Разное

- Компрессия и голденгейт
- Партиционируем и жмем огромную таблицу, с работающей репликацией

Компрессия и голденгейт

Представим ситуацию. У нас есть табличка, которая содержит информацию о заказах. Когда-то заказов в табличке были сотни, потом тысячи, десятки тысяч, и в какой-то момент оказалось, что заказов у нас миллиарды, а бизнес не хочет удалять исторические данные. При этом место у нас не резиновое и стоит дорого.

Тогда мы вспоминаем, что Oracle предоставляет неплохой инструмент для сжатия данных в таблицах — от классических Advanced/OLTP/Basic до специфических компрессий, которые есть на Exadata: FOR QUERY HIGH/LOW и FOR ARCHIVE HIGH/LOW. И это хорошо.

Но в таблицу идёт активная вставка данных (ведь у нас очень много покупателей и заказов), плюс на таблице выполняются операции обновлений, потому что заказы могут меняться, отменяться и т.д. Поэтому, если мы просто сожмём таблицу на нашей замечательной Exadata, мы, конечно, получим выигрыш в месте (возможно, даже 20-кратный), но при этом полностью «убьём» приложения, которые изменяют таблицу.

Остановимся на том, что наша таблица заполняется на неком хранилище репликатом GoldenGate. С таблицей работает координированный репликат, потому что если не распараллеливать вставки и обновления, то у нас начинает накапливаться отставание.

Как только мы сожмём таблицу, наш координированный репликат начнёт выдавать ошибку ORA-00060 (deadlock detected) и падать на взаимоблокировках: одновременно несколько потоков пытаются изменить один и тот же блок.

Но выход есть. Надо таблицу партиционировать. Мы предполагаем, что заказы старше определённой даты уже не могут быть отменены или изменены. Мы партиционируем таблицу по интервалу в один месяц, сжимаем «старые» партиции, которые не будут меняться, той компрессией, которую мы выбрали, а новые партиции оставляем как есть — несжатыми.

По мере заполнения партиций мы можем автоматически сжимать устаревающие партиции. Для этого нам надо предусмотреть, чтобы индексы (для работы репликата нужен уникальный индекс в любом случае) были локальными.

Пример реализации

Создадим таблицу заказов, партиционированную по диапазону дат с интервалом в один месяц:

```
CREATE TABLE orders (
    order_id      NUMBER PRIMARY KEY,
    order_date    DATE,
    customer_id   NUMBER,
    order_status  VARCHAR2(20),
    ...
)
PARTITION BY RANGE (order_date) INTERVAL (INTERVAL '1' MONTH)
(
    PARTITION p_historical VALUES LESS THAN (TO_DATE('2020-01-01','YYYY-MM-DD'))
);
```

Затем создадим локальный уникальный индекс на поле `order_id`, необходимый для работы репликации:

```
CREATE UNIQUE INDEX orders_pk ON orders (order_id, order_date) LOCAL;
```

Теперь можно сжать партицию, данные в которой уже не меняются. Например, для партиции за январь 2020 года:

```
ALTER TABLE orders
    MOVE PARTITION p_2020_jan
    PARALLEL 16
    COMPRESS FOR ARCHIVE HIGH;


SELECT 'ALTER INDEX ' || INDEX_NAME || ' REBUILD PARTITION ' || PARTITION_NAME || ' PARALLEL 16;'
FROM DBA_IND_PARTITIONS IP
WHERE IP.STATUS = 'UNUSABLE';
```

Это сжатие не затронет активные партиции, и репликат продолжит работать без конфликтов за блоки.

Таким образом, партиционирование в сочетании с выборочным сжатием позволяет эффективно экономить место, не жертвуя производительностью операций DML и не создавая проблем для параллельной репликации.

