3 Guidance on Building a Physical Database from the SDP

3.1 Conceptual Data Reference Model as a Framework for Implementation Methods

As described in Guidance Part 2 volumes, the SDP Project used a system engineering approach for developing user driven requirements for schedule and related data. A set of Use Cases on Integrated Trip Planning, Dynamic Generation and Presentation of Public Timetables, and Generation of Ad Hoc Scheduling were developed to identify the specific schedule data requirements for these downstream applications. The effort wanted to ensure that the data meaning and relationships were well defined. A Conceptual Data Reference Model (CDRM) was developed to meet that need. The CDRM is meant to be used as a framework to unambiguously describe the SDP data concepts and their relationship to each other. Different technical methods may be used to physically represent and store schedule data. The three methods include: logical data model, physical database, and XML Schema. The major objective of the project was to implement the XML Schema and validate the requirements in one or more downstream applications (related to the use cases).

The SDP logical and physical methods provide alternative approaches for the storing the data. Different needs may exist for storing the data using a different method, yet ensuring that there is a seamless, automated way to transfer the data from format to format. In fact, the Data Mapper (csv2xml) application, described in Chapter 5 of this document, uses a comma separated value (csv) representation which may be described as a logical model of the CDRM to convert native data using the interim format to an XML document. A physical database is a more efficient way to store data sets of different versions and multiple agencies together then file server containing multiple SDP XML Documents.

3.2 Differences between the SDP CDRM and Implementation Methods

The SDP CDRM uses an entity-relationship (ER) method to represent real-world phenomena. The model is driven by a set of requirements described by current, local practice and by best practices advocated by the information technology and transit industries. The CDRM uses an abstract ER diagram (ERD) modeling method to represent these real-world phenomena. Although a similar notation is used to describe the Logical and Physical models, they do not include the same information or serve the same purpose. Differences between the CDRM and Logical, CDRM and Physical and CDRM and XML Schema models are described below.

Difference between a CDRM and SDP Logical Entity Relationship Model: A CDRM shows the relationship between entities, but does not carry related keys to related entities. For example, in a system that supports more than one schedule version per agency, the schedule version identifier must be included in every entity in the logical model. A *logical model (expressed as an ERD)* shows these primary and foreign keys, and thus describes key storage requirements related to the data set. The CDRM makes no assumptions about how a model is applied; rather, it describes real-world relationships.

Difference between a CDRM and SDP Physical Implementation: Similar to the relationship between a conceptual and logical model, the physical implementation supports primary and foreign keys, the procedures that validate these relationships, and specific formats and syntax related to each data type described in the model. Specifically, these rules and procedures are defined for a specific database management system such as Oracle 9i, MS Access 2003, etc.

Difference between a CDRM and SDP XML Schema Implementation: The SDP XML Schema's primary purpose is to facilitate the sharing of data across different information systems, particularly via the Internet. The SDP XML Schema uses the CDRM to describe schedule and related data concepts and preserve the relationship requirements among data concepts for one schedule version and for a single transit operator. A set of rules were used to migrate the data concepts from the CDRM to the XML Schema implementation.

3.3 An Example of Migrating the CDRM to a Logical, Physical and XML Schema Representation

As described above, there are rules for implementing the CDRM from the conceptual framework to its logical, physical and XML schema formats. The following sections show how the CDRM for the same data concept, Schedule Calendar Date, is transformed to a logical, physical, and XML Schema model. Each model depicted in Figures 1 through 4, is summarized in the list below:

- Conceptual Data Reference Model—Figure 1
  - Note the CDRM, Logical and Physical models are all represented using ERD notation.
- Logical ERD Model—***Error! Reference source not found.*** and Figure 3
  - Note the key identifiers become primary keys (pk), and related entities include related or foreign keys (fk);
- Physical Model—Figure 4. Note the attributes are specified with specific data types that reference the specific database management system specifications.

3.3.1 Example of the Conceptual Data Reference Model

The CDRM is expressed as an Entity Relationship Diagram (ERD). Figure 1 shows an example for the basic representation of the schedule calendar date concept.

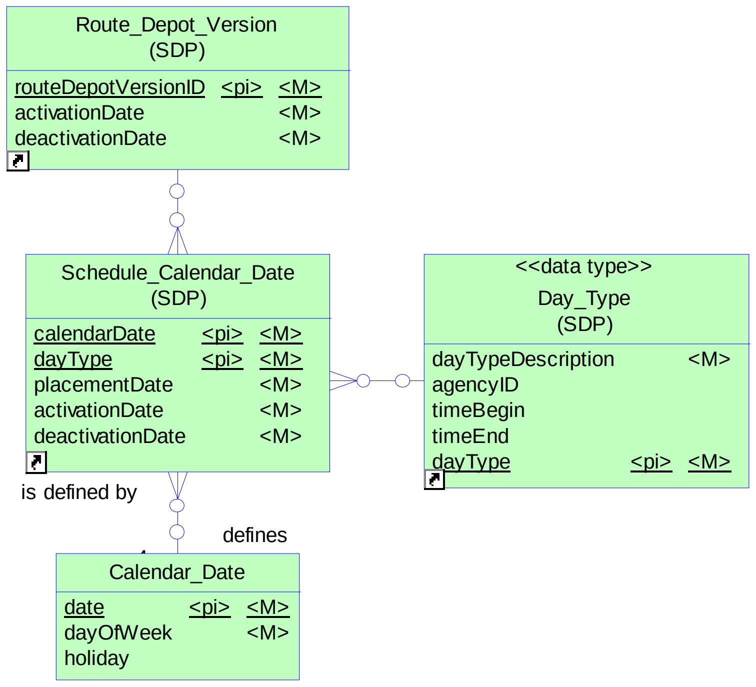


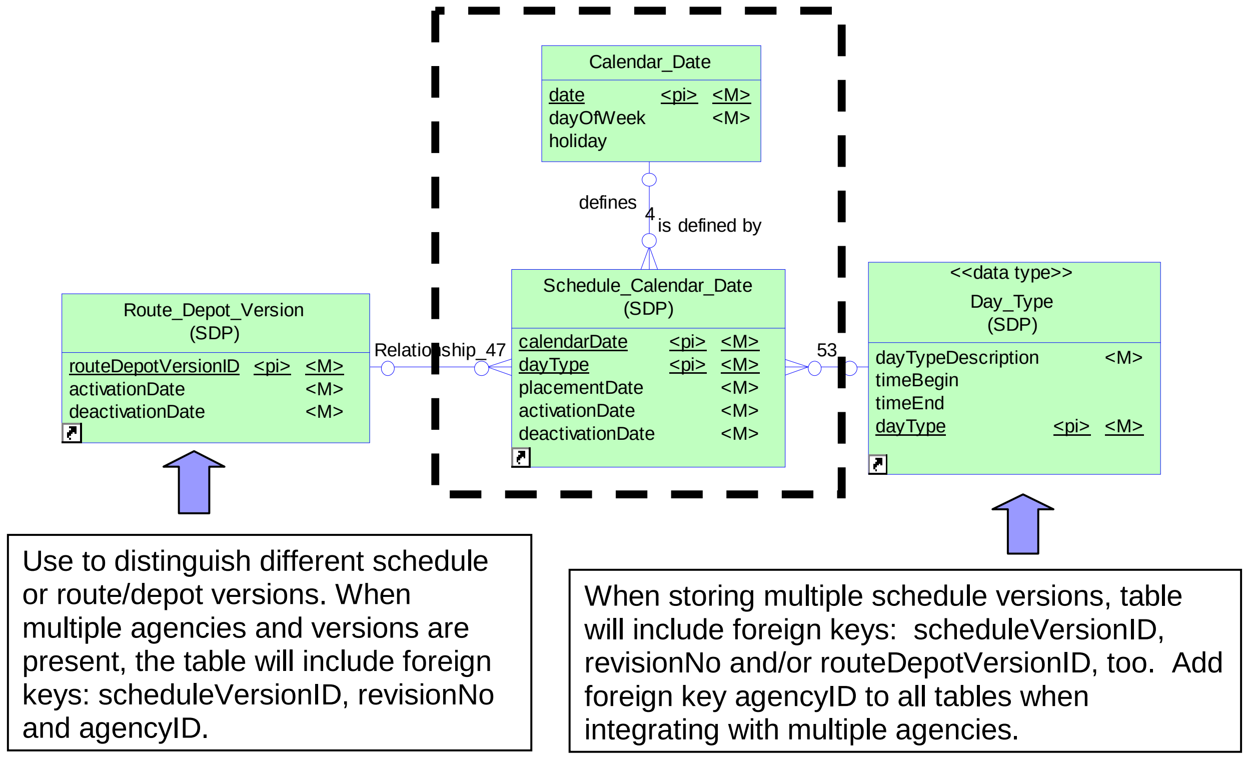
Figure 1: Conceptual ER Model of Schedule Calendar Date Concept

The following paragraph describes the requirements of Figure 1:

> "Transit service is scheduled for each day of operation. Service components may be scheduled to operate on different dates depending on a number of factors. These factors may be schedule based; for example, special trips are designated when there is an event at Shea Stadium or service to evacuate workers from the city during a snow storm. The Schedule Calendar Date associates the relevant schedule components (designated by the Route Depot Version) and an index related to the appropriate trips (designated by the day type) into a table which is used as a reference.
>
> "A Schedule Calendar Date is created for each set of schedule version components and the trips that operate on the specific dayType. In some cases, the schedule version components are scheduled for only part of a day, for example, the schedule components vary when the Mets play games that begin at 5 p.m. versus at 7 p.m." [from *SDP Functional Requirements*, p. 102]

3.3.2 Example of the Logical Entity-Relationship Representation

The logical model is driven by application requirements related to how the data are stored and accessed. In the example illustrated in **Error! Reference source not found.**, the Schedule Calendar Date entity inherits related keys designating the schedule version when more than one schedule version is present. When an organization changes its schedule mid-version, the entity is required to include a revision number, and when a transit agency issues their schedule by route or by route and depot, the route-depot version is also included in the entity. When this model is extended to a regional repository, each entity must designate the authority that issued the data, as such, the functional entities Route_Depot_Version, Schedule_Calendar_Date and Day_Type include the agency identifier (agencyID). The actual implementation of the conceptual to logical model may be seen in Figure 3.



Use to distinguish different schedule or route/depot versions. When multiple agencies and versions are present, the table will include foreign keys: scheduleVersionID, revisionNo and agencyID.

When storing multiple schedule versions, table will include foreign keys: scheduleVersionID, revisionNo and/or routeDepotVersionID, too. Add foreign key agencyID to all tables when integrating with multiple agencies.

Figure 2: Migrating from Conceptual to Logical Model

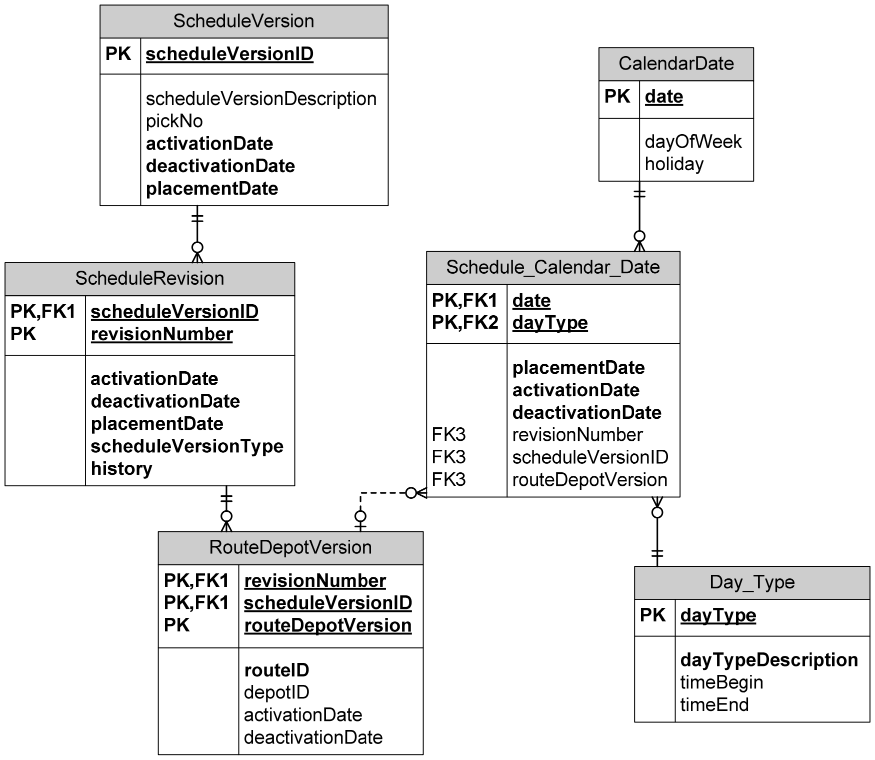


Figure 3: Logical Model of Schedule Calendar Date Concept

3.3.3 Example of the Physical Database Implementation

The physical model is similar to the logical model except the data types are defined by the database management system. The physical database also supports procedures that enforce referential integrity triggers (primary and foreign keys) when data are added, changed or deleted from the database. A generic physical model for the Schedule Calendar Date concept is illustrated in Figure 4. As is shown in the figure, this is similar to the logical model shown in Figure 3 **except** for defining specific data types and showing the procedures. For organizations with specific database management systems, the logical and physical representations are somewhat redundant since logical and physical models will use the same data type definitions.

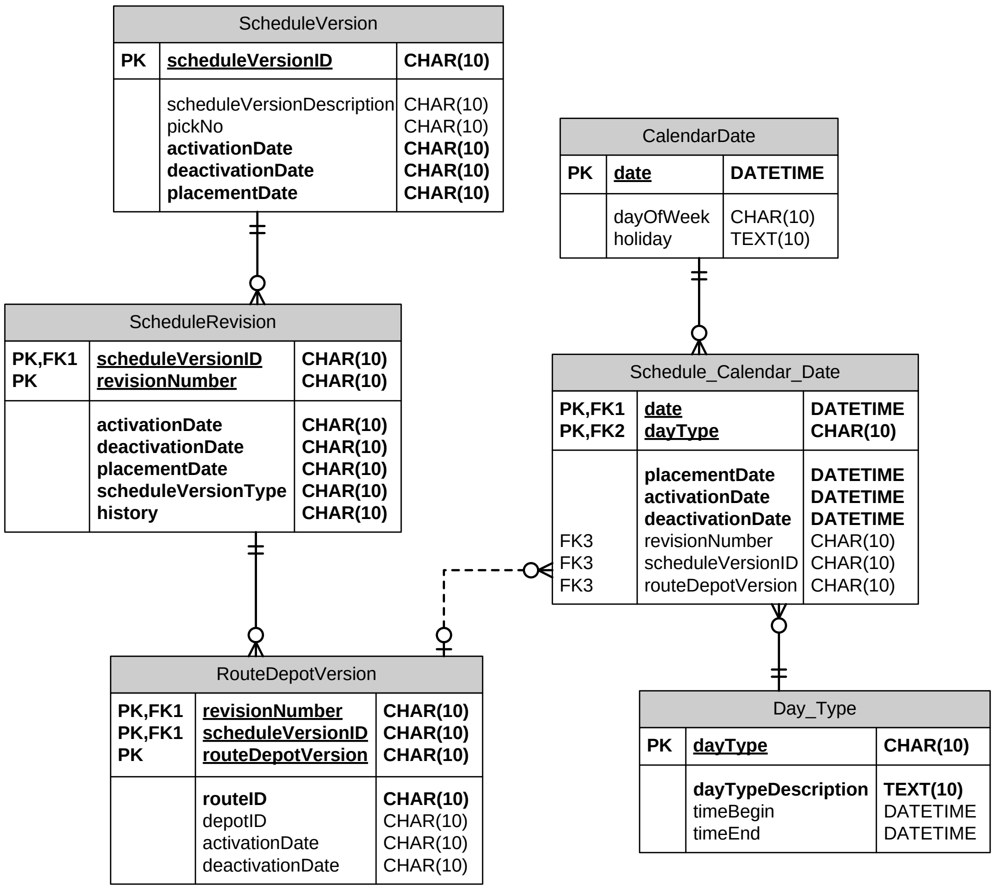

ScheduleVersion
PK scheduleVersionID CHAR(10)
scheduleVersionDescription CHAR(10)
pickNo CHAR(10)
activationDate CHAR(10)
deactivationDate CHAR(10)
placementDate CHAR(10)
CalendarDate
PK date DATETIME
dayOfWeek CHAR(10)
holiday TEXT(10)
ScheduleRevision
PK,FK1 scheduleVersionID CHAR(10)
PK revisionNumber CHAR(10)
activationDate CHAR(10)
deactivationDate CHAR(10)
placementDate CHAR(10)
scheduleVersionType CHAR(10)
history CHAR(10)
Schedule_Calendar_Date
PK,FK1 date DATETIME
PK,FK2 dayType CHAR(10)
placementDate DATETIME
activationDate DATETIME
deactivationDate DATETIME
FK3 revisionNumber CHAR(10)
FK3 scheduleVersionID CHAR(10)
FK3 routeDepotVersion CHAR(10)
RouteDepotVersion
PK,FK1 revisionNumber CHAR(10)
PK,FK1 scheduleVersionID CHAR(10)
PK routeDepotVersion CHAR(10)
routeID CHAR(10)
depotID CHAR(10)
activationDate CHAR(10)
deactivationDate CHAR(10)
Day_Type
PK dayType CHAR(10)
dayTypeDescription TEXT(10)
timeBegin DATETIME
timeEnd DATETIME

Figure 4: Physical Model of the Schedule Calendar Date

3.4 Database Scripts and Referential Integrity Issues

Using specialized data modeling software, the CDRM may be used to generate a physical database loading script. The physical database should be optimized for its use. Because primary and foreign keys are not specifically identified in the conceptual model, tables are not optimized for queries, and multiple agencies or versions are not assumed in the CDRM, the script will need to be revised and augmented. For inquiries on obtaining a script to generate a SDP physical database, please contact: tsdea@consystec.com.

3.4.1 Referential Integrity Issues

Referential integrity in a database ensures that tables ensures the identity and relationships among data concepts are unique, consistent and unambiguous. The utility to ensure these characteristics are the assignment of primary and foreign keys in the physical database management system. Although the CDRM does not use the terms primary and foreign keys, it classifies one or more identifying keys (primary keys) for each entity (which becomes a table in the physical database). The model also inserts related identifying keys in some entities and in some cases there are non-identifying keys in an entity. The CDRM entity, their identifying keys, related identifying keys and non-identifying keys are listed in Table 1: CDRM Entity with its Identifying and Non-Identifying Keys.

Table 1: CDRM Entity with its Identifying and Non-Identifying Keys

IDENTIFYING KEY NAME	IDENTIFYING RELATED KEY NAME(S)	OTHER RELATED KEYS	ENTITY
agencyID		effectiveDate endDate	Agency
amenityID		locationID effectiveDate endDate	Amenity
amenityCode			Amenity_Type
blockID		scheduleVersionID effectiveDate endDate	Block
blockTime	blockID	scheduleVersionID	Block_Event_Time
seqNo	tripID routeID	scheduleVersionID	Block_Trip_Sequence
date			Calendar_Date
connectionID	From: locationID To: locationID	effectiveDate endDate	Connection_Seg
dayType		dateTimeBegin dateTimeEnd scheduleVersionID	Day_Type
depotID		transitFacilityID	Depot

IDENTIFYING KEY NAME	IDENTIFYING RELATED KEY NAME(S)	OTHER RELATED KEYS	ENTITY
		locationID effectiveDate endDate	
connectionNum	To: tripTime, tripID, routeID From: tripTime, tripID, routeID	scheduleVersionID	Event_Connection
facPCID	transitFacilityID		Facility_Plant_Component
locationID		featureType_cd effectiveDate endDate	Location
mode			Mode
noteAssociationID	noteID tripID	tripTime scheduleVersionID effectiveDate endDate	Note_Association
noteID		scheduleVersionID effectiveDate endDate	Note_Entry
organizationUnitID		agencyID effectiveDate endDate	Organizational_Unit
passAccessID		locationID effectiveDate endDate	Passenger_Access_Component
passengerAccessCode			Passenger_Access_Type
patternID	routeID	routeDirection scheduleVersionID effectiveDate endDate	Pattern
plantCompID	componentID (amenityID, transitFacilityID, stopID, trackID, passAccesID) featureType_cd	effectiveDate endDate	Plant_Component
	trackNo stopID		Platform_Track
portalID		locationID effectiveDate endDate	Portal

IDENTIFYING KEY NAME	IDENTIFYING RELATED KEY NAME(S)	OTHER RELATED KEYS	ENTITY
relativeLocationID	locationID		Relative_Location
routeID	routeBeginDate routeEndDate	mode scheduleVersionID effectiveDate endDate	Route
routeDepotVersionID	revisionNo scheduleVersionID	dayType effectiveDate endDate	Route_Depot_Version
routeDirection	routeID	scheduleVersionID	Route_Direction
routeGroupingID		scheduleVersionID effectiveDate endDate	Route_Grouping
routeGroupingCode			Route_Grouping_Type
calendarDate			Schedule_Calendar_Date
revisionNumber	scheduleVersionID		Schedule_Revision
scheduleVersionID	activationDate deactivationDate	agencyID	Schedule_Version
scheduleVersionType			Schedule_Version_Type
identifier		locationID	Service_Area
revisionNo	plantCompID	activationDate deactivationDate placementDate modificationDate creationDate	Status
statusTypeCode			Status_Code_Type
timepointID		locationID scheduleVersionID effectiveDate endDate	Timepoint
tth	routeID routeDirection	scheduleVersionID	Timetable_Header
trackNo		effectiveDate endDate	Track
transferClusterName		Set of locationID effectiveDate endDate	Transfer_Cluster
transitFacilityID		locationID effectiveDate endDate	Transit_Facility
tranPathID	locationID	Ordered list of locationID	Transit_Path

IDENTIFYING KEY NAME	IDENTIFYING RELATED KEY NAME(S)	OTHER RELATED KEYS	ENTITY
		effectiveDate endDate	
seqNo	locationID patternID		Transit_Path_Event
seqNo	locationID (first & last) patternID	Ordered list of locationID	Transit_Point_Event
stopID		locationID effectiveDate endDate	Transit_Stop
tripID	routeID dayType	scheduleVersionID effectiveDate endDate	Trip
tripEventTypeCode			Trip_Event_Type
tripTime	tripID routeID	scheduleVersionID	Trip_Time

3.4.2 Temporal Integrity Issues

Transit schedule and related data are composed of sets of data with differing, overlapping and temporary data versions and revisions in different states. To that end, temporal data principles are applied to the major entities using two fields for time: effectiveDate and endDate. The principles for preserving the history while enabling so-called redundant data to remain in the database necessitate including the two dates as primary keys. In some cases, like ScheduleVersion, Route, Day Type, Plant Component Status, additional temporal keys are included (e.g., activationDate/deactivationDate, routeBeginDate/routeEndDate) in order to allow for temporary data with similar identifying keys to remain in the

For example, TriMet implemented this approach in their database. The description of their use of dates is described in a passage from the FTA *Best Practices for Using Geographic Data in Transit: A Location Referencing Guidebook*:

> "…The Location Table cannot lose the old record keyed to the Location ID; therefore, a method to manage the updated records must be implemented.
>
> "Adding one attribute to the unique identifier (primary key) of the Location Table, called location_end_date, can achieve this consistency. When a location ID is first introduced, the location_begin_date is set to the date of first use (see Example Table #1 below: 5-1-03 is the new date). The location_end_date is set to a date in the far-off future (in Example Table #1, location_end_date is 12-31-9999). If a change is made to one of the Location Table attributes, for example, if the Public_Description is changed, the location_end_date for the existing row is set to the date of closure (in Example Table #2, the date is now 6-30-03). In addition, a new row (or record) may be introduced for the

Location ID. On the new record, the location_begin_date value is set for the original location_end_date plus one day later (7-01-03, and the location_end_date is set to a date in the far-off future (12-31-9999). Using this approach, a consistent set of values is provided for the Location ID with the referring transit feature matching the Location ID, and inclusive of location_begin_date and location_end_date.

“Example of the Public_Description changing:

Example Table #1

Loc_ID	location_begin_date	location_end_date	Public_Description
5	5-1-03	12-31-9999	Union & 5th

Example Table #2

Loc_ID	location_begin_date	location_end_date	Public_Description
5	5-1-03	6-30-03	Union & 5th
5	7-1-03	12-31-9999	MLK& 5th

“In this manner, historical data can be matched accurately. Alternatively, if the location changes (e.g., nearside to farside), a new ID (and new record) is created and the old ID is retired.”
[from FTA-NJ-26-7044-2003.1, *Best Practices for Using Geographic Data in Transit: A Location Referencing Guidebook* , April 2005. pf., 79. free download from http://www.fta.dot.gov/documents/LRG_FinalPublication.pdf]

Many of the principles for managing temporal databases may be found in:

Richard T. Snodgrass, *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers, San Francisco CA, 2000. – free download at http://www.cs.arizona.edu/people/rts/tdbbook.pdf

3.5 Implementation of a SDP Database in MS Access

The SDP demonstration contains a sample Microsoft Access database instance of the SDP (Sample SDP database). The referential integrity procedures and triggers are not installed in this database. The database executes a macro at startup that reads a set of SDP csv file format files at run-time and loads the data into the database. The user may then view table data and create new queries to examine the SDP data and structure.

3.5.1 Software Installation

The Sample SDP database contains a startup macro application written in Visual Basic for Access that runs under the Windows Operating System. Table 2 outlines the operating system and version requirements for the database instance example.

Table 2: SDP Access Database Operating System and Version Requirements

System	Version
SDP XML Schema	Version 1_0
Microsoft Access	2003 SP2
MS Windows	Windows XP

Prerequisites: Microsoft Windows and Access 2003 Database.

The TSDEA web site (http://www.consystec.com/tsdea/rstwg/docs.html) contains a copy of the Microsoft Access 2003 database content file, but not the database software itself.

3.5.2 Data Files

3.5.2.1 Set Up the SDP Directory Structure

The figure below shows where to install the SDP instance database, called sdp_db.mdb. You may choose any SDP root path (directory). We have labeled this SDP root as $SDP in the figure below and use this label in the remainder the user's manual.

If you are copying files from the CD, the directory structure is set up as described in Figure 5.

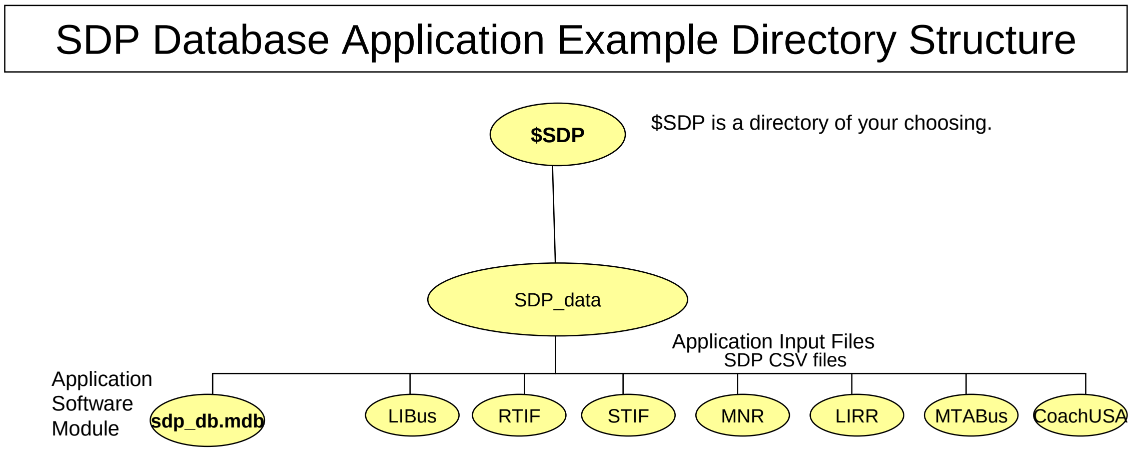

SDP Database Application Example Directory Structure
$SDP
$SDP is a directory of your choosing.
SDP_data
Application Input Files
SDP CSV files
Application Software Module
sdp_db.mdb
LIBus
RTIF
STIF
MNR
LIRR
MTABus
CoachUSA

Figure 5: SDP Database Application Directory Structure

The Sample SDP Database Instance is located in the SDP_data sub-directory of the $SDP directory. We will use a forward-slash notation to indicate sub-directory relationships of directory tree, for example, $SDP/SDP_processing.

SDP CSV data files, which can be imported to the database at run-time are located in an agency-specific directory of $SDP/SDP_data/, for example, $SDP/SDP_data/LIBus for Long Island Bus, and $SDP/SDP_data/RTIF for New York City Transit Rail files.

3.5.2.2 List of SDP CSV Data Input Files

The Sample SDP database reads CSV data from an agency-specific sub-directory of $SDP/SDP_data/"agency". The list of files read depends on the agency and whether the agency provides bus, commuter rail, or subway, etc. service. The list of files, using the LIBus as an example, is listed below:

- sdp_agency.csv
- sdp_direction.csv
- sdp_location.csv
- sdp_Note.csv
- sdp_NoteTimeAssoc.csv
- sdp_NoteTripAssoc.csv
- sdp_pattern.csv
- sdp_patternEventList.csv
- sdp_relativeLocation.csv
- sdp_route.csv
- sdp_rtDepotVersion.csv
- sdp_RtDirection.csv
- sdp_scheduleRevision.csv
- sdp_scheduleVersion.csv
- sdp_stop.csv
- sdp_TimeEventType.csv
- sdp_timepoint.csv
- sdp_trips.csv
- sdp_tripTimes.csv

3.5.2.3 List of SDP Tables

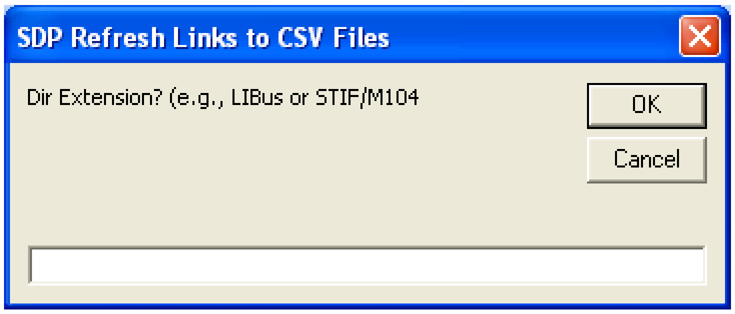
The output of the initialization macro is a series of tables with the same names as the csv files imported.

