ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Arenadata DB для Разработчиков

Часть 5

ADB – аналитическая СУБД для больших данных

Аспекты миграции из Oracle Database:
Расширения. Запросы. Функции.

ETL vs ELT:
Принципы построения Хранилища.
Обновление справочников и таблиц.

Сессия вопросов и ответов.

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Ключевые аспекты миграции из Oracle в Greenplum

Факторы, которые необходимо учесть про миграции

1. Некоторые выражения могут переноситься один в один, другие необходимо адаптировать.
2. Целевая СУБД может не поддерживать какие-либо выражения и синтаксические конструкции.
3. В некоторых случаях достаточно изменить синтаксис, в других необходимо менять логику кода.
4. Некоторые паттерны хорошо работают в СУБД-источнике, но приводят к значительной деградации производительности в целевой СУБД.

Расширение для совместимости с Oracle

В Greenplum 6 для обеспечения совместимости с Oracle есть расширение orafce:

- https://gpdb.docs.pivotal.io/6-4/ref_guide/modules/orafce_ref.html
- https://github.com/greenplum-db/gpdb/tree/master/gpcontrib/orafce/doc/orafce_documentation
- Установка: CREATE EXTENSION orafce;
- Функции размещены в наборе схем.
- Может использоваться с Ora2pg.

В предыдущих версиях GreenPlum использовался пакет orafunc:

- https://gpdb.docs.pivotal.io/5270/install_guide/install_oracle_functions.html
- Установка: $ psql -d testdb -f $GPHOME/share/postgresql/contrib/orafunc.sql
- Функции размещены в схеме oracompat.

Ограничения Greenplum Database SQL

1. По курсору можно проходить только вперед.
2. В операторе `CREATE TABLE` для hash-distributed таблиц при объявлении `UNIQUE` или `PRIMARY KEY` необходимо указывать все поля ключа дистрибуции. Объявление `UNIQUE` или `PRIMARY KEY` не поддерживается для таблиц с ключом распределения `randomly`.
3. При использовании конструкции `CREATE UNIQUE INDEX` необходимо указывать все поля ключа дистрибуции. Объявление `CREATE UNIQUE INDEX` не поддерживается для таблиц с ключом распределения `randomly`. Использование индексов в целом не эффективно в Greenplum в большинстве случаев.
4. `VOLATILE` и `STABLE` функции не могут быть выполнены на сегментах.
5. Внешние ключи не ограничивают вставку и изменение данных, но пользователи могут создавать и отслеживать внешние ключи в system catalog.

Ограничения Greenplum Database SQL

6. Конструкции для работы с последовательностями `CURRVAL`, `NEXTVAL` и `LASTVAL` не поддерживаются. Вместо них следует использовать функции `currval()`, `nextval()`, `setval()`.
7. Конструкиця `CREATE PROCEDURE` не поддерживается, вместо неё следует использовать `CREATE FUNCTION`, возвращающую `void`.
8. Конструкция `MERGE` не поддерживается. http://www.greenplumguru.com/?p=104
9. `FETCH FIRST` or `FETCH NEXT` конструкции не поддерживаются, вместо используются конструкции `LIMIT` and `LIMIT OFFSET`.
10. `TRUNCATE TABLE` конструкция не поддерживает опции `CONTINUE IDENTITY` и `RESTART IDENTITY`.

Ограничения Greenplum Database SQL

12. Триггеры не поддерживаются. Вместо используйте правила (rules):

```
-- Создание rule для вставки строк в таблицу b2001, когда пользователь пытается вставить строки в таблицу rank:

    CREATE RULE b2001 AS ON INSERT TO rank
    WHERE gender='M' and year='2001'
    DO INSTEAD
    INSERT INTO b2001 VALUES (NEW.id, NEW.rank,
    NEW.year, NEW.gender, NEW.count);
```

13. Packages не поддерживаются, используйте схемы для организации функций в группы.

14. Package-level variables не поддерживаются, храните значения переменных в temporary tables сессии.

ARENADATA

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Аспекты миграции из Oracle. Запросы

Update поля в таблице

Oracle:

```
UPDATE table_1
   SET value_1 = 500
   WHERE value_2 IN (‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’);
```

- Если предполагается обновление значительного объема записей (более нескольких миллионов), лучше создать дополнительную таблицу и вставлять обновленные записи в новую таблицу вместо обновления записей в исходной.
- Важно помнить, что после основных вставок необходимо догрузить в новую таблицу те записи, которые не были затронуты при обновлении.

Update поля в таблице

Greenplum:

```
--Вставляем в таблицу только изменившиеся записи
INSERT INTO test.table_1_dif (id, val_1, val_2)
 select id, 500 as val_1, val_2
   FROM test.table_1
      WHERE val_2 IN (‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’);
--Вставляем в таблицу оставшиеся – не изменившиеся записи
INSERT INTO test.table_1_dif
   SELECT A.*
   FROM table_1 A
   LEFT JOIN table_1_dif B
      ON (A.id = B.id)
   WHERE B.id IS NULL;
```

Работа с датами

Типы дат:

- В Oracle есть два основных типа даты – это date и timestamp. При этом date хранит в себе дату + время, а timestamp – дату, время до миллисекунд, временную зону.
- В Greenplum же есть три основных типа даты – это date, timestamp with time zone, timestamp without timezone. При этом тип date хранит только дату без времени. Поэтому при конвертации из Oracle в Greenplum надо помнить о том, что тип date необходимо мигрировать в тип timestamp(0) without time zone.

