Physical Data Storage for the Application Developer

Dave Abercrombie, Convio
dabercrombie@convio.com
Northern California Oracle Users Group
Fall Conference, November 13 2009
at Oracle Corporation, Redwood Shores, CA

Application service provider (ASP), aka Software as a service (SaaS):

- Mostly non-profit clients, membership organizations.
- Online marketing, fundraising, events, advocacy, CMS, donor management.
- Distributed denial of service (DDOS) against ourselves! ☺

convio®
MOVE PEOPLE™

NOT talking about

Tablespaces, data files, extents, schema owners

Disks, spindles, RAID, SAN, HBA, filesystem

Block size, min_extents, max_extents

pct_free, pct_used, init_trans, max_trans, ITL

LMT, ASSM, freelists, high water marks

chaining, migration

Bitmap, reverse-key, and cluster indexes

Temp tablespace, global temporary tables

WILL talk about

Blocks:

- What’s in them (data!),
- Reading them
- Optimizer estimates

Measurement:

- Execution plans
- Performance statistics

Blocks

All data are within blocks

Must be in memory (block buffer cache)

Overhead for physical reading

Overhead for logical reading

Logical read overhead (forget all this!)

Cache buffer chain (CBC): is block in cache?

- Hash its file# and block# to determine CBC bucket
- Get the latch for the hashed CBC bucket
- Search the buffer header linked list for the buffer's "DBA"
- Check the SCN of the buffer to see if it is "current" (updates)
  - If not, keep searching for correct "clone",
  - If no suitable clone found, then make one
- Release the latch, have fun with the data.

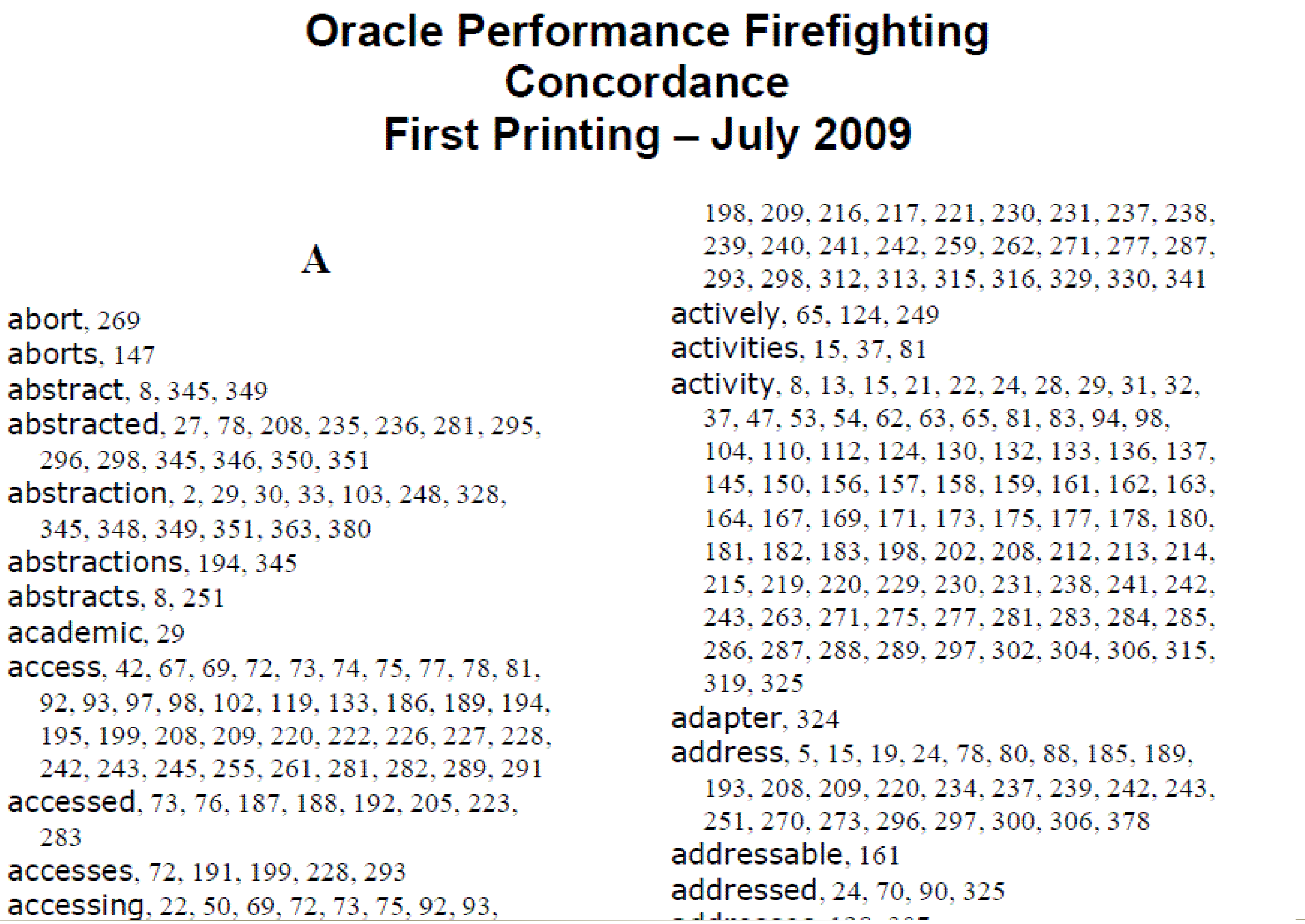
See Craig Shallahamer's "Oracle Performance Firefighting", OraPub 2009

