DATABASE DESIGN OF OHIO SPS TEST

A Thesis Presented to
The Faculty of the
Department of Civil Engineering
and Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirement for the Degree
Master of Science

by
Jiayan Liu

August, 1997
I would like to thank Dr. Shad M. Sargand and the Ohio University Civil Engineering Department for providing me the opportunity to attend graduate school. I would also like to thank Dr. Shad M. Sargand, Dr. Glenn A. Hazen and Dr. Larry E. Snyder for being on my thesis examination committee. Without their help, this thesis would never have been done. Issam Khoury, Paulette Cook, and other students of Civil Engineering Department have also given me invaluable assistance in the making of this thesis. A special appreciation is extended to David J. Beegle. His extreme patience and insightful instruction gave me great help during this work.

Moreover, I have to thank Yanling Zhang’s editing help on this paper.

Finally, I would like to thank my family and my husband, Kanyu, for giving me unbounded support and love, which indirectly makes this thesis possible.
# Contents

1 Introduction .......................... 1
1.1 Overview .............................................. 1
1.2 Objective ............................................. 2
1.3 Outline .............................................. 3

2 SPS Description ....................... 5
2.1 Introduction to SPS and Site Description .............................................. 5
2.2 SPS Projects Description ............................................. 6
2.3 Instrumentation ............................................. 7
2.4 Data Acquisition ............................................. 9
2.5 Test Procedure ............................................. 9

3 Basic Concepts ......................... 11
3.1 Database and Relational Database ............................................. 11
3.2 Client-Server Architecture ............................................. 14

4 Database Design Process ............. 17
4.1 Requirements Collection and Analysis ............................................. 18
   4.1.1 Requirements ............................................. 18
4.1.2 Data Dictionary .................................................. 18
4.2 Conceptual Database Design ................................. 19
  4.2.1 Conceptual Schema Design .............................. 19
  4.2.2 Transaction Design ..................................... 21
4.3 Choice of DBMS ................................................. 24
4.4 Logical Database Design .................................. 25
  4.4.1 System-independent Mapping .......................... 25
4.5 Physical Database Design .................................. 26
  4.5.1 Using DLL Statements ................................. 26

5 SPSDB System Implementation ............................... 29
  5.1 Building Basetable ........................................ 29
  5.2 Building Interface ......................................... 30
    5.2.1 Introduction to Interface Design Tool - Oracle Forms 4.5 30
    5.2.2 Requirements of Interface ......................... 31
  5.3 Structural Design of Canvas-Views ..................... 32
  5.4 Interface Coding ........................................ 33
    5.4.1 Relation Algebra .................................. 33
    5.4.2 PL/SQL code ........................................ 36
  5.5 Examples .................................................. 37
    5.5.1 How to Realize Retrieval Functions ............... 37
    5.5.2 How to Make Graphics Interactive ............... 43
    5.5.3 How to Build a Menu ................................ 44
<table>
<thead>
<tr>
<th>Page</th>
<th>6 User Manual of SPSDB</th>
<th>7 Conclusions and Discussions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A Objects of Form</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B C code for Transforming the Format of Raw Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C PL/SQL Codes in Oracle Forms 4.5</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Information of instrumentation installation ................................ 8

3.1 Operations on relations ...................................................... 13

4.1 Data Dictionary of SPSDB database ........................................ 20

4.2 Storage space of different data types ....................................... 28
List of Figures

2.1 Locations of SPS-1, SPS-2, SPS-8 and SPS-9 ................................. 6
2.2 Instrumentation information of sections in SPS ............................. 7
3.1 An example of relation (table) ................................................. 12
3.2 The client-server architecture in SPSDB system ............................ 15
4.1 Phases of SPSDB database design ............................................. 17
4.2 The ER (Entity-Relationship) schema of the SPS data .................... 21
4.3 Flow chart of transaction 4 ................................................... 23
4.4 ER-to-Relational Mapping ..................................................... 25
5.1 Canvas-View navigation design ................................................. 32
5.2 PL/SQL code structure .......................................................... 36
5.3 Retrieval window ............................................................... 37
5.4 Query key section of retrieval window ...................................... 38
5.5 Example of using properties, canvas-view editor, and PL/SQL editor to build a button ................................................................. 39
5.6 Test block .............................................................................. 40
5.7 Truck block ........................................................................... 40
5.8 Sensor-Data block ................................................................... 41
Chapter 1

Introduction

1.1 Overview

The behavior of subgrade soils and paving materials under the load of pavement greatly
influences the performance of the pavement. Therefore, a study of soil mechanics and
pavement material becomes both necessary and essential.

In the early stages of development, pavement design was based only on past experi-
ences. After a profound knowledge of structural properties of soil was developed, a rational
basis was established in pavement design. During that time soil mechanics was applied
to the design as well. The Ohio Strategic Highway Research Program Specific Pavement
Studies Test Road-Instrumentation Plan [11] indicates that the usual method of today’s
pavement design combines standard traffic loads numerically with assumed thickness and
material properties of layer to predict the structural response of a pavement system.
However, as there are too many factors influencing the performance of the pavement, the-
oretical analysis and empirical laboratory results are not enough. Moreover, assumptions
are sometimes far different from actual conditions, which can lead to the failure of a whole
design.

Therefore, pavement performance measured under actual field conditions is critical
regardless of the procedures used for the design. It is important to acquire the data of
strain, displacement, and pressure of a pavement system under known conditions, so that
dynamic loading and layer thickness can be determined directly. Physical properties of the
pavement layers must be obtained in the laboratory under the same conditions experienced
in the field. To estimate these properties at the time dynamic response measurements are
being conducted, temperature, moisture content, and frost depth must be monitored in
the pavement and combined with results obtained in the laboratory [11].

The Strategic Highway Research Program (SHRP) is a nation-wide research program.
This program developed some invaluable experimental data concerning structural changes
related to the environment, load, and distress over time.

1.2 Objective

The most valuable part of the SHRP is the obtained experimental data. How to organize
and apply these test data is critical for this research. As the large amount of test data
increase, manual management may be impossible or, at least, ineffective. A file system or
a database management system would then be more effective for storing, retrieving, and
updating these data.

The Database Management System (DBMS) should be chosen rather than a traditional
file system. This decision is based on several considerations.

1. There are several relationships in this system. For example, in each test, trucks are
used to simulate traffic loading and response data of the pavement acquired from
sensors. There are relationships among tests, trucks and data, and the DBMS is
much more powerful in dealing with these relationships than is a traditional file
system.
2. The DBMS avoids storing redundancy. In a traditional file system, truck information is stored repeatedly for several tests if the same truck is used during an experiment. However in DBMS, a common attribute can be used to join two relations together. The “Truck_ID” is stored in the “SPSPACE_TEST” table as a reference to locate the corresponding records in the “SPS_TRUCK” table. The information for each truck needs to be stored only once, and hence, to reduce the storing redundancy.

3. The DBMS handles data insertion, deletion, and modification more easily than does a file system.

4. The volume of data is large and can be larger by more input of data in the future.

In short, it is more efficient to store and organize test data with a Database Management System (DBMS) than without.

1.3 Outline

- Chapter 2 describes the SPS project, including instrumentation, test procedure, test acquisition, and test data analysis.

- Chapter 3 describes basic concepts of distributed database systems.

- Chapter 4 describes in detail the design procedures of the SPS database (SPSDB). Requirement analysis, conceptual design, choice of a DBMS, and logical and physical design will be discussed.

- Chapter 5 describes implementation of the SPSDB system, including how to establish basetables of the SPSDB server, how to build interfaces of the SPSDB clients by
Oracle Forms 4.5 and how to make the interface interactive with the server by adding a trigger to items. Examples are presented.

- Chapter 6 is a user manual for the SPSDB.

- Chapter 7 includes conclusions and discussions.
Chapter 2

SPS Description

2.1 Introduction to SPS and Site Description

Specific Pavement Studies (SPS), which are parts of the Strategic Highway Research Project (SHRP), consist of nine experiments. Four of them are funded by the Ohio Department of Transportation and the Federal Highway Administration. Six universities in Ohio are participating in this project.

The experiment site is located on a three-mile stretch of U.S. 23, about 25 miles north of Columbus in Delaware County, as seen in Figure 2.1 [12]. The reasons for selecting this site are that the topography, soil and climate are uniform, and that the performance of pavement sections can be compared with each other.

Four experiments are conducted in the 38 test sections. In 33 test sections, seasonal and dynamic response instrumentations are installed in different layers of the highway. During the test, a falling weight deflectometer and controlled vehicle loadings are used to simulate actual traffic loads, while data acquisition systems are utilized to gather response data from a variety of sensors in these sections. These obtained data are then analyzed by engineers and civil engineering researchers aiming to learning the effects of climate and cumulative traffic loadings on pavement performance.
2.2 SPS Projects Description

The four experiments in SPS are as follows.

