HP 3000 and HP 9000 Computer Systems

Up and Running with ALLBASE/SQL

Printed in U.S.A. 19901201

First E1290
Customer Order Number 36389-90011
DRAFT 9/12/97 20:40

Print History

The following table lists the printings of this document, together with the respective release dates for each edition. The software version indicates the version of the software product at the time this document was issued. Many product releases do not require changes to the document. Therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	Date	Software Version
First	December 1990	36217-02A.E0.00 (Series 800 HP-UX 7.08) 36217-02A.E1.00 (Series 800 HP-UX 8.0) HP79725A.E1.00 (Series 300 HP-UX 8.0) HP79725A.E1.00 (Series 400 HP-UX 8.0) 36216-02A.E1.00 (Series 900 MPE XL 3.0)

Note

The E releases of ALLBASE/SQL for use with HP-UX (36217-02A.E0.00, 36217-02A.E1.00, and HP79725A.E1.00) are compatible with both the 7.08 and the 8.0 releases of the HP-UX operating system. All references in this document to the 8.0 HP-UX release also apply to the 7.08 release. The E release of ALLBASE/SQL for use with MPE XL (36216-02A.E1.00) is compatible with the 3.0 release of the MPE XL operating system.

ALLBASE/SQL MPE XL Documents

General Reference

- ALLBASE/SQL Reference Manual (36216-90001)
- ALLBASE/ISQL Reference Manual (36216-90004)
- ALLBASE/SQL Message Manual (36216-90009)
- ALLBASE/SQL Quick Reference Guide (36216-90038)
- HP ALLBASE/QUERY User Guide (92534-90001)

Database And Network Administration

- ALLBASE/SQL Database Administration Guide (36216-90005)
- ALLBASE/NET User's Guide (36216-90031)
- ALLBASE/Turbo CONNECT Administrator's Guide (36385-90001)

Embedded SQL Programming Guides

- ALLBASE/SQL C Application Programming Guide (36216-90023)
- ALLBASE/SQL FORTRAN Application Programming Guide (36216-90030)
- HP SQL Pascal Application Programming Guide (36216-90007)
- HP SQL COBOL Application Programming Guide (36216-90006)

4GL Programming Guides

- HP ALLBASE/4GL Developer Administration Manual (30601-64001)
- HP ALLBASE/4GL Developer Reference Manual (30601-64002))
- HP ALLBASE/4GL Developer Self-Paced Training Guide (30601-64003)
- HP ALLBASE/4GL Run-Time Administration Manual (30602-64001)

LG200145_004

ALLBASE/SQL HP-UX Documents

General Reference

- ALLBASE/SQL Reference Manual (36217-90001)
- ALLBASE/ISQL Reference Manual (36217-90004)
- ALLBASE/SQL Message Manual (36217-90009)
- ALLBASE/SQL Quick Reference Guide (36217-90012)
- HP ALLBASE/QUERY User Guide (92523-90001)

Database And Network Administration

- ALLBASE/SQL Database Administration Guide (36217-90005)
- ALLBASE/NET User's Guide (36217-90093)

Embedded SQL Programming Guides

- ALLBASE/SQL C Application Programming Guide (36217-90014)
- ALLBASE/SQL FORTRAN Application Programming Guide (36217-90013)
- HP SQL Pascal Application Programming Guide (36217-90007) (79725-90010)
- HP SQL COBOL Application Programming Guide (36217-90058)

4GL Programming Guides

- HP ALLBASE/4GL Developer Administration Manual (92440-90027)
- HP ALLBASE/4GL Developer Reference Manual (92440-90005)
- HP ALLBASE/4GL Developer Self-Paced Training Guide (92440-64003)
- HP ALLBASE/4GL Run-Time Administration Manual (92440-64001)

LG200145_003

Preface

This book describes how to get an ALLBASE/SQL DBEnvironment up and running in the shortest possible time. ALLBASE/SQL is Hewlett-Packard's relational database management system, which is offered on HP 3000 computers using the MPE XL operating system and on HP 9000 computers using the HP-UX operating system.

This manual contains basic information about ALLBASE/SQL database design, creation, and administration. It is intended for new users of ALLBASE/SQL. Topics are discussed in separate chapters, as follows:

- Chapter 1, "Very Basic ... ," presents basic ideas, tasks, and concepts.
- Chapter 2, "Looking at Data," shows how to create an elementary database design before you create an ALLBASE/SQL DBEnvironment.
- Chapter 3, "Setting Up a Database with ISQL," takes you step by step through the process of configuring a DBEnvironment, then creating and loading tables.
- Chapter 4, "Practice with ALLBASE/SQL Using PartsDBE," details the steps for setting up PartsDBE, the ALLBASE/SQL sample DBEnvironment, which is used for most of the examples in the ALLBASE/SQL documentation.
- Chapter 5, "Comparing ALLBASE/SQL with TurboIMAGE," describes ALLBASE/SQL from the perspective of the TurboIMAGE user, maps ALLBASE/SQL concepts to TurboIMAGE concepts, and describes ALLBASE/TurboCONNECT.
- Chapter 6, "Glossary," gives basic definitions of terms used in ALLBASE/SQL.

Contents

1. **Very Basic ...**

What Is a Database? 1-1
What Is a Relational Database? 1-2
Rows and Columns 1-3
Sample Database Table 1-3
Data Types and Sizes 1-4
Using Several Tables 1-4
What Is SQL? 1-5
What Is ALLBASE/SQL? 1-5
SQLCore and DBCore 1-5
ISQL 1-6
ALLBASE/Query 1-6
SQLUtil 1-6
SQLGEN 1-6
ALLBASE/4GL 1-6
Preprocessors 1-6
ALLBASE/Net 1-7
ALLBASE/Turbo CONNECT 1-7
Other Products 1-7
What Is a DBEnvironment? 1-8
The DBECon File 1-9
DBEFiles 1-10
DBEFileSets 1-10
Databases 1-10
Tables and Indexes 1-11
System Catalog 1-11
Log Files 1-12
How Do I Create a DBEnvironment? 1-12
How Do I Create a Database? 1-13
Commands to Create Databases 1-13
How Do I Access a Database? 1-13
Queries and Other Data Manipulation 1-14
How Do I Control Database Access? 1-14
Where Can I Get Help with ALLBASE/SQL? 1-15

2. Looking at Data

Understanding the Process 2-2
A Small Sample Database 2-2
How Will the Data Be Used? 2-3
Distinguishing Entities and Attributes 2-3
Listing Entities 2-3
Listing Attributes 2-4
Identifying Relationships between Entities 2-4
Locating Distinguishing Key Items 2-5
From Entities to Tables 2-5
Creating the Table Design 2-6
Data Type and Size 2-6
Character Data 2-6
NULL Values 2-6
Modifying the Table Design 2-7
Table Descriptions 2-7
Defining Indexes 2-8
Designing Database Security Schemes 2-9
Estimating Table and Index Size 2-10
Designing Applications 2-10
Further Information 2-10

3. Setting Up a Database with ISQL

Running ISQL 3-1
Creating a DBEnvironment 3-2
DBECon File 3-3
DBEFile0 3-3
Log File 3-3
Creating DBEFileSets 3-3
Creating DBEFiles for Table and Index Data . . . 3-4
Committing Work 3-5
Adding DBEFiles to DBEFileSets 3-6
Creating Tables 3-6
Creating the Albums Table 3-7
Creating the Titles Table 3-7
Entering Data into Tables 3-8
Entering Data with the SQL INSERT Command . 3-8
Entering Data with the ISQL LOAD Command . 3-9
LOADing from an INTERNAL File 3-9
LOADing from an EXTERNAL File 3-10
Performing Queries 3-11
Creating Views 3-12
Granting Authorities 3-12
Creating an Index 3-13
Location of Tables and Indexes 3-14
Examining the System Catalog 3-16
Updating Statistics in the System Catalog 3-17
In Review 3-18

4. Practice with ALLBASE/SQL Using PartsDBE

Setting up PartsDBE 4-2
Using SQLSetup 4-2
Creating PartsDBE 4-3
Using Setup Scripts 4-4
HP-UX Systems 4-4
MPE XL Systems 4-4
Looking at the Files Created for PartsDBE . . . 4-4
HP-UX Systems 4-4
MPE XL Systems 4-6
Examining PartsDBE 4-7
Examining the Tables and Views 4-8
View Definitions 4-9
Using the INFO Command 4-10
Examining Indexes 4-11
Examining the Authority Structure 4-12
Groups 4-12
Table Authorities 4-13
Column Authorizations 4-13
Using the Preprocessors 4-15
Sample Application Programs 4-15
For HP-UX: 4-16
For MPE XL: 4-17
Examining Startup Parameters with SQLUtil . . . 4-17
Creating a Schema File with SQLGEN 4-20
Purging PartsDBE 4-21

5. Comparing ALLBASE/SQL with TurboIMAGE

Basic Structures 5-2
Procedures for Starting Up 5-3
Use of a Schema 5-4
Root File versus DBECon File and System Catalog 5-4
Data Files for Data sets versus DBEFiles for Tables 5-4
Naming Conventions 5-5
Tables and Indexes versus Data Sets 5-5
Automatic Masters versus Indexes 5-5
Manual Masters versus Hash Structures 5-6
Master/Detail versus Referential Integrity 5-6
Sort Items versus Indexes 5-6
Mapping of Data Types 5-7
Basic Mapping 5-7
Compound Items 5-7
Null Handling 5-7
Differences in Security 5-8
TurboIMAGE Security 5-8
Granting and Revoking Authorities 5-8
Defining ALLBASE/SQL Groups 5-8
Defining Views in ALLBASE/SQL 5-8
Differences in Accessing Databases 5-9
Interactive Access 5-9
Programmatic Access 5-9

4GL 5-9
Differences in Concurrency Control 5-9
Locking Mechanisms 5-10
Sample Mapping of a TurboIMAGE Database to an ALLBASE/SQL DBEnvironment 5-11
Using ALLBASE/Turbo CONNECT 5-13

6. Glossary of Terms in ALLBASE/SQL

Ad Hoc Query 6-1
Archive Logging 6-1
Attribute 6-1
Authority 6-2
Authorization Group 6-2
Base Table 6-2
Class 6-2
Clustering Index 6-2
Column 6-2
Column Authorization 6-2
Column List 6-2
Concurrency 6-2
Constraint 6-2
Cursor Stability (CS) 6-2
Data Analysis 6-3
Database 6-3
Database Administrator (DBA) 6-3
Database Design 6-3
Data Control Language 6-3
Data Definition 6-3
Data Definition Language 6-3
Data Manipulation 6-3
Data Manipulation Language 6-3
Data Type 6-3
DBA Authority 6-4
DBCore 6-4
DBECon File 6-4
DBECreator 6-4
DBEFile 6-4
DBEFileSet 6-4
DBEnvironment 6-4
DBEUserID 6-4
Embedded SQL Program 6-5
Entity 6-5
Explicit Locking 6-5
Expression 6-5
Foreign Key 6-5
Group 6-5
Hash Structure 6-5
Host Variable 6-5
Implicit Locking 6-6
Index 6-6
Index Scan 6-6

Integrity Constraint 6-6
ISQL 6-6
Isolation Level 6-6
Join 6-6
Key 6-6
Key Column 6-6
Key Value 6-7
Locking 6-7
Logging 6-7
Message Catalog 6-7
Message File 6-7
Modified Source File 6-7
Module 6-7
Native Language 6-7
Nonarchive Logging 6-8
Normalization 6-8
Object 6-8
Optimizer 6-8
Owner 6-8
Predicate 6-8
Preprocessor 6-8
Primary Key 6-8
Projection 6-9
Query 6-9
Query Language 6-9
Query Result 6-9
Read Committed (RC) 6-9
Read Uncommitted (RU) 6-9
Referential Constraint 6-9
Relation 6-9
Relational Operations 6-9
Relationship 6-9
Repeatable Read (RR) 6-9
Result Table 6-10
Rollback Recovery 6-10
Rollforward Recovery 6-10
Row 6-10
Run Authority 6-10
Schema 6-10
Section 6-10
Serial Scan 6-10
Special Authority 6-10
SQL 6-11
SQLCore 6-11
SQLGEN 6-11
SQLMigrate 6-11
SQLUtil 6-11
Structured Query Language 6-11
Subquery 6-11
SYSTEM 6-11
System Catalog 6-11

System Table 6-12
System View 6-12
Table 6-12
Table Authority 6-12
Transaction 6-12
Unique Constraint 6-12
Unique Index 6-12
Validation 6-12
View 6-12

Figures

1-1. Relational Operations 1-3
1-2. ALLBASE/SQL DBEnvironment 1-8
3-1. ISQL Banner 3-2
3-2. System.Table Display 3-17
4-1. SQLSetup Menu 4-3
4-2. Information on Tables and Views 4-8
4-3. View Definitions in the System Catalog 4-9
4-4. Output of the INFO Command 4-10
4-5. System Catalog Information on Indexes 4-11
4-6. Groups in the System Catalog 4-12
4-7. Table Authorities in the System Catalog 4-13
4-8. Column Authorities in the System Catalog . . . 4-14
4-9. SQLUtil Banner 4-17
4-10. SQLGEN Banner 4-20
5-1. TurboIMAGE Architecture 5-2
5-2. ALLBASE/SQL Architecture 5-3
5-3. Using ALLBASE/Turbo CONNECT 5-13

Tables

5-1. Mapping of TurboIMAGE and ALLBASE/SQL Data Types 5-7

1

Very Basic . . .

