311-CD-106-001

EOSDIS Core System Project

Version 2.0 Planning and Data Processing Subsystem Database Design and Schema Specifications for the ECS Project

This document has not yet been approved by the Government for general use or distribution.

Draft

January 1998

Raytheon Systems Company
Upper Marlboro, Maryland

Version 2.0 Planning and Data Processing Subsystem Database Design and Schema Specifications for the ECS Project

DRAFT

January 1998

Prepared Under Contract NAS5-60000
CDRL Item #050

RESPONSIBLE ENGINEER

Mary S. Armstrong for /s/ 12/31/97

Maureen Magunda Date
EOSDIS Core System Project

SUBMITTED BY

T. W. Fisher /s/ 12/31/97

Terry Fisher, ECS CCB Chairman Date
EOSDIS Core System Project

Raytheon Systems Company
Upper Marlboro, Maryland

311-CD-106-001

This page intentionally left blank.

Preface

This document describes the data design and database specification for the Planning and Data Processing Subsystem. It is one of ten documents comprising the detailed database design specifications for each of the ECS subsystems.

The subsystem database design specifications for the as delivered system include:

311-CD-100 Access Control (ACL) Subsystem Database Design and Database Schema Specifications for the ECS Project

311-CD-101 Data Distribution (DDIST) Subsystem Database Design and Database Schema Specifications for the ECS Project

311-CD-102 Data Management (DM) Subsystem Database Design and Database Schema Specifications for the ECS Project

311-CD-103 Ingest Subsystem Database Design and Database Schema Specifications for the ECS Project

311-CD-104 Interoperability Subsystem (IOS) Database Design and Database Schema Specifications for the ECS Project

311-CD-105 Management Support Subsystem (MSS) Database Design and Database Schema Specifications for the ECS Project

311-CD-106 Planning and Data Processing Subsystem (PDPS) Database Design and Database Schema Specifications for the ECS Project

311-CD-107 Science Data Server (SDSRV) Subsystem Database Design and Database Schema Specifications for the ECS Project

311-CD-108 Storage Management (STMGMT) Subsystem Database Design and Database Schema Specifications for the ECS Project

311-CD-109 Subscription Server (SUBSRV) Subsystem Database Design and Database Schema Specifications for the ECS Project

This submittal meets the milestone specified in the Contract Data Requirements List (CDRL) of NASA Contract NAS5-60000. It is a formal contract deliverable with an approval code 1. It requires Government review and approval prior to acceptance and use. This document is under ECS contractor configuration control. Once approved, contractor approved changes will be handled in accordance with Class I and lass II change control requirements described in the EOS Configuration Management Plan, and changes to this document shall be made by Document Change Notice (DCN) or by complete revision.

Entity Relationship Diagrams (ERDs) presented in this document have been exported directly from tools and some cases contain too much detail to be easily readable within hard copy page constraints. The reader is encouraged to view these drawings on-line using the Portable Document Format (PDF) electronic copy available via the ECS Data Handling System (ECS) on the world-wide web at http://edhs1.gsfc.nasa.gov.

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Raytheon Systems Company
1616 McCormick Drive
Upper Marlboro, MD 20774-5372

Abstract

This document outlines “as-built” database design and database schema of the Planning and Data Processing Subsystem database including the physical layout of the database and initial installation parameters.

Keywords: data**,** database, design, configuration, database installation, scripts, security, data model, data dictionary, replication, performance tuning, SQL server, database security, replication, database scripts

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Draft
iii through xii	Draft
1-1 and 1-2	Draft
2-1 and 2-2	Draft
3-1 through 3-2	Draft
4-1 through 4-258	Draft
5-1 and 5-2	Draft
6-1 and 6-2	Draft
7-1 and 7-2	Draft
8-1 and 8-2	Draft
A-1 through A-16	Draft
AB-1 through AB-7	

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
311-CD-106-001	Draft	January 1998	97-1755

This page intentionally left blank.

Contents

Preface

Abstract

1. Introduction

1.1 Identification 1-1

1.2 Scope 1-1

1.3 Purpose 1-1

1.4 Audience 1-1

2. Related Documents

2.1 Applicable Documents 2-1

2.2 Information Documents 2-1

3. Database Configurations

3.1 Server Configurations 3-1

3.2 Storage Device Layouts 3-1

4. Data Design

4.1 Database Overview 4-1

4.1.1 Physical Data Model Entity Relationship Diagram 4-1

4.1.2 Tables 4-2

4.1.3 Columns 4-33

4.1.4 Column Domains 4-79

4.1.5 Rules 4-79

4.1.6 Defaults 4-79

4.1.7 Views 4-79

4.1.8 Integrity Constraints....4-80
4.1.9 Triggers....4-92
4.1.10 Stored Procedures....4-254
4.2 File Usage....4-256
4.2.1 Files Definitions....4-257
4.2.2 Attributes....4-257
4.2.3 Attribute Domains....4-257

5. Performance and Tuning Factors

5.1 Indexes....5-1
5.2 Segments....5-1
5.3 Named Caches....5-1

6. Database Security

6.1 Initial Users....6-1
6.2 Groups....6-1
6.3 Roles. 6-1
6.4 Object Permissions....6-2

7. Replication

7.1 Replication Overview 7-1
7.2 Replication Definitions 7-1
7.3 Replication Subscriptions 7-1
7.4 Replication Database Configuration....7-1
7.5 Replication Server Configuration 7-1

8. Scripts

8.1 Installation Scripts.....8-1

8.2 De-Installation Scripts.....8-1

8.3 Backup/Recovery Scripts.....8-1

8.4 Miscellaneous Scripts 8-1

List of Figures

4-1 ERD Key..... 4-1

Appendix - PDPS ERDs

Abbreviations and Acronyms

This page intentionally left blank.

1. Introduction

1.1 Identification

This Planning and Data Processing Subsystem (PDPS) Database Design and Database Schema Specification document, Contract Data Requirement List (CDRL) Item #050, whose requirements are specified in Data Item Description (DID) 311/DV1, is a required deliverable under the Earth Observing System (EOS) Data and Information System (EOSDIS) Core System (ECS), Contract NAS5-60000.

1.2 Scope

The PDPS Database Design and Database Schema Specification document describes the data design and database specifications to support the data requirements of Release 2 Drop 3 PDPS software.

1.3 Purpose

The purpose of the PDPS Database Design and Database Schema Specification document is to support the maintenance of PDPS data and databases throughout the life cycle of ECS. This document communicates the database implementation in sufficient detail to support ongoing configuration management.

1.4 Audience

This document is intended to be used by ECS maintenance and operations staff. The document is organized as follows:

Section 1 provides information regarding the identification, purpose, scope and audience of this document.

Section 2 provides a listing of the related documents, which were used as a source of information for this document.

Section 3 provides a mapping data bases to hardware components.

Section 4 contains the PDPS physical data model which is the database tables, triggers, stored procedures, and flat files.

Section 5. provides a description of database performance and tuning features such as indexes, caches, and data segments.

Section 6 provides a description of the security infrastructure used and list of the users, groups, and permissions available upon initial installation.

Section 7 contains replication design and implementation details.

Section 8 provides a description of database and database related scripts used for installation, de-installation, backup/recovery, and other miscellaneous functions.

2. Related Documents

2.1 Applicable Documents

The following documents, including Internet links, are referenced in the PDPS Database Design and Database Schema Specification, or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume. Internet links cannot be guaranteed for accuracy or currency.

305-CD- V.0 Segment/Subsystem Design Specification for the ECS Project

920-TDG-009 Release B0 GSFC DAAC Database Information

920-TDN- 009 Release B0 NSIDC DAAC Database Information

920-TDE-009 Release B0 EDC DAAC Database Information

920-TDL-009 Release B0 LARC DAAC Database Information

920-TDS-009 Release B0 SMC DAAC Database Information

920-TDM-009 Release B0 Mini-DAAC Database Information

2.2 Information Documents

The following documents, although not directly applicable, amplify or clarify the information presented in this document. These documents are not binding on this document.

3. Database Configurations

3.1 Server Configurations

The database configuration of the PDPS server varies from DAAC to DAAC based on individualized DAAC requirements and hardware availability. These DAAC-specific database configurations are detailed on the following documents:

920-TDG-009 Release B0 GSFC DAAC Database Information

920-TDN- 009 Release B0 NSIDC DAAC Database Information

920-TDE-009 Release B0 EDC DAAC Database Information

920-TDL-009 Release B0 LARC DAAC Database Information

920-TDS-009 Release B0 SMC DAAC Database Information

920-TDM-009 Release B0 Mini-DAAC Database Information

These documents are maintained as part of the ECS baseline and available on the world-wide web at the URL http://pete.hitc.com/baseline/.

3.2 Storage Device Layouts

Storage Device layouts, disk partitions, vary from DAAC to DAAC based on the amount of data storage expected to be needed to accommodate a particular DAAC's storage requirements. Disk partitions for the PDPS server at each DAAC is detailed in the following documents:

922-TDG-XXX Release B0 PDPS Disk Partitions

922-TDN-XXX Release B0 PDPS Disk Partitions

922-TDE-XXX Release B0 PDPS Disk Partitions

922-TDL-XXX Release B0 PDPS Disk Partitions

922-TDS-XXX Release B0 PDPS Disk Partitions

922-TDM-XXX Release B0 PDPS Disk Partitions

These documents are maintained as part of the ECS baseline and available on the world-wide web at the URL http://pete.hitc.com/baseline/.

This page intentionally left blank.

4. Data Design

4.1 Database Overview

The PDPS database implements the large majority of the persistent data requirements for the PDPS subsystem. The database is designed in such a manner as to satisfy business policy while maintaining data integrity and consistency. Database tables are implemented using the Sybase Relational Database Management system (DBMS) version 11.0.1. All components of the PDPS database are described in the section which follow in sufficient detail to support maintenance needs.

4.1.1 Physical Data Model Entity Relationship Diagram

The Entity Relationship Diagram(ERD) presents a schematic depiction of the PDPS physical data model. The ERDs presented here for the PDPS database were produced using the S-Designor Data Architect Computer Aided Software Engineering (CASE) tool. ERDs represent the relationship between entities or database tables. The key for the symbols used in the ERDs follows.

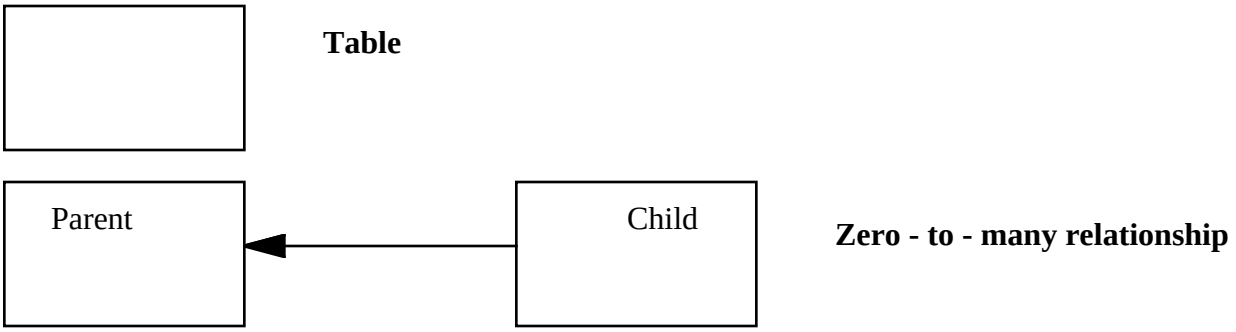


Figure 4-1. ERD Key

The ERDs for the PDPS database are shown in Appendix A – Planning and Data Processing Subsystem ERDs

4.1.2 Tables

A listing of each the tables in the PDPS database is given here. A brief definition of each of these tables follows.

Table
DpPrAutosysMapList
DpPrAutosysMapList
DpPrDiskAllocation
DpPrDprDependList
DpPrGranuleLocation
DpPrInputData
DpPrPcf
DpPrPge
DpPrPgeExitDependency
DpPrPgeExitDependency
DpPrPgeExitMessage
DpPrPgeExitMessage
DpPrRscCpuRamAllocation
PlActivePlan
PlAlternate
PlAlternateInputValues
PlCluster
PlDasDelta
PlDataGranuleShort
PlDataHistory
PlDataProcessingRequest
PlDataSourceMaster
PlDataTypeFiles
PlDataTypeReq
PlDatRecord
PlDprCompletion
PlDprData
PlErrorAction
PlEsdtParam
PlEsdtParmValues
PlFileTypeReq
PlGroundEventResourceExplosion
PlMetadataChecks
PlModeRecord
PlOnDemandPR
PlOnDemandReplanValues
PlOutputDataYield
PlOutputFileType
PlPgeDetailData
PlPgeDetailData
PlPgeDetailParameters
PlPgeDetailTime
PlPgeMaster
PlPgePerformance
PlPgePriority

Table (continued)
PlPlanMetadata
PlPlans
PlPrCollectionExplosion
PlPrMetadataValue
PlProductionRequest
PlPrParameter
PlPrParameter
PlPRPriority
PlPrVsPlan
PlRoutineArrival
PlRscComputer
PlRscComputer
PlRscDiskPartition
PlRscDiskPartition
PlRscRealComputer
PlRscRealComputer
PlRscString
PlTile
PlTileCoordinates
PlTileScheduled
PlTileScheduled
PlUsedByCenter
PlUserParameterConstraints
PlUserParameterConstraints
PlUserPriority
RpReservationInterval
RpResourceReservation
RpRsvRscExplosion

Table: DpPrAutosysMapList

Description

Maintains a mapping of instances of Autosys to physical resources.

Column List

Name	Code	Type	P	M
autosysid	autosysId	varchar(20)	No	Yes
autosysidkey	autosysIdKey	int	Yes	Yes
resourcestring	resourceString	varchar(20)	No	Yes

Table: DpPrCreationQueue

Description

Maintains a list of DPRs currently being processed by Autosys.

Column List

Name	Code	Type	P	M
autosysid	autosysId	varchar(20)	No	Yes
dprid	dprId	varchar(27)	Yes	Yes
hold	hold	bit	No	Yes
priority	priority	int	No	Yes

Table: DpPrDiskAllocation

Description

Describes the disk resources allocated for a job.

Column List

Name	Code	Type	P	M
computerid	computerId	int	No	Yes
diskallocationactual	diskAllocationActual	int	No	No
diskallocationid	diskAllocationId	numeric(9,0)	Yes	Yes
diskallocationpath	diskAllocationPath	varchar(255)	No	No
diskallocationsize	diskAllocationSize	int	No	No
diskallocationtype	diskAllocationType	smallint	No	Yes
diskallocationuser	diskAllocationUser	varchar(27)	No	No
diskpartitionid	diskPartitionId	int	No	Yes

Table: DpPrDprDependList

Description

A list of DPRs which depend on the completion of the current DPR.

Column List

Name	Code	Type	P	M
dependentdprid	dependentDprId	varchar(27)	Yes	Yes
dprid	dprId	varchar(27)	Yes	Yes

Table: DpPrExecutable

Description

Contains information about a particular execution of a PGE.

Column List

Name	Code	Type	P	M
exec_location	execLocation	varchar(240)	No	No
exec_name	execName	varchar(60)	Yes	Yes
exec_permission	execPermission	char(5)	No	Yes
exec_shell	execShell	varchar(10)	No	Yes
execlayer	execLayer	int	No	No
execmachine	execMachine	varchar(60)	Yes	Yes
operatingsystem	operatingSystem	varchar(60)	No	No
sswid	sswId	varchar(17)	Yes	Yes

Table: DpPrFile

Description

Contains information about files housing data granules.

Column List

Name	Code	Type	P	M
datainfofilesize	fileSize	float	No	No
datainfopath	path	varchar(255)	No	No
filename	fileName	varchar(140)	Yes	Yes
filenumber	fileNumber	int	No	No
filetype	fileType	int	No	No
machineid	machineId	varchar(60)	Yes	Yes
numberofusage	numberOfUsage	int	No	No
sequencenumber	sequenceNumber	int	No	No
universalreference	universalReference	varchar(255)	No	Yes

Table: DpPrGranuleLocation

Description

Contains information about the physical location of a data granule.

Column List

Name	Code	Type	P	M
dprId	dprId	varchar(27)	No	No
granuleid	granuleId	varchar(30)	Yes	Yes
machineid	machineId	varchar(60)	Yes	Yes
stagestate	stageState	int	No	Yes

Table: DpPrInputData

Description

Information about an input granule being staged for data processing.

Column List

Name	Code	Type	P	M
input_data_id	inputDataId	numeric(7,0)	Yes	Yes
input_data_staging_percent	inputDataStagingPercent	float	No	No
pge_id	pgeId	varchar(19)	No	Yes
processing_type	processingType	char(1)	No	Yes

Table: DpPrPcf

Description

Information about a Process Control File (PCF).

Column List

Name	Code	Type	P	M
dpr_id	dprId	varchar(30)	Yes	Yes
pcf_location	pcfLocation	varchar(240)	No	No
pcf_name	pcfName	varchar(60)	No	Yes
pcf_permission	pcfPermission	char(5)	No	Yes
pcf_type	pcfType	int	Yes	Yes

Table: DpPrPge

Description

Information describing a PGE.

Column List

Name	Code	Type	P	M
cpu_resource	pgeCpu	int	No	No
dprid	dprId	varchar(30)	Yes	Yes
execmachine	execMachine	varchar(60)	No	No
execname	execName	varchar(60)	No	No
pge_commands	pgeCommands	varchar(120)	No	Yes
pge_environment	pgeEnvironment	varchar(240)	No	No
pge_shell	pgeShell	varchar(30)	No	Yes
pge_state	pgeState	int	No	Yes
sswid	sswId	varchar(17)	No	No

Table: DpPrPgeExitDependency

Description

TBS

Column List

Name	Code	Type	P	M
dependencysswid	dependencySswId	varchar(17)	Yes	Yes
exitcode	exitCode	int	Yes	Yes
exitoperation	exitOperation	varchar(3)	No	No
pgesswid	pgeSswId	varchar(17)	Yes	Yes

Table: DpPrPgeExitMessage

Description

TBS

Column List

Name	Code	Type	P	M
exitcode	exitCode	int	Yes	Yes
exitmessage	exitMessage	varchar(240)	No	No
pgesswid	pgeSswId	varchar(17)	Yes	Yes

Table: DpPrPredStagingProfile

Description

Information required to stage ahead ("predictively") the input for a PGE.

Column List

Name	Code	Type	P	M
num_of_execution	numOfExecution	int	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
processing_type	processingType	char(1)	Yes	Yes
unit_staging_time	unitStagingTime	int	No	No

Table: DpPrQaDbParameters

Description

TBS

Column List

Name	Code	Type	P	M
minmaxdatatype	minmaxDataType	varchar(20)	No	No
parm_id	parmId	numeric(7,0)	Yes	Yes
parm_keyword	parmKeyword	varchar(20)	No	Yes
parm_max	parmMax	varchar(20)	No	Yes
parm_min	parmMin	varchar(20)	No	Yes
parm_name	parmName	varchar(20)	No	Yes
parm_type	parmType	varchar(20)	No	Yes

Table: DpPrResourceLock

Description

Information for managing resource locking during data processing.

Column List

Name	Code	Type	P	M
attempts	attempts	int	No	No
ecsunit	ecsUnit	int	No	No
jobid	jobId	varchar(40)	Yes	Yes
priority	priority	int	No	No
state	state	int	No	No

Table: DpPrRscCpuRamAllocation

Description

Information about memory allocation for data processing.

Column List

Name	Code	Type	P	M
computerid	computerId	int	Yes	Yes
cpuramallocationcpucount	cpuRamAllocationCpuCount	int	No	No
cpuramallocationjobid	cpuRamAllocationJobId	varchar(40)	Yes	Yes
cpuramallocationramsize	cpuRamAllocationRamSize	float	No	No

Table: PlActivePlan

Description

Contains information about the currently active plan.

Column List

Name	Code	Type	P	M
is_currently_active	ISCURRENTLYACTIVE	bit	No	Yes
plan_end_time	PLANENDTIME	datetime	No	No
plan_start_time	PLANSTARTTIME	datetime	No	No
planid	PLANID	varchar(20)	Yes	Yes

Table: PlAlternate

Description

Contains information about alternate inputs to PGEs.

Column List

Name	Code	Type	P	M
data_type_id	DATATYPEID	varchar(20)	Yes	Yes
default_order	DEFAULTORDER	int	No	No
default_timer	DEFAULTTIMER	integer	No	No
logical_id	logicalId	int	No	Yes
logicalId	LOGICALID	int	Yes	Yes
pge_id	PGEID	varchar(19)	Yes	Yes
primary_type	PRIMARYTYPE	varchar(20)	No	No
runtimeid	RUNTIMEID	int	No	No
temp_flag	TEMPORALFLAG	bit	No	Yes
waitfor	WAITFORFLAG	bit	No	Yes

Table: PlAlternateInputValues

Description

Information about alternate inputs to a production request.

Column List

Name	Code	Type	P	M
datatype	dataType	varchar(20)	Yes	Yes
logicalid	logicalId	int	Yes	Yes
order	theOrder	int	No	No
primarytype	primaryType	varchar(20)	No	No
productionrequestid	productionRequestId	varchar(20)	Yes	Yes
temporalflag	temporalFlag	bit	No	Yes
timerwait	timerWait	int	No	No
type	type	int	No	No

Table: PlCluster

Description

Defines the cluster of tiles used by a PGE.

Column List

Name	Code	Type	P	M
cluster_id	CLUSTERID	numeric(7,0)	Yes	Yes
last_orbit	LASTORBIT	int	No	No
num_tiles	NUMTILES	int	No	No
pge_id	PGEID	varchar(19)	No	Yes
predict_availability	PREDICTAVAILABILITY	datetime	No	No

Table: PlDasDelta

Description

A list of the deltas for each data type.

Column List

Name	Code	Type	P	M
criteria_id	criteriaId	numeric(7,0)	No	Yes
data_type_id	dataTypeId	varchar(20)	Yes	Yes
delta	delta	int	No	No

Table: PlDataAvailTimes

Description

Data start and stop times from the Data Availability Schedule.

Column List

Name	Code	Type	P	M
avail_times_id	availTimesId	numeric(7,0)	Yes	Yes
messageid	messageId	varchar(20)	No	No
product_origin	productOrigin	varchar(20)	No	No
start_time	startTime	datetime	No	No
stop_time	stopTime	datetime	No	No

Table: PlDataGranuleShort

Description

Basic information describing a data granule.

Column List

Name	Code	Type	P	M
absolutedeletetime	absoluteDeletionTime	datetime	No	No
availability	availability	bit	No	Yes
availability_actual	availabilityActual	datetime	No	No
availability_predicted	availabilityPredicted	datetime	No	No
baseline_time	baselineTime	datetime	No	No
data_type_id	dataTypeId	varchar(20)	No	Yes
granule_id	granuleId	varchar(30)	Yes	Yes
granulesize	granuleSize	float	No	No
predicted_staging_time	predictedStagingTime	integer	No	No
sciencegroup	scienceGroup	varchar(5)	No	No
stagestate	stageState	int	No	No
start_time	startTime	datetime	No	No
stop_time	stopTime	datetime	No	No
tile_id	tileId	int	No	No
UR	universalReference	varchar(255)	No	No

Table: PlDataHistory

Description

A record of the activity of a data granule from its source to its arrival at the Data Server.

Column List

Name	Code	Type	P	M
data_type_id	dataTypeId	varchar(20)	Yes	Yes
granule_id	granuleId	varchar(30)	No	No
last_actual_arrival	lastActualArrival	datetime	No	No
last_predicted_arrival	lastPredictedArrival	datetime	No	No
moving_avg_time	movingAvgTime	datetime	No	No
num_estimate	numEstimate	int	No	No
pge_id	pgeId	varchar(19)	No	No
std_deviation	stdDeviation	int	No	No

Table: PlDataProcessingRequest

Description

Information describing a Data ProcessingRequest (DPR).

Column List

Name	Code	Type	P	M
actual_start	actualStart	datetime	No	No
alarm_time	alarmTime	datetime	No	No
autosysid	autosysId	varchar(20)	No	No
baseline_time	baselineTime	datetime	No	No
completion_state	completionState	varchar(10)	No	No
data_start_time	dataStartTime	datetime	No	No
data_stop_time	dataStopTime	datetime	No	No
dpr_collection_id	dprCollectionId	varchar(10)	No	No
dpr_id	dprId	varchar(27)	Yes	Yes
latemessagessent	lateMessagesSent	int	No	No
name	name	varchar(27)	No	No
pge_id	pgeId	varchar(19)	No	Yes
predicted_start	predictedStart	datetime	No	No
priority	priority	int	No	No
request_id	productionRequestId	varchar(20)	No	Yes
runstate	runState	varchar(20)	No	No
sswid	sswId	varchar(17)	No	Yes

Table: PlDataSourceMaster

Description

Information about when external data will arrive in ECS.

Column List

Name	Code	Type	P	M
data_type_id	dataTypeId	varchar(20)	Yes	Yes
predicted_method	predictionMethod	varchar(18)	No	No
supplier_name	supplierName	varchar(30)	No	No

Table: PlDataTypeFiles

Description

Information about multi-file data types.

Column List

Name	Code	Type	P	M
datatypeid	dataTypeId	varchar(20)	Yes	Yes
filetypeid	fileTypeId	int	No	No
filetypename	fileTypeName	varchar(40)	Yes	Yes
maxnumfiles	maxNumFiles	int	No	No

Table: PlDataTypeMaster

Description

Basic information about a data type.

Column List

Name	Code	Type	P	M
archive_center	archiveCenter	varchar(20)	No	No
catalogue_category	catalogueCategory	varchar(15)	No	No
data_type_id	dataTypeId	varchar(20)	Yes	Yes
data_type_name	dataTypeName	varchar(20)	No	No
description	dataTypeDescription	varchar(60)	No	No
DS_UR_string	dataServUrString	varchar(255)	No	No
dynamic_flag	dynamicFlag	int	No	Yes
dynamictype	dynamicType	int	No	No
hdfflag	hdfFlag	bit	No	Yes
instrument	instrument	varchar(10)	No	No
interimlastpgetouse	interimLastPgeToUse	varchar(19)	No	No
interimlongduration	interimLongDuration	int	No	No
interimshortduration	interimShortDuration	int	No	No
maxfilesofanytype	maxFilesOfAnyType	int	No	No
maxfiletypes	maxFileTypes	int	No	No
nominal_size	nominalSize	float	No	No
platformname	platformName	varchar(20)	No	No
processing_center	processingCenter	varchar(20)	No	No
processinglevel	processingLevel	varchar(6)	No	No
provider	provider	varchar(50)	No	No
QA_subscription	qaSubscription	bit	No	Yes
spatial_flag	spatialFlag	bit	No	Yes
subscription_flag	subscriptionFlag	int	No	No
version	version	varchar(20)	No	No

Table: PlDataTypeReq

Description

Information about the association between a PGE and its input data types.

Column List

Name	Code	Type	P	M
data_type_id	dataTypeId	varchar(20)	Yes	Yes
data_type_req	dataTypeRequirement	int	No	No
endtimeoffset	endTimeOffset	integer	No	No
input_type	inputType	int	No	Yes
linkid	linkId	int	No	No
logical_id	logicalId	int	Yes	Yes
numneeded	numNeeded	int	No	No
pcffiletype	pcfFileType	int	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
QA_threshold	qaThreshold	varchar(40)	No	No
sciencegroup	scienceGroup	varchar(5)	No	No
starttimeoffset	startTimeOffset	integer	No	No
whereclause	whereClause	varchar(255)	No	No

Table: PlDatRecord

Description

Information about the predicted availability of data granules received from outside the local DAAC.

Column List

Name	Code	Type	P	M
avail_times_id	availTimesId	numeric(7,0)	No	No
data_type_id	dataTypeId	varchar(20)	Yes	Yes
datareadytime	dataReadyTime	datetime	No	No
estimatedgranulevolume	estimatedGranuleVolume	float	No	No
messageid	messageId	varchar(20)	No	No
predicted_availability	predictedAvailability	datetime	No	No
start_time	startTime	datetime	Yes	Yes
stop_time	stopTime	datetime	Yes	Yes
tileid	tileId	varchar(20)	No	No

Table: PlDprCompletion

Description

TBS

Column List

Name	Code	Type	P	M
completiondate	completionDate	datetime	Yes	Yes
dprelapsedtime	dprElapsedTime	float	No	No
dprid	dprId	varchar(27)	Yes	Yes
exitcode	exitCode	int	No	No
pgeblockinputoperations	pgeBlockInputOperations	int	No	No
pgeblockoutputoperations	pgeBlockOutputOperations	int	No	No
pgecputime	pgeCpuTime	float	No	No
pgeelapsedtime	pgeElapsedTime	float	No	No
pgemaxmemoryuse	pgeMaxMemoryUse	float	No	No
pgepagefaults	pgePageFaults	int	No	No
pgesharedmemoryuse	pgeSharedMemoryUse	float	No	No
pgeswaps	pgeSwaps	int	No	No
platform	platform	varchar(60)	No	No
systemcputime	systemCpuTime	float	No	No

Table: PlDprData

Description

Information about the relationship between a DPR and a data granule.

Column List

Name	Code	Type	P	M
accepted	accepted	int	No	Yes
datatyperequirement	dataTypeRequirement	int	No	No
dpr_id	dprId	varchar(27)	Yes	Yes
granule_id	granuleId	varchar(30)	Yes	Yes
ioflag	ioFlag	int	Yes	Yes
linkid	linkId	int	No	No
logicalid	logicalId	int	Yes	Yes
numNeeded	numNeeded	int	No	No
order	theOrder	int	No	No
primary_type	primaryType	varchar(20)	No	No
temporal_flag	temporalFlag	bit	No	Yes
timerexp	timerExp	bit	No	Yes
timerstart	timerStart	bit	No	Yes
timewait	timeWait	int	No	No
type	type	int	No	No
waitforflag	waitForFlag	bit	No	Yes

Table: PlErrorAction

Description

TBS

Column List

Name	Code	Type	P	M
action	action	int	No	Yes
alarm_text	alarmText	varchar(20)	No	No
error_action_id	errorActionId	numeric(7,0)	Yes	Yes
pge_id	pgeId	varchar(19)	No	Yes
pr_id	productionRequestId	varchar(20)	No	No
status	status	int	No	No
timeframe_data_type	timeframeDataType	int	No	No

Table: PlEsdtParam

Description

An explosion table for PlDataTypeMaster. Contains the parameter list used to retrieve the metadata associated with a newly arrived instance of the data type.

Column List

Name	Code	Type	P	M
datatypeid	dataTypeId	varchar(20)	Yes	Yes
paramname	paramName	varchar(20)	Yes	Yes
paramtype	paramType	varchar(20)	No	No

Table: PlEsdtParmValues

Description

An explosion table for PlDataGranuleShort. Contains lists of ESDT parameter values associated with a data granule, i.e. metadata values associated with the data type used to determine the suitability of the data granule for production.

Column List

Name	Code	Type	P	M
ESDT_name	esdtParmName	varchar(20)	Yes	Yes
ESDT_parm_value	esdtParmVal	varchar(80)	No	No
granule_id	granuleId	varchar(30)	Yes	Yes

Table: PlFileTypeReq

Description

An explosion table for PlDataTypeReq showing the various file types associated with a data type that is input to a PGE.

Column List

Name	Code	Type	P	M
datatypeid	dataTypeId	varchar(20)	Yes	Yes
filetype	fileType	int	Yes	Yes
logicalid	logicalId	int	Yes	Yes
maxnumfiles	maxNumFiles	int	No	No
pgeid	pgeId	varchar(19)	Yes	Yes

Table: PlGroundEvent

Description

Information about a ground event, i.e. the allocation of resources to some non-production task such as maintenance.

Column List

Name	Code	Type	P	M
description	groundEventDescription	varchar(40)	No	No
duration	groundEventDuration	int	No	No
ground_event_id	groundEventId	int	Yes	Yes
ground_event_name	groundEventName	varchar(40)	No	No
priority	groundEventPriority	int	No	No
win_end	groundEventEnd	datetime	No	No
win_start	groundEventStart	datetime	Yes	Yes

Table: PlGroundEventResourceExplosion

Description

A join table between PlGroundEvent and PlResource.

Column List

Name	Code	Type	P	M
ground_event_id	groundEventId	int	Yes	Yes
resourceid	resourceId	int	Yes	Yes
win_start	groundEventStart	datetime	No	Yes

Table: PlMetadataChecks

Description

Provides a metadata field and its corresponding value to be checked against the actual metadata of the specified input data granule when deciding if a particular input should be used or if its corresponding PGE should be executed.

Column List

Name	Code	Type	P	M
data_type_id	dataTypeId	varchar(20)	Yes	Yes
ioflag	ioFlag	int	Yes	Yes
logical_id	logicalId	int	No	Yes
metadata_oper	operator	varchar(3)	No	No
metadata_parm_name	paraName	varchar(20)	Yes	Yes
metadata_type	type	varchar(5)	No	No
metadata_value	value	varchar(40)	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
PlD_pge_id	PlD_pgeId	varchar(19)	No	Yes
PlO_logical_id	PlO_logicalId	int	No	Yes
PlO_pge_id	PlO_pgeId	varchar(19)	No	Yes

Table: PlModeRecord

Description

TBS

Column List

Name	Code	Type	P	M
availtimesid	availTimesId	numeric(7,0)	No	No
datarate	dataRate	decimal(7,4)	No	No
instrument_name	instrumentName	varchar(20)	Yes	Yes
mode_name	modeName	varchar(30)	No	Yes
mode_start_time	modeStartTime	datetime	Yes	Yes
mode_stop_time	modeStopTime	datetime	No	No
pgecollectionid	pgeCollectionId	varchar(20)	No	No

Table: PlOnDemandPR

Description

Additional information for an On Demand Production Request (in addition to that found in the production request table).