3.5.3 Application Execution

AutoExec Macro Application Execution:

- AutoExec Macro: At startup the sdp_db.mdb database executes a software module that prompts the user for a name of a sub-directory where SDP csv files are located: for example, libus, rtif, stif, mnr, lirr, mtabus, etc.

An example of the prompt display is shown in Figure 6:


SDP Refresh Links to CSV Files

Dir Extension? (e.g., LIBus or STIF/M104

Figure 6: SDP Startup Macro Initialization Directory Prompt

Once the data files are loaded the application will display “Done!”.

To launch the MS Access database, navigate to the $SDP/SDP_data directory. If you are using a graphical interface to navigate to the $SDP/SDP_data directory, then double-click on the file called sdp_db.mdb to start the application.

After startup MS Access will contain a list of tables that correspond with the csv files imported. This is shown in Figure 7:

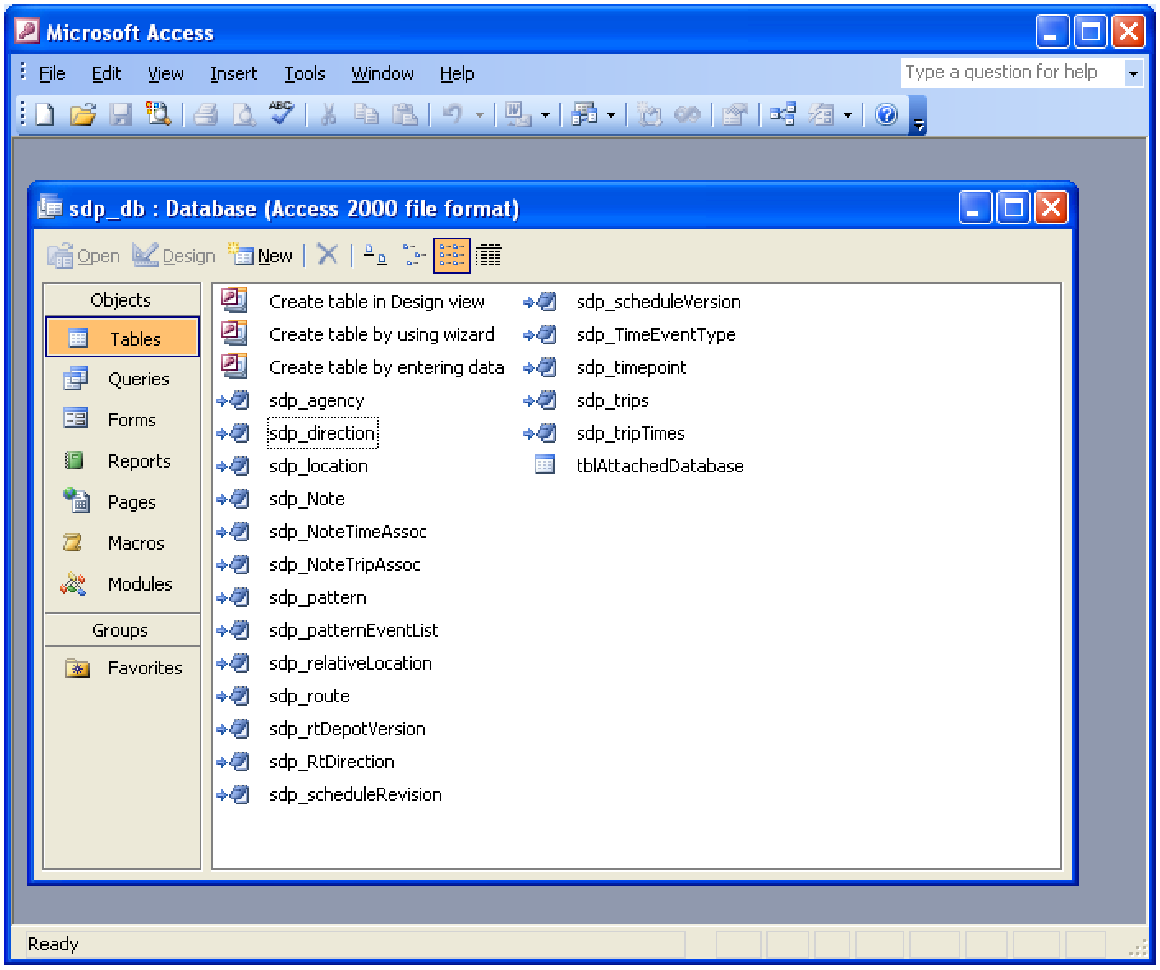


Figure 7: SDP Startup Macro Initialization Directory Results