Scaling To Infinity: Making Star Transformations Sing

nocoug.org

Thursday 15-November 2012
Tim Gorman
www.EvDBT.com

Speaker Qualifications

- Co-author…
  1. "*Oracle8 Data Warehousing*", 1998 John Wiley & Sons
  2. "*Essential Oracle8i Data Warehousing*", 2000 John Wiley & Sons
  3. "*Oracle Insights: Tales of the Oak Table*", 2004 Apress
  4. *"Basic Oracle SQL"* 2009 Apress
  5. "*Expert Oracle Practices: Database Administration with the Oak Table*", 2010 Apress
- 28 years in IT…
  - "C" programmer/developer, system admin, network admin (1984-1990)
  - Consultant and technical consulting manager at Oracle (1990-1998)
  - Independent consultant (http://www.EvDBT.com) since 1998
  - B of D, Rocky Mountain Oracle Users Group (http://www.RMOUG.org) since 1995
  - B of D, Oracle Developer Tools Users Group (http://www.ODTUG.com) since 2012
  - Oak Table network (http://www.OakTable.net) since 2002
  - Oracle ACE since 2007, Oracle ACE Director since 2012

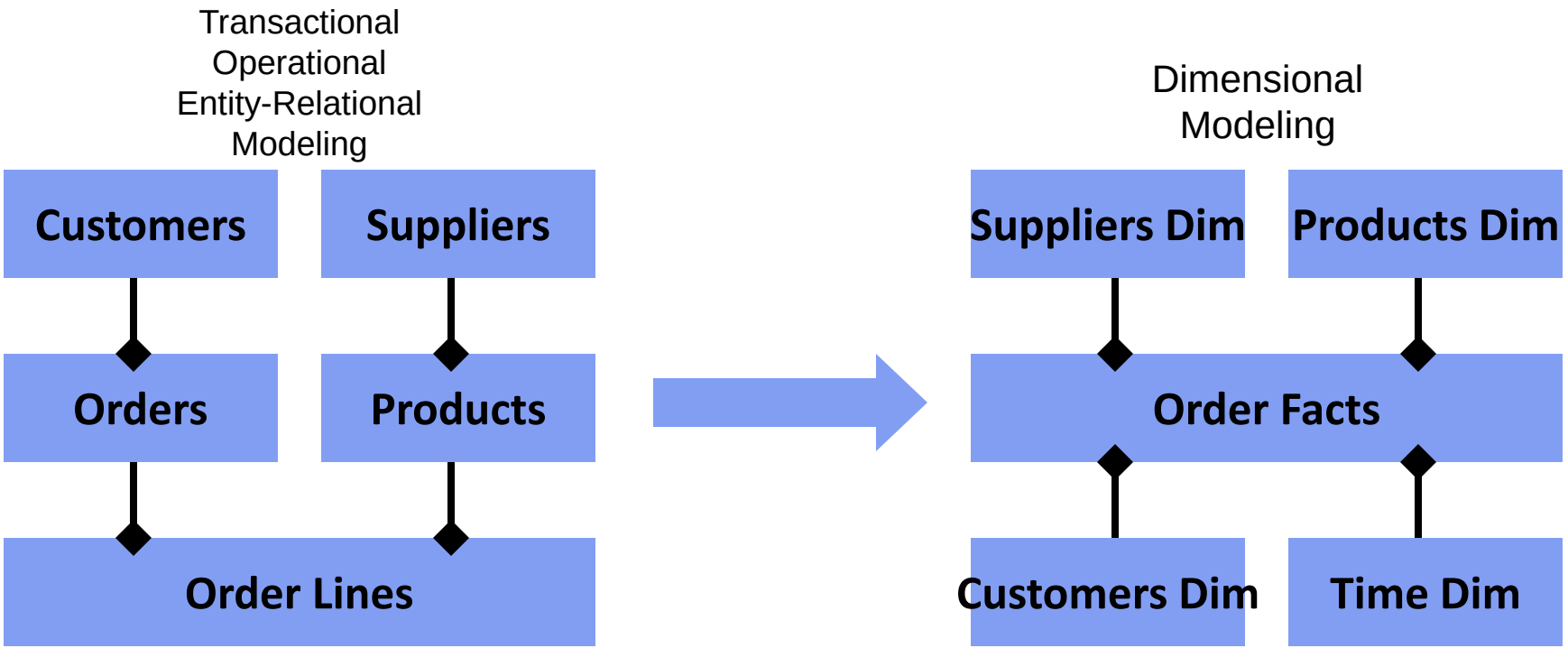
Agenda

1. Dimensional data models and DW/BI apps
2. Join methods in Oracle
   - Cartesian, NL, SM, HA, and STAR
3. Enabling Star Transformation
   - Parameters, permissions, and bitmap indexes
4. Enhancing with bitmap-join indexes
   - Parameters and constraints
   - Restrictions and other hoops
   - Data loading