Column List

Name	Code	Type	P	M
orderid	orderId	varchar(20)	No	No
orderrequestid	orderRequestId	varchar(20)	No	No
productionrequestid	productionRequestId	varchar(20)	Yes	Yes

Table: PlOnDemandReplanValues

Description

The resource usage thresholds for replan notification: one per resourceType/userType combination.

Column List

Name	Code	Type	P	M
criteria_id	criteriaId	numeric(7,0)	No	Yes
on_demand_replan_id	onDemandReplanId	numeric(7,0)	Yes	Yes
replan_value	replanValue	float	No	No
resource_type	resourceType	int	No	Yes
usertype	userType	varchar(20)	No	No

Table: PlOutputDataYield

Description

Information about the output data granules associated with a PGE.

Column List

Name	Code	Type	P	M
data_type_id	dataTypeId	varchar(20)	Yes	Yes
linkid	linkId	int	No	No
logical_id	logicalId	int	Yes	Yes
max_size	maxSize	int	No	No
metadatalogid	metadataLogId	int	No	No
min_size	minSize	int	No	No
pcffiletype	pcfFileType	int	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
sciencegroup	scienceGroup	varchar(5)	No	No
yield	yield	int	No	No

Table: PlOutputFileType

Description

Information about file types associated with an output data granule.

Column List

Name	Code	Type	P	M
datatypeid	dataTypeId	varchar(20)	Yes	Yes
filetype	fileType	int	Yes	Yes
logicalid	logicalId	int	Yes	Yes
maxnumfiles	maxNumFiles	int	No	No
pgeid	pgeId	varchar(19)	Yes	Yes

Table: PlPgeCollection

Description

Information describing a collection of PGEs.

Column List

Name	Code	Type	P	M
data_type_id	dataTypeId	varchar(20)	No	No
default_mode	defaultMode	varchar(20)	No	Yes
pge_collection_id	pgeCollectionId	varchar(20)	Yes	Yes

Table: PlPgeDetailData

Description

Details about data requirements for scheduling a PGE.

Column List

Name	Code	Type	P	M
input_data_id	inputDataId	int	No	Yes
instances_per_pgerun	instancesPerPgerun	int	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
primary_data_source	primaryDataSource	char(30)	No	No

Table: PlPgeDetailParameters

Description

Details regarding the user-defined parameters associated with a PGE.

Column List

Name	Code	Type	P	M
default_value	pgeParameterDefault	varchar(200)	No	No
description	description	varchar(80)	No	No
logical_id	pgeParameterLogicalId	int	Yes	Yes
pge_id	pgeId	varchar(19)	Yes	Yes
pgeparametername	pgeParameterName	varchar(50)	No	No

Table: PlPgeDetailTime

Description

Details of time associated with the scheduling of a PGE.

Column List

Name	Code	Type	P	M
orbitoffset	orbitOffset	int	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
processing_boundary	processingBoundary	varchar(40)	No	No
processing_period	processingPeriod	varchar(20)	No	No

Table: PlPgeMaster

Description

Main source of information about a PGE.

Column List

Name	Code	Type	P	M
instrument	instrument	varchar(20)	No	No
minnumoutputs	minNumOutputs	int	No	No
normal_staging_time	normalStagingTime	integer	No	No
orbit_number	orbitNumber	int	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
pge_name	pgeName	varchar(12)	No	Yes
pge_test_operational	pgeTestOperational	int	No	Yes
pge_version	pgeVersion	varchar(5)	No	Yes
pgecollectionid	pgeCollectionId	varchar(20)	No	No
pgescheduletype	pgeScheduleType	char(1)	No	No
platform	platform	varchar(25)	No	No
profiledescription	profileDescription	varchar(255)	No	No
profileid	profileId	int	No	No
reprocess_staing_time	reprocessStagingTime	integer	No	No
sswid	sswId	varchar(17)	No	No
subscriptionflag	subscriptionFlag	char(1)	No	No

Table: PlPgeOrbitModel

Description

A table used to calculate the estimated times for an orbit.

Column List

Name	Code	Type	P	M
orbit_number	orbitNumber	int	Yes	Yes
orbitlength	orbitLength	int	No	No
orbitstart	orbitStart	datetime	No	No
platform	platform	varchar(25)	Yes	Yes

Table: PlPgePerformance

Description

Performance statistics for a PGE.

Column List

Name	Code	Type	P	M
block_in_operation	blockInputOperation	int	No	No
block_out_operation	blockOutputOperation	int	No	No
cpu_time	cpuTime	int	No	No
dprelapsedtime	dprElapsedTime	int	No	No
faults	faults	int	No	No
max_memory	maxMemory	float	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
pgeelapsedtime	pgeElapsedTime	int	No	No
run_block_in_operation	runBlockInOperation	int	No	No
run_block_out_operation	runBlockOutOperation	int	No	No
run_cpu_time	runCpuTime	int	No	No
run_max_memory	runMaxMemory	float	No	No
run_shared_memory	runSharedMemory	float	No	No
run_swaps	runSwaps	int	No	No
rundprelapsed	runDprElapsed	int	No	No
runfaults	runFaults	int	No	No
runpgeelapsed	runPgeElapsed	int	No	No
shared_memory	sharedMemory	float	No	No
swaps	swaps	int	No	No

Table: PlPgePriority

Description

Relates a particular PGE to a particular priority. Used by production planning to determine the priority of jobs that use that PGE.

Column List

Name	Code	Type	P	M
pgeid	pgeId	varchar(19)	Yes	Yes
priority	priority	int	No	Yes
strategyid	strategyId	varchar(20)	Yes	Yes

Table: PlPlanMetadata

Description

Represents the metadata that will be used to describe the plan within the document data server.

Column List

Name	Code	Type	P	M
planid	planId	varchar(20)	Yes	Yes
planmetadatadaac	planMetadataDaac	char(4)	No	No
planmetadatadatasets	planMetadataDatasets	varchar(20)	No	No
planmetadatadescription	planMetadataDescription	char(18)	No	No
planmetadataend	planMetadataEnd	datetime	No	No
planmetadataforecast	planMetadataForecast	char(18)	No	No
planmetadatastart	planMetadataStart	datetime	No	No

Table: PlPlans

Description

Describes a production plan.

Column List

Name	Code	Type	P	M
active_status	planActiveStatus	char(1)	No	Yes
description	planDescription	varchar(255)	No	No
end_time	planCreation	datetime	No	No
plan_name	planName	varchar(20)	No	Yes
planend	planEnd	datetime	No	No
planid	planId	varchar(20)	Yes	Yes
planstart	planStart	datetime	No	No
start_time	planActivation	datetime	No	No
strategy_id	strategyId	varchar(20)	No	Yes

Table: PlPrCollection

Description

Describes a group of related production requests awaiting processing that have been designated as a collection.

Column List

Name	Code	Type	P	M
collection_start	collectionStart	datetime	No	No
collection_stop	collectionStop	datetime	No	No
origination_time	originationTime	datetime	No	No
pr_collection_id	prCollectionId	varchar(10)	Yes	Yes
priority	priority	int	No	No
requester_id	requesterId	varchar(25)	No	No

Table: PlPrCollectionExplosion

Description

Contains the list of production requests belonging to a collection.

Column List

Name	Code	Type	P	M
collection_start	collectionStart	datetime	No	No
collection_stop	collectionStop	datetime	No	No
origination_time	originationTime	datetime	No	No
pr_collection_id	prCollectionId	varchar(10)	No	Yes
pr_explosion_id	prExplosionId	varchar(10)	Yes	Yes
priority	priority	int	No	No
requester_id	requesterId	varchar(25)	No	No

Table: PlPrMetadataValue

Description

Contains expected values of parameters associated with a production request.

Column List

Name	Code	Type	P	M
datatypeid	dataTypeId	varchar(20)	Yes	Yes
ioflag	ioFlag	int	Yes	Yes
metadatavalue	metadataValue	varchar(40)	No	Yes
paraname	paraName	varchar(20)	Yes	Yes
productionrequestid	productionRequestId	varchar(20)	Yes	Yes

Table: PlProductionRequest

Description

Information associated with a production request.

Column List

Name	Code	Type	P	M
collection_start	collectionStart	datetime	No	No
collection_stop	collectionStop	datetime	No	No
description	description	varchar(255)	No	No
num_dpr_to_keep	numDprToKeep	int	No	No
num_dpr_to_skip	numDprToSkip	int	No	No
originationdate	originationDate	datetime	No	No
output_data_type	outputDataType	varchar(20)	No	No
pge_id	pgeIdentifier	varchar(19)	No	Yes
pr_collection_id	prCollectionId	varchar(10)	No	No
PR_type	prType	int	No	No
priority	productionRequestPriority	int	No	No
request_id	productionRequestId	varchar(20)	Yes	Yes
requester_id	requesterId	varchar(25)	No	No
skip_first_flag	skipFirstFlag	bit	No	Yes
target_date	targetDate	datetime	No	No
target_delta	targetDelta	int	No	No
user_type	userType	varchar(20)	No	No

Table: PlProductionStrategy

Description

Describes a production strategy used by a DAAC to determine priorities for data processing requests.

Column List

Name	Code	Type	P	M
aging_delta	agingDelta	int	No	No
defaultpgepriority	defaultPGEPriority	int	No	No
defaultprpriority	defaultPRPriority	int	No	No
defaultuserpriority	defaultUserPriority	int	No	No
inter_DAAC_delta	interDaacDelta	int	No	No
pge_weight	pgeWeight	float	No	No
PR_type_weight	prTypeWeight	float	No	No
strategy_id	strategyId	varchar(20)	Yes	Yes
user_selected_priority_weight	userSelectedPriorityWeight	float	No	No
user_weight	userWeight	float	No	No

Table: PlPrParameter

Description

Description of a parameter associated with a production request.

Column List

Name	Code	Type	P	M
parameter_value	parameterValue	varchar(30)	No	No
pgeid	pgeId	varchar(19)	Yes	Yes
pgeparameterlogicalid	pgeParameterLogicalId	int	Yes	Yes
PlP_pge_id	PlP_pgeId	varchar(19)	No	Yes
request_id	productionRequestId	varchar(20)	Yes	Yes

Table: PlPRPriority

Description

A list of PR-based priorities associated with a production strategy.

Column List

Name	Code	Type	P	M
priority	priority	int	No	Yes
prtype	prType	int	Yes	Yes
strategyid	strategyId	varchar(20)	Yes	Yes

Table: PlPrVsPlan

Description

A join table relating production requests to production plans.

Column List

Name	Code	Type	P	M
planid	planId	varchar(20)	No	Yes
priority	priority	int	No	No
prvsplanid	prVsPlanId	numeric(9,0)	Yes	Yes
request_id	productionRequestId	varchar(20)	No	Yes

Table: PlReplanCriteria

Description

Information about criteria for replan notification.

Column List

Name	Code	Type	P	M
auto_plan	autoPlan	bit	No	Yes
check_plan	checkPlan	bit	No	Yes
checkplandas	checkPlanDAS	bit	No	Yes
checkplanod	checkPlanOD	bit	No	Yes
checkplanrs	checkPlanRS	bit	No	Yes
criteria_id	criteriaId	numeric(9,0)	Yes	Yes
DAS_diff_time_frame	dasDiffTimeFrame	int	No	No
default_to_thresh	defaultToThresh	bit	No	Yes
rsc_change_time_frame	rscChangeTimeFrame	int	No	No

Table: PlRescUseThresh

Description

Contains resource usage thresholds for OnDemand production.

Column List

Name	Code	Type	P	M
resc_use_thresh_id	rescUseThreshId	numeric(9,0)	Yes	Yes
resource_type	resourceType	int	No	Yes
threshold	threshold	float	No	Yes
user_type	userType	varchar(20)	No	Yes

Table: PlResource

Description

Base table for a physical resource.

Column List

Name	Code	Type	P	M
activitytypeid	activityTypeId	int	No	No
resource_id	resourceId	int	Yes	Yes
resource_name	resourceName	varchar(60)	No	Yes
resourcestate	resourceState	int	No	No
resourcetype	resourceType	varchar(15)	No	No

Table: PlResourceRequirement

Description

Describes the resource requirements of a PGE.

Column List

Name	Code	Type	P	M
computer	computer	varchar(60)	No	No
cpu	numOfCPUs	int	No	No
deleteflag	deleteFlag	int	No	No
diskspace	diskSpace	float	No	No
exetarfilediskspace	exeTarFileDiskSpace	float	No	No
exetarur	exeTarUR	varchar(255)	No	No
exeuntarfilediskspace	exeUntarFileDiskSpace	float	No	No
mcfname	mcfName	varchar(30)	No	No
pgeid	pgeId	varchar(19)	No	No
ramsize	ramSize	float	No	No
sswid	sswId	varchar(17)	Yes	Yes
string	string	varchar(60)	No	No
toplevelshellname	topLevelShellName	varchar(30)	No	No

Table: PlRoutineArrival

Description

Describes the most frequent method for predicting data arrivals within ECS.

Column List

Name	Code	Type	P	M
boundary	boundary	varchar(40)	No	No
data_type_id	dataTypeId	varchar(20)	Yes	Yes
delay	delay	int	No	No
period	period	varchar(20)	No	No

Table: PlRscComputer

Description

Describes the computers in the resource configuration for the production system.

Column List

Name	Code	Type	P	M
computerid	computerId	int	Yes	Yes
computername	computerName	varchar(60)	No	No
cpu	totalCpu	int	No	No
cpuallocation	cpuAllocation	int	No	No
max_disk_space	maxDiskSpace	int	No	No
operating_system	operatingSys	varchar(60)	No	No
ramallocation	ramAllocation	float	No	No
realcomputerid	realComputerId	int	No	No
stringid	stringId	int	No	No
total_ram	totalRam	int	No	No

Table: PlRscDiskPartition

Description

Describes the disk resources for data production.

Column List

Name	Code	Type	P	M
block_size	blockSize	int	No	No
computerid	computerId	int	No	No
device_id	partitionName	varchar(60)	No	No
diskpartitionid	diskPartitionId	int	Yes	Yes
diskusage	diskUsage	float	No	No
partition_size	partitionSize	float	No	No
sysallocation	sysAllocation	double precision	No	No
userallocation	userAllocation	double precision	No	No

Table: PlRscRealComputer

Description

Describes a real (as opposed to virtual) computer that is part of the resource configuration for production.

Column List

Name	Code	Type	P	M
realcomputerid	realComputerId	int	Yes	Yes
realcomputername	realComputerName	varchar(60)	No	No

Table: PlRscString

Description

Describes a string, i.e. a logical collection of resources allocated for processing the data collected by an instrument.

Column List

Name	Code	Type	P	M
autosysidkey	autosysIdKey	int	No	No
stringid	stringId	int	Yes	Yes
stringname	stringName	varchar(60)	No	No

Table: PlTile

Description

Describes a tile, or geographical area of data, that will be processed by a tile-scheduled PGE.

Column List

Name	Code	Type	P	M
cluster_id	clusterId	numeric(9,0)	No	Yes
hnum	hnum	int	No	Yes
tile_id	tileId	numeric(9,0)	Yes	Yes
vnum	vnum	int	No	Yes

Table: PlTileCoordinates

Description

Contains the list of coordinates associated with a tile.

Column List

Name	Code	Type	P	M
coordinates_id	coordinatesId	numeric(7,0)	Yes	Yes
latitude	latitude	float	No	Yes
longitude	longitude	float	No	Yes
tile_id	tileId	numeric(7,0)	No	No

Table: PlTileScheduled

Description

Used by tile-scheduled PGEs in generating DPRs and their input data.

Column List

Name	Code	Type	P	M
boundary	boundary	datetime	No	No
definition	definition	varchar(20)	No	No
delay_factor	delayFactor	datetime	No	No
L0_data_type	l0DataType	varchar(20)	No	No
num_clusters	numClusters	int	No	No
period	period	datetime	No	No
pge_id	pgeId	varchar(19)	Yes	Yes
prod_delta_time	prodDeltaTime	datetime	No	No

Table: PlUsedByCenter

Description

An explosion table for PlDataTypeMaster, containing a list of DAACs that employ the data type.

Column List

Name	Code	Type	P	M
DAAC_name	daacName	varchar(20)	Yes	Yes
data_type_id	dataTypeId	varchar(20)	Yes	Yes

Table: PlUserParameterConstraints

Description

An explosion table for PlPgeDetailParameters, describing the user constraints defined for the PGE's parameters.

Column List

Name	Code	Type	P	M
constraints	constraints	varchar(20)	No	No
pgeid	pgeId	varchar(19)	No	No
pgeparameterlogicalid	pgeParameterLogicalId	int	No	Yes
PlP_pge_id	PlP_pgeId	varchar(19)	No	Yes
userparmconstraintsid	userParmConstraintsId	numeric(9,0)	Yes	Yes

Table: PlUserPriority

Description

Describes a user-based priority for a production strategy.

Column List

Name	Code	Type	P	M
priority	priority	int	No	Yes
strategyid	strategyId	varchar(20)	Yes	Yes
usertype	userType	varchar(20)	Yes	Yes

Table: RpActivityType

Description

Describes an activity type associated with a resource plan.

Column List

Name	Code	Type	P	M
activityname	activityName	varchar(20)	No	No
activitytypeid	activityTypeId	int	Yes	Yes

Table: RpReservationInterval

Description

Describes an interval used in resource planning.

Column List

Name	Code	Type	P	M
flag	flag	int	No	No
reservationid	reservationId	int	Yes	Yes
starttime	startTime	datetime	Yes	Yes
stoptime	stopTime	datetime	No	No

Table: RpResourceReservation

Description

Describes a resource reservation used in resource planning.

Column List

Name	Code	Type	P	M
activitytypeid	activityTypeId	int	No	No
actualend	actualEnd	datetime	No	No
actualstart	actualStart	datetime	No	No
description	description	varchar(30)	No	No
editeddate	editedDate	datetime	No	No
endtime	endTime	datetime	No	No
everyn	everyN	int	No	No
exclusive	exclusive	int	No	No
frequency	frequency	varchar(20)	No	No
name	name	varchar(60)	No	No
originator	originator	varchar(30)	No	No
priority	priority	int	No	No
reservationid	reservationId	int	Yes	Yes
sponsor	sponsor	varchar(30)	No	No
starttime	startTime	datetime	No	No
state	state	varchar(15)	No	No

Table: RpRscPlanComments

Description

Comments for a resource plan.

Column List

Name	Code	Type	P	M
comment	comment	varchar(255)	No	No
commentid	commentId	int	Yes	Yes
commenttype	commentType	varchar(20)	Yes	Yes
ordernum	orderNum	int	Yes	Yes

Table: RpRsvRscExplosion

Description

A join table relating resource reservations to actual resources.

Column List

Name	Code	Type	P	M
reservationid	reservationId	int	Yes	Yes
resourceid	resourceId	int	Yes	Yes

4.1.3 Columns

Brief definitions of each of the columns present in the database tables defined above are contained herein.

Column: absoluteDeletionTime

Description

TBS

Column: accepted

Description

Indicates whether the data granule, when available, has passed metadata checks.

Column: action

Description

TBS

Column: planActiveStatus

Description

Indicates whether the plan is currently active.

Column: activityName

Description

Name for the activity.

Column: activityTypeId

Description

Type of activity associated with the resource.

Column: actualStart

Description

The date and time when the DPR is actually processed.

Column: actualEnd

Description

TBS

Column: agingDelta

Description

The amount to be added to a priority for an older DPR that still has not completed.

Column: alarmText

Description

TBS

Column: alarmTime

Description

TBS

Column: archiveCenter

Description

DAAC where the data type is archived.

Valid Values: See 311-CD-107-001

Column: attempts

Description

TBS

Column: autoPlan

Description

Indicates whether the criteria should initiate an automatic replan.

Column: autosysId

Description

The instance of Autosys designated by the plan to process this DPR.

Column: autosysIdKey

Description

A numerical identifier for the mapping.

Column: availTimesId

Description

A reference to the available times table.

Column: availability

Description

Flag to indicate availability of the data.

Column: availabilityActual

Description

The date and time the data was actually made available to the subscription manager.

Column: availabilityPredicted

Description

The predicted date and time when data will be available to the subscription manager.

Column: availTimesId

Description

TBS

Column: baselineTime

Description

The availability date and time of the data granule according to the baselined active plan, agreed to by the DAACs requiring the granule.

Column: blockInputOperation

Description

Number of input blocks used by the PGE during SSI&T.

Column: blockSize

Description

Block size for the partition.

Column: boundary

Description

A boundary time from which predicted times can be derived.

Column: boundary

Description

A time from which all periods are referenced. For tile clusters it is the time of the first orbit of a cluster.

Column: catalogueCategory

Description

Indicates whether the data type is generated from a PGE, an intermediate file, an input file received from another site, etc.

Column: checkPlan

Description

Indicates whether this criteria should be checked (and if met, the operator notified that a replan is necessary).

Column: checkPlanDAS

Description

Indicates whether criteria should be checked for Data Availability Schedules.

Column: checkPlanOD

Description

Indicates whether criteria should be checked for On Demand production.

Column: checkPlanRS

Description

TBS

Column: clusterId

Description

A reference to a cluster associated with the tile.

Column: collectionStart

Description

Start time of the collection.

Column: collectionStart

Description

The start time, in terms of data collection, for the production request.

Column: collectionStop

Description

The stop time, in terms of data collection, for the production request.

Column: comment

Description

TBS

Column: commentId

Description

TBS

Column: commentType

Description

TBS

Column: completionDate

Description

TBS

Column: completionState

Description

A status indicator describing the current status of the DPR.

Column: computer

Description

A computer within the string required by the PGE.

Column: computerId

Description

The computer where the memory has been allocated.

Column: computerName

Description

Name for the computer.

Column: constraints

Description

A constraint for the PGE parameter.

Column: coordinatesId

Description

TBS

Column: pgeCpu

Description

TBS

Column: cpuAllocation

Description

The number of CPUs allocated.

Column: cpuRamAllocationCpuCount

Description

TBS

Column: cpuRamAllocationJobId

Description

Identifier uniquely defining a particular memory allocation within a computer.

Column: cpuRamAllocationRamSize

Description

TBS

Column: cpuTime

Description

Amount of CPU time used during SSI&T.

Column: criteriaId

Description

A reference to the criteria identifier from the PlReplanCriteria table.

Column: daacName

Description

Name of the DAAC.

Valid Values: See 311-CD-107-001

Column: dasDiffTimeFrame

Description

TBS

Column: dataStartTime

Description

The start time for the data that is input to the DPR.

Column: dataStopTime

Description

The stop time for the data that is input to the DPR.

Column: dataRate

Description

TBS

Column: dataReadyTime

Description

TBS

Column: dataType

Description

Data type of the alternate input.

Column: dataTypeId

Description

Identifier for the data type.

Column: dataTypeRequirement

Description

The data type requirement of the PGE: a select statement to be applied to the data granule table to obtain the inputs for a given DPR.

Column: defaultMode

Description

TBS

Column: DEFAULTORDER

Description

Default priority of an alternate input data type.

Column: DEFAULTTIMER

Description

The default length of time that the Subscription Manager will wait for an alternate input data type to arrive.

Column: defaultToThresh

Description

Indicates whether replan thresholds should default to resource usage thresholds.

Column: defaultPGEPriority

Description

Default value for the PGE-based priority.

Column: defaultPRPriority

Description

Default value for the PR-based priority.

Column: defaultUserPriority

Description

Default value for the user-based priority.

Column: definition

Description

A definition used to create the tile.

Column: delay

Description

The average delay between data collection and the arrival in ECS of the corresponding data granule.

Column: delayFactor

Description

Amount of time used in calculating cluster availability.

Column: deleteFlag

Description

TBS

Column: delta

Description

The delta associated with the data type, i.e. the maximum delay from when a data granule was predicted to arrive in the Data Availability Schedule (DAS) before a limited automatic replan should be triggered.

Column: dependencySswId

Column: dependentDprId

Description

Identifier for a dependent DPR.

Column: description

Description

A textual description of the production request.

Column: planDescription

Description

Description of the purpose of the plan.

Column: description

Description

A description of the user parameter.

Column: dataTypeDescription

Description

A textual description of the data type.

Column: partitionName

Description

Name for the partition.

Column: diskAllocationActual

Description

The actual amount of disk space being consumed by the allocated file.

Column: diskAllocationId

Description

Internal identifier to uniquely track disk allocations.

Column: diskAllocationPath

Description

The location of the disk allocation as a file system path name.

Column: diskAllocationSize

Description

The amount of disk space specified for the original allocation request.

Column: diskAllocationType

Description

The type of allocation: system files or user (science software) files.

Column: diskAllocationUser

Description

The job associated with the file defined by the path column.

Column: diskPartitionId

Description

Identifier for the disk partition associated with the allocation.

Column: diskSpace

Description

The amount of disk space required for the PGE.

Column: diskUsage

Description

TBS

Column: dprCollectionId

Description

The collection identifier, if this DPR is part of a collection.

Column: dprElapsedTime

Description

Amount of elapsed time for the DPR.

Column: dprId

Description

The identifier of the DPR currently associated with the data granule.

Column: dataServUrString

Description

Universal Reference to the Data Server providing services for the data type.

Column: dynamicFlag

Description

Tells whether the data type is static or dynamic.

Column: dynamicType

Description

TBS

Column: ecsUnit

Description

TBS

Column: editedDate

Column: planCreation

Description

When the plan was created.

Column: endTime

Description

TBS

Column: endTimeOffset

Description

TBS

Column: errorActionId

Column: esdtParmName

Description

Name of the ESDT.

Column: esdtParmVal

Description

The ESDT parameter value.

Column: estimatedGranuleVolume

Description

TBS

Column: everyN

Column: exclusive

Description

TBS

Column: execLocation

Description

The disk location of the executable.

Column: execName

Description

The name of the executable.

Column: execPermission

Description

The file system permissions setting for the executable file.

Column: execShell

Description

For executables which are shell scripts, the type of shell for which the script was written.

Column: execLayer

Description

TBS

Column: execMachine

Description

Machine designated for execution of the PGE.

Column: execMachine

Description

The machine on which this particular execution is performed.

Column: execName

Description

Name of the executable file.

Column: exeTarFileDiskSpace

Description

TBS

Column: exeTarUR

Description

TBS

Column: exeUntarFileDiskSpace

Description

TBS

Column: exitCode

Column: exitCode

Column: exitMessage

Column: exitOperation

Column: faults

Description

Number of page faults during SSI&T.

Column: fileName

Description

Name of the file in the file system.

Column: fileNumber

Description

number of file (if part of a multi-file granule)

Column: fileSize

Description

size of file

Column: fileType

Description

A file type associated with this data type.

Column: fileTypeId

Description

TBS

Column: fileTypeName

Description

Name of a file type associated with the data type.

Column: flag

Description

TBS

Column: frequency

Description

TBS

Column: granuleId

Description

Identifier for a data granule.

Column: granuleSize

Description

The storage requirements of the data granule (units TBS).

Column: groundEventDescription

Description

Description of a ground event.

Column: groundEventDuration

Description

The duration of the ground event.

Column: groundEventId

Description

Identifier for a ground event.

Column: groundEventName

Description

Name for a ground event.

Column: groundEventPriority

Description

The priority of the ground event.

Column: hdfFlag

Description

TBS

Column: hnum

Description

TBS

Column: hold

Description

Flag indicating that Autosys has this DPR on hold.

Column: inputDataId

Description

An artificial row identifier (Sybase identity).

Column: inputDataId

Description

ID of the input data upon whose arrival the PGE is scheduled.

Column: inputDataStagingPercent

Description

The percentage of data received that has arrived locally from the data server.

Column: inputType

Description

TBS

Column: instancesPerPgerun

Description

TBS

Column: instrument

Description

The instrument name associated with the data type.

Column: instrument

Description

The instrument for which the PGE is appropriate.

Column: instrumentName

Description

Instrument collecting the data.

Column: interDaacDelta

Description

TBS

Column: interimLastPgeToUse

Description

TBS

Column: interimLongDuration

Description

TBS

Column: interimShortDuration

Description

TBS

Column: ioFlag

Description

Indicates whether the granule is input or output with respect to the DPR.

Column: ISCURRENTLYACTIVE

Description

Flag to indicate whether the plan is currently active.

Column: jobId

Description

Identifier for the current job.

Column: l0DataType

Description

The L0 data type upon which the tiles are based. Used to calculate the time to move the L0 data from EDOS to the dataserver.

Column: lastActualArrival

Description

The last actual arrival time of the data type to the data server.

Column: LASTORBIT

Description

The number of the orbit that contains the last data granule for this cluster.

Column: lastPredictedArrival

Description

The last predicted arrival time of a data type.

Column: lateMessagesSent

Description

TBS

Column: latitude

Description

Coordinate for the latitude.

Column: linkId

Description

TBS

Column: logicalId

Description

Identifier to distinguish among multiple instances of a PGE/dataType combination.

Column: longitude

Description

Coordinate for the longitude.

Column: machineId

Description

Identifier of the machine housing the data granule.

Column: maxDiskSpace

Description

Maximum disk space (storage) on this computer.

Column: maxMemory

Description

Maximum amount of memory required by the PGE during SSI&T.

Column: maxSize

Description

Maximum size of the output data granule.

Column: maxFilesOfAnyType

Description

The maximum number of files corresponding to this data type, regardless of file type.

Column: maxFileTypes

Description

The maximum number of file types associated with this data type.

Column: mcfName

Description

Name of the metadata control file associated with the PGE.

Column: messageId

Description

TBS

Column: type

Description

The type (int, string, etc.) of the metadata parameter to be compared.

Column: value

Description

The value for the metadata parameter to be compared with the actual metadata value.

Column: metadataLogId

Description

TBS

Column: metadataValue

Description

The metadata value.

Column: minSize

Description

Minimum size of the output data granule.

Column: minmaxDataType

Column: minNumOutputs

Description

The minimum number of data granules the PGE should generate.

Column: modeName

Column: modeStartTime

Column: modeStopTime

Column: movingAvgTime

Description

The moving average of the source to data server interval for a given data type.

Column: name

Description

TBS

Column: nominalSize

Description

The nominal size of the data type.

Column: normalStagingTime

Description

The time normally required to stage (make local) the input data for the PGE.

Column: numClusters

Description

Number of clusters for this PGE.

Column: numDprToKeep

Description

For intermittently activated PGEs, the number of DPRs to be kept for this PR.

Column: numDprToSkip

Description

For intermittently activated DPRs, the number of DPRs to skip (after the number to keep have been created).

Column: numEstimate

Description

The number of samples used to compute the moving average of the source to data server interval.

Column: numOfExecution

Description

TBS

Column: NUMTILES

Description

Number of tiles in this cluster.

Column: numOfCPUs

Description

The number of CPUs required to run a PGE.

Column: numberOfUsage

Description

How many DPRs use this file.

Column: numNeeded

Description

The number of input granules required by the DPR.

Column: onDemandReplanId

Description

A sequence number (Sybase identity).

Column: operatingSys

Description

The operating system name and version running on this computer.

Column: operatingSystem

Description

TBS

Column: operator

Description

The operation (=, >, <) for the comparison of the metadata field.

Column: orbitNumber

Description

The number of the orbit.

Column: orbitLength

Description

The length of the orbit.

Column: orbitOffset

Description

TBS

Column: orbitStart

Description

The starting time of the orbit.

Column: theOrder

Description

The relative priority of an alternate input data granule.

Column: orderId

Description

The order identifier of the On Demand Production Request as received by PDPS from the Data Server.

Column: orderNum

Description

TBS

Column: orderRequestId

Description

The order request identifier of the On Demand Production Request as received by PDPS from the Data Server.

Column: originationTime

Description

The time the collection request was entered.

Column: originationDate

Description

Date and time when the production request was entered.

Column: originator

Column: outputDataType

Description

Identifies the type of product generated by the production request.

Column: parameterValue

Description

Value for the parameter.

Column: paramName

Description

The parameter name.

Column: paramType

Description

The parameter type.

Column: paraName

Description

Name of the parameter.

Column: parmId

Column: parmKeyword

Column: parmMax

Column: parmMin

Column: parmName

Column: parmType

Column: partitionSize

Description

Total size of the partition.

Column: path

Description

Location of the file as a path name.

Data type of the output granule.

Column: pcfLocation

Description

Pathname of the directory containing the PCF.

Column: pcfName

Description

Filename of the PCF.

Column: pcfPermission

Description

File permissions that need to be placed on the PCF.

Column: pcfType

Description

Possible values: TBS.

Column: pcfFileType

Description

TBS

Column: period

Description

The time period used in predicting arrivals (hour, day, set of orbits, etc.)

Column: pgeCollectionId

Description

Identifier for the collection.

Column: pgeCommands

Description

Startup condition information for the activation shell.

Column: pgeEnvironment

Description

Environment variables that need to be set prior to running the PGE.

Column: pgeId

Description

The PGE associated with this input.

Column: pgeName

Description

Name for the PGE.

Column: pgeParameterLogicalId

Description

ID of the parameter for this PGE.

Column: pgeShell

Description

The processing shell which activates the science software's outer shell.

Column: pgeState

Description

TBS

Column: pgeTestOperational

Description

Indicates if the PGE's status is test or operational.

Column: pgeVersion

Description

Version number of the PGE.

Column: pgeWeight

Description

The weight given to the priority, based on the PGE.

Column: pgeBlockInputOperations

Description

TBS

Column: pgeBlockOutputOperations

Description

TBS

Column: pgeCollectionId

Description

If the PGE belongs to a collection, the identifier for that collection.

Column: pgeCpuTime

Column: pgeElapsedTime

Description

Amount of elapsed time for the PGE during SSI&T.

Column: pgeId

Description

Identifier of the PGE for which the data granule is output.

