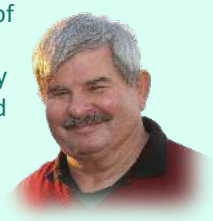
CS307

Database Principles

Stéphane Faroult
sfaroult@roughsea.com

朱悦铭 Zhu Yueming zhuym@sustc.edu.cn

We have seen that if the idea of "data warehousing" was first introduced by Bill Inmon, many original ideas were contributed by Ralph Kimball.



Bill Inmon

Ralph Kimball

In particular the idea of "facts" and heavily denormalized "dimensions".

We'll store date hierarchies that will ease aggregates

Dimension table

FullDate
DayOfWeek
DayNumInMonth
DayNumOverall
DayName
DayAbbrev
WeekdayFlag
WeekNumInYear
WeekNumOverall
WeekBeginDate
WeekBeginDateKey
Month
MonthNumOverall
MonthName
Quarter
Year
YearMonth
FiscalMonth
FiscalQuarter
FiscalYear
LastDayInMonthFlag
SameDayYearAgo

Hierarchy

Year
↓
Quarter
↓
Month
↓
Week
↓
Day

Dimension table

FullDate
DayOfWeek
DayNumInMonth
DayNumOverall
DayName
DayNameInSpanish
DayNameInFrench
DayNameInRussian
DayNameInArabic
DayAbbrev
DayAbbrevInSpanish
DayAbbrevInFrench
DayAbbrevInRussian
DayAbbrevInArabic
WeekdayFlag
WeekNumInYear
WeekNumOverall
WeekBeginDate
WeekBeginDateKey
Month
MonthNumOverall
MonthName

We can even adapt to multinational environments and do what we should never do in a decent database.

But then ANYBODY will be able to query with decent performance.

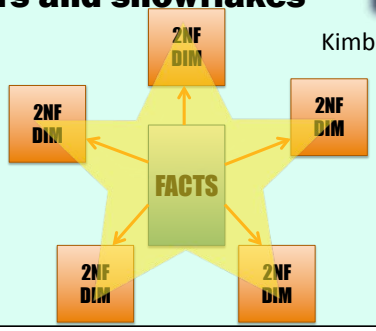
2

When you just have one facts table surrounded by dimensions, you have a "star schema".

Great!

Stars and snowflakes

Kimball ❤ ☆

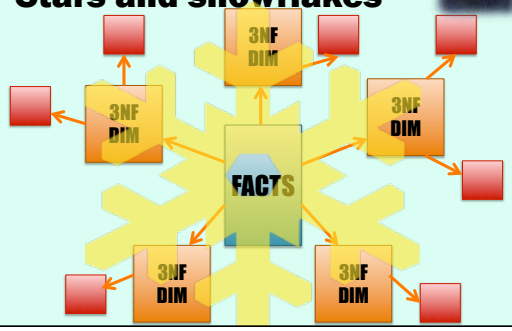


2

With additional joins you get a snowflake and Kimball dislikes it.

Yuk

Stars and snowflakes



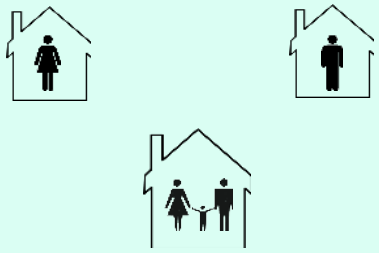
3

Slowly Changing Dimensions

A third important concept, a concept inherent to the idea of "long term data", is the idea of "slowly changing dimensions", or SCD. The problem is that dimensions change over the years. Take stores, for instance ("per store" could be an interesting sales result): new stores can be opened, other stores closed, a store can move to better premises in the same city.

B2C

If you are in a "business to consumer" (B2C) type of company, in the long term your customers will age and evolve. Households will change.



Where are my demographics?

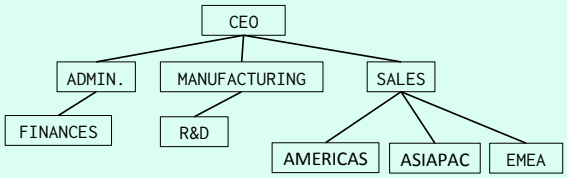
B2B

If you are in a "business to business" (B2B) type of company, some of your customers/ suppliers may merge or be taken over by others. You may need to still identify them as the same companies yet.

M&A

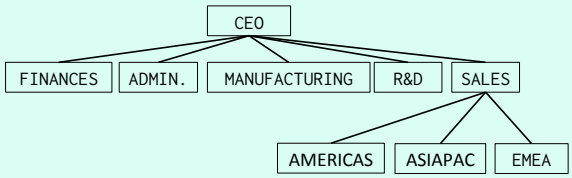
Where is my customer/supplier?