Oracle:

```
SELECT SYSDATE FROM dual;

SELECT SYSDATE -1 FROM dual;
```

Greenplum:

```
SELECT NOW();

SELECT now()-INTERVAL '1 day';
SELECT current_date - 1;
```

Работа с датами

Oracle:

```
SELECT TRUNC(SYSDATE) FROM dual;
```

Greenplum:

```
SELECT now()::DATE::TIMESTAMP; -- Если требуется получить timestamp

SELECT now()::DATE; -- Если требуется получить date
```

Разность между датами

Отличия в реализации разности дат:

- В Oracle результатом разности между датами является количество дней между этими датами типа number, т.е. число с дробной частью.
- В Greenplum результатом является тип interval, который содержит дни, часы, минуты и т.д.. Поэтому его нельзя сравнивать с числами без должного приведения к значению, как это практикуют при работе с Oracle.

Oracle:

```
SELECT SYSDATE-(SYSDATE-100) FROM dual;
```

Greenplum:

```
SELECT (EXTRACT(epoch FROM (current_date - (current_date-INTERVAL '100 days'))))/86400)
```

Подзапросы в «FROM» должны иметь алиас

Oracle:

```
CREATE OR REPLACE VIEW V_with_WITH
(c1,c2) AS
WITH IDS AS (
   SELECT t1.id
   FROM table_1 t1)
SELECT c1,c2 FROM (SELECT id c1, t2.id c2
                   FROM table_2 t2,
ids
                   WHERE id = t2.id)
```

Greenplum:

```
CREATE OR REPLACE VIEW V_with_WITH (c1,c2)
AS
WITH IDS AS (
   SELECT s.id
   FROM table_1 t1)
SELECT c1,c2 FROM (SELECT id AS c1, my_id AS
c2
                   FROM table_2 t2, IDS
                   WHERE id = my_id) AS
tabAl;
```

Коррелированный подзапрос в блоке SELECT

Oracle:

```
SELECT T1.a,

      (SELECT COUNT(DISTINCT T2.z) FROM t2
WHERE t1.x = t2.y) dt2

FROM t1;
```

В целях повышения производительности при использовании оптимизатора Postgres необходимо переписать запрос с двумя соединениями, сначала произвести INNER JOIN к таблице t1, а затем соединение LEFT JOIN к t1.

Greenplum:

```
SELECT t1.a, dt2 FROM t1
       LEFT JOIN
        (SELECT t2.y AS csq_y, COUNT(DISTINCT t2.z) AS dt2
              FROM t1, t2 WHERE t1.x = t2.y
              GROUP BY t1.x) -- INNER JOIN
       ON (t1.x = csq_y);
```

Неподдерживаемые конструкции в TRUNCATE

Oracle:

```
TRUNCATE TABLE table_1 REUSE STORAGE;
```

Greenplum:

```
TRUNCATE test.table_1;

TRUNCATE TABLE test.table_1;
```

Когда необходимо переписывать:

Всегда вследствие несовместимости синтаксиса.

Как необходимо переписывать:

Убираем все неподдерживаемые конструкции.

Запросы с использованием ROWNUM

Oracle:

```
SELECT COUNT(*)
INTO vCnt
FROM table_1 t1
WHERE t1.id > vMinId
AND ROWNUM <= 1;
```

Greenplum:

```
SELECT COUNT(*)
INTO vCnt
FROM test.table_1 t1
WHERE t1.id > vMinId
LIMIT 1;
```

Когда необходимо переписывать:

Всегда, так как синтаксис не совместим.

Как необходимо переписывать:

Необходимо переписывать с использованием LIMIT.

Выражением SELECT в UPDATE в блоке SET

Oracle:

```
UPDATE  test.table_1 t1
   SET (value_1, value_2) =
   (SELECT value_1, value_2
       FROM test.table_2 t2
       WHERE t2.id = t1.id)
```

Greenplum:

```
UPDATE test.table_1 t1
SET value_1 = t2.val1,
value_2 = t2.val2
FROM  (SELECT value_1 as val1,
              value_2 as val2
    FROM test.table_2) t2
WHERE t2.id = t1.id;
```

Когда необходимо переписывать:

- Всегда вследствие несовместимости синтаксиса.

Как необходимо переписывать:

- Необходимо переписать с использованием UPDATE FROM.

Самый простой вариант – вынести соответствующий SELECT в подзапрос в блок FROM.

При этом важно не забыть вынести условие из SELECT’а в блок WHERE выражения UPDATE.

NVL

Oracle:

```
SELECT NVL(ISNUMBER,0) FROM table_1 t1;
```

Greenplum:

```
SELECT COALESCE(ISNUMBER, CAST(0 AS NUMERIC))
FROM table_1 t1;
```

Функция decode

Oracle:

```
WITH table_1 AS
      (SELECT 1 AS ID FROM dual
          UNION ALL
          SELECT NULL FROM dual)
SELECT ID, DECODE (DECODE (ID, NULL, 1,
NULL), NULL, 'Ц', 'Ф')
   FROM table_1;
```

Когда необходимо переписывать:

Необходимо переписывать, если есть вложенность функций decode друг в друга.

