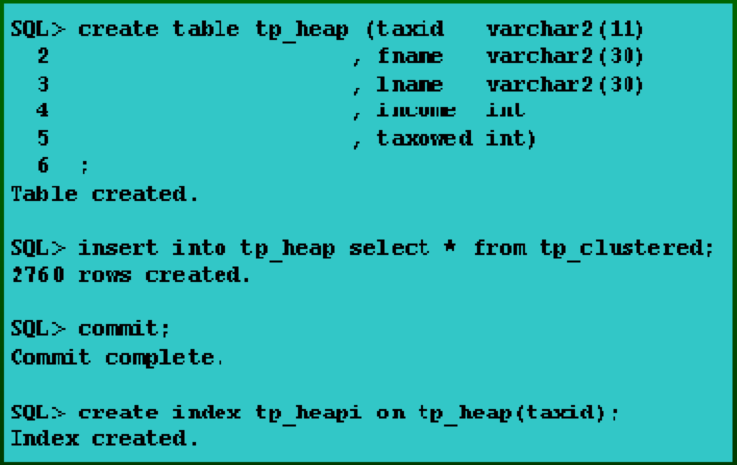
Hashed Clusters

- Avoids index all together
- Great for very large fixed size tables

taxpayer Hashed Cluster

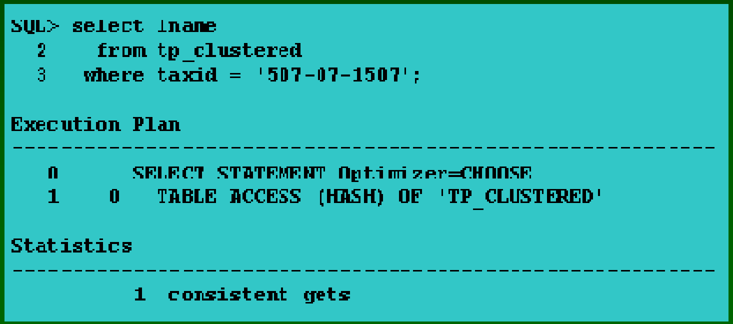


```
SQL> create cluster tpc  (taxid varchar2(11))
  2    hashkeys 10000
  3        size   200
  4  ;
Cluster created.

SQL> create table tp_clustered (taxid    varchar2(11)
  2                            , fname    varchar2(30)
  3                            , lname    varchar2(30)
  4                            , income   int
  5                            , taxowed int)
  6  cluster tpc(taxid)
  7  ;
Table created.

SQL> insert into tp_clustered
  2    select to_char(          mod(rownum,  1000), 'FM000')
  3           || '-' || to_char(mod(rownum,   100), 'FM00')
  4           || '-' || to_char(mod(rownum, 10000), 'FM0000') taxid
  5         , object_name                                      fname
  6         , object_name                                      lname
  7         , rownum                                           income
  8         , rownum                                           taxowed
  9      from all_objects
 10  ;
2760 rows created.
SQL> commit;
Commit complete.
```



```
SQL> create table tp_heap (taxid    varchar2(11)
  2                       , fname    varchar2(30)
  3                       , lname    varchar2(30)
  4                       , income   int
  5                       , taxowed int)
  6  ;
Table created.

SQL> insert into tp_heap select * from tp_clustered;
2760 rows created.

SQL> commit;
Commit complete.

SQL> create index tp_heapi on tp_heap(taxid);
Index created.
```