Blocks

Table

Index

Example index



Oracle Performance Firefighting
Concordance
First Printing – July 2009

A

abort, 269
aborts, 147
abstract, 8, 345, 349
abstracted, 27, 78, 208, 235, 236, 281, 295, 296, 298, 345, 346, 350, 351
abstraction, 2, 29, 30, 33, 103, 248, 328, 345, 348, 349, 351, 363, 380
abstractions, 194, 345
abstracts, 8, 251
academic, 29
access, 42, 67, 69, 72, 73, 74, 75, 77, 78, 81, 92, 93, 97, 98, 102, 119, 133, 186, 189, 194, 195, 199, 208, 209, 220, 222, 226, 227, 228, 242, 243, 245, 255, 261, 281, 282, 289, 291
accessed, 73, 76, 187, 188, 192, 205, 223, 283
accesses, 72, 191, 199, 228, 293
accessing, 22, 50, 69, 72, 73, 75, 92, 93,

198, 209, 216, 217, 221, 230, 231, 237, 238, 239, 240, 241, 242, 259, 262, 271, 277, 287, 293, 298, 312, 313, 315, 316, 329, 330, 341
actively, 65, 124, 249
activities, 15, 37, 81
activity, 8, 13, 15, 21, 22, 24, 28, 29, 31, 32, 37, 47, 53, 54, 62, 63, 65, 81, 83, 94, 98, 104, 110, 112, 124, 130, 132, 133, 136, 137, 145, 150, 156, 157, 158, 159, 161, 162, 163, 164, 167, 169, 171, 173, 175, 177, 178, 180, 181, 182, 183, 198, 202, 208, 212, 213, 214, 215, 219, 220, 229, 230, 231, 238, 241, 242, 243, 263, 271, 275, 277, 281, 283, 284, 285, 286, 287, 288, 289, 297, 302, 304, 306, 315, 319, 325
adapter, 324
address, 5, 15, 19, 24, 78, 80, 88, 185, 189, 193, 208, 209, 220, 234, 237, 239, 242, 243, 251, 270, 273, 296, 297, 300, 306, 378
addressable, 161
addressed, 24, 70, 90, 325

Book index

Compact

Easy to search – sorted

Page number pointers

Includes portions of book content

Not always present

Sometimes more than one (subjects, people, etc.)

Cost to produce and maintain

Oracle index

Compact

Easy to search – sorted

ROWID pointers

Includes portions of table content

Not always present

Sometimes more than one (col_a, col_b, etc.)