Column: pgeMaxMemoryUse

Description

TBS

Column: pgePageFaults

Description

TBS

Column: pgeParameterDefault

Description

Default value for the user parameter.

Column: pgeParameterLogicalId

Description

Parameter number for the PGE.

Column: pgeParameterLogicalId

Description

Logical identifier for a PGE parameter.

Column: pgeParameterName

Description

Name of the user parameter.

Column: pgeScheduleType

Description

Indicates whether the PGE will be scheduled by time, by data arrival, or by some other method.

Column: pgeSharedMemoryUse

Description

TBS

Column: pgeSswId

Description

TBS

Column: pgeSwaps

Description

TBS

Column: planActivation

Description

Column: PLANENDTIME

Description

End time defining the right bracket of the plan time frame.

Column: planName

Description

Name for the plan.

Column: PLANSTARTTIME

Description

Start time defining the left bracket of the plan time frame.

Column: planEnd

Description

Ending point of the time frame for the plan.

Column: planId

Description

Identifier for the plan.

Column: planMetadataDaac

Description

The site at which the plan was produced.

Column: planMetadataDatasets

Description

Describes the data sets which are planned for production within the plan.

Column: planMetadataDescription

Description

Captures the operator's description of the purpose of the plan.

Column: planMetadataEnd

Description

The end date of the plan.

Column: planMetadataForecast

Description

The number of days in the time period covered by the plan.

Column: planMetadataStart

Description

The start date of the plan.

Column: planStart

Description

Starting point of the time frame for the plan.

Column: platform

Description

The satellite providing the data.

Column: platformName

Description

The platform associated with the instrument.

Column: PlD_pgeId

Description

TBS

Column: PlO_logicalId

Description

Identifier to distinguish among multiple instances of a PGE/dataType combination.

Column: PlO_pgeId

Description

Identifier of the PGE.

Column: PlP_pgeId

Description

Identifier of the PGE.

Column: prCollectionId

Description

Identifier for the collection, if this production request if part of a collection.

Column: prExplosionId

Description

Identifier for the current row.

Column: productionRequestId

Column: prType

Description

Indicates the production type: routine, on-demand, or reprocessing.

Column: prTypeWeight

Description

The weight given to the priority, based on the production request type.

Column: PREDICTAVAILABILITY

Description

The predicted availability time of all data granules in this cluster.

Column: predictedAvailability

Description

The date and time that this data is predicted to be available at the local DAAC.

Column: predictionMethod

Description

The method by which data availability prediction occurs (e.g. routine arrival, FOS-based).

Column: predictedStagingTime

Description

An estimate of the length of time that it will take to stage this data granule prior to data processing.

Column: predictedStart

Description

The predicted date and time when this DPR will be run by Data Processing.

Column: primaryDataSource

Description

Identifies the source of the data.

Column: primaryType

Description

The data type of the primary alternate input associated with this data granule.

Column: priority

Description

Priority of the reservation.

The user-based priority.

Column: priority

Description

The numerical priority assigned to this DPR (usually inherited from the PR).

Column: productionRequestPriority

Description

The priority requested by the user when submitting the production request.

Column: processingBoundary

Description

Identifies the time needed for data for extrapolation purposes.

Column: processingCenter

Description

The name of the DAAC that produces this data type.

Column: processingPeriod

Description

Identifies the acquisition period for the input data.

Column: processingType

Description

TBS

Column: processingLevel

Description

TBS

Column: prodDeltaTime

Description

Estimated time to produce all intermediate products used to produce tiles.

Column: productOrigin

Description

Where the data availability information came from, .e.g. EDOS or a DAAC name.

When the plan was or will be activated.

Column: productionRequestId

Description

Identifier for the production request. (This should match the entry in the table PlProductionRequest.)

Column: profileDescription

Description

The pathname of the PGE profile.

Column: profileId

Description

An identifier for the profile information associated with the PGE.

Column: provider

Description

The DAAC that is maintaining the data.

Column: prType

Description

The production request type: standard, on-demand, or reprocessing.

Column: prVsPlanId

Description

An identity for this table.

Column: qaSubscription

Description

Flag indicating whether a subscription has been set up for the QA of this data type.

Column: qaThreshold

Description

The quality threshold to be applied to the data granules as a test of their suitability as input to the PGE.

Column: ramAllocation

Description

Available RAM.

Column: ramSize

Description

Amount of RAM required for the PGE.

Column: realComputerId

Description

The ID of the associated real computer. (An entry in this table could be a virtual computer.)

Column: realComputerName

Description

Name of the real computer.

Column: replanValue

Description

The threshold which, if exceeded, would indicate that a replan should be done to incorporate the On Demand Production Request.

Column: reprocessStagingTime

Description

The staging time normally required for reprocessing.

Column: requesterId

Description

Identifier of the requester entering the collection.

Column: rescUseThreshId

Description

Identity for this table.

Column: reservationId

Description

Identifier for the resource reservation.

Column: resourceId

Description

Numeric identifier for the resource.

Column: resourceName

Description

Name for the resource.

Column: resourceState

Description

Current state of the resource.

Column: resourceString

Description

Resource identifier associated with this instance of Autosys.

Column: resourceType

Description

Type of resource.

Column: rscChangeTimeFrame

Description

TBS

Column: runBlockInOperation

Description

Number of input blocks required during production.

Column: runBlockOutOperation

Description

Number of output blocks required during production.

Column: runCpuTime

Description

Amount of CPU time required by the PGE during production.

Column: runMaxMemory

Description

Maximum amount of memory required by the PGE during production.

Column: runSharedMemory

Description

Amount of shared memory used during production.

Column: runSwaps

Description

Number of swaps during production.

Column: runDprElapsed

Description

Elapsed time required by the DPR during production.

Column: runFaults

Description

Number of page faults during production.

Column: runPgeElapsed

Description

Elapsed time required by the PGE during production.

Column: runState

Description

TBS

Column: RUNTIMEID

Description

TBS

Column: scienceGroup

Description

TBS

Column: sequenceNumber

Description

TBS

Column: sharedMemory

Description

Amount of shared memory used during SSI&T.

Column: skipFirstFlag

Description

TBS

Column: spatialFlag

Description

Flag indicating spatial data, i.e. data that requires spatial coordinates in addition to start and stop times.

Column: sponsor

Description

TBS

Column: sswId

Description

The science software identifier associated with this PGE.

Column: stageState

Description

TBS

Column: startTime

Description

The start date and time for the data granule.

Description

Start time for the interval.

Column: startTimeOffset

Description

TBS

Column: state

Description

TBS

Column: status

Column: stdDeviation

Description

Specifies the acceptable standard deviation from the historical average of source to data server time.

Column: stopTime

Description

Stop time of the data (should match that in the data granule table).

Column: strategyId

Description

Identifier for the production strategy.

Column: string

Description

The string required by the PGE.

Column: stringId

Description

Resource identifier for the string.

Column: stringName

Description

Name for the string.

Column: subscriptionFlag

Description

Flag indicating whether Planning currently holds a subscription on this data type.

Column: supplierName

Description

Name of the supplier of the data.

Column: swaps

Description

Number of swaps during SSI&T.

Column: sysAllocation

Description

Size within the partition allocated for system usage.

Column: systemCpuTime

Description

TBS

Column: targetDate

Description

The desired time for completion of the production request.

Column: targetDelta

Description

TBS

Column: temporalFlag

Description

Indicates whether this alternate data granule can be replaced with the most recent data granule of the same type, if one cannot be found in the current production request time frame.

Column: threshold

Description

The resource usage threshold for the particular resource and user types.

Column: tileId

Description

For data granules that are geographical tiles rather than time-continuous data, the identifier of the tile within the start and stop time of the orbit set.

Column: timeframeDataType

Column: timerExp

Description

Indicates whether the timer has expired.

Column: timerStart

Description

Indicates whether the timer has started.

Column: timeWait

Description

The length of time the Subscription Manager will wait for an alternate input data granule to arrive.

Column: topLevelShellName

Description

Description of the shell in which the PGE is executed.

Column: totalCpu

Description

Total number of CPUs for this computer.

Column: totalRam

Description

Total RAM for this computer.

Column: type

Description

Indicates the type of alternate input: required, primary, or backup.

Column: unitStagingTime

Description

TBS

Column: universalReference

Description

The Universal Reference associated with this data granule.

Column: userAllocation

Description

Size within the partition allocated for production usage.

Column: userSelectedPriorityWeight

Description

The weight assigned to the priority by the user.

Column: userParmConstraintsId

Description

An identity for this table.

Column: userType

Description

The type of user submitting the production request.

Column: userWeight

Description

The weight given to the priority, based on the user.

Column: version

Description

TBS

Column: vnum

Description

TBS

Column: waitForFlag

Description

A flag indicating whether Planning should wait for this alternate input after the timer has expired. If FALSE, then Planning will permit the PGE to run without this input.

Column: whereClause

Description

TBS (How does this differ from the dataTypeRequirement???)

Column: yield

Description

The number of data granules produced from the DPR.

4.1.4 Column Domains

Domains specify the ranges of values allowed for a given table column. Sybase supports the definition of specific domains to further limit the format of data for a given column. Sybase domains are, in effect, user-defined data types. The are no domains defined in the PDPS database.

4.1.5 Rules

Sybase supports the definitions of rules. Rules provide a means for enforcing domain constraints on a given column. All rules defined in Sybase for the PDPS database are described herein.

4.1.6 Defaults

Defaults are used to supply a value for a column when one is not defined at insert time. There are no defaults defined in Sybase in the PDPS database.

4.1.7 Views

Sybase allows the definition of views as a means of limiting an application or users access to data in a table or tables. Views create a logical table from columns found in one or more tables. All views defined in the PDPS databases are described herein.

View List

Name	Code	Upd	Gen
PlDataGranule	PlDataGranule	Yes	Yes

View: PlDataGranule

Code

```
/* ============================================================ */
/*   View: PlDataGranule                                        */
/* ============================================================ */
create view PlDataGranule as
select PlDataGranuleShort.*, PlDataTypeMaster.dynamicFlag,
PlDataTypeMaster.interimLastPgeToUse, PlDataTypeMaster.interimLongDuration,
PlDataTypeMaster.interimShortDuration, PlDataTypeMaster.processingLevel,
PlDataTypeMaster.maxFileTypes, PlDataTypeMaster.maxFilesOfAnyType,
PlDataTypeMaster.hdfFlag
from PlDataGranuleShort, PlDataTypeMaster
where PlDataGranuleShort.dataTypeId = PlDataTypeMaster.dataTypeId
```

4.1.8 Integrity Constraints

Sybase version 11.0.1 allows the enforcement of referential integrity via the use of declarative integrity constraints. Integrity constraints allow the SQL server to enforce primary and foreign key integrity checks without automatically without requiring programming. Sybase 11 is only ANSI-92 compliant, however, therefore its constraints support “restrict-only” operations. This means that a row can not be deleted or updated if their are rows in other tables having a foreign key dependency on that row. Cascade delete and update operations can not be performed if a declarative constraint has been used. All declarative integrity constraints defined in the PDPS database are described herein.

Dependencies on Table: DpPrAutosysMapList

Reference by List

Referenced by	Primary Key	Foreign Key
PlRscString	autosysIdKey	autosysIdKey

Dependencies on Table: DpPrCreationQueue

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrDprDependList	dprId	dprId
DpPrGranuleLocation	dprId	dprId
DpPrPcf	dprId	dprId

Dependencies on Table: DpPrExecutable

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrPge	sswId execName execMachine	sswId execName execMachine

Dependencies on Table: DpPrFile

Dependencies on Table: DpPrInputData

Reference by List

Referenced by	Primary Key	Foreign Key
PlPgeDetailData	inputDataId	inputDataId

Dependencies on Table: DpPrPge

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrDprDependList	dprId	dprId
DpPrGranuleLocation	dprId	dprId
DpPrPcf	dprId	dprId

Dependencies on Table: DpPrPgeExitMessage

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrPgeExitDependency	pgeSswId exitCode	pgeSswId exitCode

Dependencies on Table: DpPrPredStagingProfile

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrInputData	processingType pgeId	processingType pgeId

Dependencies on Table: PlCluster

Reference by List

Referenced by	Primary Key	Foreign Key
PlTile	clusterId	clusterId

Dependencies on Table: PlDasDelta

Reference by List

Referenced by	Primary Key	Foreign Key
PlAlternate	dataTypeId	dataTypeId
PlDataGranule	dataTypeId	dataTypeId
PlDataGranuleShort	dataTypeId	dataTypeId

Dependencies on Table: PlDataAvailTimes

Reference by List

Referenced by	Primary Key	Foreign Key
PlDatRecord	availTimesId	availTimesId
PlModeRecord	availTimesId	availTimesId

Dependencies on Table: PlDataGranuleShort

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrGranuleLocation	granuleId	granuleId
PlDataGranule	granuleId	granuleId
PlDataHistory	granuleId	granuleId
PlDprData	granuleId	granuleId
PlEsdtParmValues	granuleId	granuleId
PlTimer	granuleId	granuleId

Dependencies on Table: PlDataHistory

Reference by List

Referenced by	Primary Key	Foreign Key
PlAlternate	dataTypeId	dataTypeId
PlDataGranule	dataTypeId	dataTypeId
PlDataGranuleShort	dataTypeId	dataTypeId

Dependencies on Table: PlDataProcessingRequest

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrDprDependList	dprId	dprId
DpPrGranuleLocation	dprId	dprId
DpPrPcf	dprId	dprId
PlDprCompletion	dprId	dprId
PlDprData	dprId	dprId
PlTimer	dprId	dprId

Dependencies on Table: PlDataSourceMaster

Reference by List

Referenced by	Primary Key	Foreign Key
PlAlternate	dataTypeId	dataTypeId
PlDataGranule	dataTypeId	dataTypeId
PlDataGranuleShort	dataTypeId	dataTypeId
PlDataTypeFiles	dataTypeId	dataTypeId

Dependencies on Table: PlDataTypeMaster

Reference by List

Referenced by	Primary Key	Foreign Key
PlAlternate	dataTypeId	dataTypeId
PlDataGranule	dataTypeId	dataTypeId
PlDataGranuleShort	dataTypeId	dataTypeId
PlDataTypeFiles	dataTypeId	dataTypeId
PlDataTypeReq	dataTypeId	dataTypeId
PlDatRecord	dataTypeId	dataTypeId
PlEsdtParam	dataTypeId	dataTypeId
PlOutputDataYield	dataTypeId	dataTypeId
PlPgeCollection	dataTypeId	dataTypeId
PlPrMetadataValue	dataTypeId	dataTypeId

Dependencies on Table: PlDataTypeReq

Reference by List

Referenced by	Primary Key	Foreign Key
PlFileTypeReq	pgeId dataTypeId logicalId	pgeId dataTypeId logicalId
PlMetadataChecks	pgeId dataTypeId logicalId	pgeId dataTypeId logicalId

Dependencies on Table: PlDatRecord

Reference by List

Referenced by	Primary Key	Foreign Key
PlDataGranule	dataTypeId startTime stopTime	dataTypeId startTime stopTime
PlDataGranuleShort	dataTypeId startTime stopTime	dataTypeId startTime stopTime

Dependencies on Table: PlOnDemandPR

Reference by List

Referenced by	Primary Key	Foreign Key
PlAlternateInputValues	productionRequestId	productionRequestId
PlDataProcessingRequest	productionRequestId	productionRequestId
PlErrorAction	productionRequestId	productionRequestId
PlPrCollectionExplosion	productionRequestId	productionRequestId
PlPrMetadataValue	productionRequestId	productionRequestId

Dependencies on Table: PlOutputDataYield

Reference by List

Referenced by	Primary Key	Foreign Key
PlFileTypeReq	pgeId dataTypeId logicalId	pgeId dataTypeId logicalId
PlMetadataChecks	pgeId dataTypeId logicalId	pgeId dataTypeId logicalId
PlOutputFileType	pgeId dataTypeId logicalId	pgeId dataTypeId logicalId

Dependencies on Table: PlPgeCollection

Reference by List

Referenced by	Primary Key	Foreign Key
PlModeRecord	pgeCollectionId	pgeCollectionId
PlPgeMaster	pgeCollectionId	pgeCollectionId

Dependencies on Table: PlPgeDetailData

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrInputData	pgeId	pgeId
DpPrPredStagingProfile	pgeId	pgeId
PlAlternate	pgeId	pgeId
PlCluster	pgeId	pgeId
PlDataHistory	pgeId	pgeId
PlDataProcessingRequest	pgeId	pgeId
PlDataTypeReq	pgeId	pgeId
PlErrorAction	pgeId	pgeId
PlFileTypeReq	pgeId	pgeId
PlMetadataChecks	pgeId	pgeId
PlOutputDataYield	pgeId	pgeId
PlOutputFileType	pgeId	pgeId
PlPgeDetailParameters	pgeId	pgeId

Dependencies on Table: PlPgeDetailParameters

Reference by List

Referenced by	Primary Key	Foreign Key
PlPrParameter	pgeId pgeParameterLogicalId	pgeId pgeParameterLogicalId
PlUserParameterConstraints	pgeId pgeParameterLogicalId	pgeId pgeParameterLogicalId

Dependencies on Table: PlPgeDetailTime

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrInputData	pgeId	pgeId
DpPrPredStagingProfile	pgeId	pgeId
PlAlternate	pgeId	pgeId
PlCluster	pgeId	pgeId
PlDataHistory	pgeId	pgeId
PlDataProcessingRequest	pgeId	pgeId
PlDataTypeReq	pgeId	pgeId
PlErrorAction	pgeId	pgeId
PlFileTypeReq	pgeId	pgeId
PlMetadataChecks	pgeId	pgeId
PlOutputDataYield	pgeId	pgeId
PlOutputFileType	pgeId	pgeId
PlPgeDetailParameters	pgeId	pgeId

Dependencies on Table: PlPgeMaster

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrInputData	pgeId	pgeId
DpPrPredStagingProfile	pgeId	pgeId
PlAlternate	pgeId	pgeId
PlCluster	pgeId	pgeId
PlDataHistory	pgeId	pgeId
PlDataProcessingRequest	pgeId	pgeId
PlDataTypeReq	pgeId	pgeId
PlErrorAction	pgeId	pgeId
PlFileTypeReq	pgeId	pgeId
PlMetadataChecks	pgeId	pgeId
PlOutputDataYield	pgeId	pgeId
PlOutputFileType	pgeId	pgeId
PlPgeDetailParameters	pgeId	pgeId

Dependencies on Table: PlPgePerformance

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrInputData	pgeId	pgeId
DpPrPredStagingProfile	pgeId	pgeId
PlAlternate	pgeId	pgeId
PlCluster	pgeId	pgeId
PlDataHistory	pgeId	pgeId
PlDataProcessingRequest	pgeId	pgeId
PlDataTypeReq	pgeId	pgeId
PlErrorAction	pgeId	pgeId
PlFileTypeReq	pgeId	pgeId
PlMetadataChecks	pgeId	pgeId
PlOutputDataYield	pgeId	pgeId
PlOutputFileType	pgeId	pgeId
PlPgeDetailParameters	pgeId	pgeId
PlPgePriority	pgeId	pgeId
PlResourceRequirement	pgeId	pgeId

Dependencies on Table: PlPlans

Reference by List

Referenced by	Primary Key	Foreign Key
PlPrVsPlan	planId	planId

Dependencies on Table: PlPrCollection

Reference by List

Referenced by	Primary Key	Foreign Key
PlPrCollectionExplosion	prCollectionId	prCollectionId
PlProductionRequest	prCollectionId	prCollectionId

Dependencies on Table: PlProductionRequest

Reference by List

Referenced by	Primary Key	Foreign Key
PlAlternateInputValues	productionRequestId	productionRequestId
PlDataProcessingRequest	productionRequestId	productionRequestId
PlErrorAction	productionRequestId	productionRequestId
PlPrCollectionExplosion	productionRequestId	productionRequestId
PlPrMetadataValue	productionRequestId	productionRequestId
PlPrParameter	productionRequestId	productionRequestId
PlPrVsPlan	productionRequestId	productionRequestId

Dependencies on Table: PlProductionStrategy

Reference by List

Referenced by	Primary Key	Foreign Key
PlPgePriority	strategyId	strategyId
PlPlans	strategyId	strategyId
PlPRPriority	strategyId	strategyId
PlUserPriority	strategyId	strategyId

Dependencies on Table: PlReplanCriteria

Reference by List

Referenced by	Primary Key	Foreign Key
PlDasDelta	criteriaId	criteriaId
PlOnDemandReplanValues	criteriaId	criteriaId

Dependencies on Table: PlResource

Reference by List

Referenced by	Primary Key	Foreign Key
PlGroundEventResourceExplosion	resourceId	resourceId
RpRsvRscExplosion	resourceId	resourceId

Dependencies on Table: PlResourceRequirement

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrExecutable	sswId	sswId
DpPrPge	sswId	sswId
PlDataProcessingRequest	sswId	sswId
PlPgeMaster	sswId	sswId

Dependencies on Table: PlRoutineArrival

Reference by List

Referenced by	Primary Key	Foreign Key
PlAlternate	dataTypeId	dataTypeId
PlDataGranule	dataTypeId	dataTypeId
PlDataGranuleShort	dataTypeId	dataTypeId
PlDataTypeFiles	dataTypeId	dataTypeId
PlDataTypeReq	dataTypeId	dataTypeId
PlDatRecord	dataTypeId	dataTypeId
PlEsdtParam	dataTypeId	dataTypeId
PlFileTypeReq	dataTypeId	dataTypeId
PlMetadataChecks	dataTypeId	dataTypeId
PlOutputDataYield	dataTypeId	dataTypeId
PlOutputFileType	dataTypeId	dataTypeId
PlPgeCollection	dataTypeId	dataTypeId
PlPrMetadataValue	dataTypeId	dataTypeId
PlUsedByCenter	dataTypeId	dataTypeId

Dependencies on Table: PlRscComputer

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrDiskAllocation	computerId	computerId
DpPrRscCpuRamAllocation	computerId	computerId
PlRscDiskPartition	computerId	computerId

Dependencies on Table: PlRscDiskPartition

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrDiskAllocation	diskPartitionId	diskPartitionId

Dependencies on Table: PlRscRealComputer

Reference by List

Referenced by	Primary Key	Foreign Key
PlRscComputer	realComputerId	realComputerId

Dependencies on Table: PlRscString

Reference by List

Referenced by	Primary Key	Foreign Key
PlRscComputer	stringId	stringId

Dependencies on Table: PlTile

Reference by List

Referenced by	Primary Key	Foreign Key
PlDataGranule	tileId	tileId
PlDataGranuleShort	tileId	tileId
PlDatRecord	tileId	tileId
PlTileCoordinates	tileId	tileId

Dependencies on Table: PlTileCoordinates

Dependencies on Table: PlTileScheduled

Reference by List

Referenced by	Primary Key	Foreign Key
DpPrInputData	pgeId	pgeId
DpPrPredStagingProfile	pgeId	pgeId
PlAlternate	pgeId	pgeId
PlCluster	pgeId	pgeId
PlDataHistory	pgeId	pgeId
PlDataProcessingRequest	pgeId	pgeId
PlDataTypeReq	pgeId	pgeId
PlErrorAction	pgeId	pgeId
PlFileTypeReq	pgeId	pgeId
PlMetadataChecks	pgeId	pgeId
PlOutputDataYield	pgeId	pgeId
PlOutputFileType	pgeId	pgeId
PlPgeDetailParameters	pgeId	pgeId
PlPgePriority	pgeId	pgeId
PlPrParameter	pgeId	pgeId
PlResourceRequirement	pgeId	pgeId

Dependencies on Table: PlTimer

Reference by List

Referenced by	Primary Key	Foreign Key
PlDprData	dprId granuleId logicalId	dprId granuleId logicalId

Dependencies on Table: RpActivityType

Reference by List

Referenced by	Primary Key	Foreign Key
PlResource	activityTypeId	activityTypeId
RpResourceReservation	activityTypeId	activityTypeId

Dependencies on Table: RpResourceReservation

Reference by List

Referenced by	Primary Key	Foreign Key
RpReservationInterval	reservationId	reservationId
RpRsvRscExplosion	reservationId	reservationId

Dependencies on Table: RpRsvRscExplosion

4.1.9 Triggers

Sybase supports the enforcement of business policy via the use of triggers. A trigger is best defined as set of activities or checks that should be performed automatically when ever a row is inserted, updated, or deleted from a given table. Sybase version 11.0.1 allows the definition of insert, update, and delete trigger per table. A listing of each the triggers in the PDPS database is given here. A brief definition of each of these triggers follows.

Trigger List

Table	Trigger	Description
DpPrCreationQueue	TrigUpdDpPrCreationQueue	
DpPrCreationQueue	TrigDelDpPrCreationQueue	
DpPrDiskAllocation	TrigInsDpPrDiskAllocation	
DpPrDiskAllocation	TrigUpdDpPrDiskAllocation	
DpPrDprDependList	TrigInsDpPrDprDependList	
DpPrDprDependList	TrigUpdDpPrDprDependList	
DpPrExecutable	TrigUpdDpPrExecutable	
DpPrExecutable	TrigDelDpPrExecutable	
DpPrInputData	TrigInsDpPrInputData	
DpPrInputData	TrigUpdDpPrInputData	
DpPrPcf	TrigInsDpPrPcf	
DpPrPcf	TrigUpdDpPrPcf	
DpPrPge	TrigInsDpPrPge	
DpPrPge	TrigUpdDpPrPge	
DpPrPge	TrigDelDpPrPge	
DpPrPredStagingProfile	TrigUpdDpPrPredStagingProfile	
DpPrPredStagingProfile	TrigDelDpPrPredStagingProfile	
DpPrQaDbParameters	TrigInsDpPrQaDbParameters	
DpPrQaDbParameters	TrigUpdDpPrQaDbParameters	
PlActivePlan	TrigInsPlActivePlan	
PlActivePlan	TrigUpdPlActivePlan	

Trigger List (continued)

Table	Trigger	Description
PlAlternate	TrigInsPlAlternate	
PlAlternate	TrigUpdPlAlternate	
PlAlternateInputValues	TrigInsPlAlternateInputValues	
PlAlternateInputValues	TrigUpdPlAlternateInputValues	
PlCluster	TrigInsPlCluster	
PlCluster	TrigUpdPlCluster	
PlCluster	TrigDelPlCluster	
PlDasDelta	TrigInsPlDasDelta	
PlDasDelta	TrigUpdPlDasDelta	
PlDataAvailTimes	TrigDelPlDataAvailTimes	
PlDataGranuleShort	TrigInsPlDataGranuleShort	
PlDataGranuleShort	TrigUpdPlDataGranuleShort	
PlDataGranuleShort	TrigDelPlDataGranuleShort	
PlDataProcessingRequest	TrigInsPlDataProcessingRequest	
PlDataProcessingRequest	TrigUpdPlDataProcessingRequest	
PlDataProcessingRequest	TrigDelPlDataProcessingRequest	
PlDataSourceMaster	TrigInsPlDataSourceMaster	
PlDataSourceMaster	TrigUpdPlDataSourceMaster	
PlDataSourceMaster	TrigDelPlDataSourceMaster	
PlDataTypeMaster	TrigUpdPlDataTypeMaster	
PlDataTypeMaster	TrigDelPlDataTypeMaster	
PlDataTypeReq	TrigInsPlDataTypeReq	
PlDataTypeReq	TrigUpdPlDataTypeReq	
PlDataTypeReq	TrigDelPlDataTypeReq	
PlDatRecord	TrigInsPlDatRecord	
PlDatRecord	TrigUpdPlDatRecord	
PlDatRecord	TrigDelPlDatRecord	
PlDprData	TrigInsPlDprData	
PlDprData	TrigUpdPlDprData	
PlDprData	TrigDelPlDprData	
PlErrorAction	TrigInsPlErrorAction	
PlErrorAction	TrigUpdPlErrorAction	
PlEsdtParam	TrigInsPlEsdtParam	
PlEsdtParam	TrigUpdPlEsdtParam	
PlEsdtParmValues	TrigInsPlEsdtParmValues	
PlEsdtParmValues	TrigUpdPlEsdtParmValues	
PlMetadataChecks	TrigInsPlMetadataChecks	
PlMetadataChecks	TrigUpdPlMetadataChecks	

Trigger List (continued)

Table	Trigger	Description
PlModeRecord	TrigInsPlModeRecord	
PlModeRecord	TrigUpdPlModeRecord	
PlOnDemandPR	TrigInsPlOnDemandPR	
PlOnDemandPR	TrigUpdPlOnDemandPR	
PlOnDemandReplanValues	TrigInsPlOnDemandReplanValues	
PlOnDemandReplanValues	TrigUpdPlOnDemandReplanValues	
PlOutputDataYield	TrigInsPlOutputDataYield	
PlOutputDataYield	TrigUpdPlOutputDataYield	
PlOutputDataYield	TrigDelPlOutputDataYield	
PlPgeCollection	TrigUpdPlPgeCollection	
PlPgeCollection	TrigDelPlPgeCollection	
PlPgeDetailParameters	TrigInsPlPgeDetailParameters	
PlPgeDetailParameters	TrigUpdPlPgeDetailParameters	
PlPgeDetailParameters	TrigDelPlPgeDetailParameters	
PlPgeDetailTime	TrigInsPlPgeDetailTime	
PlPgeDetailTime	TrigUpdPlPgeDetailTime	
PlPgeMaster	TrigUpdPlPgeMaster	
PlPgeMaster	TrigDelPlPgeMaster	
PlPgePerformance	TrigInsPlPgePerformance	
PlPgePerformance	TrigUpdPlPgePerformance	
PlPgePriority	TrigInsPlPgePriority	
PlPgePriority	TrigUpdPlPgePriority	
PlPlans	TrigInsPlPlans	
PlPlans	TrigUpdPlPlans	
PlPlans	TrigDelPlPlans	
PlPrCollection	TrigUpdPlPrCollection	
PlPrCollection	TrigDelPlPrCollection	
PlPrCollectionExplosion	TrigInsPlPrCollectionExplosion	
PlPrCollectionExplosion	TrigUpdPlPrCollectionExplosion	
PlPrMetadataValue	TrigInsPlPrMetadataValue	
PlPrMetadataValue	TrigUpdPlPrMetadataValue	
PlProductionRequest	TrigInsPlProductionRequest	
PlProductionRequest	TrigUpdPlProductionRequest	
PlProductionRequest	TrigDelPlProductionRequest	
PlProductionStrategy	TrigUpdPlProductionStrategy	
PlProductionStrategy	TrigDelPlProductionStrategy	
PlPrParameter	TrigInsPlPrParameter	
PlPrParameter	TrigUpdPlPrParameter	
PlPRPriority	TrigInsPlPRPriority	

Trigger List (continued)

Table	Trigger	Description
PlPRPriority	TrigUpdPlPRPriority	
PlPrVsPlan	TrigInsPlPrVsPlan	
PlPrVsPlan	TrigUpdPlPrVsPlan	
PlReplanCriteria	TrigDelPlReplanCriteria	
PlRescUseThresh	TrigInsPlRescUseThresh	
PlRescUseThresh	TrigInsPlRescUseThresh	
PlResource	TrigUpdPlResource	
PlResource	TrigDelPlResource	
PlResourceRequirement	TrigInsPlResourceRequirement	
PlResourceRequirement	TrigUpdPlResourceRequirement	
PlResourceRequirement	TrigDelPlResourceRequirement	
PlRoutineArrival	TrigInsPlRoutineArrival	
PlRoutineArrival	TrigUpdPlRoutineArrival	
PlRscComputer	TrigInsPlRscComputer	
PlRscComputer	TrigUpdPlRscComputer	
PlRscComputer	TrigDelPlRscComputer	
PlRscDiskPartition	TrigInsPlRscDiskPartition	
PlRscDiskPartition	TrigUpdPlRscDiskPartition	
PlRscDiskPartition	TrigDelPlRscDiskPartition	
PlRscString	TrigInsPlRscString	
PlRscString	TrigUpdPlRscString	
PlRscString	TrigDelPlRscString	
PlTile	TrigInsPlTile	
PlTile	TrigUpdPlTile	
PlTile	TrigDelPlTile	
PlTileCoordinates	TrigInsPlTileCoordinates	
PlTileCoordinates	TrigUpsPlTileCoordinates	
PlTileScheduled	TrigInsPlTileScheduled	
PlTileScheduled	TrigUpdPlTileScheduled	
PlTileScheduled	TrigDelPlTileScheduled	
PlUsedByCenter	TrigInsPlUsedByCenter	
PlUsedByCenter	TrigUpdPlUsedByCenter	
PlUserParameterConstraints	TrigInsPlUserParamConstraints	
PlUserParameterConstraints	TrigUpdPlUserParamConstraints	
PlUserPriority	TrigInsPlUserPriority	
PlUserPriority	TrigUpdPlUserPriority	

Trigger: TrigUpdDpPrCreationQueue

Trigger Code

```
/*    */
CREATE TRIGGER TrigUpdDpPrCreationQueue
ON DpPrCreationQueue
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @numrows int  /* number of rows updated */
   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( dprId )
   BEGIN

      /* cascade update to child table */

      UPDATE DpPrDprDependList
      SET t.dprId = i.dprId
      FROM DpPrDprDependList t, inserted i, deleted d
      WHERE t.dprId = d.dprId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   RETURN

END
```

```
go
```