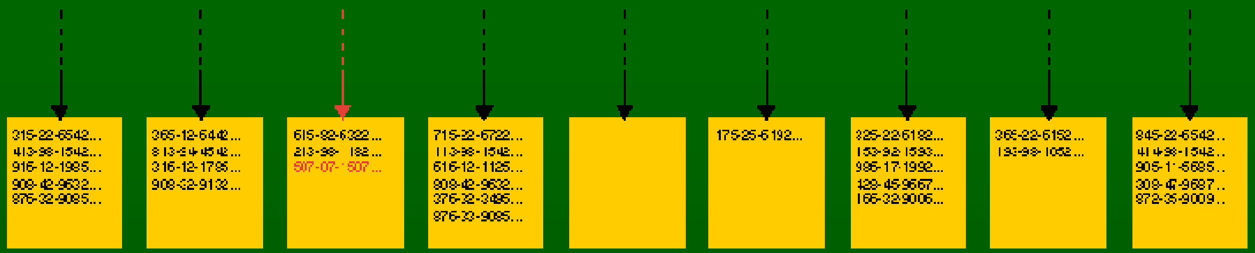
Hash Cluster

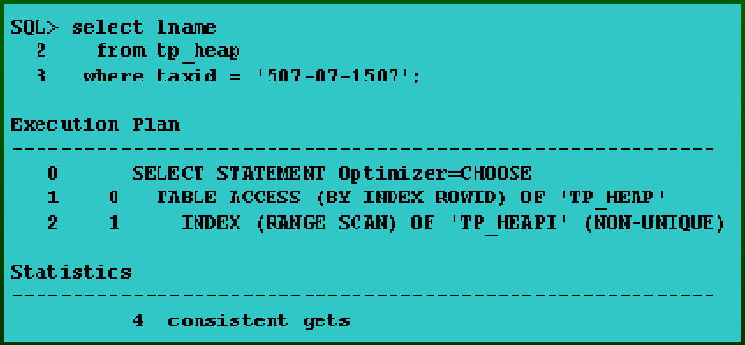


```
SQL> select lname
  2     from tp_clustered
  3   where taxid = '507-07-1507';

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (HASH) OF 'TP_CLUSTERED'

Statistics
----------------------------------------------------------
          1  consistent gets
```

507-07-1507