YEAR *N*



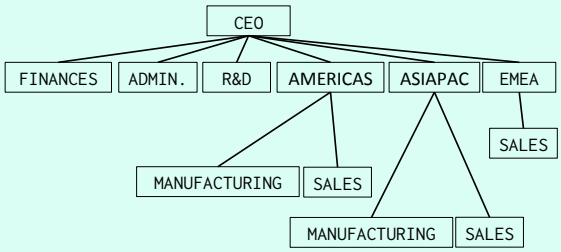
Most very big companies are in a perpetual state of reorganization.

YEAR *N+1*



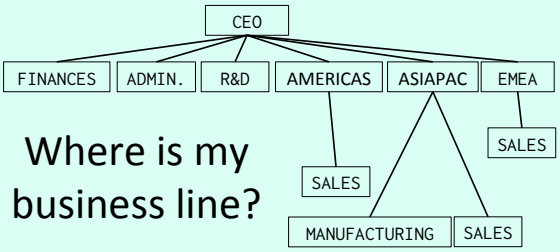
It may be linked to the recognition of the importance of one particular department, or the development of new markets.

YEAR *N+3*



It may be linked to changing business conditions, to new management fashions.

YEAR *N+5*



Where is my business line?

A part of the business may be sold off too. Finally, sometimes it's linked to the arrival of new managers or personal rivalries.

Kimball has tried to classify how to handle slowly changing dimensions. One way to handle them is to ignore them (iPhone 25 same product as iPhone 1)

Type 0 SCD

Keep as it was first inserted

Or you can update, and consider that there is Russia and there never was such thing as Soviet Union. Or consider that there is BDR and DDR, and there is Germany. Or that BDR and DDR became Germany in 1990.

Type 1 SCD Update dimension - lose old record

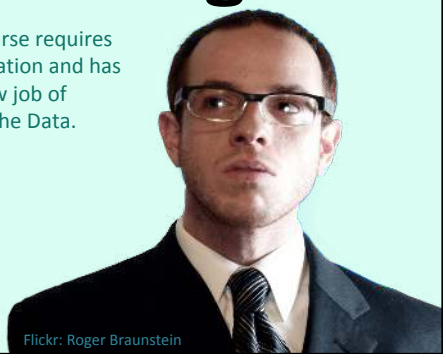
Type 2 SCD Add a new record –- Disconnect old facts/new facts

Type 3 SCD Add old/new/date information to the dimension

In practice most SCDs are type 1 or 2

Data Manager

All this of course requires due consideration and has created a new job of Guardian of the Data.



Flickr: Roger Braunstein

THE PLUMBING

Now, we have seen that in all likelihood most queries on a data warehouse will be massive aggregates, that take time. Worst, they will be run by people high up in the hierarchy, whose time is precious (especially when you consider their salaries) and who, having signed the checks and knowing how much all this did cost, expect bang for their bucks.

How can we have blazingly fast aggregates? With a traditional DBMS (there are other options, remember the columnar database presented by Guy Harrison) there are basically two solutions

Speeding up aggregates

Pre-computing aggregates

NOT mutually exclusive ...

When we want a query that says, for instance, "what was the sum of sales per province and month last year", from an SQL standpoint getting good performance is tough. We'll return at most 22x12 rows, which is peanuts (it won't be much better with municipalities, autonomous regions and special administrative regions). We'll scan a small table of geographical areas, and a small table of dates. Neither "province" nor "month" will be very selective.

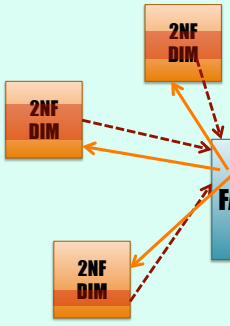
Because of 2NF, cardinality is often low

The only restriction "last year", is rather weak

Only broad criteria

And we may aggregate MANY rows.

Dimensions to facts



Very often, you go from referencing table to referenced table when you join. Here, all criteria will systematically be applied to reference tables that DON'T point to the facts table (it's the opposite)

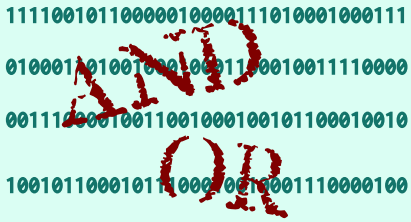
For low cardinality columns, advanced products such as Oracle implements "bitmap indexes". Their entries are chunks of bits that cover many rows and contain a bit that says whether the row contains the key or not.

Bitmap indexes

MON	10000001000000100000010000001000000
TUE	01000000100000010000001000000100000
WED	00100000010000001000000100000010000
THU	00010000001000000100000010000001000
FRI	00001000000100000010000001000000100
SAT	00000100000010000001000000100000010
SUN	00000010000001000000100000010000001

The advantage of bitmap indexes is that by applying AND and OR operations to the bitmaps you can quickly identify the rows that satisfy several conditions (Shenzhen and November) and fetch them.