Greenplum:

```
WITH table_1 as
   (SELECT 1::NUMERIC AS ID
    UNION ALL
    SELECT NULL::NUMERIC)
SELECT ID,
-- Некорректная работа
   DECODE(DECODE(ID, NULL, 1, NULL), NULL,
'Ц', 'Ф'),
-- Корректная работа
   CASE WHEN DECODE(ID, NULL, 1, NULL) IS
NULL THEN 'Ц' ELSE 'Ф' END        FROM
table_1;
```

first_value()

Функция first_value()в Oracle и Greenplum синтаксически одинакова, однако, существуют нюансы в работе этой функции при наличии сортировки записей по окну (windowing) - в присутствии конструкции ORDER BY. Поэтому при миграции подобных конструкций в GP дополнительно проверяем на NULL и вводим сортировку:

Oracle:

```
Select  ID,
        SRT,
        first_value (val)  over (
partition by ID
order by SRT
) as  FV
FROM TEST
```

Greenplum:

```
Select ID,
        SRT,
        first_value(val) over(
                        partition by ID
                        order by SRT,
decode(val, NULL, 0, 1) desc, val) as FV
  FROM TEST
```

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Аспекты миграции из Oracle. Функции

Type is table of number

Oracle:

```
CREATE PROCEDURE proc_8(a IN NUMBER)IS
TYPE tB IS TABLE OF NUMBER INDEX BY
BINARY_INTEGER;
   b tB;
BEGIN
   SELECT value
   BULK COLLECT INTO b
   FROM t1;
END PROCEDURE;
```

Greenplum:

```
CREATE OR REPLACE FUNCTION proc_8(IN a NUMERIC)
RETURNS VOID
AS $procedure$
DECLARE

BEGIN
   DROP TABLE IF EXISTS tB;
   CREATE TEMPORARY TABLE tB (n NUMERIC);
   INSERT INTO tB
      SELECT value FROM t1;
   RETURN;
END; $procedure$
LANGUAGE plpgsql;
```

FOR loop with table

Oracle:

```
CREATE PROCEDURE proc_9(a IN NUMBER) IS
    TYPE tB IS TABLE OF NUMBER INDEX BY
BINARY_INTEGER;
      b tB;
      n NUMBER;
BEGIN
   SELECT value
      BULK COLLECT INTO b
      FROM table_1;
   n:=0;
   FOR i IN b.first .. b.last LOOP
      n := n + b(i);
   END LOOP;
END PROCEDURE;
```

Greenplum:

```
CREATE OR REPLACE FUNCTION proc_9(IN a NUMERIC)
RETURNS VOID
   AS $procedure$
   DECLARE
      n NUMERIC;
      i RECORD;
BEGIN
   DROP TABLE IF EXISTS tB CASCADE;
   CREATE TEMPORARY TABLE tB (z NUMERIC) WITH
 OIDS;
   INSERT INTO tB
   SELECT value FROM table_1;
   n := 0;
   FOR i IN (SELECT * FROM tB) LOOP
      n := n + i.z;
   END LOOP;
   RETURN;
END; $procedure$
LANGUAGE plpgsql;
```

Delete from collection

Oracle:

```
CREATE PROCEDURE proc_10
  IS
    a NUMBER := 0;
    TYPE tB IS TABLE OF var_table%ROWTYPE;
    b tB;
    TYPE tElement IS RECORD(
       x NUMBER,
       y NUMBER);
    e tElement;
    TYPE tC IS TABLE OF c INDEX BY BINARY_INTEGER;
    c tC;
BEGIN
    c.DELETE;
    e.DELETE;
    c.DELETE(4);
END;
```

Greenplum:

```
CREATE TYPE tElement AS(x NUMERIC, y NUMERIC);
CREATE OR REPLACE FUNCTION proc_10()
RETURNS VOID
AS $procedure$
DECLARE
   a NUMERIC DEFAULT 0;
   e tElement;
BEGIN
DROP TABLE IF EXISTS tC CASCADE;
CREATE TEMPORARY TABLE tC
   (SWC_Element tElement, SWC_index serial8);
DROP TABLE IF EXISTS tB CASCADE;
CREATE TEMPORARY TABLE tB AS SELECT * FROM var_table
WITH NO DATA;
   DELETE FROM c;
   DELETE FROM tB;
   DELETE FROM c WHERE SWC_index = 4;
   RETURN;
END; $procedure$
LANGUAGE plpgsql;
```

Last First element of collections

Oracle:

```
CREATE PROCEDURE proc_11 IS
   a NUMBER := 0;
   z NUMBER := 0;
   TYPE tElement IS RECORD(
      x NUMBER,
      y NUMBER
   e tElement;
   TYPE tB IS TABLE OF tElement
 INDEX BY
 BINARY_INTEGER;
   b tB;
BEGIN
   a := b.first;
   z := b.last;
END;
```

Greenplum:

```
CREATE TYPE tElement AS(x NUMERIC, y NUMERIC);
CREATE OR REPLACE FUNCTION proc_11()
RETURNS VOID
AS $procedure$
DECLARE
   a NUMERIC DEFAULT 0;
   z NUMERIC DEFAULT 0;
   e tElement;
BEGIN
DROP TABLE IF EXISTS tB CASCADE;
   CREATE TEMPORARY TABLE tB
      (Element tElement, SWC_index serial8)
       WITH OIDS;
   a := (SELECT min(SWC_index) FROM tB);
   z := (SELECT max(SWC_index) FROM tB);
   RETURN;
END; $procedure$
LANGUAGE plpgsql;
```

Assign value null to variable is TYPE IS RECORD.