- SPS-1
  Strategic Study of Structural Factors for Flexible Pavement

- SPS-2
  Strategic Study of Structural Factors for Rigid Pavement

- SPS-8
  Study of Environmental Effects in the Absence of Heavy Traffic (PCC and AC)

- SPS-9
  Asphalt Program Field Verification Studies

Each experiment includes a number of sections. Refer to Figure 2.2 [12].
Figure 2.2: Instrumentation information of sections in SPS

2.3 Instrumentation

The information of instrumentation installation can be seen in Table 2.1.

The specific measured parameters are listed as follows:

- Pavement response parameters:
  - Strain
  - Pressure
  - Displacement
  - Joint Opening

- Seasonal parameters measured:
  - Temperature
  - Moisture
  - Frost Depth
Table 2.1: Information of instrumentation installation

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Section Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pavement-Seasonal Parameters</td>
<td>18</td>
</tr>
<tr>
<td>Pavement Parameters</td>
<td>15</td>
</tr>
<tr>
<td>No Instrumentation</td>
<td>5</td>
</tr>
</tbody>
</table>

- Soil Suction
- Water Table Elevation

The sensors used in the project are listed as follows:

- **Strain**
  - PMR-60 SG
  - Dynatest Past II-AC SG
  - KM100-B ST
  - Carison A-8 SM
  - VCE-4200 VWSG

- **Vertical Defection Measurement**
  - GPD 121-500 LVDT
  - GPD 121-250 LVDT

- **Temperature Measurement**
  - KM100-B ST
  - Carison A-8 SM
  - MRC Thermistor
  - VCE-4200 VWSG

- **Pressure**
  - Geokon 3500 PC
• Frost Depth
  – CRREL Resistivity Probe

• Soil Moisture
  – FHWA TDR Probe

2.4 Data Acquisition

The data acquisition system consists of a datalogger and additional equipment. A datalogger is used to collect and store data, and data acquisition systems are utilized to monitor the following two types of parameters:

• Seasonal Parameters
  – CR7 system for monitoring strain gauges and LVDTs
  – CR10 system for monitoring thermistor, resistivity, and TDR probes and vibrating wire strain gauges

• Dynamic Response Parameters
  – Megadac 5180AC system for monitoring strain gauges, LVDTs and Pressure Cells

At the experiment site, the data acquisition systems collect and store data over extended periods of time.

2.5 Test Procedure

At the test site, the programs written and tested in the CGER laboratory at Ohio University need to be downloaded to dataloggers from a computer. After dataloggers begin
working and all systems work properly, the computer can be disconnected and the data-loggers are put in a safe, dry location at the site.

Dataloggers will collect data at 30-minute intervals until the computer is reconnected and the data are transferred. The whole test will continue until enough data are acquired, which usually lasts at least ten days. After the test data are obtained, they are processed and analyzed for information on the magnitude of strains, deformations, and temperature changes in the pavement.
Chapter 3

Basic Concepts

3.1 Database and Relational Database

- Database

An integrated collection of related data. "Data" here means known facts that can be recorded and that have implicit meaning. In Ramez Elmasri and Shamkant B. Navathe's book [10], three properties of database have been defined:

- A database represents some aspect of the real world, sometimes called the miniworld or the Universe of Discourse (UoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning.
- A database is designed, built, and populated with data for a specific purpose.

- Relational Database

A type of database based on the relational model is called a relational database. The relational model is based on the data structure relation. It is a popular data model used at present and includes three components [8]:

- Relational Data Structure

A relation model represents the database as a collection of relations. "Relation"
is also called “table” in the “physical” database. Figure 3.1 is an example of what a relation or a table is.

![Figure 3.1: An example of relation (table)](image)

In relational model terminology, a row in the table is called a “tuple” and a column header is called an “attribute”. The value of each column must be a uniform data type, which is called “domain”.

- Relational Model Constraints

  Relational model constraints govern the organization of the data structures. They are as follows:

* Domain Constrains

  The value of each attribute must be in the domain of that attribute.

* Key Constrains

  • Primary Key

  An attribute or a group of attributes that uniquely identify a row in a table. As a rule, every table has one and only one primary key [8].
<table>
<thead>
<tr>
<th>Operation</th>
<th>Type</th>
<th>Resulting Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>Binary</td>
<td>Rows combined from two relations, duplicate rows eliminated</td>
</tr>
<tr>
<td>Intersection</td>
<td>Binary</td>
<td>Rows common to two relations</td>
</tr>
<tr>
<td>Difference</td>
<td>Binary</td>
<td>Rows existing in the first relation but not in the second</td>
</tr>
<tr>
<td>Projection</td>
<td>Unary</td>
<td>Rows containing some of the columns from the source relation</td>
</tr>
<tr>
<td>Selection</td>
<td>Unary</td>
<td>Rows from source relation that meet query criteria</td>
</tr>
<tr>
<td>Product</td>
<td>Binary</td>
<td>Concatenation of every row in one relation with every row in another</td>
</tr>
<tr>
<td>Join</td>
<td>Binary</td>
<td>Concatenation of rows from one relation and related rows from another</td>
</tr>
</tbody>
</table>

Table 3.1: Operations on relations

- **Foreign Key**

  A common attribute in two tables which links one to the other. Usually it is a primary key of one table and a foreign key of the other.

* Operations on Relations

Refer to Table 3.1 [8].
3.2 Client-Server Architecture

A distributed database is a collection of data that logically belongs to the same system but is physically spread over the sites of computer network [10].

In distributed system, client-server architecture is needed to split tasks among multiple processors. Prior to client-server architecture, one processor is responsible not only for all database processing, but also for I/O processing. Thus the task execution is slowed down and the system becomes too complex to be effective. In client-sever architecture, however, each processor is in charge of only a set of specific and focused subtasks. The client is the processor sending requirements, while the server is another processor receiving, processing and replying these requirements.

In addition, client-server system splits interface management and most of application processing from database processing.

A communication software connects a client and a server and, it is responsible for transmitting commands from a client to a server, and sending results back from the server to the client.

Usually, a good client interface is able to hide the complexity of client-server system from users, which make the users think of the whole system as just a local system and ignore where the server processor locates.

The primary advantage of the client-server architecture lies in the fact that a database stored in one server can be accessed by several clients.

In SPSDB system design, Oracle7 RDBMS is chosen as the server, SQL*net as the network connection and Oracle Developer/2000 Forms 4.5 as the client development tool.
An example is given in Figure 3.2 showing the client-server architecture further in detail.

![Client-server architecture diagram](image)

**Figure 3.2:** The client-server architecture in SPSDB system

The following procedure is an example showing how a connection and communication take place between a client and a server.

1. Designing a form by developer/2000 Oracle Form 4.5 to retrieve the test information by test ID.

2. Running the form on a PC, which is a client in the client-server architecture.

3. Sending PL/SQL blocks to the UNIX database server machine by the PC. TCP/IP, the protocol used in network communication, listens the process, picks up the requests and sends them to the Oracle SQL*Net listener, which sends the requests to a server machine.

4. The server processes the requests and selects the test information requested from the database by given test ID.
5. The results are sent back to the client machine by network and are displayed on the screen of PC by the client-user interface program developed by Oracle Forms 4.5.
Chapter 4

Database Design Process

The phases of SPSDB database design are shown in Figure 4.1.

Figure 4.1: Phases of SPSDB database design

Each phase of design will be discussed in detail in following sections of this chapter.
4.1 Requirements Collection and Analysis

Potential users of the database and the engineers who conducted the road tests are inter-
viewed for feedback. The feedback along with the objectives of the project, the hardware
and software capabilities and restraints draws the outline of the design specifications.

4.1.1 Requirements

After interviewing the potential users, the following requirements are asked to be met.

The SPSDB will be used, maintained, and updated by researchers of the Civil Engi-
neering Department at Ohio University to do analysis, calculation, and modeling on the
data collected from the SPS. Those engineers and researchers who are interested in this
program will be provided the access to the system too.

The data will be maintained and updated by a system administrator of the Department
of Civil Engineering. A password qualification process is required for maintaining and
updating the data for security considerations. All users may retrieve the test data by either
one of the keywords, such as test ID, test date, SPS number, section number, material,
run number, environmental temperature, truck speed, and response data of sensors. The
details of sensors used in the tests will also be built into the database. Users with interest
in the data can retrieve information on type, manufacture, parameters, and position of
sensor. All of these activities need to be carried out through a friendly client interface
designed for the SPSDB database. To make better illustrations, users may also retrieve
the layout and sensor shapes.

4.1.2 Data Dictionary

The data can be grouped into four parts:
1. Test

Attributes are used to describe a test, attributes such as the test ID, test date, run number, SPS number, section name, section number, PCC or AC design, environment temperature, and truck used in the test.

2. Truck

GVW, space of wheels and loads of trucks.

3. Response Data of Sensors

Sensor type, sensor number, peak value, etc.

4. Sensor

Name, type, shape, manufacture, parameter, usage range and measurement.

A data dictionary can therefore be built to enumerate and define individual data elements collected from the tests. The data dictionary only consists of the items needed to be stored in the database, while relationships among these items are considered in a later design phase. The data dictionary is shown in Table 4.1.

4.2 Conceptual Database Design

4.2.1 Conceptual Schema Design

The entity-relationship (ER) model is used in the conceptual schema design phase. By using the ER model, we can derive the entities, the attributes of these entities, and the relationships among the entities.

An entity represents a real-world object or concept. An attribute represents certain property of interest that further describes an entity. A relationship among two or more entities represents an interaction among the entities [10].
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test ID</td>
<td>Unique index number for each test</td>
</tr>
<tr>
<td>Test Date</td>
<td>The date of the test</td>
</tr>
<tr>
<td>SPS No</td>
<td>SPS number</td>
</tr>
<tr>
<td>Section ID</td>
<td>Experiment section number</td>
</tr>
<tr>
<td>Section Name</td>
<td>Experiment section name</td>
</tr>
<tr>
<td>Run No</td>
<td>Truck run number</td>
</tr>
<tr>
<td>Speed</td>
<td>Truck speed during experiment</td>
</tr>
<tr>
<td>Temperature</td>
<td>Environmental temperature</td>
</tr>
<tr>
<td>Truck ID</td>
<td>The type of truck</td>
</tr>
<tr>
<td>Truck load</td>
<td>The loading of truck</td>
</tr>
<tr>
<td>Truck space</td>
<td>The space between two truck wheels</td>
</tr>
<tr>
<td>Sensor ID</td>
<td>The number of sensor</td>
</tr>
<tr>
<td>Sensor type</td>
<td>The type of sensor</td>
</tr>
<tr>
<td>Sensor test result</td>
<td>Sensor response data in experiment</td>
</tr>
<tr>
<td>Sensor position</td>
<td>X, Y, Z coordinates of a sensor</td>
</tr>
<tr>
<td>Sensor manufacturer</td>
<td>Manufacturer of sensor</td>
</tr>
<tr>
<td>Sensor Parameter</td>
<td>Parameters of sensor</td>
</tr>
<tr>
<td>Sensor measurement</td>
<td>Usage range of sensor</td>
</tr>
</tbody>
</table>

Table 4.1: Data Dictionary of SPSDB database
To design the database, an ER schema diagram is built (Figure 4.2) to illustrate the infrastructure of the test data. This schema lays the foundation for building the SPSDB.

Figure 4.2: The ER (Entity-Relationship) schema of the SPS data

4.2.2 Transaction Design

The followings are the most frequently used transactions:

- Retrieval Transactions

  Retrieve data from database and display them on the screen or print them out as a
• Update Transactions
Modify current data or insert new data.

• Mixed Transactions
Combine retrieval transactions and update transactions.

In this system, all the above transactions are designed as follows:

• Retrieval Transactions

1. List all test information
   Input: Null
   Output: All test information in a table

2. List all truck information
   Input: Null
   Output: All truck information in a table

3. List all types of information of sensors
   Input: Null
   Output: Listed names and manufacturers in a table
   Input: Sensor ID
   Output: Listed parameters of the sensor
   Input: Sensor ID
   Output: Listed measurement range of the sensor
   Input: Sensor ID
   Output: Listed usage range of the sensor
4. Retrieve information about one test, including the test ID, test date, SPS number, section number and name, run number, truck speed, temperature, truck wheel spaces, truck loads, truck weight and test results

Input: Test ID/test date/SPS number/section number/section name/run number/truck speed/temperature

The procedures of the transaction are shown in Figure 4.3.

![Flow chart of transaction 4](image)

**Figure 4.3: Flow chart of transaction 4**

Output: Test ID, test date, SPS number, section number and name, run number, truck speed, temperature, truck wheel spaces, truck loads, truck weight, and test results provided by embedded sensors
5. Retrieve information about one sensor with graphic interface.

   Input: Sensor Number

   Output: Name, manufacturer, parameter, measurement range, usage range, position, and picture of this sensor

6. Show plans or profiles of different sections

   Input: Name of view

   Output: Section view

- Update Transactions

   A log-in name and password are requested to update the database. After logging-in, the update transactions will be similar to retrieving a transaction. Focusing a field, modifying a value in the field and saving a modification would result in an updated transaction.

4.3 Choice of DBMS

Oracle7 is selected as the Relational Database Management System (RDBMS) of the SPSDB because it has the following features.

1. There is a PL/SQL processing engine in Oracle7. The PL stands for Procedural Language. PL/SQL is designed specially for a client-server system. A block of programs can be submitted in response to only a single request by using PL/SQL language to reduce network traffic. The PL/SQL will be elaborated in greater detail in section 5.2.2.
2. Oracle7 is able to store PL/SQL blocks as stored procedures, functions and packages. Moreover, it stores both compiled and textual procedures. Clients only need to pass parameters to the server and call the corresponding stored procedures when they make some transactions, and thus, the network traffic load will be minimized. Storing the compiled procedures also speeds up processing during running time, thus, significantly increasing the efficiency of a client-server system.

3. If integrity constraints are included in the table definition, they will be enforced automatically by the Oracle7 server when users insert, update, or delete the data. This facilitates the client coding work for validation and increases the reliability of the database.

4.4 Logical Database Design

4.4.1 System-independent Mapping

Refer to Figure 4.4 about the ER-to-Relational Mapping.

![Figure 4.4: ER-to-Relational Mapping](image-url)
4.5 Physical Database Design

4.5.1 Using DLL Statements

DDL is Data Definition Language. Each DBMS has its own DDL for defining the relation schemas. Most DDLs are based on the SQL language. The DDL used here is also an SQL DDL. Here is the DDL file for this system.

```
DROP TABLE SPS_TRUCK;
CREATE TABLE SPS_TRUCK (TRUCK-ID NUMBER (2) NOT NULL,
GVW NUMBER(6) NOT NULL,
LOAD1 NUMBER(6) NOT NULL,
LOAD2 NUMBER(6) NOT NULL,
LOAD3 NUMBER(6) NOT NULL,
LOAD4 NUMBER(6) NOT NULL,
LOAD5 NUMBER(6) NOT NULL,
SPACE1_2 NUMBER(3) NOT NULL,
SPACE2_3 NUMBER(3) NOT NULL,
SPACE3_4 NUMBER(3) NOT NULL,
SPACE4_5 NUMBER(3) NOT NULL,
PRIMARY KEY(TRUCK_ID)));

DROP TABLE SPSPAVE_TEST;
CREATE TABLE SPSPAVE_TEST (TEST-ID VARCHAR(20) NOT NULL,
TEST_DATE DATE NOT NULL,
SPS_NO NUMBER (1) NOT NULL,
SECTION_ID VARCHAR(5) NOT NULL,
SECTION_NAME VARCHAR(5) NOT NULL,
RUN_NO NUMBER(2) NOT NULL,
SPEED NUMBER(4,1) NOT NULL,
TEMP NUMBER(3) NOT NULL,
TRUCK_ID NUMBER (2) NOT NULL,
PRIMARY KEY(TEST_ID),
FOREIGN KEY(TRUCK_ID)REFERENCES SPS_TRUCK(TRUCK_ID)));

DROP TABLE SPS_DATA;
CREATE TABLE SPS_DATA (TEST_ID VARCHAR(20) NOT NULL,
DROP TABLE SEN_TYPE;
CREATE TABLE SEN_TYPE (  
  ID INT NOT NULL,  
  NAME VARCHAR(30) NOT NULL,  
  MANUFACTURER VARCHAR(50) NOT NULL,  
  PRIMARY KEY (ID)  
);

DROP TABLE SEN_POS;
CREATE TABLE SEN_POS (  
  ID INT NOT NULL,  
  X NUMBER(5,1) NOT NULL,  
  Y NUMBER(5,1) NOT NULL,  
  Z NUMBER(5,1) NOT NULL,  
  TYPE_ID INT NOT NULL,  
  PRIMARY KEY (ID),  
  FOREIGN KEY (TYPE_ID) REFERENCES SEN_TYPE(ID)  
);

DROP TABLE SEN_PARA;
CREATE TABLE SEN_PARA (  
  ID INT NOT NULL,  
  PARAMETER VARCHAR(20) NOT NULL,  
);
Table 4.2: Storage space of different data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size(bytes)</th>
<th>Maximum Size(bytes) of the type</th>
</tr>
</thead>
<tbody>
<tr>
<td>char(n)</td>
<td>n</td>
<td>255</td>
</tr>
<tr>
<td>varchar(n)</td>
<td>actual length(≤n)</td>
<td>2000</td>
</tr>
<tr>
<td>Number(p,s)</td>
<td>p</td>
<td>38</td>
</tr>
<tr>
<td>Date</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>Long</td>
<td>actual length</td>
<td>2g</td>
</tr>
</tbody>
</table>

DROP TABLE SEN-MEAS;
CREATE TABLE SEN-MEAS (  
ID INT NOT NULL,  
MEASUREMENT VARCHAR(20) NOT NULL,  );

DROP TABLE SEN-SEC;
CREATE TABLE SEN-SEC (  
ID INT NOT NULL,  
SECTION VARCHAR(5) NOT NULL,  );

QUIT

In the "create.sql" file, each attribute is defined by either a data type, data length or maximum length. These three factors determine the physical storage of the corresponding defined attributes in a database. The bytes occupied by each data type are summarized in Table 4.2.
Chapter 5

SPSDB System Implementation

5.1 Building Basetable

Several files are needed to set up tables (relations) of the SPS database system. They are listed as follows:

- "*.com" file
  
  "*.com" file is an executable file. It includes a sequence of commands to load data tables.
  
  Example: ‘‘dbload.com’’

  sqlplus username/password @create
  sqlload username/password control=spspave_test.ctl data=spspave_test.dat
  sqlload username/password control=sps_truck.ctl data=sps_truck.dat
  sqlload username/password control=sps_data.ctl data=sps_data.dat
  sqlload username/password control=sen_type.ctl data=sen_type.dat
  sqlload username/password control=sen_pos.ctl data=sen_pos.dat
  sqlload username/password control=sen_para.ctl data=sen_para.dat
  sqlload username/password control=sen_meas.ctl data=sen_meas.dat
  sqlload username/password control=sen_sec.ctl data=sen_sec.dat

- "*.sql" file
  
  "*.sql" file is written by SQL language and executable in an “sqlplus” environment.
  It defines the data type of each attribute and is able to create and drop tables. In chapter 4.5.1, the “*.sql” file is illustrated in the example “create.sql” file.
• "*.ctl" file

"*.ctl" file is a control file. It determines the format in which the data will be loaded into the table. An example is shown below.

Example: `spspave_test.ctl`

```sql
load data
replace
into table spspave_test
fields terminated by "":"'
(TEST_ID,TEST_DATE,SPS_NO,SECTION_ID,SECTION_NAME,RUN_NO,SPEED,TEMP,TRUCK_ID)
```

• "*.dat" file

It includes the raw data in the format controlled by the corresponding "*.ctl" file for creating tables.

Example: `spspave_test.dat`

```
951205R023440:05-DEC-95:8:J1:PCC:5:1.5:6:1
951206R024750:06-DEC-95:8:J1:PCC:1:46.0:3:2
960315R174032:15-MAR-96:8:J1:PCC:1:46.0:0:3:3
... ...
```

The difficulty in building basetable is that the raw data for the "SPS_DATA" relation are not in a format which can be used directly. Furthermore, it is practically impossible to input data manually because the amount of data is too large and will be even larger during the SPS project. To solve this problem, a transforming file is written in C language which is shown in Appendix B. The original data files are transformed to the format specified in the "SPS_DATA.CTL" file by the programs.

5.2 Building Interface

5.2.1 Introduction to Interface Design Tool - Oracle Forms 4.5

Oracle Forms 4.5, a part of Developer/2000, is a comprehensive set of application development tools provided by the Oracle company. A very powerful tool for building client-server
database applications, it is also optimized to the selected RDBMS Oracle7 server. Oracle Forms applications include three types of modules, namely, Forms, Menus and Libraries.

- Forms are collections of objects and codes, including blocks and canvas view [3].
- Menus are collections of menu objects and menu command codes, which can be attached to Forms [3].
- Libraries are collections of PL/SQL procedures, functions, and packages [3].

These modules are used in the SPSDB implementation. In the environment of the Oracle Forms 4.5 designer, a variety of objects such as windows, text items, check boxes, and buttons can be created with properties set by the designer. These objects are called interface items, which can display information to the user and interact with the user application.

Another important part of Oracle Forms 4.5 is the Canvas-views, which are the background on which items and objects are placed. Each canvas-view needs to be assigned to a window. However, several canvas-views can share one window.

5.2.2 Requirements of Interface

- Informational
  Provide users with as much information as possible in an interface.

- Visualized
  Use different colors and fonts to group different information and highlight the important points on the page.
- Friendly

Easy to use and easy to guess the functions of items. Moreover, help information should be available.

5.3 Structural Design of Canvas-Views

For this section, Figure 5.1 shows a better picture about the navigation relationships among canvas-views of SPSDB.

![Diagram of Canvas-View navigation design]

Figure 5.1: Canvas-View navigation design
5.4 Interface Coding

The interface is designed by Oracle Forms 4.5, a part of the Developer/2000 package. The basic objects in Oracle Forms 4.5 consist of blocks and items. Items are the smallest units to interact with users whereas blocks are containers of items. They can be connected to the basetable of the database. In these kinds of blocks, the items directly map to the specific columns in the basetable. These relationships allow query, updating, insertion, deletion and other operations on records in the corresponding tables.

The programming model of the Oracle Form is an event-driven model. When interacting with objects in interfaces, users can create an event by pressing a button or pressing a key. The functionalities are implemented by writing codes driven by this event.

Using triggers is the primary method to attach a code to a form. A trigger is a block of PL/SQL codes added to an object in a form, which can be triggered by a specific event such as the pressing of a mouse button. Appendix C shows PL/SQL codes written for SPSDB implementation.

There are a variety of built-in procedures in Oracle Form4.5. These procedures can be called directly by the user code. Moreover, users can define user-named procedures, functions and package, too. After these program units are built, they can be called just like built-in procedures.

5.4.1 Relation Algebra

Relation algebra is a collection of operations that is to manipulate entire relations [10]. In this section, pseudocodes are used to expresses relation algebra. In the following formula, \( \sigma \) is used to denote SELECT and \( \pi \) is used to denote PROJECT.
• Browse information of SPS truck test

$$ \pi_{all}(SPS\_PAVE\_TEST) $$

• Browse information of SPS truck

$$ \pi_{all}(SPS\_TRUCK) $$

• Browse information of sensors

$$ \pi_{all}(SEN\_TYPE, SEN\_PARA, SEN\_MEAS, SEN\_POS, SEN\_SEC) $$

• Query information of test, truck, and sensor of an SPS truck test

Given: testid

$$ SPS\_PAVE\_BLOCK \leftarrow \pi_{all}(\sigma_{TEST\_ID=testid}(SPS\_PAVE\_TEST)) $$

or Given: date

$$ SPS\_PAVE\_BLOCK \leftarrow \pi_{all}(\sigma_{TEST\_DATE=date}(SPS\_PAVE\_TEST)) $$

or Given: section_no

$$ SPS\_PAVE\_BLOCK \leftarrow \pi_{all}(\sigma_{SECTION\_NO=section\_no}(SPS\_PAVE\_TEST)) $$

or Given: material type

$$ SPS\_PAVE\_BLOCK \leftarrow \pi_{all}(\sigma_{SECTION\_NAME=material\_type}(SPS\_PAVE\_TEST)) $$

or Given: run_no

$$ SPS\_PAVE\_BLOCK \leftarrow \pi_{all}(\sigma_{RUN\_NO=run\_no}(SPS\_PAVE\_TEST)) $$
or Given: speed

\[ \text{SPSPACE\_BLOCK} \leftarrow \pi_{all}(\sigma_{SPEED=\text{speed}}(\text{SPSPACE\_TEST})) \]

or Given: temperature

\[ \text{SPSPACE\_BLOCK} \leftarrow \pi_{all}(\sigma_{TEMP=\text{temperature}}(\text{SPSPACE\_TEST})) \]

or Given: truckid

\[ \text{SPSPACE\_BLOCK} \leftarrow \pi_{all}(\sigma_{TRUCK\_ID=\text{truckid}}(\text{SPSPACE\_TEST})) \]

\[ \text{trID} \leftarrow \sigma_{TRUCK\_ID}(\text{SPSPACE\_BLOCK}) \]

\[ \text{SPS\_TRUCK\_BLOCK} \leftarrow \pi_{GVW,space,load}(\sigma_{SPS\_TRUCK\_TRUCK\_ID=\text{trID}}(\text{SPS\_TRUCK})) \]

\[ \text{teID} \leftarrow \sigma_{TEST\_ID}(\text{SPSPACE\_BLOCK}) \]

\[ \text{SPS\_SENSOR} \leftarrow \pi_{all}(\sigma_{SPS\_DATA\_TEST\_ID=\text{teID}}(\text{SPS\_DATA})) \]

- Show section view of sensor layouts

Given: name of section view show section view by name

\[ \text{sno} \leftarrow \sigma_{SENSOR\_NO}(\text{SPS\_SENSOR\_BLOCK}) \]

\[ \text{SPS\_TYPE\_BLOCK} \leftarrow \pi_{NAME,MANUFACTURER}(\sigma_{SPS\_TYPE\_ID=\text{sno}}(\text{SPS\_TYPE})) \]

\[ \text{SPS\_POS\_BLOCK} \leftarrow \pi_{X,Y,Z}(\sigma_{SPS\_TYPE\_ID=\text{sno}}(\text{SPS\_TYPE})) \]

\[ \text{SPS\_PARA\_BLOCK} \leftarrow \pi_{parameter}(\sigma_{SPS\_TYPE\_ID=\text{sno}}(\text{SPS\_TYPE})) \]

\[ \text{SPS\_MEAS\_BLOCK} \leftarrow \pi_{measurement}(\sigma_{SPS\_TYPE\_ID=\text{sno}}(\text{SPS\_TYPE})) \]

\[ \text{SPS\_SEC\_BLOCK} \leftarrow \pi_{section}(\sigma_{SPS\_TYPE\_ID=\text{sno}}(\text{SPS\_TYPE})) \]
5.4.2 PL/SQL code

PL/SQL language functions are not limited to the SQL database language functions, such as manipulating data or processing transaction; they also include features of procedural programming languages such as variable, constant declarations, assignment, looping and conditional branching.

The PL/SQL language is a block of code in Oracle Forms 4.5. Its structure is shown in Figure 5.2.

![PL/SQL code structure diagram](image)

Figure 5.2: PL/SQL code structure

The functions of triggers, PL/SQL type menu items, user-named subprograms, and PL/SQL packages are realized by the attached PL/SQL code. Moreover, the PL/SQL language can optimize the client-server architecture, and pack several program units into a single block. Sending such one packed block through the network will reduce the network traffic to a great extent.
5.5 Examples

Examples presented in this section show how to implement some functions of SPSDB in detail. A number of objects are used, such as button item, image item, text item, displayed item, master-detail blocks, content and stack view, multiple window, and triggers. PL/SQL codes are illustrated in the examples, and procedures of actual implementations are explained as well.

5.5.1 How to Realize Retrieval Functions

![Retrieval window](image)

Figure 5.3: Retrieval window
Figure 5.3 is about retrieving information on the test truck and test results. It can be used for browsing or updating the data, depending on a global variable set.

![Figure 5.4: Query key section of retrieval window](image)

There are four parts in Figure 5.3. One includes a series of buttons, shown in Figure 5.4, which are used to select an attribute as a query condition. The function of these buttons is to show a list of distinct values of the corresponding attribute.

After a button is selected, a displayed item will show an attribute name, and a text item will show the selected value of that attribute.

The phases to build these functions are as follows.

1. Drawing buttons on canvas.

2. Setting properties of these buttons.

3. Using “When-Button-Pressed” trigger to make the buttons interactive.

4. Writing PL/SQL code by PL/SQL editor for the trigger.
Figure 5.5: Example of using properties, canvas-view editor, and PL/SQL editor to build a button
The other three parts are three basetable blocks with master-detail relations.

![Test Information](image)

**Figure 5.6: Test block**

![Truck Information](image)

**Figure 5.7: Truck block**

The block "test" is a master block based on "SPSPACE.TEST" which shows the information of tests. The block "truck" is a detailed block of the block "test" based on the "SPS_TRUCK" table. The common attribute "truck_ID" connects the "test" block and the "truck" block. "Truck_ID" is both the foreign key of "SPSPACE.TEST" and the primary key of "SPS_TRUCK".

The block "test_result" is also a detailed block of the block "test" based on the "SPS_DATA" table. The common attribute "sensor_ID" connects the "test_result" block and the "test" block. "sensor_ID" is both the foreign key of "SPSPACE.TEST" and the primary key of "SPS_DATA".
Figure 5.8: Sensor-Data block

Figure 5.9: Icon buttons
Aside from these four parts, there are several icon buttons shown in Figure 5.9. The second one is the query button whose function of it is to query test, truck and test data based on the selected value of an attribute.

From the window shown in Figure 5.3, users can navigate to sensor windows. By clicking the left button of the mouse when a sensor No is selected, the users can access to “Sensor_View”, which is shown in Figure 5.10.

Figure 5.10: Retrieve sensor view by sensor number
5.5.2 How to Make Graphics Interactive

In Figure 5.10, a layout of sensors is shown. The symbols of the sensors in the layout are interactive. When users click on the sensor, information about position, name, manufacturer, parameter, measurement, etc., will appear.

The followings are the procedures for building a sensor layout:

1. Drawing background of sensor layout.

2. Drawing all sensors as image items.

3. Adding trigger “When-Image-Pressed” to make sensors interactive.

Because there are a number of sensors, adding a trigger to each image item is not effective, the program unit properties in Oracle Forms 4.5 will be used. After procedures written for the trigger and the level of the trigger are defined in form level, all image items can share these procedures by calling them by parameters.

When a sensor is pressed, the corresponding information will be shown in the table and a yellow block will highlight the clicked sensor.

In this window, only the layouts of the sensors need to be changed, the other parts remain the same. So the views for the layout are set as stack-views. A stack-view is a type of canvas-view, which can be programmed to be displayed or hidden during the program running time [3].

Because some users need to see the shape of sensors, the picture of current sensors can be viewed by pressing “View-Sensor-Shape”.


5.5.3 How to Build a Menu

Menu modules are separate from form modules. After a menu is built and generated in a menu design environment, a "*.mmx" file is available. Then the "*.mmx" file is attached to the form which uses this menu. When the form is in run, the menu will be invoked and shown in the menu bar of the form.

Figure 5.11 shows the menu used in the SPSDB form.

```
File  Go_To  Retrieve  Graphics  Help  Window
```

Figure 5.11: The menu attached to form SPSDB

The procedures for building a menu are as follows.

- Use menu editor to design a menu. All items included in the menu are specified. Refer to Figure 5.12.

- Write PL/SQL codes in the PL/SQL editor to realize the functions of a menu item. Refer to Figure 5.13.
Figure 5.12: Menu editor
Figure 5.13: PL/SQL codes attached to a menu item

```
declare
    albut number;
begin
    if :global.flag=1 then
        do_key('commit_form');
    else
        albut:=show_alert('noauthorized');
    end if;
end;
```
Chapter 6

User Manual of SPSDB

Figure 6.1: First window of SPSDB

Users can click the “Browse” button for browsing the database through the SPSDB, but no authorization is given for modification. When the “Update” button is clicked, a log-in window will appear. The user needs a log-in name and a password to modify the database.
Figure 6.2: Sensor information table

In the left table of this window, “Type” and “Manufacturer” are shown. The other corresponding parameters of a sensor are shown in the right tables when the sensor is selected from the left table. The row in the gray-colored background is the currently selected row.
Figure 6.3: Retrieve test information by query key
Clicking one button in the upper-left rectangle to select an attribute as the query key. The attribute name which the button represents is shown at the bottom of upper-left rectangle with a blank text item when the corresponding button is pressed; a list of distinct values of the attribute will then appear in the window.
Figure 6.4: Test ID list

Users can select one test from test ID list. The selected test ID will show in the blank text item at the bottom of the upper-left rectangle. By pressing the "query" button, a list of records retrieved by the query condition specified in Figure 6.3 will be shown in this window.
Figure 6.5: Test list
Retrieved data will fill out the corresponding cells after a test is selected from the “test list”.

Figure 6.6: Query results
A sensor layout including the information of a sensor can be shown by clicking the "Sensor-No". If more information is needed, the button "More" can be clicked.
Figure 6.8: Sensor shape view
Sensor shape can be shown when button "View_Sensor_Shape" is pressed.
Figure 6.9: Warning message
A message is shown to warn an unauthorized user that changes cannot be saved.
Figure 6.10: Login window

The log-in window for authorizing an update. A log-in name and a password are need to update the data in the system.
If a user has the authority to update data, focusing a field in the table by using a cursor, typing a new value, and selecting “save” in the “File” menu will yield an updated transaction. In this example, the highlighted “3” will be modified.
Chapter 7

Conclusions and Discussions

- When the SPSDB system has been built up, it can be used to browse information from the SPS by engineers and researchers in civil engineering. The data stored in the database can be used as references for research. The SPSDB also provides several ways to retrieve data from the database. A query key can be specified by pressing a button in the retrieve interface or can be conveniently selected from a pop-up window. A number of graphics are also available in the SPSDB, such as sensor layouts and sensor pictures, which give the user visual assistance in learning about the SPS. Generally, retrieving through SPSDB is very effective.

- Because Graphic User Interface (GUI) is used in the design, the SPSDB can be learned with little training. Buttons, icons and Help information will facilitate how to use the system. The information about tests, trucks and sensors is interweaved; users can retrieve information from one relation to another conveniently.

- SPSDB can also be modified by an authorized user through interfaces rather than through a basetable. A log-in name and a password are required to update, and this enhances the security of system. The advantage of this feature is that updating data can be found through query, which is much faster than searching one-by-one.

- As a database, Oracle Forms 4.5 is a good, comprehensive design tool. It provides
rich objects, triggers, and built-in procedures. All of them make it possible to build complex transactions and reduce the load of coding for designers. Oracle Forms 4.5 is designed to build an interface through a canvas-view, to set the property of an item in the properties window, and to attach a PL/SQL code to a trigger. Moreover, using the properties class and the visual attributes can define and update visual characteristics of items.

- In order to make the database much more useful, more data from tests are needed to be saved in the SPSDB in the future.
Bibliography


Appendix A

Objects of Form

Figure A.1: Blocks of Oracle Forms 4.5
Figure A.2: Canvas-Views of Oracle Forms 4.5
Figure A.3: Navigator of Oracle Forms 4.5
Figure A.4: Program Units of Oracle Forms 4.5
Figure A.5: PL/SQL editor of Oracle Forms 4.5

```sql
declare
  itm char(40) := :System.current_Item;
  blk char(40) := :System.current_block;
  upitm char(40);
  newqty number;
  al_but number;
begin
  upitm := upper(itm);
  if (upitm = 'EXIT') then
    if (:global.flag=0) then
      al_but:=show_alert('nochange');
      exit_form(no_validate);
    elsif (:global.Flag=1) then
      al_but:=show_alert('change');
      exit_form(do_commit);
    end if;
```
Appendix B

C code for Transforming the Format of Raw Data

```c
/* this program searches input files which need to be transformed from specific directory. Then execute a file to do transformation one by one. */

#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>

void list(char C501); 
void catch(char C501);
void exc (char [21], char [50]);

void main()
{
    int i;
    char ch, dire[50];

    if( fork()==0 )
    {
        execl("/bin/rm", "rm", "basefile",0);
        perror("execl");
        exit(0);
    }
    wait(0);

    for(i=0; i<13; i++)
    {
        strcpy(dire, "./data2/k13");
```
ch='a'+i;
strcat(dire,&ch);
list(dire);
}
}

void list(char dire[50])
{
  int fd;
  if( (fd=open("tmplist", 0_WRONLY|0_CREAT|0_TRUNC, 0777))<0 )
    { perror("open");
      exit(0);
    }

  if( fork()==0 )
    {
    close(1);
    dup(fd);
    exec1("/bin/ls", "ls", dire, 0);
    perror("execll");
    exit(0);
    }
  close(fd);
  wait(0);
  catch(dire);
}

void catch(char dire[50])
{
  char s[21];
  FILE *fp;

  if( (fp=fopen("tmplist", "r"))==NULL )
    { perror("fopen");
      exit(0);
    }

  while(1)
    {
    if( fgets(s, 20, fp)==NULL )
      break;
    else
      {
          {
            s[7]='\0';
            exc(s, dire);
            strcpy(s, ""); //So, set to null at least until s[6]
This program reads input from input files. Outputs are in the format specified by "sps-data.ctl" file which can be used as basetable.
--written by Jiayan Liu--
int getline (char [], int);
void initials(int [40]);
FILE *fp, *fw;

struct line{
    char sensor_t;
    int sensor_id; //initial
    float wheel[wheelno];
    float peak[peakno];
} list[size];

//input and output files as arguments

void main(int argc, char *argv[]) {
    int flag, i, j, in, len, rowno, left, right, sensor[40];
    char id[30], time[10], line[MAXLINE], row[MAXLINE], tmp1[7];
    char tmp2[10], tmpstr[MAXLINE], tstr[20];

    if (argc<3) {
        perror("argc:\n");
        exit(0);
    }
    if ((fp=fopen(argv[argc-2], "r")) == NULL) {
        perror("fopenr:\n");
        exit(0);
    }
    if ((fw=fopen(argv[argc-1], "a+")) == NULL) {
        perror("fopenw:\n");
        exit(0);
    }

    initials(sensor);
    getline(line, MAXLINE);
    sprintf(id, "%c%c%c%c%c%c",
            line[6], line[7], line[0], line[1], line[3], line[4], 'R');

    len=getline(line, MAXLINE);
    strcpy(time, "");
    j=0;
    for(i=0; i<=7; i++) {
        if(i!=2 && i!=5)
            time[j++]=line[i];
    }
    time[j]='/0';
strcat(id, time);
for (i=0; i<3; i++)
    len=getline(line, MAXLINE);

rowno=0;
while(i)
{
    for(i=0; i<=wheelno; i++)
        list[rowno].wheel[i]=0.0;
    for(i=0; i<=peakno; i++)
        list[rowno].peak[i]=0.0;

    len=getline(line, MAXLINE);
    sscanf(line,"%s", tmp1,tmp2);

    switch(tmp2[0]) {
        case 'D':
            list[rowno].sensor_t='D';break;
        case 'F':
            list[rowno].sensor_t='F';break;
        case 'L':
            list[rowno].sensor_t='L';break;
        case 'P':
            list[rowno].sensor_t='P';break;
    }
    if( isdigit( tmp2[strlen(tmp2)-2] ) )
        list[rowno].sensor_id=10+atoi(&tmp2[strlen(tmp2)-1]);
    else
        list[rowno].sensor_id=atoi(&tmp2[strlen(tmp2)-1]);

    getline (line, MAXLINE);
    len=getline (line, MAXLINE);
    while(line[0]!="\n" && len!=0 )
{
    if(line[2]=='-') {
    left=atoi(&line[1]);
    right=atoi(&line[3]);
    j=0;
    strcpy(tmp2, "");
    for(i=7; i<=11; i++)
        tmp2[j++]=line[i];
    tmp2[j]="\0";
    for (i=left; i<=right; i++)
        sscanf(tmp2, "%f", &list[rowno].wheel[i-1]);
    if(left==2)
        sscanf(tmp2, "%f", &list[rowno].peak[2]);
    else if(left==4)
```c
sscanf(tmp2, "%f", &list[rowno].peak[5]);
}
else
{
    in = atoi(&line[1]);
    if(len > 11)
    {
        j = 0;
        strcpy(tmp2, "\0");
        for(i = 7; i <= 11; i++)
            tmp2[j++] = line[i];
        tmp2[j] = '\0';
        if(strcmp(tmp2, "---") == 0 || strcmp(tmp2, "\0") == 0)
            strcpy(tmp2, "0.0");
        sscanf(tmp2, "%f", &list[rowno].peak[5]);
    }

    if(len > 20)
    {
        j = 0;
        strcpy(tmp2, "\0");
        for(i = 16; i <= 20; i++)
            tmp2[j++] = line[i];
        tmp2[j] = '\0';
        if(strcmp(tmp2, "---") == 0 || strcmp(tmp2, "\0") == 0)
            strcpy(tmp2, "0.0");
        if(in == 1)
            sscanf(tmp2, "%f", &list[rowno].peak[0]);
        else
            sscanf(tmp2, "%f", &list[rowno].peak[4]);
    }

    if(len > 28)
    {
        j = 0;
        strcpy(tmp2, "\0");
        for(i = 24; i <= 28; i++)
            tmp2[j++] = line[i];
        tmp2[j] = '\0';
        if(strcmp(tmp2, "---") == 0 || strcmp(tmp2, "\0") == 0)
            strcpy(tmp2, "0.0");
        switch(in)
        {
            case 1:
                sscanf(tmp2, "%f", &list[rowno].peak[1]);
                break;
            case 2:
                sscanf(tmp2, "%f", &list[rowno].peak[2]);
                break;
            case 4:
```

```c
sscanf(tmp2, "%f", &list[rowno].peak[5]);
} break;
}

if(len>36)
{
    j=0;
    strcpy(tmp2, "");
    for(i=33; i<=36; i++)
        tmp2[j++]=line[i];
    tmp2[j]={'\0'};
    if( strcmp(tmp2, "---")==0 || strcmp(tmp2, " ")==0 )
        strcpy(tmp2, "0.0");
    switch(in)
    {
    case 3:
        sscanf(tmp2, "%f", &list[rowno].peak[3]);
        break;
    case 5:
        sscanf(tmp2, "%f", &list[rowno].peak[6]);
        break;
    }
} //end else
len=getline(line, MAXLINE);
} //end 'while(line')