HASH(507-07-1507)

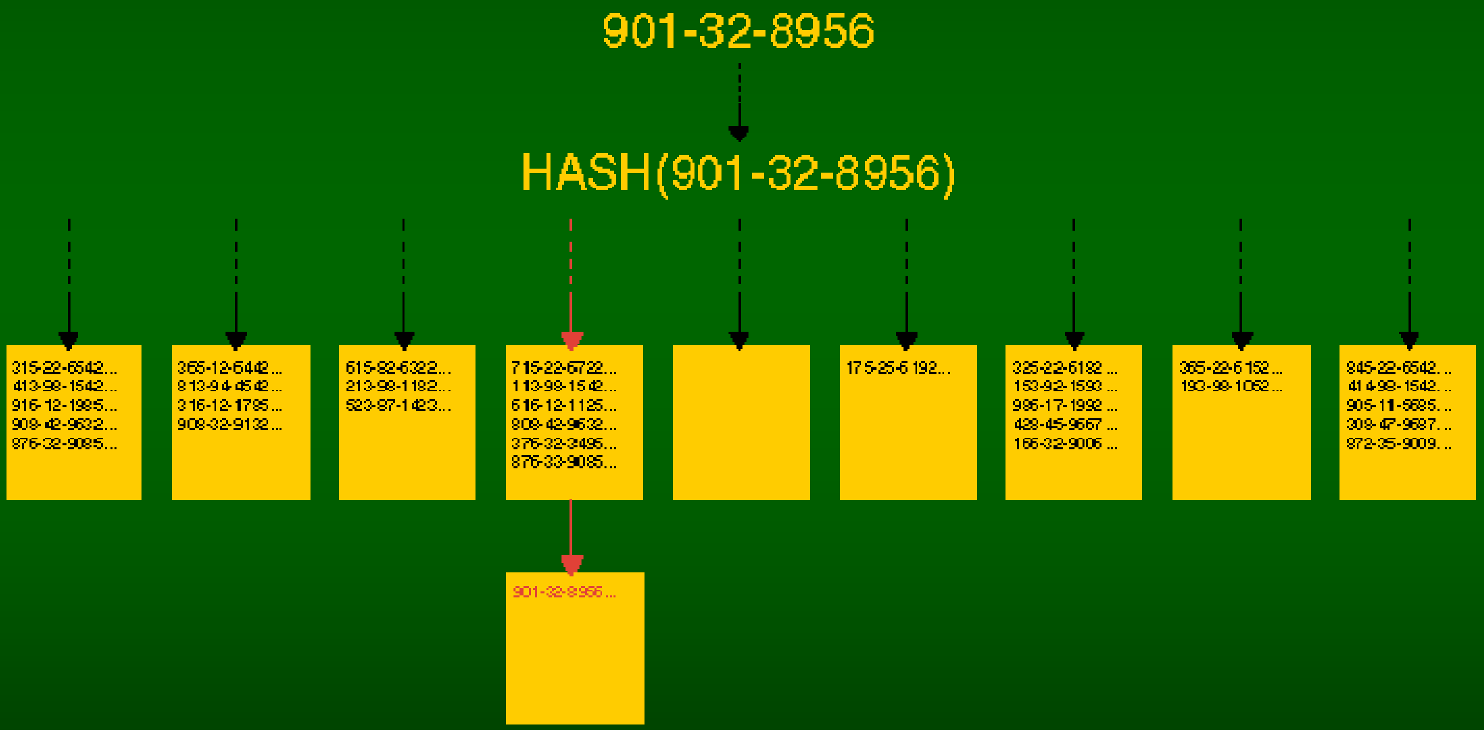




```
SQL> select lname
  2     from tp_heap
  3   where taxid = '507-07-1507';

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'TP_HEAP'
   2    1     INDEX (RANGE SCAN) OF 'TP_HEAPI' (NON-UNIQUE)

Statistics
----------------------------------------------------------
          4  consistent gets
```