Oracle:

```
CREATE PROCEDURE proc_12 IS
   a NUMBER := 0;
   TYPE tElement IS RECORD(
      x NUMBER,
      y NUMBER);
   e tElement;
BEGIN
   e := NULL;
END;
```

Greenplum:

```
CREATE TYPE tElement AS(x NUMERIC,
   y NUMERIC);
CREATE OR REPLACE FUNCTION proc_12()
RETURNS VOID
AS $procedure$
DECLARE
   a  NUMERIC DEFAULT 0;
   e  tElement;
BEGIN
   SELECT NULL INTO e;
   RETURN;
END; $procedure$
LANGUAGE plpgsql;
```


ARENADATA

Copyright
arenadata.io

if record.exists

Oracle:

```
CREATE PROCEDURE proc_17 IS
   n NUMBER := 0;
   TYPE tElement IS RECORD(
      x NUMBER,
      y NUMBER);
   e tElement;
   TYPE tB IS TABLE OF tElement INDEX BY BINARY_INTEGER;
   b tB;
BEGIN
   IF b.EXISTS(4) THEN
      dbms_output.put_line('Exists');
   ELSE
      dbms_output.put_line('Not exists');
   END IF;
END;
```

if record.exists

Greenplum:

```
CREATE TYPE tElement AS(
      x NUMERIC,
      y NUMERIC);
CREATE OR REPLACE FUNCTION proc_17()
RETURNS VOID
AS $procedure$
DECLARE
   n NUMERIC DEFAULT 0;
   e tElement;
```

```
BEGIN
   DROP TABLE IF EXISTS tB;
      CREATE TEMPORARY TABLE tB
      (SWC_Element tElement,
       SWC_index serial8);
   IF (SELECT SWC_Element FROM tB WHERE
SWC_index = 4) THEN
      RAISE NOTICE 'Exists';
   ELSE
      RAISE NOTICE 'Not Exists';
   END IF;
END; $procedure$
LANGUAGE plpgsql;
```

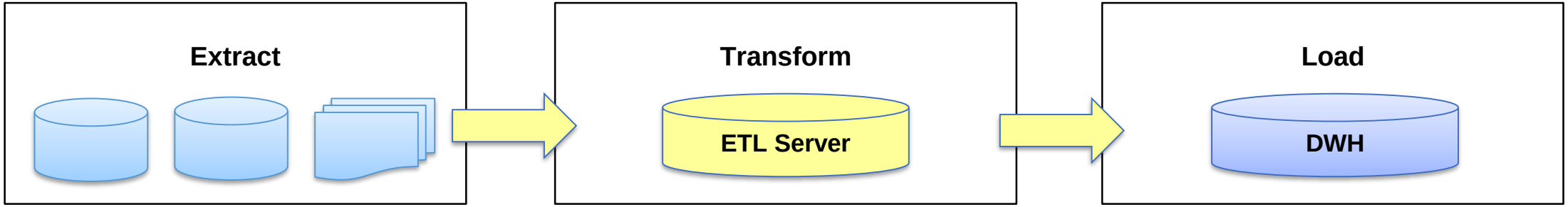
ARENADATA

Корпоративная платформа хранения
и обработки больших данных

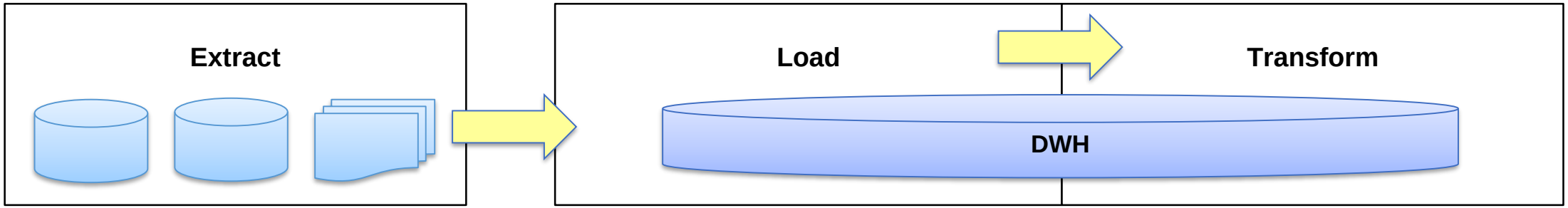
ETL и ELT

ETL и ELT подходы

ETL – классический подход: выгрузка с источника, обработка на сервере ETL, загрузка данных в приемник.



ELT – загружает сырые данные из источника сразу в целевую систему, преобразование данных происходит уже на стороне СУБД-приемнике.