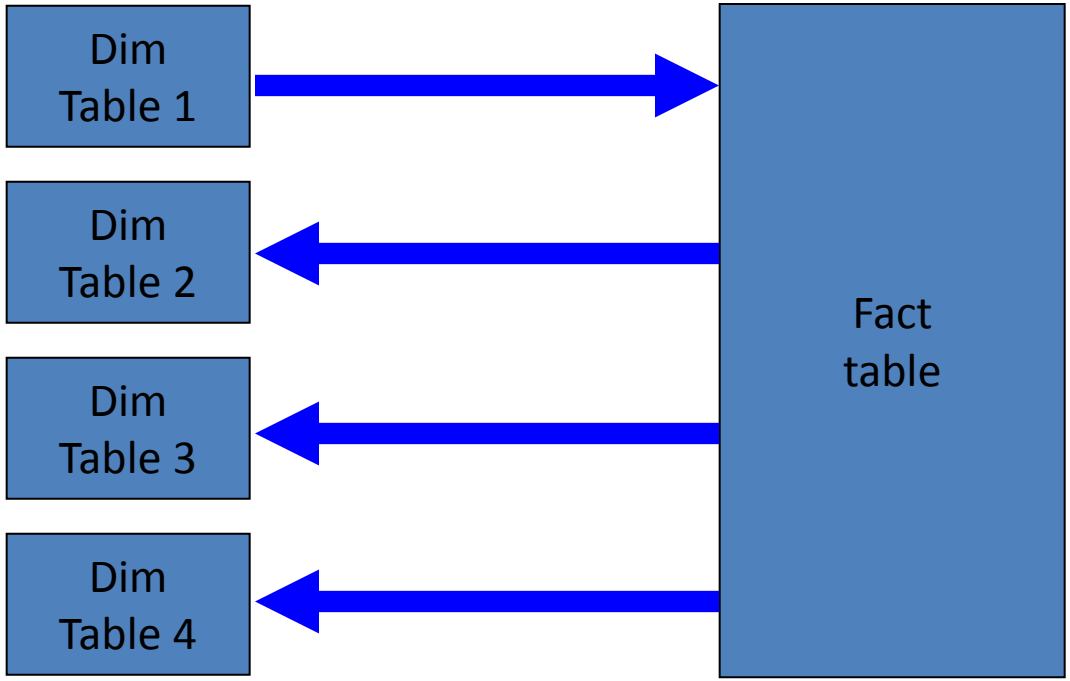
Why Dimensional Data Models?

- BI analysts (and tools) want a big spreadsheet
  - Lots and lots of attribute and measure columns
    - Attributes characterize the data
    - Measures are usually additive and numeric
    - Keys are invisible
- Dimensional data model is really just that spreadsheet
  - Normalized to recursive depth of one
    - More sophisticated models might normalize further (snowflake)
    - Normalized entities are dimension tables
      - Columns are primary key and attribute columns
    - Original spreadsheet is the fact table
      - Columns are foreign-keys to dimensions and measures

Why Dimensional Data Models?



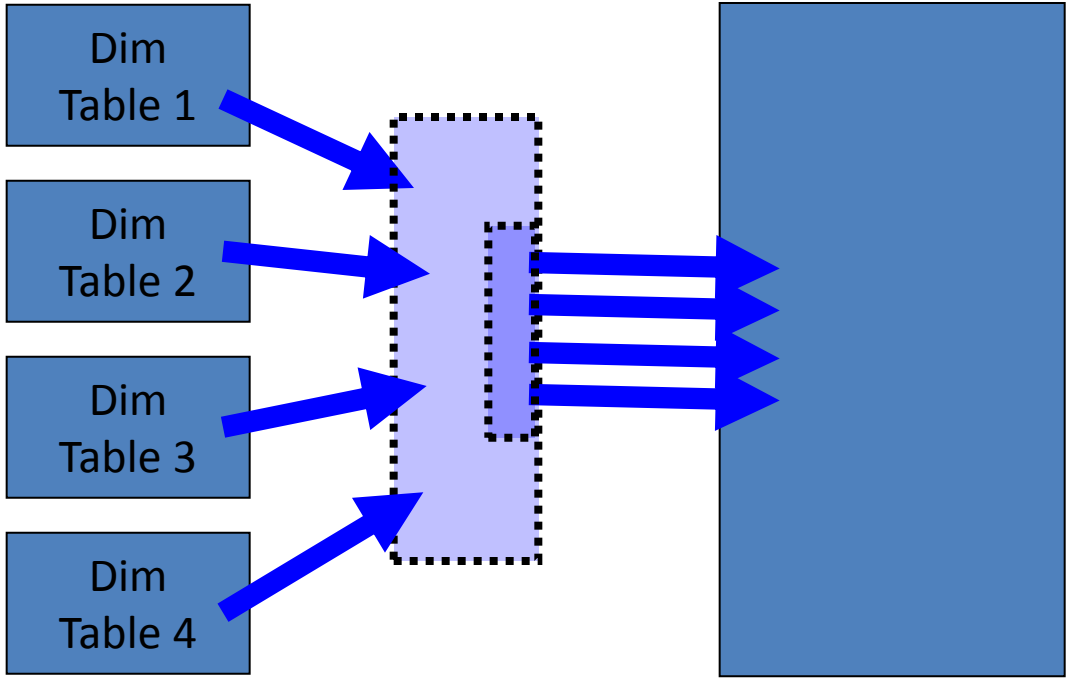
Why Star Transformations?



Standard join methods (NL, SM, HA):

- Filter result set in one of the dimension tables
- Join from that dimension table to the fact along a low-cardinality dimension key
- Join back from fact to other dimensions using dimension PK
  - Filtering rows retrieved along the way
- Wasteful and inefficient

Why Star Transformations?



Star transformation:

- Filter on query set in each dimension
- Merge result set from all dimensions
- Join to the fact from merged result set, using BITMAP MERGE index scan

Enabling Star Transformations

- Parameters
  - star_transformation_enabled = true | temp_disable
- CBO statistics
  - Must be gathered first
- Requires bitmap indexes
  - On all dimension key columns on fact table

The Virtuous Cycle

- Data Warehouses are *non-volatile, time-variant*, integrated, and subject-oriented
  - Bill Inmon's *Four Characteristics of a Data Warehouse*
- Non-volatile time-variant data *implies*…
  - Data warehouses should primarily be INSERT (and SELECT)
- INSERT-only data warehouses *implies*…
  - Tables and indexes range-partitioned by a DATE column
- Tables range-partitioned by DATE *enables*…
  - Data loading using EXCHANGE PARTITION load technique
- Data loading using EXCHANGE PARTITION *enables*…
  - Bitmap indexes and bitmap-join indexes
- Bitmap indexes *enable*…
  - Star transformations
- Star transformations *enable*…
  - Optimal query-execution plan for dimensional data models
  - Bitmap-join indexes
- Bitmap-join indexes *enable*…
  - Further optimization of star transformations

The Death Spiral

- Volatile point-in-time data *implies*…
  - Mass MERGE/UPDATE conventional-path operations contending with each other
- Bitmap indexes must be dropped/recreated rather than updated
  - For how long does this remain feasible?
- No bitmap indexes, so no star transformations
  - System thrashes itself to death on hash joins
- No star transformations…
  - Analytic queries are off-loaded to new downstream systems
    - Queries shift from aggregated and analytical to simple “dumps”
- ETL dominates workload
  - Because that is all that the database can do…
  - Designation subtly shifts from “data warehouse” or “data mart” to “staging system”…
- Bring on Exadata, Teradata, Netezza, GreenPlum, etc.
  - Oracle is junk!

The importance of EXCHANGE PARTITION

Point: Single-column bitmap indexes on dimension-key columns on the fact table are required for star transformation

Counter-point: Bitmap indexes become difficult (*impossible*?) to maintain when data volume increases into (and beyond) Tbytes

Catch-22? Does this mean that the Oracle RDBMS cannot handle large data warehouses?

Using Star Transformations


```
SQL> select  count(distinct F.SALES_ORDER_NUM) as cnt_sales_order,
```

(...several lines removed for brevity...)

```
 22          D_ORG.ORG_HIER9_NAME as ORG_NAME,
 23          D_ORG.ORG_HIER9_NUM as ORG_NUM
 24  from    W_INT_ORG_DH                        D_ORG,
 25          W_DAY_D                             D_DT,
 26          W_SALES_ORDER_LINE_F                F,
 27          W_STATUS_D                          D_STATUS,
 28          W_XACT_TYPE_D                       D_XACT,
 29          W_USER_D                            D_USER
 30  where   D_STATUS.W_STATUS_CODE = 'Closed'
 31  and     D_STATUS.W_STATUS_CLASS in ('SALES_ORDER_PROCESS')
 32  and     D_DT.PER_NAME_MONTH = '2011 / 08'
 33  and     D_DT.PER_NAME_WEEK in ('2011 Week32', '2011 Week33')
 34  and     D_ORG.HIERARCHY_NAME in ('PB FIN REPORTING')
 35  and     D_ORG.FIXED_HIER_LEVEL in (10)
 36  and     D_USER.FULL_NAME = 'Mcintyre, Mr. Daniel'
 37  and     F.ORDER_STATUS_WID = D_STATUS.ROW_WID
 38  and     F.XACT_TYPE_WID = D_XACT.ROW_WID
 39  and     F.INVENTORY_ORG_WID = D_ORG.ORG_WID
 40  and     F.X_INVOICE_DT_WID = D_DT.ROW_WID
 41  and     F.CREATED_BY_WID = D_USER.ROW_WID
 42  group by D_ORG.ORG_HIER9_NAME,
 43           D_ORG.ORG_HIER9_NUM;
```

Using Star Transformations


```
SQL> select  count(distinct F.SALES_ORDER_NUM) as cnt_sales_order,
```

(...several lines removed for brevity...)

```
 22          D_ORG.ORG_HIER9_NAME as ORG_NAME,
 23          D_ORG.ORG_HIER9_NUM as ORG_NUM
 24  from    W_INT_ORG_DH                        D_ORG,
 25          W_DAY_D                             D_DT,
 26          W_SALES_ORDER_LINE_F                F,
 27          W_STATUS_D                          D_STATUS,
 28          W_XACT_TYPE_D                       D_XACT,
 29          W_USER_D                            D_USER
 30  where   D_STATUS.W_STATUS_CODE = 'Closed'
 31  and     D_STATUS.W_STATUS_CLASS in ('SALES_ORDER_PROCESS')
 32  and     D_DT.PER_NAME_MONTH = '2011 / 08'
 33  and     D_DT.PER_NAME_WEEK in ('2011 Week32', '2011 Week33')
 34  and     D_ORG.HIERARCHY_NAME in ('PB FIN REPORTING')
 35  and     D_ORG.FIXED_HIER_LEVEL in (10)
 36  and     D_USER.FULL_NAME = 'Mcintyre, Mr. Daniel'
 37  and     F.ORDER_STATUS_WID = D_STATUS.ROW_WID
 38  and     F.XACT_TYPE_WID = D_XACT.ROW_WID
 39  and     F.INVENTORY_ORG_WID = D_ORG.ORG_WID
 40  and     F.X_INVOICE_DT_WID = D_DT.ROW_WID
 41  and     F.CREATED_BY_WID = D_USER.ROW_WID
 42  group by D_ORG.ORG_HIER9_NAME,
 43           D_ORG.ORG_HIER9_NUM;
```