Trigger: TrigDelDpPrCreationQueue

Trigger Code

```
--
CREATE TRIGGER TrigDelDpPrCreationQueue
ON DpPrCreationQueue
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in table DpPrDprDependList */

   DELETE DpPrDprDependList FROM DpPrDprDependList t, deleted d
   WHERE t.dprId = d.dprId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsDpPrDiskAllocation

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsDpPrDiskAllocation
ON DpPrDiskAllocation
FOR INSERT
AS
BEGIN
```

```
   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key diskPartitionId has a
     matching primary key in table PlRscDiskPartition */

   IF (SELECT COUNT(*)
      FROM PlRscDiskPartition t, inserted i
      WHERE t.diskPartitionId = i.diskPartitionId
      AND   t.computerId = i.computerId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for (diskPartitionId,computerId) found in table
PlRscDiskPartition"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdDpPrDiskAllocation

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdDpPrDiskAllocation
ON DpPrDiskAllocation
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
```

```
   IF @num_rows = 0
      RETURN

   IF UPDATE ( diskPartitionId ) OR UPDATE ( computerId )
   BEGIN

      /* Verify that foreign key diskPartitionId has a
       * matching primary key in table PlRscDiskPartition */

      IF (SELECT COUNT(*)
         FROM PlRscDiskPartition t, inserted i
         WHERE t.diskPartitionId = i.diskPartitionId
         AND   t.computerId = i.computerId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for (diskPartitionId,computerId) found in table
PlRscDiskPartition"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsDpPrDprDependList

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsDpPrDprDependList
ON DpPrDprDependList
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN
```

```
   /* Verify that foreign key dprId has a
    * matching primary key in table DpPrCreationQueue */

   IF (SELECT COUNT(*)
      FROM DpPrCreationQueue t, inserted i
      WHERE t.dprId = i.dprId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dprId found in table DpPrCreationQueue"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdDpPrDprDependList

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdDpPrDprDependList
ON DpPrDprDependList
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( dprId )
   BEGIN

      /* Verify that foreign key dprId has a
       * matching primary key in table DpPrCreationQueue */
```

```
      IF (SELECT COUNT(*)
         FROM DpPrCreationQueue t, inserted i
         WHERE t.dprId = i.dprId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for dprId found in table DpPrCreationQueue"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigUpdDpPrExecutable

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdDpPrExecutable
ON DpPrExecutable
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @numrows int  /* number of rows updated */
   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
      RETURN

   /* see if the primary key was updated */

   IF UPDATE (sswId) OR UPDATE(execName) OR UPDATE(execMachine)
   BEGIN

      /* cascade update to child table */
```

```
      UPDATE DpPrPge
      SET t.sswId = i.sswId,
         t.execName = i.execName,
         t.execMachine = i.execMachine
      FROM DpPrPge t, inserted i, deleted d
      WHERE t.sswId = d.sswId AND
            t.execName = d.execName AND
            t.execMachine = d.execMachine

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelDpPrExecutable

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelDpPrExecutable
ON DpPrExecutable
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
      RETURN

   /* delete matching rows in table DpPrPge */

   DELETE DpPrPge FROM DpPrPge t, deleted d
   WHERE t.sswId = d.sswId
   AND   t.execName = d.execName
   AND   t.execMachine = d.execMachine
```

```
   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
         ROLLBACK TRANSACTION
         RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsDpPrInputData

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsDpPrInputData
ON DpPrInputData
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key has a matching
    * primary key in table DpPrPredStagingProfile */

   IF (SELECT COUNT(*)
      FROM DpPrPredStagingProfile t, inserted i
      WHERE t.processingType = i.processingType AND
            t.pgeId = i.pgeId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId/processingType in DpPrPredStagingProfile"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
```

go

Trigger: TrigUpdDpPrInputData

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdDpPrInputData
ON DpPrInputData
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( pgeId ) OR UPDATE ( processingType )
   BEGIN

      /* Verify that foreign key has a matching
       * primary key in table DpPrPredStagingProfile */

      IF (SELECT COUNT(*)
         FROM DpPrPredStagingProfile t, inserted i
         WHERE t.processingType = i.processingType AND
               t.pgeId = i.pgeId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for processingType/pgeId in table
DpPrPredStagingProfile"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsDpPrPcf

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsDpPrPcf
ON DpPrPcf
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key has a matching
    * primary key in table DpPrPge          */

   IF (SELECT COUNT(*)
      FROM DpPrPge t, inserted i
      WHERE t.dprId = i.dprId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dprId in table DpPrPge"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdDpPrPcf

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdDpPrPcf
ON DpPrPcf
FOR UPDATE
AS
BEGIN
```

```
   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( dprId )
   BEGIN

      /* Verify that foreign key has a matching
       * primary key in table DpPrPge          */

      IF (SELECT COUNT(*)
         FROM DpPrPge t, inserted i
         WHERE t.dprId = i.dprId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for dprId in table DpPrPge"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsDpPrPge

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsDpPrPge
ON DpPrPge
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
```

```
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key has a matching
    * primary key in table DpPrExecutable          */

   IF (SELECT COUNT(*)
      FROM DpPrExecutable t, inserted i
      WHERE t.sswId = i.sswId
      AND   t.execName = i.execName
      AND   t.execMachine = i.execMachine)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for (sswId/execName/execMachine) in table
DpPrExecutable"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdDpPrPge

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdDpPrPge
ON DpPrPge
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @numrows int  /* number of rows updated */
   SELECT @numrows = @@rowcount

   /* if no rows were updated, then return */

   IF @numrows = 0
     RETURN
```

```
   /* see if the primary key was updated */

   IF UPDATE ( dprId )
   BEGIN

      /* cascade update to child table */

      UPDATE DpPrPcf
      SET t.dprId = i.dprId
      FROM DpPrPcf t, inserted i, deleted d
      WHERE t.dprId = d.dprId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

  /* see if any foreign keys were updated */

  IF UPDATE ( sswId ) OR UPDATE ( execName ) OR UPDATE ( execMachine )
  BEGIN
     IF (SELECT COUNT(*)
        FROM DpPrExecutable t, inserted i
        WHERE t.sswId = i.sswId
        AND   t.execName = i.execName
        AND   t.execMachine = i.execMachine)
        != @numrows
     BEGIN
        RAISERROR 31112 "No match for (sswId/execName/execMachine) in table
DpPrExecutable"
        ROLLBACK TRANSACTION
        RETURN
     END
  END

  RETURN

END
go
```

Trigger: TrigDelDpPrPge

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelDpPrPge
ON DpPrPge
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in table DpPrPcf */

   DELETE DpPrPcf FROM DpPrPcf t, deleted d
   WHERE t.dprId = d.dprId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
         ROLLBACK TRANSACTION
         RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdDpPrPredStagingProfile

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigUpdDpPrPredStagingProfile
ON DpPrPredStagingProfile
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @numrows int  /* number of rows updated */
   SELECT @numrows = @@rowcount
```

```
   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( processingType ) OR UPDATE ( pgeId )
   BEGIN

      /* cascade update to child table */

      UPDATE DpPrInputData
      SET t.processingType = i.processingType,
         t.pgeId = i.pgeId
      FROM DpPrInputData t, inserted i, deleted d
      WHERE t.processingType = d.processingType AND
           t.pgeId = d.pgeId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelDpPrPredStagingProfile

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelDpPrPredStagingProfile
ON DpPrPredStagingProfile
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */
```

```
   IF @@rowcount = 0
     RETURN

   /* delete matching rows in table DpPrInputData */

   DELETE DpPrInputData FROM DpPrInputData t, deleted d
   WHERE t.processingType = d.processingType
   AND   t.pgeId = d.pgeId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsDpPrQaDbParameters

Trigger Code

```
/*
 *
 * Note that this is special trigger, not just for referential integrity.
 *                                                    */
CREATE TRIGGER TrigInsDpPrQaDbParameters
ON DpPrQaDbParameters
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that column parmType has a matching
    * value in column dataTypeName in table PlDataTypeMaster */
```

```
   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeName = i.parmType) != @num_rows
   BEGIN
      RAISERROR 31111 "Value supplied for parmType does not match any dataTypeName in
table PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that either parmKeyword or (parmMin, parmMax) are set,
    * but not both.                                    */

   IF (SELECT COUNT(*) FROM inserted i WHERE i.parmKeyword != "" AND
     (i.parmMin != "" OR i.parmMax != "") ) > 0
   BEGIN
      RAISERROR 31111 "Attempt to insert by both keyword and maxmin into
DpPrQaDbParameters"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdDpPrQaDbParameters

Trigger Code

```
/*
 *
 * Note that this is special trigger, not for referential integrity.
 *                                                  */
CREATE TRIGGER TrigUpdDpPrQaDbParameters
ON DpPrQaDbParameters
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
```

```
   IF @num_rows = 0
      RETURN

   /* Verify that column parmType has a matching
    * value in column dataTypeName in table PlDataTypeMaster */

   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeName = i.parmType) != @num_rows
   BEGIN
      RAISERROR 31117 "Value of parmType does not match any dataTypeName in table
PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that either parmKeyword or (parmMin, parmMax) are set,
    * but not both                                        */

   IF (SELECT COUNT(*) FROM inserted i WHERE i.parmKeyword != "" AND
     (i.parmMin != "" OR i.parmMax != "") ) > 0
   BEGIN
      RAISERROR 31112 "Attempt to insert by both keyword and maxmin into
DpPrQaDbParameters"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlActivePlan

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlActivePlan
ON PlActivePlan
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */
```

```
   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key planId has a matching
     primary key in table PlPlans          */

   IF (SELECT COUNT(*)
      FROM PlPlans t, inserted i
      WHERE t.planId = i.planId) != @num_rows
   BEGIN
      RAISERROR 31112 "No match for planId found in table PlPlans"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlAlternate

Trigger Code

```
CREATE TRIGGER TrigInsPlAlternate
ON PlAlternate
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key (pgeId, dataTypeId, logicalId)  has a matching
     primary key in table PlDataTypeReq */

   IF (SELECT COUNT(*)
      FROM PlDataTypeReq t, inserted i
```

```
      WHERE t.pgeId = i.pgeId
      AND   t.dataTypeId = i.dataTypeId
      AND   t.logicalId = i.logicalId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for (pgeId, dataTypeId, logicalId) in PlDataTypeReq)"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlAlternate

Trigger Code

```
CREATE TRIGGER TrigUpdPlAlternate
ON PlAlternate
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key (pgeId, dataTypeId, logicalId) has a matching
      primary key in table PlDataTypeReq                    */

   IF (SELECT COUNT(*)
      FROM PlDataTypeReq t, inserted i
      WHERE t.pgeId = i.pgeId
      AND   t.dataTypeId = i.dataTypeId
      AND   t.logicalId = i.logicalId) != @num_rows
   BEGIN
      RAISERROR 31112 "Invalid foreign key (no matching primary key in PlDataTypeReq"
      ROLLBACK TRANSACTION
      RETURN
   END
```

```
   RETURN

END
go
```

Trigger: TrigInsPlAlternateInputValues

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlAlternateInputValues
ON PlAlternateInputValues
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlProductionRequest */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for productionRequestId in table PlProductionRequest"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlAlternateInputValues

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlAlternateInputValues
ON PlAlternateInputValues
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlProductionRequest     */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31112 "No match for productionRequestId in table PlProductionRequest"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlCluster

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlCluster
ON PlCluster
FOR INSERT
AS
BEGIN
```

```
   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key pgeId has a matching
     primary key in table PlTileScheduled */

   IF (SELECT COUNT(*)
      FROM PlTileScheduled t, inserted i
      WHERE t.pgeId = i.pgeId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId in PlTileScheduled"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlCluster

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlCluster
ON PlCluster
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Note that primary key is an identity, which cannot be updated */
```

```
   /* Verify that key pgeId has a matching
      primary key in table PlTileScheduled */

   IF (SELECT COUNT(*)
       FROM PlTileScheduled t, inserted i
       WHERE t.pgeId = i.pgeId) != @num_rows
   BEGIN
       RAISERROR 31112 "No match for pgeId in PlTileScheduled"
       ROLLBACK TRANSACTION
       RETURN
   END

   RETURN

END
go
```

Trigger: TrigDelPlCluster

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlCluster
ON PlCluster
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
      RETURN

   /* delete matching rows in PlTile */

   DELETE PlTile
   FROM   PlTile t, deleted d
   WHERE  t.clusterId = d.clusterId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
       ROLLBACK TRANSACTION  /* undo all deletes */
       RETURN
   END
```

```
   RETURN

END
go
```

Trigger: TrigInsPlDasDelta

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlDasDelta
ON PlDasDelta
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key criteriaId has a matching
     primary key in table PlReplanCriteria */

   IF (SELECT COUNT(*)
      FROM PlReplanCriteria t, inserted i
      WHERE t.criteriaId = i.criteriaId) != @num_rows
   BEGIN
      RAISERROR 31111 "No matching primary key found in PlReplanCriteria"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlDasDelta

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlDasDelta
ON PlDasDelta
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key criteriaId has a matching
     primary key in table PlReplanCriteria     */

   IF (SELECT COUNT(*)
      FROM PlReplanCriteria t, inserted i
      WHERE t.criteriaId = i.criteriaId) != @num_rows
   BEGIN
      RAISERROR 31112 "No match for criteriaId found in PlReplanCriteria"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigDelPlDataAvailTimes

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlDataAvailTimes
ON PlDataAvailTimes
FOR DELETE
AS
BEGIN
```

```
   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlDatRecord */

   DELETE PlDatRecord
   FROM   PlDatRecord t, deleted d
   WHERE  t.availTimesId = d.availTimesId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlDataGranuleShort

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlDataGranuleShort
ON PlDataGranuleShort
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key tileId, if not null or zero, has a matching
     primary key in table PlTile */
/*
   IF (SELECT COUNT(*)
      FROM PlTile t, inserted i
```

```
      WHERE t.tileId = i.tileId) +
      (SELECT COUNT(*) FROM inserted i
       WHERE i.tileId = 0 ) +
      (SELECT COUNT(*) FROM inserted i
       WHERE i.tileId is null ) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for tileId in table PlTile"
      ROLLBACK TRANSACTION
      RETURN
   END
*/
   /* Verify that key dataTypeId has a matching
     primary key in table PlDataTypeMaster */

   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeId = i.dataTypeId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dataTypeId in table PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlDataGranuleShort

Trigger Code

```
CREATE TRIGGER TrigUpdPlDataGranuleShort
ON PlDataGranuleShort
FOR UPDATE
AS

   DECLARE @numrows int  --/* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   --/* if no rows were deleted, then return */
```

```
IF @numrows = 0
  RETURN

--/* see if the primary key was updated */

IF UPDATE ( granuleId )
BEGIN

   --/* cascade update to child tables */

   UPDATE PlDprData
   SET    t.granuleId = i.granuleId
   FROM   PlDprData t,
          inserted i, deleted d
   WHERE  t.granuleId = d.granuleId

   IF @@transtate = 2 --/* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   UPDATE PlEsdtParmValues
   SET    t.granuleId = i.granuleId
   FROM   PlEsdtParmValues t,
          inserted i, deleted d
   WHERE  t.granuleId = d.granuleId

   IF @@transtate = 2 --/* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   UPDATE PlDataHistory
   SET    t.granuleId = i.granuleId
   FROM   PlDataHistory t,
          inserted i, deleted d
   WHERE  t.granuleId = d.granuleId

   IF @@transtate = 2 --/* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END
```

```
   END

   --/* If foreign keys have been updated, verify that they
     --match the appropriate primary key.                  */
/*
   IF UPDATE ( tileId )
   BEGIN

      --/* Verify that key tileId, if not zero, has a
        --matching primary key in table PlTile       */

      IF (SELECT COUNT(*)
          FROM PlTile t, inserted i
          WHERE t.tileId = i.tileId) +
         (SELECT COUNT(*)
          FROM inserted i
          WHERE i.tileId = 0)
          != @numrows
      BEGIN
          RAISERROR 31112 "No match for tileId in table PlTile"
          ROLLBACK TRANSACTION
          RETURN
      END

   END
*/
   IF UPDATE ( dataTypeId )
   BEGIN

      --/* Verify that key dataTypeId has a matching
        --primary key in table PlDataTypeMaster     */

      IF (SELECT COUNT(*)
          FROM PlDataTypeMaster t, inserted i
          WHERE t.dataTypeId = i.dataTypeId) != @numrows
      BEGIN
          RAISERROR 31112 "No match for dataTypeId in table PlDataTypeMaster"
          ROLLBACK TRANSACTION
          RETURN
      END

   END

   IF UPDATE ( universalReference )
```

```
   BEGIN

      --/* Cascade update to DpPrFile */

      UPDATE DpPrFile
      SET    t.universalReference = i.universalReference
      FROM   DpPrFile t, inserted i, deleted d
      WHERE  t.universalReference = d.universalReference

   END

   RETURN

END
go
```

Trigger: TrigDelPlDataGranuleShort

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlDataGranuleShort
ON PlDataGranuleShort
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlDprData */

   DELETE PlDprData
   FROM   PlDprData t, deleted d
   WHERE  t.granuleId = d.granuleId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlEsdtParmValues */
```

```
   DELETE PlEsdtParmValues
   FROM   PlEsdtParmValues t, deleted d
   WHERE  t.granuleId = d.granuleId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlDataProcessingRequest

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlDataProcessingRequest
ON PlDataProcessingRequest
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key productionRequestId has a matching
     primary key in table PlProductionRequest */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for productionRequestId in table PlProductionRequest"
      ROLLBACK TRANSACTION
      RETURN
```

```
   END

   /* Verify that foreign key pgeId has a matching
      primary key in table PlPgeMaster */

   IF (SELECT COUNT(*)
       FROM PlPgeMaster t, inserted i
       WHERE t.pgeId = i.pgeId)
       != @num_rows
   BEGIN
       RAISERROR 31111 "No match for pgeId in table PlPgeMaster"
       ROLLBACK TRANSACTION
       RETURN
   END

   /* Verify that foreign key sswId, if specified,  has a matching
      primary key in table PlResourceRequirement */

   IF ((SELECT COUNT(*)
       FROM PlResourceRequirement t, inserted i
       WHERE t.sswId = i.sswId) +
      (SELECT COUNT(*)
       FROM inserted i
       WHERE i.sswId is null) +
      (SELECT COUNT(*)
       FROM inserted i
       WHERE i.sswId = ""))
       != @num_rows
   BEGIN
       RAISERROR 31111 "No match for sswId in table PlResourceRequirement"
       ROLLBACK TRANSACTION
       RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlDataProcessingRequest

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlDataProcessingRequest
```

```
ON PlDataProcessingRequest
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( dprId )
   BEGIN

      /* cascade update to child tables */

      UPDATE PlDprData
      SET t.dprId = i.dprId
      FROM PlDprData t,
          inserted i, deleted d
      WHERE t.dprId = d.dprId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlDataProcessingRequest

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlDataProcessingRequest
ON PlDataProcessingRequest
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlDprData */

   DELETE PlDprData
   FROM   PlDprData t, deleted d
   WHERE  t.dprId = d.dprId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlDataSourceMaster

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlDataSourceMaster
ON PlDataSourceMaster
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
```

```
      RETURN

   /* Verify that key dataTypeId has a matching
     primary key in table PlDataTypeMaster */

   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeId = i.dataTypeId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dataTypeId found in table PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlDataSourceMaster

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlDataSourceMaster
ON PlDataSourceMaster
FOR UPDATE
AS
BEGIN

   DECLARE @numrows int  /* number of rows updated */
   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if dataTypeId was updated */

   IF UPDATE ( dataTypeId )
   BEGIN

      /* This is a foreign key, as well as primary */
      /* First verify that the new key is valid    */
```

```
      IF (SELECT COUNT(*)
         FROM PlDataTypeMaster t, inserted i
         WHERE t.dataTypeId = i.dataTypeId) != @numrows
      BEGIN
         RAISERROR 31112 "No match for dataTypeId found in table PlDataTypeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

      /* cascade update to child table */

      UPDATE PlRoutineArrival
      SET    t.dataTypeId = i.dataTypeId
      FROM   PlRoutineArrival t,
             inserted i, deleted d
      WHERE  t.dataTypeId = d.dataTypeId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlDataSourceMaster

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlDataSourceMaster
ON PlDataSourceMaster
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
```

```
      RETURN

   /* delete matching rows in PlRoutineArrival */

   DELETE PlRoutineArrival
   FROM   PlRoutineArrival t, deleted d
   WHERE  t.dataTypeId = d.dataTypeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigDelPlDataTypeMaster

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlDataTypeMaster
ON PlDataTypeMaster
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
      RETURN

   /* delete matching rows in PlDatRecord  */

   DELETE PlDatRecord
   FROM   PlDatRecord t, deleted d
   WHERE  t.dataTypeId = d.dataTypeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
```

```
END

/* delete matching rows in PlDataSourceMaster  */

DELETE PlDataSourceMaster
FROM   PlDataSourceMaster t, deleted d
WHERE  t.dataTypeId = d.dataTypeId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlEsdtParam  */

DELETE PlEsdtParam
FROM   PlEsdtParam t, deleted d
WHERE  t.dataTypeId = d.dataTypeId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlUsedByCenter  */

DELETE PlUsedByCenter
FROM   PlUsedByCenter t, deleted d
WHERE  t.dataTypeId = d.dataTypeId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlDataTypeReq  */

DELETE PlDataTypeReq
FROM   PlDataTypeReq t, deleted d
WHERE  t.dataTypeId = d.dataTypeId

IF @@transtate = 2  /* previous statement aborted */
```

```
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlOutputDataYield  */

   DELETE PlOutputDataYield
   FROM   PlOutputDataYield t, deleted d
   WHERE  t.dataTypeId = d.dataTypeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlDataTypeReq

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlDataTypeReq
ON PlDataTypeReq
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key pgeId has a matching
      primary key in table PlPgeMaster            */

   IF (SELECT COUNT(*)
      FROM PlPgeMaster t, inserted i
```

```
      WHERE t.pgeId = i.pgeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId in table PlPgeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that foreign key dataTypeId has a matching
     primary key in table PlDataTypeMaster            */

   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeId = i.dataTypeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dataTypeId in table PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlDataTypeReq

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlDataTypeReq
ON PlDataTypeReq
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
```

```
  RETURN

/* see if the primary key was updated */

IF UPDATE ( pgeId ) OR UPDATE ( dataTypeId )
BEGIN

   /* cascade update to child tables */

   UPDATE PlMetadataChecks
   SET    t.pgeId = i.pgeId,
          t.dataTypeId = i.dataTypeId
   FROM   PlMetadataChecks t,
          inserted i, deleted d
   WHERE  t.pgeId = d.pgeId
   AND    t.dataTypeId = d.dataTypeId
   AND    t.ioFlag = 0

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   UPDATE PlAlternate
   SET    t.pgeId = i.pgeId,
          t.dataTypeId = i.dataTypeId
   FROM   PlAlternate t,
          inserted i, deleted d
   WHERE  t.pgeId = d.pgeId
   AND    t.dataTypeId = d.dataTypeId

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

END

/* see if any foreign keys were updated */

IF UPDATE ( pgeId )
BEGIN
```

```
      /* Verify that foreign key pgeId has a matching
       * primary key in table PlPgeMaster            */

      IF (SELECT COUNT(*)
         FROM PlPgeMaster t, inserted i
         WHERE t.pgeId = i.pgeId)
         != @numrows
      BEGIN
         RAISERROR 31112 "No match for pgeId in table PlPgeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   IF UPDATE ( dataTypeId )
   BEGIN

      /* Verify that foreign key dataTypeId has a matching
       * primary key in table PlDataTypeMaster            */

      IF (SELECT COUNT(*)
         FROM PlDataTypeMaster t, inserted i
         WHERE t.dataTypeId = i.dataTypeId)
         != @numrows
      BEGIN
         RAISERROR 31112 "No match for dataTypeId in table PlDataTypeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlDataTypeReq

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlDataTypeReq
ON PlDataTypeReq
```

```
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlMetadataChecks */

   DELETE PlMetadataChecks
   FROM   PlMetadataChecks t, deleted d
   WHERE  t.pgeId = d.pgeId
   AND    t.dataTypeId = d.dataTypeId
   AND    t.ioFlag = 0

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlAlternate */

   DELETE PlAlternate
   FROM   PlAlternate t, deleted d
   WHERE  t.pgeId = d.pgeId
   AND    t.dataTypeId = d.dataTypeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlDatRecord

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlDatRecord
ON PlDatRecord
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key availTimesId has a matching
     primary key in table PlDataAvailTimes */

   IF (SELECT COUNT(*)
      FROM PlDataAvailTimes t, inserted i
      WHERE t.availTimesId = i.availTimesId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for availTimesId in table PlDataAvailTimes"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that key dataTypeId has a matching
     primary key in table PlDataTypeMaster */

   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeId = i.dataTypeId) != @num_rows
   BEGIN
      raiserror 31111 "No match for dataTypeId in table PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlDatRecord

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlDatRecord
ON PlDatRecord
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* see if the primary key was updated */
/*
   IF UPDATE ( dataTypeId )
   BEGIN

      /* cascade update to child table */

      UPDATE PlEDOSDATRecord
      SET    t.dataTypeId = i.dataTypeId
      FROM   PlEDOSDATRecord t,
             inserted i, deleted d
      WHERE  t.dataTypeId = d.dataTypeId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END
*/
   /* If foreign keys were updated, then check for matching
      primary key                                    */

   IF UPDATE ( availTimesId )
   BEGIN

      /* Verify that key availTimesId has a matching
         primary key in table PlDataAvailTimes       */
```

```
    IF (SELECT COUNT(*)
       FROM PlDataAvailTimes t, inserted i
       WHERE t.availTimesId = i.availTimesId) != @num_rows
    BEGIN
       RAISERROR 31112 "No match for availTimesId in table PlDataAvailTimes"
       ROLLBACK TRANSACTION
       RETURN
    END
  END

  /* Verify that key dataTypeId has a matching
     primary key in table PlDataTypeMaster     */

  IF UPDATE ( dataTypeId )
  BEGIN

    IF (SELECT COUNT(*)
       FROM PlDataTypeMaster t, inserted i
       WHERE t.dataTypeId = i.dataTypeId) != @num_rows
    BEGIN
       RAISERROR 31112 "No match for dataTypeId in table PlDataTypeMaster"
       ROLLBACK TRANSACTION
       RETURN
    END

  END

  RETURN

END
go
```

Trigger: TrigDelPlDatRecord

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlDatRecord
ON PlDatRecord
FOR DELETE
AS
BEGIN

  /* if no rows were deleted, then return */
```

```
   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlEDOSDATRecord */
/*
   DELETE PlEDOSDATRecord
   FROM   PlEDOSDATRecord t, deleted d
   WHERE  t.dataTypeId = d.dataTypeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END
*/
   RETURN

END
go
```

Trigger: TrigInsPlDprData

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlDprData
ON PlDprData
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key dprId has a matching
     primary key in table PlDataProcessingRequest */

   IF (SELECT COUNT(*)
      FROM PlDataProcessingRequest t, inserted i
      WHERE t.dprId = i.dprId) != @num_rows
```

```
   BEGIN
      RAISERROR 31111 "No match for dprId in table PlDataProcessingRequest"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that key granuleId has a matching
     primary key in table PlDataGranuleShort           */

   IF (SELECT COUNT(*)
      FROM PlDataGranuleShort t, inserted i
      WHERE t.granuleId = i.granuleId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for granuleId in table PlDataGranuleShort"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlDprData

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlDprData
ON PlDprData
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key dprId has a matching
     primary key in table PlDataProcessingRequest */

   IF UPDATE ( dprId )
```

```
   BEGIN
      IF (SELECT COUNT(*)
         FROM PlDataProcessingRequest t, inserted i
         WHERE t.dprId = i.dprId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for dprId in table PlDataProcessingRequest"
         ROLLBACK TRANSACTION
         RETURN
      END
   END

   /* Verify that key granuleId has a matching
     primary key in table PlDataGranuleShort            */

   IF UPDATE ( granuleId )
   BEGIN
      IF (SELECT COUNT(*)
         FROM PlDataGranuleShort t, inserted i
         WHERE t.granuleId = i.granuleId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for granuleId in table PlDataGranuleShort"
         ROLLBACK TRANSACTION
         RETURN
      END
   END

   RETURN

END
go
```

Trigger: TrigDelPlDprData

Trigger Code

```
CREATE TRIGGER TrigDelPlDprData
ON PlDprData
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN
```

```
   /* delete matching rows in PlTimer */

   DELETE PlTimer
   FROM   PlTimer t, deleted d
   WHERE  t.dprId = d.dprId
   AND    t.granuleId = d.granuleId
   AND    t.logicalId = d.logicalId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlErrorAction

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlErrorAction
ON PlErrorAction
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlPgeMaster            */

   IF (SELECT COUNT(*)
      FROM PlPgeMaster t, inserted i
      WHERE t.pgeId = i.pgeId)
      != @num_rows
```

```
   BEGIN
      RAISERROR 31111 "No match for pgeId in table PlPgeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlErrorAction

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlErrorAction
ON PlErrorAction
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlPgeMaster            */

   IF UPDATE ( pgeId )
   BEGIN
      IF (SELECT COUNT(*)
         FROM PlPgeMaster t, inserted i
         WHERE t.pgeId = i.pgeId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for pgeId in table PlPgeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END
   END
```

```
   RETURN

END
go
```

Trigger: TrigInsPlEsdtParam

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlEsdtParam
ON PlEsdtParam
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlDataTypeMaster        */

   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeId = i.dataTypeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dataTypeId in table PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlEsdtParam

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlEsdtParam
ON PlEsdtParam
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlDataTypeMaster        */

   IF UPDATE ( dataTypeId )
   BEGIN
      IF (SELECT COUNT(*)
         FROM PlDataTypeMaster t, inserted i
         WHERE t.dataTypeId = i.dataTypeId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for dataTypeId in table PlDataTypeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END
   END

   RETURN

END
go
```

Trigger: TrigInsPlEsdtParmValues

Trigger Code

```
/*    */
CREATE TRIGGER TrigInsPlEsdtParmValues
ON PlEsdtParmValues
FOR INSERT
AS
BEGIN
```

```
   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlDataGranuleShort       */

   IF (SELECT COUNT(*)
      FROM PlDataGranuleShort t, inserted i
      WHERE t.granuleId = i.granuleId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for granuleId in table PlDataGranuleShort"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlEsdtParmValues

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlEsdtParmValues
ON PlEsdtParmValues
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
```

```
 * primary key in table PlDataGranuleShort       */

   IF UPDATE ( granuleId )
   BEGIN
      IF (SELECT COUNT(*)
         FROM PlDataGranuleShort t, inserted i
         WHERE t.granuleId = i.granuleId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for granuleId in table PlDataGranuleShort"
         ROLLBACK TRANSACTION
         RETURN
      END
   END

   RETURN

END
go
```

Trigger: ti_plfiletypereq

Trigger Code

```
/*  Insert trigger "ti_plfiletypereq" for table "PlFileTypeReq"  */
create trigger ti_plfiletypereq on PlFileTypeReq for insert as
begin
   declare
      @numrows  int,
      @numnull  int,
      @errno    int,
      @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
      return


   /*  Parent "PlDataTypeReq" must exist when inserting a child in "PlFileTypeReq"  */
   if update(pgeId) or
      update(dataTypeId) or
      update(logicalId)
   begin
      if (select count(*)
         from   PlDataTypeReq t1, inserted t2
```

```
        where  t1.pgeId = t2.pgeId
         and   t1.dataTypeId = t2.dataTypeId
         and   t1.logicalId = t2.logicalId) != @numrows
       begin
         select @errno  = 30002,
               @errmsg = 'Parent does not exist in "PlDataTypeReq". Cannot create child in
"PlFileTypeReq".'
         goto error
       end
   end

   /*  Parent "PlOutputDataYield" must exist when inserting a child in "PlFileTypeReq"  */
   if update(pgeId) or
     update(dataTypeId) or
     update(logicalId)
   begin
     if (select count(*)
        from   PlOutputDataYield t1, inserted t2
        where  t1.pgeId = t2.pgeId
         and   t1.dataTypeId = t2.dataTypeId
         and   t1.logicalId = t2.logicalId) != @numrows
       begin
         select @errno  = 30002,
               @errmsg = 'Parent does not exist in "PlOutputDataYield". Cannot create child in
"PlFileTypeReq".'
         goto error
       end
   end

   /*  Parent "PlPgeDetailData" must exist when inserting a child in "PlFileTypeReq"  */
   if update(pgeId)
   begin
     if (select count(*)
        from   PlPgeDetailData t1, inserted t2
        where  t1.pgeId = t2.pgeId) != @numrows
       begin
         select @errno  = 30002,
               @errmsg = 'Parent does not exist in "PlPgeDetailData". Cannot create child in
"PlFileTypeReq".'
         goto error
       end
   end

   /*  Parent "PlPgeDetailTime" must exist when inserting a child in "PlFileTypeReq"  */
   if update(pgeId)
```

```
   begin
     if (select count(*)
         from   PlPgeDetailTime t1, inserted t2
         where  t1.pgeId = t2.pgeId) != @numrows
        begin
          select @errno  = 30002,
                 @errmsg = 'Parent does not exist in "PlPgeDetailTime". Cannot create child in
"PlFileTypeReq".'
          goto error
        end
   end

   /*  Parent "PlPgeMaster" must exist when inserting a child in "PlFileTypeReq"  */
   if update(pgeId)
   begin
     if (select count(*)
         from   PlPgeMaster t1, inserted t2
         where  t1.pgeId = t2.pgeId) != @numrows
        begin
          select @errno  = 30002,
                 @errmsg = 'Parent does not exist in "PlPgeMaster". Cannot create child in
"PlFileTypeReq".'
          goto error
        end
   end

   /*  Parent "PlPgePerformance" must exist when inserting a child in "PlFileTypeReq"  */
   if update(pgeId)
   begin
     if (select count(*)
         from   PlPgePerformance t1, inserted t2
         where  t1.pgeId = t2.pgeId) != @numrows
        begin
          select @errno  = 30002,
                 @errmsg = 'Parent does not exist in "PlPgePerformance". Cannot create child in
"PlFileTypeReq".'
          goto error
        end
   end

   /*  Parent "PlRoutineArrival" must exist when inserting a child in "PlFileTypeReq"  */
   if update(dataTypeId)
   begin
     if (select count(*)
         from   PlRoutineArrival t1, inserted t2
```

```
        where  t1.dataTypeId = t2.dataTypeId) != @numrows
       begin
         select @errno  = 30002,
               @errmsg = 'Parent does not exist in "PlRoutineArrival". Cannot create child in
"PlFileTypeReq".'
         goto error
       end
   end

   /*  Parent "PlTileScheduled" must exist when inserting a child in "PlFileTypeReq"  */
   if update(pgeId)
   begin
     if (select count(*)
        from   PlTileScheduled t1, inserted t2
        where  t1.pgeId = t2.pgeId) != @numrows
       begin
         select @errno  = 30002,
               @errmsg = 'Parent does not exist in "PlTileScheduled". Cannot create child in
"PlFileTypeReq".'
         goto error
       end
   end

   return

/*  Errors handling  */
error:
   raiserror @errno @errmsg
   rollback  transaction
end
go
```

