ORACLE COST-BASED OPTIMIZER BASICS

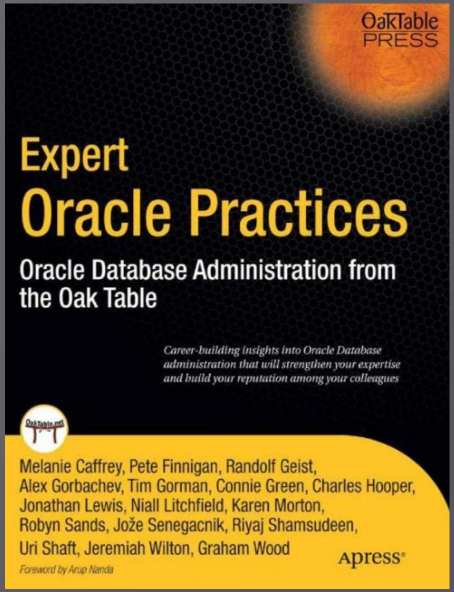
Randolf Geist

http://oracle-randolf.blogspot.com

info@sqltools-plusplus.org

ABOUT ME

- Independent consultant
  - Available for consulting
  - In-house workshops
    - Cost-Based Optimizer
    - Performance By Design
  - Performance Troubleshooting
- Oracle ACE Director
- Member of OakTable Network



OVERVIEW

- Optimizer Basics – Key Concepts
- Proactive: Performance by design
- Reactive: Troubleshooting

OPTIMIZER BASICS

- Three main questions you should ask when looking for an efficient execution plan:
  - How much data? How many rows / volume?
  - How scattered / clustered is the data?
  - Caching?

=> Know your data!

OPTIMIZER BASICS

- Why are these questions so important?
  - Two main strategies:
    - One “Big Job”
      => How much data, volume?
    - Few/many “Small Jobs”
      => How many times / rows?
      => Effort per iteration? Clustering / Caching

OPTIMIZER BASICS

- Optimizer's cost estimate is based on:
  - How much data? How many rows / volume?
  - How scattered / clustered? (partially)
  - (Caching?) Not at all

HOW MANY ROWS?

- Single table cardinality
- Join cardinality
- Filter subquery / Aggregation cardinality

SINGLE TABLE CARDINALITY

- Selectivity of predicates applying to a single table

```
select
        *
from
        t1
      , t2
where
-- Filter predicates
        t1.attr1 = 1
and     t2.attr1 = 1
-- Filter predicates
-- can also be applied to
-- join predicates
and     t1.id > 0
and     t2.id > 0
-- Join predicates
and     t1.id = t2.id
```

SINGLE TABLE CARDINALITY

- Selectivity of predicates applying to a single table

```
select
            *
from
            t1
          , t2
where
-- Filter predicates
            t1.attr1 = 1
and         t2.attr1 = 1
-- Filter predicates
-- can also be applied to
-- join predicates
and         t1.id > 0
and         t2.id > 0
-- Join predicates
and         t1.id = t2.id
```

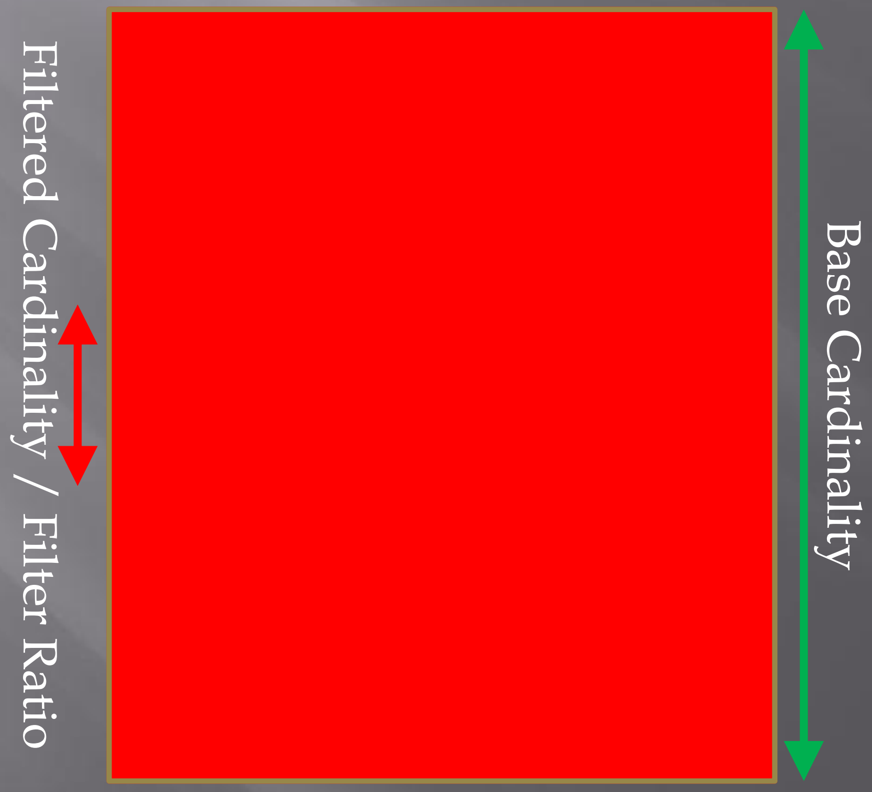
SINGLE TABLE CARDINALITY

- Selectivity of predicates applying to a single table

```
select
            *
from
            t1
          , t2
where
-- Filter predicates
            t1.attr1 = 1
and         t2.attr1 = 1
-- Filter predicates
-- can also be applied to
-- join predicates
and         t1.id > 0
and         t2.id > 0
-- Join predicates
and         t1.id = t2.id
```

SINGLE TABLE CARDINALITY

- Selectivity of predicates applying to a single table



SINGLE TABLE CARDINALITY

- Optimizer challenges
  - Skewed column value distribution
  - Gaps / clustered values
  - Correlated column values
  - Complex predicates and expressions
  - Bind variables

SINGLE TABLE CARDINALITY

Demo!

optimizer_basics_single_table_cardinality_testcase.sql

SINGLE TABLE CARDINALITY

- Impact NOT limited to a “single table”
- Influences the favored Single Table Access Path (Full Table Scan, Index Access etc.)
- Influences the Join Order and Join Methods (NESTED LOOP, HASH, MERGE)

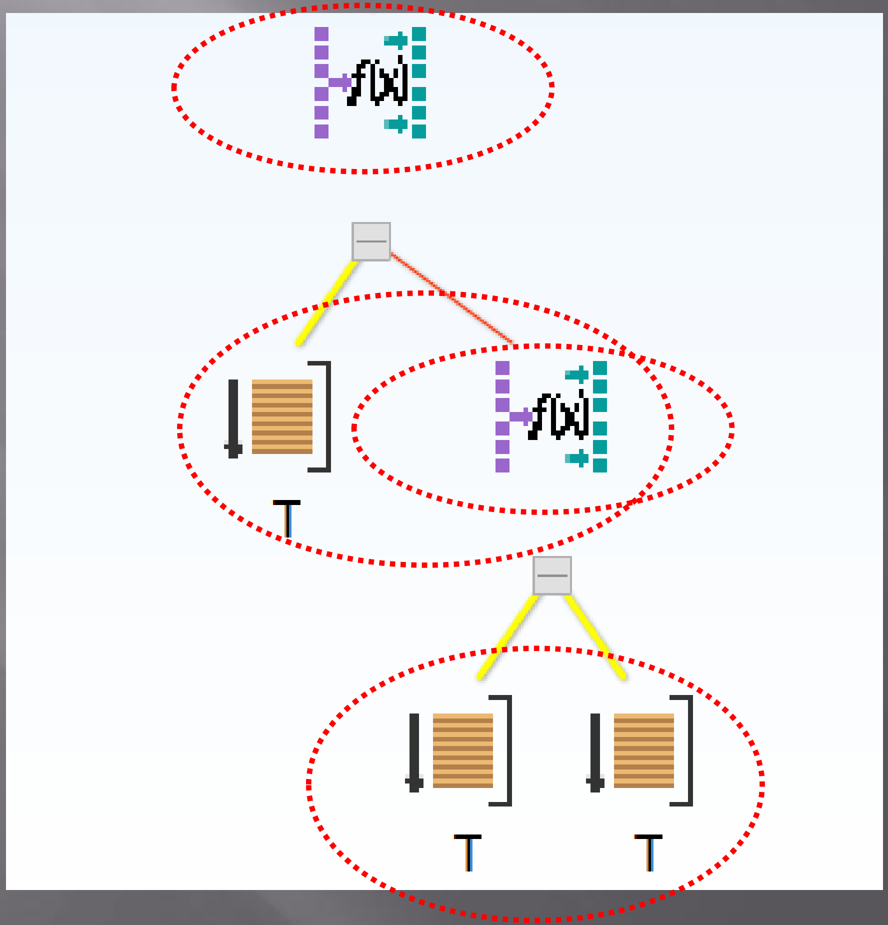
=> An incorrect single table cardinality potentially screws up whole execution plan!

JOIN CARDINALITY

- Oracle joins exactly two row sources at a time
- If more than two row sources need to be joined, multiple join operations are required
- Many different join orders possible (factorial!)