star_transformation_enabled = false


```
|   0 | SELECT STATEMENT                        |                       |     1 |   169 |  6455   (1)| 00:01:57 |
|   1 |  SORT GROUP BY                          |                       |     1 |   169 |  6455   (1)| 00:01:57 |
|*  2 |   TABLE ACCESS BY LOCAL INDEX ROWID     | W_SALES_ORDER_LINE_F  |     1 |    46 |  6454   (1)| 00:01:57 |
|   3 |    NESTED LOOPS                         |                       |     1 |   169 |  6454   (1)| 00:01:57 |
|   4 |     MERGE JOIN CARTESIAN                |                       |    54 |  6642 |    65   (0)| 00:00:02 |
|   5 |      MERGE JOIN CARTESIAN               |                       |     1 |   105 |    62   (0)| 00:00:02 |
|   6 |       MERGE JOIN CARTESIAN              |                       |     1 |    70 |    60   (0)| 00:00:02 |
|   7 |        MERGE JOIN CARTESIAN             |                       |     1 |    36 |     8   (0)| 00:00:01 |
|   8 |         TABLE ACCESS BY INDEX ROWID     | W_DAY_D               |     1 |    25 |     3   (0)| 00:00:01 |
|   9 |          BITMAP CONVERSION TO ROWIDS    |                       |       |       |            |          |
|  10 |           BITMAP AND                    |                       |       |       |            |          |
|* 11 |            BITMAP INDEX SINGLE VALUE    | W_DAY_D_M17           |       |       |            |          |
|  12 |            BITMAP OR                    |                       |       |       |            |          |
|* 13 |             BITMAP INDEX SINGLE VALUE   | W_DAY_D_M16           |       |       |            |          |
|* 14 |             BITMAP INDEX SINGLE VALUE   | W_DAY_D_M16           |       |       |            |          |
|  15 |         BUFFER SORT                     |                       |     1 |    11 |     5   (0)| 00:00:01 |
|  16 |          TABLE ACCESS BY INDEX ROWID    | W_USER_D              |     1 |    11 |     8   (0)| 00:00:01 |
|  17 |           BITMAP CONVERSION TO ROWIDS   |                       |       |       |            |          |
|* 18 |            BITMAP INDEX SINGLE VALUE    | W_USER_D_M5           |       |       |            |          |
|  19 |        BUFFER SORT                      |                       |     1 |    34 |    52   (0)| 00:00:01 |
|  20 |         TABLE ACCESS BY INDEX ROWID     | W_INT_ORG_DH          |     1 |    34 |    60   (0)| 00:00:02 |
|  21 |          BITMAP CONVERSION TO ROWIDS    |                       |       |       |            |          |
|  22 |           BITMAP AND                    |                       |       |       |            |          |
|* 23 |            BITMAP INDEX SINGLE VALUE    | W_INT_ORG_DH_M23      |       |       |            |          |
|* 24 |            BITMAP INDEX SINGLE VALUE    | W_INT_ORG_DH_M15      |       |       |            |          |
|  25 |       BUFFER SORT                       |                       |     1 |    35 |     2   (0)| 00:00:01 |
|* 26 |        TABLE ACCESS BY INDEX ROWID      | W_STATUS_D            |     1 |    35 |     2   (0)| 00:00:01 |
|* 27 |         INDEX RANGE SCAN                | W_STATUS_D_U2         |     2 |       |     1   (0)| 00:00:01 |
|  28 |      BUFFER SORT                        |                       |  2021 | 36378 |    63   (0)| 00:00:02 |
|  29 |       INDEX FAST FULL SCAN              | W_XACT_TYPE_D_U2      |  2021 | 36378 |     3   (0)| 00:00:01 |
|  30 |     PARTITION RANGE ALL                 |                       |       |       |            |          |
|  31 |      BITMAP CONVERSION TO ROWIDS        |                       |       |       |            |          |
|  32 |       BITMAP AND                        |                       |       |       |            |          |
|* 33 |        BITMAP INDEX SINGLE VALUE        | W_SLS_ORD_LN_F_F26    |       |       |            |          |
|* 34 |        BITMAP INDEX SINGLE VALUE        | W_SLS_ORD_LN_F_F32    |       |       |            |          |
|* 35 |        BITMAP INDEX SINGLE VALUE        | W_SLS_ORD_LN_F_F56    |       |       |            |          |
```

- Elapsed time = 01:00:55.19

```
Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
  895189202  consistent gets
      52033  physical reads
    5472500  redo size
        982  bytes sent via SQL*Net to client
        492  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          5  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

star_transformation_enabled = true

0	SELECT STATEMENT		1	164	958	(1)
1	TEMP TABLE TRANSFORMATION					
2	LOAD AS SELECT	SYS_TEMP_0FD9D6674_54381A51				
* 3	TABLE ACCESS BY INDEX ROWID	W_INT_ORG_DH	1	34	32	(0)
* 4	INDEX SKIP SCAN	W_INT_ORG_DH_F3	375		25	(0)
5	SORT GROUP BY		1	164	926	(1)
11	TABLE ACCESS BY INDEX ROWID	W_DAY_D	1	25	3	(0)
12	BITMAP CONVERSION TO ROWIDS					
13	BITMAP AND					
* 14	BITMAP INDEX SINGLE VALUE	W_DAY_D_M17				
15	BITMAP OR					
* 16	BITMAP INDEX SINGLE VALUE	W_DAY_D_M16				
* 17	BITMAP INDEX SINGLE VALUE	W_DAY_D_M16				
18	PARTITION RANGE ALL		8336	374K	910	(0)
19	TABLE ACCESS BY LOCAL INDEX ROWID	W_SALES_ORDER_LINE_F	8336	374K	910	(0)
20	BITMAP CONVERSION TO ROWIDS					
21	BITMAP AND					
22	BITMAP MERGE					
23	BITMAP KEY ITERATION					
24	BUFFER SORT					
25	TABLE ACCESS FULL	SYS_TEMP_0FD9D6674_54381A51	1	13	2	(0)
* 26	BITMAP INDEX RANGE SCAN	W_SLS_ORD_LN_F_F26				
27	BITMAP MERGE					
28	BITMAP KEY ITERATION					
29	BUFFER SORT					
* 30	TABLE ACCESS BY INDEX ROWID	W_STATUS_D	1	35	2	(0)
31	BITMAP CONVERSION TO ROWIDS					
* 32	BITMAP INDEX SINGLE VALUE	W_STATUS_D_M3				
* 33	BITMAP INDEX RANGE SCAN	W_SLS_ORD_LN_F_F32				
* 34	TABLE ACCESS BY INDEX ROWID	W_STATUS_D	1	35	2	(0)
35	BITMAP CONVERSION TO ROWIDS					
* 36	BITMAP INDEX SINGLE VALUE	W_STATUS_D_M3				
37	TABLE ACCESS FULL	SYS_TEMP_0FD9D6674_54381A51	1	29	2	(0)
38	TABLE ACCESS BY INDEX ROWID	W_USER_D	1	11	2	(0)
39	BITMAP CONVERSION TO ROWIDS					
* 40	BITMAP INDEX SINGLE VALUE	W_USER_D_M5				
41	INDEX FAST FULL SCAN	W_XACT_TYPE_D_U2	2021	36378	3	(0)

- Elapsed time = 00:00:14.09

