Leture 14

Slowly Changing Dimensions

- Type 0 SCD: Keep as it was first inserted
- 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

One great illustration about SCD blog

Make Aggregate Faster

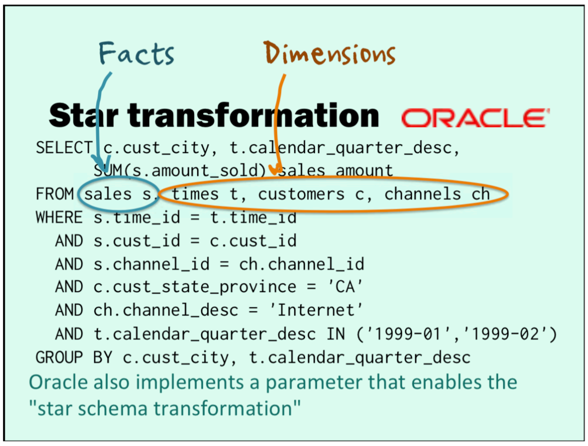
Speeding Up Aggregates

Bitmap index

When the colume values are discrete and limited, we can use bitmap index to improve efficiency

One great illustration blog

Star Transformation



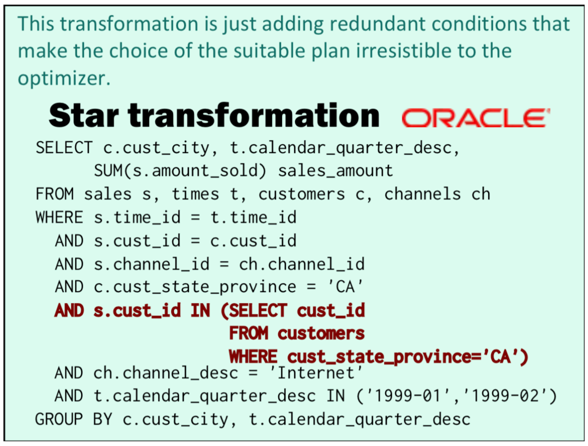
Facts

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
```

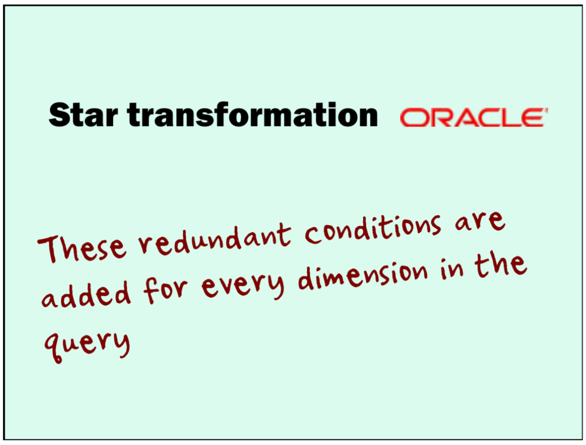
Oracle also implements a parameter that enables the "star schema transformation"



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

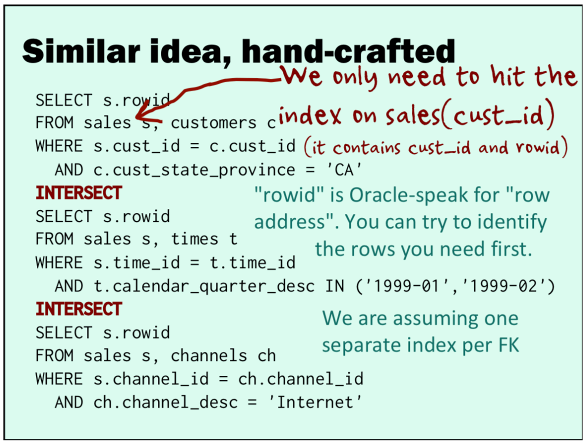
Star transformation ORACLE

```
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

```
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'
```

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

"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

Precomputing aggregates

Materialized Views

- saved query **result**

Performance

Fundamentals

- Database Design
- Indexing

*Keep in mind that the real benefit of index is only when you return **very few** rows.*

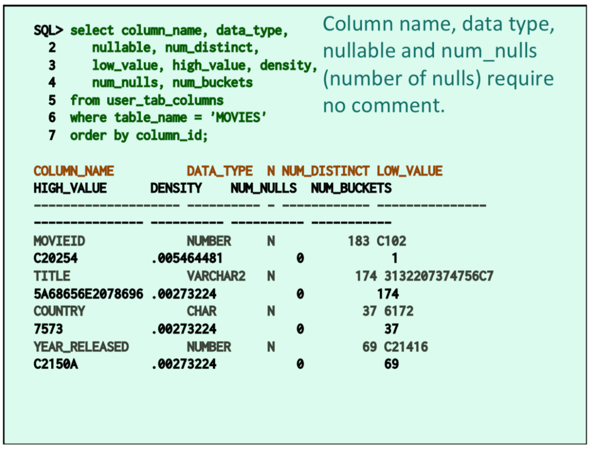
Distinct keys: one composite index better than several single-column indexes

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.

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.

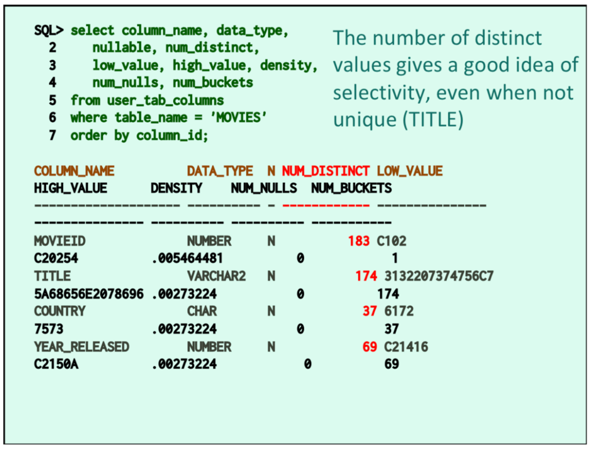
Lower clustering factor = **Fewer** blocks to fetch and inspect



```
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)

NUM_BUCKETS

num_buckets is what Oracle uses for tracking distribution with 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.

Height-Balanced Histogram

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.

Hybrid Histogram

We'll cram all the films from one country in one bucket and won't **split them between two buckets**.

Histogram Issues

- Far away values
  - 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.
  - 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.
- False range scan
  - 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.

Column
between
Constants

  - 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.

Constant
between
Columns

Correlation

The optimizer assumes **3rd Normal Form**

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

> **start_date and end_date are not completely independent**
>
> 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.

artificial correlation

EAV Model

Stability Issues

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.