JOIN CARDINALITY

- Tree shape of execution plan

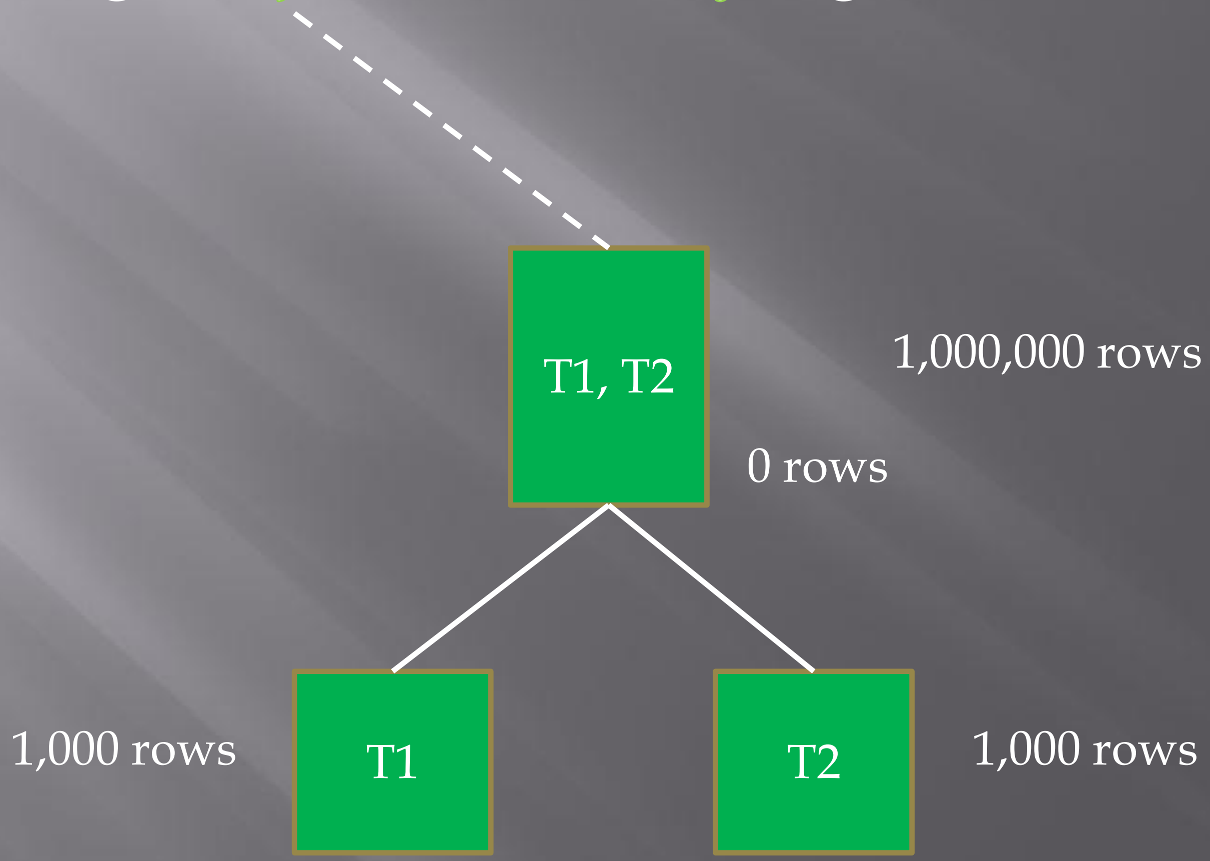


JOIN CARDINALITY

- Challenges
  - Getting the join selectivity right!
  - A join can mean anything between no rows and a Cartesian product

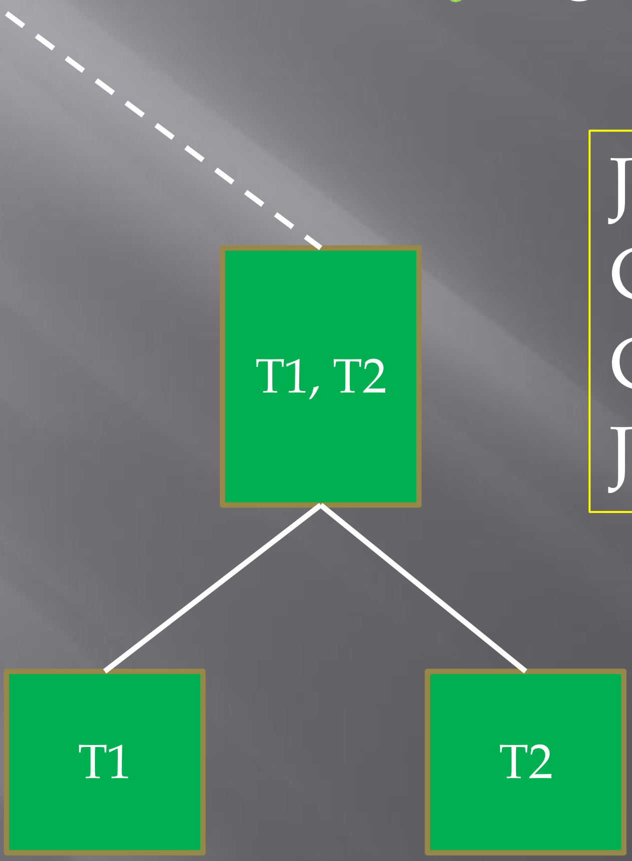
JOIN CARDINALITY

- Getting the join selectivity right



JOIN CARDINALITY

- Getting the join cardinality right



Join cardinality =
Cardinality T1 *
Cardinality T2 *
Join selectivity

JOIN CARDINALITY

- Challenges
  - Semi Joins (EXISTS (), = ANY())
  - Anti Joins (NOT EXISTS (), <> ALL())
  - Non-Equi Joins (Range, Unequal etc.)

JOIN CARDINALITY

- Even for the most common form of a join - the Equi-Join – there are several challenges
  - Non-uniform join column value distribution
  - Partially overlapping join columns
  - Correlated column values
  - Expressions
  - Complex join expressions (multiple AND, OR)

JOIN CARDINALITY

Demo!

optimizer_basics_join_cardinality_testcase.sql

JOIN CARDINALITY

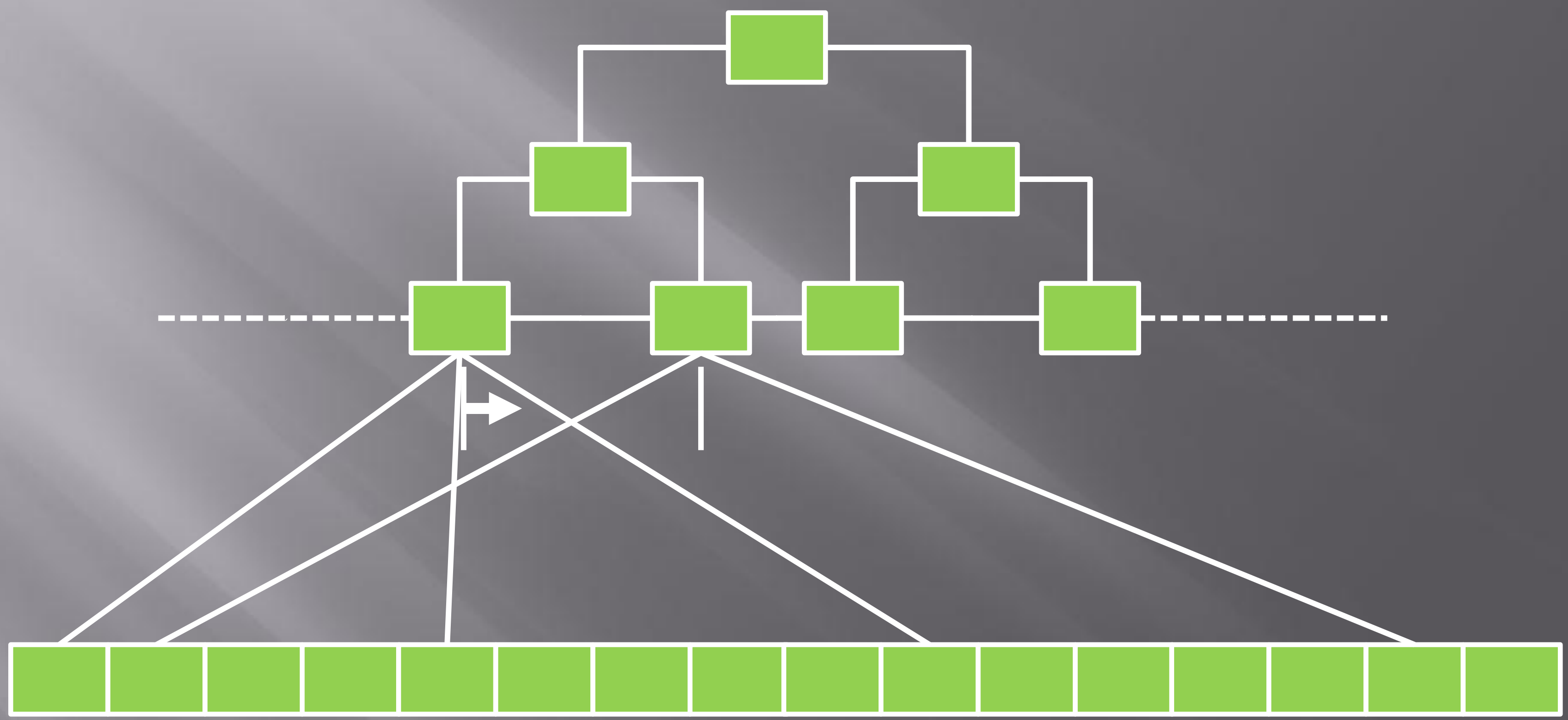
- Influences the Join Order and Join Methods (NESTED LOOP, HASH, MERGE)

=> An incorrect join cardinality/selectivity potentially screws up whole execution plan!

HOW SCATTERED / CLUSTERED?