Bitmap indexes



Facts Dimensions

Star transformation ORACLE

```
SELECT c.cust_city, t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount
FROM sales s, times t, customers c, channels ch
WHERE s.time_id = t.time_id
  AND s.cust_id = c.cust_id
  AND s.channel_id = ch.channel_id
  AND c.cust_state_province = 'CA'
  AND ch.channel_desc = 'Internet'
  AND t.calendar_quarter_desc IN ('1999-01','1999-02')
GROUP BY c.cust_city, t.calendar_quarter_desc
```

Oracle also implements a parameter that enables the "star schema transformation"

This transformation is just adding redundant conditions that make the choice of the suitable plan irresistible to the optimizer.

Star transformation ORACLE

```
SELECT c.cust_city, t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount
FROM sales s, times t, customers c, channels ch
WHERE s.time_id = t.time_id
  AND s.cust_id = c.cust_id
  AND s.channel_id = ch.channel_id
  AND c.cust_state_province = 'CA'
  AND s.cust_id IN (SELECT cust_id
                    FROM customers
                    WHERE cust_state_province='CA')
  AND ch.channel_desc = 'Internet'
  AND t.calendar_quarter_desc IN ('1999-01','1999-02')
GROUP BY c.cust_city, t.calendar_quarter_desc
```

Star transformation ORACLE

These redundant conditions are added for every dimension in the query

Similar idea, hand-crafted

We only need to hit the index on sales(cust_id) (it contains cust_id and rowid)

```
SELECT s.rowid
FROM sales s, customers c
WHERE s.cust_id = c.cust_id
  AND c.cust_state_province = 'CA'
INTERSECT
SELECT s.rowid
FROM sales s, times t
WHERE s.time_id = t.time_id
  AND t.calendar_quarter_desc IN ('1999-01','1999-02')
INTERSECT
SELECT s.rowid
FROM sales s, channels ch
WHERE s.channel_id = ch.channel_id
  AND ch.channel_desc = 'Internet'
```

"rowid" is Oracle-speak for "row address". You can try to identify the rows you need first.

We are assuming one separate index per FK

Similar idea, hand-crafted

Join to sales, and aggregate

Once you know exactly which rows you want, you can prey on them in the big table.

Database Machines/Appliances

TERADATA

NETEZZA an IBM Company

Greenplum

ExadataX3

and others …

Several companies have created dedicated hardware that is more specially geared towards the data warehouse type of queries. Some databases, called columnar databases, store data by columns rather than rows to speed up aggregates.

Do I need to say it doesn't come cheap?

Precomputing aggregates

Another, simple way to speed up aggregates is NOT to compute them on the fly but, if we have an idea about the aggregates that will be computed most often, to precompute them every time we upload the database, at least partially.

"Materialized views", which are actually tables, are saved query results (very often aggregates) that are prescheduled and automatically refreshed. You can query the materialized view directly, but some optimizers are smart enough to recognize when they can use a pre-computed result in a query on the original table. It's OK on data that isn't "real time data".

Materialized Views

NOT views

View = saved query text

Materialized view = saved query result

You specify in the creation statement the refresh schedule. It can be a full cancel and replace or, sometimes, an incremental refresh that requires a trigger-populated log on the original table(s).

Materialized Views

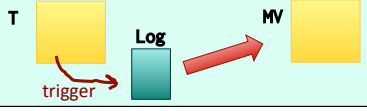
Refresh schedule

Of course, it's all plain replicated data.

Full refresh

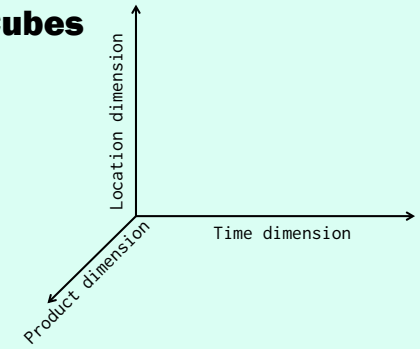
```
insert into ...
select ...
```

Fast refresh



Some products go further and try to compute every possible aggregate possible, detail AND all possible sums.

Cubes



If you don't ask for anything that strays from the predefined path, you get immediate results.

Cubes

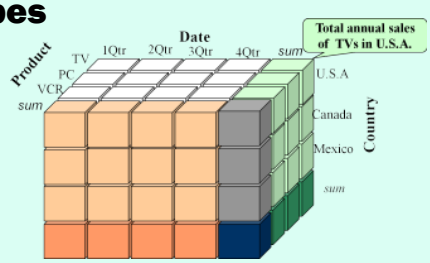


Image found on slidewiki.org

Cubes

Things become funny when you have MANY dimensions – the number of combinations explodes ...