```
Note
-----
   - star transformation used for this statement

Statistics
----------------------------------------------------------
        120  recursive calls
         10  db block gets
     271871  consistent gets
         61  physical reads
       2716  redo size
        982  bytes sent via SQL*Net to client
        492  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

star_transformation_enabled = temp_disable


```
|   0 | SELECT STATEMENT                          |                       |      1 |    292 |    956   (1)|00:00:18
|   1 |  SORT GROUP BY                            |                       |      1 |    292 |    956   (1)|00:00:18
|*  2 |   HASH JOIN                               |                       |      1 |    292 |    955   (1)|00:00:18

|   7 |         TABLE ACCESS BY INDEX ROWID       | W_DAY_D               |      1 |     25 |      3   (0)|00:00:01
|   8 |          BITMAP CONVERSION TO ROWIDS      |                       |        |        |              |
|   9 |           BITMAP AND                      |                       |        |        |              |
|* 10 |            BITMAP INDEX SINGLE VALUE      | W_DAY_D_M17           |        |        |              |
|  11 |            BITMAP OR                      |                       |        |        |              |
|* 12 |             BITMAP INDEX SINGLE VALUE     | W_DAY_D_M16           |        |        |              |
|* 13 |             BITMAP INDEX SINGLE VALUE     | W_DAY_D_M16           |        |        |              |
|  14 |         PARTITION RANGE ALL               |                       |   8336 |  1375K|    910   (0)|00:00:17
|  15 |          TABLE ACCESS BY LOCAL INDEX ROWID| W_SALES_ORDER_LINE_F  |   8336 |  1375K|    910   (0)|00:00:17
|  16 |           BITMAP CONVERSION TO ROWIDS     |                       |        |        |              |
|  17 |            BITMAP AND                     |                       |        |        |              |
|  18 |             BITMAP MERGE                  |                       |        |        |              |
|  19 |              BITMAP KEY ITERATION         |                       |        |        |              |
|  20 |               BUFFER SORT                 |                       |        |        |              |
|* 21 |                TABLE ACCESS BY INDEX ROWID| W_INT_ORG_DH          |      1 |     34 |     32   (0)|00:00:01
|* 22 |                 INDEX SKIP SCAN           | W_INT_ORG_DH_F3       |    375 |        |     25   (0)|00:00:01
|* 23 |               BITMAP INDEX RANGE SCAN     | W_SLS_ORD_LN_F_F26    |        |        |              |
|  24 |             BITMAP MERGE                  |                       |        |        |              |
|  25 |              BITMAP KEY ITERATION         |                       |        |        |              |
|  26 |               BUFFER SORT                 |                       |        |        |              |
|* 27 |                TABLE ACCESS BY INDEX ROWID| W_STATUS_D            |      1 |     35 |      2   (0)|00:00:01
|  28 |                 BITMAP CONVERSION TO ROWIDS|                      |        |        |              |
|* 29 |                  BITMAP INDEX SINGLE VALUE| W_STATUS_D_M3         |        |        |              |
|* 30 |               BITMAP INDEX RANGE SCAN     | W_SLS_ORD_LN_F_F32    |        |        |              |
|* 31 |        TABLE ACCESS BY INDEX ROWID        | W_STATUS_D            |      1 |     35 |      2   (0)|00:00:01
|  32 |         BITMAP CONVERSION TO ROWIDS       |                       |        |        |              |
|* 33 |          BITMAP INDEX SINGLE VALUE        | W_STATUS_D_M3         |        |        |              |
|  34 |       TABLE ACCESS BY INDEX ROWID         | W_USER_D              |      1 |     11 |      2   (0)|00:00:01
|  35 |        BITMAP CONVERSION TO ROWIDS        |                       |        |        |              |
|* 36 |         BITMAP INDEX SINGLE VALUE         | W_USER_D_M5           |        |        |              |
|* 37 |      TABLE ACCESS BY INDEX ROWID          | W_INT_ORG_DH          |      1 |     34 |     32   (0)|00:00:01
|* 38 |       INDEX SKIP SCAN                     | W_INT_ORG_DH_F3       |    375 |        |     25   (0)|00:00:01
|  39 |    INDEX FAST FULL SCAN                   | W_XACT_TYPE_D_U2      |   2021 |  36378 |      3   (0)|00:00:01
```

- Elapsed = 00:00:13.21

```
Note
-----
   - star transformation used for this statement

Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
     271859  consistent gets
          0  physical reads
        808  redo size
        982  bytes sent via SQL*Net to client
        492  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Why Bitmap-Join Indexes?

- Sometimes the performance of star transformations is not fast enough, or consumes too many resources…
  - Problems with…
    - global temporary tables
    - PGA memory consumption
    - temp tablespace utilization for sorts/hashes
- Sometimes queries should adopt a star transformation…
  - But they don't…

Why Bitmap-Join Indexes?

```
SQL> create bitmap index W_SLS_ORD_LN_F_BJI01
  2  on W_SALES_ORDER_LINE_F(S.W_STATUS_CODE,
  3                          S_D.W_STATUS_CLASS,
  4                          D.PER_NAME_MONTH,
  5                          D.PER_NAME_WEEK,
  6                          U.FULL_NAME)
  7  from  W_STATUS_D              S,
  8        W_DAY_D                 D,
  9        W_USER_D                U,
 10        W_SALES_ORDER_LINE_F    F
 11  where F.ORDER_STATUS_WID    = S.ROW_WID
 12  and   F.X_INVOICE_DT_WID    = D.ROW_WID
 13  and   F.CREATED_BY_WID      = U.ROW_WID
 14  local
 15  tablespace dw_idx
 16  compute statistics;

Index created.
```

Enabling Bitmap-Join Indexes

- Parameters
  - query_rewrite_enabled = true
- Permissions
  - query rewrite *or* global query rewrite
- Constraints
  - Must have PK constraint enabled on all dimension tables
- CBO statistics
  - Must be gathered first
- Index resides on fact table
  - Column-list refers to dimensional attributes
  - Where-clause contains only join-predicates

```
SQL> select  sum(case when F_SOL.X_EXTENDED_AMOUNT is null
```

(…several dozen lines of code redacted for brevity…)

```
 36  from    W_DAY_D                    D_DT1,
 37          W_DAY_D                    D_DT2,
 38          W_USER_D                   D_USR,
 39          W_EMPLOYEE_D               D_EMP,
 40          WC_ORDER_TYPE_D            D_ORDTYP,
 41          W_PAYMENT_TERMS_D          D_PMTTRM,
 42          W_CUSTOMER_LOC_D           D_CUSTLOC,
 43          W_BUSN_LOCATION_D          D_BUSLOC,
 44          W_XACT_TYPE_D              D_XCTTYP,
 45          W_PARTY_D                  D_PARTY,
 46          W_SALES_ORDER_LINE_F       F_SOL,
 47          WC_SALESCREDIT_REVN_H      F_SCRH
 48  where   D_DT1.DAY_DT = TO_DATE('2011-08-07 00:00:00' , 'YYYY-MM-DD HH24:MI:SS')
 49  and     D_DT2.DAY_DT = TO_DATE('2011-08-07 00:00:00' , 'YYYY-MM-DD HH24:MI:SS')
 50  and     D_USR.FULL_NAME = 'Ables, Mr. Quinton R'
 51  and     D_EMP.FULL_NAME = 'Abbott, Mr. Charles R'
 52  and     D_ORDTYP.ORDER_TYPE_NAME = 'COUNTER SALE'
 53  and     D_PMTTRM.PAYMENT_TERM_CODE = '.5PC NT 15TH'
 54  and     D_BUSLOC.BUSN_LOC_TYPE in ('PLANT')
 55  and     F_SOL.X_FULFILLMENT_DT_WID = D_DT1.ROW_WID
 56  and     F_SOL.ORDERED_ON_DT_WID = D_DT2.ROW_WID
 57  and     F_SOL.CREATED_BY_WID = D_USR.ROW_WID
 58  and     F_SOL.X_ORDER_TYPE_WID = D_ORDTYP.ROW_WID
 59  and     F_SOL.PAYMENT_TERMS_WID = D_PMTTRM.ROW_WID
 60  and     F_SOL.PLANT_LOC_WID = D_BUSLOC.ROW_WID
 61  and     D_CUSTLOC.ROW_WID = F_SOL.CUSTOMER_BILL_TO_LOC_WID
 62  and     D_XCTTYP.ROW_WID = F_SOL.XACT_TYPE_WID;
```

Using Bitmap-Join Indexes


```
SQL> select  sum(case when F_SOL.X_EXTENDED_AMOUNT is null
```

(…several dozen lines of code redacted for brevity…)

```
 36  from    W_DAY_D                    D_DT1,
 37          W_DAY_D                    D_DT2,
 38          W_USER_D                   D_USR,
 39          W_EMPLOYEE_D               D_EMP,
 40          WC_ORDER_TYPE_D            D_ORDTYP,
 41          W_PAYMENT_TERMS_D          D_PMTTRM,
 42          W_CUSTOMER_LOC_D           D_CUSTLOC,
 43          W_BUSN_LOCATION_D          D_BUSLOC,
 44          W_XACT_TYPE_D              D_XCTTYP,
 45          W_PARTY_D                  D_PARTY,
 46          W_SALES_ORDER_LINE_F       F_SOL,
 47          WC_SALESCREDIT_REVN_H      F_SCRH
 48  where   D_DT1.DAY_DT = TO_DATE('2011-08-07 00:00:00' , 'YYYY-MM-DD HH24:MI:SS')
 49  and     D_DT2.DAY_DT = TO_DATE('2011-08-07 00:00:00' , 'YYYY-MM-DD HH24:MI:SS')
 50  and     D_USR.FULL_NAME = 'Ables, Mr. Quinton R'
 51  and     D_EMP.FULL_NAME = 'Abbott, Mr. Charles R'
 52  and     D_ORDTYP.ORDER_TYPE_NAME = 'COUNTER SALE'
 53  and     D_PMTTRM.PAYMENT_TERM_CODE = '.5PC NT 15TH'
 54  and     D_BUSLOC.BUSN_LOC_TYPE in ('PLANT')
 55  and     F_SOL.X_FULFILLMENT_DT_WID = D_DT1.ROW_WID
 56  and     F_SOL.ORDERED_ON_DT_WID = D_DT2.ROW_WID
 57  and     F_SOL.CREATED_BY_WID = D_USR.ROW_WID
 58  and     F_SOL.X_ORDER_TYPE_WID = D_ORDTYP.ROW_WID
 59  and     F_SOL.PAYMENT_TERMS_WID = D_PMTTRM.ROW_WID
 60  and     F_SOL.PLANT_LOC_WID = D_BUSLOC.ROW_WID
 61  and     D_CUSTLOC.ROW_WID = F_SOL.CUSTOMER_BILL_TO_LOC_WID
 62  and     D_XCTTYP.ROW_WID = F_SOL.XACT_TYPE_WID;
```

```
|    0 | SELECT STATEMENT                          |                          |      1 |   318 | 39873   (2)|
|    1 |  PX COORDINATOR                           |                          |        |       |            |
|    2 |   PX SEND QC (ORDER)                      | :TQ10004                 |      1 |   318 | 39873   (2)|
|    3 |    SORT ORDER BY                          |                          |      1 |   318 | 39873   (2)|
|    5 |     PX SEND RANGE                         | :TQ10003                 |      1 |   318 | 39873   (2)|
|    6 |      HASH GROUP BY                        |                          |      1 |   318 | 39873   (2)|
|    7 |       PX RECEIVE                          |                          |      1 |   318 | 39870   (2)|
|    8 |        PX SEND HASH                       | :TQ10002                 |      1 |   318 | 39870   (2)|
|    9 |         NESTED LOOPS                      |                          |      1 |   318 | 39870   (2)|
|   10 |          NESTED LOOPS                     |                          |      1 |   306 | 39870   (2)|
|   11 |           NESTED LOOPS                    |                          |      1 |   294 | 39870   (2)|
|* 12 |            HASH JOIN                      |                          |      1 |   267 | 39870   (2)|
|   13 |             PX RECEIVE                    |                          |      1 |   250 | 35982   (2)|
|   14 |              PX SEND HASH                 | :TQ10001                 |      1 |   250 | 35982   (2)|
|   15 |               NESTED LOOPS                |                          |      1 |   250 | 35982   (2)|

|   21 |                 NESTED LOOPS              |                          |      1 |   118 | 35981   (2)|
|   22 |                  PX BLOCK ITERATOR        |                          |      1 |    93 | 35980   (2)|
|* 23 |                   TABLE ACCESS FULL       | W_SALES_ORDER_LINE_F     |      1 |    93 | 35980   (2)|
|* 24 |                  TABLE ACCESS BY INDEX ROWID| WC_ORDER_TYPE_D         |      1 |    25 |     1   (0)|
|* 25 |                   INDEX UNIQUE SCAN       | WC_ORDER_TYPE_D_PK       |      1 |       |     0   (0)|
|* 26 |                  TABLE ACCESS BY INDEX ROWID | W_PAYMENT_TERMS_D      |      1 |    18 |     1   (0)|
|* 27 |                   INDEX UNIQUE SCAN       | W_PAYMNT_TRM_D_P1        |      1 |       |     0   (0)|
|* 28 |                 TABLE ACCESS BY INDEX ROWID | W_USER_D               |      1 |    11 |     1   (0)|
|* 29 |                  INDEX UNIQUE SCAN        | W_USER_D_P1              |      1 |       |     0   (0)|
|* 30 |                TABLE ACCESS BY INDEX ROWID | W_BUSN_LOCATION_D      |      1 |    28 |     1   (0)|
|* 31 |                 INDEX UNIQUE SCAN         | W_BUSN_LOC_D_P1          |      1 |       |     0   (0)|
|   32 |               TABLE ACCESS BY INDEX ROWID | W_XACT_TYPE_D            |      1 |    18 |     1   (0)|
|* 33 |                INDEX UNIQUE SCAN          | W_XACT_TYPE_D_P1         |      1 |       |     0   (0)|
|   34 |              TABLE ACCESS BY INDEX ROWID  | W_PARTY_D                |      1 |    32 |     1   (0)|
|* 35 |               INDEX UNIQUE SCAN           | W_PARTY_D_P1             |      1 |       |     0   (0)|
|   36 |             TABLE ACCESS BY INDEX ROWID   | W_CUSTOMER_LOC_D         |      1 |    25 |     1   (0)|
|* 37 |              INDEX UNIQUE SCAN            | W_CUST_LOC_D_PI          |      1 |       |     0   (0)|
|   38 |            BUFFER SORT                    |                          |        |       |            |
|   39 |             PX RECEIVE                    |                          | 2874K  |   46M |  3884   (2)|
|   40 |              PX SEND HASH                 | :TQ10000                 | 2874K  |   46M |  3884   (2)|
|   41 |               TABLE ACCESS FULL           | WC_SALESCREDIT_REVN_H    | 2874K  |   46M |  3884   (2)|
|* 42 |            TABLE ACCESS BY INDEX ROWID    | W_EMPLOYEE_D             |      1 |    27 |     1   (0)|
|* 43 |             INDEX UNIQUE SCAN             | W_EMPLOYEE_D_P1          |      1 |       |     0   (0)|
|* 44 |           TABLE ACCESS BY INDEX ROWID     | W_DAY_D                  |      1 |    12 |     0   (0)|
|* 45 |            INDEX RANGE SCAN               | W_DAY_D_M39              |      1 |       |     0   (0)|
|* 46 |          TABLE ACCESS BY INDEX ROWID      | W_DAY_D                  |      1 |    12 |     0   (0)|
|* 47 |           INDEX RANGE SCAN                | W_DAY_D_M39              |      1 |       |     0   (0)|
```

- Elapsed = 00:10:12.81

```
Statistics
----------------------------------------------------------
       3388  recursive calls
          0  db block gets
    2287060  consistent gets
    2276872  physical reads
       2116  redo size
       1104  bytes sent via SQL*Net to client
       1560  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
         84  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

```
SQL> create bitmap index W_SLS_ORD_LN_F_BJI02
  2  on W_SALES_ORDER_LINE_F(D.DAY_DT,
  3                          U.FULL_NAME,
  4                          O.ORDER_TYPE_NAME,
  5                          P.PAYMENT_TERM_CODE,
  6                          L.BUSN_LOC_TYPE)
  7  from  W_DAY_D                 D,
  8        W_USER_D                U,
  9        WC_ORDER_TYPE_D         O,
 10        W_PAYMENT_TERMS_D       P,
 11        W_BUSN_LOCATION_D       L,
 12        W_SALES_ORDER_LINE_F    F
 13  where F.ORDERED_ON_DT_WID     = D.ROW_WID
 14  and   F.CREATED_BY_WID        = U.ROW_WID
 15  and   F.X_ORDER_TYPE_WID      = O.ROW_WID
 16  and   F.PAYMENT_TERMS_WID     = P.ROW_WID
 17  and   F.PLANT_LOC_WID         = L.ROW_WID
 18  local
 19  tablespace dw_idx
 20  compute statistics;

Index created.
```

...BJI to the rescue...