Trigger: ti_plgroundeventresourceexplos

Trigger Code

```
/*  Insert trigger "ti_plgroundeventresourceexplos" for table
"PlGroundEventResourceExplosion"  */
create trigger ti_plgroundeventresourceexplos on PlGroundEventResourceExplosion for insert as
begin
   declare
     @numrows  int,
     @numnull  int,
     @errno    int,
     @errmsg   varchar(255)
```

```
   select  @numrows = @@rowcount
   if @numrows = 0
      return


   /*  Parent "PlResource" must exist when inserting a child in
"PlGroundEventResourceExplosion"  */
   if update(resourceId)
   begin
      if (select count(*)
          from   PlResource t1, inserted t2
          where  t1.resourceId = t2.resourceId) != @numrows
         begin
            select @errno  = 30002,
                   @errmsg = 'Parent does not exist in "PlResource". Cannot create child in
"PlGroundEventResourceExplosion".'
            goto error
         end
   end

   return

/*  Errors handling  */
error:
   raiserror @errno @errmsg
   rollback  transaction
end
go
```

Trigger: TrigInsPlMetadataChecks

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlMetadataChecks
ON PlMetadataChecks
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
```

```
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key (pgeId, dataTypeId) has a matching
     primary key in table PlDataTypeReq or PlOutputDataYield */

   IF (SELECT COUNT(*)
      FROM PlDataTypeReq t, inserted i
      WHERE i.ioFlag = 0
      AND t.pgeId = i.pgeId
      AND t.dataTypeId = i.dataTypeId) +
     (SELECT COUNT(*)
      FROM PlOutputDataYield t, inserted i
      WHERE i.ioFlag != 0
      AND t.pgeId = i.pgeId
      AND t.dataTypeId = i.dataTypeId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for (pgeId, dataTypeId) in table PlDataTypeReq or
PlOutputDataYield"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlMetadataChecks

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlMetadataChecks
ON PlMetadataChecks
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN
```

```
   IF UPDATE ( pgeId ) OR UPDATE ( dataTypeId )
   BEGIN

      /* Verify that foreign key (pgeId, dataTypeId) has a matching
       * primary key in table PlDataTypeReq or PlOutputDataYield */

      IF (SELECT COUNT(*)
         FROM PlDataTypeReq t, inserted i
         WHERE i.ioFlag = 0
         AND t.pgeId = i.pgeId
         AND t.dataTypeId = i.dataTypeId) +
        (SELECT COUNT(*)
         FROM PlOutputDataYield t, inserted i
         WHERE i.ioFlag != 0
         AND t.pgeId = i.pgeId
         AND t.dataTypeId = i.dataTypeId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for (pgeId, dataTypeId) in table PlDataTypeReq or
PlOutputDataYield"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlModeRecord

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlModeRecord
ON PlModeRecord
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
```

```
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key availTimesId has a matching
     primary key in table PlDataAvailTimes              */

   IF (SELECT COUNT(*)
      FROM PlDataAvailTimes t, inserted i
      WHERE t.availTimesId = i.availTimesId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for availTimesId in table PlDataAvailTimes"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that foreign key pgeCollectionId has a matching
     primary key in table PlPgeCollection                */

   IF (SELECT COUNT(*)
      FROM PlPgeCollection t, inserted i
      WHERE t.pgeCollectionId = i.pgeCollectionId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeCollectionId in table PlPgeCollection"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlModeRecord

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlModeRecord
ON PlModeRecord
FOR UPDATE
AS
BEGIN
```

```
/* See how many rows were updated */

DECLARE @num_rows int
SELECT @num_rows = @@rowcount
IF @num_rows = 0
   RETURN

IF UPDATE ( availTimesId )
BEGIN

   /* Verify that foreign key availTimesId has a matching
      primary key in table PlDataAvailTimes              */

   IF (SELECT COUNT(*)
       FROM PlDataAvailTimes t, inserted i
       WHERE t.availTimesId = i.availTimesId)
       != @num_rows
   BEGIN
       RAISERROR 31112 "No match for availTimesId in table PlDataAvailTimes"
       ROLLBACK TRANSACTION
       RETURN
   END

END

IF UPDATE ( pgeCollectionId )
BEGIN

   /* Verify that foreign key pgeCollectionId has a matching
      primary key in table PlPgeCollection                */

   IF (SELECT COUNT(*)
       FROM PlPgeCollection t, inserted i
       WHERE t.pgeCollectionId = i.pgeCollectionId)
       != @num_rows
   BEGIN
       RAISERROR 31112 "No match for pgeCollectionId in table PlPgeCollection"
       ROLLBACK TRANSACTION
       RETURN
   END

END

RETURN
```

```
END
go
```

Trigger: TrigInsPlOnDemandPR

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlOnDemandPR
ON PlOnDemandPR
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlProductionRequest     */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for productionRequestId in PlProductionRequest"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlOnDemandPR

Trigger Code

```
/*   */
```

```
CREATE TRIGGER TrigUpdPlOnDemandPR
ON PlOnDemandPR
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( productionRequestId )
   BEGIN

      /* Verify that every foreign key has a matching
       * primary key in table PlProductionRequest     */

      IF (SELECT COUNT(*)
         FROM PlProductionRequest t, inserted i
         WHERE t.productionRequestId = i.productionRequestId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for productionRequestId in PlProductionRequest"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlOnDemandReplanValues

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlOnDemandReplanValues
ON PlOnDemandReplanValues
FOR INSERT
AS
```

```
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key criteriaId has a matching
     primary key in table PlReplanCriteria            */

   IF (SELECT COUNT(*)
      FROM PlReplanCriteria t, inserted i
      WHERE t.criteriaId = i.criteriaId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for criteriaId in table PlReplanCriteria"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlOnDemandReplanValues

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlOnDemandReplanValues
ON PlOnDemandReplanValues
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN
```

```
   /* Verify that foreign key criteriaId has a matching
    * primary key in table PlReplanCriteria            */

   IF UPDATE ( criteriaId )
   BEGIN

      IF (SELECT COUNT(*)
         FROM PlReplanCriteria t, inserted i
         WHERE t.criteriaId = i.criteriaId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for criteriaId in table PlReplanCriteria"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlOutputDataYield

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlOutputDataYield
ON PlOutputDataYield
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that pgeId has a matching
      primary key in table PlPgeMaster */

   IF (SELECT COUNT(*)
```

```
      FROM PlPgeMaster t, inserted i
      WHERE t.pgeId = i.pgeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId in table PlPgeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that dataTypeId has a matching
     primary key in table PlDataTypeMaster */

   IF (SELECT COUNT(*)
      FROM PlDataTypeMaster t, inserted i
      WHERE t.dataTypeId = i.dataTypeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dataTypeId in table PlDataTypeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlOutputDataYield

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlOutputDataYield
ON PlOutputDataYield
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */
```

```
IF @numrows = 0
  RETURN

/* see if the primary key was updated */

IF UPDATE ( pgeId ) OR UPDATE ( dataTypeId )
BEGIN

   /* cascade update to child tables */

   UPDATE PlMetadataChecks
   SET    t.pgeId = i.pgeId,
          t.dataTypeId = i.dataTypeId
   FROM   PlMetadataChecks t,
          inserted i, deleted d
   WHERE  t.pgeId = d.pgeId
   AND    t.dataTypeId = d.dataTypeId
   AND    t.ioFlag = 1

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

END

/* see if the foreign keys were updated */

IF UPDATE ( pgeId )
BEGIN

   /* Verify that pgeId has a matching
    * primary key in table PlPgeMaster */

   IF (SELECT COUNT(*)
      FROM PlPgeMaster t, inserted i
      WHERE t.pgeId = i.pgeId)
      != @numrows
   BEGIN
      RAISERROR 31112 "No match for pgeId in table PlPgeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END
```

```
   END

   IF UPDATE ( dataTypeId )
   BEGIN

      /* Verify that dataTypeId has a matching
       * primary key in table PlDataTypeMaster */

      IF (SELECT COUNT(*)
         FROM PlDataTypeMaster t, inserted i
         WHERE t.dataTypeId = i.dataTypeId)
         != @numrows
      BEGIN
         RAISERROR 31112 "No match for dataTypeId in table PlDataTypeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlOutputDataYield

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlOutputDataYield
ON PlOutputDataYield
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlMetadataChecks */

   DELETE PlMetadataChecks
   FROM   PlMetadataChecks t, deleted d
```

```
   WHERE  t.pgeId = d.pgeId
   AND    t.dataTypeId = d.dataTypeId
   AND    t.ioFlag = 1

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: ti_ploutputfiletype

Trigger Code

```
/*  Insert trigger "ti_ploutputfiletype" for table "PlOutputFileType"  */
create trigger ti_ploutputfiletype on PlOutputFileType for insert as
begin
   declare
      @numrows  int,
      @numnull  int,
      @errno    int,
      @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
      return


   /*  Parent "PlOutputDataYield" must exist when inserting a child in "PlOutputFileType"  */
   if update(pgeId) or
      update(dataTypeId) or
      update(logicalId)
   begin
      if (select count(*)
          from   PlOutputDataYield t1, inserted t2
          where  t1.pgeId = t2.pgeId
           and   t1.dataTypeId = t2.dataTypeId
           and   t1.logicalId = t2.logicalId) != @numrows
         begin
           select @errno  = 30002,
```

```
                @errmsg = 'Parent does not exist in "PlOutputDataYield". Cannot create child in
"PlOutputFileType".'
            goto error
        end
    end

    /*  Parent "PlPgeDetailData" must exist when inserting a child in "PlOutputFileType"  */
    if update(pgeId)
    begin
      if (select count(*)
          from   PlPgeDetailData t1, inserted t2
          where  t1.pgeId = t2.pgeId) != @numrows
        begin
            select @errno  = 30002,
                @errmsg = 'Parent does not exist in "PlPgeDetailData". Cannot create child in
"PlOutputFileType".'
            goto error
        end
    end

    /*  Parent "PlPgeDetailTime" must exist when inserting a child in "PlOutputFileType"  */
    if update(pgeId)
    begin
      if (select count(*)
          from   PlPgeDetailTime t1, inserted t2
          where  t1.pgeId = t2.pgeId) != @numrows
        begin
            select @errno  = 30002,
                @errmsg = 'Parent does not exist in "PlPgeDetailTime". Cannot create child in
"PlOutputFileType".'
            goto error
        end
    end

    /*  Parent "PlPgeMaster" must exist when inserting a child in "PlOutputFileType"  */
    if update(pgeId)
    begin
      if (select count(*)
          from   PlPgeMaster t1, inserted t2
          where  t1.pgeId = t2.pgeId) != @numrows
        begin
            select @errno  = 30002,
                @errmsg = 'Parent does not exist in "PlPgeMaster". Cannot create child in
"PlOutputFileType".'
            goto error
```

```
        end
    end

    /*  Parent "PlPgePerformance" must exist when inserting a child in "PlOutputFileType"  */
    if update(pgeId)
    begin
      if (select count(*)
          from   PlPgePerformance t1, inserted t2
          where  t1.pgeId = t2.pgeId) != @numrows
        begin
          select @errno  = 30002,
                 @errmsg = 'Parent does not exist in "PlPgePerformance". Cannot create child in
"PlOutputFileType".'
          goto error
        end
    end

    /*  Parent "PlRoutineArrival" must exist when inserting a child in "PlOutputFileType"  */
    if update(dataTypeId)
    begin
      if (select count(*)
          from   PlRoutineArrival t1, inserted t2
          where  t1.dataTypeId = t2.dataTypeId) != @numrows
        begin
          select @errno  = 30002,
                 @errmsg = 'Parent does not exist in "PlRoutineArrival". Cannot create child in
"PlOutputFileType".'
          goto error
        end
    end

    /*  Parent "PlTileScheduled" must exist when inserting a child in "PlOutputFileType"  */
    if update(pgeId)
    begin
      if (select count(*)
          from   PlTileScheduled t1, inserted t2
          where  t1.pgeId = t2.pgeId) != @numrows
        begin
          select @errno  = 30002,
                 @errmsg = 'Parent does not exist in "PlTileScheduled". Cannot create child in
"PlOutputFileType".'
          goto error
        end
    end
```

```
   return

/*  Errors handling  */
error:
   raiserror @errno @errmsg
   rollback  transaction
end
go
```

Trigger: ti_plpgecollection

Trigger Code

```
/*  Insert trigger "ti_plpgecollection" for table "PlPgeCollection"  */
create trigger ti_plpgecollection on PlPgeCollection for insert as
begin
   declare
      @numrows  int,
      @numnull  int,
      @errno    int,
      @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
      return


   /*  Parent "PlDataTypeMaster" must exist when inserting a child in "PlPgeCollection"  */
   if update(dataTypeId)
   begin
      select @numnull = (select count(*)
                         from   inserted
                         where  dataTypeId is null)
      if @numnull != @numrows
         if (select count(*)
             from   PlDataTypeMaster t1, inserted t2
             where  t1.dataTypeId = t2.dataTypeId) != @numrows - @numnull
         begin
            select @errno  = 30002,
                   @errmsg = 'Parent does not exist in "PlDataTypeMaster". Cannot create child in
"PlPgeCollection".'
            goto error
         end
   end
```

```
   /*  Parent "PlRoutineArrival" must exist when inserting a child in "PlPgeCollection"  */
   if update(dataTypeId)
   begin
     select @numnull = (select count(*)
                       from   inserted
                       where  dataTypeId is null)
     if @numnull != @numrows
       if (select count(*)
           from   PlRoutineArrival t1, inserted t2
           where  t1.dataTypeId = t2.dataTypeId) != @numrows - @numnull
       begin
         select @errno  = 30002,
                @errmsg = 'Parent does not exist in "PlRoutineArrival". Cannot create child in
"PlPgeCollection".'
         goto error
       end
   end

   return

/*  Errors handling  */
error:
   raiserror @errno @errmsg
   rollback  transaction
end
go
```

Trigger: TrigUpdPlPgeCollection

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlPgeCollection
ON PlPgeCollection
FOR UPDATE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   IF UPDATE ( pgeCollectionId )
   BEGIN
```

```
        UPDATE PlPgeMaster
        SET    t.pgeCollectionId = i.pgeCollectionId
        FROM   PlPgeMaster t,
               inserted i, deleted d
        WHERE  t.pgeCollectionId = d.pgeCollectionId

        IF @@transtate = 2 /* previous stmt aborted */
        BEGIN
           ROLLBACK TRANSACTION
              RETURN
        END

        UPDATE PlModeRecord
        SET    t.pgeCollectionId = i.pgeCollectionId
        FROM   PlModeRecord t,
               inserted i, deleted d
        WHERE  t.pgeCollectionId = d.pgeCollectionId

        IF @@transtate = 2 /* previous stmt aborted */
        BEGIN
           ROLLBACK TRANSACTION
              RETURN
        END

    END

    RETURN

END
go
```

Trigger: TrigDelPlPgeCollection

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlPgeCollection
ON PlPgeCollection
FOR DELETE
AS
BEGIN

    /* if no rows were deleted, then return */
```

```
   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlModeRecord */

   DELETE PlModeRecord
   FROM   PlModeRecord t, deleted d
   WHERE  t.pgeCollectionId = d.pgeCollectionId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: ti_plpgedetaildata

Trigger Code

```
/*  Insert trigger "ti_plpgedetaildata" for table "PlPgeDetailData"  */
create trigger ti_plpgedetaildata on PlPgeDetailData for insert as
begin
   declare
     @numrows  int,
     @numnull  int,
     @errno    int,
     @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
     return


   /*  Parent "DpPrInputData" must exist when inserting a child in "PlPgeDetailData"  */
   if update(inputDataId)
   begin
     if (select count(*)
         from   DpPrInputData t1, inserted t2
         where  t1.inputDataId = t2.inputDataId) != @numrows
        begin
```

```
            select @errno  = 30002,
                  @errmsg = 'Parent does not exist in "DpPrInputData". Cannot create child in
"PlPgeDetailData".'
            goto error
         end
    end

    return

/*  Errors handling  */
error:
    raiserror @errno @errmsg
    rollback  transaction
end
go
```

Trigger: TrigInsPlPgeDetailParameters

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPgeDetailParameters
ON PlPgeDetailParameters
FOR INSERT
AS
BEGIN

    /* See how many rows were inserted */

    DECLARE @num_rows int
    SELECT @num_rows = @@rowcount
    IF @num_rows = 0
        RETURN

    /* Verify that foreign key has a matching
       primary key in table PlPgeMaster       */

    IF (SELECT COUNT(*)
        FROM PlPgeMaster t, inserted i
        WHERE t.pgeId = i.pgeId)
        != @num_rows
    BEGIN
        RAISERROR 31111 "No match for pgeId in table PlPgeMaster"
        ROLLBACK TRANSACTION
        RETURN
```

```
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPgeDetailParameters

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigUpdPlPgeDetailParameters
ON PlPgeDetailParameters
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( pgeParameterLogicalId )
   BEGIN

      /* cascade update to child tables */

      UPDATE PlUserParameterConstraints
      SET t.pgeParameterLogicalId = i.pgeParameterLogicalId
      FROM PlUserParameterConstraints t,
          inserted i, deleted d
      WHERE t.pgeParameterLogicalId = d.pgeParameterLogicalId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
```

```
      END

      UPDATE PlPrParameter
      SET t.pgeParameterLogicalId = i.pgeParameterLogicalId
      FROM PlPrParameter t, inserted i, deleted d
      WHERE t.pgeParameterLogicalId = d.pgeParameterLogicalId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   /* see if the foreign key was updated */

   IF UPDATE ( pgeId )
   BEGIN

      /* Verify that foreign key has a matching
       * primary key in table PlPgeMaster       */

      IF (SELECT COUNT(*)
         FROM PlPgeMaster t, inserted i
         WHERE t.pgeId = i.pgeId)
         != @numrows
      BEGIN
         RAISERROR 31112 "No match for pgeId in table PlPgeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlPgeDetailParameters

Trigger Code

```
/*                                    */
```

```
CREATE TRIGGER TrigDelPlPgeDetailParameters
ON PlPgeDetailParameters
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows PlUserParameterConstraints */

   DELETE PlUserParameterConstraints
   FROM PlUserParameterConstraints t, deleted d
   WHERE t.pgeParameterLogicalId = d.pgeParameterLogicalId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlPrParameter */

   DELETE PlPrParameter
   FROM PlPrParameter t, deleted d
   WHERE t.pgeParameterLogicalId = d.pgeParameterLogicalId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlPgeDetailTime

Trigger Code

```
/*   */
```

```
CREATE TRIGGER TrigInsPlPgeDetailTime
ON PlPgeDetailTime
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlPgeMaster            */

   IF (SELECT COUNT(*)
      FROM PlPgeMaster t, inserted i
      WHERE t.pgeId = i.pgeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId in table PlPgeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPgeDetailTime

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlPgeDetailTime
ON PlPgeDetailTime
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
```

```
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( pgeId )
   BEGIN

      /* Verify that foreign key has a matching
       * primary key in table PlPgeMaster       */

      IF (SELECT COUNT(*)
         FROM PlPgeMaster t, inserted i
         WHERE t.pgeId = i.pgeId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for pgeId in table PlPgeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: ti_plpgemaster

Trigger Code

```
/*  Insert trigger "ti_plpgemaster" for table "PlPgeMaster"  */
create trigger ti_plpgemaster on PlPgeMaster for insert as
begin
   declare
      @numrows  int,
      @numnull  int,
      @errno    int,
      @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
      return
```

```
   /*  Parent "PlPgeCollection" must exist when inserting a child in "PlPgeMaster"  */
   if update(pgeCollectionId)
   begin
     select @numnull = (select count(*)
                       from   inserted
                       where  pgeCollectionId is null)
     if @numnull != @numrows
       if (select count(*)
           from   PlPgeCollection t1, inserted t2
           where  t1.pgeCollectionId = t2.pgeCollectionId) != @numrows - @numnull
       begin
         select @errno  = 30002,
                @errmsg = 'Parent does not exist in "PlPgeCollection". Cannot create child in
"PlPgeMaster".'
         goto error
       end
   end

   /*  Parent "PlResourceRequirement" must exist when inserting a child in "PlPgeMaster"  */
   if update(sswId)
   begin
     select @numnull = (select count(*)
                       from   inserted
                       where  sswId is null)
     if @numnull != @numrows
       if (select count(*)
           from   PlResourceRequirement t1, inserted t2
           where  t1.sswId = t2.sswId) != @numrows - @numnull
       begin
         select @errno  = 30002,
                @errmsg = 'Parent does not exist in "PlResourceRequirement". Cannot create child
in "PlPgeMaster".'
         goto error
       end
   end

   return

/*  Errors handling  */
error:
   raiserror @errno @errmsg
   rollback  transaction
end
go
```

Trigger: TrigUpdPlPgeMaster

Trigger Code

```
CREATE TRIGGER TrigUpdPlPgeMaster
ON PlPgeMaster
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( pgeId )
   BEGIN

      /* cascade update to child tables */

      UPDATE PlOutputDataYield
      SET    t.pgeId = i.pgeId
      FROM PlOutputDataYield t,
          inserted i, deleted d
      WHERE t.pgeId = d.pgeId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlDataTypeReq
      SET t.pgeId = i.pgeId
      FROM PlDataTypeReq t,
          inserted i, deleted d
      WHERE t.pgeId=d.pgeId
```

```
IF @@transtate = 2 /* previous stmt aborted */
BEGIN
   ROLLBACK TRANSACTION
      RETURN
END

UPDATE PlTileScheduled
SET    t.pgeId = i.pgeId
FROM   PlTileScheduled t, inserted i, deleted d
WHERE  t.pgeId = d.pgeId

IF @@transtate = 2 /* previous stmt aborted */
BEGIN
   ROLLBACK TRANSACTION
      RETURN
END

UPDATE PlPgeDetailParameters
SET    t.pgeId = i.pgeId
FROM PlPgeDetailParameters t, inserted i, deleted d
WHERE t.pgeId = d.pgeId

IF @@transtate = 2 /* previous stmt aborted */
BEGIN
   ROLLBACK TRANSACTION
      RETURN
END

UPDATE PlPgeDetailTime
SET    t.pgeId = i.pgeId
FROM PlPgeDetailTime t, inserted i, deleted d
WHERE t.pgeId = d.pgeId

IF @@transtate = 2 /* previous stmt aborted */
BEGIN
   ROLLBACK TRANSACTION
      RETURN
END

UPDATE PlErrorAction
SET    t.pgeId = i.pgeId
FROM   PlErrorAction t, inserted i, deleted d
WHERE  t.pgeId = d.pgeId
```

```
      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlProductionRequest
      SET    t.pgeIdentifier = i.pgeId
      FROM   PlProductionRequest t,
             inserted i, deleted d
      WHERE  t.pgeIdentifier = d.pgeId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlDataProcessingRequest
      SET    t.pgeId = i.pgeId
      FROM   PlDataProcessingRequest t,
             inserted i, deleted d
      WHERE  t.pgeId = d.pgeId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlPgeMaster

Trigger Code

```
CREATE TRIGGER TrigDelPlPgeMaster
ON PlPgeMaster
FOR DELETE
AS
```

```
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlOutputDataYield */

   DELETE PlOutputDataYield
   FROM   PlOutputDataYield t, deleted d
   WHERE  t.pgeId = d.pgeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlDataTypeReq */

   DELETE PlDataTypeReq
   FROM   PlDataTypeReq t, deleted d
   WHERE  t.pgeId = d.pgeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlTileScheduled */

   DELETE PlTileScheduled
   FROM PlTileScheduled t, deleted d
   WHERE t.pgeId = d.pgeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlPgeDetailTime */
```

```
DELETE PlPgeDetailTime
FROM   PlPgeDetailTime t, deleted d
WHERE  t.pgeId = d.pgeId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlErrorAction */

DELETE PlErrorAction
FROM   PlErrorAction t, deleted d
WHERE  t.pgeId = d.pgeId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlPgeDetailParameters */

DELETE PlPgeDetailParameters
FROM   PlPgeDetailParameters t, deleted d
WHERE  t.pgeId = d.pgeId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlProductionRequest */

DELETE PlProductionRequest
FROM   PlProductionRequest t, deleted d
WHERE  t.pgeIdentifier = d.pgeId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END
```

```
   /* delete matching rows in PlDataProcessingRequest */

   DELETE PlDataProcessingRequest
   FROM   PlDataProcessingRequest t, deleted d
   WHERE  t.pgeId = d.pgeId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlPgePerformance

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPgePerformance
ON PlPgePerformance
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlPgeMaster            */

   IF (SELECT COUNT(*)
      FROM PlPgeMaster t, inserted i
      WHERE t.pgeId = i.pgeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId in PlPgeMaster"
```

```
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPgePerformance

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlPgePerformance
ON PlPgePerformance
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( pgeId )
   BEGIN

      /* Verify that every foreign key has a matching
         primary key in table PlPgeMaster            */

      IF (SELECT COUNT(*)
          FROM PlPgeMaster t, inserted i
          WHERE t.pgeId = i.pgeId)
          != @num_rows
      BEGIN
          RAISERROR 31112 "No match for pgeId in PlPgeMaster"
          ROLLBACK TRANSACTION
          RETURN
      END

   END
```

```
   RETURN

END
go
```

Trigger: TrigInsPlPgePriority

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPgePriority
ON PlPgePriority
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key strategyId has a matching
     primary key in table PlProductionStrategy         */

   IF (SELECT COUNT(*)
      FROM PlProductionStrategy t, inserted i
      WHERE t.strategyId = i.strategyId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for strategyId in table PlProductionStrategy"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPgePriority

Trigger Code

```
/*   */
```

```
CREATE TRIGGER TrigUpdPlPgePriority
ON PlPgePriority
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( strategyId )
   BEGIN

      /* Verify that foreign key strategyId has a matching
       * primary key in table PlProductionStrategy        */

      IF (SELECT COUNT(*)
         FROM PlProductionStrategy t, inserted i
         WHERE t.strategyId = i.strategyId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for strategyId in table PlProductionStrategy"
         ROLLBACK TRANSACTION
      RETURN

      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlPlans

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPlans
ON PlPlans
FOR INSERT
AS
```

```
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key strategyId has a matching
     primary key in table PlProductionStrategy        */

   IF (SELECT COUNT(*)
      FROM PlProductionStrategy t, inserted i
      WHERE t.strategyId = i.strategyId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for strategyId in table PlProductionStrategy"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPlans

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlPlans
ON PlPlans
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
```

```
  RETURN

/* see if the primary key was updated */

IF UPDATE ( planId )
BEGIN

   /* cascade update to child tables */

   UPDATE PlActivePlan
   SET t.planId = i.planId
   FROM PlActivePlan t,
       inserted i, deleted d
   WHERE t.planId = d.planId

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   UPDATE PlPlanMetadata
   SET t.planId = i.planId
   FROM PlPlanMetadata t, inserted i, deleted d
   WHERE t.planId = d.planId

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   UPDATE PlPrVsPlan
   SET t.planId = i.planId
   FROM PlPrVsPlan t, inserted i, deleted d
   WHERE t.planId = d.planId

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

END
```

```
   /* see if the foreign key was updated */

   IF UPDATE ( strategyId )
   BEGIN

      /* Verify that foreign key strategyId has a matching
         primary key in table PlProductionStrategy        */

      IF (SELECT COUNT(*)
         FROM PlProductionStrategy t, inserted i
         WHERE t.strategyId = i.strategyId) != @numrows
      BEGIN
         RAISERROR 31112 "No match for strategyId in table PlProductionStrategy"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlPlans

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlPlans
ON PlPlans
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
      RETURN

   /* delete matching rows in PlActivePlan */

   DELETE PlActivePlan
   FROM   PlActivePlan t, deleted d
   WHERE  t.planId = d.planId
```

```
   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlPlanMetadata */

   DELETE PlPlanMetadata
   FROM   PlPlanMetadata t, deleted d
   WHERE  t.planId = d.planId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlPrVsPlan */

   DELETE PlPrVsPlan
   FROM   PlPrVsPlan t, deleted d
   WHERE  t.planId = d.planId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPrCollection

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigUpdPlPrCollection
ON PlPrCollection
FOR UPDATE
AS
```

```
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   IF UPDATE ( prCollectionId )
   BEGIN

      UPDATE PlProductionRequest
      SET    t.prCollectionId = i.prCollectionId
      FROM   PlProductionRequest t,
             inserted i, deleted d
      WHERE  t.prCollectionId = d.prCollectionId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlPrCollectionExplosion
      SET    t.prCollectionId = i.prCollectionId
      FROM   PlPrCollectionExplosion t,
             inserted i, deleted d
      WHERE  t.prCollectionId = d.prCollectionId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlPrCollection

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlPrCollection
ON PlPrCollection
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlPrCollectionExplosion */

   DELETE PlPrCollectionExplosion
   FROM   PlPrCollectionExplosion t, deleted d
   WHERE  t.prCollectionId = d.prCollectionId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlPrCollectionExplosion

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPrCollectionExplosion
ON PlPrCollectionExplosion
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
```

```
      RETURN

   /* Verify that foreign key prCollectionId has a matching
     primary key in table PlPrCollection                    */

   IF (SELECT COUNT(*)
      FROM PlPrCollection t, inserted i
      WHERE t.prCollectionId = i.prCollectionId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for prCollectionId in table PlPrCollection"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPrCollectionExplosion

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlPrCollectionExplosion
ON PlPrCollectionExplosion
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( prCollectionId )
   BEGIN

      /* Verify that foreign key prCollectionId has a matching
       * primary key in table PlPrCollection                    */

      IF (SELECT COUNT(*)
```

```
        FROM PlPrCollection t, inserted i
        WHERE t.prCollectionId = i.prCollectionId)
        != @num_rows
      BEGIN
        RAISERROR 31112 "No match for prCollectionId in table PlPrCollection"
        ROLLBACK TRANSACTION
        RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlPrMetadataValue

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPrMetadataValue
ON PlPrMetadataValue
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key productionRequestId has a matching
      primary key in table PlProductionRequest                    */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for productionRequestId in table PlPrMetadataValue"
      ROLLBACK TRANSACTION
      RETURN
```

```
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPrMetadataValue

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlPrMetadataValue
ON PlPrMetadataValue
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( productionRequestId )
   BEGIN

      /* Verify that foreign key productionRequestId has a matching
         primary key in table PlProductionRequest                    */

      IF (SELECT COUNT(*)
         FROM PlProductionRequest t, inserted i
         WHERE t.productionRequestId = i.productionRequestId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for productionRequestId in table PlPrMetadataValue"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN
```

```
END
go
```

Trigger: TrigInsPlProductionRequest

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlProductionRequest
ON PlProductionRequest
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that pgeId has a matching
     primary key in table PlPgeMaster  */

   IF (SELECT COUNT(*)
      FROM PlPgeMaster t, inserted i
      WHERE t.pgeId = i.pgeIdentifier)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeIdentifier in table PlPgeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlProductionRequest

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigUpdPlProductionRequest
```

```
ON PlProductionRequest
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( productionRequestId )
   BEGIN

      /* cascade update to child tables */

      UPDATE PlPrVsPlan
      SET t.productionRequestId = i.productionRequestId
      FROM PlPrVsPlan t, inserted i, deleted d
      WHERE t.productionRequestId=d.productionRequestId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlAlternateInputValues
      SET t.productionRequestId = i.productionRequestId
      FROM PlAlternateInputValues t,
          inserted i, deleted d
      WHERE t.productionRequestId=d.productionRequestId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END
```

```
    UPDATE PlDataProcessingRequest
    SET t.productionRequestId = i.productionRequestId
    FROM PlDataProcessingRequest t,
        inserted i, deleted d
    WHERE t.productionRequestId=d.productionRequestId

    IF @@transtate = 2 /* previous stmt aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

    UPDATE PlPrMetadataValue
    SET t.productionRequestId = i.productionRequestId
    FROM PlPrMetadataValue t, inserted i, deleted d
    WHERE t.productionRequestId=d.productionRequestId

    IF @@transtate = 2 /* previous stmt aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

    UPDATE PlPrParameter
    SET t.productionRequestId = i.productionRequestId
    FROM PlPrParameter t, inserted i, deleted d
    WHERE t.productionRequestId=d.productionRequestId

    IF @@transtate = 2 /* previous stmt aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

END

/* see if foreign keys were updated */

IF UPDATE ( pgeIdentifier )
BEGIN

    /* Verify that pgeIdentifier has a matching
     * primary key in table PlPgeMaster */

    IF (SELECT COUNT(*)
```

```
        FROM PlPgeMaster t, inserted i
        WHERE t.pgeId = i.pgeIdentifier)
        != @numrows
      BEGIN
        RAISERROR 31112 "No match for pgeIdentifier in table PlPgeMaster"
        ROLLBACK TRANSACTION
        RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlProductionRequest

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlProductionRequest
ON PlProductionRequest
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in table PlPrVsPlan */

   DELETE PlPrVsPlan FROM PlPrVsPlan t, deleted d
   WHERE t.productionRequestId = d.productionRequestId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlAlternateInputValues */
```