ETL и ELT подходы

Плюсы ELT:

- Используются мощности распределенного кластера MPP систем.
- Параллельная обработка данных.
- Данные быстрее загружаются в DWH в виде stage-слоя.
- Нет необходимости в дополнительной инфраструктуре ETL-контура.

Минусы ELT:

- Пока еще не так много промышленных ETL инструментов, поддерживающих ELT (pushdown-оптимизацию).
- Как следствие, процесс загрузки зачастую превращается в набор SQL инструкций.
- Возможная конкуренция ELT-процессов за ресурсы кластера с бизнес-процессами.

Обновление справочников (SCD1)

FULL – полная перезапись справочника

До изменений:

id	ФИО
1	Иванов Иван
2	Петров Петр

После изменений:

id	ФИО
1	Иванов Александр
3	Сидоров Сидр

- DELETE - INSERT – относительно медленно, но без блокировок на чтение.
- TRUNCATE - INSERT – быстрое обновление, но с блокировкой ACCEESS EXCLUSIVE.

Обновление справочников (SCD1)

FULL – полная перезапись справочника

- EXCHANGE PARTITION – быстрое обновление, с отложенной ACCEESS EXCLUSIVE блокировкой.

Данные подготавливаются во временной таблице и затем подменяется партиция в целевой таблице через выражение exchange partition:

```
Create table stage (id int, part int);
Create table target (id int, part int) partition by list (part) (default partition main);
set gp_enable_exchange_default_partition to 'on';
alter table target EXCHANGE DEFAULT PARTITION with table stage;
```

Обновление справочников (SCD1)

DELTA MERGE – обновление справочника дельтой с источника

- Выражение MERGE в Greenplum не поддерживается.
- UPDATE – это DELETE – INSERT.

- Пример реализации через delete-insert (есть удаления в дельте):

  sp_dim_client_rnd_merge_1

До изменений:

id	ФИО
1	Иванов Иван
2	Петров Петр

После изменений:

id	ФИО
1	Иванов Александр
3	Сидоров Сидр

Обновление справочников (SCD1)

DELTA MERGE – обновление справочника дельтой с источника

- Пример реализации через ранжирование (нет удалений на источнике):

  sp_dim_client_rnd_merge_2

До изменений:

id	ФИО
1	Иванов Иван
2	Петров Петр

После изменений:

id	ФИО
1	Иванов Александр
2	Петров Петр
3	Сидоров Сидр

- Также вариант реализации можно посмотреть по ссылке

  http://www.greenplumguru.com/?p=104

Обновление исторических справочников (SCD2)

Закрываем период у измененных и удаленных на источнике записей, добавляем новые записи с открытым периодом.

До изменений:

id	ФИО	Дата с	Дата по
1	Иванов Иван	01.04.2020	31.12.2100
2	Петров Петр	01.04.2020	31.12.2100

После изменений:

id	ФИО	Дата с	Дата по
1	Иванов Иван	01.04.2020	19.05.2020
1	Иванов Александр	20.05.2020	31.12.2100
2	Петров Петр	01.04.2020	19.05.2020
3	Сидоров Сидр	20.05.2020	31.12.2100

Примерный алгоритм:

- Создаем временную таблицу с полями из целевой и stage-таблицы
- FULL OUTER JOIN запросом по целевой и stage-таблице определяем изменившиеся/удаленные/добавленные записи
- Добавляем ранжирование для исключения дублей на источнике
- Удаляем изменившиеся и удаленные записи в целевой таблице (с открытым периодом).
- Вставляем новые записи с открытым периодом; вставляем удаленные/измененные записи с закрытым периодом.
- Еще раз вставляем изменившиеся записи, но уже новые значения с открытым периодом.

Обновление исторических справочников (SCD2)

При данном алгоритме **распределение данных** в целевой и stage таблицах может оказывать существенное влияние на больших объемах данных, т.к. используется JOIN для определения изменений.

При частом изменении данных в справочнике имеет смысл сделать историческую таблицу партиционированной по полю date_to, это позволит:

- При определении расхождений в целевой и stage таблицах сканировать только дефолтную партицию с открытым периодом в целевой таблице.
- При соединении таблицы фактов и исторического справочника есть вероятность просканировать сильно меньшее количество данных в справочнике, т.к. в большинстве случаев запросы выполняются по текущим/недавним данным.

Пример можно посмотреть в учебной базе:

sp_dim_account_hist_rnd_merge

Обновление исторических справочников (SCD2)

Псевдо-код реализации MERGE алгоритма для исторического справочника

```
CREATE TEMPORARY TABLE tmp_foo(t_id integer
 ,t_val text
 ,t_hashdiff uuid
 ,date_from date
 ,date_to date
 ,s_id integer
 ,s_val text
 ,s_hashdiff uuid
 ,op_type varchar(1)
 ,rn int) ON COMMIT DROP DISTRIBUTED RANDOMLY;
```

Обновление исторических справочников (SCD2)