Before venturing into the tasks you want to accomplish with ALLBASE/SQL, we give a bit of thought to some basic ideas, tasks, and concepts:

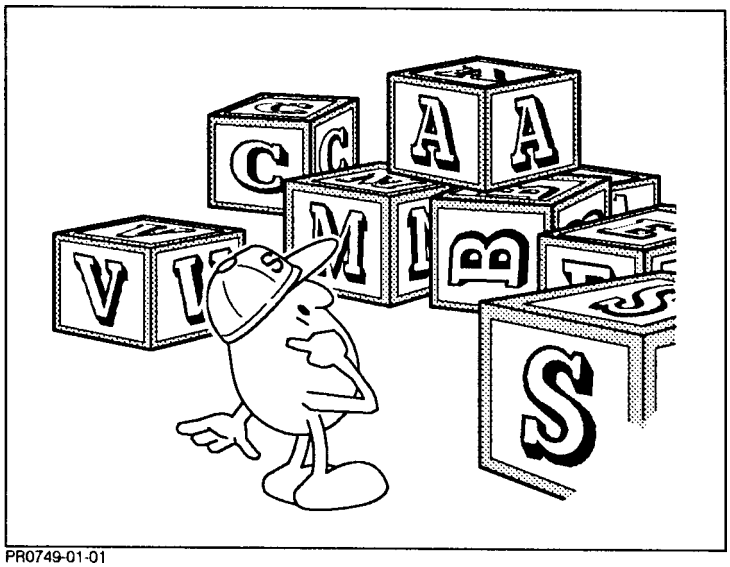
- What is a database?
- What is a relational database?
- What is SQL?
- What is ALLBASE/SQL?
- What is a DBEnvironment?
- How do I create a DBEnvironment?
- How do I create a database?
- How do I access a database?
- How do I control database access?
- Where can I get help with ALLBASE/SQL?

SQL stands for **Structured Query Language**, which is defined by ANSI standards in the United States and by X/OPEN standards in Europe. In addition to standard SQL, ALLBASE/SQL uses some terminology of its own. A prime goal in this chapter is to provide you with a "working vocabulary" in both standard SQL terminology and the additions used by ALLBASE/SQL. If you already have this working vocabulary, skip ahead to the next chapter, "Looking at Data." Then come back here whenever you want to review the basics. You can also refer to the glossary in chapter 6 for definitions of basic terms.

What Is a Database?

A **database** is a structured arrangement of data elements designed for the easy selection of information. Unlike a collection of flat files, a database contains both data and structural information used in extracting data from the files in which data resides.

In ALLBASE/SQL, databases are located inside a *DBEnvironment*, a structure that is fully defined later in this chapter.



PR0749-01-01

What Is a Relational Database?

A relational database is a collection of data arranged in **tables**, also known as **relations**. Tables are subject to the following **relational operations**, each of which lets you retrieve data in a specific way:

- **Selection**, which lets you extract a subset of rows.
- **Projection**, which lets you extract a subset of columns.
- **Joining**, which lets you extract from more than one table at a time.

In practice, these operations frequently appear together. SQL statements that use these operations are known as **queries**. Three queries that use the SQL SELECT command to illustrate selection, projection, and joining are shown in Figure 1-1.

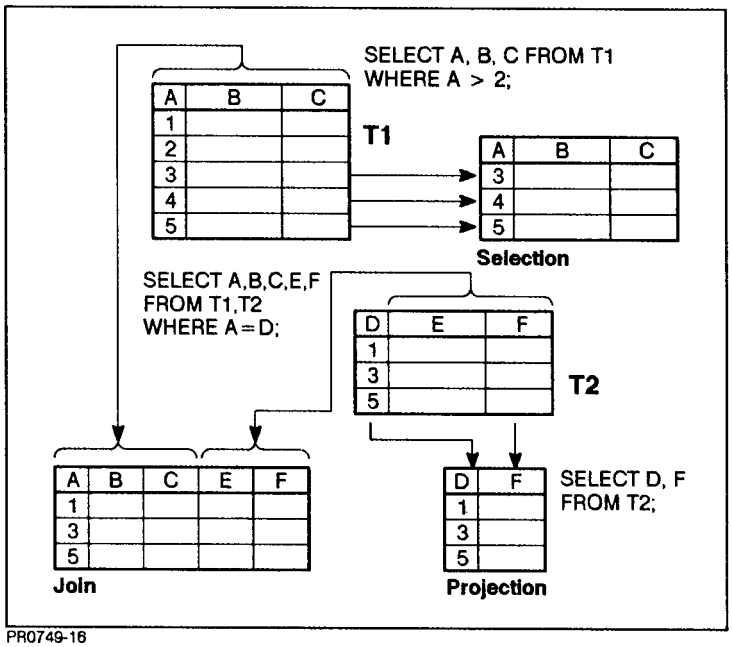


Figure 1-1. Relational Operations

Rows and Columns

When you look at data in relational terms, you can assume several things:

- Tables are arranged in **rows** and **columns**, which are like records and fields in an ordinary file.
- Each column has a specific data type and size.
- Each row contains one element for every column.
- A column can contain NULL values if you allow it to.

Sample Database Table

The following is a portion of a database table consisting of names and account balances for an employee credit union:

Employee Accounts

Last Name	First Name	Telephone	Employee Number	Balance
Harrison	Gerald	7233	2432099	142.59
Abelson	Annette	4312	3510044	2345.09
Stanley	Peter	NULL	3540011	321.98
Walters	Georgia	2554	9124772	1230.10

Notice that the third row contains a NULL value in the third column instead of a value for Telephone.

Data Types and Sizes

Each column can accept data of a specific type and size. Here is the breakdown for the sample table above:

Column Name	Data Type
Last Name	VARCHAR(15)
First Name	VARCHAR(15)
Telephone	SMALLINT
Employee Number	INTEGER
Balance	DECIMAL(10,2)

Data types are described further in chapter 2.

Using Several Tables

You can put the same data into several different tables such as the following:

Table 1. Employees Table

Last Name	First Name	Employee Number
Harrison	Gerald	2432099
Abelson	Annette	3510044
Stanley	Peter	3540011
Walters	Georgia	9124772

Table 2. Telephone Table

Last Name	First Name	Telephone
Harrison	Gerald	7233
Abelson	Annette	4312
Stanley	Peter	NULL
Walters	Georgia	2554

Table 3. Accounts Table

Employee Number	Account Balance
2432099	142.59
3510044	2345.09
3540011	321.98
9124772	1230.10

You decide which arrangements of data work best for you by using the processes of **data analysis** and **database design**.

In data analysis, you investigate the various ways your data can be used. In database design, you create specific table structures based on your analysis. The design phase results in a set of table descriptions, sometimes known as a schema, for your database.

Chapter 2 presents an introduction to data analysis and database design.

What Is SQL?

The way into a relational database is through a **query language**—which is a set of operators, **expressions**, and commands that let you manipulate the database in various ways. You create queries as well as other kinds of commands in ALLBASE/SQL by using SQL (Structured Query Language); and you issue the commands directly, through an interactive command processor, or indirectly, through an application program.

SQL includes commands that let you do the following:

- Create databases.
- Access databases.
- Provide security.
- Promote data integrity.
- Regulate concurrent access.

SQL commands are printed in capitals (for example, SELECT) in this book and throughout the ALLBASE/SQL document set.

What Is ALLBASE/SQL?

ALLBASE/SQL is Hewlett-Packard's proprietary relational database management system. Closely tuned to the architecture of HP computers, ALLBASE/SQL gives you enormous flexibility in designing and using SQL database applications on a small or large scale.

ALLBASE is a family of relational database products that includes the components of ALLBASE/SQL and several related tools defined in the following paragraphs.

SQLCore and DBCore

Two components which together form the back end of ALLBASE/SQL. **SQLCore** accepts SQL commands and processes them; **DBCore** performs file access operations at the operating system level and also controls concurrent access to data, guaranteeing consistency. To use SQLCore and DBCore, you employ a front-end process, such as **ISQL**, **ALLBASE/Query**, or one of your own application programs.

ISQL An interactive command processor which lets you enter SQL commands at the keyboard and observe query results, messages, and other information on a video display. Remember that SQL is a language—not a software system. So you need an interactive way to submit SQL commands to SQLCore and DBCore for processing.

ISQL is the main tool used by ALLBASE/SQL programmers and database administrators to create and modify the schema of an ALLBASE/SQL DBEnvironment. It is also used by anyone who needs to submit queries using the SQL language. ISQL is especially useful for loading and unloading data.

ALLBASE/Query A screen-oriented, menu-driven approach to designing queries and producing quick reports from data in ALLBASE/SQL databases. ALLBASE/Query contains extensive online help, so it is especially appropriate for occasional users.

SQLUtil A **database administrator**'s tool for displaying and setting the basic parameters of a DBEnvironment (explained further in the next section); storing and restoring DBEnvironments; setting the size of system buffers; and purging DBEnvironments. The database administrator (DBA) is the individual who creates and maintains objects in a DBEnvironment. **SQLUtil** is seldom needed by the ordinary user.

SQLGEN A database administrator's tool that examines the structure of a DBEnvironment and creates files of SQL and ISQL commands for unloading it, re-creating it, and reloading it with data. The file for re-creating a DBEnvironment is sometimes known as a **schema**. The schema shows you all the CREATE commands that went into the original development of the DBEnvironment.

ALLBASE/4GL A fourth-generation programming tool. ALLBASE/4GL is a screen-oriented, menu-driven environment that lets you design and build complete ALLBASE/SQL applications without conventional programming.

Preprocessors Tools that convert source programs containing SQL commands into source code that can be compiled in a programming language of your choice. Different preprocessors let you code applications in C, COBOL, FORTRAN, and Pascal.

ALLBASE/Net Software that permits you to set up and maintain DBEnvironments in networks. Through the NETUtil program, you make DBEnvironments on host systems available to local users.

ALLBASE/Turbo CONNECT Software on MPE XL systems that lets you attach a TurboIMAGE database to a DBEnvironment and then use SQL to query the TurboIMAGE database as if it were a set of ALLBASE/SQL tables.

Other Products Software that makes use of ALLBASE/SQL to meet specialized needs. Information Access/PC lets you access ALLBASE/SQL tables from your PC and integrate data from them into PC-based applications. Business Report Writer lets you develop complex reports using ALLBASE/SQL data as well as data from other sources.

What Is a DBEnvironment?

In ALLBASE/SQL, you create one or more databases in a structure called a DBEnvironment. The structure of a DBEnvironment is shown in Figure 1-2.

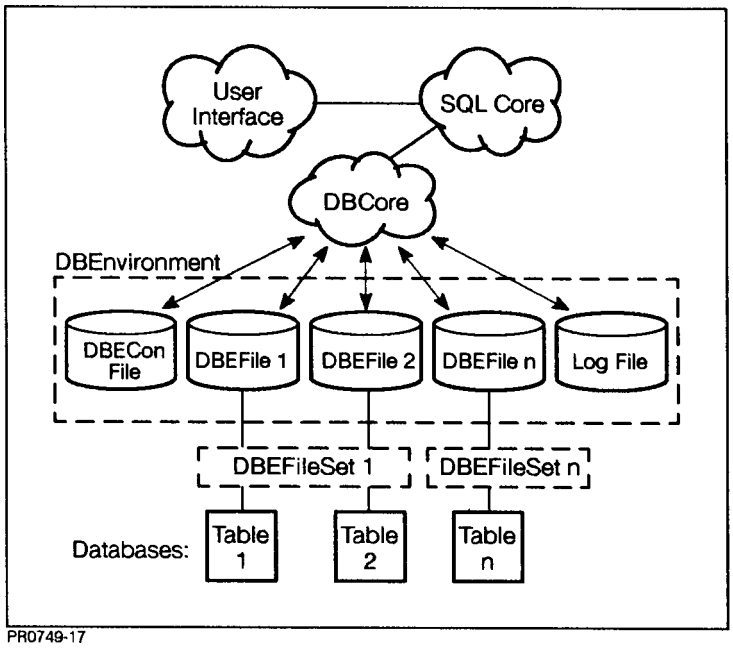
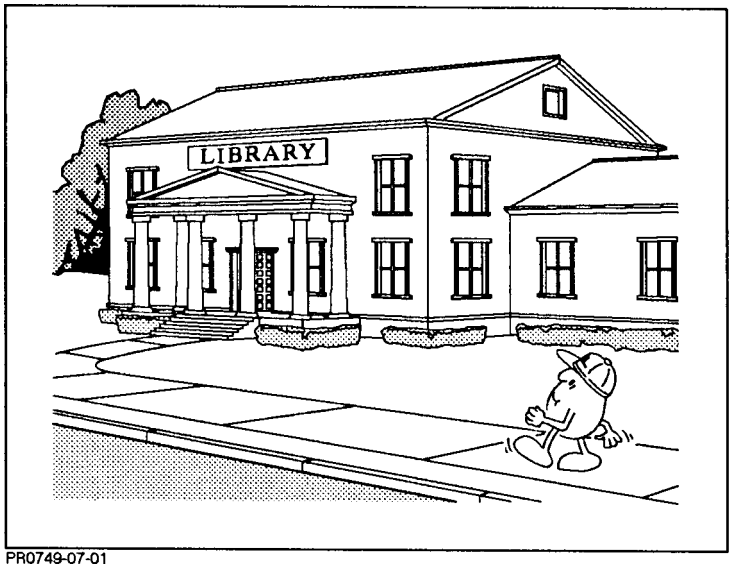


Figure 1-2. ALLBASE/SQL DBEnvironment

The following **objects** are the most important parts of the DBEnvironment:

- The DBECon file.
- DBEFiles.
- DBEFileSets.
- Databases.
- Tables and indexes.
- Authorities.
- System catalog.
- Log files.

