Meeting 11/22/17

Wednesday, November 22, 2017 9:00 AM

Agenda:
Experiment with cardinality

Attendees:
Kirby, Andy
Alex, Trevor

Ordered rows in the same way every time
- Doesn't move columns with same data
- Symbol table looked almost identical

- Col 1 and col 2 had all A's, but they were not shifted

- Optimize when you pull all data from a query, not just one

Wants to build a table, and from that table create other tables that gives him all the suites he wants. Specifically, block cardinality.
- Length of the string
    - 1 char, 2, 4, 8, 16, 32, 64
    - Look at block level cardinality of string
        - Does the string only possess 1 single value (bunch of A's, half as half bs?)

- How do they compress
- What's the difference between 8, 16, 32k block as far as how they compress

- We want to guarantee what's in a block? What's the length and what's the cardinality?

- We should all run the same experiments to ensure we get the same thing

If it's a char, we replace with number (normalize) long term
Wants to get rid of key value necessity

EXTENT TRIMMING (locally managed database is what HP uses)
https://blogs.oracle.com/datawarehousing/parallel-load:-uniform-or-autoallocate-extents
Make sure we're not counting table blocks
- Most loads leverage direct path ops where the load process directly formats and writes oracle blocks to disk instead of going thru buffer cache
- Loading process allocates extents and fills them with data during load
- During parallel loads, each loader process will allocate its own extent and no two processes work the same extent

- **Autoallocate will dynamically adjust the size of an extent and trim the extent after the load in case it is not fully loaded**

http://www.oracle.com/technetwork/issue-archive/2007/07-may/o37asktom-101781.html
- This article was referenced in the above article
- Oracle Database allocates a new extent for each file loaded (number of extents per table = number of load files per table), instead of using up available space in extents

    - One solution is to use a much smaller uniform extent size, and the other (and easier) solution is to use AUTOALLOCATE extent sizes, which permits Oracle Database to trim the extents back to the smallest size possible.
- Parallel Data Definition Language (DDL) and operations such as a direct path load via SQL *Loader rely on direct path operations. (The data is not passed to the buffer cache to be written later)
- A direct path insert (INSERT APPEND) writes above a segment's high water mark, and each parallel execution server again writes to its own set of extents.
    - Bad for data warehouse environment

UNIFORM SIZE, with which every extent in the tablespace is always precisely the same size
AUTOALLOCATE, with which Oracle Database decides how big each extent should be, by using an internal algorithm

Both approaches solve the problem of having 99MB of free space, followed by 1MB of used space, followed by 99MB of free space, and so on.

The UNIFORM SIZE approach obviate extent trimming altogether. When you use UNIFORM SIZE extents, Oracle cannot perform extent trimming. All extents are of a single size - none can be smaller or larger than that single size.

AUTOALLOCATE extents DO support extent trimming. They use a few specific extent sizes and have the ability to use space of different sizes - the algorithm permits the use of all free space in the tablespace over time.

- It can reduce the size of the request it makes to attempt to use all of the free space.

"If your goal is to do direct path loading in parallel as often as possible, [Tom Kyte] suggests using AUTOALLOCATE. Parallel direct path operations will not use space under the objects hwm (the space on the free list). So, unless you do some conventional path inserts into these tables as well, UNIFORM allocation will permanently contain additional free space that it will never use. "

http://blog.orapub.com/20160928/does-oracle-table-column-order-affect-performance.html
Put most commonly accessed data first

https://community.oracle.com/thread/3654221
From Christian Antognini's Troubleshooting Oracle Performance:

Optimal Column Order
Little care is generally taken to find the optimal column order for a table. Depending on the situation, this might have no impact at all or may cause a significant overhead. To understand what situations this might cause significant overhead in, it's essential to describe how the database engine stores rows into blocks.

The essential thing to understand in this format is that the database engine doesn't know the offset of the columns in a row. For example, if it has to locate column 3, it has to start by locating column 1 (that's simple, since the length of the header is known). Then, based on the length of column 1, it locates column 2. Finally, based on the length of column 2, it locates column 3. So whenever a row has more than a few columns, a column near the beginning of the row might be located much faster than a column near the end of the row.

For next week:

- Keep playing with tokens / token tables
    - How it combines columns
- Can use that info to create multiple column tests