Псевдо-код реализации MERGE алгоритма для исторического справочника

```
        INSERT INTO tmp_foo
        SELECT t.*,s.*
          -- определяем тип операции
                ,case
                    when s.hashdiff<>t.hashdiff then 'U'
                    when t.acccode is null then 'I'
                    when s.acccode is null then 'D'
                 end op_type
                ,row_number() over (partition by s.id order by s.load_date_time
desc) as rn -- ранжируем записи на источнике, чтобы потом исключить дубли
        FROM dim_foo_hist t -- target
        FULL OUTER JOIN stg_foo s ON s.id=t.id -- stage
        WHERE t.date_to ='12.31.2100‘ -- партиция по date_to сокращает скан
целевой исторической таблицы
        AND (t.hashdiff<>s.hashdiff  OR t.id IS NULL OR s.id IS NULL);
```

Обновление исторических справочников (SCD2)

Псевдо-код реализации MERGE алгоритма для исторического справочника

```
-- Удаляем удаленные/измененные записи
      DELETE FROM dim_foo_hist t
      USING tmp_foo s
      WHERE t.id=s.t_id AND s.op_type IN ('U','D')
      AND t.date_to = '12.31.2100'; -- скан только по default партиции
```

```
-- вставляем новые записи + старые измененные/удаленные записи с закрытым интервалом
INSERT INTO dim_foo_hist
SELECT case
           when op_type = 'U' then t_id
           when op_type = 'I' then s_id
           when op_type = 'D' then t_id
        end as id
       ,case
           when op_type = 'U' then t_val
           when op_type = 'I' then s_val
           when op_type = 'D' then t_val
        end as val
       ,case
           when op_type = 'U' then date_from
           when op_type = 'I' then now()
           when op_type = 'D' then date_from
        end as date_from
       ,case
           when op_type = 'U' then now() - interval '1' day
           when op_type = 'I' then null
           when op_type = 'D' then now() - interval '1' day
        end as date_to
FROM tmp_foo
WHERE rn=1; -- Учитываем только одну запись, в случае дублей на источнике
```

Обновление исторических справочников (SCD2)

Псевдо-код реализации MERGE алгоритма для исторического справочника

```
--Делаем еще одну вставку - это уже новая запись для U с открытым интервалом
      INSERT INTO dim_foo_hist
      SELECT s_id,
             s_val,
             now() as date_from,'12.31.2100' as date_to
      FROM tmp_foo
      WHERE op_type='U' AND rn=1
```

Обновление исторических справочников (SCD4)

Insert only (SCD4)

- Исторический справочник состоит из 2 таблицы:

  Таблица с текущими данными (с открытым периодом)

  Таблица с историческими данными (с закрытыми периодами)

Текущие данные:

id	ФИО	Дата с	Дата по
1	Иванов Александр	20.05.2020	31.12.2100
3	Сидоров Сидр	20.05.2020	31.12.2100

Исторические данные:

id	ФИО	Дата с	Дата по
1	Иванов Иван	01.04.2020	19.05.2020
2	Петров Петр	01.04.2020	19.05.2020

- Создается представление, объединяющее эти две таблицы.
- При обновлении данных удаленные и измененные в источнике данные добавляются в историческую таблицу с закрытым периодом
- Таблица с текущими значениями перезаписывается (либо обновляется, в зависимости от объема обновляемых данных по отношению к полному объему таблицы).

Обновление исторических справочников (SCD4)

```
-- добавляем в историческую таблицу с закрытым периодом измененные и удаленные на источнике данные
insert into dim_foo_hist
select id
        ,val
        ,date_from
        ,now()-interval '1' day as date_to
from dim_foo t -- current data
left join stg_foo s on s.id=t.id -- stage
where t.hashdiff<>s.hashdiff -- измененные данные
or s.id is null; -- удаленные на источнике данные
```

Обновление исторических справочников (SCD4)

```
-- полностью обновляем таблицу с текущими значениями
	delete from dim_foo;
	insert into dim_foo
	select *
		,now() as date_from
		,'12.31.2100'::date
	from stg_foo;

	create view vw_foo_hist as
	select * from dim_foo
	union all
	select * from dim_foo_hist;
```

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Самостоятельная работа. Загрузка данных из Oracle