Objects are structures created and stored in an ALLBASE/SQL DBEnvironment. SQLCore and DBCore connect the user interface, such as ISQL or an application program, with the objects of the DBEnvironment.

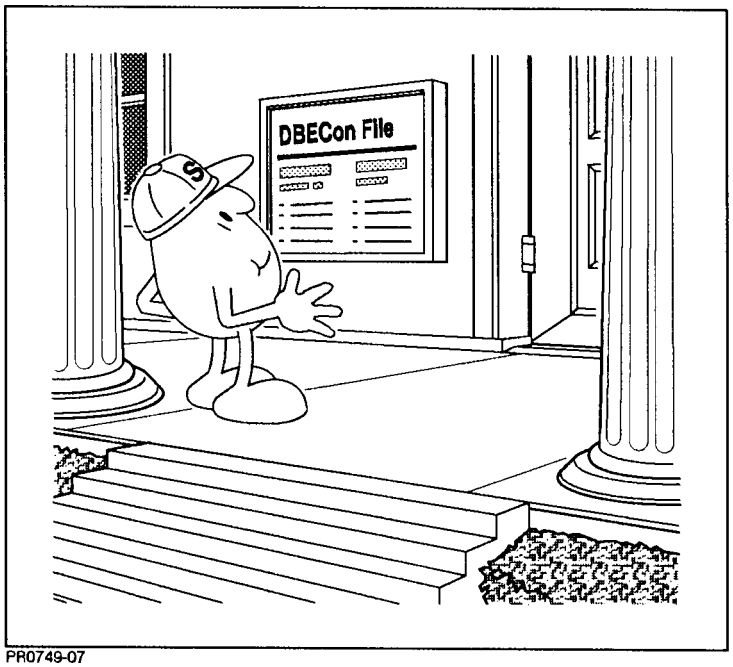


PR0749-07-01

To better understand the DBEnvironment and its objects, imagine that it is like a library, which is used to store books, periodicals, or other information. Like a library, the DBEnvironment is a *physical location* for information, so you need to set aside physical space for it. Also like a library, the DBEnvironment uses a *logical method* for storing and retrieving information.

The DBECon File

When you create a DBEnvironment, ALLBASE/SQL creates a physical file known as the DBEnvironment Configuration File or **DBECon file**. This file contains basic information that is used every time the DBEnvironment is opened. The DBECon file has the same name as the DBEnvironment itself. If the DBEnvironment is like a library, the DBECon file is like a building directory that points to the other DBEnvironment components.



PR0749-07

DBEFiles You must allocate physical storage space by creating **DBEFiles**—operating system files that hold table data, index data, or both. Like the individual bookcases in the library, they have a specific capacity. DBEFiles have both physical names—operating system names—and logical names, by which the files are known internally to the ALLBASE/SQL system catalog, to be explained shortly.

DBEFileSets DBEFiles are grouped together in logical groupings known as **DBEFileSets**. These are something like the different subject categories in a library. You create a DBEFileSet with a logical name (this corresponds to the subject category name). Then you add DBEFiles to the DBEFileSet as needed, as you would add bookcases to the category to add more space. For example, a section may be labeled "Computer Periodicals" and hold many bookcases with different magazines in them. DBEFileSets do not have a specific capacity; you can always add more DBEFiles to create more space in them.

Databases The data in a DBEnvironment is stored in databases, which are groups of tables having the same **owner**. The database in a DBEnvironment is a bit like division ownership of certain books or periodicals in a library. In a university, some of the periodicals may belong to a computer science library, others may belong to a medical library, though these may be housed inside the same building and use the same card catalog. In such a subdivision of the library, certain books and periodicals could only be checked out by people belonging to the appropriate division or having special permission.

When you create an object in the DBEnvironment, you are its owner by default, and therefore it belongs to your database.

Tables and Indexes

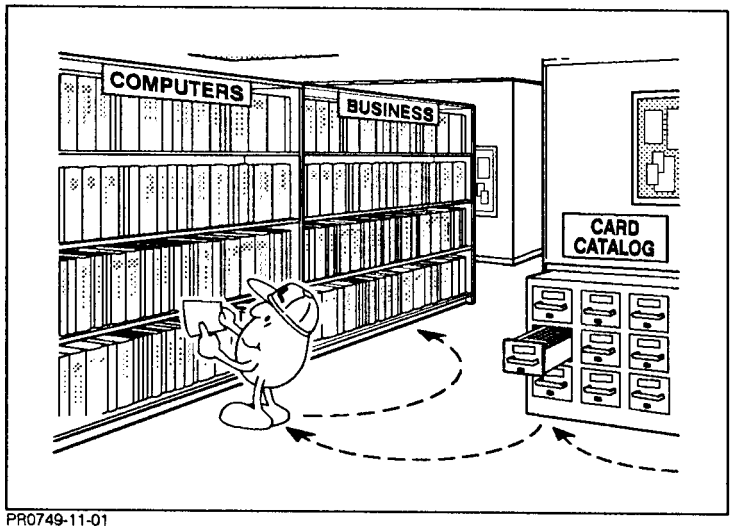
The most important objects you create are tables. The table is like a periodical stored in the library, and the rows in the table are like individual articles in the periodical. The **index** on a table is like the index to the content of a group of periodicals in the bookcase. The index may appear on the same shelf as the periodical or it may be located along with other indexes in a different bookcase.

Tables and indexes have only logical names; a table name is like the title of the periodical. When you create a table, you only need to define its name and characteristics and associate it with a DBEFileSet. This is like adding a new periodical to a subject category in the library.

More detailed information about the organization of data in tables is presented in chapter 2, "Looking at Data."

System Catalog

To enable readers to find issues of *ComputerWorld*, the librarian puts an entry in a card catalog that shows where the issues are stored. In the DBEnvironment, SQLCore inserts an entry in the **system catalog** for each database object you create. The system catalog is created at the time you create a DBEnvironment.



PR0749-11-01

The system catalog is physically located in a DBEFile known as DBEFILE0. This file contains extra space for the additional entries that are made by SQLCore each time you create a new object. Like a good librarian, ALLBASE/SQL keeps the system catalog up-to-date for you, without your being aware of it.

The system catalog is a set of tables for internal use by SQLCore. These tables are associated with a DBEFileSet known as **SYSTEM**. SQLCore uses the system catalog to look up database objects

much as you would use the card catalog in a library to look up the bookcase and shelf number for a book or periodical.

Log Files

ALLBASE/SQL provides **logging** of all transactions that take place in the DBEnvironment. Like the library's record of items checked out and returned, the log file is a record of the rows in database tables that are added, deleted, or changed. The log makes it possible to keep data consistent when multiple users are accessing the system, and it makes recovery possible in the event of a system failure. Information about the system log is stored in the DBECon file.

Because logging is essentially automatic, it will not be discussed any further in this book. For more information, you can refer to the chapter entitled "Backup and Recovery" in the *ALLBASE/SQL Database Administration Guide*.

How Do I Create a DBEnvironment?

You use the SQL START DBE NEW command to create a DBEnvironment:

```
START DBE 'DBEName' NEW
```

Once the DBEnvironment exists, you can create databases within it.

The START DBE NEW command lets you supply options to specify many of the run-time characteristics of the DBEnvironment. The simple form of the command shown above uses default values for these options. The defaults are described in detail in the *ALLBASE/SQL Database Administration Guide* chapter entitled "DBEnvironment Configuration and Security."

When you create a DBEnvironment, you are granted the broadest kind of **authority**—permission in an ALLBASE/SQL DBEnvironment—to create and remove objects. This authority is known as DBA (database administrator) authority.

How Do I Create a Database?

To create a database, you need to perform at least some of the following tasks:

- Create tables.
- Create views.
- Create indexes.

These tasks are part of the **data definition** process, which also includes placing the tables in specific DBEFilesets, and using the SQL DROP command to remove tables or views that are no longer needed.

Most data definition tasks are carried out when you are setting up the database for the first time. But you may need to do some data definition tasks later, when, for example, you need additional database capacity.

Commands to Create Databases

You use SQL CREATE commands to create database tables and all the other components of a database. Each of these commands is shown in more detail later in this book; the following is only one example:

```
CREATE PUBLICREAD TABLE Employees
(LastName VARCHAR(15) NOT NULL,
FirstName VARCHAR(15) NOT NULL,
EmpNumber INTEGER NOT NULL)
```

After creating a table, you use the ISQL INPUT and LOAD commands or an application program to put data into it. You can also add single rows to a table by using the SQL INSERT command. (These tasks are explained and illustrated in chapter 3.)

How Do I Access a Database?

You use the SQL CONNECT command to establish a connection to the DBEnvironment. You can then access a particular database as follows:

- Through ISQL.
- Through application programs you create yourself.

During database access, you perform queries or other operations that manipulate data by inserting, deleting, or modifying rows in tables. This process is called **data manipulation**.

Queries and Other Data Manipulation

Data manipulation commands access the data in databases. An example is a query using the SQL SELECT command, which displays a selection of data from database tables. Here is a simple query for information from the Employees table. The asterisk means *all* rows and columns:

```
SELECT * FROM Employees;
```

Below is the **query result**:

```
select * from employees;
--------------+-----------+-----------
LASTNAME      |FIRSTNAME  |EMPNUMBER
--------------+-----------+-----------
Harrison      |Gerald     |    2432099
Abelson       |Annette    |    3510044
Stanley       |Peter      |    3540011
Walters       |Georgia    |    9124772
```

The query result is also known as a **result table**.

Other data manipulation commands include the SQL INSERT, UPDATE, and DELETE commands. These let you add rows to a table, update specific column values in existing rows, or delete rows. Many examples of data manipulation are shown in later chapters.

How Do I Control Database Access?

You use **data control language** to determine who has access to the information in a database. This is very important for security. Data control language confers authorities on specific users to perform specific tasks. The most powerful authority is known as **DBA authority** (database administrator authority). The **DBECreator**, that is, the person who creates a new DBEnvironment, automatically has DBA authority in that DBEnvironment. Someone with DBA authority can then use the GRANT command to give authorities to other users. The following example grants permission to update the Employees table to a user known as Henry:

```
GRANT UPDATE ON Employees to Henry
```

The controls you define for database security can be as simple or as elaborate as you wish.

Where Can I Get Help with ALLBASE/SQL?

Because ALLBASE/SQL has so many components and tasks, it is helpful to know where to find information about each one. Here is a list of the documents in the ALLBASE/SQL document set, preceded by a short title:

DBA *ALLBASE/SQL Database Administration Guide*
SQL *ALLBASE/SQL Reference Manual*
ISQL *ALLBASE/ISQL Reference Manual*
APG Four application programming guides, which explain the use of the preprocessors:
- *ALLBASE/SQL C Application Programming Guide*
- *ALLBASE/SQL COBOL Application Programming Guide*
- *ALLBASE/SQL Pascal Application Programming Guide*
- *ALLBASE/SQL FORTRAN Application Programming Guide*

MSG *ALLBASE/SQL Message Manual*
QRG *ALLBASE/SQL Quick Reference Guide*
4GL *ALLBASE/4GL Documentation*
ABQ *ALLBASE/Query Documentation*
NET *ALLBASE/NET User's Guide*
ATC *ALLBASE/TurboCONNECT User's Guide*

The following table shows which manuals contain information about each task. Part numbers for each manual appear in the documentation map in the front of this book.

Information about ALLBASE/SQL Tasks

Task	Title	Section Reference
Creating a DBEnvironment	DBA	"DBEnvironment Configuration and Security"
Creating, Dropping Tables	DBA SQL	"Database Creation and Security" "SQL Commands"—CREATE TABLE and DROP TABLE.
Accessing Databases	ISQL SQL ABQ APG	"Using ISQL" section on Queries "SQL Queries" All sections "Simple Data Manipulation"
Granting Authorities	SQL	"SQL Commands"—GRANT and REVOKE.
Loading, Unloading Data	ISQL DBA DBA	"Using ISQL" and "ISQL Commands"—LOAD and UNLOAD "Maintenance" "SQLGEN" appendix
Maintaining DBEnvironments	DBA DBA	"Maintenance" "SQLUtil" appendix
Migrating DBEnvironments	DBA DBA	"Tasks and Tools" "SQLMigrate" appendix
Creating ALLBASE applications	APG, 4GL	All sections
Setting up and Using ALLBASE/SQL Networks	NET	All sections
Setting up and Using TurboIMAGE Access	ATC	All sections
Checking Syntax	QRG	All sections
Understanding Warnings and Errors	MSG	All sections

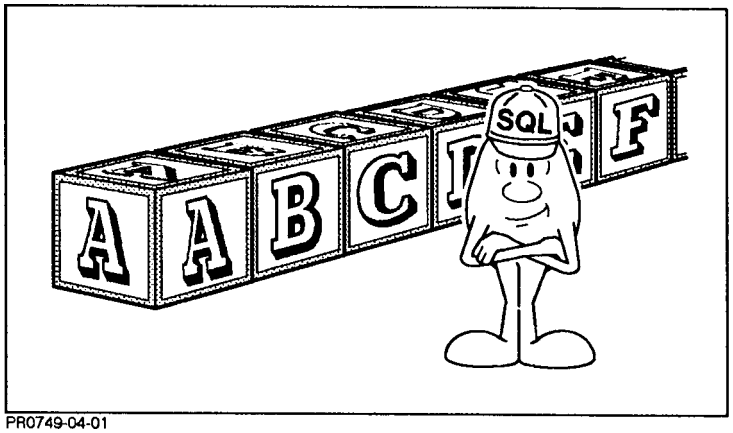
2

Looking at Data

This chapter presents the basic steps in data analysis and database design:

- Understanding the process.
- Distinguishing entities and attributes.
- Identifying relationships between entities.
- Locating distinguishing key items.
- Creating the table design.
- Defining indexes.
- Defining views.
- Estimating table and index size.
- Designing applications.