Cost to produce and maintain

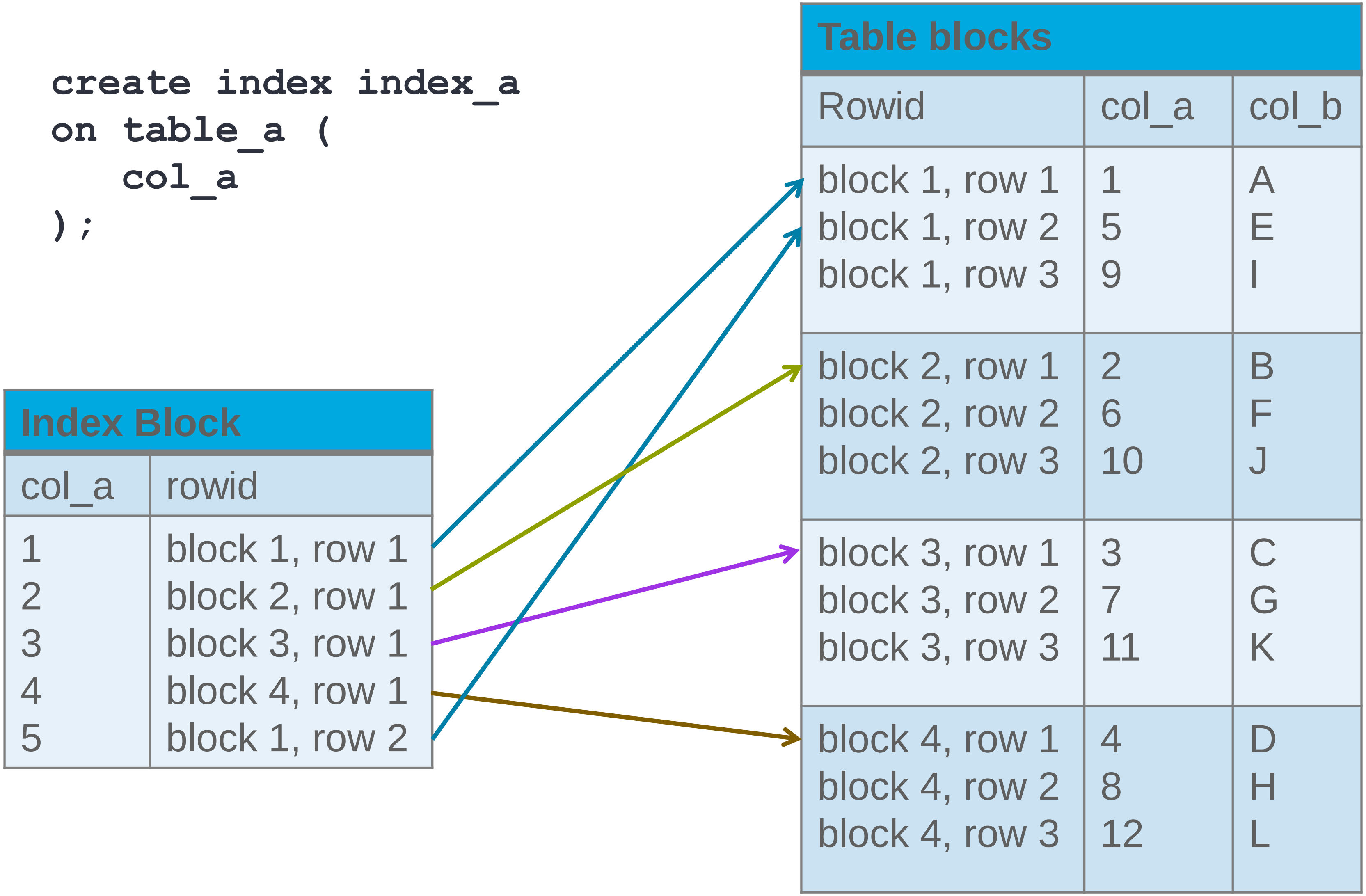
Example Data

```
SQL> select col_a, col_b from table_a order by col_a;

COL_A      COL_B
----- ----------
A              1
B              2
C              3
D              4
E              5
F              6
G              7
H              8
I              9
J             10
K             11
L             12
```

Index entries: data values & ROWID

```
create index index_a
on table_a (
   col_a
);
```