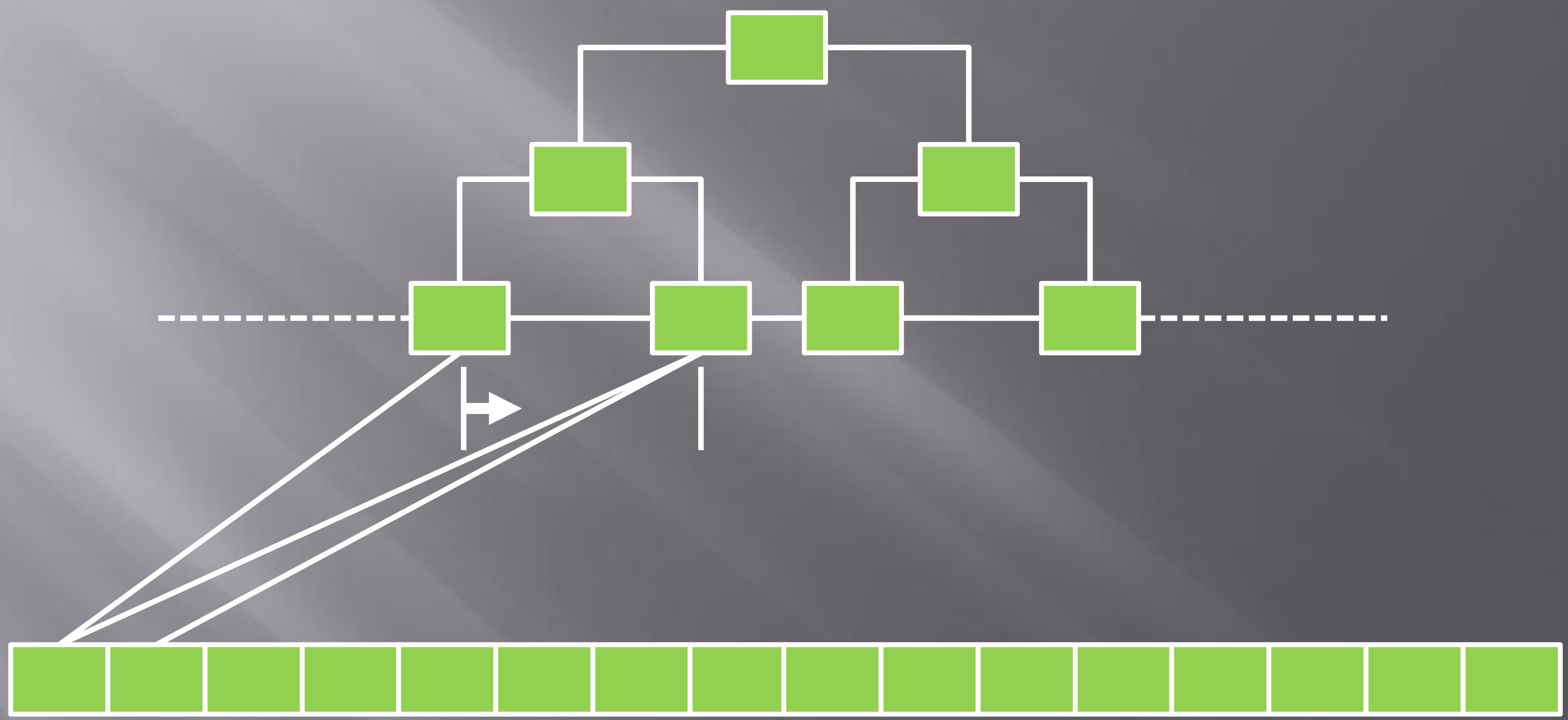
- Data is organized in blocks
- Many rows can fit into a single block
- According to a specific access pattern data can be either scattered across many different blocks or clustered in the same or few blocks
- Does make a tremendous difference in terms of efficiency of a “Small Job”

HOW SCATTERED / CLUSTERED?



1,000 rows => visit 1,000 table blocks: 1,000 * 5ms = 5 s

HOW SCATTERED / CLUSTERED?



1,000 rows => visit 10 table blocks: 10 * 5ms = 50 ms

HOW SCATTERED / CLUSTERED?

- Scattered data means potentially many more blocks to compete for the Buffer Cache for the same number of rows
- => Caching!

- Scattered data can result in increased
  - physical disk I/O
  - logical I/O
  - write disk I/O (Log Writer, DB Writer)
  - free buffer waits

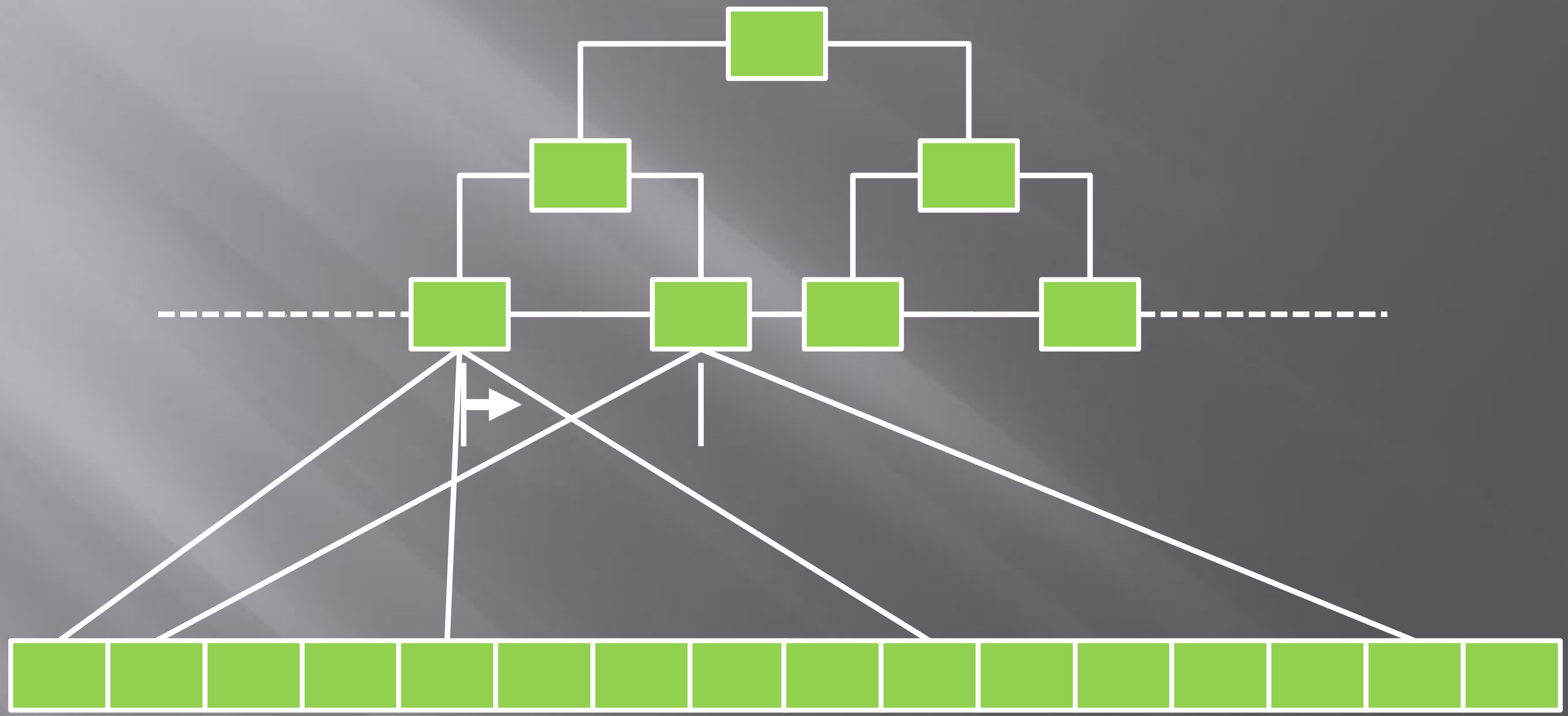
HOW SCATTERED / CLUSTERED?

- Most OLTP data has a natural clustering
- Data arriving around the same time is usually clustered together in a heap organized table
- Depends on the physical organization
- Partitioning for example can influence this clustering even for heap organized tables

HOW SCATTERED / CLUSTERED?

- Clustering of data can be influenced by physical implementation
- Physical design matters
  - Segment space management (MSSM / ASSM)
  - Partitioning
  - Index/Hash Cluster
  - Index Organized Tables (IOT)
  - Index design / multi-column composite indexes
- There is a reason why the Oracle internal data dictionary uses clusters all over the place

HOW SCATTERED / CLUSTERED?



No table access => only index blocks are visited!

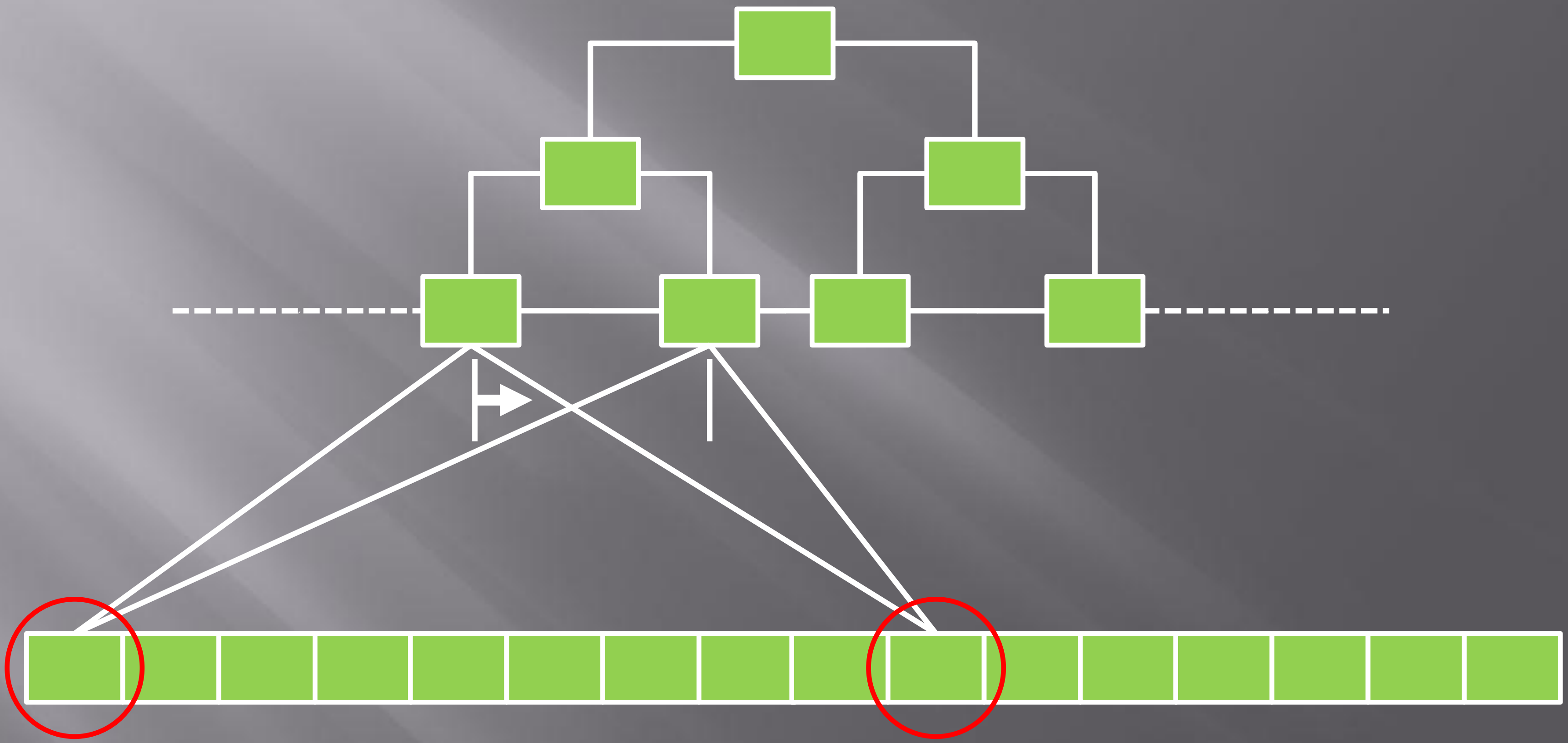
HOW SCATTERED / CLUSTERED?