if(len==0)
    break;
else
{
    sprintf(row,
"%s:%c:%d:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:%5.1f:
%5.1f:%5.1f:%5.1f:%d",
    id,
    list[rowno].sensor_t,
    list[rowno].sensor_id,
    list[rowno].peak[0],
    list[rowno].wheel[0],
    list[rowno].peak[1],
    list[rowno].wheel[1],
    list[rowno].peak[2],
    list[rowno].wheel[2],
    list[rowno].peak[3],
    list[rowno].peak[4],
    list[rowno].wheel[3],
    list[rowno].peak[5],
    list[rowno].wheel[4],
    list[rowno].peak[6],
    list[rowno].wheel[6],
    list[rowno].peak[7],
    list[rowno].wheel[7],
    list[rowno].peak[8],
    list[rowno].wheel[8],
    list[rowno].peak[9],
    list[rowno].wheel[9],
    list[rowno].peak[10],
    list[rowno].wheel[10],
    list[rowno].peak[11],
    list[rowno].wheel[11],
    list[rowno].peak[12],
```
list[rowno].peak[6],
sensor[rowno]);

j=0;
strcpy(tmpstr, ""));
for (i=0; i<=strlen(row); i++)
{
    if (row[i]!=' ')
        tmpstr[j++]=row[i];
}
tmpstr[j]='/0';
strcpy(row, tmpstr);

i=0;
strcpy(tstr, ""));
strcpy(tmpstr, ""));
while(row[i]!='/0')
{
    j=0;
    while( row[i]!='/.' & row[i]!='/0' )
    {
        tstr[j++]=row[i++];
    }
tstr[j]='/0';
if( strcmp(tstr, "0.0")!=0 )
{
    strcat(tmpstr, tstr);
    flag=1;
} else
    flag=0;
if(row[i]!='/0')
{
    strcat(tmpstr, ".");
i++;}
else if(flag==0)
    strcat(tmpstr, "1");
} fprintf(fw, "%s\n", tmpstr);
rowno++;
}
} //end 'while(1')
} //end 'main()'

int getline (char s[], int lim)
{
    int c, i;
i=0;
while (--lim>0 && (c=getc(fp)) != EOF && c != '\n')
  s[i++]= c;
if (c == '\n')
  s[i++]= c;
  s[i] = '\0';
return (i);
}

void initials(
    int s[40])
{
  s[0]=60; s[1]=85;
  s[8]=70; s[9]=71;
  s[10]=82;
  s[15]=66; s[16]=67;
  s[17]=73; s[18]=78;
  s[19]=58; s[20]=91;
  s[21]=57; s[22]=56;
  s[23]=55; s[24]=54;
  s[27]=52; s[28]=51;
  s[29]=59; s[30]=74;
}
Appendix C

PL/SQL Codes in Oracle Forms 4.5

--PL/SQL codes in program units

--AllQUERY (Package Spec)
PACKAGE allquery IS
  procedure querybyatt( attrname char, attrvalue number, f char);
  procedure querybyatt( attrname char, attrvalue date);
  procedure querybyatt( attrname char, attrvalue varchar2, f integer);
END allquery;

--ALLQUERY (Package Body)
PACKAGE BODY allquery IS
  lov_return BOOLEAN;
  PROCEDURE querybyatt(attrname in char, attrvalue in number, f in char) IS
    begin
    if(f='p') then
      if (populate_group_with_query
        ('test',
        'SELECT *
         FROM spspave_test
         WHERE spspave_test.sps_no:=test.sps_no') ) <>0 then
        message('fail1');
        RAISE FORM_TRIGGER_FAILURE;
      end if;
      elsif(f='s') then

if (populate_group_with_query
    ('test',
     'SELECT *
     FROM spspave_test
     WHERE spspave_test.speed=:test.speed') )
<>0 then
    message('fail1');
    RAISE FORM_TRIGGER_FAILURE;
end if;
elsif(f='t') then
    if (populate_group_with_query
        ('test',
         'SELECT *
         FROM spspave_test
         WHERE spspave_test.temp=:test.temp') )
<>0 then
    message('fail1');
    RAISE FORM_TRIGGER_FAILURE;
end if;
elsif(f='k') then
    if (populate_group_with_query
        ('test',
         'SELECT *
         FROM spspave_test
         WHERE spspave_test.truck-id=:test.truck-id') )
<>0 then
    message('fail1');
    RAISE FORM_TRIGGER_FAILURE;
end if;
end if;
--Set_Lov_Property('test',GROUP_NAME,'test');
lov_return:=Show_Lov('test',20, 50);
    go_block('sps_truck');
    execute_query;
    go_block('sps_data');
    execute_query;

END querybyatt;

PROCEDURE querybyatt(attrname in char, attrvalue in date ) IS
begin
    if (populate_group_with_query
        ('test',
         'SELECT *
         FROM spspave_test
         WHERE spspave_test.truck-id=:test.truck-id') )
<>0 then
    message('fail1');
    RAISE FORM_TRIGGER_FAILURE;
end if;
end if;
end if;
WHERE spspave_test.test_date=:test.test_date')

<>0 then
  message('fail2');
  RAISE FORM_TRIGGER_FAILURE;
end if;

--Set_Lov_Property('test',GROUP_NAME,'test');
lov_return:=Show_Lov('test',20,50);

go_block('sps_truck');
execute_query;
go_block('sps_data');
execute_query;

END querybyatt;

PROCEDURE querybyatt(attrname in char, attrvalue in varchar2,
f in integer) IS
begin
  if(f=0) then
    if (populate_group_with_query ('test',
      'SELECT *
      FROM spspave_test
      WHERE spspave_test.test_id=:test.test_id')
      <>0 then
      message('fail3');
      RAISE FORM_TRIGGER_FAILURE;
    end if;
  elsif(f=1) then
    if (populate_group_with_query ('test',
      'SELECT *
      FROM spspave_test
      WHERE spspave_test.section_id=:test.section_id'
      <>0 then
      message('fail3');
      RAISE FORM_TRIGGER_FAILURE;
    end if;
  elsif(f=2) then
    if (populate_group_with_query ('test',
      'SELECT *
      FROM spspave_test
      WHERE spspave_test.section_name=:test.section_name'
      <>0 then
      message('fail3');
      RAISE FORM_TRIGGER_FAILURE;
    end if;
  end if;
end if;
RAISE FORM_TRIGGER_FAILURE;
end if;
end if;
--Set_Lov_Property('test',GROUP_NAME,'test');
lov_return:=Show_Lov('test',20, 50);

  go_block('sps_truck');
  execute_query;
  go_block('sps_data');
  execute_query;

END querybyatt;
end allquery;

 Procedure getsensorinfo(sensorno in integer)IS
 BEGIN
  imagesen(:sen_pos.type, :sen_type.name, :sen_type.manufacturer);
  imagepara(:sen_pos.type, :sen_type.parameter);
  imagemeas(:sen_pos.type, :sen_type.measurement);
  imagesection(:sen_pos.type, :sen_type.section);
 END;

 Procedure gettid(thisid in integer,
 X in out number,
 Y in out number,
 Z in out number,
 TID in out integer) IS
 BEGIN
 select x, y, z, type_id
 into X, Y, Z, TID
 from sen_pos
 where id= thisid;
 END;

 Procedure imagemeas(tid in integer,
 Meas in out varchar2) IS
 cursor meas_cur is select measurement from sen_meas where id=tid;

str varchar(20);
BEGIN
    Meas:='';
    open meas_cur;
    loop
        fetch meas_cur into str;
        exit when meas_cur%notfound;
        Meas := Meas || str || '&';
    end loop;
END;

-------------------------------------------------------------------
PROCEDURE imagepara(tid in integer,
    Para in out varchar2) IS
    cursor para_cur is select parameter from sen_para
    where id=tid;
    str varchar(20);
BEGIN
    Para:= ' ';
    open para_cur;
    loop
        fetch para_cur into str;
        exit when para_cur%notfound;
        Para:=Para || str || '&';
    end loop;
END;

-------------------------------------------------------------------
PROCEDURE imagesection(tid in integer,
    Sect in out varchar2) IS
    cursor sect_cur is select section from sen_sec where
    id=tid;
    str varchar(5);
BEGIN
    Sect:='';
    open sect_cur;
    loop
        fetch sect_cur into str;
        exit when sect_cur%notfound;
        Sect:=Sect || str || '&';
    end loop;
END;
PROCEDURE imagesen(tid in integer,
    Name in out varchar,
    Manuf in out varchar) IS
BEGIN
    select name, manufacturer
    into Name, Manuf
    from sen_type
    where id=tid;
END;

BEGIN

    x:=to_number(get_item_property(viewname||'.s'||sennum, x_pos));
    y:=to_number(get_item_property(viewname||'.s'||sennum, y_pos));
    set_item_property(viewname||'.pointer', position, x-5, y-5);
    set_item_property(viewname||'.pointer', displayed, property_true);

END;

--********************************************************************************
PROCEDURE setitempro(blk in char,
    item in char) IS
    lov_return BOOLEAN;
BEGIN

    --set_item_property('test.'||item||'_but', label, 'X');
    set_item_property('test.'||item, displayed, property_true);
    lov_return:=show_lov(item, 15, 15);

END;

--********************************************************************************
PROCEDURE showmodify1(viewname varchar,
    id char)IS
BEGIN

    show_view(viewname);

END;

--********************************************************************************
PROCEDURE showsensorshape(T1D in out integer) IS
    stype char;
BEGIN
    stype :=to_char(T1D);
PROCEDURE showsensorview( sview in varchar2) IS
BEGIN
   --hide_view('data');
   set_window_property('sensor_view', width, 340);
   set_window_property('sensor_view', x_pos, 80);
   set_window_property('sensor_view', visible, property_true);
   --go_block('sen_pos');
   --go_block('sen_type');