Партиционируем и жмем огромную таблицу, с работающей репликацией

Итак, мы решили, что будем партиционировать таблицу ORDERS, про которую шла речь в предыдущем примере. В принципе, мы можем просто создать рядом таблицу с атрибутом компрессии на уровне таблицы и сделать инсерт из нашей большой таблицы. Но так не получится. Инсерт будет работать очень долго, параллелиться он не будет, так как партиции, куда льются данные, сжаты, и в результате сначала отработают воркеры, которые делают селект из исходной таблицы, а потом мы будем ждать, когда данные в один поток зальются в приемник. Мы так делать не будем. Мы проверим, какие партиции у нас в принципе могут быть созданы с интервалом в месяц:

```
SELECT /*+ PARALLEL(t 32)  INDEX(t ORDERS_DATE_IDX) */
      COUNT(1),
      TO_CHAR(t.order_date, 'rrrr.mm')
FROM   orders t
GROUP BY TO_CHAR(t.order_date, 'rrrr.mm');
```

(у нас есть индекс по полю, которое будет впоследствии ключом партиционирования, и это хорошо)

Получим что-то вроде такого результата:

COUNT(1)	ORDER_DATE
14281	2013.12
16223	2014.01
15018	2014.02
14926	2014.03
22395	2014.04
18620	2014.05
18111	2014.06

27140	2014.07
234983	2014.08
492579	2014.09
...	...
436314181	2025.12
570153932	2026.01
346233850	2026.02

Теперь создадим таблицу, в которой есть все партиции, которые нам нужны. Атрибут компрессии мы выставлять не будем:

```
CREATE TABLE orders_parts (
    order_id      NUMBER PRIMARY KEY,
    order_date    DATE,
    customer_id   NUMBER,
    order_status  VARCHAR2(20),
    ...
)
PARTITION BY RANGE (order_date) INTERVAL (INTERVAL '1' MONTH)
(
  PARTITION P_BEFORE_2013 VALUES LESS THAN (DATE '2013-12-01'),
  PARTITION P_201312 VALUES LESS THAN (DATE '2014-01-01') TABLESPACE CUST_DATA,
  PARTITION P_201401 VALUES LESS THAN (DATE '2014-02-01') TABLESPACE CUST_DATA,
  PARTITION P_201402 VALUES LESS THAN (DATE '2014-03-01') TABLESPACE CUST_DATA,
  ........
  PARTITION P_202511 VALUES LESS THAN (DATE '2025-12-01') TABLESPACE CUST_DATA,
  PARTITION P_202512 VALUES LESS THAN (DATE '2026-01-01') TABLESPACE CUST_DATA,
  PARTITION P_202601 VALUES LESS THAN (DATE '2026-02-01') TABLESPACE CUST_DATA,
  PARTITION P_202602 VALUES LESS THAN (DATE '2026-03-01') TABLESPACE CUST_DATA
);
```

После чего нам нужно будет залить данные до уровня, когда мы уверены, что эти данные останутся неизменными.

Для этого используем DBMS_PARALLEL_EXECUTE...

```
SET SERVEROUTPUT ON SIZE UNLIMITED
SET LINESIZE 200
SET PAGESIZE 0
```

```
SET FEEDBACK OFF
SET VERIFY OFF

ALTER SESSION ENABLE PARALLEL DML;

-- 1. Создаём задачу
BEGIN
  DBMS_PARALLEL_EXECUTE.CREATE_TASK('LOAD_ORDERS_PARTS');
END;
/

-- 2. Создаём чанки (диапазоны дат) для всех партиций, кроме двух последних
BEGIN
  DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_SQL(
    task_name    => 'LOAD_ORDERS_PARTS',
    sql_stmt     =>
      'WITH part_info AS (
         SELECT partition_name,
                CASE
                  WHEN partition_name = ''P_BEFORE_2013'' THEN DATE ''1900-01-01''
                  ELSE TO_DATE(SUBSTR(partition_name, 3, 6) || ''01'', ''YYYYMMDD'')
                END AS start_date,
                CASE
                  WHEN partition_name = ''P_BEFORE_2013'' THEN DATE ''2013-12-01''
                  ELSE ADD_MONTHS(TO_DATE(SUBSTR(partition_name, 3, 6) || ''01'', ''YYYYMMDD''), 1)
                END AS end_date,
                partition_position
         FROM user_tab_partitions
         WHERE table_name = ''ORDERS_PARTS''
       )
       SELECT start_date, end_date
       FROM part_info
       WHERE partition_position <= (SELECT MAX(partition_position)-2 FROM part_info)
       ORDER BY partition_position',
    by_rowid => FALSE
  );
END;
/

-- 3. Запускаем задачу с параллельностью 16
```

```
BEGIN
  DBMS_PARALLEL_EXECUTE.RUN_TASK(
    task_name      => 'LOAD_ORDERS_PARTS',
    sql_stmt       => 'INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
    SELECT /*+ PARALLEL(orders,16) */ * FROM orders
    WHERE order_date >= :start_id AND order_date < :end_id',
    language_flag  => DBMS_SQL.NATIVE,
    parallel_level => 16
  );
END;
/

-- 4. Зачистка задач
exec DBMS_PARALLEL_EXECUTE.DROP_TASK('LOAD_ORDERS_PARTS');

-- 5. Посмотреть, что там с обработкой чанков
SELECT
       task_name,
       status
FROM
       user_parallel_execute_tasks;

SELECT
        chunk_id,
        status,
        start_rowid,
        end_rowid
FROM
        user_parallel_execute_chunks
WHERE
        task_name = 'LOAD_ORDERS_PARTS'
ORDER BY
        chunk_id;
```