План. Входные данные

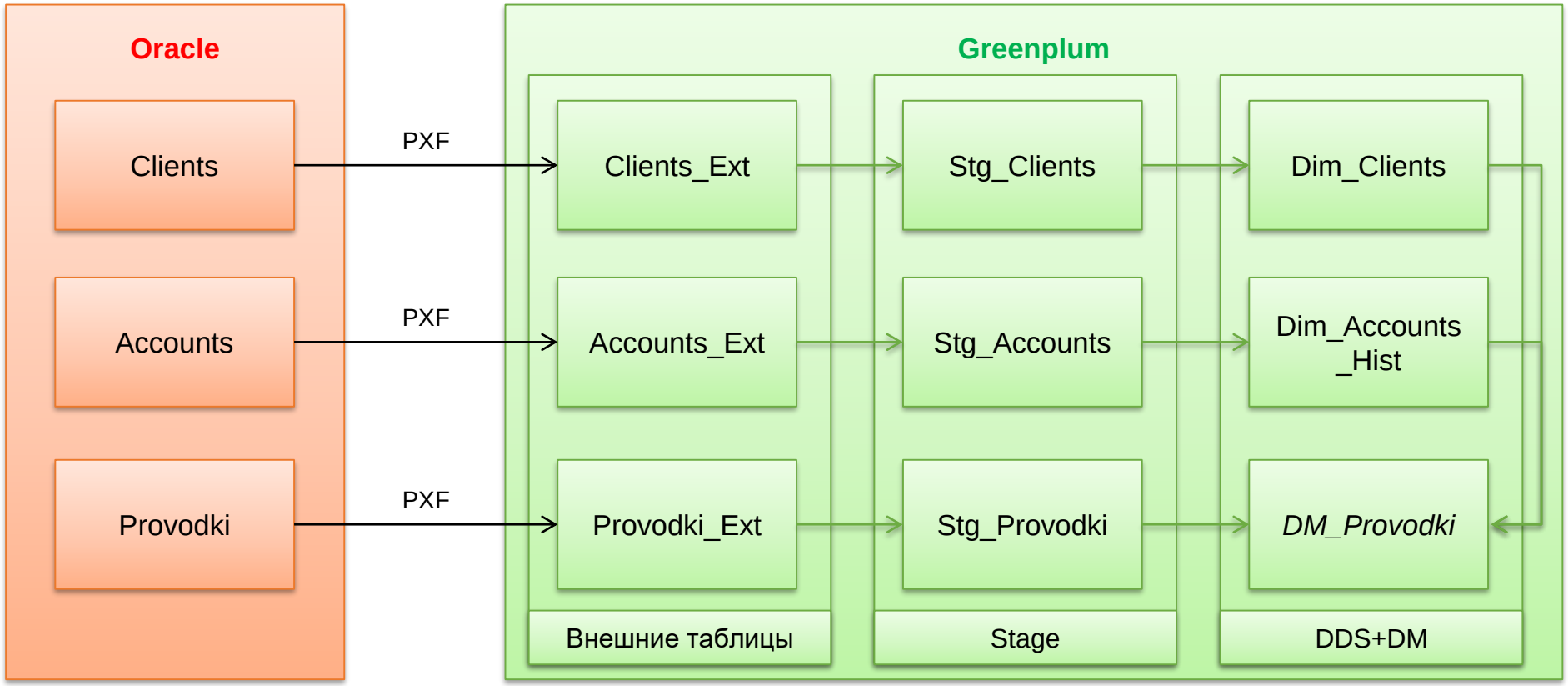
В Oracle хранятся 3 таблицы:

- Клиенты (1 млн. записей);
- Счета (3 млн.);
- Проводки (транзакции – 1 млн. в день).

В DWH Greenplum несколько слоев:

- Слой внешних таблиц для загрузки из Oracle.
- Stage слой с первичными данными (плюс hash по полям).
- DDS слой: справочники и витрина с транзакциями.

План. Схема объектов и взаимодействия



План. Описание схемы и порядок действий

Загрузите данные в DWH ADB с помощью PXF:

- Клиенты в ADB будут перезаписываться полностью.
- Счета будут реализованы в виде исторического справочника SCD2.
- Фактовая таблица проводок будет обновляться с помощью `EXCHANGE PARTITION`.

В ваших БД реализованы процедуры загрузки для таблиц с `randomly` распределением. Для клиентов также есть версия для распределения `replicated`.

Необходимо будет создать вверсии для таблиц с распределением по ключу.

Выполнить все операции и зафиксировать время выполнения процедур загрузки данных от источника до витрины для каждого типа таблиц.

План. Цель работы – сравнить скорость работы

Замер производительности обновления справочника в 1 млн записей:

1 million dimension update (in ms)				
distibution	stg	scd1 (full)		scd2 (merge)
	Insert from Oracle	Truncate	Delete	delete-Insert
by id	7998,67	1344,33	1479,67	2449,00
randomly	8256,33	1427,00	1670,67	2955,33
replicated	15172,67	8307,33	8992,33	7529,00

Лабораторная работа. Пробная загрузка

Создайте внешнюю таблицу в Greenplum:

```
create external table etl.stg_foo_ext
(proid integer
 ,operday date ,dbacc integer
 ,dbcur varchar(3) ,cracc integer
 ,crcur varchar(3) ,dbsum decimal(31,10)
 ,crsum decimal(31,10) ,purpose varchar(500))
LOCATION
('pxf://C##ETL_USER.PROVODKI?PROFILE=JDBC&JDBC_DRIVER=oracle.jdbc.driver.OracleDriver&D
B_URL=jdbc:oracle:thin:@//oracle-
server:1521/XE&USER=c##etl_user&PASS=c##etl_user&&PARTITION_BY=operday:date&RANGE=2020-
03-08:2020-04-20&INTERVAL=1:day')
FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import');
```

Лабораторная работа. Пробная загрузка

Создайте stage-таблицу в Greenplum:

```
create table etl.stg_foo_rnd
(proid integer
 ,operday date
 ,dbacc integer
 ,dbcur varchar(3)
 ,cracc integer
 ,crcur varchar(3)
 ,dbsum decimal(31,10)
 ,crsum decimal(31,10)
 ,purpose varchar(500)
)
with (appendonly=true , orientation=column, compresstype=zstd, compresslevel=1)
distributed randomly;
```

Лабораторная работа. Пробная загрузка

Загрузка stage-слоя из СУБД Oracle.