- There is only a single measure of clustering in Oracle:
  The index clustering factor
- The index clustering factor is represented by a single value
- The logic measuring the clustering factor by default does not cater for data clustered across few blocks (ASSM!)

HOW SCATTERED / CLUSTERED?

- Challenges
  - Getting the index clustering factor right
  - There are various reasons why the index clustering factor measured by Oracle might not be representative
    - Multiple freelists / freelist groups (MSSM)
    - ASSM
    - Partitioning
    - SHRINK SPACE effects

HOW SCATTERED / CLUSTERED?

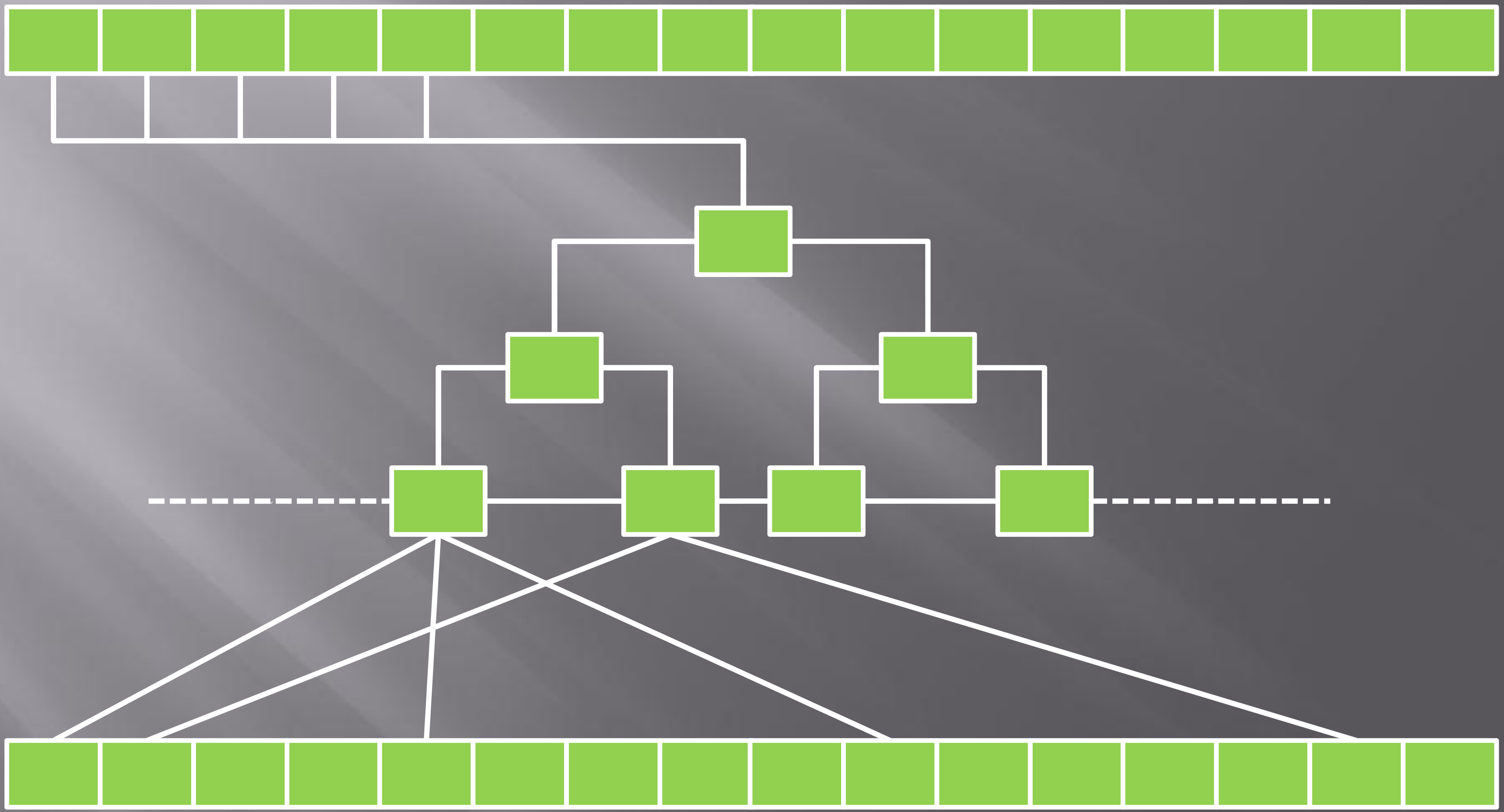


Re-visiting the same recent table blocks

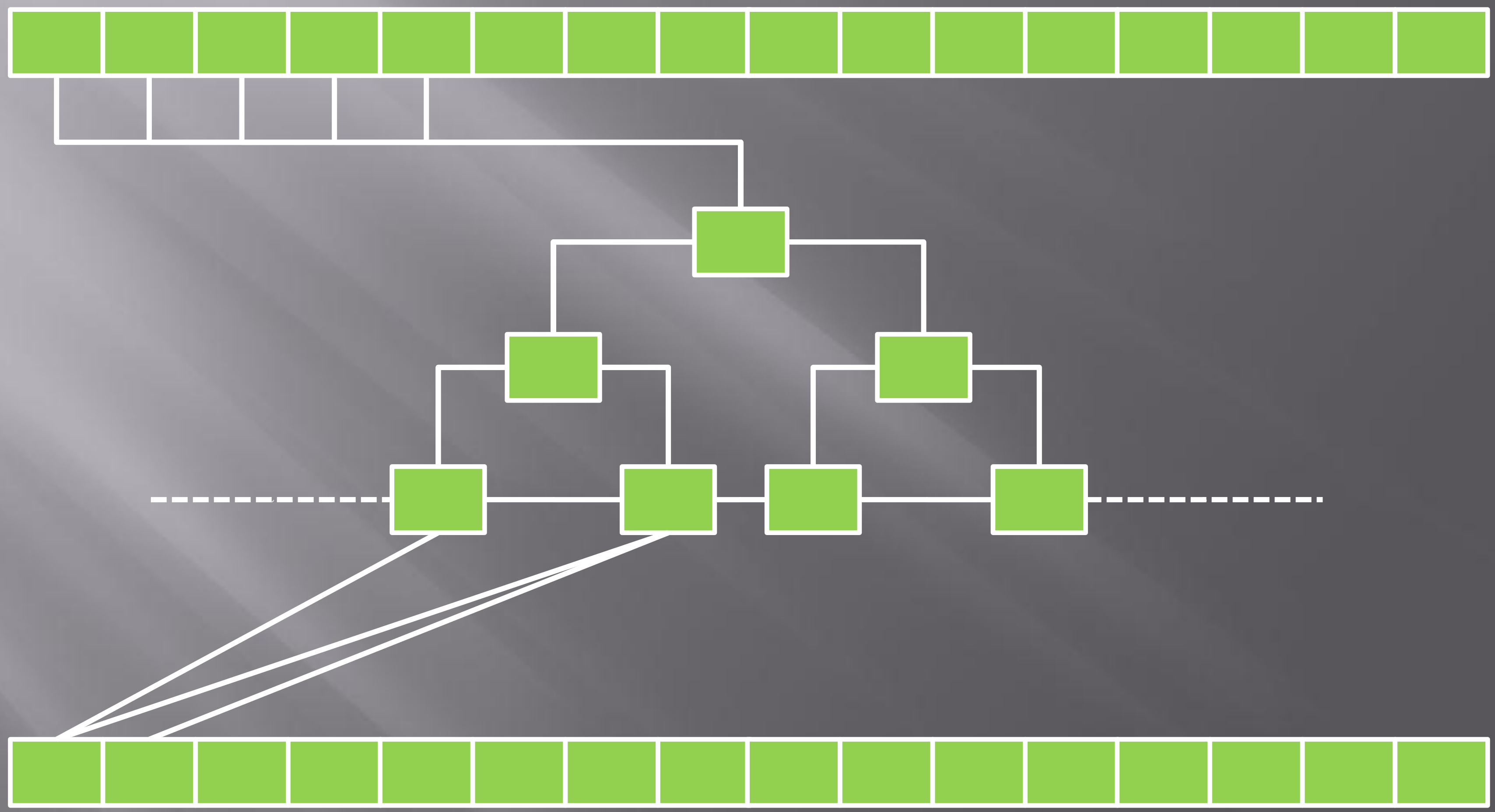
HOW SCATTERED / CLUSTERED?

- Challenges
  - There is no inter-table clustering measurement
  - The optimizer therefore doesn't really have a clue about the clustering of joins
  - You may need to influence the optimizer's decisions if you know about this clustering

HOW SCATTERED / CLUSTERED?



HOW SCATTERED / CLUSTERED?



HOW SCATTERED / CLUSTERED?

HOW SCATTERED / CLUSTERED?

