Awesome yes this works, and it’s pruning!

|  |
| --- |
| drop table PartitionedTable purge;  CREATE TABLE PartitionedTable  (  id number,  PartitionKey VARCHAR2(128 BYTE), -- using varchar  created date,  trunc\_created date generated always AS (trunc(created)) virtual  )  PARTITION BY LIST (trunc\_created, PartitionKey) AUTOMATIC  (  PARTITION PSTART VALUES (to\_date('01/01/1970', 'DD/MM/YYYY'), 'start')  );  insert into PartitionedTable (id, partitionkey, created) VALUES (1,100,to\_date('01/02/1990','DD/MM/YYYY'));  insert into PartitionedTable (id, partitionkey, created) VALUES (1,200,to\_date('01/02/1990','DD/MM/YYYY'));  insert into PartitionedTable (id, partitionkey, created)select OBJECT\_ID,OBJECT\_ID, max(created) from all\_objects where rownum <2 group by OBJECT\_ID;  commit;  exec dbms\_stats.gather\_table\_stats(ownname => 'SYSTEM',tabname => 'PARTITIONEDTABLE', granularity => 'ALL');  -- prune 35nrrd3pyrsuj  select /\* 22xqv11u52m5wa2a1 \*/ \* from PartitionedTable where PartitionKey ='200';  -- prune 5m59m5nxnwz7h  select /\* 22xqv11u52m5wb2b1 \*/ \* from PartitionedTable  where created >= to\_date('19920101','yyyymmdd') ;    -- prune cfyhddy58n920  select /\* 22xqv11u52m5wc2c1 \*/ \* from PartitionedTable  where created <= to\_date('19980101','yyyymmdd') ;    -- prune 4xxuycaygzqbs  select /\* 22xqv11u52m5wd2d1 \*/ \* from PartitionedTable  where PartitionKey ='9'  and created >= to\_date('20180101','yyyymmdd'); |

Although when I did a larger dataset test since the detail of virtual key partition is by day (323 partitions) the loading is slower and querying is slower as well compared to a

RANGE partition by month + subpartition by LIST (4 partitions + 8 subpartitions).

It was 209secs vs 34secs when my laptop was at 100% CPU. I think the overhead (on insert and query) is the metadata operations on 323 partitions

|  |
| --- |
| -- prune a65p8s82svc75 (this query is slow 200secs vs 34secs RANGE LIST virt col TEST\_RANGE\_LIST\_VIRT  select /\* TEST\_LIST\_RANGE\_AUTO\_VIRTd \*/ \* from TEST\_LIST\_RANGE\_AUTO\_VIRT  where Trunc(Cast(begin\_date AS DATE)) >= to\_date('20190501','yyyymmdd') ;  **200seconds**  --SQL\_ID CHILD PLAN\_HASH EXECS AVG\_ETIME AVG\_LIO SUBSTR(SQL\_FULLTEXT,1,4000)  --------------- ------ ---------- ------------ ----------------- ------------ --------------------------------------------------------------------------------  --a65p8s82svc75 0 2702690513 1 **209.038185** 290 select /\* TEST\_LIST\_RANGE\_AUTO\_VIRTd \*/ \* from TEST\_LIST\_RANGE\_AUTO\_VIRT  --Plan hash value: 2702690513  --  ------------------------------------------------------------------------------------------------------------------------  --| Id | Operation | Name | E-Rows |E-Bytes| Cost (%CPU)| E-Time | Pstart| Pstop |  ------------------------------------------------------------------------------------------------------------------------  --| 0 | SELECT STATEMENT | | | | 65 (100)| | | |  --| 1 | PARTITION LIST ITERATOR| | 72434 | 10M| 65 (2)| 00:00:01 | KEY | KEY |  --|\* 2 | TABLE ACCESS FULL | TEST\_LIST\_RANGE\_AUTO\_VIRT | 72434 | 10M| 65 (2)| 00:00:01 | KEY | KEY |  ------------------------------------------------------------------------------------------------------------------------  **34seconds**  --SQL\_ID CHILD PLAN\_HASH EXECS AVG\_ETIME AVG\_LIO SUBSTR(SQL\_FULLTEXT,1,4000)  --------------- ------ ---------- ------------ ----------------- ------------ --------------------------------------------------------------------------------  --fvhvnbdvdkdr7 0 2428588542 1 **33.628897** 988 select /\* TEST\_RANGE\_LIST\_VIRTd0 \*/ \* from TEST\_RANGE\_LIST\_VIRT  --Plan hash value: 2428588542  --  --------------------------------------------------------------------------------------------------------------------  --| Id | Operation | Name | E-Rows |E-Bytes| Cost (%CPU)| E-Time | Pstart| Pstop |  --------------------------------------------------------------------------------------------------------------------  --| 0 | SELECT STATEMENT | | | | 35 (100)| | | |  --| 1 | PARTITION RANGE ITERATOR| | 5626 | 857K| 35 (0)| 00:00:01 | 594 |1048575|  --| 2 | PARTITION LIST ALL | | 5626 | 857K| 35 (0)| 00:00:01 | 1 | 2 |  --|\* 3 | TABLE ACCESS FULL | TEST\_RANGE\_LIST\_VIRT | 5626 | 857K| 35 (0)| 00:00:01 | 1187 |1048575|  --------------------------------------------------------------------------------------------------------------------  -- |

