Portrait Dialogue

Partitioning tables in Portrait Dialogue

Release 6.2

Pitney Bowes Software

Portrait Software Support

If you need help with something that's not covered by this documentation, try the Knowledge Base on our web site. **http://support.portraitsoftware.com** and follow the links to your product.

You can also download other Portrait Software documentation from the site. If you don't have a username and password—or you've forgotten them—please contact us through one of the channels below.

If you find a problem with the use, installation, or documentation of this product, please contact us using any of the following methods:

Email: software.support@pb.com

Phone

- USA/Canada 1-800-335-3860 (toll-free)
- Rest of world +44 800 840 0001

When you report a problem, it helps if you can tell us:

- The name of the software application
- The circumstances in which the problem arose
- What error messages you saw (if any);
- The version of the software that you were using.

Pitney Bowes Software Inc.

January 9, 2015

Contents

Introduction

This document acts as a guideline that can be used by IT specialists and database administrators to increase Portrait Dialogue (application server) performance by using database partitioning.

NB Creating and managing partitions are complex tasks and requires in depth knowledge of the database platform used.

Related documents

Portrait Dialogue 6.1 release notes

Software release

Portrait Dialogue 6.1

Overview

About partitioning

Partitioning enhances the performance, manageability, and availability of database objects and allows tables and indexes to be subdivided into smaller pieces, enabling them to be managed and accessed at a finer level of granularity.

Partitioning is supported by both Microsoft SQL Server (since 2005) and Oracle (since 8), both of these DBMS platforms are supported by Portrait Dialogue.

Portrait Software has conducted tests to investigate partioning performance gains in large installations containing millions of rows of data; results are discussed in the following sections.

Partitioning the MESSAGE_LOG table

The *message_log* table was partitioned using the *ml_mbl_id* (the foreign key to the *message_bundle_log* table) column as a key and tests were conducted to investigate any performance gains when generating and sending messages.

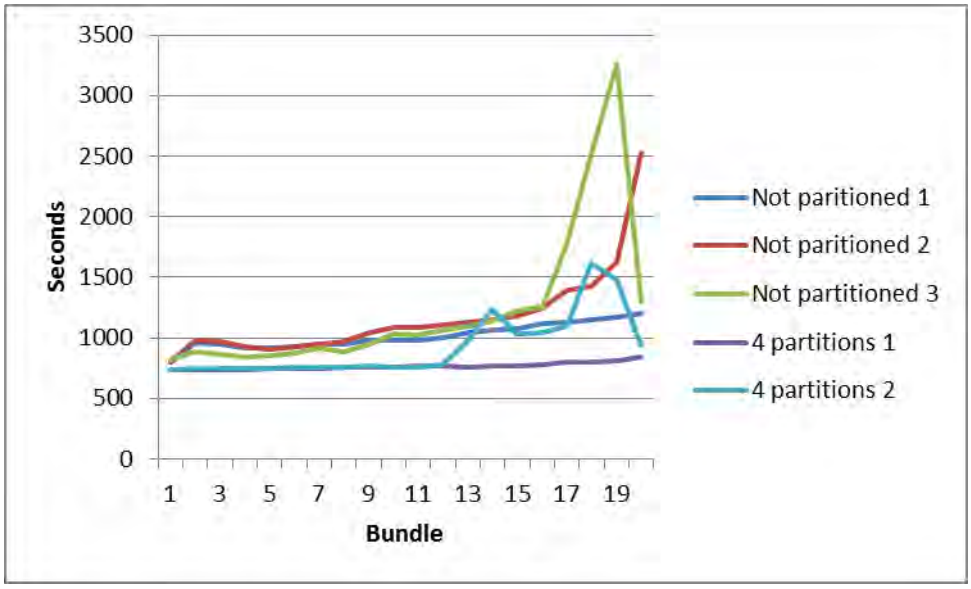
The tests showed that performance when generating messages increases on Oracle database but not on SQL Server. No performance gains were found on either platform when sending messages.

Partitioning on Oracle RDBMS

Partitioning was done on the *message_log* table only, using hash partitioning with four partitions. All partitions were put on the same tablespace. The script used is shown in Appendix A. An empty database was partitioned and messages were then generated and sent.