These terms are explained as each step is discussed in the following sections. If you already have a table design in mind and you want to set it up at once, skip this chapter; come back later if you wish. But if you don't know what data to put into which tables, then read on!



PR0749-04-01

Understanding the Process

Designing a database to be built with ALLBASE/SQL means examining the data you wish to store and then putting it into a form that ALLBASE/SQL can understand. In other words, you look at the logical relationships that exist within the data and then create a relational design (tables, views, indexes, etc.). The process can be complex, and you could use a number of formal design methodologies. However, you can get a good first approximation by using an intuitive approach, which is sketched very briefly in this chapter.

Database design is the subject of much theoretical discussion and debate, but everyone agrees that good design results in good performance. Good database design also gives you the greatest flexibility in formulating your queries and in restructuring your databases when that becomes necessary. For these reasons, time spent on analysis and design "up front" results in time and money saved in developing a production system.

A reading list at the end of this chapter provides references to additional information. If you are about to embark on a complex design, be sure to consult this material.

A Small Sample Database

To examine some data modeling techniques, let's imagine a small sample database. Suppose a radio station wants to create a database of classical music recordings for use by program directors and announcers. The station needs this information to plan a schedule of broadcasts, to maintain a log of what is played, and to publish a monthly listener's guide. Here are some specific data items that will be needed:

- Recording company and date recorded.
- Album title.
- Album price.
- Medium.
- List of selections and timings.
- Total timing of each album.
- Names of orchestra, conductor, singers, accompanists.
- Composer's name, birthplace, and dates of birth and death.
- Comments on composers, albums, and selections.
- Date, time, and announcer for each selection played.

You can probably think of other items of information that might be useful (for example, the date the album was acquired), but let's use these for now. How should this data be organized?

PR0749-02-1

How Will the Data Be Used?

A typical user of the database would be the program director, who might ask questions like the following:

What selections do I have by Beethoven that are less than 20 minutes long?

What did we broadcast last year on Beethoven's birthday?

How many different versions of Beethoven's Fifth Symphony do we have, who are the conductors, and what are the timings?

What composers represented in our library were born in March?

What selections did George play last Tuesday morning on his chamber music show?

Distinguishing Entities and Attributes

As you begin to design the music database, you are either creating a new system or you are transferring data from a non-relational system to ALLBASE/SQL. Regardless of where you're coming from, you need to take a comprehensive look at all the information needs that will be served by the database system. This means identifying the elements for which you need to store and retrieve information. These elements are known as **entities**. A list of entities grows out of studying how the data is used by its owners.

Listing Entities

By approaching the problem intuitively, you can probably identify four different categories of information required by the radio station:

- Album information.
- Selection information.
- Composer information.
- Station log information.

These are the entities in the data.

Listing Attributes

Next, you need to define the **attributes** of each entity, which are the useful pieces of information to be stored in tables. In addition to supplying informational detail, some attributes are used to distinguish one entity from another. As you subdivide your data, make sure that for each entity you define, at least one attribute can uniquely identify an instance of the entity. This attribute or group of attributes is known as a **key**.

Attributes for Four Entities

Album Entity	Selection Entity	Composer Entity	Station Log Entity
Album Name	Selection Title	Name	Selection Title
Medium	Composer Name	Date of Birth	Start Time
Album Cost	Timing	Birthplace	End Time
Recording Company	Performers	Comment	Announcer
Date Recorded	Comment		Comment
Manufacturer's Code			
Comment			

As the design evolves, entities eventually become database tables, and attributes eventually become columns. Note, however, that at this stage you have not yet identified the form of the database tables. Before you can do that, you need to identify **relationships**.

Identifying Relationships between Entities

After subdividing the data by entities, the next step of design is to identify meaningful relationships between the entities described so far. For each relationship you identify, an attribute or group of attributes must support the relationship.

What are the relationships among the entities in the sample data? For the *Album* and *Selection* entities, there is a relationship of *Content*, that is, each album *contains* a specific group of selections. For the *Composer* and *Selection* entities, the relationship is one of *Authorship*; each composer has *created* one or more selections.

Another kind of relationship exists between *Selection* and *Composer*, namely, *Period*. The attributes that define *Period* are the composer's "Name" and "Birth Date."

Locating Distinguishing Key Items

Each entity should have one or more attributes which can uniquely distinguish a particular occurrence of that entity. Each distinguishing attribute or group of attributes is known as a **key value**. Also, for each relationship you have defined, an attribute or group of attributes should specify the relationship by forming a link between the entities. In the case of *Station Log* and *Selection*, the links are "Selection Name" and "Selection Title." In the case of *Selection* and *Composer*, the link is "Composer Name," which is an attribute of *Composer*, but would need to be added to the list of attributes for *Selection*. In the case of *Album* and *Selection,* there is a possible link in "Album Code," which would have to be added to the list of attributes for *Selection*.

From Entities to Tables

Once you have added the necessary key attributes that support relationships, you have arrived at a set of relational tables: the attributes are now columns, and the key values are now **key columns**. From what has been done so far, you can see a set of relational tables emerging, as follows (* indicates a key column):

Columns and Keys for Four Tables

Albums Table	Selections Table	Composers Table	Log Table
*AlbumCode	*AlbumCode	*ComposerName	*AlbumCode
AlbumTitle	*SelectionName	Birth	*SelectionName
Medium	ComposerName	Death	StartTime
AlbumCost	Timing	Birthplace	EndTime
RecordingCo	Performers	Comment	Announcer
DateRecorded			
MfgCode			
Comment			

Note that the Selections and Log tables each have two key columns, whereas the other tables have only one apiece. Remember that a key must uniquely identify each entry in the table.

Creating the Table Design

The next step is to define the characteristics of each column you have defined. For each potential column value, you need to answer the following questions:

- What is the **data type** and size?
- For character data, should values be fixed or variable length?
- Are null values allowed?

Data Type and Size

Some possible data types in ALLBASE/SQL are:

CHAR	Fixed length character string.
VARCHAR	Variable length character string.
INTEGER	Four-byte integer values.
SMALLINT	Two-byte integer values.
DECIMAL	Fixed-point packed decimal values.
FLOAT	Floating point numbers.
DATE	Date values.
TIME	Clock time values.
DATE-TIME	Timestamp values (date and time combined).
INTERVAL	Elapsed time values.

Decide whether you wish to use numeric or alphanumeric (character) data types. If a column value needs to participate in arithmetic operations, it should be either a numeric or date/time data type.

Make sure that your data types are consistent from table to table where columns are to be joined. For example, when you want to look up all the selections for a specific album title, the AlbumCode column in the Albums table must be consistent with the AlbumCode column in the Selections table. If it is not, you may not get all the data you expect.

Character Data

In the case of character data, decide whether the type should be fixed length (CHAR) or variable length (VARCHAR). When a character column will contain values of uniform size, such as two-character alphabetic codes, use CHAR. If the size of values is not expected to be uniform, use a VARCHAR type, and specify the expected maximum size.

NULL Values

Consider whether or not information will be available when deciding whether to permit NULL values. A NULL value is the absence of data for a specific column. For example, you might permit NULL values in the ComposerName column of the Selections table, because a selection may be anonymous; but you should *not* permit the ComposerName column in the Composers table to be NULL.

Note 

Key columns should never be NULL.

Modifying the Table Design

One further consideration: Is all the data within a particular entity accessed at the same time? If not, it may be wise to consider subdividing as you convert the entity into a table description. You might move some of the information in the Albums table to a different table if it is not used very often.

A related consideration: Is some of the data from two or more tables always accessed together? In this case, consider combining two tables into one. For example, you might include AlbumCode and AlbumTitle in the Selections table if these data items are always included with other data about selections.

The formal term for modifying the table design by examining the relationships among columns is called **normalization**, which is described in the "Logical Design" chapter of the *ALLBASE/SQL Database Administration Guide*. Also, refer to the list of references at the end of this chapter.

Table Descriptions

When you have made decisions on these issues, you can create a description of each table, as follows:

Albums Table

Table Name	Column Name	Data Type	NOT NULL	Size
Albums	AlbumCode	INTEGER	NOT NULL	4 bytes
	AlbumTitle	VARCHAR(40)		40 bytes
	Medium	CHAR(2)		2 bytes
	AlbumCost	DECIMAL(6,2)		4 bytes
	RecordingCo	CHAR(10)	NOT NULL	10 bytes
	DateRecorded	DATE		16 bytes
	MfgCode	VARCHAR(40)		40 bytes
	Comment	VARCHAR(80)		80 bytes

Titles Table

Table Name	Column Name	Data Type	NOT NULL	Size
Selections	AlbumCode	INTEGER	NOT NULL	4 bytes
	Selection	VARCHAR(40)	NOT NULL	40 bytes
	ComposerName	VARCHAR(16)		16 bytes
	Timing	INTERVAL		16 bytes
	Performers	VARCHAR(40)		40 bytes
	Comment	VARCHAR(80)		80 bytes

Composers Table

Table Name	Column Name	Data Type	NOT NULL	Size
Composers	ComposerName	VARCHAR(16)	NOT NULL	16 bytes
	Birth	DATE		16 bytes
	Death	DATE		16 bytes
	Birthplace	VARCHAR(40)		40 bytes
	Comment	VARCHAR(80)		80 bytes

Log Table

Table Name	Column Name	Data Type	NOT NULL	Size
Log	AlbumCode	INTEGER	NOT NULL	4 bytes
	SelectionName	VARCHAR(40)	NOT NULL	40 bytes
	StartTime	DATETIME	NOT NULL	16 bytes
	EndTime	DATETIME	NOT NULL	16 bytes
	Announcer	VARCHAR(40)	NOT NULL	40 bytes

The sizes shown here are taken from the *ALLBASE/SQL Database Administration Guide* chapter on "Physical Design."

Defining Indexes

After you have designed the structure of the tables in the database, consider which columns are good candidates for the creation of indexes. An index is an object which you create after creating the table. It is not absolutely necessary to create an index on a table, but doing so can help ALLBASE/SQL point more quickly to the row you need. For example, in the Albums table, you might consider creating a unique index on the AlbumCode column, because using a unique index may allow SQLCore to arrive at the required row more quickly than by doing a scan of every row in the table. Also, the unique index guarantees that the AlbumCode is a unique number.

Suppose your application prints the album titles of all the albums containing selections by a specific composer. This requires a join between the Albums table and the Titles table. The join might be slow to execute because, first, ALLBASE/SQL would have to search every row of the Titles table to find the entries for the composer. Then it would have to search every row of the Albums table to find every match with the album code found in the Titles table along with the composer's name. The join column in this query is the AlbumCode column; if you create an index on the AlbumCode column of each table, the query might execute faster.

Furthermore, the AlbumCode column in the Albums table would be a good candidate for creation of a unique index, because the value in this column should not be duplicated. In the case of the Titles table, the index should not be unique, because the table can have many

rows with the same album code. That is, an album can contain many selections.

Some tables may also be good candidates for creation as **hash structures**, which are essentially self-indexed. For more information on this topic, refer to the chapter "Using SQL" in the *ALLBASE/SQL Reference Manual*.

Designing Database Security Schemes

You can provide security for data at the level of the DBEnvironment itself, or at the level of individual tables. At the DBEnvironment level, you can provide CONNECT authorization to just those users who need access.

ALLBASE/SQL also has several kinds of **TABLE authorities**, so that you can control the kind of access that is possible for different users of each column in each table. For each table, you should ask the following questions:

Which users need to SELECT?

Which users need to add new rows or delete existing rows?

Which users need to modify existing rows?

You can classify the users with similar needs by creating a **group** and then adding those users to it. Make a list of the user groups you need to accomodate in the security design. For example, the Music database might have the following groups and authorities:

Group Name	Type of Authority
Managers group	ALL authorities on all tables
Announcers group	INSERT authority on Log table, SELECT on others
Librarians group	ALL authorities on Albums, Titles, and Composers tables

After creating groups, you can grant and revoke authorities to individual users or groups for each table.

Do some tables require general access for most columns but restricted access for some? For these, you can create **views**, which can be made available to all users while the base table is restricted to those with a need to manipulate all columns.

Estimating Table and Index Size

In order to implement your table design, you need to estimate the amount of disk space required for your tables and any indexes you plan to create. Then, when you create DBEFiles to contain the data, specify a size that is big enough to hold the rows you want to store plus some room for growth.

The tutorial in chapter 3 uses file sizes that are more than adequate for the sample database you are creating. Before creating a large database, calculate your space requirements carefully. Complete information about size calculation for tables and indexes is given in the "Physical Design" chapter of the *ALLBASE/SQL Database Administration Guide.*

Designing Applications

Once your table design is complete, you can begin to design your applications in detail. At this point, you may need to modify the table design to improve performance. Keep in mind, however, that later applications may need to use the same data in very different ways. Therefore, table design should remain somewhat independent of application design.