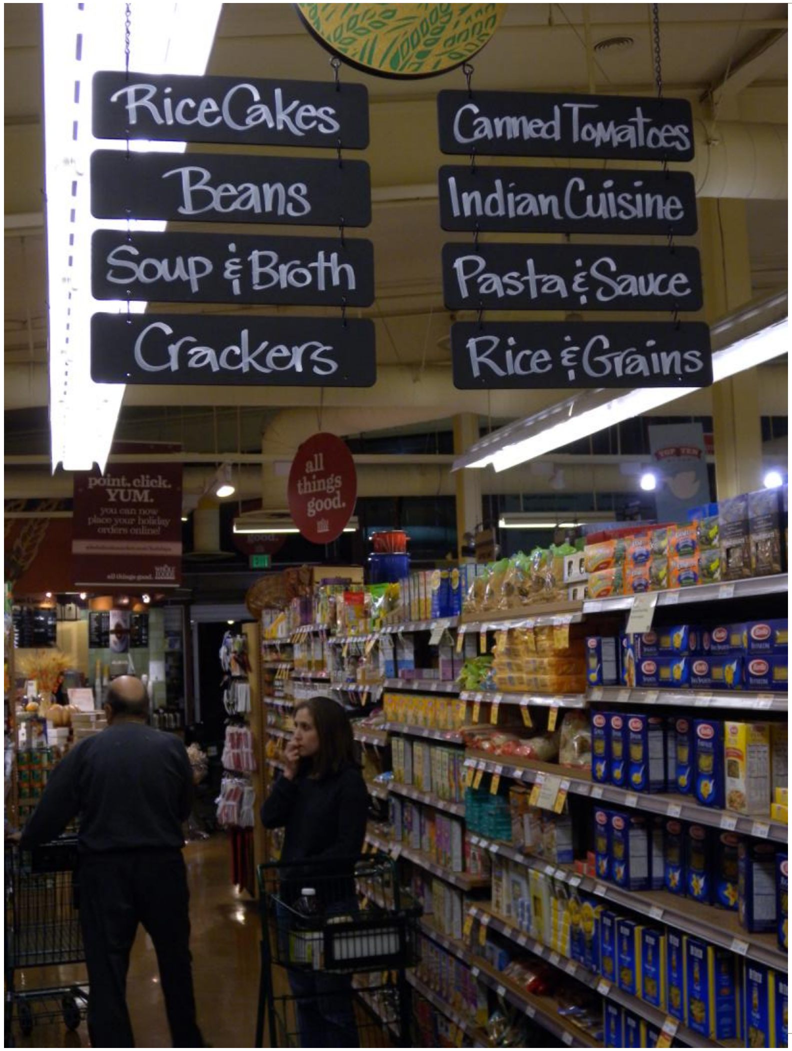
Index Block

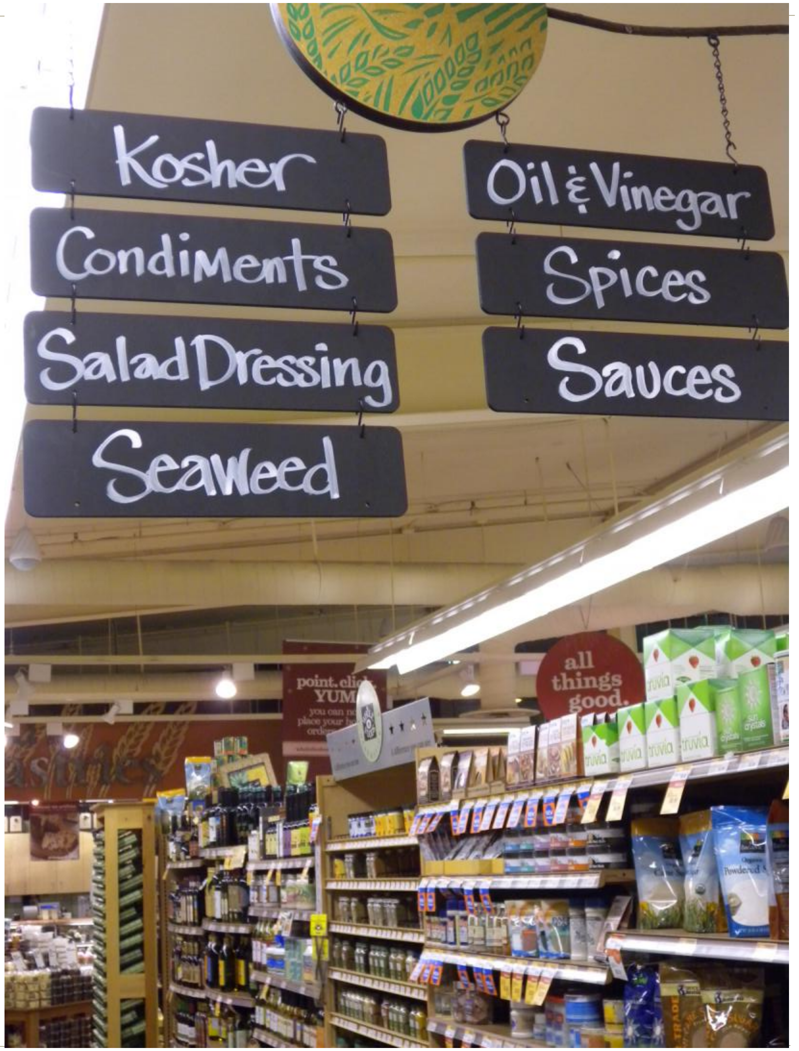
col_a	rowid
1	block 1, row 1
2	block 2, row 1
3	block 3, row 1
4	block 4, row 1
5	block 1, row 2

Table blocks

Rowid	col_a	col_b
block 1, row 1	1	A
block 1, row 2	5	E
block 1, row 3	9	I
block 2, row 1	2	B
block 2, row 2	6	F
block 2, row 3	10	J
block 3, row 1	3	C
block 3, row 2	7	G
block 3, row 3	11	K
block 4, row 1	4	D
block 4, row 2	8	H
block 4, row 3	12	L

Where is the pasta? … the bread?





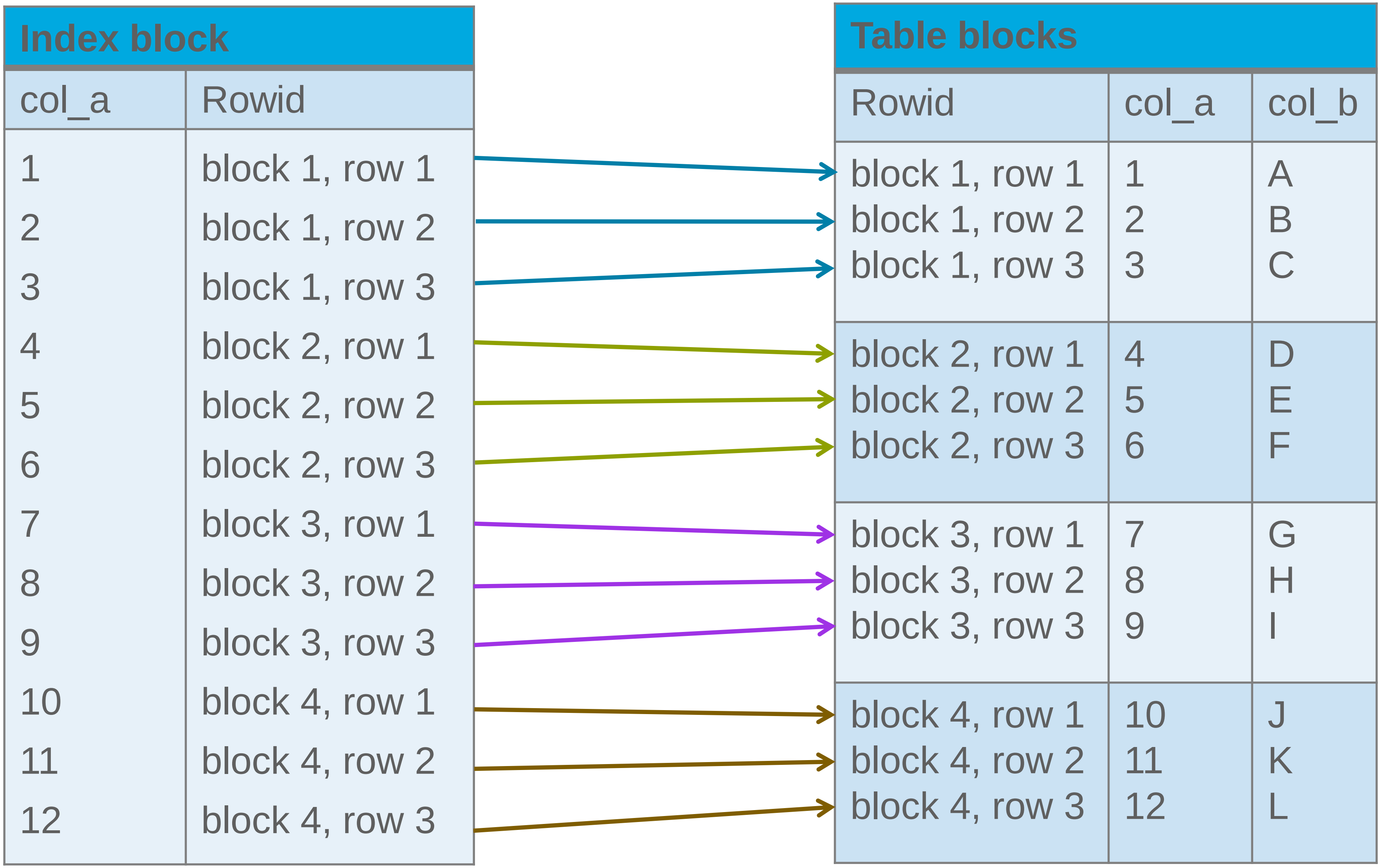
Grocery shopping with a list

Efficient:
aisle layout **same order** as shopping list

Inefficient:
aisle layout **different order** from shopping list
(consider visiting every aisle)

Does not matter for one item

LOW clustering factor: 4 (block count)



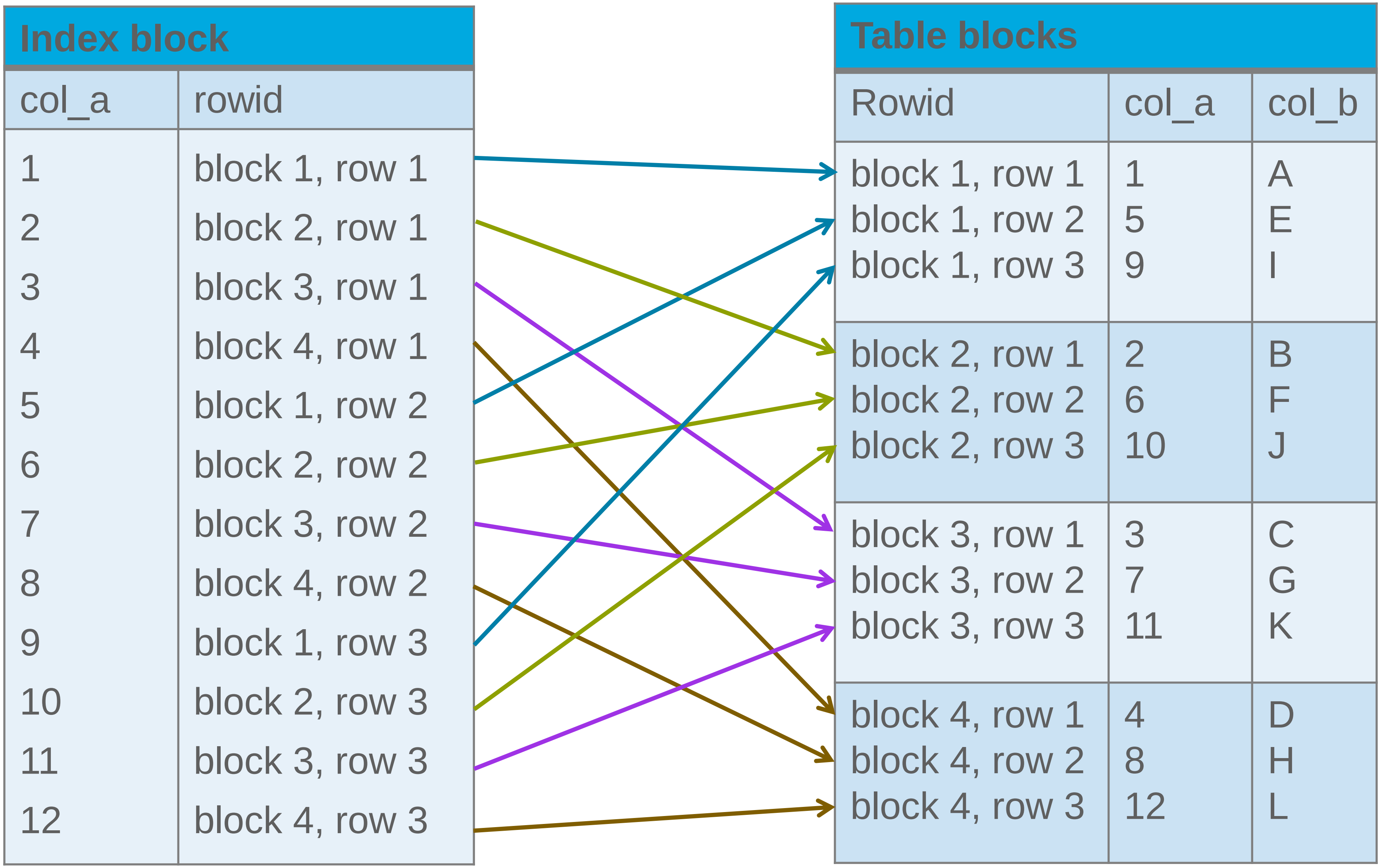
Index block

col_a	Rowid
1	block 1, row 1
2	block 1, row 2
3	block 1, row 3
4	block 2, row 1
5	block 2, row 2
6	block 2, row 3
7	block 3, row 1
8	block 3, row 2
9	block 3, row 3
10	block 4, row 1
11	block 4, row 2
12	block 4, row 3

Table blocks

Rowid	col_a	col_b
block 1, row 1 block 1, row 2 block 1, row 3	1 2 3	A B C
block 2, row 1 block 2, row 2 block 2, row 3	4 5 6	D E F
block 3, row 1 block 3, row 2 block 3, row 3	7 8 9	G H I
block 4, row 1 block 4, row 2 block 4, row 3	10 11 12	J K L

HIGH clustering factor: 12 (row count)



Index block

col_a	rowid
1	block 1, row 1
2	block 2, row 1
3	block 3, row 1
4	block 4, row 1
5	block 1, row 2
6	block 2, row 2
7	block 3, row 2
8	block 4, row 2
9	block 1, row 3
10	block 2, row 3
11	block 3, row 3
12	block 4, row 3

Table blocks

Rowid	col_a	col_b
block 1, row 1	1	A
block 1, row 2	5	E
block 1, row 3	9	I
block 2, row 1	2	B
block 2, row 2	6	F
block 2, row 3	10	J
block 3, row 1	3	C
block 3, row 2	7	G
block 3, row 3	11	K
block 4, row 1	4	D
block 4, row 2	8	H
block 4, row 3	12	L

Table access via an index (range)

Efficient (low clustering factor)
table layout **same order** as index

Inefficient (high clustering factor):
table layout **different order** from index
(consider visiting every block: “full scan”)

Does not matter for primary key lookup

Grocery shopping with a list

Efficient:
aisle layout **same order** as shopping list

Inefficient:
aisle layout **different order** from shopping list
(consider visiting every aisle)

Does not matter for one item

Store layout varies

Shopping list efficiency

Optimal list order depends on store layout

Varies from store to store

Varies over time

Index efficiency

Optimal index depends on physical data layout

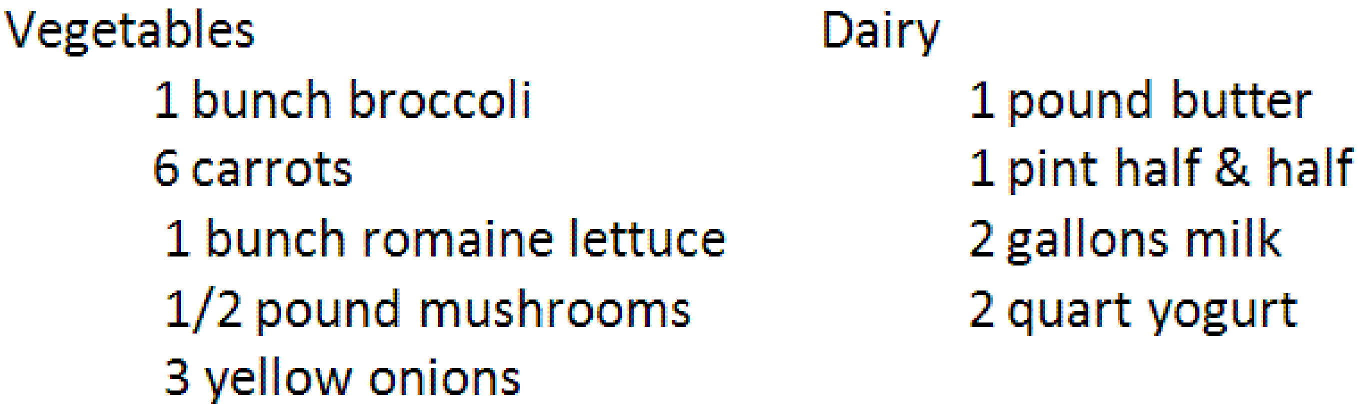
Performance may vary by schema

Performance may vary with time

Testing requires representative data

May depend on data loading technique

Which questions require a trip to store?