Generating messages

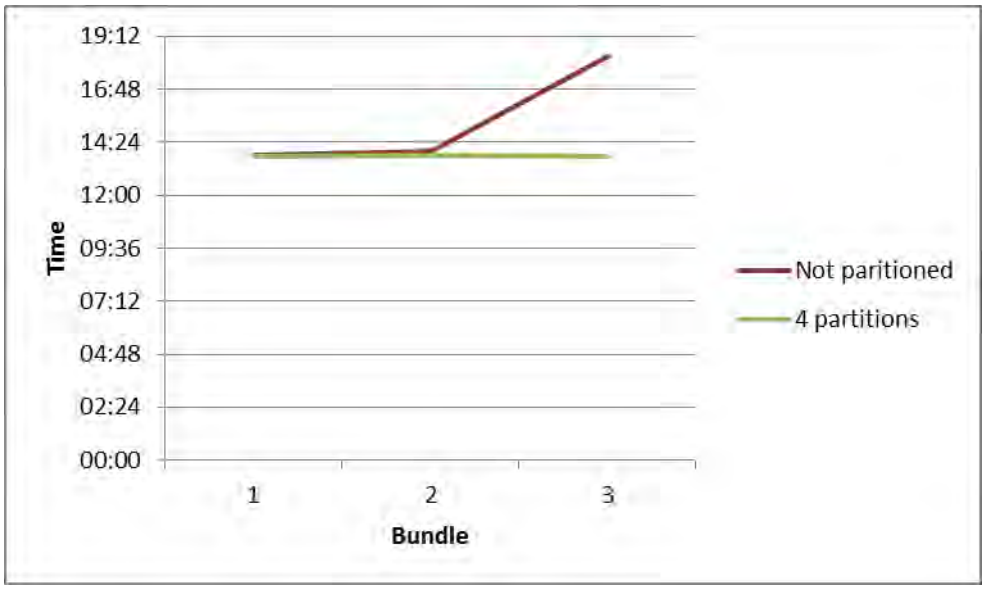
An operation was set up to generate 4 million email messages of approximately 9 KB each. The bundle size was set to 200 000 resulting in 20 bundles. We ran two tests with partitioning and three without and recorded the time it took to generate each bundle.



As can be seen in the chart, partitioning increased performance with at least 20%, and the performance gain increased as the number of records increased.

Sending messages

The system was configured to send 100 messages per transaction in 8 threads. We then timed sending 3 different bundles of 200 000 messages each. The email server used was Port 25 configured as a "black hole".



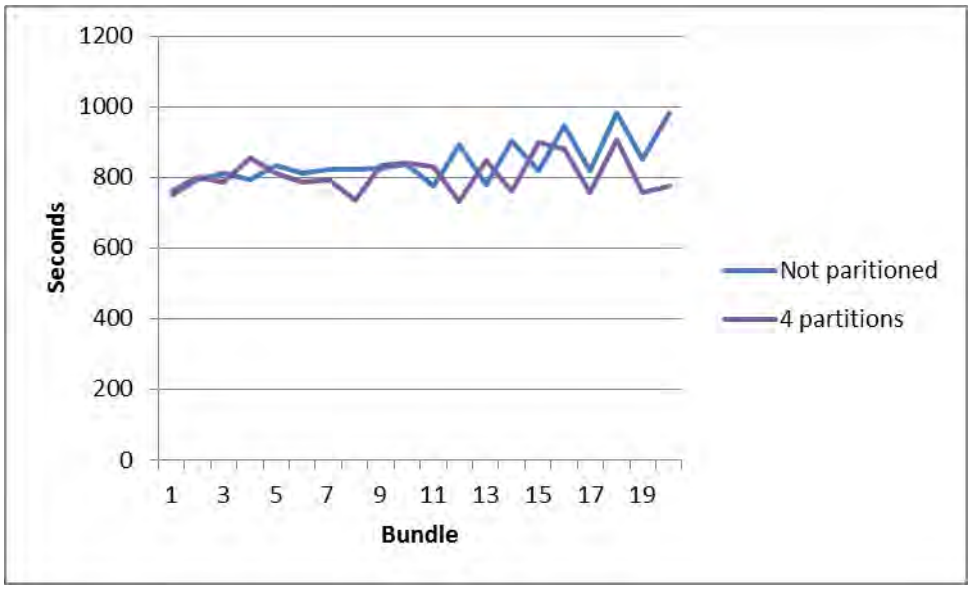
As can be seen in the chart, no performance gain was achieved via partioning. The peak in the "Not partitioned" graph at the end can be attributed to the Oracle RDBMS doing some internal background tasks.