Demo!

optimizer_basics_inter_table_clustering_testcase.sql

CACHING

- The optimizer's model by default doesn't consider caching of data
- Every I/O is assumed to be physical I/O
- But there is a huge difference between logical I/O (measured in microseconds) and physical I/O (measured in milliseconds)

CACHING

- You might have knowledge of particular application data that is "hot" and usually stays in the Buffer Cache

- Therefore certain queries against this "hot" data can be designed based on that knowledge

- The optimizer doesn't know about this. You may need to influence the optimizer's decisions

CACHING

- Oracle obviously played with the idea of introducing an per object caching component into the cost calculation in 9i and 10g

- You can see this from the undocumented parameters _optimizer_cache_stats and _cache_stats_monitor as well as the columns AVG_CACHED_BLOCKS and AVG_CACHE_HIT_RATIO in the data dictionary

CACHING

- It is important to point out that even logical I/O is not "free"
- So even by putting all objects entirely in the Buffer Cache inefficient execution plans may still lead to poor performance
- Excessive logical I/O, in particular on "hot blocks", can lead to latch contention and CPU starvation

SUMMARY

- Cardinality and Clustering determine whether the “Big Job” or “Small Job” strategy should be preferred
- If the optimizer gets these estimates right, the resulting execution plan will be efficient within the boundaries of the given access paths
- Know your data and business questions

PERFORMANCE BY DESIGN

- How to apply these concepts, where to go from here?
  - Read Jonathan Lewis' article "Designing Efficient SQL" at Red Gate's "Simple Talk"

Probably the best coverage of the concepts outlined here including clustering and caching

http://www.simple-talk.com/sql/performance/designing-efficient-sql-a-visual-approach/

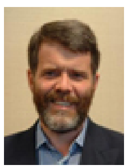
PERFORMANCE BY DESIGN

- Ho … from he…

Red

the

ing



Designing
Approach
25 February 20
by Jonathan Le

Remember that you can understand your application and your data better than the Optimizer; sometimes the Optimizer chooses a bad execution plan because it doesn't **know** the data, or how your application handles that data, as well as you do.

If a picture paints a thousand words...

...why not draw your query? If you've got a complex SQL statement with many tables, you have a real need to collect and present a lot of information in a way that can be easily grasped. Drawing a picture is a good idea, especially if you're trying to debug someone else's SQL.

My approach is simple:

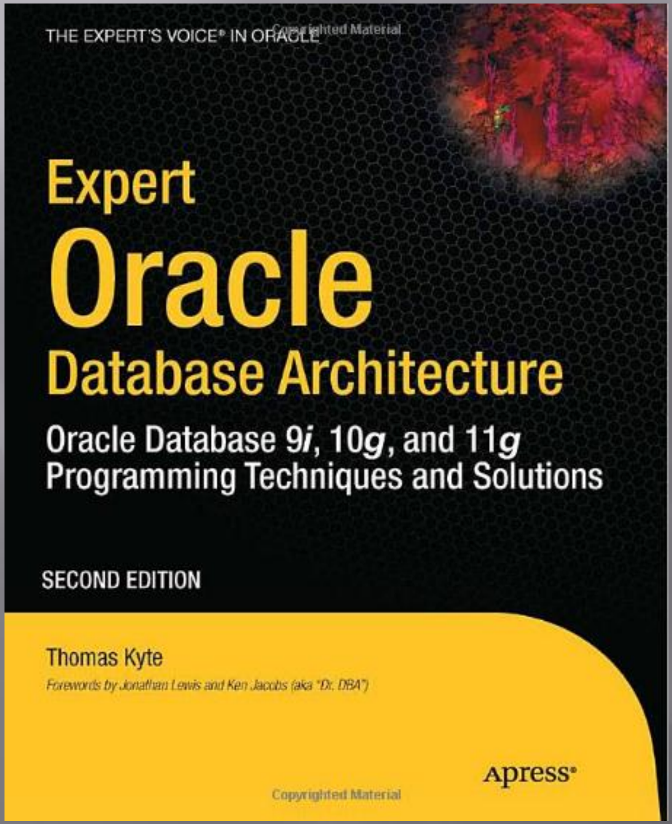
- Read through the SQL statement and draw a box for each table and a line between boxes for every join condition.
- If you are aware of the cardinality of the join (one-to-one, one-to-many, many-to-many), then put a "crow's foot" at the "many" end(s) of the line.
- If you have a filter predicate on a table, draw an arrow coming into the box and write the predicate by it.
- If a "table" is actually an in-line view, or sub-query including multiple tables, draw a light outline around the entire group of tables.

For example, say you have a schema which defines a fairly simple order-processing system: customers place orders, orders can have multiple order lines, an order line is for a product, and products come from suppliers; some products can be substituted by other products. One day you are asked to report on *"orders placed over the last week by customers based in London, for products supplied by producers based in Leeds that could have been supplied from an alternative location"*

http://www.simple-talk.com/sql/performance/designing-efficient-sql-a-visual-approach/

PERFORMANCE BY DESIGN

- How to apply these concepts, where to go from here?

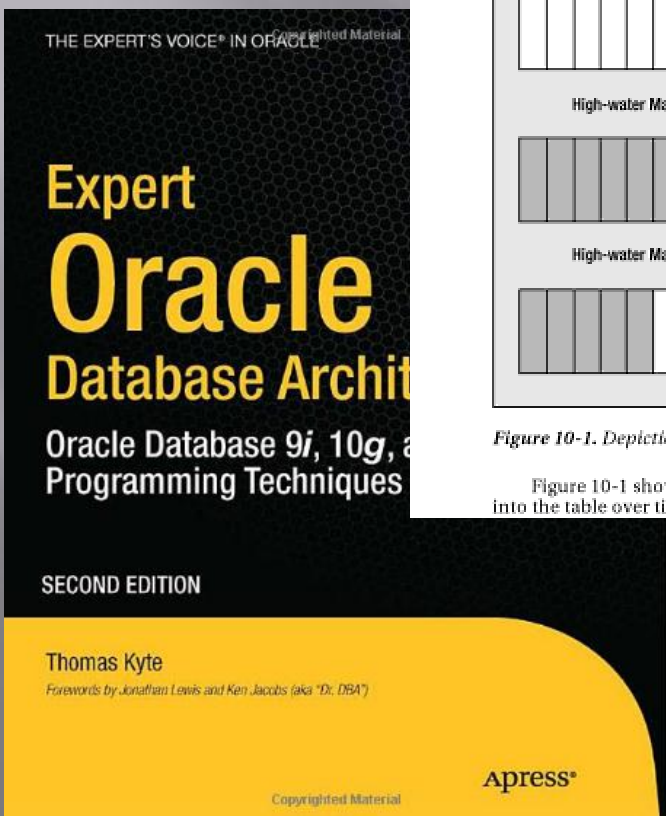


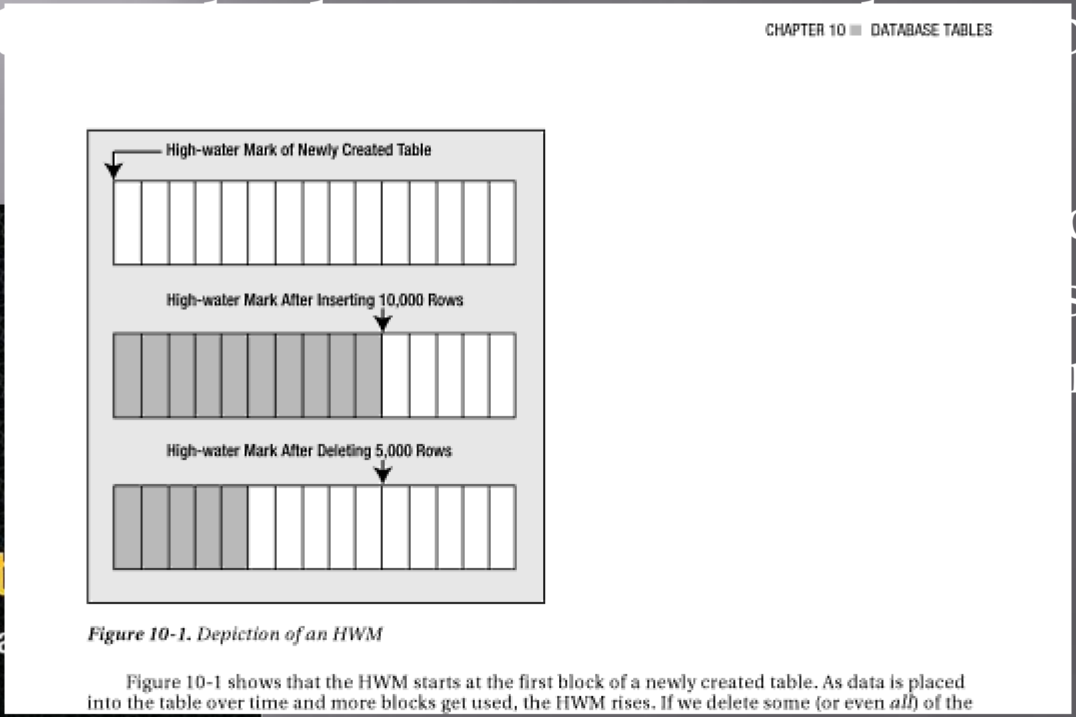
  - Read one of Tom Kyte's books to learn more about the pro's and con's of clusters and index organized tables

PERFORMANCE BY DESIGN

- How t ... o go from here?

ooks to
s and con's
nized





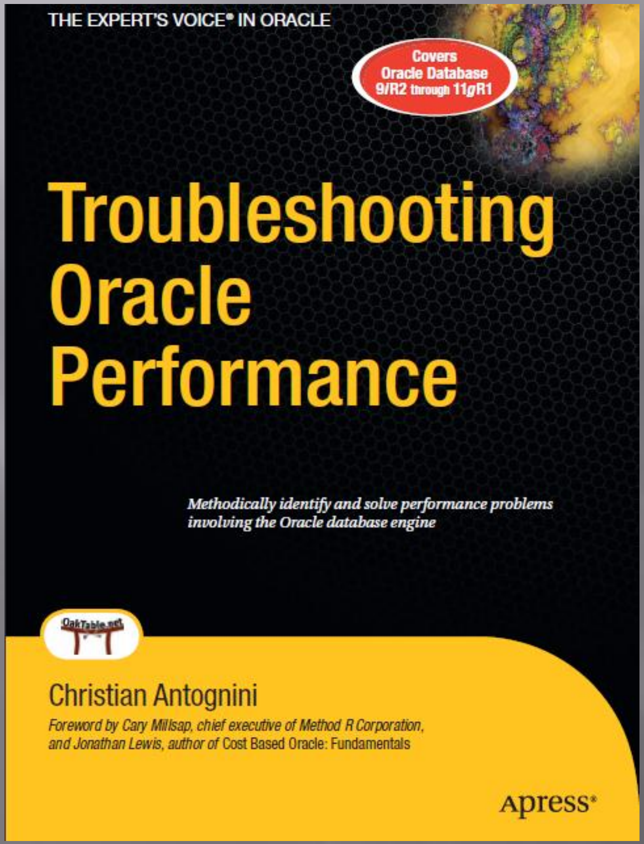
CHAPTER 10 ■ DATABASE TABLES

Figure 10-1. Depiction of an HWM

Figure 10-1 shows that the HWM starts at the first block of a newly created table. As data is placed into the table over time and more blocks get used, the HWM rises. If we delete some (or even *all*) of the

PERFORMANCE BY DESIGN

- How to apply these concepts, where to go from here?