TERLINGUA SOFTWARE

Hash Cluster Overflow



901-32-8956

HASH(901-32-8956)

315-22-6542...
413-98-1542...
916-12-1985...
909-42-9632...
876-32-9085...

365-12-6442...
813-94-4542...
316-12-1785...
908-32-9132...

615-82-6322...
213-98-1182...
523-87-1423...

715-22-6722...
113-98-1542...
616-12-1125...
809-42-9632...
376-32-3495...
876-33-9085...

175-25-6192...

325-22-6182 ...
153-92-1593 ...
986-17-1992 ...
428-45-9667 ...
166-32-9006 ...

365-22-6152...
193-98-1062...

845-22-6542...
414-98-1542...
905-11-5685...
309-47-9687...
972-35-9009...

901-32-8956...

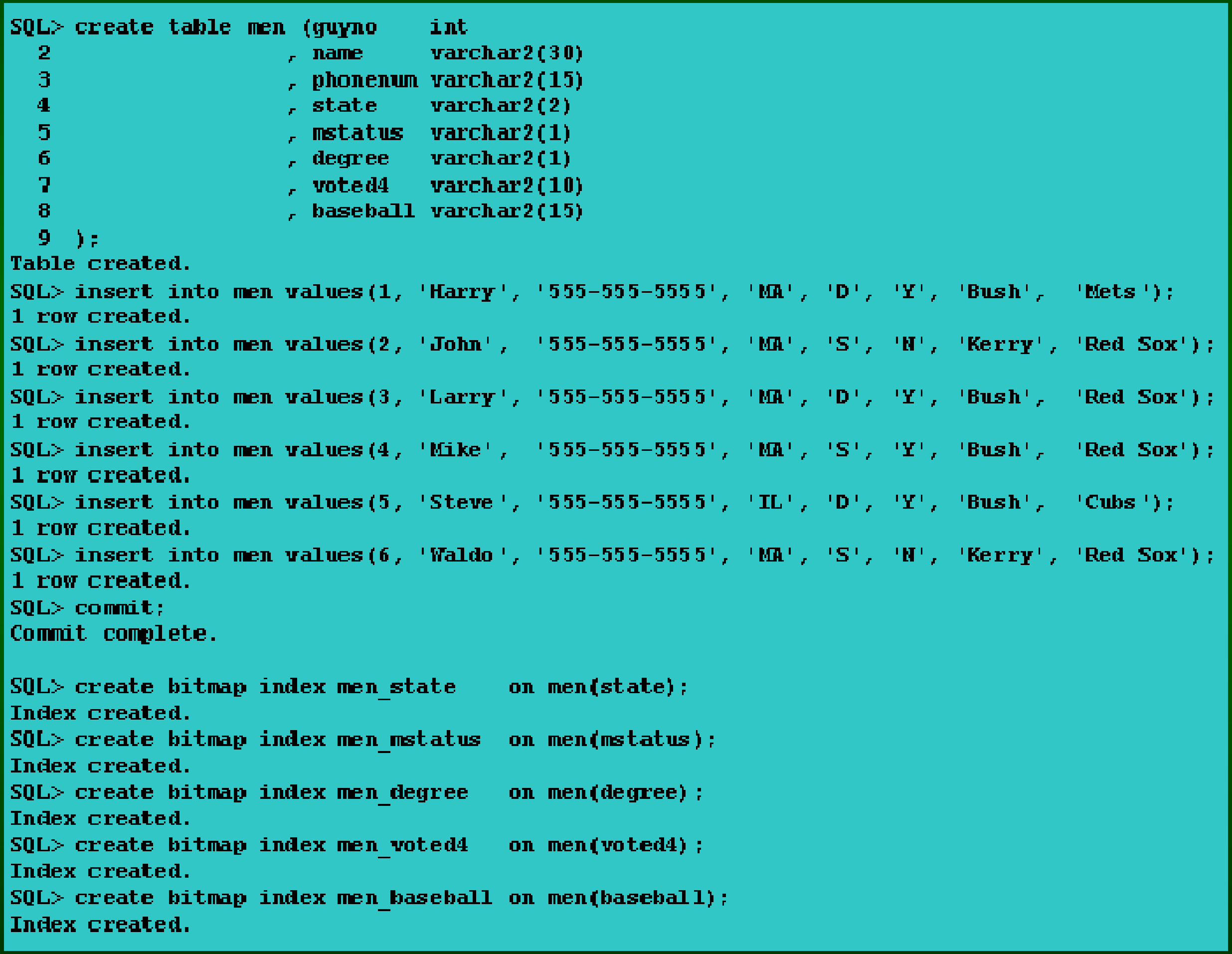
Hashed Cluster Issues

- Great for large fixed sized table
- Can be space inefficient
- Hash collisions possible
- Overflow blocks defeat gains
- May need periodic rebuild