FUNDAMENTALS

Database Design

Indexing

There are two basic pre-conditions for good performance: a proper database design, and proper indexing. A bad design leads to very hard queries, hard to write, and hard for the optimizer to improve. Proper constraints provide you with the main indexes required by joins.

Test: 5 million rows

Rows returned	Speed improvement with index
~ 1,000,000	x1.1
~ 500,000	x1.0
~ 50,000	x1.0
~5,000	x1.1
~1,000	x20.8

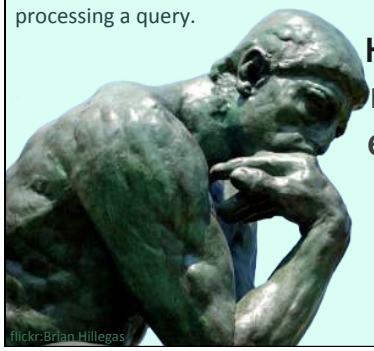
Keep in mind that the real benefit of index is only when you return very few rows. The measurements above were taken with MySQL, with Oracle and SQL Server scans perform even better.

If you wite queries in a way that prevents from using indexes ...

... options will be limited for the optimizer

The optimizer can recombine a query, it can switch between join, correlated/uncorrelated subquery, it cannot rewrite functions called at the wrong place or automatic data type conversions that return a result, but prevent from descending indexes. Badly writing queries is like closing doors.

The main problem of the optimizer is getting an estimate of how many rows are returned at each step when processing a query.



It won't be able to compute it and decide about using or not using indexes (few rows, indexes good, many rows, indexes bad) if it hasn't statistics. But what kind of statistics?

People

1,300,000,000

60,000,000

In terms of population size, there is no contest between China and France.

Family Names

3,000

1,300,000

BUT when it comes to surnames, it's the opposite. France and Italy are the countries with the most different surnames. It's common for French schoolboys (more than girls) to call each other by their surnames only, it sometimes happens in companies too.

One Family Name

400,000

46

It means that on average, there are almost half a million Chinese per surname, and under 50 French.
But averages are misleading.

The number of Lis in China is about the same as the population of the Philippines (12th most populated country in the world)

李 > 100,000,000

王 > 95,000,000

张 (張) > 90,000,000

The numbers of Wangs and Zhangs are pretty impressive too.

Even in France, some surnames are extremely common. And the top one is common in the UK and in the US too (although not pronounced the same)

Martin > 230,000

Bernard > 100,000

Dubois > 90,000

Other surnames look different only because of spelling variations (Lefebvre, Lefèvre, Lefébure are basically the same name – and mean the same as "Smith").

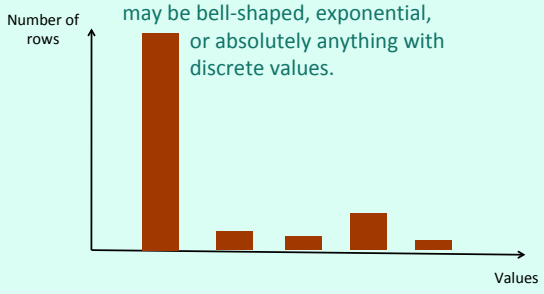
Basic statistics on columns are composed of minimum and maximum values, as well as the number of distinct values; but as we have just seen with surnames, dividing the number of rows in a table by the number of distinct values in a column (assuming uniform distribution) gives a pretty wrong idea of how many rows will be associated with one value.

Minimum value

Maximum value

Number of distinct values

We need a better idea of a distribution that may be bell-shaped, exponential, or absolutely anything with discrete values.



What does the optimizer know about data?

We are going to see what a product such as Oracle knows about data values in a column (which can be indexed or not; usually detailed statistics are collected over indexed columns)

```
SQL> create index movie_country_index on movies(country);

Index created.

SQL> begin
  2    dbms_stats.gather_table_stats(ownname=>user,
  3                                  tabname=>'movies',
  4                                  cascade=>true,
  5                method_opt=>'for all columns size auto');
  6  end;
  7  /

PL/SQL procedure successfully completed.

SQL>
```

I'm creating an index on countries on table MOVIES in a subset of the film database, and calling the Oracle procedure that collects statistics on a table.

```
SQL> select table_name, num_rows, blocks, avg_row_len
  2  from user_tables
  3  where table_name = 'MOVIES';

TABLE_NAME                     NUM_ROWS   BLOCKS     AVG_ROW_LEN
------------------------------ ---------- ---------- -----------
MOVIES                            183          5          26

SQL>
```