  - Learn how to read, interpret and understand Oracle execution plans => Chapter 6 of "Troubleshooting Oracle Performance" by Christian Antognini

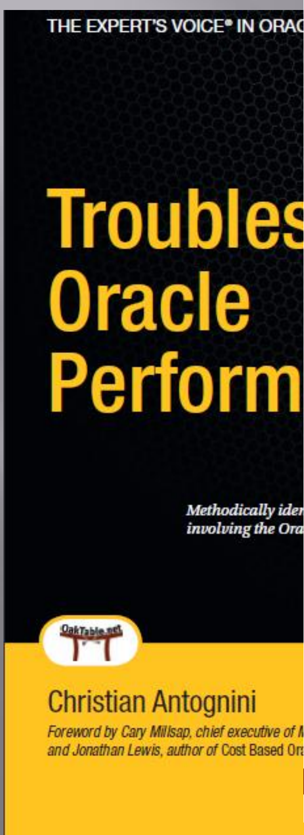
  - This knowledge is required in order to compare your understanding of the query to the optimizer's understanding

PE… …GN

- Ho… …from her…

…nd …plans …ting …stian

…order …ng of



- Operation 1 is the root of the tree. It has three children: 2, 5, and 8.
- Operation 2 has two children: 3 and 4.
- Operations 3 and 4 have no children.
- Operation 5 has one child: 6.
- Operation 6 has one child: 7.
- Operation 7 has no children.
- Operation 8 has one child: 9.
- Operation 9 has no children.

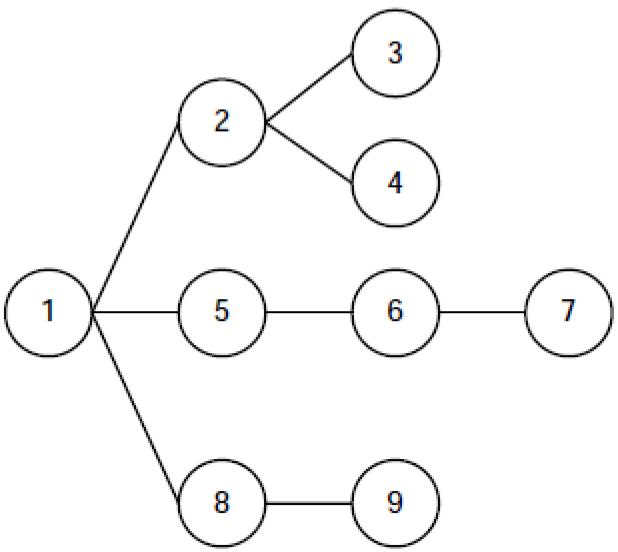


Figure 6-2. *Parent-child relationships between execution plan operations*

PERFORMANCE BY DESIGN

- How to apply these concepts, where to go from here?

```sql
select
        distinct
        p.prod_id
      , p.prod_name
from
        products p
      , sales s
where
        p.prod_id = s.prod_id
and     s.quantity_sold < (
          select
                  avg(quantity_sold)
          from
                  sales s_a
          where
                  s_a.prod_id = p.prod_id
        )
;
```

  - Be aware of Query Transformations: The optimizer might rewrite your query to something that is semantically equivalent but potentially more efficient
  - This might take you by surprise when trying to understand the execution plan favored by the optimizer

Query transformation examples by courtesy of Joze Senegacnik (OOW 2010)

PERFORMANCE BY DESIGN

- How to apply these concepts, where to go from here?

```
select
        distinct
        p.prod_id
      , p.prod_name
from
        products p
      , sales s
      , (
          select
                  avg(quantity_sold) as avg_qnt
                , prod_id
          from
                  sales s_a
          group by
                  prod_id
        ) v
where
        p.prod_id = s.prod_id
and     v.prod_id = p.prod_id
and     s.quantity_sold < v.avg_qnt
;
```

Be aware of Query Transformations: The optimizer might rewrite your query to something that is semantically equivalent but potentially more efficient

This might take you by surprise when trying to understand the execution plan favored by the optimizer

Query transformation examples by courtesy of Joze Senegacnik (OOW 2010)

PERFORMANCE BY DESIGN

- Ho … from
  he…

  …ations:
  …your

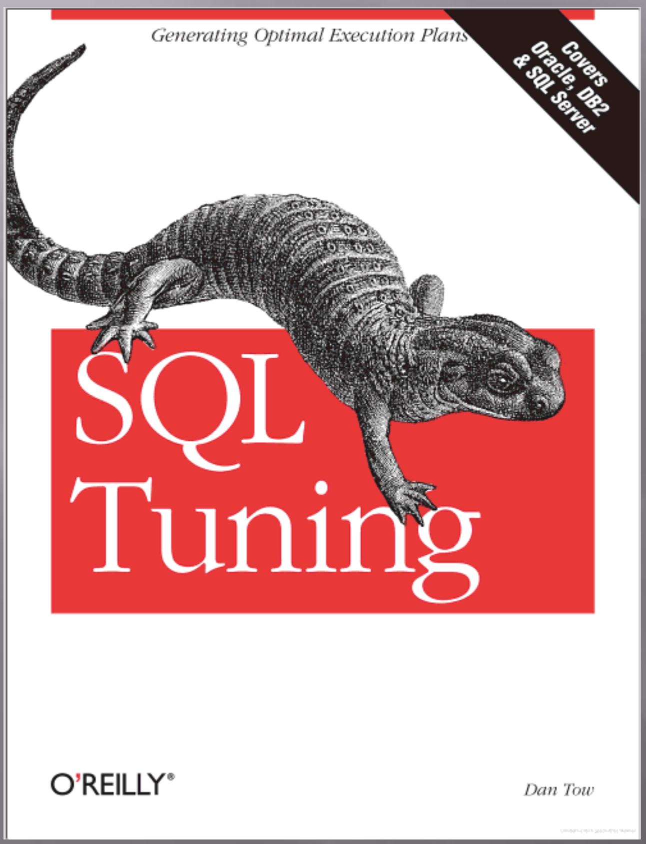
```
select
        distinct
        prod_id
      , prod_name
from
        (
          select
                  p.prod_id
                , p.prod_name
                , case
                  when s.quantity_sold < avg(s.quantity_sold) over (partition by s.prod_id)
                  then s.rowid
                  end as qnt_filter
          from
                  products p
                , sales s
          where
                  p.prod_id = s.prod_id
        )
where
        qnt_filter is not null
;
```

…se when trying to understand the execution plan favored by the optimizer

Query transformation examples by courtesy of Joze Senegacnik (OOW 2010)

PERFORMANCE BY DESIGN

- If you want a more formal approach



- Read "SQL Tuning" by Dan Tow
  - Teaches a formal approach how to design and visualize an execution plan
  - Focuses on "robust" execution plans in an OLTP environment
  - The formal approach doesn't take into account clustering and caching, however it is mentioned in the book at some places

PERFORMANCE BY DESIGN

- If

w
to
n plan
lans
e into
ook at

values), you find master join ratios equal to exactly 1.0, as in every case in this example, from A7/A6, A11/A6, A5/A4, and A9/A4.

Check for any unique filter conditions that you would annotate with an asterisk (Step 6). In the case of this example, there are no such conditions.

Then, place all of these numbers onto the query diagram, as shown in Figure 5-5.