Bitmapped Indexes

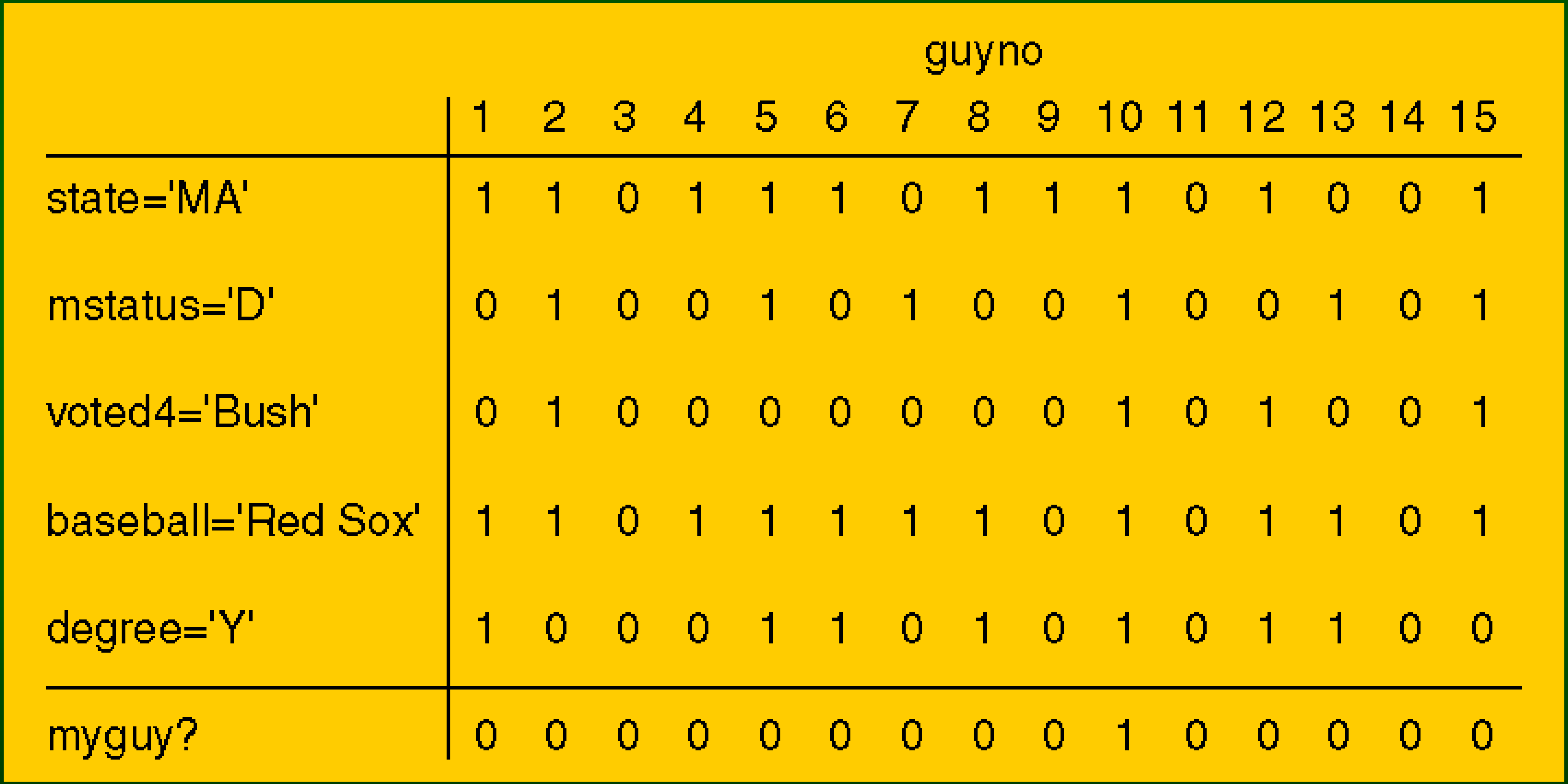
- Low cardinality data
- Great for ad-hoc queries when testing for equality
- Efficiently represented list of rowids that have the same key value
- Excellent for count(*) or existence test