```
DELETE PlAlternateInputValues
FROM PlAlternateInputValues t, deleted d
WHERE t.productionRequestId = d.productionRequestId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlDataProcessingRequest */

DELETE PlDataProcessingRequest
FROM PlDataProcessingRequest t, deleted d
WHERE t.productionRequestId = d.productionRequestId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlPrMetadataValue */

DELETE PlPrMetadataValue
FROM PlPrMetadataValue t, deleted d
WHERE t.productionRequestId = d.productionRequestId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END

/* delete matching rows in PlPrParameter */

DELETE PlPrParameter
FROM PlPrParameter t, deleted d
WHERE t.productionRequestId = d.productionRequestId

IF @@transtate = 2  /* previous statement aborted */
BEGIN
   ROLLBACK TRANSACTION  /* undo all deletes */
   RETURN
END
```

```
   RETURN

END
go
```

Trigger: TrigUpdPlProductionStrategy

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigUpdPlProductionStrategy
ON PlProductionStrategy
FOR UPDATE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* see if the primary key was updated; if so, cascade the update
     to dependent tables */

   IF UPDATE ( strategyId )
   BEGIN

      UPDATE PlPgePriority
      SET    t.strategyId = i.strategyId
      FROM   PlPgePriority t,
            inserted i, deleted d
      WHERE  t.strategyId = d.strategyId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlUserPriority
      SET    t.strategyId = i.strategyId
      FROM   PlUserPriority t,
            inserted i, deleted d
      WHERE  t.strategyId = d.strategyId
```

```
    IF @@transtate = 2 /* previous stmt aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

    UPDATE PlPRPriority
    SET    t.strategyId = i.strategyId
    FROM   PlPRPriority t,
          inserted i, deleted d
    WHERE  t.strategyId = d.strategyId

    IF @@transtate = 2 /* previous stmt aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

    UPDATE PlPgePriority
    SET    t.strategyId = i.strategyId
    FROM   PlPgePriority t,
          inserted i, deleted d
    WHERE  t.strategyId = d.strategyId

    IF @@transtate = 2 /* previous stmt aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

    UPDATE PlPlans
    SET    t.strategyId = i.strategyId
    FROM   PlPlans t,
          inserted i, deleted d
    WHERE  t.strategyId = d.strategyId

    IF @@transtate = 2 /* previous stmt aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

END
```

```
   RETURN

END
go
```

Trigger: TrigDelPlProductionStrategy

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlProductionStrategy
ON PlProductionStrategy
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
      RETURN

   /* delete matching rows in PlPgePriority */

   DELETE PlPgePriority
   FROM   PlPgePriority t, deleted d
   WHERE  t.strategyId = d.strategyId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlUserPriority */

   DELETE PlUserPriority
   FROM   PlUserPriority t, deleted d
   WHERE  t.strategyId = d.strategyId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END
```

```
   /* delete matching rows in PlPRPriority */

   DELETE PlPRPriority
   FROM   PlPRPriority t, deleted d
   WHERE  t.strategyId = d.strategyId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlPlans */

   DELETE PlPlans
   FROM   PlPlans t, deleted d
   WHERE  t.strategyId = d.strategyId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlPrParameter

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPrParameter
ON PlPrParameter
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
```

```
      RETURN

   /* Verify that foreign key productionRequestId has a matching
      primary key in table PlProductionRequest                    */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for productionRequestId in table PlProductionRequest"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that foreign key (pgeParameterLogicalId, pgeId) has a matching
      primary key in table PlPgeDetailParameters                          */

   IF (SELECT COUNT(*)
      FROM PlPgeDetailParameters t, inserted i
      WHERE t.pgeParameterLogicalId = i.pgeParameterLogicalId
      AND   t.pgeId = i.pgeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeParameterLogicalId/pgeId in table
PlPgeDetailParameters"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPrParameter

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlPrParameter
ON PlPrParameter
FOR UPDATE
AS
BEGIN
```

```
/* See how many rows were updated */

DECLARE @num_rows int
SELECT @num_rows = @@rowcount
IF @num_rows = 0
   RETURN

IF UPDATE ( productionRequestId )
BEGIN

   /* Verify that foreign key productionRequestId has a matching
    * primary key in table PlProductionRequest                    */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31112 "No match for productionRequestId in table PlProductionRequest"
      ROLLBACK TRANSACTION
      RETURN
   END

END

IF UPDATE ( pgeParameterLogicalId ) OR UPDATE ( pgeId )
BEGIN

   /* Verify that foreign key (pgeParameterLogicalId, pgeId) has a matching
    * primary key in table PlPgeDetailParameters    */

   IF (SELECT COUNT(*)
      FROM PlPgeDetailParameters t, inserted i
      WHERE t.pgeParameterLogicalId = i.pgeParameterLogicalId
      AND   t.pgeId = i.pgeId)
      != @num_rows
   BEGIN
      RAISERROR 31112 "No match for pgeParameterLogicalId/pgeId in table
PlPgeDetailParameters"
      ROLLBACK TRANSACTION
      RETURN
   END

END
```

```
   RETURN

END
go
```

Trigger: TrigInsPlPRPriority

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPRPriority
ON PlPRPriority
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key strategyId has a matching
     primary key in table PlProductionStrategy         */

   IF (SELECT COUNT(*)
      FROM PlProductionStrategy t, inserted i
      WHERE t.strategyId = i.strategyId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for strategyId in table PlProductionStrategy"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPRPriority

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlPRPriority
ON PlPRPriority
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( strategyId )
   BEGIN

      /* Verify that foreign key strategyId has a matching
         primary key in table PlProductionStrategy        */

      IF (SELECT COUNT(*)
         FROM PlProductionStrategy t, inserted i
         WHERE t.strategyId = i.strategyId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for strategyId in table PlProductionStrategy"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlPrVsPlan

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlPrVsPlan
ON PlPrVsPlan
FOR INSERT
AS
```

```
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key productionRequestId has a matching
     primary key in table PlProductionRequest                    */

   IF (SELECT COUNT(*)
      FROM PlProductionRequest t, inserted i
      WHERE t.productionRequestId = i.productionRequestId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for productionRequestId in table PlProductionRequest"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that foreign key planId has a matching
     primary key in table PlPlans                    */

   IF (SELECT COUNT(*)
      FROM PlPlans t, inserted i
      WHERE t.planId = i.planId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for planId in table PlPlans"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlPrVsPlan

Trigger Code

```
/*   */
```

```
CREATE TRIGGER TrigUpdPlPrVsPlan
ON PlPrVsPlan
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( productionRequestId )
   BEGIN

      /* Verify that foreign key productionRequestId has a matching
       * primary key in table PlProductionRequest                    */

      IF (SELECT COUNT(*)
         FROM PlProductionRequest t, inserted i
         WHERE t.productionRequestId = i.productionRequestId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for productionRequestId in table PlProductionRequest"
         ROLLBACK TRANSACTION
         RETURN
      END

   END


   IF UPDATE ( planId )
   BEGIN

      /* Verify that foreign key planId has a matching
       * primary key in table PlPlans                 */

      IF (SELECT COUNT(*)
         FROM PlPlans t, inserted i
         WHERE t.planId = i.planId)
         != @num_rows
      BEGIN
         RAISERROR 31112 "No match for planId in table PlPlans"
         ROLLBACK TRANSACTION
```

```
        RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlReplanCriteria

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlReplanCriteria
ON PlReplanCriteria
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlDasDelta */

   DELETE PlDasDelta
   FROM   PlDasDelta t, deleted d
   WHERE  t.criteriaId = d.criteriaId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlOnDemandReplanValues */

   DELETE PlOnDemandReplanValues
   FROM   PlOnDemandReplanValues t, deleted d
   WHERE  t.criteriaId = d.criteriaId

   IF @@transtate = 2  /* previous statement aborted */
```

```
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlRescUseThresh

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlRescUseThresh
ON PlRescUseThresh
FOR INSERT, UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that resourceType is between 0 and 3 */

   IF (SELECT COUNT(*)
      FROM inserted i
      WHERE i.resourceType BETWEEN 0 AND 3) != @num_rows
   BEGIN
      RAISERROR 31111 "Attempt to insert invalid resourceType into PlRescUseThresh"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that userType is either DSS_USER or else DPS_USER */

   IF (SELECT COUNT(*)
      FROM inserted i
      WHERE (i.userType = "DSS_USER") OR (i.userType = "DPS_USER") ) != @num_rows
   BEGIN
```

```
      RAISERROR 31111 "Attempt to insert invalid userType into PlRescUseThresh"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that, if resourceType is 3 (cumulative), then threshold is between 0 and 1 */

   IF (SELECT COUNT(*)
      FROM inserted i
      WHERE i.resourceType = 3
      AND   ((i.threshold < 0.0) OR (i.threshold > 1.0)) ) > 0
   BEGIN
      RAISERROR 31111 "Attempt to insert invalid threshold for Cumulative into
PlRescUseThresh"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlRescUseThresh

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlRescUseThresh
ON PlRescUseThresh
FOR INSERT, UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that resourceType is between 0 and 3 */

   IF (SELECT COUNT(*)
      FROM inserted i
```

```
      WHERE i.resourceType BETWEEN 0 AND 3) != @num_rows
   BEGIN
      RAISERROR 31111 "Attempt to insert invalid resourceType into PlRescUseThresh"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that userType is either DSS_USER or else DPS_USER */

   IF (SELECT COUNT(*)
      FROM inserted i
      WHERE (i.userType = "DSS_USER") OR (i.userType = "DPS_USER") ) != @num_rows
   BEGIN
      RAISERROR 31111 "Attempt to insert invalid userType into PlRescUseThresh"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that, if resourceType is 3 (cumulative), then threshold is between 0 and 1 */

   IF (SELECT COUNT(*)
      FROM inserted i
      WHERE i.resourceType = 3
      AND   ((i.threshold < 0.0) OR (i.threshold > 1.0)) ) > 0
   BEGIN
      RAISERROR 31111 "Attempt to insert invalid threshold for Cumulative into
PlRescUseThresh"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: ti_plresource

Trigger Code

```
/*  Insert trigger "ti_plresource" for table "PlResource"  */
create trigger ti_plresource on PlResource for insert as
begin
   declare
      @numrows  int,
```

```
      @numnull  int,
      @errno    int,
      @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
      return


   /*  Parent "RpActivityType" must exist when inserting a child in "PlResource"  */
   if update(activityTypeId)
   begin
      select @numnull = (select count(*)
                         from   inserted
                         where  activityTypeId is null)
      if @numnull != @numrows
         if (select count(*)
             from   RpActivityType t1, inserted t2
             where  t1.activityTypeId = t2.activityTypeId) != @numrows - @numnull
         begin
            select @errno  = 30002,
                   @errmsg = 'Parent does not exist in "RpActivityType". Cannot create child in
"PlResource".'
            goto error
         end
   end

   return

/*  Errors handling  */
error:
   raiserror @errno @errmsg
   rollback  transaction
end
go
```

Trigger: TrigUpdPlResource

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlResource
ON PlResource
FOR UPDATE
AS
```

```
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   IF UPDATE ( resourceId )
   BEGIN

      UPDATE PlRscString
      SET    t.stringId = i.resourceId
      FROM   PlRscString t,
             inserted i, deleted d
      WHERE  t.stringId = d.resourceId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlRscComputer
      SET    t.computerId = i.resourceId
      FROM   PlRscComputer t,
             inserted i, deleted d
      WHERE  t.computerId = d.resourceId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlRscDiskPartition
      SET    t.diskPartitionId = i.resourceId
      FROM   PlRscDiskPartition t,
             inserted i, deleted d
      WHERE  t.diskPartitionId = d.resourceId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END
```

```
   END

   RETURN

END
go
```

Trigger: TrigDelPlResource

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlResource
ON PlResource
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in table PlRscString */

   DELETE PlRscString FROM PlRscString s,
                        deleted d
   WHERE s.stringId = d.resourceId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   /* delete matching rows in table PlRscComputer */

   DELETE PlRscComputer FROM PlRscComputer c,
                         deleted d
   WHERE c.computerId = d.resourceId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
```

```
        RETURN
    END

    /* delete matching rows in PlRscDiskPartition */

    DELETE PlRscDiskPartition
    FROM PlRscDiskPartition p, deleted d
    WHERE p.diskPartitionId = d.resourceId

    IF @@transtate = 2 /* previous statement aborted */
    BEGIN
       ROLLBACK TRANSACTION
          RETURN
    END

    RETURN

END
go
```

Trigger: TrigInsPlResourceRequirement

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlResourceRequirement
ON PlResourceRequirement
FOR INSERT
AS
BEGIN

    /* See how many rows were inserted */

    DECLARE @num_rows int
    SELECT @num_rows = @@rowcount
    IF @num_rows = 0
       RETURN

    /* Verify that column string, if not "" or null, has a matching
       value in column stringName in table PlRscString */

    IF (SELECT COUNT(*)
       FROM PlRscString t, inserted i
       WHERE t.stringName = i.string) +
       (SELECT COUNT(*) FROM inserted i WHERE i.string = "") +
```

```
      (SELECT COUNT(*) FROM inserted i WHERE i.string is null)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for string in table PlRscString"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that column computer, if not "" or null, has a matching
     value in column computerName in table PlRscComputer */

   IF (SELECT COUNT(*)
      FROM PlRscComputer t, inserted i
      WHERE t.computerName = i.computer) +
      (SELECT COUNT(*) FROM inserted i WHERE i.computer = "") +
      (SELECT COUNT(*) FROM inserted i WHERE i.computer is null)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for computer in PlRscComputer"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlResourceRequirement

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlResourceRequirement
ON PlResourceRequirement
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN
```

```
IF UPDATE ( string )
BEGIN

   /* Verify that column string has a matching
    * value in column stringName in table PlRscString  */

   IF ((SELECT COUNT(*)
       FROM PlRscString t, inserted i
       WHERE t.stringName = i.string) +
      (SELECT COUNT(*)
       FROM inserted i
       WHERE i.string = "") +
      (SELECT COUNT(*)
       FROM inserted i
       WHERE i.string is null))
       != @num_rows
   BEGIN
       RAISERROR 31112 "No match for stringName in table PlRscString"
       ROLLBACK TRANSACTION
       RETURN
   END

END

IF UPDATE ( computer )
BEGIN

   /* Verify that column computer has a matching
    * value in column computerName in table PlRscComputer */

   IF ((SELECT COUNT(*)
       FROM PlRscComputer t, inserted i
       WHERE t.computerName = i.computer) +
      (SELECT COUNT(*)
       FROM inserted i
       WHERE i.computer = "") +
      (SELECT COUNT(*)
       FROM inserted i
       WHERE i.computer is null))
       != @num_rows
   BEGIN
       RAISERROR 31112 "No match for computerName in PlRscComputer"
       ROLLBACK TRANSACTION
       RETURN
```

```
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlResourceRequirement

Trigger Code

```
/*                                    */
CREATE TRIGGER TrigDelPlResourceRequirement
ON PlResourceRequirement
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlPgeMaster */

   DELETE PlPgeMaster
   FROM   PlPgeMaster t, deleted d
   WHERE  t.sswId = d.sswId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlDataProcessingRequest */

   DELETE PlDataProcessingRequest
   FROM   PlDataProcessingRequest t, deleted d
   WHERE  t.sswId = d.sswId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
```

```
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in PlGroundEventAllocation */
/*
   DELETE PlGroundEventAllocation
   FROM   PlGroundEventAllocation t, deleted d
   WHERE  t.sswId = d.sswId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END
*/
   /* delete matching rows in DpPrPgeExitDependency */

   DELETE DpPrPgeExitDependency
   FROM   DpPrPgeExitDependency t, deleted d
   WHERE  t.pgeSswId = d.sswId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   /* delete matching rows in DpPrPgeExitMessage */

   DELETE DpPrPgeExitMessage
   FROM   DpPrPgeExitMessage t, deleted d
   WHERE  t.pgeSswId = d.sswId

   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlRoutineArrival

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlRoutineArrival
ON PlRoutineArrival
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key dataTypeId has a matching
     primary key in table PlDataSourceMaster          */

   IF (SELECT COUNT(*)
      FROM PlDataSourceMaster t, inserted i
      WHERE t.dataTypeId = i.dataTypeId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for dataTypeId in table PlDataSourceMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlRoutineArrival

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlRoutineArrival
ON PlRoutineArrival
FOR UPDATE
AS
```

```
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( dataTypeId )
   BEGIN

      /* Verify that foreign key dataTypeId has a matching
       * primary key in table PlDataSourceMaster          */

      IF (SELECT COUNT(*)
         FROM PlDataSourceMaster t, inserted i
         WHERE t.dataTypeId = i.dataTypeId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for dataTypeId in table PlDataSourceMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlRscComputer

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlRscComputer
ON PlRscComputer
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
```

```
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key computerId has a matching
     primary key in table PlResource          */

   IF (SELECT COUNT(*)
      FROM PlResource t, inserted i
      WHERE t.resourceId = i.computerId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for computerId in table PlResource"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that foreign key stringId has a matching
     primary key in table PlRscString              */

   --IF (SELECT COUNT(*)
      --FROM PlRscString t, inserted i
      --WHERE t.stringId = i.stringId) != @num_rows
   --BEGIN
      --RAISERROR 31111 "No match for stringId in table PlRscString"
      --ROLLBACK TRANSACTION
      --RETURN
   --END

   RETURN

END
go
```

Trigger: TrigUpdPlRscComputer

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlRscComputer
ON PlRscComputer
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */
```

```
BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( computerId )
   BEGIN

      /* cascade update to child tables */

      UPDATE PlRscDiskPartition
      SET t.computerId = i.computerId
      FROM PlRscDiskPartition t, inserted i, deleted d
      WHERE t.computerId = d.computerId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   /* see if foreign keys were updated */

   IF UPDATE ( computerId )
   BEGIN

      /* Verify that computerId has a matching
       * primary key in table PlResource       */

      IF (SELECT COUNT(*)
         FROM PlResource t, inserted i
         WHERE t.resourceId = i.computerId)
         != @numrows
      BEGIN
         RAISERROR 31112 "No match for computerId in table PlResource"
         ROLLBACK TRANSACTION
         RETURN
```

```
    END

  END

  IF UPDATE ( stringId )
  BEGIN

    /* Verify that stringId has a matching
     * primary key in table PlRscString     */

    IF ((SELECT COUNT(*)
        FROM PlRscString t, inserted i
        WHERE t.stringId = i.stringId) +
        (SELECT COUNT(*)
         FROM inserted i
         WHERE i.stringId is null) +
        (SELECT COUNT(*)
         FROM inserted i
         WHERE i.stringId = 0))
        != @numrows
    BEGIN
        RAISERROR 31112 "No match for stringId in table PlRscString"
        ROLLBACK TRANSACTION
        RETURN
    END

  END

  RETURN

END
go
```

Trigger: TrigDelPlRscComputer

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlRscComputer
ON PlRscComputer
FOR DELETE
AS
BEGIN

  /* if no rows were deleted, then return */
```

```
   IF @@rowcount = 0
     RETURN

   /* delete matching rows in PlRscDiskPartition */

   DELETE PlRscDiskPartition
   FROM   PlRscDiskPartition c, deleted d
   WHERE  c.computerId = d.computerId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   /* delete matching rows in PlResource */

   DELETE PlResource
   FROM   PlResource t, deleted d
   WHERE  t.resourceId = d.computerId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlRscDiskPartition

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlRscDiskPartition
ON PlRscDiskPartition
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */
```

```
   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key diskPartitionId has a matching
     primary key in table PlResource              */

   IF (SELECT COUNT(*)
      FROM PlResource t, inserted i
      WHERE t.resourceId = i.diskPartitionId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for diskPartitionId in table PlResource"
      ROLLBACK TRANSACTION
      RETURN
   END

   /* Verify that foreign key computerId has a matching
     primary key in table PlRscComputer              */

   --IF (SELECT COUNT(*)
      --FROM PlRscComputer t, inserted i
      --WHERE t.computerId = i.computerId) != @num_rows
   --BEGIN
      --RAISERROR 31111 "No match for computerId in table PlRscComputer"
      --ROLLBACK TRANSACTION
      --RETURN
   --END

   RETURN

END
go
```

Trigger: TrigUpdPlRscDiskPartition

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlRscDiskPartition
ON PlRscDiskPartition
FOR UPDATE
AS
```

```
   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( diskPartitionId )
   BEGIN

      /* cascade update to child tables */

      UPDATE DpPrDiskAllocation
      SET t.diskPartitionId = i.diskPartitionId
      FROM DpPrDiskAllocation t, inserted i, deleted d
      WHERE t.diskPartitionId = d.diskPartitionId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   /* see if foreign keys were updated */

   IF UPDATE ( computerId )
   BEGIN

      /* Verify that computerId has a matching
       * primary key in table PlRscComputer    */

      IF ((SELECT COUNT(*)
         FROM PlRscComputer t, inserted i
         WHERE t.computerId = i.computerId) +
         (SELECT COUNT(*)
          FROM inserted i
          WHERE i.computerId is null) +
```

```
        (SELECT COUNT(*)
         FROM inserted i
         WHERE i.computerId = 0))
        != @numrows
      BEGIN
        RAISERROR 31112 "No match for computerId in table PlRscComputer"
        ROLLBACK TRANSACTION
        RETURN
      END

    END

    IF UPDATE ( diskPartitionId )
    BEGIN

      /* Verify that diskPartitionId has a matching
       * primary key in table PlResource           */

      IF (SELECT COUNT(*)
        FROM PlResource t, inserted i
        WHERE t.resourceId = i.diskPartitionId)
        != @numrows
      BEGIN
        RAISERROR 31112 "No match for diskPartitionId in table PlResource"
        ROLLBACK TRANSACTION
        RETURN
      END

    END

    RETURN

END
go
```

Trigger: TrigDelPlRscDiskPartition

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlRscDiskPartition
ON PlRscDiskPartition
FOR DELETE
AS
BEGIN
```

```
   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* delete matching rows in DpPrDiskAllocation */

   DELETE DpPrDiskAllocation
   FROM   DpPrDiskAllocation c, deleted d
   WHERE  c.diskPartitionId = d.diskPartitionId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   /* delete matching rows in PlResource */

   DELETE PlResource
   FROM   PlResource t, deleted d
   WHERE  t.resourceId = d.diskPartitionId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlRscString

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlRscString
ON PlRscString
FOR INSERT
AS
BEGIN
```

```
   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that key stringId has a matching
     primary key in table PlResource         */

   IF (SELECT COUNT(*)
      FROM PlResource t, inserted i
      WHERE t.resourceId = i.stringId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for stringId in PlResource"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlRscString

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlRscString
ON PlRscString
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN
```

```
    /* see if the primary key was updated */

    IF UPDATE ( stringId )
    BEGIN

        /* cascade update to child tables */

        UPDATE PlRscComputer
        SET t.stringId = i.stringId
        FROM PlRscComputer t, inserted i, deleted d
        WHERE t.stringId = d.stringId

        IF @@transtate = 2 /* previous stmt aborted */
        BEGIN
            ROLLBACK TRANSACTION
                RETURN
        END

    END

    /* see if foreign keys were updated */

    IF UPDATE ( stringId )
    BEGIN

        /* Verify that stringId has a matching
         * primary key in table PlResource     */

        IF (SELECT COUNT(*)
            FROM PlResource t, inserted i
            WHERE t.resourceId = i.stringId)
            != @numrows
        BEGIN
            RAISERROR 31112 "No match for stringId in table PlResource"
            ROLLBACK TRANSACTION
            RETURN
        END

    END

    RETURN

END
go
```

Trigger: TrigDelPlRscString

Trigger Code

```
/*   */
CREATE TRIGGER TrigDelPlRscString
ON PlRscString
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
     RETURN

   /* zero matching stringId in table PlRscComputer */

   UPDATE PlRscComputer
   SET    c.stringId = 0
   FROM   PlRscComputer c, deleted d
   WHERE  c.stringId = d.stringId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   /* delete matching rows in PlResource */

   DELETE PlResource
   FROM   PlResource t, deleted d
   WHERE  t.resourceId = d.stringId

   IF @@transtate = 2 /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   RETURN

END
```

```
go
```

Trigger: TrigInsPlTile

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlTile
ON PlTile
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key clusterId has a matching
     primary key in table PlCluster                    */

   IF (SELECT COUNT(*)
      FROM PlCluster t, inserted i
      WHERE t.clusterId = i.clusterId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for clusterId in table PlCluster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlTile

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlTile
ON PlTile
FOR UPDATE
```

```
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN

   SELECT @numrows = @@rowcount

   /* if no rows were deleted, then return */

   IF @numrows = 0
     RETURN

   /* see if the primary key was updated */

   IF UPDATE ( tileId )
   BEGIN

      /* cascade update to child tables */

      UPDATE PlTileCoordinates
      SET t.tileId = i.tileId
      FROM PlTileCoordinates t, inserted i, deleted d
      WHERE t.tileId = d.tileId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

      UPDATE PlDataGranuleShort
      SET t.tileId = i.tileId
      FROM PlDataGranuleShort t, inserted i, deleted d
      WHERE t.tileId = d.tileId

      IF @@transtate = 2 /* previous stmt aborted */
      BEGIN
         ROLLBACK TRANSACTION
            RETURN
      END

   END

   /* see if foreign keys were updated */
```

```
   IF UPDATE ( clusterId )
   BEGIN

      /* Verify that foreign key clusterId has a matching
       * primary key in table PlCluster                  */

      IF (SELECT COUNT(*)
         FROM PlCluster t, inserted i
         WHERE t.clusterId = i.clusterId) != @numrows
      BEGIN
         RAISERROR 31112 "No match for clusterId in table PlCluster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigDelPlTile

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlTile
ON PlTile
FOR DELETE
AS
BEGIN

   /* if no rows were deleted, then return */

   IF @@rowcount = 0
      RETURN

   /* delete matching rows in PlTileCoordinates */

   DELETE PlTileCoordinates
   FROM   PlTileCoordinates t, deleted d
   WHERE  t.tileId = d.tileId
```

```
   IF @@transtate = 2  /* previous statement aborted */
   BEGIN
      ROLLBACK TRANSACTION  /* undo all deletes */
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigInsPlTileCoordinates

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlTileCoordinates
ON PlTileCoordinates
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key tileId has a matching
     primary key in table PlTile                    */

   IF (SELECT COUNT(*)
      FROM PlTile t, inserted i
      WHERE t.tileId = i.tileId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for tileId in table PlTile"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpsPlTileCoordinates

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpsPlTileCoordinates
ON PlTileCoordinates
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( tileId )
   BEGIN

      /* Verify that foreign key tileId has a matching
       * primary key in table PlTile                  */

      IF (SELECT COUNT(*)
         FROM PlTile t, inserted i
         WHERE t.tileId = i.tileId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for tileId in table PlTile"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlTileScheduled

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlTileScheduled
ON PlTileScheduled
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that every foreign key has a matching
     primary key in table PlPgeMaster            */

   IF (SELECT COUNT(*)
      FROM PlPgeMaster t, inserted i
      WHERE t.pgeId = i.pgeId)
      != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId in table PlPgeMaster"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlTileScheduled

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigUpdPlTileScheduled
ON PlTileScheduled
FOR UPDATE
AS

   DECLARE @numrows int  /* number of rows updated */

BEGIN
```

```
SELECT @numrows = @@rowcount

/* if no rows were deleted, then return */

IF @numrows = 0
  RETURN

/* see if the primary key was updated */

IF UPDATE ( pgeId )
BEGIN

   /* cascade update to child tables */

   UPDATE PlCluster
   SET t.pgeId = i.pgeId
   FROM PlCluster t, inserted i, deleted d
   WHERE t.pgeId = d.pgeId

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

   UPDATE PlDataHistory
   SET t.pgeId = i.pgeId
   FROM PlDataHistory t, inserted i, deleted d
   WHERE t.pgeId = d.pgeId

   IF @@transtate = 2 /* previous stmt aborted */
   BEGIN
      ROLLBACK TRANSACTION
         RETURN
   END

END

/* see if foreign keys were updated */

IF UPDATE ( pgeId )
BEGIN

   /* Verify that every foreign key has a matching
```

```
     * primary key in table PlPgeMaster            */

    IF (SELECT COUNT(*)
       FROM PlPgeMaster t, inserted i
       WHERE t.pgeId = i.pgeId)
       != @numrows
    BEGIN
       RAISERROR 31112 "No match for pgeId in table PlPgeMaster"
       ROLLBACK TRANSACTION
       RETURN
    END

  END

  RETURN

END
go
```

Trigger: TrigDelPlTileScheduled

Trigger Code

```
/*                                        */
CREATE TRIGGER TrigDelPlTileScheduled
ON PlTileScheduled
FOR DELETE
AS
BEGIN

  /* if no rows were deleted, then return */

  IF @@rowcount = 0
    RETURN

  /* delete matching rows in PlCluster */

  DELETE PlCluster
  FROM   PlCluster t, deleted d
  WHERE  t.pgeId = d.pgeId

  IF @@transtate = 2  /* previous statement aborted */
  BEGIN
     ROLLBACK TRANSACTION  /* undo all deletes */
     RETURN
```

```
   END

   RETURN

END
go
```

Trigger: ti_pltimer

Trigger Code

```
/*  Insert trigger "ti_pltimer" for table "PlTimer"  */
create trigger ti_pltimer on PlTimer for insert as
begin
   declare
      @numrows  int,
      @numnull  int,
      @errno    int,
      @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
      return


   /*  Parent "PlDataGranuleShort" must exist when inserting a child in "PlTimer"  */
   if update(granuleId)
   begin
      if (select count(*)
          from   PlDataGranuleShort t1, inserted t2
          where  t1.granuleId = t2.granuleId) != @numrows
         begin
            select @errno  = 30002,
                   @errmsg = 'Parent does not exist in "PlDataGranuleShort". Cannot create child in
"PlTimer".'
            goto error
         end
   end

   /*  Parent "PlDataProcessingRequest" must exist when inserting a child in "PlTimer"  */
   if update(dprId)
   begin
      if (select count(*)
          from   PlDataProcessingRequest t1, inserted t2
          where  t1.dprId = t2.dprId) != @numrows
```

```
        begin
          select @errno  = 30002,
                 @errmsg = 'Parent does not exist in "PlDataProcessingRequest". Cannot create child
in "PlTimer".'
          goto error
        end
    end

    return

/*  Errors handling  */
error:
    raiserror @errno @errmsg
    rollback  transaction
end
go
```

Trigger: TrigInsPlUsedByCenter

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlUsedByCenter
ON PlUsedByCenter
FOR INSERT
AS
BEGIN

    /* See how many rows were inserted */

    DECLARE @num_rows int
    SELECT @num_rows = @@rowcount
    IF @num_rows = 0
       RETURN

    /* Verify that foreign key dataTypeId has a matching
       primary key in table PlDataTypeMaster            */

    IF (SELECT COUNT(*)
       FROM PlDataTypeMaster t, inserted i
       WHERE t.dataTypeId = i.dataTypeId) != @num_rows
    BEGIN
       RAISERROR 31111 "No match for dataTypeId in table PlDataTypeMaster"
       ROLLBACK TRANSACTION
       RETURN
```

```
   END

   RETURN

END
go
```

Trigger: TrigUpdPlUsedByCenter

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlUsedByCenter
ON PlUsedByCenter
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( dataTypeId )
   BEGIN

      /* Verify that foreign key dataTypeId has a matching
       * primary key in table PlDataTypeMaster            */

      IF (SELECT COUNT(*)
         FROM PlDataTypeMaster t, inserted i
         WHERE t.dataTypeId = i.dataTypeId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for dataTypeId in table PlDataTypeMaster"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
```

```
go
```

Trigger: TrigInsPlUserParamConstraints

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlUserParamConstraints
ON PlUserParameterConstraints
FOR INSERT
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key (pgeId, pgeParameterLogicalId) has a matching
     primary key in table PlPgeDetailParameters                    */

   IF (SELECT COUNT(*)
      FROM PlPgeDetailParameters t, inserted i
      WHERE t.pgeId = i.pgeId
      AND   t.pgeParameterLogicalId = i.pgeParameterLogicalId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for pgeId/pgeParameterLogicalId in table
PlPgeDetailParameters"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlUserParamConstraints

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlUserParamConstraints
```

```
ON PlUserParameterConstraints
FOR UPDATE
AS
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( pgeId ) OR UPDATE ( pgeParameterLogicalId )
   BEGIN

      /* Verify that foreign key (pgeId, pgeParameterLogicalId) has a matching
       * primary key in table PlPgeDetailParameters   */

      IF (SELECT COUNT(*)
         FROM PlPgeDetailParameters t, inserted i
         WHERE t.pgeId = i.pgeId
         AND   t.pgeParameterLogicalId = i.pgeParameterLogicalId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for pgeId/pgeParameterLogicalId in table
PlPgeDetailParameters"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: TrigInsPlUserPriority

Trigger Code

```
/*   */
CREATE TRIGGER TrigInsPlUserPriority
ON PlUserPriority
FOR INSERT
AS
```

```
BEGIN

   /* See how many rows were inserted */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   /* Verify that foreign key strategyId has a matching
     primary key in table PlProductionStrategy        */

   IF (SELECT COUNT(*)
      FROM PlProductionStrategy t, inserted i
      WHERE t.strategyId = i.strategyId) != @num_rows
   BEGIN
      RAISERROR 31111 "No match for strategyId in table PlProductionStrategy"
      ROLLBACK TRANSACTION
      RETURN
   END

   RETURN

END
go
```

Trigger: TrigUpdPlUserPriority

Trigger Code

```
/*   */
CREATE TRIGGER TrigUpdPlUserPriority
ON PlUserPriority
FOR UPDATE
AS
BEGIN

   /* See how many rows were updated */

   DECLARE @num_rows int
   SELECT @num_rows = @@rowcount
   IF @num_rows = 0
      RETURN