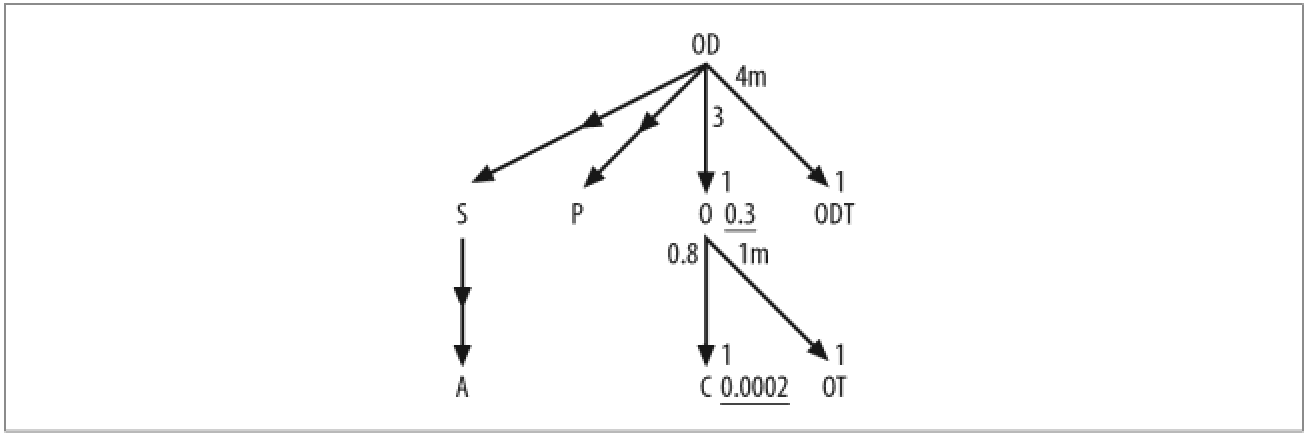
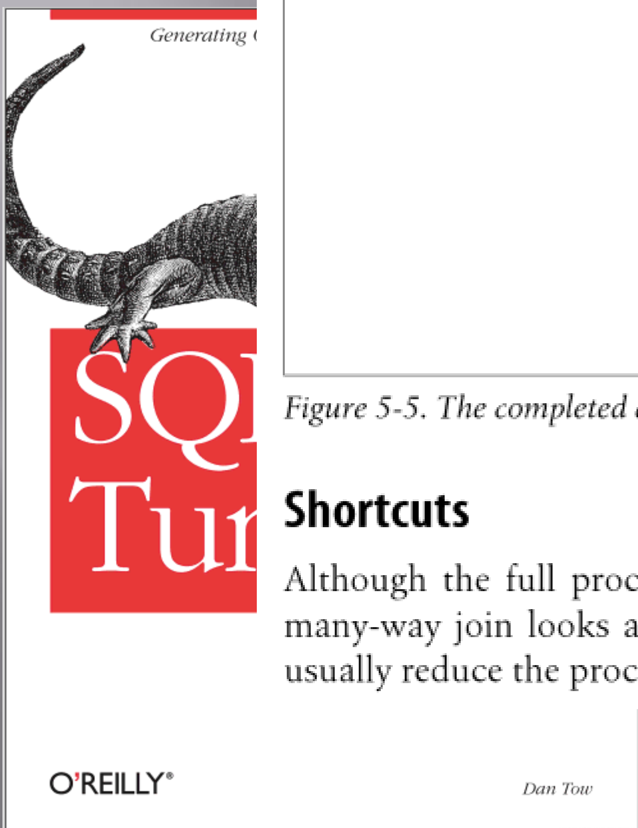


Figure 5-5. The completed query diagram for the second example

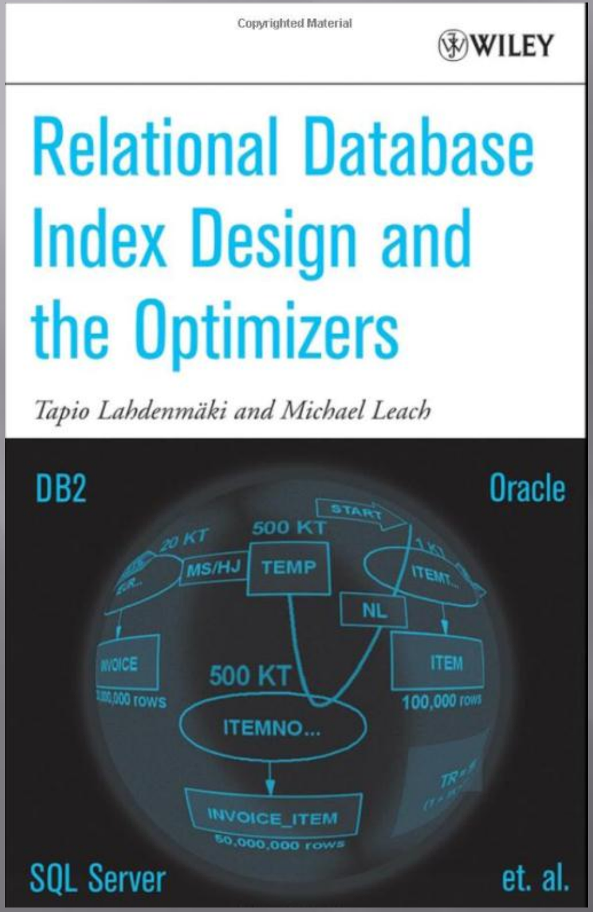
Shortcuts

Although the full process of completing a detailed, complete query diagram for a many-way join looks and is time-consuming, you can employ many shortcuts that usually reduce the process to a few minutes or even less:



PERFORMANCE BY DESIGN

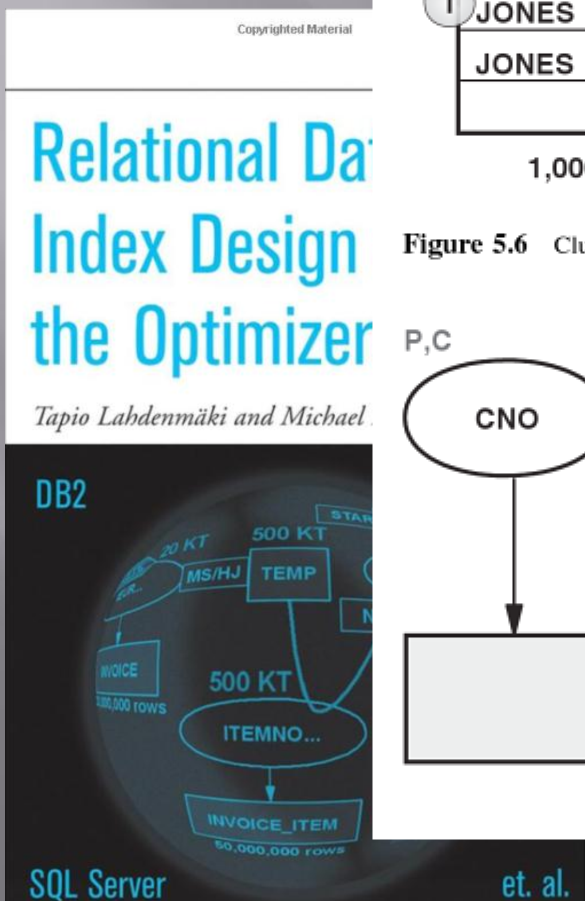
- If you want a more formal approach



- Read “Relational Database Index Design and the Optimizers” by Tapio Lahdenmäki and Michael Leach
  - Focuses on index design
  - Provides simple and more advanced formulas allowing to predict the efficiency of queries and indexes
  - Covers clustering and caching

PER … IGN

- If you … e Index … s” by … ichael … advanced … ct the … dexes … ing



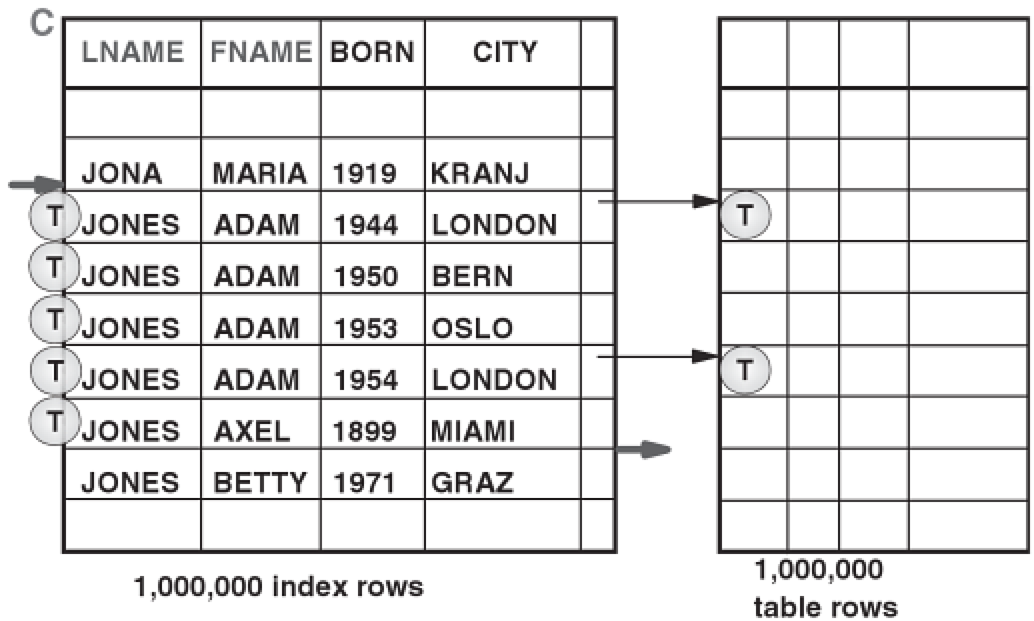


Figure 5.6 Clustering index with skip-sequential table access.

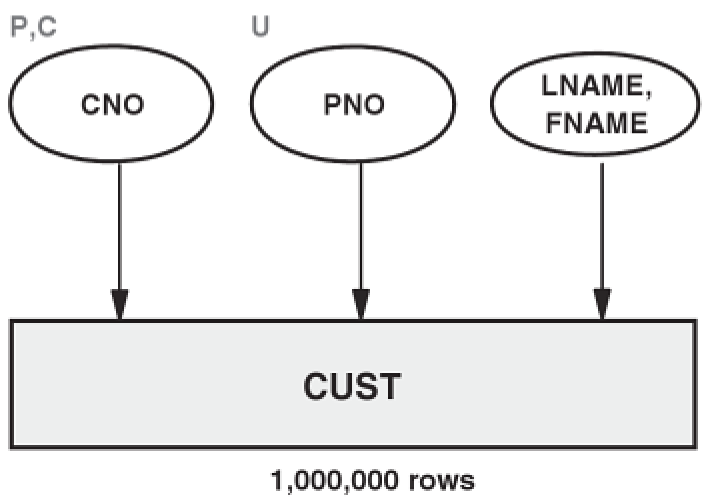


Figure 5.7 Cheapest adequate index or best possible index: Example 1A.