Partitioning on Microsoft SQL Server

Partitioning was performed on the *message_log* and *customer_message* tables with corresponding indexes, using range partitioning with five initial partitions. Partition boundaries were set to have five message bundles in each partition. All partitions were put in the same filegroup. The script used is shown in Appendix B. An empty database was partitioned and messages were then generated and sent.

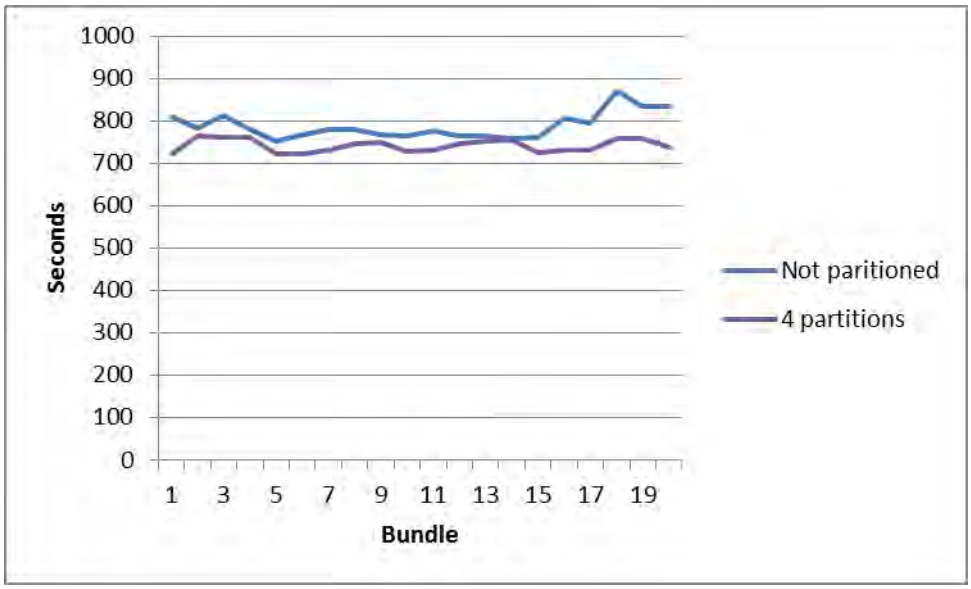
Generating messages

An operation was set up to generate 4 million email messages of approximately 9 KB each. The bundle size was set to 200 000 resulting in 20 bundles. We ran one test with partitioning and one without and recorded the time it took to generate each bundle.



As can be seen in the chart, partitioning seems to increase performance but not by a great deal. Higher performance gain is noticed as the number of message bundles increases, but the results are not definite.