Или, если не хочется по каким-то причинам использовать DBMS_PARALLEL_EXECUTE, всё то же самое сделаем вручную:

```
-- Партиция P_BEFORE_2013: все заказы до 2013-12-01
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders
```

```
WHERE order_date < DATE '2013-12-01';
COMMIT;

-- Партиция P_201401: декабрь 2013 (2013-12-01 <= order_date < 2014-01-01)
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders
WHERE order_date >= DATE '2013-12-01' AND order_date < DATE '2014-01-01';
COMMIT;

-- Партиция P_201402: январь 2014
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders
WHERE order_date >= DATE '2014-01-01' AND order_date < DATE '2014-02-01';
COMMIT;

-- Партиция P_201403: февраль 2014
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders
WHERE order_date >= DATE '2014-02-01' AND order_date < DATE '2014-03-01';
COMMIT;

-- ... аналогично для всех промежуточных месяцев до 2025 ...

-- Партиция P_202511: ноябрь 2025
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders
WHERE order_date >= DATE '2025-11-01' AND order_date < DATE '2025-12-01';
COMMIT;

-- Партиция P_202512: декабрь 2025
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders
WHERE order_date >= DATE '2025-12-01' AND order_date < DATE '2026-01-01';
COMMIT;

-- Две последнии партиции не трогаем
```

Вариант с ручным переносом может быть полезен в случае, если места у нас осталось впритык и как только мы перенесли одну партицию, мы её сразу же компрессируем.

Теперь собственно мы берём наши партиции и добавляем в них компрессию:

```
ALTER TABLE ORDERS MOVE PARTITION P_BEFORE_2013 COMPRESS FOR QUERY HIGH PARALLEL 16;
ALTER TABLE ORDERS MOVE PARTITION P_201312 COMPRESS FOR QUERY HIGH PARALLEL 16;
ALTER TABLE ORDERS MOVE PARTITION P_201401 COMPRESS FOR QUERY HIGH PARALLEL 16;
ALTER TABLE ORDERS MOVE PARTITION P_201402 COMPRESS FOR QUERY HIGH PARALLEL 16;
.... и так далее
```

После того как всё сжато, нам осталось долить данные в партиции, которые будут активно изменяться в текущем периоде. И тут у нас есть два подхода. Если мы можем позволить себе небольшой даунтайм (отключение записи в таблицу) на время, пока мы переливаем данные в последние партиции и строим нужные индексы, то, в принципе, следующий этап совсем прост. Нам нужно остановить репликат, по аналогии с тем, как мы заливали предыдущие данные, долить оставшиеся, построить локальный индекс, сделать переименование таблицы (возможно, придётся откомпилировать зависимые объекты) и снова запустить репликат.

В случае, если даунтайм, должен быть минимальный (минуты) мы сделаем по другому:

1 У нас уже есть репликат, который пишет данные в исходную таблицу, и нам нужно засечь SCN, начиная с которого мы будем заполнять нашу новую таблицу. Для этого мы создадим новый репликат с маппингом, аналогичным исходному, например так:

```
--Наш исходный репликат, потребляет трэйл ./dirdat/ad
ADD REPLICAT RORDPART, EXTTRAIL ./dirdat/ad checkpointtable GGATE.CHECKPOINTS

--На этом же этапе, подготовим репликаты, для синхронной остановки через  eventactions( stop )
--Создадим в базах - источнике и приемнике, таблицу:
create table GGATE.STOPACTION
(
  id        number GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  stopdate  date
);
--Добавим логирвоане на эту таблицу:
ALTER TABLE GGATE.STOPACTION ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;

-- В экстратор, который поставляет трэйл - файлы для репликатов, пропишем:
TABLE GGATE.STOPACTION, tokens( commit_scn=@GETENV('TRANSACTION', 'CSN') );

--Перезапустим экстратор

--Добавим конструкцию в оба репликата:
MAP GGATE.STOPACTION, target GGATE.STOPACTION, colmap(usedefaults, stop_scn=@TOKEN('commit_scn')),
eventactions( stop );
```

2 Пропишем конфиг, аналогичный исходному репликату, в наш новый репликат, оставив в новом репликате маппинг только на одну таблицу ORDER_PARTS.