Dating Example



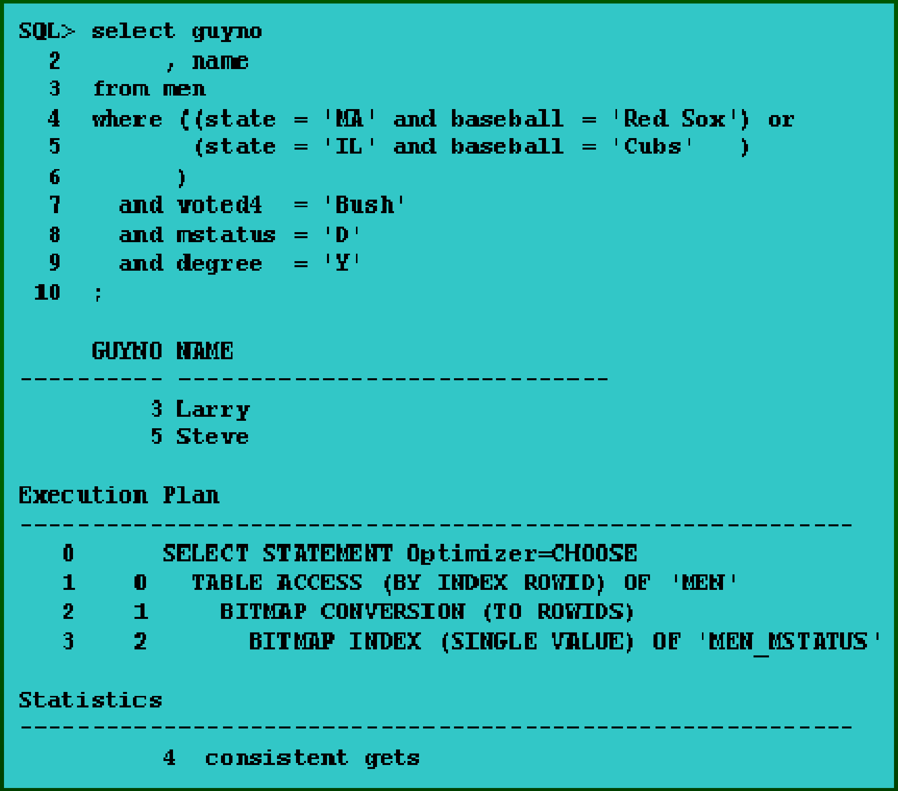
```
SQL> create table men (guyno    int
  2                  , name     varchar2(30)
  3                  , phonenum varchar2(15)
  4                  , state    varchar2(2)
  5                  , mstatus  varchar2(1)
  6                  , degree   varchar2(1)
  7                  , voted4   varchar2(10)
  8                  , baseball varchar2(15)
  9  );
Table created.
SQL> insert into men values(1, 'Harry', '555-555-5555', 'MA', 'D', 'Y', 'Bush',  'Mets');
1 row created.
SQL> insert into men values(2, 'John' , '555-555-5555', 'MA', 'S', 'N', 'Kerry', 'Red Sox');
1 row created.
SQL> insert into men values(3, 'Larry', '555-555-5555', 'MA', 'D', 'Y', 'Bush',  'Red Sox');
1 row created.
SQL> insert into men values(4, 'Mike' , '555-555-5555', 'MA', 'S', 'Y', 'Bush',  'Red Sox');
1 row created.
SQL> insert into men values(5, 'Steve', '555-555-5555', 'IL', 'D', 'Y', 'Bush',  'Cubs');
1 row created.
SQL> insert into men values(6, 'Waldo', '555-555-5555', 'MA', 'S', 'N', 'Kerry', 'Red Sox');
1 row created.
SQL> commit;
Commit complete.

SQL> create bitmap index men_state    on men(state);
Index created.
SQL> create bitmap index men_mstatus  on men(mstatus);
Index created.
SQL> create bitmap index men_degree   on men(degree);
Index created.
SQL> create bitmap index men_voted4   on men(voted4);
Index created.
SQL> create bitmap index men_baseball on men(baseball);
Index created.
```

Bush vote’n, Div, MA, Edu, Sox



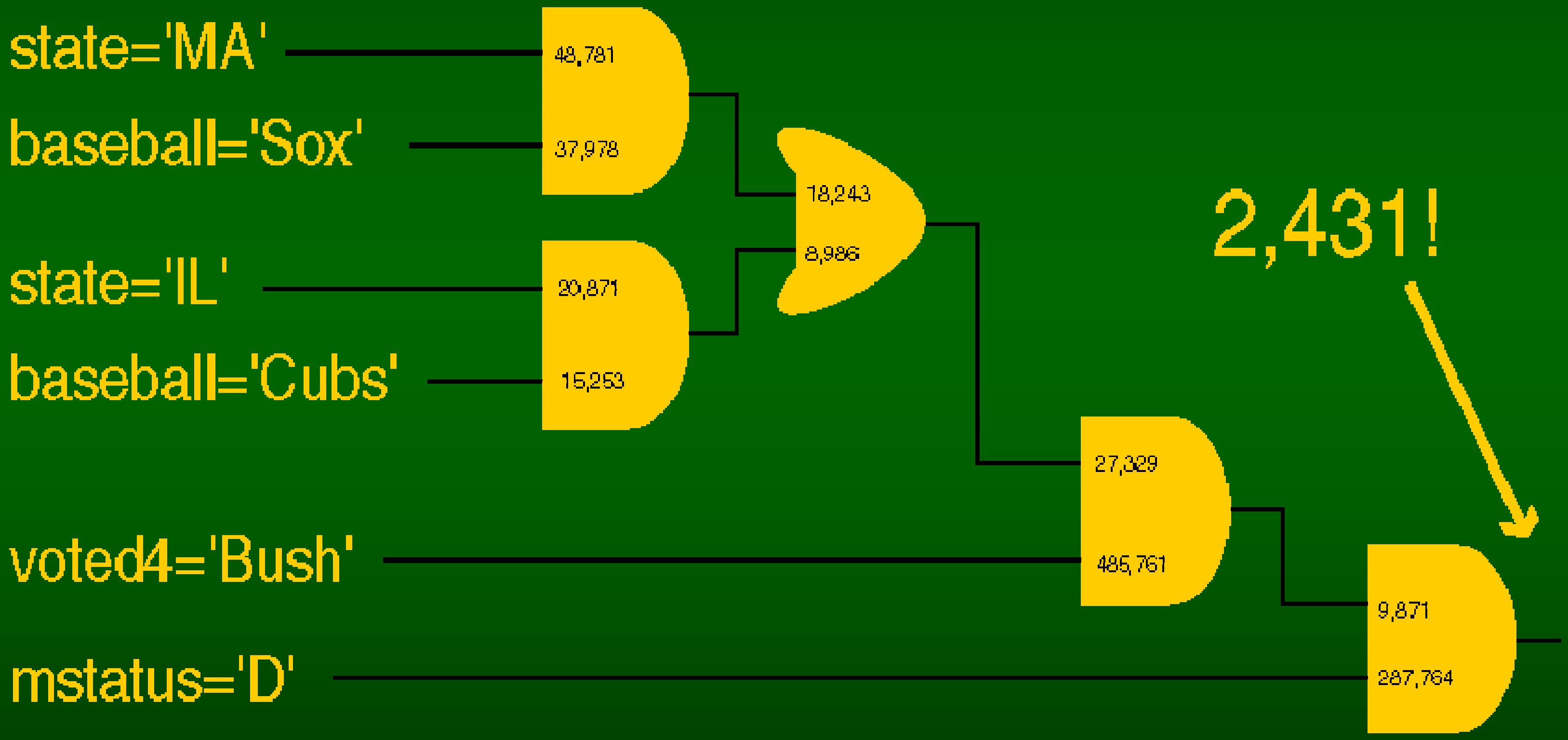
	guyno														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
state='MA'	1	1	0	1	1	1	0	1	1	1	0	1	0	0	1
mstatus='D'	0	1	0	0	1	0	1	0	0	1	0	0	1	0	1
voted4='Bush'	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1
baseball='Red Sox'	1	1	0	1	1	1	1	1	0	1	0	1	1	0	1
degree='Y'	1	0	0	0	1	1	0	1	0	1	0	1	1	0	0
myguy?	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Dating Example (cont.)



```
SQL> select guyno
            , name
     from men
     where ((state = 'MA' and baseball = 'Red Sox') or
            (state = 'IL' and baseball = 'Cubs'    )
           )
       and voted4  = 'Bush'
       and mstatus = 'D'
       and degree  = 'Y'
     ;

     GUYNO NAME
---------- ------------------------------
         3 Larry
         5 Steve

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'MEN'
   2    1     BITMAP CONVERSION (TO ROWIDS)
   3    2       BITMAP INDEX (SINGLE VALUE) OF 'MEN_MSTATUS'

Statistics
----------------------------------------------------------
          4  consistent gets
```

Dating Example (cont.)



state='MA'

48,781

baseball='Sox'

37,978

18,243

8,986

state='IL'