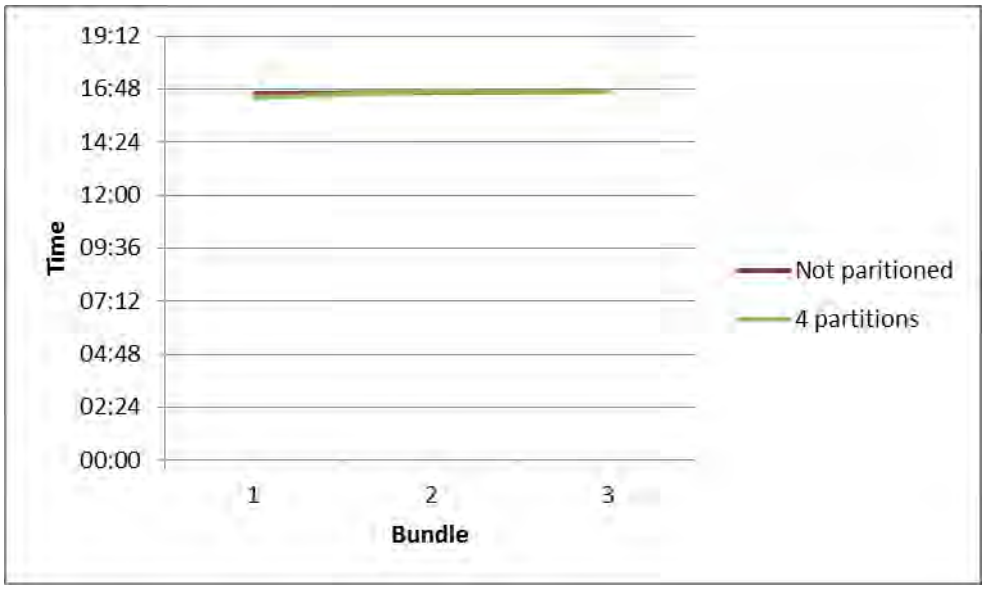
We then increased the number of partitions to nine to make room for more message bundles and ran the same tests again with 4 million messages already in the database.



As can be seen in the chart, partitioning seems to increase performance but is not significant. We no longer see higher performance gain as the number of message bundles increases.

Sending messages

Our test system was configured to send 100 messages per transaction in 6 threads. We then timed sending 3 different bundles of 200 000 messages each. The email server used was Port 25 configured as a "black hole".



As can be seen in the chart, no performace gain was observed.

Partitioning the DLG_PARTICIPANT table

Tests were done where the *dlg_participant* table was partitioned using the *dp_dlg_id* (the foreign key to the *dialogue* table) column as a key. This was done to investigate if it could increase performance when executing operations.

Tests performed on Oracle database didn't show any performance gains. Tests were not conducted on SQL Server.

Conclusions

Partitioning tables is a rather advanced task and requires good knowledge of the database platform used. This is especially true for Microsoft SQL Server, where the lack of hash partitioning requires the database administrator to add partitions as the database grows. Therefore it is advised that careful considerations are made before taking this route.

That said, our tests show that systems running on Oracle platform can obtain performance gains when the *message_log* table is partitioned

Appendix A

Script used to partition the *message_log* table on Oracle database.

NB The script will delete and recreate the table.