Another thing I found is when you modify the virtual column to TRUNC by MONTH the queries no longer prune

|  |
| --- |
| **-- this prunes**  BEGIN\_DATE DATE,  END\_DATE DATE,  begin\_date\_virtual date generated always AS (Trunc(Cast(begin\_date AS DATE))) virtual  )  PARTITION BY LIST (begin\_date\_virtual, owner, object\_type ) AUTOMATIC  (  PARTITION PSTART VALUES (to\_date('01/01/1970', 'DD/MM/YYYY'), 'START','START')  );  --prunes  select /\* TEST\_LIST\_RANGE\_AUTO\_VIRTd \*/ \* from TEST\_LIST\_RANGE\_AUTO\_VIRT  where Trunc(Cast(begin\_date AS DATE)) >= to\_date('20190501','yyyymmdd') ;  **-- this doesn’t prune**  BEGIN\_DATE DATE,  END\_DATE DATE,  begin\_date\_virtual date generated always AS (Trunc(Cast(begin\_date AS DATE),'MONTH')) virtual  )  PARTITION BY LIST (begin\_date\_virtual, owner, object\_type ) AUTOMATIC  (  PARTITION PSTART VALUES (to\_date('01/01/1970', 'DD/MM/YYYY'), 'START','START')  );  --no prune  select /\* TEST\_LIST\_RANGE\_AUTO\_VIRT\_MONTHd1b \*/ \* from TEST\_LIST\_RANGE\_AUTO\_VIRT\_MONTH  where Trunc(Cast(begin\_date AS DATE)) >= to\_date('20190501','yyyymmdd') ;  --no prune  select /\* TEST\_LIST\_RANGE\_AUTO\_VIRT\_MONTHd3b1a \*/ \* from TEST\_LIST\_RANGE\_AUTO\_VIRT\_MONTH  where Trunc(Cast(SYSDATE AS DATE),'MONTH') >= to\_date('20190501','yyyymmdd') ; |

So ultimately the RANGE INTERVAL SUBPARTITION BY LIST is the most effective and flexible

1. You can specify the virtual column (Trunc(Cast(begin\_date AS DATE))) to address the TRUNC(CAST queries which ideally this should be fixed on the app level
2. You can specify the granularity of partitions to be wider (MONTH or YEAR) to have bigger partition segments which could lead to less metadata overhead

|  |
| --- |
| BEGIN\_DATE DATE,  END\_DATE DATE,  begin\_date\_virtual invisible generated always as (Trunc(Cast(begin\_date AS DATE)))  )  PARTITION BY RANGE (begin\_date\_virtual)  INTERVAL ( NUMTOYMINTERVAL (1, 'MONTH') )  SUBPARTITION BY LIST (owner, object\_type)  SUBPARTITION TEMPLATE (  SUBPARTITION SP1 VALUES ('SYS','TABLE'),  SUBPARTITION SPDEFAULT VALUES (DEFAULT) )  (  PARTITION PSTART VALUES LESS THAN (TO\_DATE ('1970-01-01','yyyy-mm-dd')) ); |

Just a word of caution on virtual column. **Keep the expressions as simple as possible.**

Because this column value is executed on the fly and the more functions you put in there the more overhead you’ll have most likely on CPU or possibly serialize on that virtual column.

Check Adrian’s article <http://www.oracle-developer.net/display.php?id=510> , the section “**virtual columns based on pl/sql functions**” where he demonstrated it ran 3x slower vs just the simple virtual column

I had a similar scenario before where the customer added a virtual column that computes the MD5 hash of rows of an unpartitioned table on the fly.

Since the virtual column is always generated, any other SQLs hitting the table would also cause a call to the MD5 function even if that SQL doesn’t need the MD5 data.

When I profiled this query, the function is called twice 1st is on pulling the data and 2nd on sorting the rows.

And since the virtual column is always generated there’s no way to avoid calling the MD5 function to make this query faster.

Indexing on SYNC\_LOG\_ID,CREATEDDT would scan less which could help on performance but the MD5 function call would still be there.

Usually, this MD5 is handled as part of the SCD query on the ETL tool

|  |
| --- |
| **--Tabe Virtual Column Definition where it is used as on the fly MD5 – the table is 15GB unpartitioned table**  "HASH\_VAL" raw(32767) generated always AS ("APP\_USER"."HASH\_KEY\_GENERATOR"(lower(trim("COL1"))  ||lower(trim("COL2"))  ||lower(trim("COL3"))  ||lower(trim("COL4"))  ... and so on ...  ||lower(trim("COL56"))  ||lower(trim("COL57")))) virtual  **--The function used on virtual column**  CREATE OR REPLACE EDITIONABLE FUNCTION "APP\_USER"."HASH\_KEY\_GENERATOR" (v\_input VARCHAR2)  RETURN DBMS\_OBFUSCATION\_TOOLKIT.raw\_checksum  DETERMINISTIC  AS  BEGIN  RETURN DBMS\_OBFUSCATION\_TOOLKIT.md5 (  input => UTL\_RAW.CAST\_TO\_RAW (v\_input));  END hash\_key\_generator;  /  **--The CPU bound query – runs for 6 seconds and executed every minute (always fetching SYSDATE-3)**  SELECT \*  FROM (SELECT :"SYS\_B\_0"  || To\_char(DE.entryid) INDEXID,  :"SYS\_B\_1" INITIALLOADFLG,  DE.\*  FROM (SELECT de.\*,  Row\_number()  over(  PARTITION BY de.entryid  ORDER BY de.sync\_log\_id DESC) rn  FROM realtime\_sync de  WHERE de.createddt >= SYSDATE - :"SYS\_B\_2") de  WHERE rn = :"SYS\_B\_3")  WHERE "sync\_log\_id" > :1  ORDER BY "sync\_log\_id" ASC  / |

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Timeline

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated