Designing for Performance

A Case Study

Paul Baumgartel

Digitask Consultants, Inc.

The Requirements

- Capture FIX messages for reporting
- Data: order ID, key-value pairs
- Replace non-normalized design
- Support heavy transaction volume
- Allow changes to message protocol
- Provide efficient interface from JDBC client
- Respond in real time to 50+ orders / second

Existing Design

- Normalized design not used based on performance concerns
- Individual keys and individual values concatenated into two strings
- Master table held order information
- Child table held one record per message; key string and value string each in one column

Existing design problems

- Concatenation of keys and values made reporting difficult
- Inquiries for information on a specific order required client parsing of key-value strings
- No provision for referential integrity
- Non-normalized design made no provision for data integrity

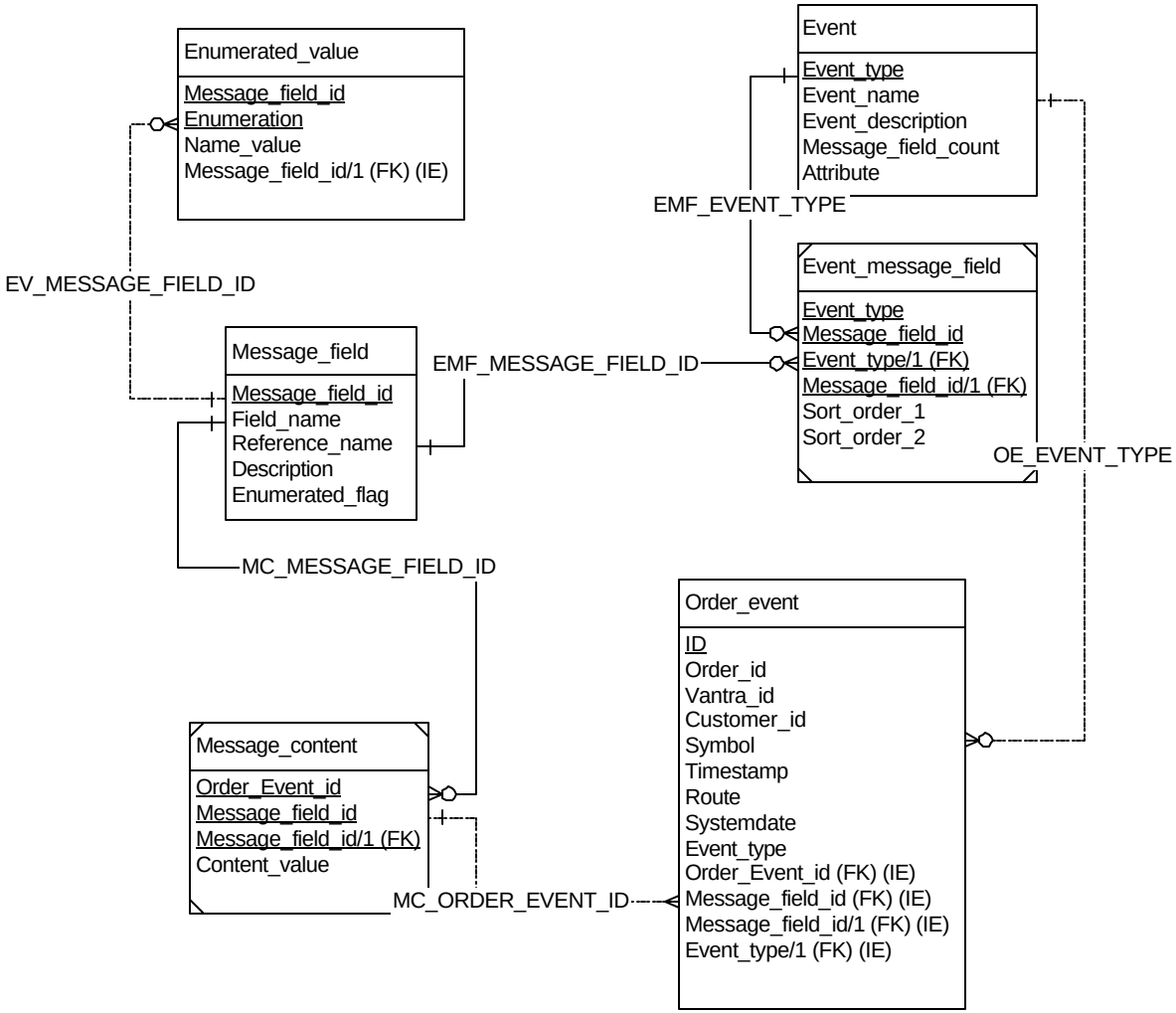
New Design Approach

- Provide lookup tables for static data (message types, field definitions, fields associated with a message)
- New message types, fields, message composition handled by adding data values
- Encapsulate all database activity in stored procedure
- Embody message dictionary in database

Performance enhancement

- Package initialization to read lookup values into package memory
- Bulk binding of inserts
- Index-organized tables to speed lookup by primary key
- Range and hash partitioning for speed and maintainability