0	SELECT STATEMENT			1	318	165	(2)
1	SORT ORDER BY			1	318	165	(2)
2	HASH GROUP BY			1	318	165	(2)
3	NESTED LOOPS			1	318	163	(0)
13	NESTED LOOPS			1	110	153	(0)
14	PARTITION RANGE ALL			1	93	76	(0)
* 15	TABLE ACCESS BY LOCAL INDEX ROWID	W_SALES_ORDER_LINE_F		1	93	76	(0)
16	BITMAP CONVERSION TO ROWIDS						
* 17	BITMAP INDEX SINGLE VALUE	W_SLS_ORD_LN_F_BJI02					
18	TABLE ACCESS BY INDEX ROWID	WC_SALESCREDIT_REVN_H		1	17	153	(0)
19	BITMAP CONVERSION TO ROWIDS						
* 20	BITMAP INDEX SINGLE VALUE	X_W_SLSCR_REVN_H_F1					
* 21	TABLE ACCESS BY INDEX ROWID	W_EMPLOYEE_D		1	27	1	(0)
* 22	INDEX UNIQUE SCAN	W_EMPLOYEE_D_P1		1		0	(0)
23	TABLE ACCESS BY INDEX ROWID	W_PARTY_D		1	32	1	(0)
* 24	INDEX UNIQUE SCAN	W_PARTY_D_P1		1		0	(0)
25	TABLE ACCESS BY INDEX ROWID	W_XACT_TYPE_D		1	18	1	(0)
* 26	INDEX UNIQUE SCAN	W_XACT_TYPE_D_P1		1		0	(0)
* 27	TABLE ACCESS BY INDEX ROWID	W_BUSN_LOCATION_D		1	28	1	(0)
* 28	INDEX UNIQUE SCAN	W_BUSN_LOC_D_P1		1		0	(0)
29	TABLE ACCESS BY INDEX ROWID	W_CUSTOMER_LOC_D		1	25	1	(0)
* 30	INDEX UNIQUE SCAN	W_CUST_LOC_D_P1		1		0	(0)
* 31	TABLE ACCESS BY INDEX ROWID	W_PAYMENT_TERMS_D		1	18	1	(0)
* 32	INDEX UNIQUE SCAN	W_PAYMNT_TRM_D_P1		1		0	(0)
* 33	TABLE ACCESS BY INDEX ROWID	WC_ORDER_TYPE_D		1	25	1	(0)
* 34	INDEX UNIQUE SCAN	WC_ORDER_TYPE_D_PK		1		0	(0)
* 35	TABLE ACCESS BY INDEX ROWID	W_USER_D		1	11	1	(0)
* 36	INDEX UNIQUE SCAN	W_USER_D_P1		1		0	(0)
* 37	TABLE ACCESS BY INDEX ROWID	W_DAY_D		1	12	1	(0)
* 38	INDEX UNIQUE SCAN	W_DAY_D_P1		1		0	(0)
* 39	TABLE ACCESS BY INDEX ROWID	W_DAY_D		1	12	1	(0)
* 40	INDEX UNIQUE SCAN	W_DAY_D_P1		1		0	(0)

- Elapsed = 00:00:00.17

```
Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
        127  consistent gets
         15  physical reads
          0  redo size
       1104  bytes sent via SQL*Net to client
       1560  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

EXCHANGE PARTITION and BJI


```
create table w_solf_tmp tablespace dw_data as
    select * from w_sales_order_line_f partition (p201202)
    where 1 = 2;

insert /*+ append parallel */ into w_solf_tmp select ... from ...

exec dbms_stats.gather_table_stats('DWUSR', 'W_SOLF_TMP',
 estimate_percent=>null, method_opt=>'FOR ALL COLUMNS SIZE AUTO')

create bitmap index W_SLS_ORD_LN_F_F1_tmp on
  w_solf_tmp(ACCT_REP_WID) tablespace dw_idx compute statistics;

create bitmap index W_SLS_ORD_LN_F_F2_tmp on
  w_solf_tmp(BOOKED_DT_WID) tablespace dw_idx compute statistics;
```

...creation of 40+ more indexes edited out for brevity...

```
create bitmap index W_SLS_ORD_LN_F_BJI01_tmp
     on W_SOLF_TMP(S.W_STATUS_CODE,
                   S.W_STATUS_CLASS,
                   D.PER_NAME_MONTH,
                   D.PER_NAME_WEEK,
                   U.FULL_NAME)
     from    W_STATUS_D          S,
             W_DAY_D             D,
             W_USER_D            U,
             W_SOLF_TMP          F
     where   F.ORDER_STATUS_WID = S.ROW_WID
     and     F.X_INVOICE_DT_WID = D.ROW_WID
     and     F.CREATED_BY_WID = U.ROW_WID
     tablespace dw_idx
     compute statistics;
```

```
create bitmap index W_SLS_ORD_LN_F_BJI02_tmp
      on W_SOLF_TMP(D.DAY_DT,
                    U.FULL_NAME,
                    O.ORDER_TYPE_NAME,
                    P.PAYMENT_TERM_CODE,
                    L.BUSN_LOC_TYPE)
      from    W_DAY_D                    D,
              W_USER_D                   U,
              WC_ORDER_TYPE_D            O,
              W_PAYMENT_TERMS_D          P,
              W_BUSN_LOCATION_D          L,
              W_SOLF_TMP                 F
      where   F.ORDERED_ON_DT_WID = D.ROW_WID
      and     F.CREATED_BY_WID = U.ROW_WID
      and     F.X_ORDER_TYPE_WID = O.ROW_WID
      and     F.PAYMENT_TERMS_WID = P.ROW_WID
      and     F.PLANT_LOC_WID = L.ROW_WID
      tablespace dw_idx
      compute statistics;
```

```
SQL> alter table w_sales_order_line_f
  2           exchange partition p201202
  3           with table w_solf_tmp
  4                   including indexes
  5                   without validation
  6                   update global indexes;

Table altered.

Elapsed: 00:00:57.02
```

Summary

- Dimensional data models require *star transformations* in Oracle
- Bitmap-join indexes are a mechanism to *force* or *enhance* *star transformations*
  - Think of them as “materialized star transformations” for indexes, similar to “materialized views” for tables?

Thank You!

Tim's contact info:

- Web: http://www.EvDBT.com
- Email: Tim@EvDBT.com