Vegetables

- 1 bunch broccoli
- 6 carrots
- 1 bunch romaine lettuce
- 1/2 pound mushrooms
- 3 yellow onions

Dairy

- 1 pound butter
- 1 pint half & half
- 2 gallons milk
- 2 quart yogurt

How many kinds of vegetables?

What is the total cost?

Any apples?

Which query requires table block reads?

Col_a (indexed)	Col_b
A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10

```
select count(col_a)
from table_a
where col_a < ‘F’
```

```
select count(col_b)
from table_a
where col_a < ‘F’
```

Same answer,
different efficiencies

Index-only queries

- Get answer directly from index:
  - No need to read table blocks
  - Ignores table data distribution
  - Pre-sorted
- Avoid select *
- Multi-column
- Index-organized tables (IOTs)
- Be careful with what you count

Index column order

Leading column required in SQL

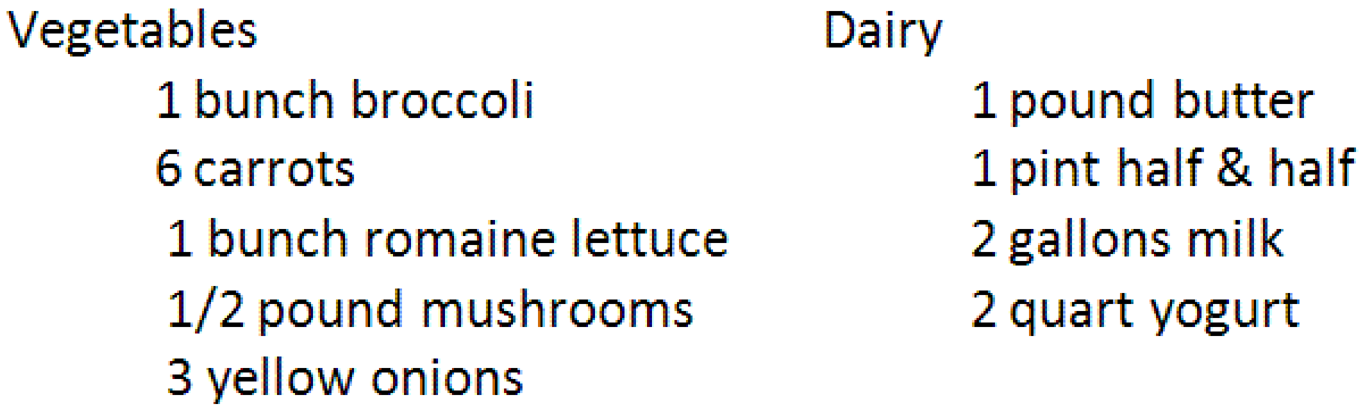
Order does <u>not</u> impact performance

Order <u>does impact</u> usability by other queries

Review all SQL – labor intensive

- Symptom: too many single-column indexes

How many trips to the store?



Vegetables

- 1 bunch broccoli
- 6 carrots
- 1 bunch romaine lettuce
- 1/2 pound mushrooms
- 3 yellow onions

Dairy

- 1 pound butter
- 1 pint half & half
- 2 gallons milk
- 2 quart yogurt

Buy everything in one trip?

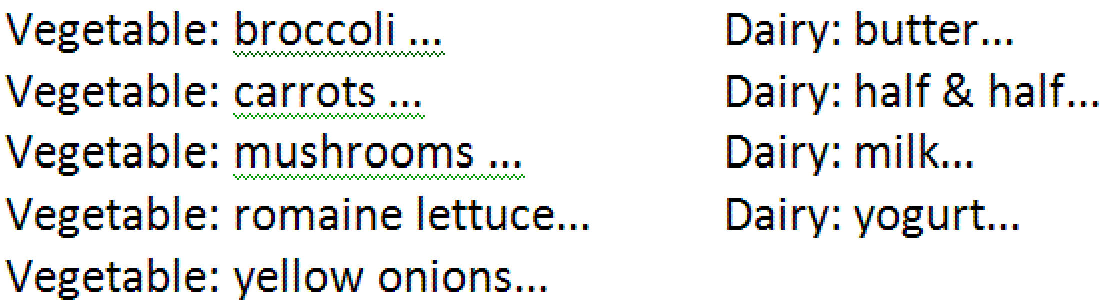
Get one item at a time, come home, unpack, go back to store for next item?