20,871

baseball='Cubs'

15,253

27,329

voted4='Bush'

485,761

9,871

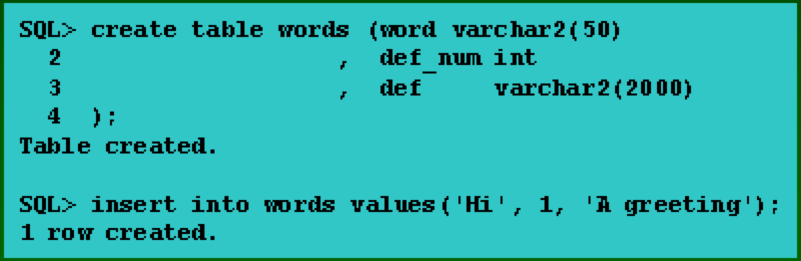
mstatus='D'

287,764

2,431!

Now it’s your turn…

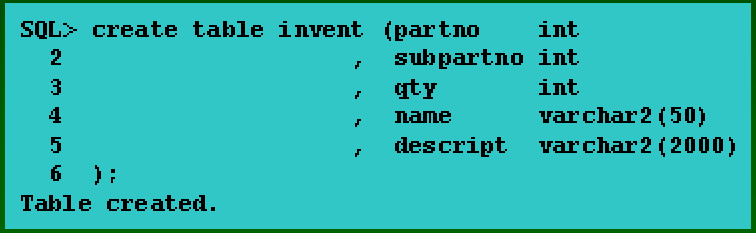
Dictionary



```
SQL> create table words (word varchar2(50)
  2                   ,  def_num int
  3                   ,  def     varchar2(2000)
  4  );
Table created.

SQL> insert into words values('Hi', 1, 'A greeting');
1 row created.
```

- Large number of words
- Query mostly

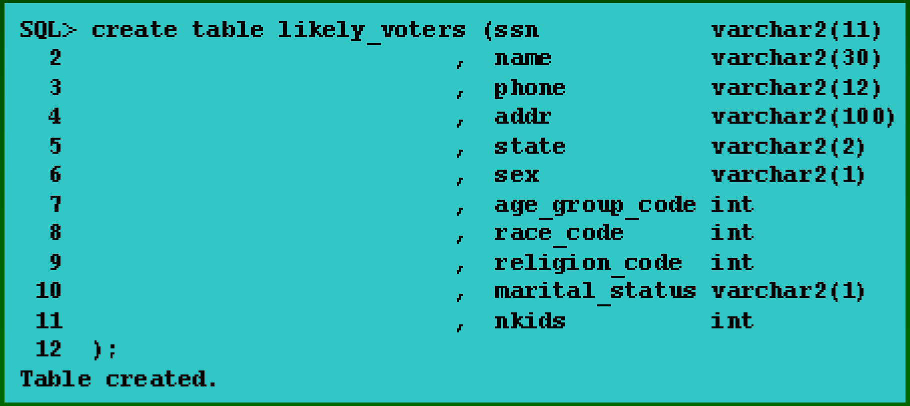
Inventory Table



```
SQL> create table invent (partno    int
  2                      , subpartno int
  3                      , qty       int
  4                      , name      varchar2(50)
  5                      , descript  varchar2(2000)
  6  );
Table created.
```

- Moderate insert of new parts
- subpartno affinity
- qty updated/queried frequently
- descript queried rarely

Likely Voter Table



```
SQL> create table likely_voters (ssn             varchar2(11)
  2                             , name           varchar2(30)
  3                             , phone          varchar2(12)
  4                             , addr           varchar2(100)
  5                             , state          varchar2(2)
  6                             , sex            varchar2(1)
  7                             , age_group_code int
  8                             , race_code      int
  9                             , religion_code  int
 10                             , marital_status varchar2(1)
 11                             , nkids          int
 12  );
Table created.
```

- Phone# or count(*) based on unforeseen criteria
- Reloaded from scratch weekly

Thank you