Further Information . . .

Database design is complex. The foregoing discussion oversimplifies many aspects of good design in the interest of getting you started. For a major production system, you need to study your data carefully and plan your tables accordingly.

Here are the titles of some recommended readings for additional help in designing your databases:

- Atre, Shaku. *Database: Structured Techniques for Design, Performance, and Management.* New York: John Wiley & Sons Inc., 1988.
- Date, Chris. "A Practical Approach to Database Design," in *Relational Database: Selected Writings*. Menlo Park, CA: Addison-Wesley Publishing Co., Inc., 1986.
- Loomis, Mary E. S. *The Database Book*. New York: Macmillan Publishing Company, 1987.
- Turk, Thomas A. "Using Data Normalization Techniques for Effective Data Base Design," *Journal of Information Systems Management*, Winter 1985.

Also, refer to the "Logical Design" and "Physical Design" chapters in the *ALLBASE/SQL Database Administration Guide*.

3

Setting Up a Database with ISQL

This chapter is a tutorial on setting up your own database using ISQL, the interactive component of ALLBASE/SQL. The tutorial takes you through the steps you need to follow in creating any ALLBASE/SQL database, large or small.

Here are the steps:

- Running ISQL.
- Creating a DBEnvironment.
- Creating DBEFileSets.
- Creating DBEFiles for table and index data.
- Adding DBEFiles to DBEFileSets.
- Creating tables.
- Entering data into tables.
- Performing queries.
- Creating views.
- Granting authorities.
- Creating an index.
- Examining the system catalog.

The example chosen to illustrate these steps is a database of information about record albums. This database uses the tables described in chapter 2, "Looking at Data." Follow the steps yourself on a system in an empty group (MPE XL) or directory (HP-UX).

Running ISQL

Before you can create a new DBEnvironment, you must run ISQL by simply entering

```
isql Return
```

at your operating system prompt. ALLBASE/SQL then displays the ISQL banner as in Figure 3-1 (some details may be slightly different on your system):

```
     IIIIIIII    SSSSSSSS     QQQQQQQ      LL
        II      SS           QQ     QQ     LL
        II      SS           QQ     QQ     LL
        II        SSSSSS     QQ     QQ     LL
        II              SS   QQ   Q QQ     LL
        II              SS   QQ     QQ     LL
     IIIIIIII    SSSSSSS       QQQQQQ QQ   LLLLLLLLL


                                WED, AUG 08, 1990,  4:14 PM
HP36217-02A.07.00.17        Interactive SQL/9000        HP SQL/HP-UX
(C)COPYRIGHT HEWLETT-PACKARD CO. 1982,1983,1984,1985,1986,1987,1988,
1989. ALL RIGHTS RESERVED.

isql=>
```

Figure 3-1. ISQL Banner

You should make a note of two things before going any further:

- You must *always* use a semicolon and press Return to terminate a command in ISQL.
- You leave ISQL by typing

  ```
  isql=> exit; Return
  ```

  If ISQL asks whether or not you want to commit work, you must reply either Y or N and press Return. A Y makes the work you have done permanent.

Creating a DBEnvironment

If you exited from ISQL, you must run ISQL again. Then, to create a new DBEnvironment, use the START DBE NEW command at the ISQL prompt, as follows:

```
isql=> START DBE 'MUSICDBE' NEW; Return
```

If you forget to use the semicolon, a continuation prompt appears:

```
>
```

Simply type a semicolon and press Return. After a brief interval, you will see the ISQL prompt return. That's it! You have created a new DBEnvironment named MUSICDBE.

Exit from ISQL as explained in the previous section. Use an operating system command to display the files in your current group or directory. Notice three newly created files:

- MUSICDBE.

- DBEFILE0.
- DBELOG1.

DBECon File

The first of these is the DBECon file or DBEnvironment configuration file. This has the same name you assigned to the DBEnvironment in the START DBE command. The DBECon file contains startup parameters for the DBEnvironment. For complete information about startup parameters, refer to the "DBEnvironment Configuration and Security" chapter of the *ALLBASE/SQL Database Administration Guide.*

DBEFile0

DBEFILE0 is a file containing the data for the SYSTEM DBEFileSet, which contains the system catalog. (You'll examine the system catalog later.)

Log File

DBELOG1 is the log file, which records operations that modify the database in any way. The log file is not discussed further here. For more information, refer to the "Backup and Recovery" chapter in the *ALLBASE/SQL Database Administration Guide*.

For more information on configuring a DBEnvironment, refer to the entry for START DBE NEW in the "SQL Commands" chapter of the *ALLBASE/SQL Reference Manual*. This entry describes all the default configuration values used in creating MUSICDBE.

Creating DBEFileSets

Before you can create tables and load data into a database, you need to provide physical file space. Physical files are known as DBEFiles, and they are grouped together in logical groupings called DBEFileSets.

After the last step, you exited from ISQL, so you must run ISQL again and connect to the newly created DBEnvironment MUSICDBE. From the ISQL prompt, issue the following commands:

```
isql=> CONNECT TO 'MUSICDBE'; Return
```

Then use the following command to create a new DBEFileSet—ALBUMFS:

```
isql=> CREATE DBEFILESET AlbumFS; Return
```

Do not forget the semicolon.

Note

ALLBASE/SQL upshifts the logical names of objects like ALBUMFS in the previous example. Thus, even though you enter them in mixed case (as shown above), they will appear in the system catalog as all uppercase. (We'll see an example later.)

MPE XL also upshifts all physical file names. Thus, the names of DBEFiles, log files, and DBECon files (including *MUSICDBE* in the

previous example), all appear as uppercase in directory displays in MPE XL, regardless of whether you entered these names in upper, lower, or mixed case. HP-UX does *not* upshift file names, so they appear in directory displays exactly as you enter them. Rembember that in the CONNECT statement, the name of the DBEnvironment is case-sensitive in HP-UX, but it is not case-sensitive in MPE XL.

Creating DBEFiles for Table and Index Data

Next, you can create DBEFiles of three different types—TABLE, INDEX, and MIXED. In most cases, it is wise to prepare separate DBEFiles for table data and for indexes, so you will create two DBEFiles—AlbumTables and AlbumIndex. (In a later step, you will associate these DBEFiles with the DBEFileSet ALBUMFS.) First, type the following command:

```
isql=> CREATE DBEFILE AlbumTables Return
> WITH PAGES=50, Return
> NAME='ALBUMD1', TYPE=TABLE; Return
```

This command creates a new DBEFile known internally to ALLBASE/SQL as AlbumTables and to the operating system as ALBUMD1. The file contains fifty 4096-byte pages, and is available for storage of table data only.

Now issue the next command:

```
isql=> CREATE DBEFILE AlbumIndex Return
> WITH PAGES=30, Return
> NAME='ALBUMI1', TYPE=INDEX; Return
```

This command creates a new DBEFile known internally to ALLBASE/SQL as AlbumIndex and to the operating system as ALBUMI1. This file contains thirty 4096-byte pages, and is available for storage of indexes only.

Notes

The third DBEFile type—MIXED—can store both tables and indexes. DBEFILE0 is an example of a MIXED DBEFile. DBEFILE0, which belongs to the DBEFileSet known as SYSTEM, was created at the time you issued the START DBE NEW command. You can create new DBEFiles and add them to SYSTEM.

If you do not specify a DBEFileSet when you create a table, it is stored in the SYSTEM DBEFileSet by default. However, since SYSTEM is already in use with system information, it is better to use separate DBEFileSets and DBEFiles for your data. You'll be creating a table in AlbumFS in the next few sections.

Committing Work

Did all three commands complete without any error messages? If so, issue the following command:

```
isql=> COMMIT WORK; Return
```

Committing work is necessary because ALLBASE/SQL processes commands in units known as **transactions**. When you issue the first SQL command in a sequence, a transaction begins, and that transaction continues until you do a COMMIT WORK or ROLLBACK WORK. The use of transactions guarantees the consistency of data within the DBEnvironment.

Note

If you received an error message saying that your transaction was rolled back, you should review what you have done, to see if you can spot the error. When you are ready, go back to the CREATE DBEFILESET command above and try again. If the files ALBUMD1 or ALBUMI1 exist, use an operating system command to erase them from the current working directory or group before attempting to issue the CREATE DBEFILE commands again.

Adding DBEFiles to DBEFileSets

After creating DBEFiles, you must add them to a DBEFileSet before they can be used. In MUSICDBE, two choices are available:

- SYSTEM, already created when you issued the START DBE command.
- ALBUMFS, which you created in an earlier step.

Use the following commands to add your DBEFiles to DBEFileSet ALBUMFS:

```
isql=> ADD DBEFILE AlbumTables Return
> TO DBEFILESET ALBUMFS; Return
isql=> ADD DBEFILE AlbumIndex Return
> TO DBEFILESET ALBUMFS; Return
isql=> COMMIT WORK; Return
```

Don't forget to COMMIT WORK!

Creating Tables

Now get ready to create some tables for MUSICDBE. Before doing this step, you need to analyze the data that is to be stored. In the next paragraphs, assume that, using the suggestions in chapter 2, you have already arrived at the following table design:

Albums Table

AlbumCode	INTEGER NOT NULL
AlbumTitle	VARCHAR(40)
Medium	CHAR(2)
AlbumCost	DECIMAL(6,2)
RecordingCo	CHAR(10) NOT NULL
DateRecorded	DATE
MfgCode	VARCHAR(40)
Comment	VARCHAR(80)

Titles Table

AlbumCode	INTEGER NOT NULL
Selection	VARCHAR(40) NOT NULL
Timing	INTERVAL
Composer	VARCHAR(40)
Performers	VARCHAR(40)
Comment	VARCHAR(80)

Now, you need to create each of these tables.

Creating the Albums Table

Use the following command to create the Albums table:

```
isql=> CREATE PUBLIC TABLE Albums Return
> (AlbumCode INTEGER NOT NULL, Return
> AlbumTitle VARCHAR(40) NOT NULL, Return
> Medium CHAR(2), Return
> AlbumCost DECIMAL(6,2), Return
> RecordingCo CHAR(10) NOT NULL, Return
> DateRecorded DATE, Return
> MfgCode VARCHAR(40), Return
> Comment VARCHAR(80)) Return
> IN ALBUMFS; Return
```

Did the command complete without errors? If not, did you do the following:

- Use a semicolon at the end?
- Include a final close parenthesis?
- Use valid data types and sizes?

Use the ISQL REDO command to examine and correct your command, then issue it again. (For help with REDO, type HELP REDO at the isql=> prompt.

When the command completes without errors, use the COMMIT WORK command to make it permanent.

Note

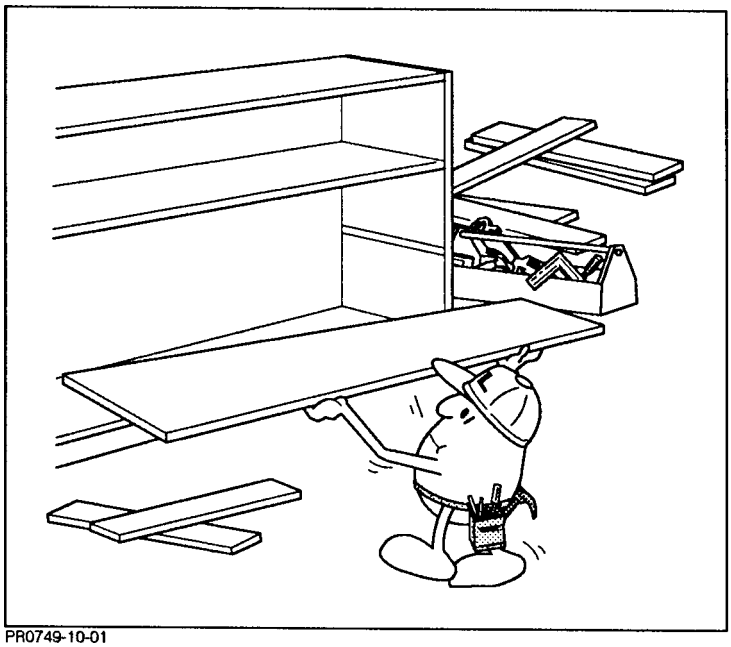
You created the Albums table as a PUBLIC table, which means that other users of the DBEnvironment need no special authorization to access the table.

Creating the Titles Table

Now, create the second table:

```
isql=> CREATE PUBLIC TABLE Titles Return
> (AlbumCode INTEGER NOT NULL, Return
> Selection VARCHAR(40) NOT NULL, Return
> Composer CHAR(40), Return
> Timing INTERVAL, Return
> Performers VARCHAR(40), Return
> Comment VARCHAR(80)) Return
> IN ALBUMFS; Return
```

If the command completes without errors, use the COMMIT WORK command.



PR0749-10-01

Entering Data into Tables

Once you have created tables, you can get data into them in several ways. Try the following two methods:

- The SQL INSERT command.
- The ISQL LOAD command.

Entering Data with the SQL INSERT Command

Use the SQL INSERT command to add rows to the tables you have created. Try the following entry for the Albums table:

```
isql=> INSERT INTO Albums Return
> VALUES (2001, Return
>  'Serenades from the 17th Century', Return
>  'ca', 30.82, Return
> 'philips', '1988-12-18', Return
> '3456-AB-0998LS', Return
> 'Authentic original instruments'); Return
```

If the command does not complete successfully, check all your punctuation carefully, and try again. When finished, COMMIT WORK.