PERFORMANCE BY DESIGN

- For application developers
  - Read “Use the Index, Luke” by Markus Winand
    - Focuses on index design
    - Provides a lot of examples how to design efficient database access using different front-end languages (Java, Perl, PHP, etc.)
    - Also available as free eBook
    - Cross database (Oracle DB2, MySQL…)

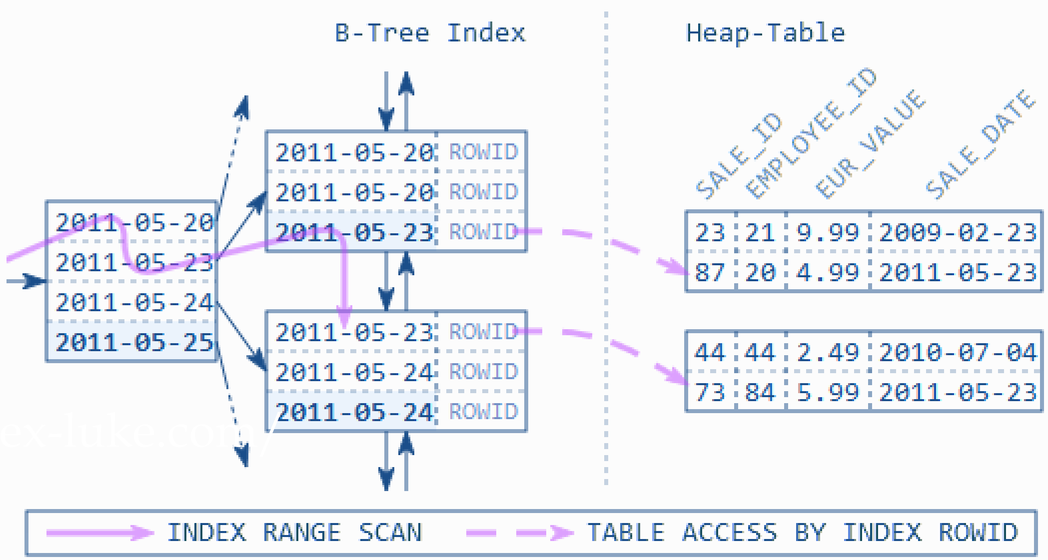
http://use-the-index-luke.com/

PER… IGN

- For ap… e” by … how to … ccess using … ges (Java, … k … 2,

The following figures show what it means to use an index on an index. For comparison, we will first look at an index access on a heap-table. The search shown in Figure 5.2 uses an index on the sale date to find all sales for 23rd May 2011. The execution involves two steps: (1) the `INDEX RANGE SCAN`; (2) the `TABLE ACCESS BY INDEX ROWID`.

Figure 5.2. Index-Based Access on a Heap-Table

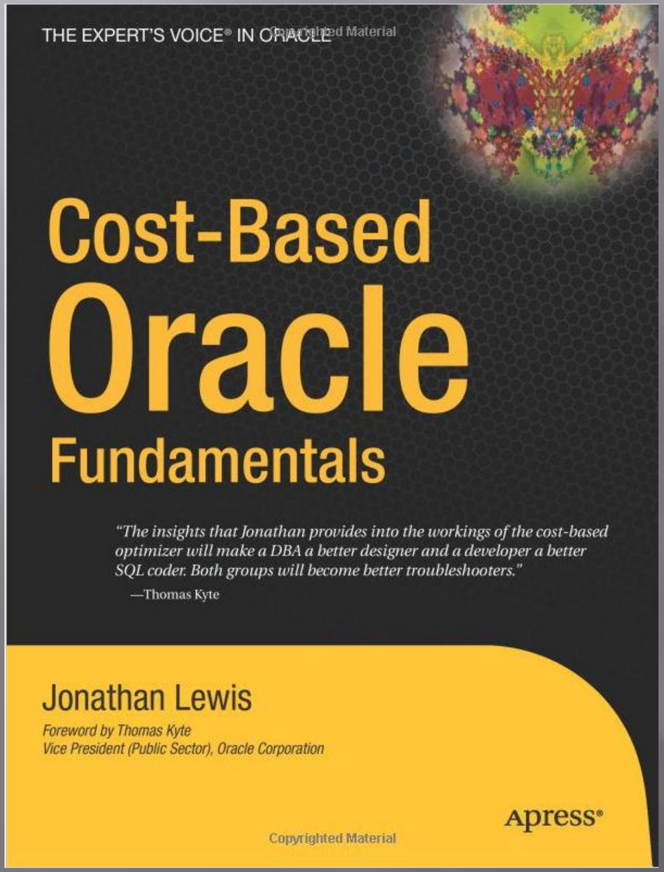


http://use-the-index-luke.com/

Although the table access might become the bottleneck, it is still limited to one access per row. That is because the index has the `ROWID`, which is a straight pointer to the row. Knowing the exact position allows the database to directly

PERFORMANCE BY DESIGN

- If you want dive into the details of the Cost-Based Optimizer



- Read "Cost-Based Oracle: Fundamentals" by Jonathan Lewis
  - Almost six years old
  - Still the best book about the Oracle optimizer
  - Covers the key concepts mentioned here in great detail

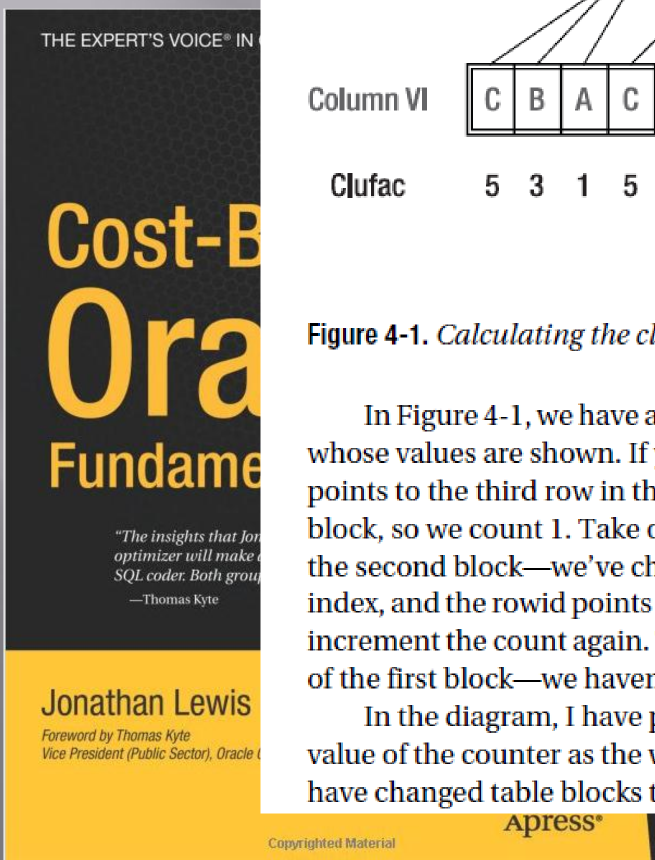
P GN

- If Cost-B

n Lewis

Oracle

ntioned



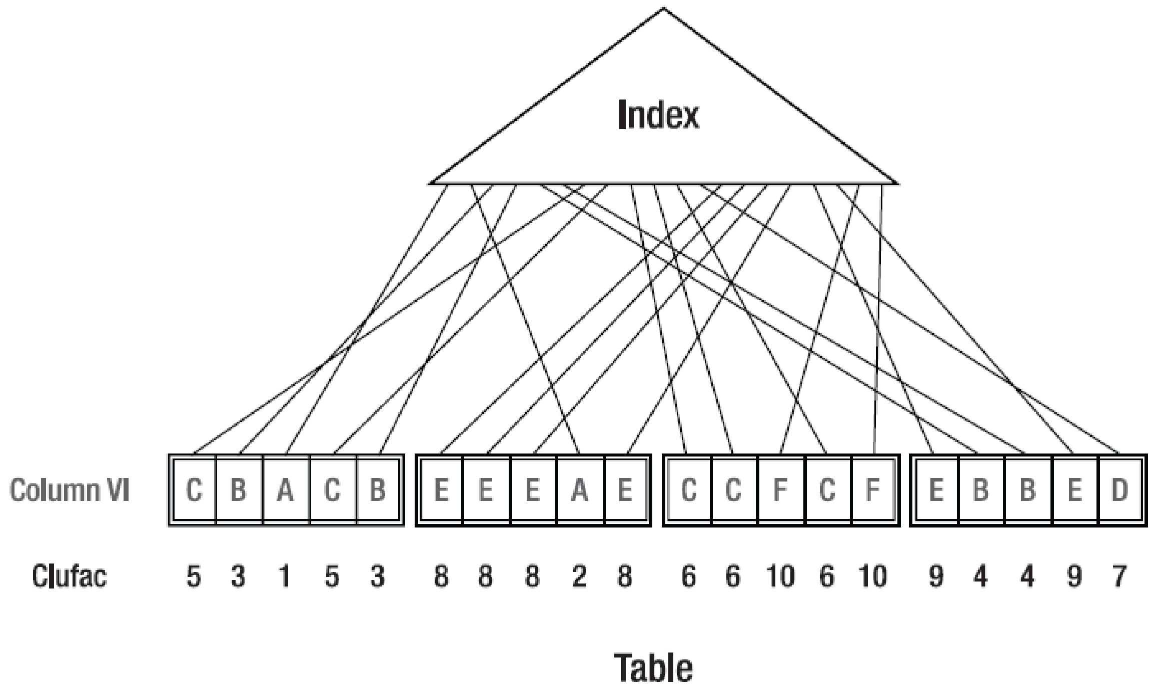


Figure 4-1. *Calculating the clustering_factor*

In Figure 4-1, we have a table with four blocks and 20 rows, and an index on the column `V1`, whose values are shown. If you start to walk across the bottom of the index, the first rowid points to the third row in the first block. We haven't visited any blocks yet, so this is a new block, so we count 1. Take one step along the index, and the rowid points to the fourth row of the second block—we've changed block, so increment the count. Take one step along the index, and the rowid points to the second row of the first block—we've changed block again, so increment the count again. Take one step along the index, and the rowid points to the fifth row of the first block—we haven't changed blocks, so don't increment the count.

In the diagram, I have put a number against each *row* of the table—this is to show the value of the counter as the walk gets to that row. By the time we get to the end of the index, we have changed table blocks ten times, so the `clustering factor` is 10.

QUESTIONS & ANSWERS

Q & A