Указываем ограничение по датам, загружаем данные в стейдж.

```
insert into stg_foo_rnd
select * from stg_foo_ext
where operday >= '05.01.2020'::date;
```

Проверяем запросы на стороне Oracle:

```
select x.sid
      ,x.serial#
      ,x.username
      ,sql_text
      ,x.sql_id
      ,x.sql_child_number
      ,optimizer_mode
      ,hash_value
      ,address
from   v$sqlarea sqlarea,v$session x
where x.sql_hash_value = sqlarea.hash_value
and x.sql_address = sqlarea.address
and x.username = ‘C##ETL_USER';
```

Лабораторная работа. Пробная загрузка

1. Создайте внешнюю таблицу к таблице проводок в oracle без инструкции PARTITION_BY.
2. Создайте локальную таблицу такой же структуры.
3. Загрузите данные за **май** из внешней таблицы в локальную таблицу.
4. Замерьте время загрузки данных в локальную таблицу. Замерьте время выполнения и проверьте выполняющиеся запросы в Oracle.
5. Пересоздайте внешнюю таблицу к таблице проводок Oracle с инструкцией PARTITION_BY по полю operday и очистьте (truncate) внутреннюю таблицу.
6. Повторите загрузку в локальную таблицу. Замерьте время выполнения и проверьте выполняющиеся запросы в Oracle.

План. Функции для обновления DWH

В тестовой БД реализованы следующие процедуры загрузки:

- sp_stg_client_rnd_load – загрузка stage-таблицы клиентов.
- sp_dim_client_rnd_full_load – с полным обновлением через delete-insert.
- sp_dim_client_rep_full_load – с полным обновлением через delete-insert.
- sp_dim_client_rnd_merge_1 – слияние через delete-insert.
- sp_dim_client_rnd_merge_2 – слияние через ранжирование.
- sp_stg_account_rnd_load – загрузка stage-таблицы счетов.
- sp_dim_account_hist_rnd_init_load – инициализирующая загрузка.
- sp_dim_account_hist_rnd_merge – обновление исторического справочника.
- sp_provodki_init_load – инициализирующая загрузка.
- sp_provodki_operday_load – обновление витрины проводок.

Лабораторная работа. DWH - загрузка данных

1. Создайте аналоги stage-таблиц stg_clients_rnd, stg_accounts_rnd, stg_provodki_rnd, но распределенные по полям:
   - Клиенты по полю clientcode
   - Счета по полю acccode
   - Проводки по полю dbacc
2. Постфикс у таблиц укажите _id вместо _rnd
3. Создайте аналоги процедур загрузки stage-таблиц sp_stg_client_rnd_load, sp_stg_account_rnd_load, sp_stg_provodki_rnd_load, но для новых таблиц.
4. Замерьте время выполнения каждой из версий (данные проводок с начала мая):
   - sp_stg_client_rnd_load, sp_stg_client_id_load
   - sp_stg_account_rnd_load, sp_stg_account_id_load
   - sp_stg_provodki_rnd_load, sp_stg_provodki_id_load

Лабораторная работа. DWH – справочники (1)

1. Создайте аналоги таблиц-справочников dim_clients_rnd, dim_accounts_hist_rnd,но распределенные по полям:

- Клиенты по полю clientcode
- Счета по полю acccode

2. Постфикс у таблиц укажите _id вместо _rnd
3. Выполните инициализирующаю загрузку счетов:

```
insert into etl.dim_accounts_hist_id
select *
       ,'05.01.2020'::date date_from
       ,'12.31.2100'::date date_to
from etl.stg_accounts_id;
```

Лабораторная работа. DWH – справочники (2)

4. Создайте аналоги процедур загрузки stage-таблиц sp_dim_client_rnd_full_load, sp_dim_account_hist_rnd_merge, но для новых таблиц:
5. В загрузке счетов используется вспомогательная таблица stg_accounts_new_id, в ней содержаться измененные данные по счетам.
6. Замерьте время выполнения каждой из версий:

- sp_dim_client_rnd_full_load, sp_dim_client_id_full_load, sp_dim_client_rep_full_load
- sp_dim_account_hist_rnd_merge, sp_dim_account_hist_id_merge

Лабораторная работа. DWH – факты

1. Создайте аналоги stage-таблиц stg_provodki_rnd и dm_provodki_rnd, но распределенные по полю dbacc.
2. Постфикс у таблиц укажите _id вместо _rnd.
3. Создайте аналог процедуры загрузки sp_provodki_operday_load используя созданные таблицы dim_clients_id, dim_accounts_hist_id, stg_provodki_id, dm_provodki_id.
4. Создайте аналог процедуры загрузки sp_provodki_operday_load используя созданные таблицы dim_accounts_hist_id, stg_provodki_id, dm_provodki_id и таблицу dim_clients_**rep**.
5. Замерьте время выполнения каждой из версий: sp_provodki_operday_load, sp_provodki_operday_load_id, sp_provodki_operday_load_rep

Лабораторная работа. DWH – результаты замеров

У вас должны получиться результаты замеров времени, похожие на данные:

	client		account		provodki	dm_pro
distibution	stg	dim (full)	stg	hist (merge)	stg	
randomly	8929	2962	23141	10153	10712	23851
id	7781	2354	23404	7657	10651	13749
id_wp	-	-	-	60538	-	-
replicated	-	6978	-	-	-	13935

Конец курса