Next, use the INSERT command to add the following row to the Titles table (you need to build the INSERT command yourself):

```
AlbumCode: 2001
Selection: 'La Bella Musica'
Composer: 'Palestrina'
Timing: '0 00:21:12.000'
Performers: 'Ancient Music Group'
```

```
Comment: 'Lute improvisations'
```

Note that INSERT is an SQL command that processes a single row of data at a time. If you want to insert many rows at a time, use an application program of your own design, or else try the ISQL LOAD command, explained below.

Entering Data with the ISQL LOAD Command

Use the ISQL LOAD command to insert data from an ordinary file into your tables. Two kinds of LOAD operation are possible:

- LOAD INTERNAL.
- LOAD EXTERNAL.

The next sections show an example of each.

Note

The sample external and internal files described in the next few paragraphs are available on MPE XL 3.0 or later systems and on HP-UX 8.0 or later systems. If you are using an earlier version of ALLBASE/SQL, you should skip ahead to the section entitled "Performing Queries."

LOADing from an INTERNAL File

You use the LOAD command with the INTERNAL option to load data from a file in INTERNAL format previously created by ISQL's UNLOAD command. Your system contains an INTERNAL file with data for the Titles table. In MPE XL, it is called TITLE.SAMPLEDB.SYS; in HP-UX, it is called /usr/lib/allbase/hpsql/sampledb/title. From ISQL, issue the apppriate command for your system:

On MPE XL:

```
isql=> LOAD FROM INTERNAL Return
> TITLE.SAMPLEDB.SYS Return
>  TO Titles; Return
```

On HP-UX:

```
isql=> LOAD FROM INTERNAL Return
> /usr/lib/allbase/hpsql/sampledb/Title Return
> TO Titles; Return
```

As loading progresses, messages tell you how many rows have been processed.

Note

If you are loading a large file, be sure to set ISQL's AUTOCOMMIT function to ON. For information, type HELP SET AUTOCOMMIT at the ISQL prompt. For the present examples, AUTOCOMMIT is not needed.

LOADing from an EXTERNAL File

Use the LOAD command with the EXTERNAL option to load data from plain ASCII files into a table. You must enter the names of the columns in the table you are loading into and the starting location in the file where each data item starts, together with the data item's length. If the column permits null values, the data file must contain null indicator characters for any entry that is null. In the following example, the question mark (?) is used as a null indicator.

From ISQL, issue one of the following commands, as appropriate for your system. Be sure to type exactly.

On MPE XL:

```
isql=> LOAD FROM EXTERNAL Return
>  ALBUM.SAMPLEDB.SYS to Albums Return
>  AlbumCode      1  4 Return
>  AlbumTitle    13 40 Return
>  Medium        53  2   ? Return
>  AlbumCost     55  6   ? Return
>  RecordingCo   61 10 Return
>  DateRecorded  71 10   ? Return
>  MfgCode       89 40   ? Return
>  Comment      137 80   ? Return
>  END; Return
```

On HP-UX:

```
isql=> LOAD FROM EXTERNAL Return
>  /usr/lib/allbase/hpsql/sampledb/Album Return
>  to Albums Return
>  AlbumCode      1  4 Return
>  AlbumTitle    13 40 Return
>  Medium        53  2   ? Return
>  AlbumCost     55  6   ? Return
>  RecordingCo   61 10 Return
>  DateRecorded  71 10   ? Return
>  MfgCode       89 40   ? Return
>  Comment      137 80   ? Return
>  END; Return
```

After you have entered the column descriptions, ISQL prompts you as follows:

```
Load depending on value in input record (Y/N)>
```

Reply N to load all the values in the file. When the command completes, issue a COMMIT WORK command:

```
isql=> commit work; Return
```

Note that the starting columns for each field of data are determined by the actual position of the data in the file itself. Thus, using EXTERNAL files, it is possible to load selected parts of each record.

For complete information about loading tables from INTERNAL and EXTERNAL files, refer to the *ALLBASE/ISQL Reference Manual.*

Performing Queries

After loading your tables, you are ready to perform some queries to see the result of your efforts at data definition and data entry. Use the SELECT command to display the information you need.

The simplest form of SELECT uses the asterisk (*) to indicate that you want to retrieve *all* the rows and columns in the table:

```
isql=> SELECT * FROM Titles; Return
```

This command retrieves all columns for all rows in the Titles table.

You can add a **predicate** (a WHERE clause) to narrow the range of rows selected to a specific subgroup. In the following example, the predicate evaluates a column in the table (Composer) against a constant value ('Palestrina'):

```
isql=> SELECT * FROM Titles Return
> WHERE Composer = 'Palestrina'; Return
```

This retrieves all rows in the Titles table whose composer is Palestrina.

You can use a **column list** to narrow the range of columns selected to a specific group:

```
isql=> SELECT AlbumTitle, Return
> AlbumCost from Albums Return
> WHERE Medium = 'cd'; Return
```

A query that retrieves information from more than one table is known as a join. In a two-table join, at least one predicate evaluates a column in the first table against a column in the second. For example, the following join query displays all album titles and selections that have the same album code, drawing on the rows in both the Albums and the Titles tables:

```
isql=> SELECT AlbumTitle, Selection Return
> FROM Albums, Titles WHERE Return
> Albums.AlbumCode = Titles.AlbumCode; Return
```

Notice that the AlbumCode column has to be qualified by its table name because it appears in both tables. For additional information about queries, refer to the "SQL Queries" chapter of the *ALLBASE/SQL Reference Manual*.

Creating Views

When you need to perform a query frequently, you can define a **view** that incorporates the column list and the predicate; then you can select from the view with a simpler command.

As an example, create a view of the Albums and Titles tables that includes *all* selections:

```
isql=> CREATE VIEW Selections AS Return
> SELECT AlbumTitle, Selection, Return
> Composer FROM Albums, Titles Return
> WHERE Albums.AlbumCode = Return
> Titles.AlbumCode; Return
```

Now use the following simple SELECT statement to display all selections by Palestrina:

```
SELECT * FROM Selections Return
> WHERE Composer = 'Palestrina'; Return
```

Views are useful when you want to define specific subsets of data that are frequently used or when you want to restrict access to particular subsets. You can create views and grant access to them, then revoke access to the underlying **base tables**.

Did you remember to COMMIT WORK? This makes the view definition permanent in the DBEnvironment.

For more information about views, see the CREATE VIEW command in the "SQL Queries" chapter of the *ALLBASE/SQL Reference Manual*.

Granting Authorities

Because you created MUSICDBE, you have DBA (database administrator authority), which lets you grant authorities to other users. In a simple authorization scheme, you first grant CONNECT authorization to permit access to the DBEnvironment itself. Then you grant table authorities to specific users or groups of users. You can also include the special user PUBLIC, which includes anyone who has the authorization to CONNECT to the DBEnvironment. Use the following commands to create an authority scheme for two groups of users, announcers and librarians, while excluding all others (PUBLIC).

```
isql=> CREATE GROUP Librarians; Return
isql=> ADD Ann, Peter TO Return
> GROUP Librarians; Return
isql=> CREATE GROUP Announcers; Return
isql=> ADD Fred, Julia TO Return
> GROUP Announcers; Return
isql=> GRANT CONNECT TO Return
> Announcers, Librarians; Return
isql=> GRANT ALL ON Albums Return
> TO Librarians; Return
isql=> GRANT ALL ON Titles Return
> TO Librarians; Return
isql=> GRANT SELECT ON Albums Return
> TO Announcers; Return
isql=> GRANT SELECT ON Titles Return
> TO Announcers; Return
isql=> REVOKE ALL ON Albums Return
> FROM PUBLIC; Return
isql=> REVOKE ALL ON Titles Return
> FROM PUBLIC; Return
```

Because you created these tables as PUBLIC (shareable by everyone) in an earlier step, you need to remove authorities from PUBLIC. This is the normal procedure for tables that will have restricted but still multi-user access in the DBEnvironment.

You can also use views to provide restricted access to portions of tables. After creating the view, you can grant access on it to a specific user or group:

```
isql=> GRANT SELECT ON Selections TO PUBLIC; Return
```

Creating an Index

You can speed up access to data by providing indexes on specific columns in your tables. Assuming your tables are large enough, an **index scan** will arrive at a specific row more quickly than a **serial scan**. When an index is used, ALLBASE/SQL looks for an entry in the index first, then goes to the row. When a serial scan is used, ALLBASE/SQL reads from the beginning of the table until the desired row is reached. Naturally, the use of the index is faster if you only need a small subset of rows.

You can also use an index to guarantee the uniqueness of specific column values. In the Albums table, for example, the AlbumCode column should be unique; in the Titles table, it should not be unique, because a single album may contain several selections.

Create a **unique index** on the AlbumCode column of the Albums table with the following command:

```
isql=> CREATE UNIQUE INDEX AlbCodeIndex Return
> ON ALBUMS (AlbumCode); Return
```

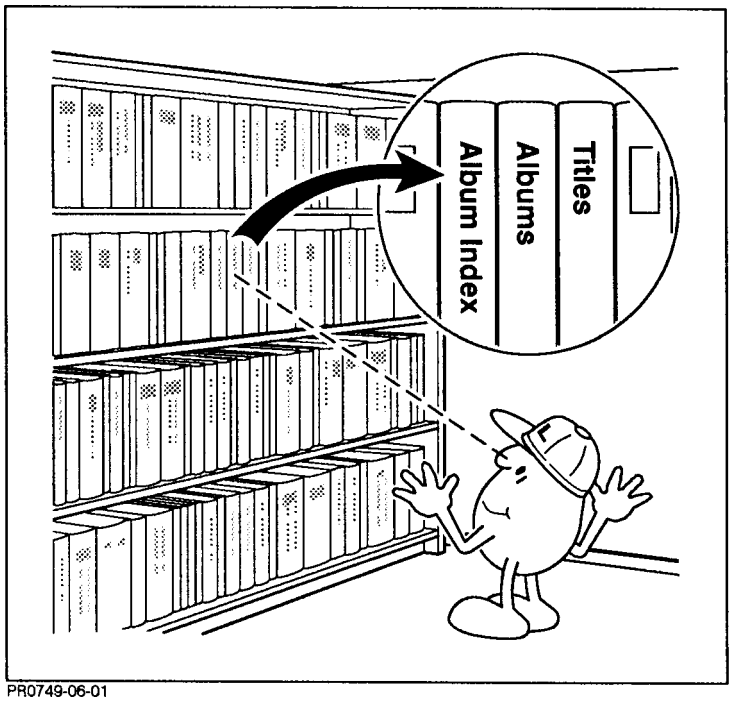
Use the following command to create a non-unique index on the AlbumCode column of the Titles table:

```
isql=> CREATE INDEX TitleCodeIndex Return
> ON TITLES (AlbumCode); Return
```

Location of Tables and Indexes

Each index is created in the same DBEFileSet as the table it is indexing. These two indexes are created in the DBEFileSet ALBUMFS, and they are physically located in the INDEX DBEFile created for that DBEFileSet in an earlier step.

As shown in the illustration, indexes and tables always appear in the same DBEFileSet (shelf area). They may be in different DBEFiles, however.



PR0749-06-01

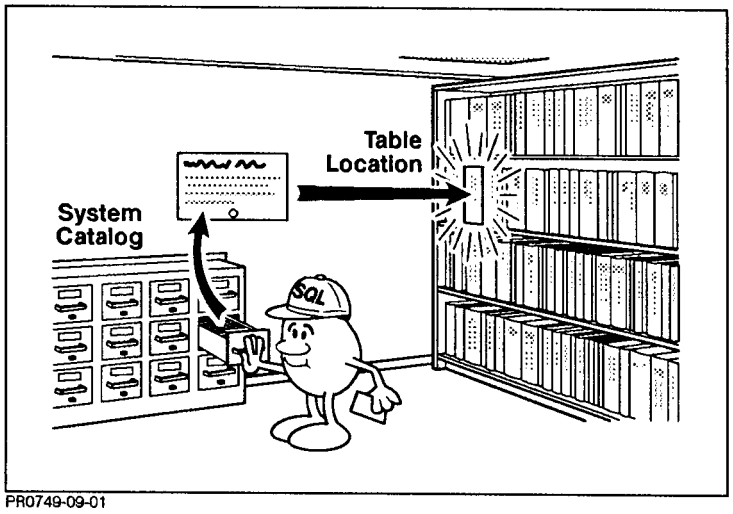
Note

When you issue a query, you do not tell ALLBASE/SQL to use an index. Instead, the SQLCore **optimizer** decides when the use of an existing index is the best way to access a specific set of data.

For more information about indexes, refer to the discussion of "Providing Data Access Paths" in the chapter "Using ALLBASE/SQL" of the *ALLBASE/SQL Reference Manual*. Also, see the CREATE INDEX command in the "SQL Commands" chapter of the *ALLBASE/SQL Reference Manual*.

Examining the System Catalog

Whenever you create objects in ALLBASE/SQL, their characteristics are stored in the **system catalog**, which is a special system-created database. The system catalog is like a listing of the contents of a DBEnvironment. (See the illustration below.)