Basic stats abot the table are the number of rows, how many data blocks are used to physically store the table (it doesn't take indexes into account) and the size of the average row.

```
SQL> select table_name, num_rows, blocks, avg_row_len
  2  from user_tables
  3  where table_name = 'MOVIES';

TABLE_NAME                     NUM_ROWS   BLOCKS     AVG_ROW_LEN
------------------------------ ---------- ---------- -----------
MOVIES                            183          5          26

SQL> select index_name, uniqueness, distinct_keys, clustering_factor
  2  from user_indexes
  3  where table_name = 'MOVIES';

INDEX_NAME                     UNIQUENES DISTINCT_KEYS CLUSTERING_FACTOR
------------------------------ --------- ------------- -----------------
SYS_C0012408                   UNIQUE              183                 1
SYS_C0012409                   UNIQUE              183                 1
MOVIE_COUNTRY_INDEX            NONUNIQUE            37                 1
```

For indexes, uniqueness, keys and "clustering factor" (indexes named SYS_Cxxx are associated with unnamed primary key and unique constraints)

Distinct keys:

one composite index

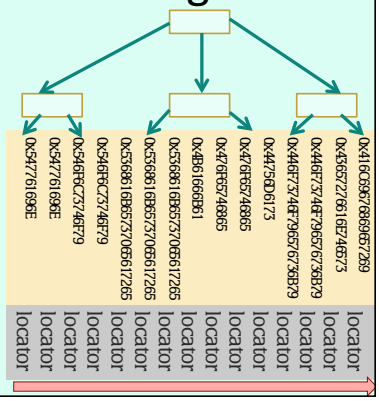
better than

several single-column indexes

But remember that if there is no condition on the first column in the index ...

Clustering factor?

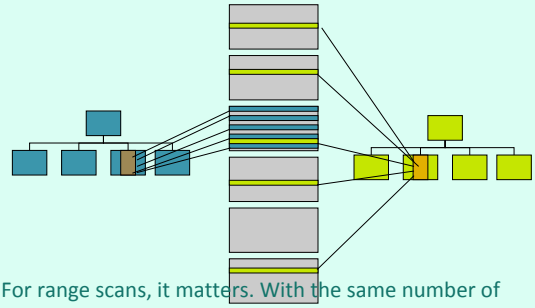
The clustering factor is obtained by taking all keys in order, and checking whether the row associated with a key is in the same block than the row associated with the previous key. If not, we increase the factor.



Relationship between **order of rows** and **order of keys**

What is the result? If the clustering factor is close to the number of rows in the table, that means that two successive keys are generally in different blocks. If it's close to the number of blocks in the table, it means that the rows are more or less in the same order as the keys.

Order of Data



For range scans, it matters. With the same number of distinct keys in two indexes, the number of table blocks to inspect may be very different.

Lower clustering factor

=

Fewer blocks
to fetch and inspect

Columns?

We have seen what the Oracle optimizer knows about tables, and what it knows about indexes. What can it know about column values?

```
SQL> select column_name, data_type,
  2     nullable, num_distinct,
  3     low_value, high_value, density,
  4     num_nulls, num_buckets
  5  from user_tab_columns
  6  where table_name = 'MOVIES'
  7  order by column_id;

COLUMN_NAME          DATA_TYPE  N NUM_DISTINCT LOW_VALUE
HIGH_VALUE      DENSITY    NUM_NULLS  NUM_BUCKETS
-------------------- ---------- - ------------ ---------------
---------------- ---------- ---------- -----------
MOVIEID              NUMBER     N          183 C102
C20254          .005464481          0           1
TITLE                VARCHAR2   N          174 3132207374756C7
5A68656E2078696 .00273224           0         174
COUNTRY              CHAR       N           37 6172
7573            .00273224           0          37
YEAR_RELEASED        NUMBER     N           69 C21416
C2150A          .00273224           0          69
```

Column name, data type, nullable and num_nulls (number of nulls) require no comment.

```
SQL> select column_name, data_type,
  2     nullable, num_distinct,
  3     low_value, high_value, density,
  4     num_nulls, num_buckets
  5  from user_tab_columns
  6  where table_name = 'MOVIES'
  7  order by column_id;

COLUMN_NAME          DATA_TYPE  N NUM_DISTINCT LOW_VALUE
HIGH_VALUE      DENSITY    NUM_NULLS  NUM_BUCKETS
-------------------- ---------- - ------------ ---------------
---------------- ---------- ---------- -----------
MOVIEID              NUMBER     N          183 C102
C20254          .005464481          0           1
TITLE                VARCHAR2   N          174 3132207374756C7
5A68656E2078696 .00273224           0         174
COUNTRY              CHAR       N           37 6172
7573            .00273224           0          37
YEAR_RELEASED        NUMBER     N           69 C21416
C2150A          .00273224           0          69
```