If you plan to use it (or a modified version) on a database with existing data, you must export data into a temporary table and then import it after the table has been recreated. The import should be done before enabling constraints.

```
- Drop existing table
ALTER TABLE CUSTOMER_ACTIVITY DROP CONSTRAINT FK_CA_2_ML
/
ALTER TABLE CUSTOMER_MESSAGE DROP CONSTRAINT FK_CM_2_ML
/
ALTER TABLE LOCKED_MESSAGE DROP CONSTRAINT FK_LM_2_ML
/
ALTER TABLE MESSAGE_OUTBOX DROP CONSTRAINT FK_MO_2_ML
/
--
-- TRIGGER: UNIQUE_MESSAGE_LOG
--
DROP TRIGGER UNIQUE_MESSAGE_LOG
/
--
-- TABLE: MESSAGE_LOG
--
DROP TABLE MESSAGE_LOG
/
-- Recreate table
--
-- TABLE: MESSAGE_LOG
--
CREATE TABLE MESSAGE_LOG(
    ML_ID                     NUMBER(18, 0)     NOT NULL,
    ML_MBL_ID                 NUMBER(9, 0)      NOT NULL,
    ML_BOUNCE_CODE            NUMBER(9, 0),
    ML_CONTENT_UNC            NVARCHAR2(254),
    ML_CONTENT_RAW            NCLOB,
    ML_CONTENT_RAW_BINARY     BLOB,
    ML_CONTROL_PARAMS         NCLOB,
    ML_MESSAGE_IDENTIFIER     NVARCHAR2(32),
    ML_IS_SENT                NVARCHAR2(1)      DEFAULT 'F' NOT NULL,
    ML_SENT_TIMESTAMP         DATE,
    ML_BOUNCED                NVARCHAR2(1)      DEFAULT 'F' NOT NULL,
    ML_BOUNCED_TIMESTAMP      DATE,
    ML_IS_ERRONEOUS           NVARCHAR2(1)      DEFAULT 'F' NOT NULL,
    ML_ERROR_TIMESTAMP        DATE,
    ML_ERROR_MSG              NCLOB,
    ML_INTERNAL_ID            NUMBER(18, 0),
    ML_CHANGED_TIMESTAMP      DATE              NOT NULL,
    ML_CHANGED_BY_USER_NAME   NVARCHAR2(32)     NOT NULL,
```

```
    CHECK (ML_IS_SENT in ('T','F')),
    CHECK (ML_BOUNCED in ('T','F')),
    CHECK (ML_IS_ERRONEOUS in ('T','F')),
    CONSTRAINT XPKMESSAGE_LOG PRIMARY KEY (ML_ID))
    PARTITION BY HASH (ML_MBL_ID)
    (PARTITION message_log_p1 TABLESPACE users,
    PARTITION message_log_p2 TABLESPACE users,
    PARTITION message_log_p3 TABLESPACE users,
    PARTITION message_log_p4 TABLESPACE users)
/
--
-- TRIGGER: UNIQUE_MESSAGE_LOG
--
CREATE OR REPLACE TRIGGER UNIQUE_MESSAGE_LOG BEFORE INSERT ON MESSAGE_LOG
REFERENCING NEW AS N
FOR EACH ROW
WHEN (N.ML_ID IS NULL)
BEGIN
  SELECT MESSAGE_LOG_SEQ.NEXTVAL INTO :N.ML_ID FROM DUAL;
END;
/
ALTER TABLE MESSAGE_LOG ADD CONSTRAINT FK_ML_2_BEC
    FOREIGN KEY (ML_BOUNCE_CODE)
    REFERENCES BOUNCE_EMAIL_CODE(BEC_BOUNCE_CODE)
/
ALTER TABLE MESSAGE_LOG ADD CONSTRAINT FK_ML_2_MBL
    FOREIGN KEY (ML_MBL_ID)
    REFERENCES MESSAGE_BUNDLE_LOG(MBL_ID)
/
--
-- INDEX: IDX_ML_INTERNAL_ID
--
CREATE INDEX IDX_ML_INTERNAL_ID ON MESSAGE_LOG(ML_INTERNAL_ID)
/
--
-- INDEX: IDX_ML_MBL_IS_SENT
--
CREATE INDEX IDX_ML_MBL_IS_SENT ON MESSAGE_LOG(ML_MBL_ID, ML_IS_SENT)
/
--
-- INDEX: IDX_ML_CUSTOM_IDENTIFIER
--
CREATE INDEX IDX_ML_CUSTOM_IDENTIFIER ON MESSAGE_LOG(ML_MESSAGE_IDENTIFIER)
/
--
-- INDEX: IDX_ML_BOUNCED
--
CREATE INDEX IDX_ML_BOUNCED ON MESSAGE_LOG(ML_BOUNCED, ML_MBL_ID)
/
--
-- INDEX: IDX_ML_PK_MBL_ID
```

```
--
CREATE INDEX IDX_ML_PK_MBL_ID ON MESSAGE_LOG(ML_ID, ML_MBL_ID)
/
--
-- INDEX: FK_ORA_ML_2_MBL
--
CREATE INDEX FK_ORA_ML_2_MBL ON MESSAGE_LOG(ML_MBL_ID)
/
--
-- INDEX: FK_ORA_ML_2_BEC
--
CREATE INDEX FK_ORA_ML_2_BEC ON MESSAGE_LOG(ML_BOUNCE_CODE)
/
ALTER TABLE CUSTOMER_ACTIVITY ADD CONSTRAINT FK_CA_2_ML
    FOREIGN KEY (CA_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
/
ALTER TABLE CUSTOMER_MESSAGE ADD CONSTRAINT FK_CM_2_ML
    FOREIGN KEY (CM_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
/
ALTER TABLE LOCKED_MESSAGE ADD CONSTRAINT FK_LM_2_ML
    FOREIGN KEY (LM_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
/
ALTER TABLE MESSAGE_OUTBOX ADD CONSTRAINT FK_MO_2_ML
    FOREIGN KEY (MO_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
/
```

Appendix B

Script used to partition the *message_log* and *customer_*message tables and indexes on a SQL Server database.

NB Script will delete and recreate the tables.

If you plan to use it (or a modified version) on a database with existing data, you must export data into temporary tables and then import it after the tables have been recreated. The import should be done before enabling constraints.

```
/* Drop existing tables */
/*
 * TABLE: CUSTOMER_MESSAGE
 */
DROP TABLE CUSTOMER_MESSAGE
go
/*
 * TABLE: MESSAGE_LOG
 */
ALTER TABLE CUSTOMER_ACTIVITY DROP CONSTRAINT FK_CA_2_ML
go
ALTER TABLE LOCKED_MESSAGE DROP CONSTRAINT FK_LM_2_ML
go
ALTER TABLE MESSAGE_OUTBOX DROP CONSTRAINT FK_MO_2_ML
go
DROP TABLE MESSAGE_LOG
go
/* Recreate tables */
CREATE PARTITION FUNCTION PF_MESSAGE_LOG (int)
AS RANGE LEFT FOR VALUES (1005, 1010, 1015, 1020)
go
CREATE PARTITION SCHEME PS_MESSAGE_LOG
AS PARTITION PF_MESSAGE_LOG
ALL TO ('PRIMARY')
go
/*
 * TABLE: MESSAGE_LOG
 */
CREATE TABLE MESSAGE_LOG(
    ML_ID                    bigint           IDENTITY(1,1),
    ML_MBL_ID                int              NOT NULL,
    ML_BOUNCE_CODE           int              NULL,
    ML_CONTENT_UNC           nvarchar(254)    NULL,
    ML_CONTENT_RAW           ntext            NULL,
    ML_CONTENT_RAW_BINARY    image            NULL,
    ML_CONTROL_PARAMS        ntext            NULL,
    ML_MESSAGE_IDENTIFIER    nvarchar(32)     NULL,
    ML_IS_SENT               nvarchar(1)      NOT NULL,
    ML_SENT_TIMESTAMP        datetime         NULL,
    ML_BOUNCED               nvarchar(1)      NOT NULL,
```

```
    ML_BOUNCED_TIMESTAMP       datetime         NULL,
    ML_IS_ERRONEOUS            nvarchar(1)      NOT NULL,
    ML_ERROR_TIMESTAMP         datetime         NULL,
    ML_ERROR_MSG               nvarchar(max)    NULL,
    ML_INTERNAL_ID             bigint           NULL,
    ML_CHANGED_TIMESTAMP       datetime         NOT NULL,
    ML_CHANGED_BY_USER_NAME    nvarchar(32)     NOT NULL,
    CONSTRAINT XPKMESSAGE_LOG PRIMARY KEY NONCLUSTERED (ML_ID)
)
go
EXEC sp_bindefault DEFAULT__F_, "MESSAGE_LOG.ML_IS_SENT"
go
EXEC sp_bindefault DEFAULT__F_, "MESSAGE_LOG.ML_BOUNCED"
go
EXEC sp_bindefault DEFAULT__F_, "MESSAGE_LOG.ML_IS_ERRONEOUS"
go
EXEC sp_bindrule valid_true_false, "MESSAGE_LOG.ML_IS_SENT"
go
EXEC sp_bindrule valid_true_false, "MESSAGE_LOG.ML_BOUNCED"
go
EXEC sp_bindrule valid_true_false, "MESSAGE_LOG.ML_IS_ERRONEOUS"
go

CREATE CLUSTERED INDEX PC_MESSAGE_LOG ON MESSAGE_LOG (ML_MBL_ID)
ON PS_MESSAGE_LOG(ML_MBL_ID)
go/*