You can look at the system catalog by performing queries on **system tables**. Issue the following query to see entries for the tables and views you have just created:

```
isql=> SELECT * FROM SYSTEM.TABLE Return
> WHERE OWNER = USER; Return
```

The next figure shows the first display you see.

```
select * from system.table where owner = USER;
------------------+-------------------+-------------------+------+-----
NAME              |OWNER              |DBEFILESET         |TYPE  |RTYPE
------------------+-------------------+-------------------+------+-----
ALBUMS            |PETERW             |ALBUMFS            |     0|
SELECTIONS        |PETERW             |SYSTEM             |     1|
TITLES            |PETERW             |ALBUMFS            |     0|









------------------------------------------------------------------------
Number of rows selected is 3
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e

isql=>
```

Figure 3-2. System.Table Display

Enter "r" to scroll right to see all the columns. A complete description of each column appears in chapter 9 of the *ALLBASE/SQL Database Administration Guide*. For now, notice the value in the TYPE column. A zero (0) indicates a table, a one (1) indicates a view.

Updating Statistics in the System Catalog

Use the UPDATE STATISTICS command for each table as follows:

```
isql=> UPDATE STATISTICS FOR TABLE ALBUMS;
isql=> UPDATE STATISTICS FOR TABLE TITLES;
```

This ensures that the current number of rows in each table and other statistical information are updated in the system catalog. The statistical information is used by SQLCore when it decides whether to use an index or not. Statistics also provide information about ongoing processes within the DBEnvironment.

In Review . . .

In this tutorial chapter, you created a default DBEnvironment structure consisting of a DBECon file, a log, and a DBEFILE0 associated with DBEFileSet SYSTEM to contain system catalog data. You created the ALBUMFS DBEFileSet, and two DBEFiles associated with it—AlbumData, for the Albums and Titles table data, and AlbumIndex, for the Albums and Titles indexes. You also created and loaded the tables themselves, created groups and granted authorities, and defined a unique and a non-unique index. You have also done several queries and examined the system catalog.

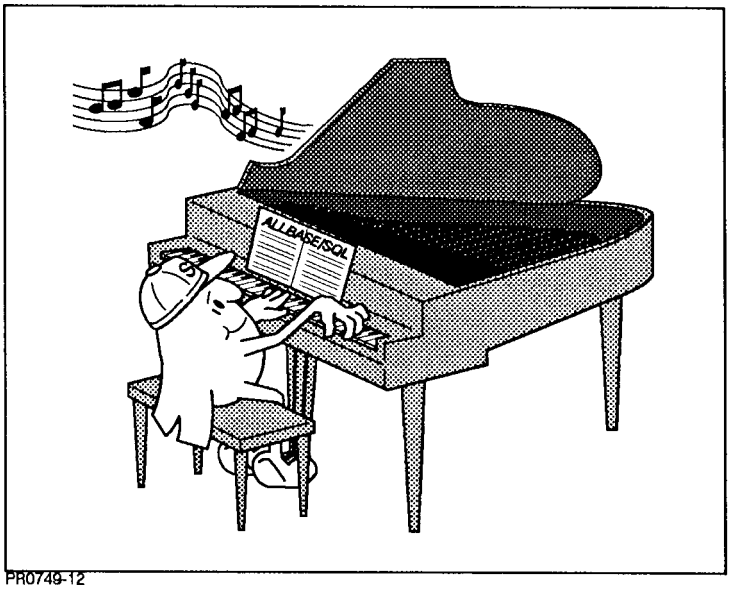
As a further exercise, create the Composers and Log tables from the descriptions in chapter 2. Consider whether to create indexes on them. Add some rows to them, using values consistent with the entries in the Albums and Titles tables. Then perform some queries. What sort of security structure would be appropriate?

There's plenty more! But you're off to a good start. As you continue with your own DBEnvironments, use the examples in the *ALLBASE/ISQL Reference Manual* and the *ALLBASE/SQL Reference Manual* as models. You can also refer to the chapter in this guide entitled "Practice with ALLBASE/SQL Using PartsDBE" for information about a comprehensive sample DBEnvironment that is supplied as a part of ALLBASE/SQL.

4

Practice with ALLBASE/SQL Using PartsDBE

The ALLBASE/SQL software includes a sample DBEnvironment known as PartsDBE and a set of sample application programs which illustrate much of the functionality of ALLBASE/SQL. Most of the examples in the ALLBASE/SQL documentation use PartsDBE. This chapter shows you how to get practice using the components of ALLBASE/SQL by creating a version of PartsDBE which you can use to try out the examples on your own system.



PR0749-12

Here are the topics covered:

- Setting up PartsDBE.
- Examining PartsDBE.
- Using the preprocessors.
- Examining startup parameters with SQLUtil.
- Creating a schema file with SQLGEN.
- Purging PartsDBE.

Setting up PartsDBE

Before beginning, change into the group and account or directory where you want to create PartsDBE. Use an empty group or directory if possible. Then choose one of the following two methods for setting up PartsDBE:

- Using SQLSetup.
- Using setup scripts.

SQLSetup is a sample database setup tool which simplifies the process of installing PartsDBE in your work space. It is available on MPE XL 3.0 and later systems, and on HP-UX 7.08 and later systems. If you are using a system prior to HP-UX 7.08 or MPE XL 3.0, use the setup procedure described under "Using Setup Scripts."

Using SQLSetup

Run SQLSetup by issuing the proper command for your system:

```
HP-UX:
$ csh /usr/lib/allbase/hpsql/sqlsetup Return

MPE XL:
: SQLSETUP.SAMPLEDB.SYS Return
```

A menu like the one in Figure 3-1 appears on your screen (some details may differ on your system).

```
    Options for Setting Up ALLBASE/SQL Sample DBEnvironments

================================================================

Choose one:

1. Create PartsDBE without sample programs
2. Create PartsDBE, copy, preprocess and compile sample programs
3. Copy, preprocess and compile sample programs only
4. Generate a schema for PartsDBE
5. Display schema for PartsDBE
6. Purge PartsDBE and sample programs
7. Help
0. Exit

================================================================

Enter your choice=>
```

Figure 4-1. SQLSetup Menu

From this menu, you select an option to create a copy of PartsDBE in your directory (HP-UX) or group and account (MPE XL). Before choosing an option, examine each line on the menu. The first option simply creates a copy of PartsDBE. The second option, in addition to creating PartsDBE, copies a set of application programs into the current directory or group, then preprocesses and compiles them. (This is time-consuming.)

Option 3 creates just the sample program set. Option 4 creates a **schema** with SQLGEN. Option 5 displays the schema once it has been created. Option 6 lets you purge the sample DBEnvironment and programs.

Choose the Help option to see more information about SQLSetup, or choose 0 to exit.

Creating PartsDBE

Choose option 1 from the SQLSetup menu. This option runs a set of ISQL command files that create the DBEnvironment, define all its tables, views, indexes and security structure, and then load it with data.

As the system creates PartsDBE, you see several messages displayed. At the end of the creation process, you see the following message:

```
Creation and Loading of PartsDBE is now complete!
```

When you return to the menu, choose 0 to exit.

Using Setup Scripts

The following paragraphs describe an alternate method for setting up PartsDBE using setup scripts that are available on all systems.

HP-UX Systems

Use the following command to set up PartsDBE:

```
$ /usr/lib/allbase/hpsql/setup 2 Return
```

You will see a display of messages showing the progress of the setup script. A listing of *setup* appears in Appendix C of the *ALLBASE/SQL Reference Manual*.

MPE XL Systems

Use the command file CREASQL to stream a job that sets up PartsDBE. First, copy the CREASQL stream file to your group and account with the following command:

```
: FCOPY FROM=CREASQL.SAMPLEDB.SYS;TO=CREASQL;NEW Return
```

Using an editor, modify CREASQL to include your password(s), user name, account name, and group name. Lines that need to be modified are shown in inverse display. Keep the edited file, then type the following command to create and load PartsDBE and copy the sample programs into your group and account:

```
: STREAM CREASQL Return
```

You will see messages showing the progress of the setup script. A listing of CREASQL appears in Appendix C of the *ALLBASE/SQL Reference Manual*.

Looking at the Files Created for PartsDBE

HP-UX Systems

Use the *ls -l* command to list the files in the current directory. You should see the following (owner and group entries will be for your system, and permissions will be those of your directories):

```
drwxrwxr-x   3 peter    dbusers     1024 Dec 27 11:23 hpsql/
```

The setup script created this directory for you. Next change into the hpsql directory, and do another listing:

```
-rw-rw-rw-   1 peter    dbusers    19297 Dec 27 11:25 isqlout
drwxrwxrwx   2 peter    dbusers     1024 Dec 27 11:24 sampledb/
```

The setup script also created the sampledb directory. The file isqlout contains the messages generated when PartsDBE was created. Use the *more* command to examine isqlout.

Finally, change into the sampledb directory, then examine the file listing. You see some files with your user name as owner and others

with hpdb as owner. To see the database files alone, issue the following command:

```
$ ls -l | grep hpdb Return
```

You should see the following list of files:

```
-rw-------   1 hpdb     dbusers   204800 Dec 27 11:25 OrderDF1
-rw-------   1 hpdb     dbusers   204800 Dec 27 11:25 OrderXF1
-rw-------   1 hpdb     dbusers      512 Dec 27 11:25 PartsDBE
-rw-------   1 hpdb     dbusers   614400 Dec 27 11:25 PartsF0
-rw-------   1 hpdb     dbusers   131072 Dec 27 11:25 PartsLG1
-rw-------   1 hpdb     dbusers   131072 Dec 27 11:25 PartsLG2
-rw-------   1 hpdb     dbusers   204800 Dec 27 11:25 PurchDF1
-rw-------   1 hpdb     dbusers   204800 Dec 27 11:25 PurchXF1
-rw-------   1 hpdb     dbusers   204800 Dec 27 11:25 RecDF1
-rw-------   1 hpdb     dbusers   204800 Dec 27 11:25 WarehDF1
-rw-------   1 hpdb     dbusers   204800 Dec 27 11:25 WarehXF1
```

These files, all owned by hpdb, are the files for the PartsDBE DBEnvironment. The other files in the directory are command files and load files containing data that was loaded into PartsDBE.

For security reasons, database files are owned by hpdb, and the sampledb directory also belongs to hpdb. This means that you cannot remove the database files with the rm command unless you are the superuser. (You can use SQLUtil, however, as shown later in this chapter.)

The programs directory is for use when you decide to copy, preprocess and compile sample application programs.

MPE XL Systems

List the files in the current group and account. You should see the following:

```
ORDERDF1  PRIV   2048W  FB          50          50   1      800  1 26
ORDERXF1  PRIV   2048W  FB          50          50   1      800  1 26
PARTSDBE  PRIV    256W  FB        1161        1161   1     1920 30  *
PARTSF0   PRIV   2048W  FB         150         150   1     2400  1 31
PARTSLG1  PRIV    256W  FB         256         256   1      512  1 29
PARTSLG2  PRIV    256W  FB         256         256   1      512  1 29
PURCHDF1  PRIV   2048W  FB          50          50   1      800  1 26
PURCHXF1  PRIV   2048W  FB          50          50   1      800  1 26
RECDF1    PRIV   2048W  FB          50          50   1      800  1 26
WAREHDF1  PRIV   2048W  FB          50          50   1      800  1 26
WAREHXF1  PRIV   2048W  FB          50          50   1      800  1 26
```

These are all PRIV files, which means that you cannot purge them without special system authority. (However, you can use SQLUtil, as shown later in this chapter.)

Additional files used for loading the sample database tables are found in SAMPLEDB.SYS.

Examining PartsDBE

In this section, you will examine the objects that were created *within* PartsDBE—tables, views, indexes, and authority structure. Information about all these objects is in the system catalog, which is automatically created by ALLBASE/SQL as the DBEnvironment is configured.

Run ISQL, then CONNECT to PartsDBE. (If you are using HP-UX, first change back to the directory from which you ran the script to create PartsDBE. You must have write permission in the directory from which you CONNECT.) Use one of the following CONNECT commands:

For HP-UX:

```
isql=> CONNECT TO 'hpsql/sampledb/PartsDBE'; Return
```

MPE XL:

```
isql=> CONNECT TO 'PartsDBE'; Return
```

Now examine the system catalog by creating queries on the system views.

Examining the Tables and Views

Use the following query exactly as shown to look at all the tables and views created by the setup script:

```
isql=> SELECT NAME, OWNER, Return
> DBEFILESET, TYPE Return
> FROM SYSTEM.TABLE Return
> WHERE OWNER <> 'SYSTEM'; Return
```

The result table is shown below.

```
select name, owner, dbefileset, type from system.table where owner  'SYST
------------------+------------------+------------------+------
NAME              |OWNER             |DBEFILESET        |TYPE
------------------+------------------+------------------+------
SUPPLYBATCHES     |MANUFDB           |WAREHFS           |     0
TESTDATA          |MANUFDB           |WAREHFS           |     0
PARTS             |PURCHDB           |WAREHFS           |     0
INVENTORY         |PURCHDB           |WAREHFS           |     0
SUPPLYPRICE       |PURCHDB           |PURCHFS           |     0
VENDORS           |PURCHDB           |PURCHFS           |     0
ORDERS            |PURCHDB           |ORDERFS           |     0
ORDERITEMS        |PURCHDB           |ORDERFS           |     0
PARTINFO          |PURCHDB           |SYSTEM            |     1
VENDORSTATISTICS  |PURCHDB           |SYSTEM            |     1
MEMBERS           |RECDB             |RECFS             |     0
CLUBS             |RECDB             |RECFS             |     0
EVENTS            |RECDB             |RECFS             |     0
------------------+------------------------------------------------------
Number of rows selected is 13
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
```

Figure 4-2. Information on Tables and Views

Each table is identified by the NAME column. The OWNER column specifies the database to which the table belongs. If a table does not belong to you (that is, if you are not the database owner), you must prefix the table name with its owner name whenever you refer to it.