Row by row

Slow by slow

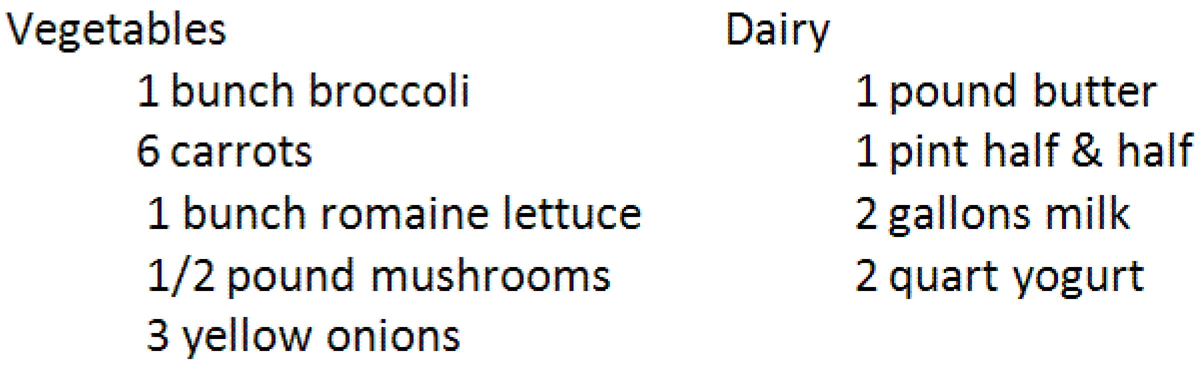
Single-row APIs

Redundancy



Vegetable: broccoli …
Vegetable: carrots …
Vegetable: mushrooms …
Vegetable: romaine lettuce…
Vegetable: yellow onions…

Dairy: butter…
Dairy: half & half…
Dairy: milk…
Dairy: yogurt…



Vegetables

- 1 bunch broccoli
- 6 carrots
- 1 bunch romaine lettuce
- 1/2 pound mushrooms
- 3 yellow onions

Dairy

- 1 pound butter
- 1 pint half & half
- 2 gallons milk
- 2 quart yogurt

Index compression & online

```
create index index_b on table_b (
    organization_id,
    member_id
) compress 1
online
;
```

Index wrap-up

Tom Kyte books:

- Expert One-on-One Oracle
- Effective Oracle by Design
- http://asktom.oracle.com

Vegetables close together

Partitioning

Groups together similar data

Improves density of relevant data in blocks

Improves usefulness of cache

Reduces physical reads

How many red M&Ms?

Estimate total count

Count colors

Assume even distribution

Divide total by colors

Optimizer math

USER_TABLES

- num_rows
- blocks

USER_TAB_COLUMNS

- num_distinct (density, num_buckets)

USER_INDEXES

- leaf_blocks
- clustering_factor

Jonathan Lewis: “Cost-Based Oracle Fundamentals” (rather advanced, unlike Kyte)

Execution plan – “Cardinality Feedback”

```
-------------------------------------------------------------------------------------------------
|Id | Operation                    | Name                | Starts | E-Rows | A-Rows |   A-Time   |
-------------------------------------------------------------------------------------------------
| 1 |  SORT ORDER BY               |                     |      1 |   1256 |  10387 |00:01:40.89 |
| 2 |   HASH JOIN SEMI             |                     |      1 |   1256 |  10387 |00:01:40.88 |
| 3 |    TABLE ACCESS BY INDEX ROWID| CONSTITUENT        |      1 |   1256 |    117K|00:01:40.47 |
| 4 |     INDEX RANGE SCAN         | ITOPS_BZ41319_CUS |      1 |    102 |    117K|00:00:00.73 |
| 5 |    INLIST ITERATOR           |                     |      1 |        |  24269 |00:00:00.05 |
| 6 |     INDEX RANGE SCAN         | GROUP_USER_INDEX  |      2 |  40875 |  24269 |00:00:00.02 |
-------------------------------------------------------------------------------------------------
```

Shows optimizer estimates (E-Rows)

Compare to actual (A-Rows): factor of ~100

Cardinality Feedback

Wolfgang Breitling

```
alter session set STATISTICS_LEVEL = ALL;
set serveroutput off
@your-query-here.sql
select * from table
     (dbms_xplan.DISPLAY_CURSOR(null, null, 'ALLSTATS'));
alter session set STATISTICS_LEVEL = TYPICAL;
```

Forget “explain plan”

Change SQL text to force re-parse between tests

Use DBMS_STATS (forget analyze table)

```
-- gather-table-stats.sql

BEGIN
DBMS_STATS.GATHER_TABLE_STATS (
   ownname          => USER,
   tabname          => upper('&table_name'),
   partname         => NULL,
   estimate_percent => NULL,                     -- let Oracle estimate
   block_sample     => FALSE,                    -- row sampling, accounts for skew
   method_opt       => 'FOR ALL COLUMNS SIZE 1',  -- no histograms
   degree           => NULL,
   granularity      => 'AUTO',
   cascade          => TRUE,                     -- get indexes too
   stattab          => NULL,
   statid           => NULL,
   statown          => NULL,
   no_invalidate    => FALSE,
   force            => FALSE
);
END;
/
show errors
```

V$mystat (forget autotrace)

Almost 400 stats, session level

“Cardinality feedback” adds cost, so does parsing

Manual snapshots (incremental data)

- consistent gets
- session logical reads
- db block changes
- redo entries
- redo size
- execute count
- table scan blocks gotten

Summary

Indexes – Oracle and books

Grocery Shopping and index efficiency

Multi-column indexes

Partitioning

Optimizer estimates

Cardinality Feedback

V$mystat