   IF UPDATE ( strategyId )
```

```
   BEGIN

      /* Verify that foreign key strategyId has a matching
       * primary key in table PlProductionStrategy        */

      IF (SELECT COUNT(*)
         FROM PlProductionStrategy t, inserted i
         WHERE t.strategyId = i.strategyId) != @num_rows
      BEGIN
         RAISERROR 31112 "No match for strategyId in table PlProductionStrategy"
         ROLLBACK TRANSACTION
         RETURN
      END

   END

   RETURN

END
go
```

Trigger: ti_rpreservationinterval

Trigger Code

```
/*  Insert trigger "ti_rpreservationinterval" for table "RpReservationInterval"  */
create trigger ti_rpreservationinterval on RpReservationInterval for insert as
begin
   declare
      @numrows  int,
      @numnull  int,
      @errno    int,
      @errmsg   varchar(255)

   select  @numrows = @@rowcount
   if @numrows = 0
      return


   /*  Parent "RpResourceReservation" must exist when inserting a child in
"RpReservationInterval"  */
   if update(reservationId)
   begin
      if (select count(*)
         from   RpResourceReservation t1, inserted t2
```

```
        where  t1.reservationId = t2.reservationId) != @numrows
       begin
         select @errno  = 30002,
               @errmsg = 'Parent does not exist in "RpResourceReservation". Cannot create child
in "RpReservationInterval".'
         goto error
       end
   end

   return

/*  Errors handling  */
error:
   raiserror @errno @errmsg
   rollback  transaction
end
go
```

4.1.10 Stored Procedures

Sybase also includes support for business policy via the use of stored procedures. Stored procedures are typically used to capture a set of activities or checks that will be performed on the database repeatedly to enforce business policy and maintain data integrity. Stored procedures are parsed and complied SQL code that reside in the database and may be called by name by an application, trigger or another stored procedure A listing of each the stored procedures in the PDPS database is given here. A brief definition of each of these stored procedures follows.

Procedure List

Name	Description
insertGroundEvent	TBS
ProcGetComments	TBS
ProcGetResources	TBS

Procedure: insertGroundEvent

Code

```
create proc insertGroundEvent
as

create table #ground_event
( name varchar(60) not null,
  descs varchar(30) null,
  start_time datetime not null,
  end_time  datetime null,
```

```
  duration int null,
  priority int null,
  constraint pk_groun_event primary key (name, start_time)
)

insert #ground_event (name, descs, start_time, end_time, priority)
select rsv.name, rsv.description, intv.startTime, intv.stopTime, rsv.priority
from RpResourceReservation rsv, RpReservationInterval intv
where rsv.state = "approved"
and rsv.reservationId = intv.reservationId

update #ground_event
set duration = datediff(minute, start_time, end_time)
from #ground_event

select * from #ground_event
go
```

Procedure: ProcGetComments

Code

```
create proc ProcGetComments @commentId int, @commentType varchar(20)
as

select comment from RpRscPlanComments
where commentId = @commentId and
     commentType = @commentType
order by orderNum
go
```

Procedure: ProcGetResources

Code

```
create proc ProcGetResources as

create table #tmpRP
( rsId int not null,
  rsName varchar(60) null,
  rsType varchar(15) null,
  actName varchar(20) null,
  stringId int null,
  operSys varchar(60) null,
  totRam int null,
  totCpu int null,
```

```
  computerId int null,
  realComputerId int null,
  partSize float null,
  blockSize int null,
  autosysIdKey int null)

insert #tmpRP(rsId, rsName, rsType, actName)
select rm.resourceId, rm.resourceName, rm.resourceType, ra.activityName
from PlResource rm, RpActivityType ra
where rm.activityTypeId *= ra.activityTypeId

update #tmpRP
set operSys = operatingSys,
    totRam = totalRam,
    totCpu = totalCpu,
    #tmpRP.stringId = rc.stringId,
    #tmpRP.realComputerId = rc.realComputerId
from PlRscComputer rc, #tmpRP
where rc.computerId = #tmpRP.rsId

update #tmpRP
set  #tmpRP.autosysIdKey = rs.autosysIdKey
from PlRscString rs, #tmpRP
where rs.stringId = #tmpRP.rsId

update #tmpRP
set #tmpRP.computerId = rd.computerId, partSize = partitionSize,
    #tmpRP.blockSize = rd.blockSize
from PlRscDiskPartition rd, #tmpRP
where rd.diskPartitionId = #tmpRP.rsId