   --show_view('sen_pos_can');
   show_view('sen_pos_can');
   show_view(sview);
   go_block(sview);
END;

PROCEDURE snotoview(blkname in varchar2) IS
   itm char(40) := system.current_item;
   blk char(40) := system.current_block;
   t_val char(40) := system.cursor_value;
   itmvalue integer :=0;
   t timer;
BEGIN
   Set_Block_Property(blkname, CURRENT_RECORD_ATTRIBUTE, 'RED');
   if(lower(blk) = blkname and lower(itm) = 'sensor_no') then
      itmvalue := to_number(t_val);
   end if;
   if(itmvalue>=1 and itmvalue<=18) then
      showsensorview('ac_plan');
      go_item('ac_plan.s'||t_val);
      pointsensor('ac_plan', t_val);
   end if;
   if(itmvalue>=19 and itmvalue<=22) then
      showsensorview('ac_profile');
      go_item('ac_profile.s'||t_val);
      pointsensor('ac_profile', t_val);
   end if;
END;
if((itmvalue>=51 and itmvalue<=78) or itmvalue=91 or
  itmvalue=92) then
  showsensorview('pcc_plan');
  go_item('pcc_plan.s'|t_val);
  pointsensor('pcc_plan', t_val);
end if;

if(itmvalue>=79 and itmvalue<=81) then
  showsensorview('dd');
  go_item('dd.s'|t_val);
  pointsensor('dd', t_val);
end if;

if(itmvalue>=82 and itmvalue<=84) then
  showsensorview('cc');
  go_item('cc.s'|t_val);
  pointsensor('cc', t_val);
end if;

if(itmvalue>=85 and itmvalue<=88) then
  showsensorview('bb');
  go_item('bb.s'|t_val);
  pointsensor('bb', t_val);
end if;

if(itmvalue=89 or itmvalue=90) then
  showsensorview('aa');
  go_item('aa.s'|t_val);
  pointsensor('aa', t_val);
end if;

if(itmvalue>0) then
  getsensorinfo(itmvalue);
end if;

end;

-- PL/SQL codes attached on items, blocks and form
--
-- ----------------------------------------------------------

-- Entrance.Browse
:global.flag:=0;
hide_view('entrance');
go_block('btest');
execute_query;

--******************************
--Entrance.Update
set_window_property('login', visible, property_true);
go_item('login.loginnme');

--******************************
--Login.Cancel
clear_block(no_validate);
set_window_property('login', visible, property_false);
go_block('entrance');

--******************************
--Login.Connect
declare
    al_button number;
    --name varchar2;
pwd varchar2;
cursor login_cur(name varchar2) is
    select password from logintab where loginnme=name;
begin
    --message(pwd);
    open login_cur(:login.loginnme);
    loop
        fetch login_cur into pwd;
        exit when login_cur%notfound;
        if(:login.password=pwd) then
            goto end_loop;
        end if;
    end loop;
    close login_cur;
clear_block('no_validate');
bell;
al_button:=show_alert('wrongpwd');
end loop>
message('label');
clear_block(no_validate);
set_window_property('login', visible, property_false);
:global.flag:=1;
go_block('test');
end;
--Button.moresen
hide_view('test');
go_block('sps_data2');
execute_query;

--Button.Query
DECLARE
group_id RecordGroup;
query_ok NUMBER;
lov_return BOOLEAN;
t_id varchar2;

begin
if ( lower(get_item_property('test.test_id', displayed))='true' ) then
  allquery.querybyatt('test_id', :test.test_id, 0);
elsif( lower(get_item_property('test.test_date', displayed))='true' ) then
  allquery.querybyatt('test_date', :test.test_date);
elsif( lower(get_item_property('test.sps_no', displayed))='true' ) then
  allquery.querybyatt('sps_no', :test.sps_no, 'p');
elsif( lower(get_item_property('test.section_id', displayed))='true' ) then
  allquery.querybyatt('section_id', :test.section_id, 1);
elsif( lower(get_item_property('test.section_name', displayed))='true' ) then
  allquery.querybyatt('section_name', :test.section_name, 2);
elsif( lower(get_item_property('test.speed', displayed ))='true' ) then
  allquery.querybyatt('speed', :test.speed, 's');
elsif( lower(get_item_property('test.temp', displayed ))='true' ) then
  allquery.querybyatt('temp', :test.temp, 't');
elsif( lower(get_item_property('test.truck_id', displayed ))='true' ) then
  allquery.querybyatt('truck_id', :test.truck_id, 'k');
end if;
end;

--Sen_Pos.More
if( lower(get_item_property('sen_pos.more', label))='more' ) then

  set_item_property('sen_pos.more', item_is_valid, property_false);
  set_item_property('sen_pos.close', position, 361, 75);
  set_item_property('sen_pos.more', position, 390, 75);
  set_item_property('sen_pos.more', label, '--');
  set_window_property('sensor_view', width, 460);
  set_window_property('sensor_view', x_pos, 5);

set_window_property('sensor_view', y_pos, 5);
end if;

--Sen_Pos.Close
clear_block(no_validate);
go_block('sen_pos');
clear_block(no_validate);
go_block('sen_type');
clear_block(no_validate);
hide_window('sensor_view');

--Sen_Pos.Pict
declare
stype char;
begin
if (lower(get_window_property('senshape', visible))='true') then
  set_window_property('senshape', visible, property_false);
end if;
set_window_property('senshape', visible, property_true);
stype:=to_char(:sen_pos.type);
show_view('gage'||stype);
end;

--Sen_Pos.Show (text item)
if :sen_pos.show='AC Section Plan' then
  show_view('ac_plan');
elsif :sen_pos.show='AC Section Profile' then
  show_view('ac_profile');
elsif :sen_pos.show='PCC Section Plan' then
  show_view('pcc_plan');
elsif :sen_pos.show='Section AA' then
  show_view('aa_can');
elsif :sen_pos.show='Section BB' then
  show_view('bb_can');
elsif :sen_pos.show='Section CC' then
  show_view('cc_can');
elsif :sen_pos.show='Section DD' then
  show_view('dd_can');
else
  show_view('sen_pos_can');
end if;
declare
lov_return boolean;
begin
lov_return := show_lov('graphics', 400, 100);

if :sen_pos.show='AC Section Plan' then
show_view('ac_plan');
elsif :sen_pos.show='AC Section Profile' then
show_view('ac_profile');
elif :sen_pos.show='PCC Section Plan' then
show_view('pcc_plan');
elif :sen_pos.show='Section AA' then
show_view('aa');
elif :sen_pos.show='Section BB' then
show_view('bb');
elif :sen_pos.show='Section CC' then
show_view('cc');
elif :sen_pos.show='Section DD' then
show_view('dd');
else
show_view('sen_pos_can');
end if;
end;

message('If you want to see the information about the sensor, click the sensor_no.');
end if;
if ( lower(get_item_property('test.test_date', displayed))='true' ) then
    set_item_property('test.test_date', displayed, property_false);
end if;
if ( lower(get_item_property('test.sps_no', displayed))='true' ) then
    set_item_property('test.sps_no', displayed, property_false);
end if;
if ( lower(get_item_property('test.section_id', displayed))='true' ) then
    set_item_property('test.section_id', displayed, property_false);
end if;
if ( lower(get_item_property('test.section_name', displayed))='true' ) then
    set_item_property('test.section_name', displayed, property_false);
end if;
if ( lower(get_item_property('test.speed', displayed))='true' ) then
    set_item_property('test.speed', displayed, property_false);
end if;
if ( lower(get_item_property('test.temp', displayed))='true' ) then
    set_item_property('test.temp', displayed, property_false);
end if;
if ( lower(get_item_property('test.truck_id', displayed))='true' ) then
    set_item_property('test.truck_id', displayed, property_false);
end if;
slen := length(itm);
itm := substr(itm, 1, slen-4);
if (lower(itm)='temp') then
    :test.input := 'Temperature';
elseif (lower(itm)='section_name') then
    :test.input := 'Material';
else
    :test.input := itm;
end if;
setitempro(blk,itm);
end;

--******************************************************************************
--SPS_DATA (When-Mouse-Down)
nnotoview('sps_data');

--******************************************************************************
--BTEST (When-New-Item-Instance)
Set_Block_Property('btest',CURRENT_RECORD_ATTRI

--When-Image-Pressed(form level)
declare
    it_id item;
lbl char(80);
itm char(40) := :System.current_Item;
blk char(40) := :system.current_block;
sensorno integer;
begin
  it_id := Find_Item(blk||''||itm);
  itm:=substr(itm,2);
  --:sen_type.name:=itm;
sensorno := to_number(itm);
  --:sen_pos.X:=sensorno;
  :sen_pos.id:=sensorno;
  pointsensor(blk, itm);
  --:showsensorshape(:sen_pos.type);
  imagesen(:sen_pos.type, :sen_type.name, :sen_type.manufacturer);
  imagepara(:sen_pos.type, :sen_type.parameter);
  imagemeas(:sen_pos.type, :sen_type.measurement);
  imagesection(:sen_pos.type, :sen_type.section);
end;