 * INDEX: IDX_ML_INTERNAL_ID
 */
CREATE INDEX IDX_ML_INTERNAL_ID ON MESSAGE_LOG(ML_INTERNAL_ID)
go

/*
 * INDEX: IDX_ML_MBL_IS_SENT
 */
CREATE INDEX IDX_ML_MBL_IS_SENT ON MESSAGE_LOG(ML_MBL_ID, ML_IS_SENT)
go

/*
 * INDEX: IDX_ML_CUSTOM_IDENTIFIER
 */
CREATE INDEX IDX_ML_CUSTOM_IDENTIFIER ON MESSAGE_LOG(ML_MESSAGE_IDENTIFIER)
go

/*
 * INDEX: IDX_ML_BOUNCED
 */
CREATE INDEX IDX_ML_BOUNCED ON MESSAGE_LOG(ML_BOUNCED, ML_MBL_ID)
go

/*
 * INDEX: IDX_ML_PK_MBL_ID
```

```
 */
CREATE UNIQUE INDEX IDX_ML_PK_MBL_ID ON MESSAGE_LOG(ML_ID, ML_MBL_ID)
go

ALTER TABLE CUSTOMER_ACTIVITY ADD CONSTRAINT FK_CA_2_ML
    FOREIGN KEY (CA_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
go
ALTER TABLE LOCKED_MESSAGE ADD CONSTRAINT FK_LM_2_ML
    FOREIGN KEY (LM_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
go
ALTER TABLE MESSAGE_OUTBOX ADD CONSTRAINT FK_MO_2_ML
    FOREIGN KEY (MO_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
go
/*
 * TABLE: CUSTOMER_MESSAGE
 */

CREATE TABLE CUSTOMER_MESSAGE(
    CM_ID             bigint            IDENTITY(1,1),
    CM_ML_ID          bigint            NOT NULL,
    CM_DP_ID          bigint            NULL,
    CM_MBL_ID         int               NOT NULL,
    CM_CUSTOMER_ID    varchar(40)       NOT NULL,
    CM_CONTEXT        nvarchar(128)     NOT NULL,
    CM_OFFER_MASK     varbinary(128)    NULL,
    CONSTRAINT XPK_CM PRIMARY KEY NONCLUSTERED (CM_ID)
)
go
EXEC sp_bindefault DEFAULT_string_zero, "CUSTOMER_MESSAGE.CM_CONTEXT"
go
CREATE CLUSTERED INDEX PC_CUSTOMER_MESSAGE ON CUSTOMER_MESSAGE (CM_MBL_ID)
ON PS_MESSAGE_LOG (CM_MBL_ID)
go
/*
 * INDEX: FK_CM_2_ML
 */
CREATE INDEX FK_CM_2_ML ON CUSTOMER_MESSAGE(CM_ML_ID)
go
/*
 * INDEX: FK_CM_2_DP
 */
CREATE INDEX FK_CM_2_DP ON CUSTOMER_MESSAGE(CM_DP_ID)
go
/*
 * INDEX: IDX_CM_CUSTOMER_ID
 */
CREATE INDEX IDX_CM_CUSTOMER_ID ON CUSTOMER_MESSAGE(CM_CUSTOMER_ID)
go
```

```
/*
 * INDEX: FK_CM_2_MBL
 */
CREATE INDEX FK_CM_2_MBL ON CUSTOMER_MESSAGE(CM_MBL_ID)
go
/*
 * INDEX: IDX_CM_PK_ML_ID
 */
CREATE INDEX IDX_CM_PK_ML_ID ON CUSTOMER_MESSAGE(CM_ID, CM_ML_ID)
go
/*
 * INDEX: IDX_CM_PK_MBL_ID
 */
CREATE INDEX IDX_CM_PK_MBL_ID ON CUSTOMER_MESSAGE(CM_ID, CM_MBL_ID)
go
ALTER TABLE CUSTOMER_MESSAGE ADD CONSTRAINT FK_CM_2_MBL
    FOREIGN KEY (CM_MBL_ID)
    REFERENCES MESSAGE_BUNDLE_LOG(MBL_ID)
go
ALTER TABLE CUSTOMER_MESSAGE ADD CONSTRAINT FK_CM_2_ML
    FOREIGN KEY (CM_ML_ID)
    REFERENCES MESSAGE_LOG(ML_ID)
go
```