select * from #tmpRP
go
```

4.2 File Usage

There are cases when the implementation of a persistent data requirement is better suited to a flat file than to a database table. A typical example of such data is system configuration information. System configuration information is fairly static and usually has no explicit relationship to other data in the enterprise. Another common use of files in ECS is as an interface mechanism between ECS and the external world. FIILSUB file usage is detailed in this section via file definitions, attribute definitions, and attribute domain definitions.

4.2.1 Files Definitions

A listing of each the files in the PDPS database is given here. A brief definition of each of these files follows.

4.2.2 Attributes

Brief definitions of each of the attributes present in the files defined above are contained herein.

4.2.3 Attribute Domains

Domains represent the ranges of valid values allowed for a given file attribute. Attributes domains for each of the attributes defined above are given here.

5. Performance and Tuning Factors

5.1 Indexes

An index provides a means of locating a row in a table based on the value of specific a columns, without having to scan each row in the table. If used appropriately, indexes can significantly increase data retrieval. Sybase allows the definition of two types of indexes, clustered and non-clustered. In a clustered index, the rows in a tables are physically stored in the sort order determined by the index. Clustered indexes are particularly useful, when the data is frequently retrieved in order. Non-clustered indexes differ from their clustered counterpart, in that data is not physically stored in sort order. Only one clustered index may be defined per table. All of the indexes defined against tables in the PDPS database are described herein.

TBS

5.2 Segments

Sybase supports the definition of segments. A segment is a named pointer to a storage device or devices. Segments are used to manually place database objects onto particular storage devices. All segments explicitly defined in the PDPS database are described herein.

TBS

5.3 Named Caches

A cache is a block of memory that is used by Sybase to house data pages that are currently being accessed. A named cache is a named block of memory that the SQL server can use to house frequently accessed tables. Assigning a table to cache causes it to be loaded into memory. This greatly increases performance by eliminating the time expense normally associated with disk i/o. Named caches used in the PDPS databases are described herein.

TBS

This page intentionally left blank.

6. Database Security

6.1 Initial Users

Upon initial installation the following users will have access to PDPS database. The level of access is limited to that associated with their assigned group and/or role. A complete definition of each if these groups and roles is given below.

TBS

6.2 Groups

Groups are a means of logically associating users with similar data access needs. Once a group has been defined, object and command permissions can be granted to that group. A user who is member of a group inherits all of the permissions granted to that group. Several group have been defined in the PDPS database upon initial installation. A definition of each of these groups is contained herein.

TBS

6.3 Roles

Roles were introduced in Sybase 10 to allow a structured means for granting users the permissions needed to perform standard database administration activities and also provide a means for easily identifying such users. There a six pre-defined roles that may be assigned to a user. A definition of each of these roles follows as well as a description of the types of activities that may be performed by each role.

System Administrator (sa_role) - This role is used to grant a specific user to permissions needed to perform standard system administrator duties including:

- installing SQL server and specific SQL server modules
- managing the allocation of physical storage
- tuning configuration parameters
- creating databases

Site Security Officer (sso_role) - This role is used to grant a specific user the permissions needed to maintain SQL server security including:

- adding server logins
- administrating passwords

- managing the audit system
- granting users all roles except sa_role

Operator (oper_role) - This role is used to grant a specific user the permissions needed to manage backup and recovery of the database including;

- dumping transactions and databases
- loading transactions and databases

Navigator (navigator_role) -This role is used to grant a specific user the permissions needed to manage the navigation server.

Replication (replication_role) - - This role is used to grant a specific user the permissions needed to manage the replication server.

Sybase Technical Support (sybase_ts_role) - This role is used to grant a specific user the permissions needed to perform database consistency checker (dbcc), a sybase supplied utility, commands that are considered outside of the realm of normal system administrator activities.

6.4 Object Permissions

TBS

7. Replication

7.1 Replication Overview

Replication as the name implies is a set of Sybase products that allow replication of data from one database to another. The PDPS database employs replication to support its warm-standby requirements. In order for replication to be accomplished the data source must defined the tables and columns that may be replicated to a data recipient. These definitions are referred to replication definitions. In the same manner a data recipient must specify the replication definitions in which he is interested. These specifications are referred to as replication subscriptions. In addition the replication database and server must be configured to support the potentially large volumes of data that will be transferred between the source and recipient databases. Each of these important parameters is outlined in detail below.

7.2 Replication Definitions

Replication definitions that have been defined against PDPS tables and columns are detailed herein.

TBS

7.3 Replication Subscriptions

Replication subscriptions that have been defined against PDPS tables and columns are detailed herein.

TBS

7.4 Replication Database Configuration

Replication Database Configuration specifications applicable to PDPS replication are contained herein.

7.5 Replication Server Configuration

Replication Server Configuration specifications applicable to PDPS replication are contained herein.

TBS

This page intentionally left blank.

8. Scripts

8.1 Installation Scripts

Any scripts used to support installation of the PDPS database are described herein. These files may be found in the directory /ecs/formal/PDPS/PLS/sybase

Script File	Description
EcPlCreatePDPSDatabase	Create database tables
EcPlInitializePDPSDatabase	Populate tables needing intialization.

8.2 De-Installation Scripts

Any scripts used to support de-installation of the PDPS database are described herein.

TBS

8.3 Backup/Recovery Scripts

Any scripts used to facilitate backup or recovery of the PDPS database are described herein.

TBS

8.4 Miscellaneous Scripts

Miscellaneous scripts applicable to the PDPS database are described herein.

TBS

This page intentionally left blank.

Appendix - PDPS ERDs

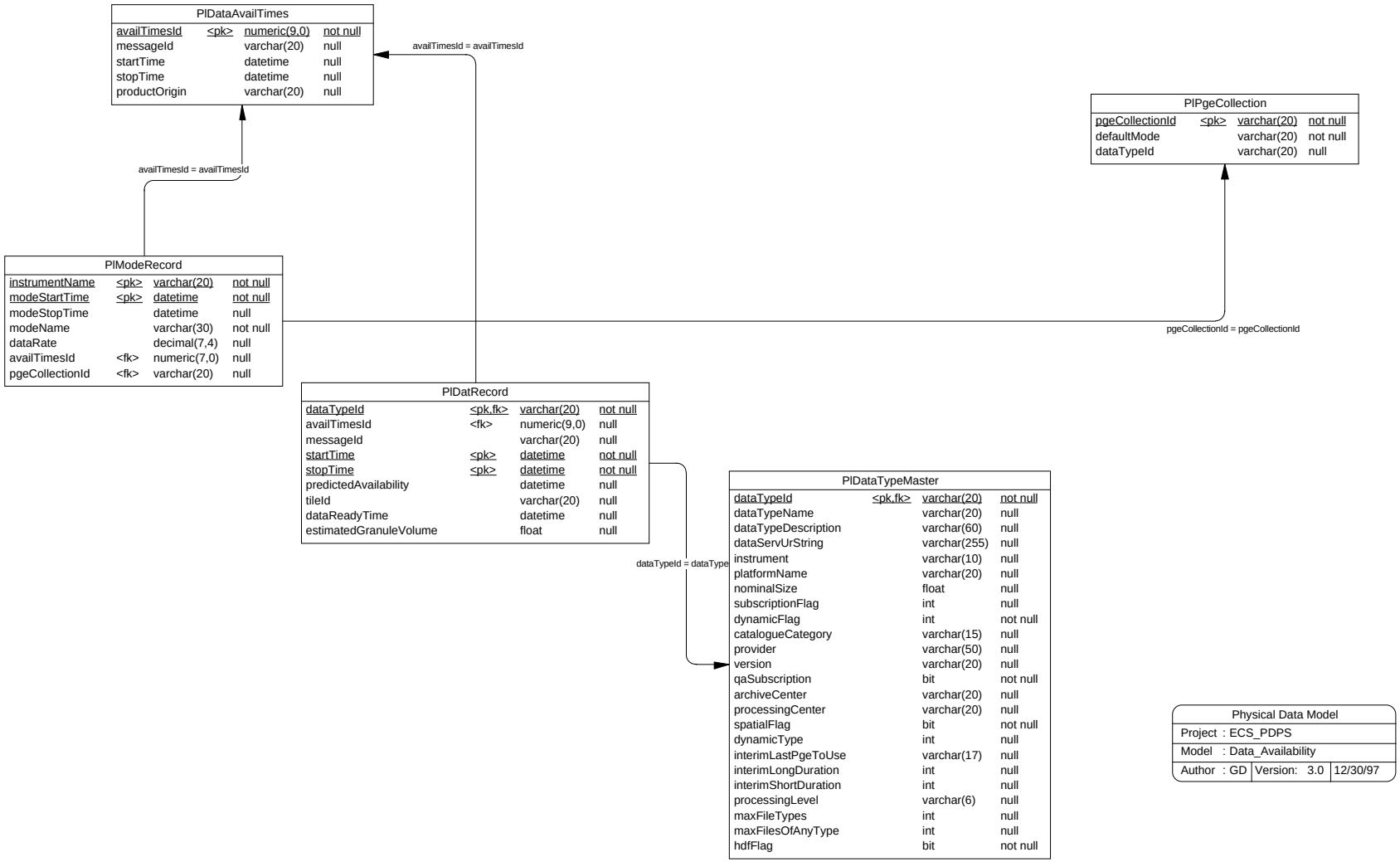
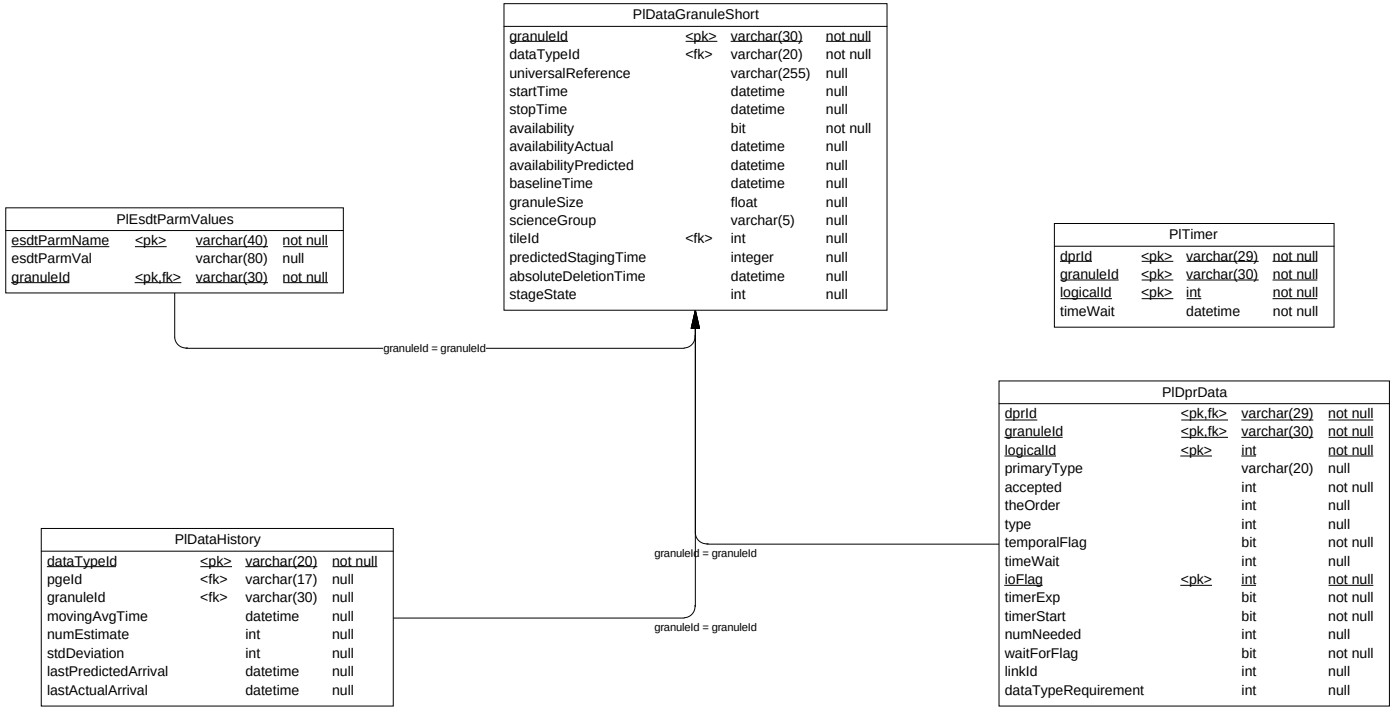


Figure A-1. Data Availability



Physical Data Model		
Project : ECS_PDPS		
Model : Data_Granules		
Author : GD	Version: 3.0	12/30/97

Figure A-2. Data Granules

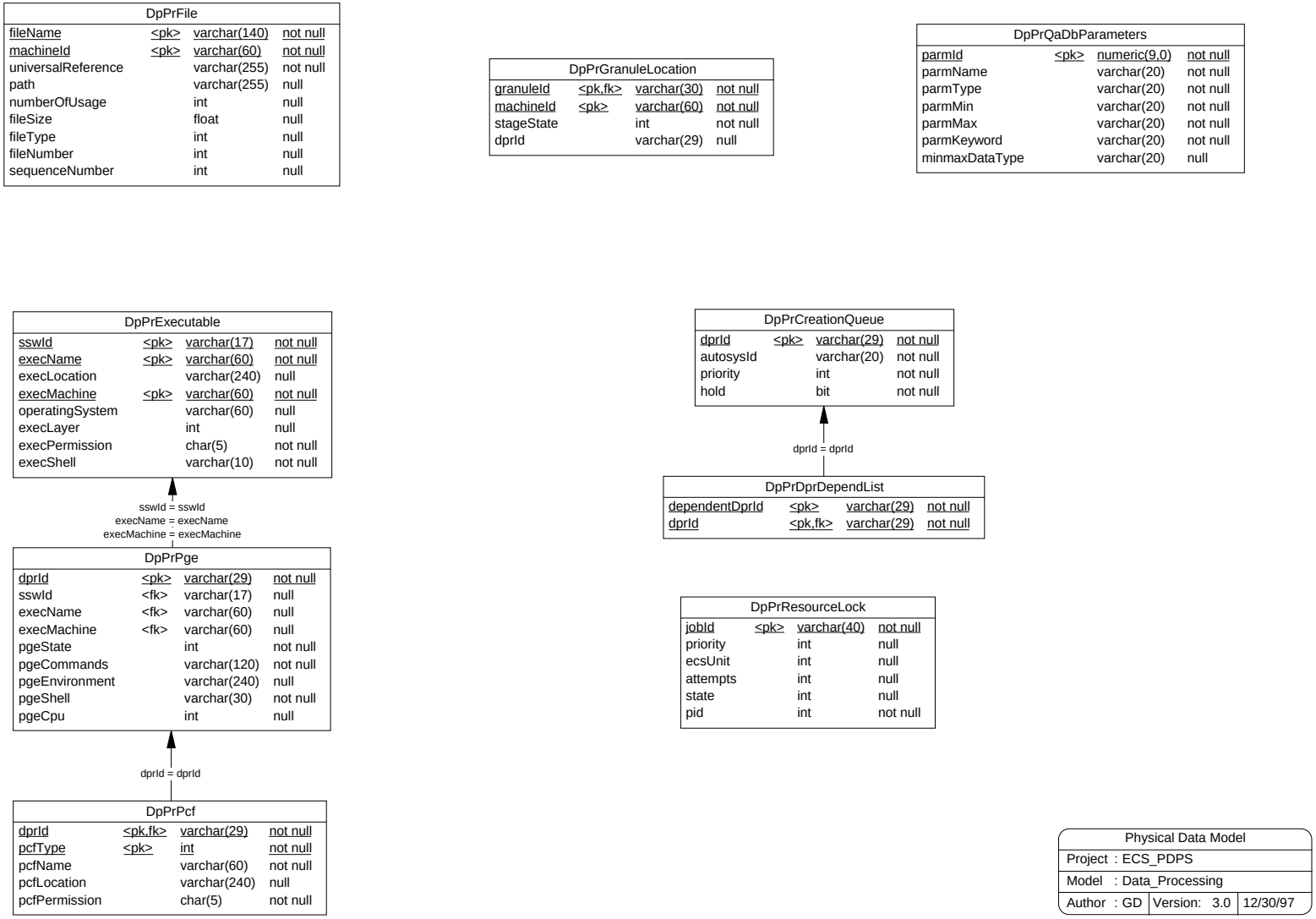


Figure A-3. Data Processing

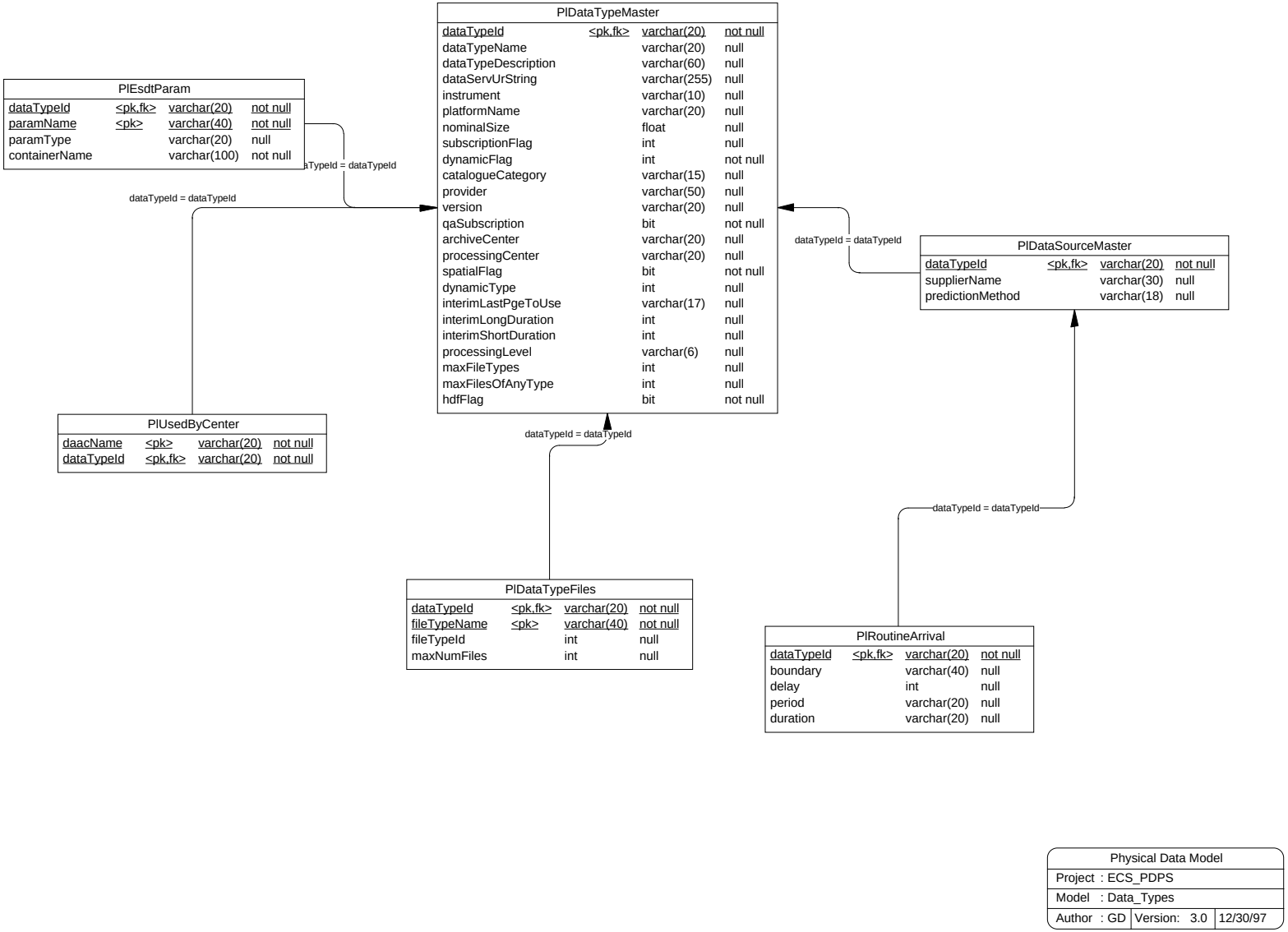


Figure A-4. Data Types

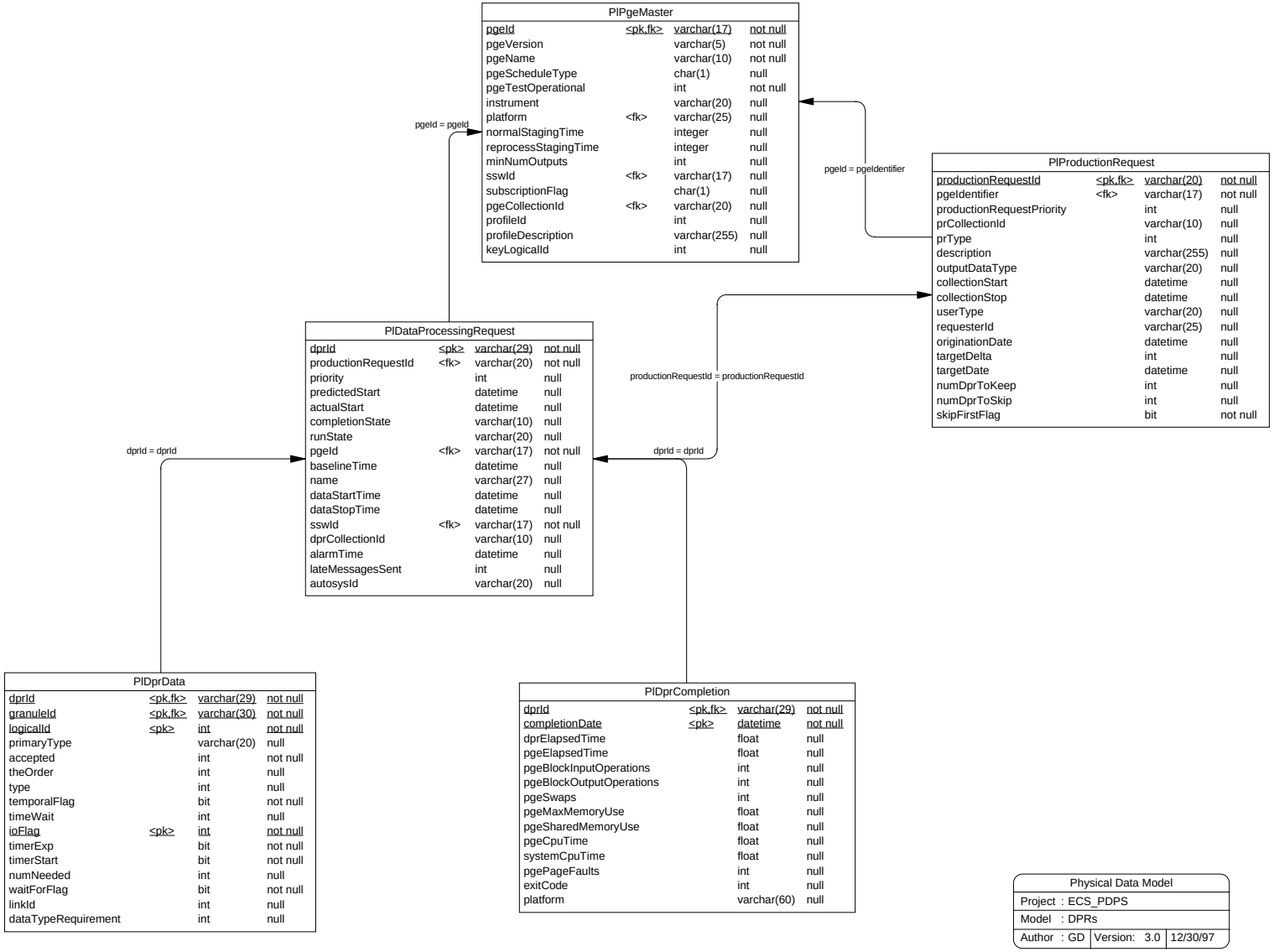


Figure A-5. DPRs

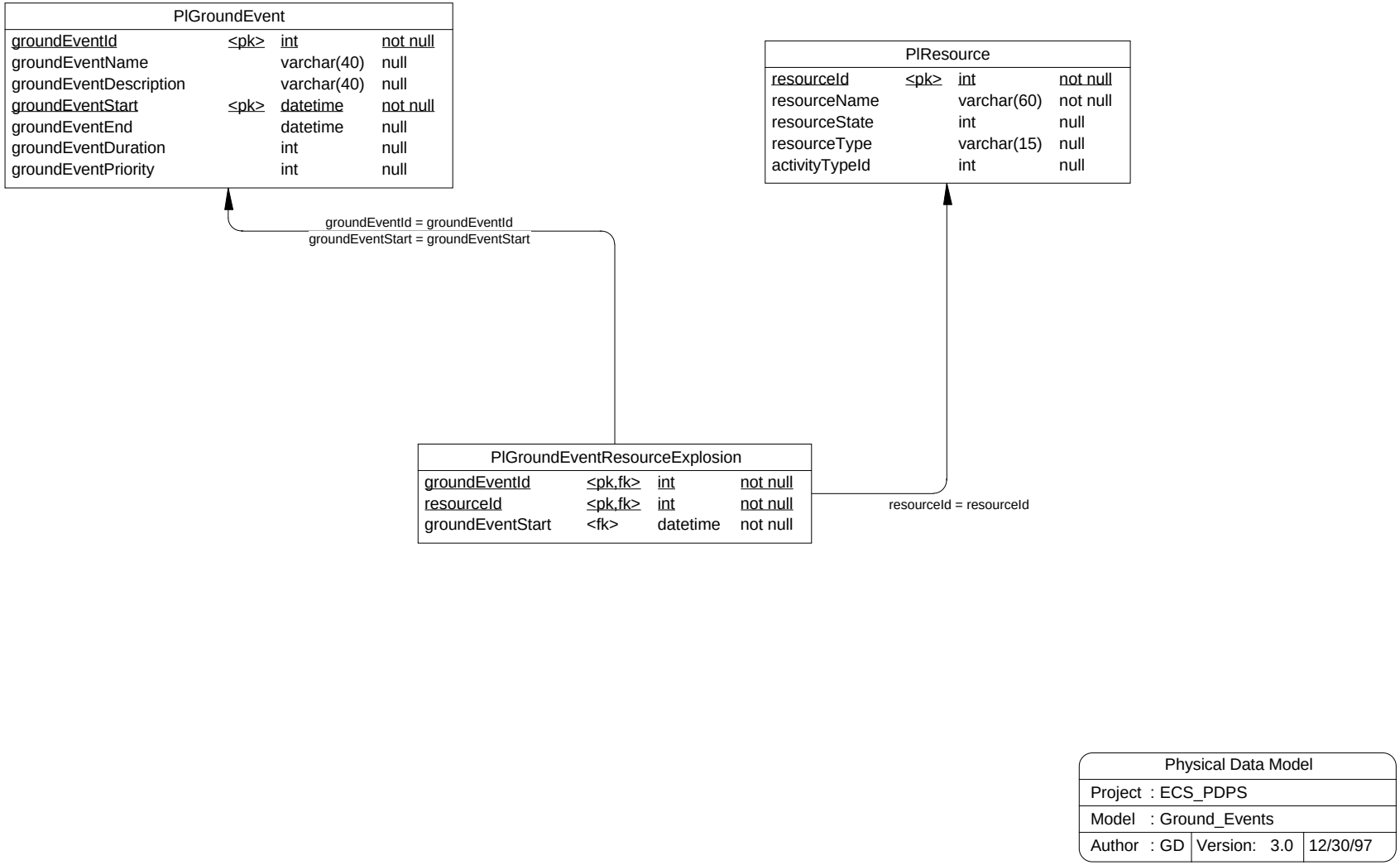


Figure A-6. Ground Events

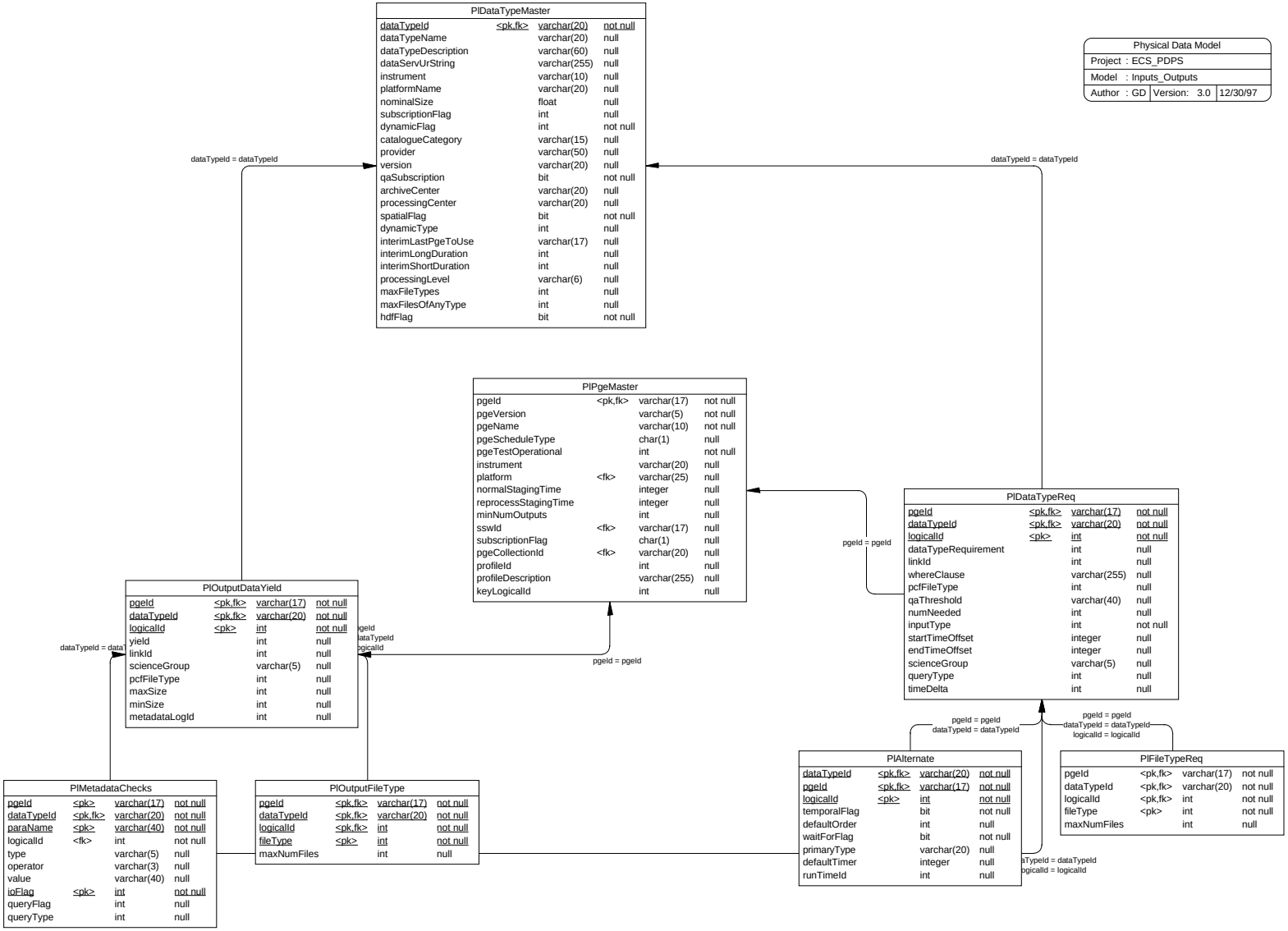


Figure A-7. Inputs_Outputs

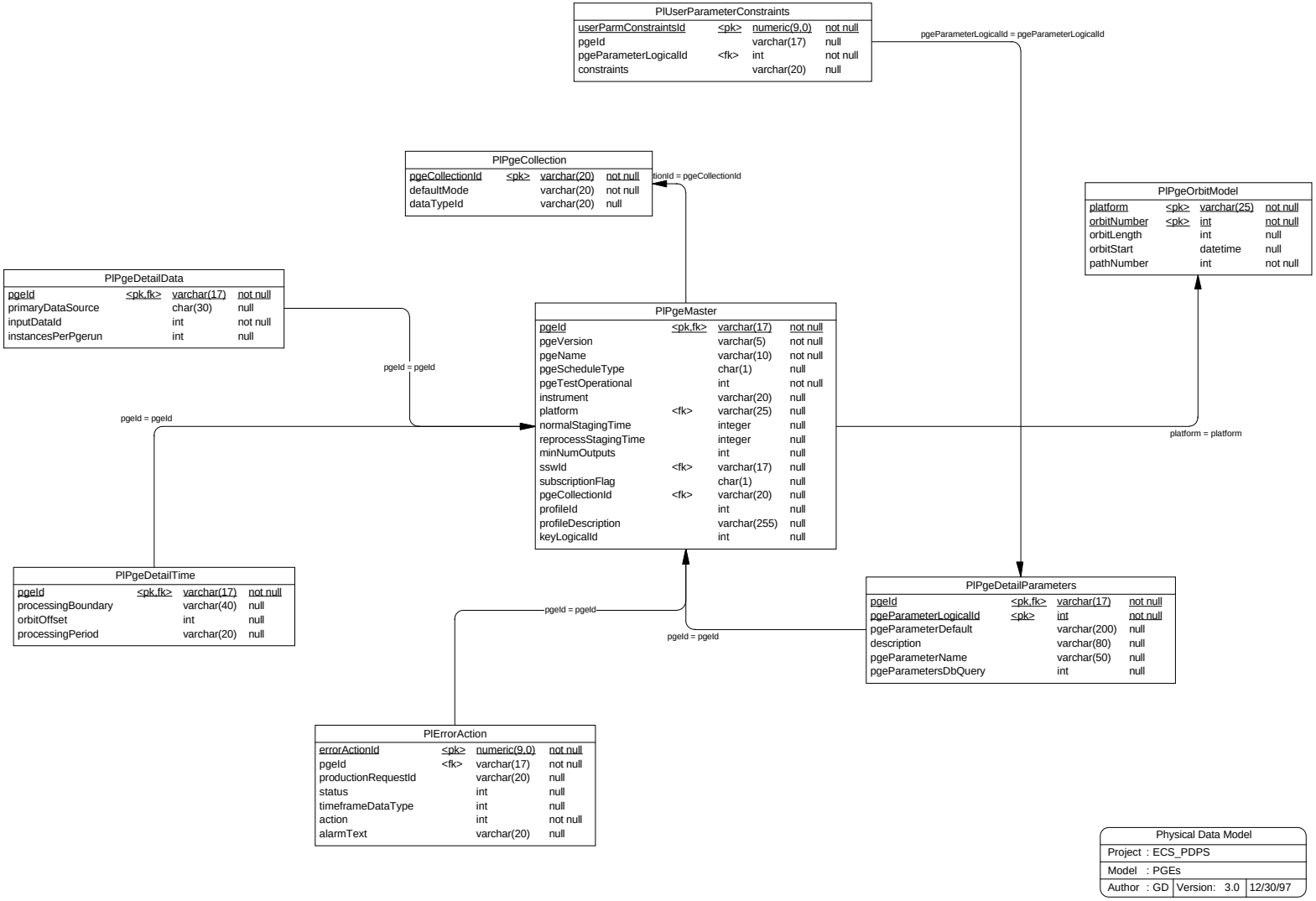


Figure A-8. PGEs

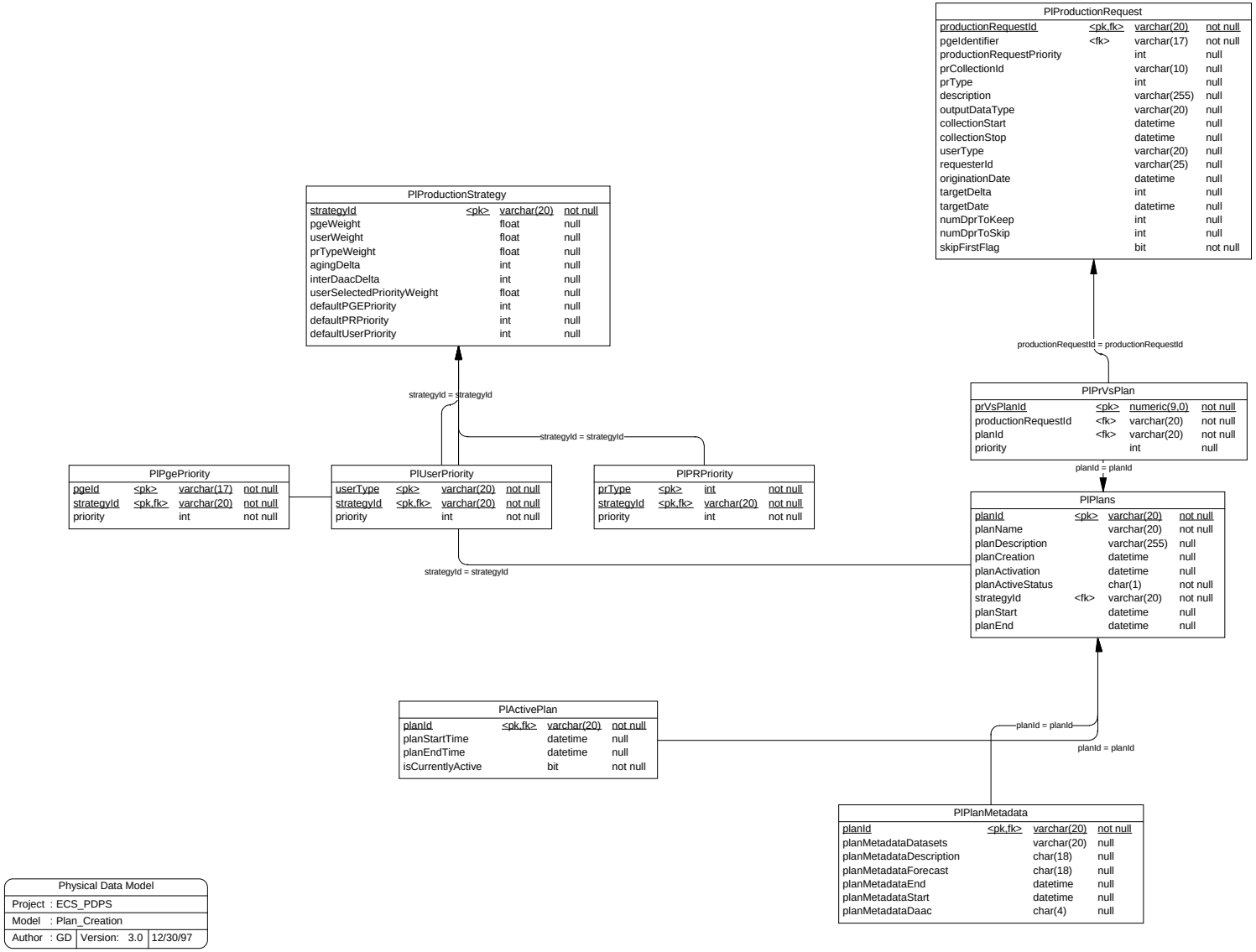


Figure A-9. Plan Creation

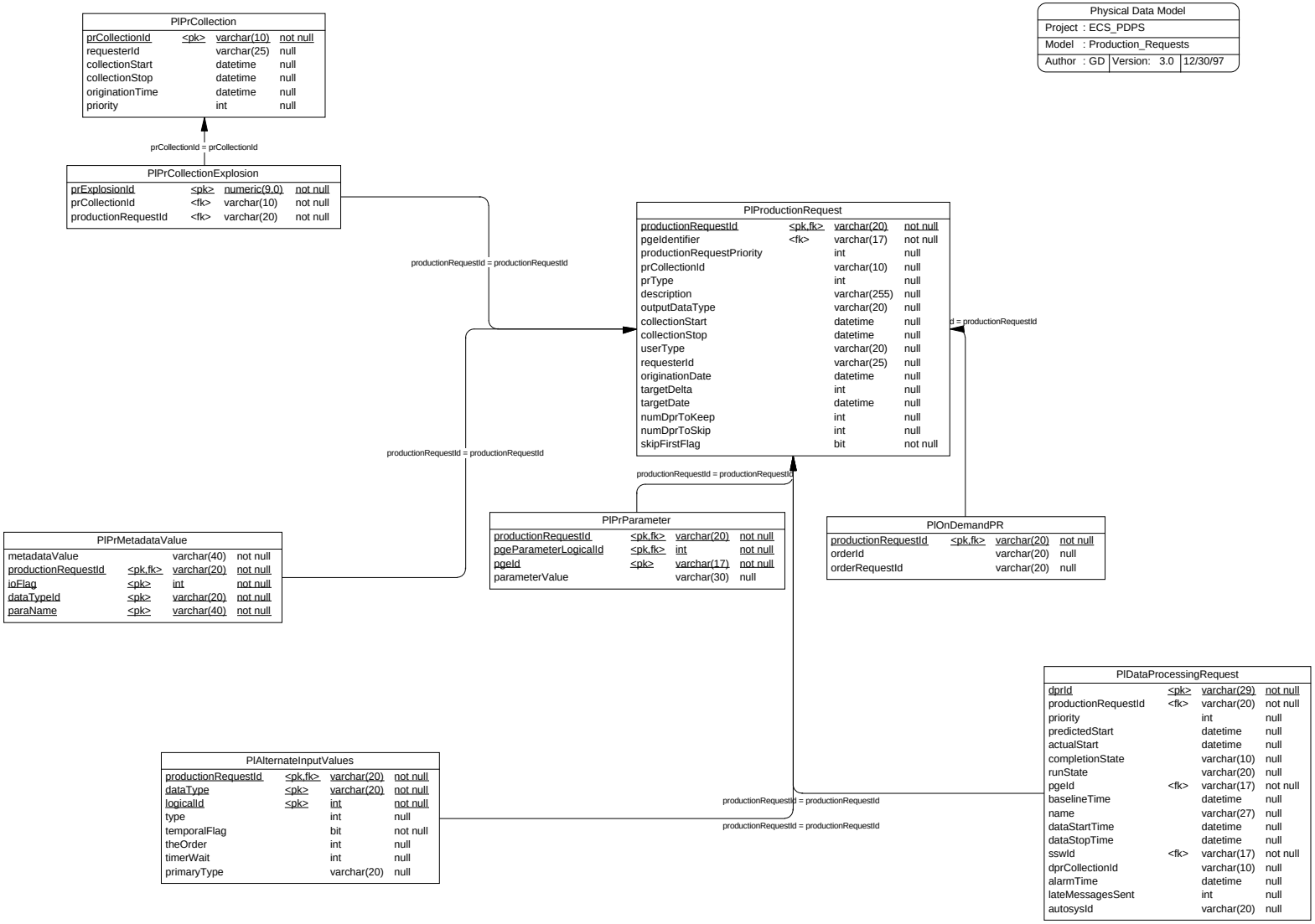
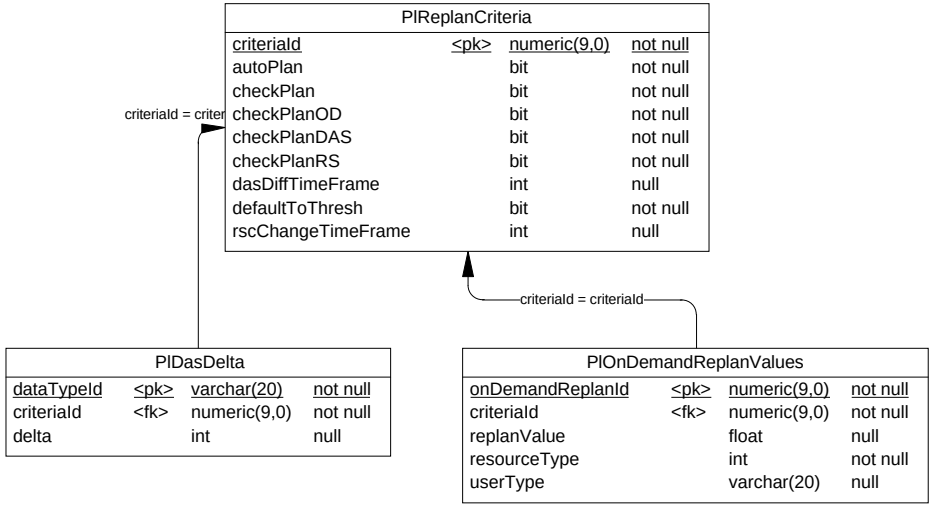


Figure A-10. Production Requests



Physical Data Model		
Project : ECS_PDPS		
Model : Replanning		
Author : GD	Version: 3.0	12/30/97

Figure A-11. Replanning

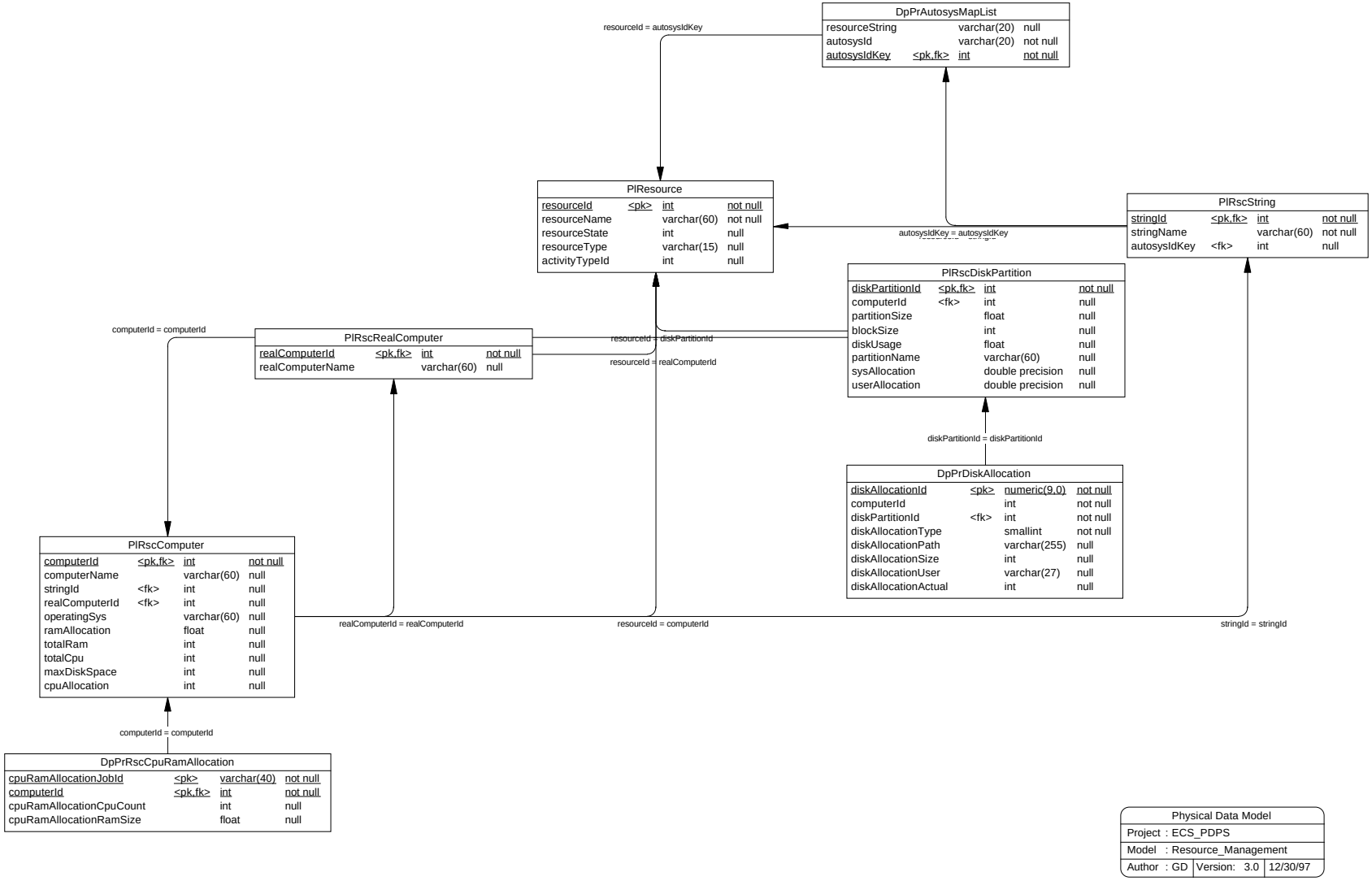


Figure A-12. Resource Management

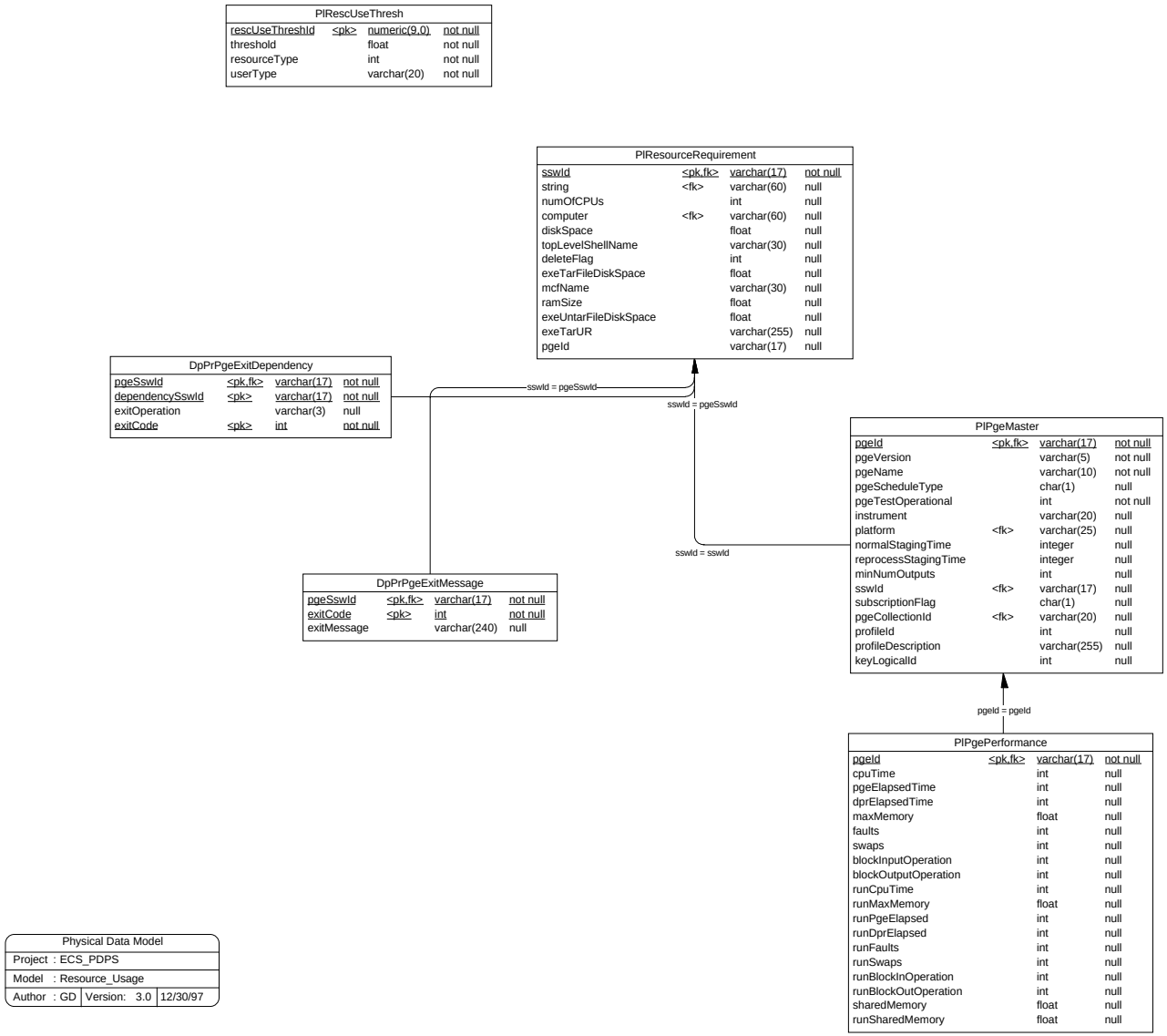
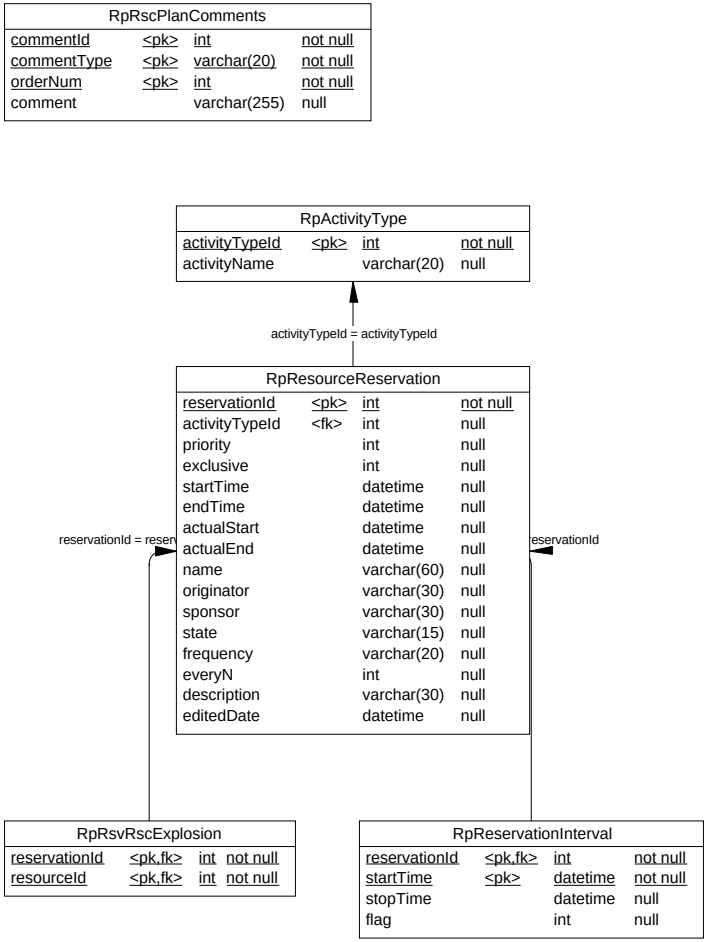
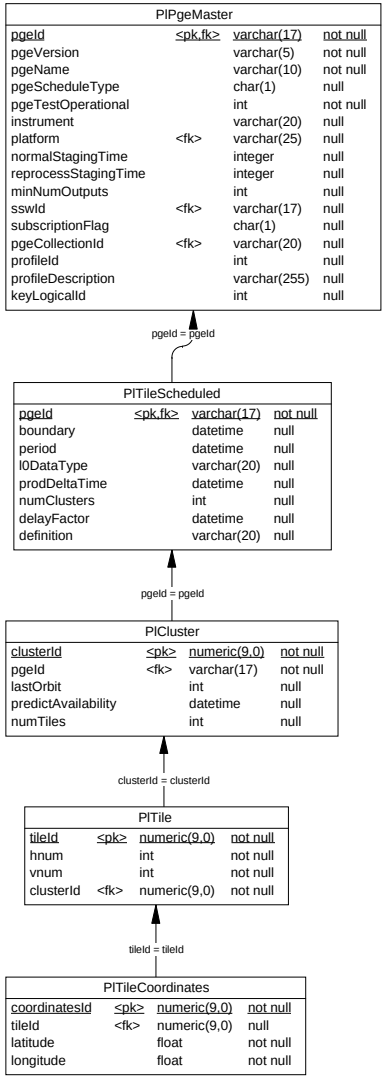


Figure A-13. Resource Usage



Physical Data Model		
Project : ECS_PDPS		
Model : Resource_Planning		
Author : GD	Version: 3.0	12/30/97

Figure A-14. Resource Planning



Physical Data Model		
Project : ECS_PDPS		
Model : Tiling		
Author : GD	Version: 3.0	12/30/97

Figure A-15. Tiling

Abbreviations and Acronyms

ACL	Access Control List
ACMHW	Access and Control Management HWCI
ADC	affiliated data center
ADSHW	Advertising Server HWCI
ADSRV	Advertising Service CSCI
AI&T	algorithm integration and test
AITHW	Algorithm Integration and Test HWCI
AITTL	Algorithm Integration and Test CSCI
AM-1	EOS AM Project spacecraft 1, morning spacecraft series -- ASTER, CERES, MISR, MODIS and MOPITT instruments
ANSI	American National Standards Institute
API	application program (or programming) interface
APID	application's process ID
AQAHW	Algorithm QA HWCI
ASCII	American Standard Code for Information Exchange
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
AVHRR	Advanced Very High-Resolution Radiometer
BER	bit error rate
BUFR	binary universal format for representation of data
CASE	Computer Aided Software Engineering
CCSDS	Consultative Committee for Space Data Systems
CD	contractual delivery 214-001
CD-ROM	compact disk -- read only memory
CDR	Critical Design Review
CDRL	contract data requirements list
CERES	Clouds and Earth's Radiant Energy System

CI	configuration item
COTS	commercial off-the-shelf (hardware or software)
CPU	central processing unit
CSCI	computer software configuration item
CSDT	Computer Science Data Type
CSMS	Communications and Systems Management Segment (ECS)
CSS	Communications Subsystem
DAAC	Distributed Active Archive Center
DAN	data availability notice
DAO	Data Assimilation Office
DAR	data acquisition request
DAS	data availability schedule
DBMS	Database Management System
DDICT	Data Dictionary CSCI
DDIST	Data Distribution Services CSCI
DDSRV	Document Data Server CSCI
DESKT	Desktop CSCI
DID	data item description
DIM	distributed information manager (SDPS)
DIMGR	Distributed Information Manager CSCI
DIPHW	Distribution and Ingest Peripheral Management HWCI
DMGHW	Data Management HWCI
DMS	Data Management Subsystem
DMWG	Data Management Working Group
DP	Data Provider
DPR	data processing request
DPREP	Science Data Preprocessing CSCI
DPS	Data Processing Subsystem
DRPHW	Data Repository HWCI

DSS	Data Server Subsystem
ECS	EOSDIS Core System
EDC	EROS Data Center
EDHS	ECS Data Handling System
EDOS	EOS Data and Operations System
EOS	Earth Observing System
EOS-AM	EOS Morning Crossing (Descending) Mission -- see AM-1
EOSDIS	Earth Observing System Data and Information System
EROS	Earth Resources Observation System
ESDIS	Earth Science Data and Information System (GSFC)
ESDT	Earth science data types
ESN	EOSDIS Science Network (ECS)
FDDI	fiber distributed data interface
FDF	flight dynamics facility
FDFEPHEM	FDF-generated definitive orbit data
FGDC	Federal Geographic Data Commuittee
FK	Foreign Key
FOO	Flight of Opportunity
FOS	Flight Operations Segment (ECS)
GB	gigabyte (10^9)
GNU	(recursive acronym: “GNU’s Not Unix”); a project supported by the Free Software Foundation dedicated to the delivery of free software
GPCP	Global Precipitation Climatology Project
GPCP	Global Precipitation Climatology Project
GPI	GOES Precipitation Index
GRIB	GRid In Binary
GSFC	Goddard Space Flight Center
GTWAY	Version 0 Interoperability Gateway CSCI
GUI	graphic user interface

GV	ground validation
HDF	hierarchical data format
HDF-EOS	an EOS proposed standard for a specialized HDF data format
HIPPI	high performance parallel interface
HMI	human machine interface
HTML	HyperText Markup Language
HTTP	Hypertext Transport Protocol
HWCI	hardware configuration item
I&T	integration and test
I/F	interface
I/O	input/output
ICD	interface control document
ICLHW	Ingest Client HWCI
ID	identification
IDE	Interactive Development Environments
IDG	Infrastructure Development Group
IDR	Incremental Design Review
IERS	International Earth Rotation Service
IMS	Information Management System (obsolete ECS element name)
INGST	Ingest Services CSCI
IOS	Interoperability Subsystem
IP	international partners
IR-1	Interim Release 1
IRD	interface requirements document
ISO	International Standards Organization
ISS	Internetworking Subsystem
IV&V	independent verification and validation
JPL	Jet Propulsion Laboratory
L0-L4	Level 0 (zero) through Level 4

LaRC	Langley Research Center (DAAC)
LIM	local information manager (SDPS)
LIMGR	Local Information Manager CSCI
LIS	Lightning Imaging Sensor
LSM	local system management (ECS)
MB	megabyte (10^6)
MDT	mean downtime
MDT	mean downtime
MFLOPS	mega (millions of) floating-point operations (10^6) per second
MISR	Multi-Angle Imaging SpectroRadiometer
MODIS	Moderate-Resolution Imaging Spectrometer
MOPITT	Measurements of Pollution in the Troposphere
MSFC	Marshall Space Flight Center
MSS	Management Support Subsystem
MTBF	mean time between failure
MTPE	Mission to Planet Earth
MTTR	mean time to restore
N/A	not applicable
NAS	National Academy of Science
NASA	National Aeronautics and Space Administration
NESDIS	National Environmental Satellite Data and Information Service
NMC	National Meteorological Center (NOAA)
NOAA	National Oceanic and Atmospheric Administration
NSIDC	National Snow and Ice Data Center (DAAC)
O/A	orbit/altitude
ODC	other data center
OSI	Open System Interconnect
PDPS	Planning and Data Processing Subsystem
PDR	Preliminary Design Review

PDS	production data set
PGE	Product Generation Executive
PGS	Product Generation System (obsolete ECS element name) (ASTER)
PK	Primary Key
PLANG	Production Planning CSCI
PLNHW	Planning HWCI
PLS	Planning Subsystem
POSIX	Portable Operating System Interface for Computer Environments
PR	Precipitation Radar (TRMM)
PRONG	Processing CSCI
QA	quality assurance
RMA	reliability, maintainability, availability
RTF	rich text format
SAA	satellite active archive
SAGE	Stratospheric Aerosol and Gas Experiment
SCF	Science Computing Facility
SDP	Science Data Processing
SDPF	Sensor Data Processing Facility (GSFC)
SDPS	Science Data Processing Segment (ECS)
SDPTK	SDP Toolkit CSCI
SDSRV	Science Data Server CSCI
SeaWIFS II	Sea-Viewing Wide Field-of-View Sensor II
SFDU	Standard Format Data Unit
SMC	System Management Center (ECS)
SPRHW	Science Processing HWCI
SRS	software requirements specification
SSM/I	Special Sensor for Microwave/Imaging (DMSP)
SST	sea surface temperature
STMGMT	Storage Management

STMGT	Storage Management Software CSCI
SUBSRV	Subscription Server
TMI	TRMM Microwave Image
TOMS	Total Ozone Mapping Spectrometer
TONS	TDRS On-board Navigational System
TRMM	Tropical Rainfall Measuring Mission (joint US-Japan)
TSDIS	TRMM Science Data and Information System
USNO	US Naval Observatory
UT	universal time
UTC	universal time code
V0	Version 0
VIRS	Visible Infrared Scanner (TRMM)
WAIS	Wide Area Information Server
WKBCH	Workbench CSCI
WKSHW	Working Storage HWCI
WWW	World-Wide Web