3 Остановим исходный репликат, и засечем SCN

```
SELECT CURRENT_SCN FROM V$DATABASE;
```

4 Переставим наш новый репликат на тот RBA, на котором у нас остановлен основной репликат:

```
--Посмотрим где у нас остановлен основной репликат
GGSCI (ggate.local.net) 2> info RORDERS

Replicat   RORDERS   Last Started 2026-02-24 21:03   Status STOPPED
Checkpoint Lag       00:02:05 (updated 00:00:02 ago)
Process ID           2394620
Log Read Checkpoint  File ./dirdat/ad002328787
                     2026-02-25 12:22:12.000000  RBA 407552118

--Переставим новый репликат на нужный RBA
GGSCI (ggate.local.net) 3> ALTER REPLICAT RORDPART EXTSEQNO 2328787 EXTRBA 407552118

--Запустим основной репликат
GGSCI (ggate.local.net) 4> START RORDERS
```

Теперь новый репликат, когда будет запущен, начнёт писать изменения ровно с того места, где был остановлен основной репликат и для которого мы засекли SCN. Это первое время простоя репликации, и мы, скорее всего, уложились в пару минут.

5 Перенесём данные из исходной таблицы, который у нас остались:

```
-- Партиция P_202601: январь 2026
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders AS OF SCN (<значени SCN, которое мы засекли>)
WHERE order_date >= DATE '2026-01-01' AND order_date < DATE '2026-02-01';
COMMIT;
-- Партиция P_202602: февраль 2026
INSERT /*+ APPEND PARALLEL(16) */ INTO orders_parts
SELECT /*+ PARALLEL(orders,16) */ * FROM orders AS OF SCN (<значени SCN, которое мы засекли>)
WHERE order_date >= DATE '2026-02-01' AND order_date < DATE '2026-03-01';
COMMIT;
```

6 Проиндексируем таблицу:

```
CREATE UNIQUE INDEX orders_parts_idx on orders_parts(order_id, order_date) local parallel 16;
alter index orders_parts_idx noparallel;
```

7 Запустим репликат, который мы создали:

```
GGSCI (gghubs.local.net) 3> START RORDPART
```

8 После того, как репликат догонит таблицу по данным, мы остановим оба репликата синхронно. Для этого у нас есть несколько вариантов:

- остановка пампа\экстратора, который формирует трэйл. Это самое простое, но, возможно, что экстратор, после рестарта, будет долго выполнят рекавер
- остановка, через событие eventactions(stop) самый подходящий для нас вариант

9 Остановка событием. Для этого, просто, делаем инсерт в таблицу, которую мы создавали на источнике:

```
insert into GGATE.STOPACTION (STOPDATE) values (SYSDATE);
COMMIT;
```

Оба репликата, синхронно остановтся

10 Переименовываем таблицы:

```
--Преименуем таблицы
ALTER TABLE ORDERS RENAME TO ORDERS_BACK;
ALTER TABLE ORDERS_PARTS RENAME TO ORDERS;

--Проверим, что у нас развалились зависимые пакеты и сделаем рекмпиляцию пакетов, которые развалились
ALTER PACKAGE ... COMPILE
```

11 Мы можем запустить 'старый' репликат и удалить 'новый'

```
GGSCI (gghubs.local.net) 3> START RORDERS

GGSCI (gghubs.local.net) 3> STOP RORDPART

--Для того, чтобы в базе не осталось ошметков в таблице чекпоинтов, перед удалением, логинимсяв базу
GGSCI (gghubs.local.net) 3> DBLOGIN USERIDALIAS <алиас>
GGSCI (gghubs.local.net) 3> DELETE RORDPART
```

Итого, у нас теперь есть партиционированная таблица, с пожатыми историческими партициями и общее время простоя репликации - несколько минут.