The number of distinct values gives a good idea of selectivity, even when not unique (TITLE)

```
SQL> select column_name, data_type,
  2     nullable, num_distinct,
  3     low_value, high_value, density,
  4     num_nulls, num_buckets
  5  from user_tab_columns
  6  where table_name = 'MOVIES'
  7  order by column_id;

COLUMN_NAME          DATA_TYPE  N NUM_DISTINCT LOW_VALUE
HIGH_VALUE      DENSITY    NUM_NULLS  NUM_BUCKETS
-------------------- ---------- - ------------ ---------------
--------------- ---------- ---------- -----------
MOVIEID              NUMBER     N          183 C102
C20254          .005464481          0           1
TITLE                VARCHAR2   N          174 3132207374756C7
5A68656E2078696 .00273224           0         174
COUNTRY              CHAR       N           37 6172
7573            .00273224           0          37
YEAR_RELEASED        NUMBER     N           69 C21416
C2150A          .00273224           0          69
```

Minimum and maximum values are encoded, "density" assumes uniform distribution.

```
SQL> select column_name, data_type,
  2     nullable, num_distinct,
  3     low_value, high_value, density,
  4     num_nulls, num_buckets
  5  from user_tab_columns
  6  where table_name = 'MOVIES'
  7  order by column_id;

COLUMN_NAME          DATA_TYPE  N NUM_DISTINCT LOW_VALUE
HIGH_VALUE      DENSITY    NUM_NULLS  NUM_BUCKETS
-------------------- ---------- - ------------ ---------------
--------------- ---------- ---------- -----------
MOVIEID              NUMBER     N          183 C102
C20254          .005464481          0           1
TITLE                VARCHAR2   N          174 3132207374756C7
5A68656E2078696 .00273224           0         174
COUNTRY              CHAR       N           37 6172
7573            .00273224           0          37
YEAR_RELEASED        NUMBER     N           69 C21416
C2150A          .00273224           0          69
```

More interesting, num_buckets is what Oracle uses for tracking distribution with histograms.

Histograms:

Useless if unique

If a column is unique, the number of buckets will be one or zero, meaning no histogram.

More of fewer than 256 distinct values?

If there are fewer than 256 values, Oracle will count how often each one occurs, and each counter will be a "bucket".

Frequency Histogram

When each individual value is counted, this is called a "frequency histogram" and the result is of course extremely precise.

This is what Oracle holds in a dictionary table.

```
SQL> select column_name, endpoint_number, endpoint_value, endpoint_actual_value
  2  from user_tab_histograms
  3  where table_name = 'MOVIES'
  4  order by 1, 2;

COLUMN_NAME          ENDPOINT_NUMBER ENDPOINT_VALUE ENDPOINT_ACTUAL_VALUE
-------------------- --------------- -------------- ------------------------
COUNTRY                            2     5.0597E+35
COUNTRY                            3     5.0601E+35
COUNTRY                            5     5.1090E+35
COUNTRY                            6     5.1094E+35
COUNTRY                           11     5.1116E+35
COUNTRY                           13     5.1601E+35
COUNTRY                           15     5.1615E+35
...
COUNTRY                          112     5.8371E+35
COUNTRY                          114     5.8373E+35
COUNTRY                          115     5.9418E+35
COUNTRY                          127     5.9430E+35
COUNTRY                          130     5.9917E+35
COUNTRY                          131     5.9921E+35
COUNTRY                          134     5.9939E+35
COUNTRY                          135     6.0472E+35
COUNTRY                          183     6.0983E+35
```

RAW

This is what it means ("end points" are cumulative).

```
Number of rows in the table: 183 (last computed ...)
Sample size 100.000% (183 rows) collected ...

ENDPOINT                       POPULARITY      FREQUENCY
------------------------------ ---------- ----------------------------------------
ar                                      2 =
at                                      1
be                                      2 =
bg                                      1
br                                      5 =
...
fr                                      7 ==
gb                                      6 ==
hk                                      7 ==
id                                      1
in                                     26 ========
it                                      9 ==
jp                                      7 ==

...
pl                                      2 =
ro                                      1
ru                                     12 ===
se                                      3 =
sg                                      1
sp                                      3 =
tw                                      1
us                                     48 ==========
```

COOKED

Height-Balanced Histogram

(Oracle <= 11gR2)

When there are more than 255 different values, Oracle will use 254 buckets but will use a different algorithm to distribute values to the bucket. Here is what it was doing up to version 12.

Assume that the limit is 5 instead of 254 "buckets"

We'll do it for countries but assume that we want to distribute them between a far smaller number of buckets than what Oracle would do.

```
SQL> select country, count(*) from movies
  2  group by country
  3  order by country;

CO    COUNT(*)
-- ----------
ar         2
at         1
be         2
bg         1
br         5
ca         2
ch         2
cn         6
cz         2
de         5
dk         3
eg         5
fr         7
gb         6
hk         7
```

Here is the number of countries, with the number of films for each.

183 rows - divide by 5 buckets

=> 37 rows by bucket

If we want to distribute them across 5 buckets, then we want 37 rows per bucket.

```
SQL> select country, count(*) from movies
  2  group by country
  3  order by country;

CO    COUNT(*)
-- ----------
ar         2
at         1
be         2
bg         1
br         5
ca         2
ch         2
cn         6
cz         2
de         5
dk         3
eg         5
fr         7    => 1 in first bucket, 6 in next
gb         6
hk         7
```

We are taking the list from the beginning, and as long as we haven't 37 rows we assume they all go to the first bucket. The first bucket fills up with the first French film, so 'fr' indicates the end of the first bucket.

```
hk         7
id         1
in        26  => 17 in 2nd bucket, 9 in next
it         9
jp         7
kr         1
ma         2
mx         2
my         1
ne         1
ng         1
nl         2
no         1
nz         1
pk         1
pl         2
ro         1
ru        12
se         3
sg         1
sp         3
tw         1
us        48
SQL>
```

And we go on, filling buckets one after the other, distributing equally (give or take one) between buckets.

```
hk      7
id      1
in     26  => 17 in 2nd bucket, 9 in next
it      9
jp      7
kr      1
ma      2
mx      2
my      1
ne      1
ng      1
nl      2
no      1
nz      1
pk      1 => end of 3rd bucket
pl      2
ro      1
ru     12
se      3
sg      1
sp      3
tw      1
us     48 => end of 4th and 5th buckets
SQL>
```

Bucket	end value
1	fr
2	in
3	pk
4	us
5	us

"bucket boundaries" are supposed to be "popular" (i.e. frequent) values. Obviously wrong for pk and we missed ru.

Hybrid Histogram

(Oracle >= 12c)

Don't split values between buckets

The height-balanced histograms were often giving less than convincing results, and many DBAs were preferring having no histograms rather than height-balanced histograms. This is why the algorithm has been changed in Oracle 12.

```
SQL> select country, count(*) from movies
  2  group by country
  3  order by country;

CO    COUNT(*)
-- ----------
ar         2
at         1
be         2
bg         1
br         5
ca         2
ch         2
cn         6
cz         2
de         5
dk         3
eg         5
fr         7 => 43 values in first bucket
gb         6
hk         7
```

With hybrid histograms, even if the goal is still to have roughly 37 countries per bucket, we'll cram all the films from one country in one bucket and won't split them between two buckets.

```
hk      7
id      1
in     26  => 40 values in 2nd bucket
it      9
jp      7
kr      1
ma      2
mx      2
my      1
ne      1
ng      1
nl      2
no      1
nz      1
pk      1
pl      2
ro      1
ru     12  => 44 values in 3rd bucket
se      3
sg      1
sp      3
tw      1
us     48 => 56 values in 4th bucket
SQL>
```

Bucket	end value	size
1	fr	43
2	in	40
3	ru	44
4	us	48

The preceding anomaly is gone.

Better but not perfect ...

This said, you will notice that even if there are MORE Italian and AS MANY Hong-Kong films as French ones, neither "it" nor "hk" appears as a popular value. We may still have cases when the optimizer fails to estimate correctly how many rows will be associated with a country value. Of course, in a complex query poor estimates may cascade.

Histogram issues

Far away values

Oracle doesn't index nulls

Another problem with Oracle is linked to something which is common with dates, the use of dummy values. As nulls may not be stored in Oracle and may have no address, they cannot be indexed. Columns "valid_until" are frequent, and rather than having a null to indicate current values, people often prefer using a dummy, far away date that can be indexed.

Histogram issues

Far away values

31st Dec ~~9~~9999

Except that using the maximum date allowed by Oracle is a bad idea. If your oldest date is 2000, asking for everything until 01/01/2018 will probably mean returning most of the rows. The optimizer may see 18 years out of a range of 8000 (remember, it knows min and max) and think that it's a small percentage that would be fetched faster with an index.

Histogram issues

False range scan

Another issue which isn't a histogram issue proper but for which you might think that histograms would help when they probably won't is what I call a "false range scan".

Histogram issues

False range scan

Real `where event_date between sysdate -7 and sysdate`

column constant constant

A true range scan is when you ask for values in a column that are between two constant (or computed) values. With histograms, the optimizer can get a fairly good idea of the number of rows returned, and great if the index has a good clustering factor.

Histogram issues

False range scan

`where event_date between sysdate -7 and sysdate`

False `where sysdate between start_date and end_date`

constant column1 column2

A false range scan looks deceivingly like a real one, except that here you are comparing ONE constant to TWO columns instead of the reverse.

Histogram issues

False range scan

`where event_date between sysdate -7 and sysdate`

```
where sysdate >= start_date
  and sysdate <= end_date
```

What you are really doing is taking the intersect of two open-ended intervals. And here, getting an estimate of the number of rows is FAR more difficult, and indexes won't help much.

Histogram issues

False range scan

```
ip_range_start    ip_range_end    location
----------------+---------------+------------

valid_credit_card_start   valid_credit_card_end
------------------------+---------------------
```

It's a problem that is commonly encountered, as soon as you have a "from" column and a "to" column.

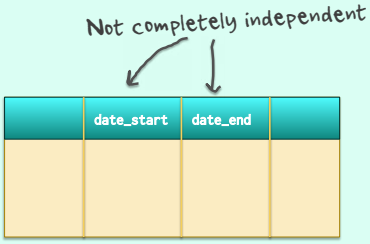
Finally, the optimizer may far overestimate a number of rows returned because of correlations between columns.

Correlation

The optimizer assumes 3rd Normal Form

... attributes only depend on the key and are independent

However, saying "independent" is often a bit far fetched.



If I tell you "how many people where born after 1980 and died before 1940" you'll tell me immediately 0. For the optimizer, if 20% of people were born after 1980 and 20% died before 1940, it will be 20%x20% = 4% of the total number of rows. You can ask Oracle to compute stats on groups of columns.

Design and Statistics

"Artificial" Correlations

Knowing which statistics to collect and, from there, trying to estimate how many rows should be returned by such or such parameter value (a crucial step in the choice of an execution plan) is hard enough for the DBMS system. Poor design choices make it far more difficult for the optimizer and increase significantly the chances of its taking a bad decision.

Let's take for example the "Entity/attribute/Value" model so loved of people who start coding without really knowing which attributes their data will ultimately hold.

PEOPLE

entity	attribute	value
567	first_name	Gene
567	surname	Tierney
567	birth	1920
567	death	1991
567	gender	F

What should be horizontal is vertical. Not only can you forget about foreign keys, but the optimizer will struggle.

468 rows
346 different first names (456 people have one)
418 different surnames
106 different birth years
71 different death years (for 237 people)
2 genders

I have computed a few statistics of my own on a relatively small sample of actors and directors. First I stored my data in a traditional way (id, first name, surname, born, died, gender)

Entity/Attribute/Values

5 different attributes – almost uniform
(null won't appear)

Rows = surname + born + gender
for everybody 1404
first_name for almost everybody 456
death year for half the people 237
Total 2097 rows

With the EAV model, the same data becomes a table with half the number of columns and more than three times the number of rows. Only known values are stored.

With a standard model, how many rows will be returned by a condition on gender is easy to compute.

`gender = 'F'` ~50%, ~230 rows

With an EAV model, because the attribute name is now part of the data, assuming data independence is wrong. 'Gender' is always associated with either 'M' or 'F'.

`attr_name = 'Gender'`	1/5	20%
`and attr_value = 'F'`	230/2097	11%

20% * 11% ?

Assuming that columns are independent, the optimizer will believe that only 2% of the 2,000 rows will be returned, and will be wrong by an order of magnitude.

High odds to pick the wrong algorithm

As, if you remember Kyte's allegory, the suitability of an algorithm depends on data volumes, an EAV model will underestimate volumes and pick on indexes and algorithms that use indexes.

"Extended statistics" on several columns

Hence the benefit of "extended statistics" (which only the top products such as Oracle allow), not only to detect genuine correlation in the data (such as between "born" and "died") but also to try to correct artificial correlation introduced by the data model (because any value cannot be associated with any attribute).

Bind variables

Taken into account during parsing

Stability issues come from bind variables that have a distribution value that isn't uniform. You'll never have a problem with ids or unique columns, just with columns in which some values are very common, and others fairly rare.

```
select …
from …
where datecol between
    cast('2013-12-01' as date)
and cast('2013-12-31' as date)
and status = 'COMPLETED'
```

COMMON

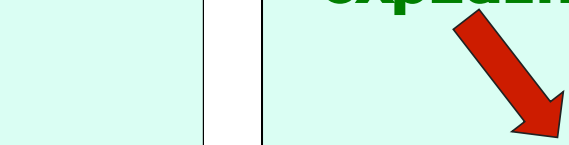
The problem may be more obvious when the column is indexed, but in fact it's more general than that because even a non-indexed column can be important for estimating the size of the result-set.

```
select …
from …
where datecol between
    cast('2013-12-01' as date)
and cast('2013-12-31' as date)
and status = 'ERROR'
```

RARE

You may be using the index on datecol to access data, but if this query is a subquery fed into an operation such as a join, it matters to know whether it returns most of the rows between the two dates or very few.

explain



And the index that may be important to use may be a completely different index.

Index used?

```
select …
from …
where datecol between
    cast(:date_min as date)
and cast(:date_max as date)
and status = :status
```

Check parameters when parsing?

As we have already seen, the parameters may be checked when parsing but because parsed queries may age out of a buffer and being reparsed with different parameters, plans may completely change later in the day. The plan is usually adequate for the first execution; it may be completely wrong for some subsequent executions.

UNPREDICTABLE

is worse than

SLOW

Unpredictability is something that everybody hates. You can live with something that its consistently mediocre. Something that can be sometimes blazingly fast and sometimes horribly slow for no apparent reason is something you want to avoid.