E-R Diagram


Enumerated_value
Message_field_id
Enumeration
Name_value
Message_field_id/1 (FK) (IE)
Event
Event_type
Event_name
Event_description
Message_field_count
Attribute
EMF_EVENT_TYPE
EV_MESSAGE_FIELD_ID
Event_message_field
Event_type
Message_field_id
Event_type/1 (FK)
Message_field_id/1 (FK)
Sort_order_1
Sort_order_2
Message_field
Message_field_id
Field_name
Reference_name
Description
Enumerated_flag
EMF_MESSAGE_FIELD_ID
OE_EVENT_TYPE
MC_MESSAGE_FIELD_ID
Order_event
ID
Order_id
Vantra_id
Customer_id
Symbol
Timestamp
Route
Systemdate
Event_type
Order_Event_id (FK) (IE)
Message_field_id (FK) (IE)
Message_field_id/1 (FK) (IE)
Event_type/1 (FK) (IE)
Message_content
Order_Event_id
Message_field_id
Message_field_id/1 (FK)
Content_value
MC_ORDER_EVENT_ID

Package source

```
CREATE OR REPLACE PACKAGE CREATE_EVENT
AS
  type number_t is table of number index by binary_integer;
  type varchar_t is table of varchar2(256) index by binary_integer;
  type event_type_t is table of event.event_type%type index
   by binary_integer;
  type event_message_field_count_t is table of
   event.message_field_count%type index by binary_integer;

--  These are PL/SQL tables.  The first holds the various event_types
--  (numeric ids); the second holds the message field counts for each
--  event type.  The numeric id is used at runtime as an index
--  into this table.

  event_types event_type_t;
  event_message_field_counts event_message_field_count_t;

--   Gets the event type IDs.

  cursor get_event_types is
    select event_type from event order by event_type;

-- Gets the field count for each event type.

  cursor get_event_info is select event_type, message_field_count from
  event order by event_type;
```

Procedure declaration

```
-- The procedure to insert an event.

procedure insert_event(
       order_id_p in varchar2,
       vantra_id_p in varchar2,
       event_type_p in varchar2,
       customer_id_p in varchar2,
       symbol_p in varchar2,
       timestamp_p in number,
       route_p in varchar2,
       systemdate_p in date,
       File_Seq_p in varchar2,
       File_Offset_p in number,
       Keys_p in varchar_t,
       Vals_p in varchar_t);

END CREATE_EVENT;
/
```

Package body declarations

```
CREATE OR REPLACE PACKAGE BODY CREATE_EVENT
AS
    procedure insert_event(
       order_id_p in varchar2,
       vantra_id_p in varchar2,
       event_type_p in varchar2,
       customer_id_p in varchar2,
       symbol_p in varchar2,
       timestamp_p in number,
       route_p in varchar2,
       systemdate_p in date,
       File_Seq_p in varchar2,
       File_Offset_p in number,
       Keys_p in varchar_t,
       Vals_p in varchar_t) is

       v_id integer;
       v_fcount integer;
       v_fid integer;
       v_oeid number_t;
       v_mfid number_t;
       v_mc varchar_t;
```

Package body source 1

```
begin

-- Get a unique ID for this event.

      select events.nextval into v_id from dual;

-- Insert the event header row.

      insert into order_event values (v_id, order_id_p, vantra_id_p,
                  customer_id_p,
                  symbol_p, timestamp_p, route_p, systemdate_p,
                  event_type_p);

-- Prepare three arrays for insert into message_content, which contains
-- the event information;
-- the array size comes from the message_field_count for this event_type.

      for v_fcount in 1..event_message_field_counts(event_type_p) LOOP

-- Get the message_field_id for this keyword.

      select message_field_id into v_fid
          from message_field where field_name = keys_p(v_fcount);
```

Package body source 2

```
-- Insert the order_event_id, message_field_id, and value into the three arrays.

        v_oeid(v_fcount) := v_id;
        v_mfid(v_fcount) := v_fid;
        v_mc(v_fcount) := vals_p(v_fcount);
      end loop;

-- Use bulk binding to pass all three arrays to the SQL engine
-- for insertion into message_content.

      FORALL i IN 1..event_message_field_counts(event_type_p)
        INSERT INTO message_content (order_event_id, message_field_id,
                                     content_value)
        VALUES (v_oeid(i), v_mfid(i), v_mc(i));

      commit;
      end;
```

Package initialization

```
-- Package initialization.  Here, we read in the event
-- types and their corresponding message field counts.
-- We populate the nth element of the PL/SQL with the message field count
-- for event_type n, so that the event_type can be used as an index into
-- the table, eliminating the need for searching.

    begin

    open get_event_info;

    for etype in get_event_types
      loop
        fetch get_event_info into event_types(etype.event_type),
              event_message_field_counts(etype.event_type);
      end loop;

    close get_event_info;

    END CREATE_EVENT;
/
```

Summary

- Normalized, extensible design
- Full use of Oracle performance-enhancing techniques
- Satisfies real-time performance requirements

Contact Information

- paul.baumgartel@aya.yale.edu
- 917 549-4717