The DBEFILESET column contains the name of the DBEFileSet an entry has been associated with, and the TYPE column indicates whether the entry is a table or a view. Entries with type 0 are tables, and entries with type 1 are views. Note that all views are automatically associated with the SYSTEM DBEFileSet.

View Definitions

You can see the view definitions by issuing the following query exactly as shown:

```
isql=> SELECT VIEWNAME, SELECTSTRING Return
> FROM SYSTEM.VIEWDEF WHERE Return
> OWNER = 'PURCHDB'; Return
```

The query result is shown in the next figure.

```
select viewname,selectstring from system.viewdef where owner = 'PURCHDB';
------------------+-------------------------------------------------------
VIEWNAME           |SELECTSTRING
------------------+-------------------------------------------------------
PARTINFO           | SELECT PurchDB.SupplyPrice.PartNumber, PurchDB.Parts.
PARTINFO           |PurchDB.SupplyPrice.VendorNumber, PurchDB.Vendors.Vend
PARTINFO           |PurchDB.Supplyprice.VendPartNumber,
PARTINFO           |PurchDB.SupplyPrice.UnitPrice, PurchDB.SupplyPrice.Dis
PARTINFO           |FROM PurchDB.Parts, PurchDB.SupplyPrice, PurchDB.Vendo
PARTINFO           |PurchDB.SupplyPrice.PartNumber = PurchDB.Parts.PartNum
PARTINFO           |PurchDB.SupplyPrice.VendorNumber = PurchDB.Vendors.Ven
VENDORSTATISTICS   | SELECT PurchDB.Vendors.VendorNumber, PurchDB.Vendors.
VENDORSTATISTICS   |, OrderDate, OrderQty, OrderQty * PurchasePrice FROM
VENDORSTATISTICS   |PurchDB.Vendors, PurchDB.Orders, PurchDB.OrderItems WH
VENDORSTATISTICS   |PurchDB.Vendors.VendorNumber = PurchDB.Orders.VendorNu
VENDORSTATISTICS   |PurchDB.Orders.OrderNumber = PurchDB.OrderItems.OrderN
------------------+-------------------------------------------------------
Number of rows selected is 12
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
```

Figure 4-3. View Definitions in the System Catalog

Scroll to the right to examine the complete select string for each view definition.

Using the INFO Command

You can see individual table descriptions by using the INFO command, which returns the column definition of a table. Use the following command for the Vendors table:

```
isql=> INFO PURCHDB.VENDORS; Return
```

The output from this ISQL command is shown below:

```
isql=> info purchdb.vendors;

Column Name          Data Type (length)     Nulls Allowed   Language
-------------------------------------------------------------------
VENDORNUMBER         Integer                NO
VENDORNAME           Char (   30)           NO              n-computer
CONTACTNAME          Char (   30)           YES             n-computer
PHONENUMBER          Char (   15)           YES             n-computer
VENDORSTREET         Char (   30)           NO              n-computer
VENDORCITY           Char (   20)           NO              n-computer
VENDORSTATE          Char (    2)           NO              n-computer
VENDORZIPCODE        Char (   10)           NO              n-computer
VENDORREMARKS        VarChar (   60)        YES             n-computer
```

Figure 4-4. Output of the INFO Command

The Column Name column lists the names of all the columns in the table. The Data Type column shows the specific data type for each column and its size (in parentheses). The third column, Nulls Allowed, indicates whether or not NULL values are permitted in the column, and the Language column indicates which language is applicable for the column if it is a character type.

Examining Indexes

The following query shows the indexes on tables in PartsDBE:

```
isql=> SELECT INDEXNAME, TABLENAME, Return
> UNIQUE, CLUSTER FROM SYSTEM.INDEX; Return
```

The query result is shown in the next figure:

```
select indexname,tablename,unique,cluster from system.index;
------------------+------------------+------+-------
INDEXNAME          |TABLENAME         |UNIQUE|CLUSTER
------------------+------------------+------+-------
PARTNUMINDEX       |PARTS             |     1|      0
PARTTONUMINDEX     |SUPPLYPRICE       |     0|      1
PARTTOVENDINDEX    |SUPPLYPRICE       |     0|      0
VENDPARTINDEX      |SUPPLYPRICE       |     1|      0
VENDORNUMINDEX     |VENDORS           |     1|      0
ORDERNUMINDEX      |ORDERS            |     1|      1
ORDERVENDINDEX     |ORDERS            |     0|      0
ORDERITEMINDEX     |ORDERITEMS        |     0|      1
INVPARTNUMINDEX    |INVENTORY         |     1|      0
------------------------------------------------------------------------
Number of rows selected is 9
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
```

Figure 4-5. System Catalog Information on Indexes

The UNIQUE and CLUSTER columns show what kind of index was created: PartNumIndex is a unique index; PartToNumIndex is a clustering index; OrderNumIndex is both unique and clustering; and OrderVendIndex is neither unique nor clustering.

Examining the Authority Structure

An authority structure consists of many elements. Some of these elements are shown below:

- Group definitions.
- Table authorizations for select, insert, update, and delete operations on tables.
- **Column authorizations** for permission to update specific columns.

Groups

Use the following query to examine the **authorization groups** in PartsDBE and their members:

```
isql=> SELECT * FROM SYSTEM.GROUP; (Return)
```

The query result is shown below:

```
select * from system.group;
------------------+------------------+------------------+----------
USERID            |GROUPID           |OWNER             |NMEMBERS
------------------+------------------+------------------+----------
PURCHMANAGERS     |PURCHMANAGERS     |PETER             |         3
MARGY             |PURCHMANAGERS     |PETER             |         0
RON               |PURCHMANAGERS     |PETER             |         0
SHARON            |PURCHMANAGERS     |PETER             |         0
PURCHDBMAINT      |PURCHDBMAINT      |PETER             |         3
ANNIE             |PURCHDBMAINT      |PETER             |         0
DOUG              |PURCHDBMAINT      |PETER             |         0
DAVID             |PURCHDBMAINT      |PETER             |         0
PURCHASING        |PURCHASING        |PETER             |         5
AJ                |PURCHASING        |PETER             |         0
JORGE             |PURCHASING        |PETER             |         0
RAGAA             |PURCHASING        |PETER             |         0
GREG              |PURCHASING        |PETER             |         0
KAREN             |PURCHASING        |PETER             |         0
RECEIVING         |RECEIVING         |PETER             |         3
AL                |RECEIVING         |PETER             |         0
------------------+------------------------------------------------------
First 16 rows have been selected.
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
```

Figure 4-6. Groups in the System Catalog

For each group, the members are listed. Note that the group and the member names are listed in the USERID column, and the number of members appears in each row where the group name appears as a USERID. The OWNER column shows the owner of the authorization group.

Table Authorities

Use the following query exactly as shown to examine the authorizations on the PurchDB.Inventory table:

```
isql=> SELECT USERID, SELECT, INSERT, Return
> UPDATE, DELETE, ALTER, INDEX Return
> FROM SYSTEM.TABAUTH WHERE Return
> NAME = 'INVENTORY'; Return
```

The query result is shown below:

```
select userid, select, insert, update,delete, alter, index from system.taba
-------------------+------+------+------+------+-----+-----
USERID              |SELECT|INSERT|UPDATE|DELETE|ALTER|INDEX
-------------------+------+------+------+------+-----+-----
PURCHMANAGERS       |Y     |N     |N     |N     |N    |N
PURCHDBMAINT        |Y     |Y     |Y     |Y     |Y    |Y
PURCHASING          |Y     |Y     |Y     |Y     |N    |N
WAREHOUSE           |Y     |Y     |Y     |Y     |N    |N
KELLY               |N     |N     |C     |N     |N    |N
PETER               |N     |N     |C     |N     |N    |N
DBEUSERS            |Y     |Y     |Y     |Y     |N    |N
-------------------------------------------------------------------
Number of rows selected is 7
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
```

Figure 4-7. Table Authorities in the System Catalog

Each row contains a USERID, which is the name of a user or group, and an entry for each type of authority. A Y in a column indicates that the USERID has that authority, an N indicates the USERID does not have that authority.

Column Authorizations

A special kind of authorization is the permission to update specific columns in a table. These permissions are shown in the SYSTEM.COLAUTH view in the system catalog.

Use the following query exactly as shown to display the column authorizations defined for the PurchDB.Inventory table:

```
isql=> SELECT USERID, TABLENAME, Return
> OWNER, COLNAME FROM Return
> SYSTEM.COLAUTH WHERE Return
> TABLENAME = 'INVENTORY'; Return
```

The query result is shown in the next figure:

```
select userid, tablename, owner, colname from system.colauth where tablenam
------------------+------------------+------------------+-----------
USERID            |TABLENAME         |OWNER             |COLNAME
------------------+------------------+------------------+-----------
KELLY             |INVENTORY         |PURCHDB           |BINNUMBER
KELLY             |INVENTORY         |PURCHDB           |QTYONHAND
KELLY             |INVENTORY         |PURCHDB           |LASTCOUNTDAT
PETER             |INVENTORY         |PURCHDB           |BINNUMBER
PETER             |INVENTORY         |PURCHDB           |QTYONHAND
PETER             |INVENTORY         |PURCHDB           |LASTCOUNTDAT
------------------+-------------------------------------------------------
Number of rows selected is 6
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
```

Figure 4-8. Column Authorities in the System Catalog

Using the Preprocessors

For large-scale database access or for batch operation, ALLBASE/SQL includes a set of **preprocessors**, which let you embed SQL commands into the source code for your own applications. Before compiling your program, you use the preprocessor to prepare the program for runtime database accesses. Separate preprocessors are available for C, COBOL, FORTRAN, and Pascal.

Sample Application Programs

In addition to the files for creating PartsDBE, your system includes a set of sample application programs that use embedded SQL. You can use option 3 from the SQLSetup menu to preprocess and compile all of the examples for a given programming language, or you can examine a single program at a time as shown in the next section. The sample application programs for MPE XL are located in SAMPLEDB.SYS, and in HP-UX, they are found in /usr/lib/allbase/hpsql/programs.

Using the sample DBEnvironment PartsDBE installed on your system as described earlier in this chapter, you can step through the process with the following commands. This example uses the C preprocessor; follow the steps shown here for your operating system.

For HP-UX: First, change into the hpsql directory created when you set up the sample DBEnvironment PartsDBE, as shown in an earlier step. Then change into the programs directory if it exists. If it does not exist, create it, then change into it as follows:

```
mkdir programs
cd programs
```

Issue the following commands, one at a time (some will take a few moments to execute):

```
$ rm cex2.sql Return   Only needed if this file exists already.
$ cp /usr/lib/allbase/hpsql/programs/cex2 cex2.sql Return
$ psqlc ../sampledb/PartsDBE -i cex2.sql -d Return

Series 800:
  $ cc cex2.c -o cex2.r -lsql -lcl -lportnls Return
Series 300:
  $ cc cex2.c -o cex2.r -lsql -lheap2 -lportnls -lpc Return

$ more cex2.sql Return
$ more cex2.c Return
```

This sequence copies the source file from the sample database directory to the local directory, then preprocesses and compiles the program. The more commands let you examine the differences between the embedded SQL source file and the preprocessor output file.

Use the following command to execute the program:

```
$ ./cex2.r Return
```

When prompted for part numbers, enter the following, and observe the results:

```
1123-P-01
1323-D-01
9999-X-01
```

For MPE XL: First, make sure you are in the group and account where PartsDBE exists. Then issue the following commands, one at a time (some take a few moments to execute):

```
: PURGE CEX2 (Return)   Only needed if this file exists already.
: FCOPY FROM=CEX2.SAMPLEDB.SYS;TO=CEX2;NEW (Return)
: PPC CEX2,PARTSDBE,CEX2P (Return)
: SAVE SQLOUT (Return)
: PRINT CEX2 (Return)
: PRINT SQLOUT (Return)
```

This sequence copies the source file from the sample database group and account to your local group, then preprocesses and compiles the program. The SAVE command keeps a copy of the preprocessor output file SQLOUT in your local group. The PRINT commands let you examine the differences between the embedded SQL source file and the preprocessor output file.

Use the following command to execute the program:

```
: CEX2P (Return)
```

When prompted for part numbers, enter the following, and observe the results:

```
1123-P-01
1323-D-01
9999-X-01
```

Examining Startup Parameters with SQLUtil

Use the SQLUtil program to examine the startup parameters of PartsDBE or any other DBEnvironment. Run the program as follows (if you are using HP-UX, first make sure you are in the same working directory from which you created PartsDBE):

```
sqlutil (Return)
```

SQLUtil displays a banner like the following (some details can be different on your system):

```
                                        WED, AUG 08, 1990, 11:30 AM
HP36217-02A.07.00.17         DBE Utility/9000            HP SQL/HP-UX
(C)COPYRIGHT HEWLETT-PACKARD CO. 1982,1983,1984,1985,1986,1987,1988,
1989. ALL RIGHTS RESERVED.

>>
```

Figure 4-9. SQLUtil Banner

Respond to the prompts as in the following dialog. For DBEName, enter hpsql/sampledb/PartsDBE in HP-UX, or PartsDBE in MPE XL.

```
>> SHOWDBE Return
DBEnvironment Name: DBEName
Maintenance Word:  Return
Output File Name (opt):  Return
-> ALL Return
```

The following display should appear (some details can vary on your system):

```
Maintenance word:
DBEnvironment Language:  n-computer
AutoStart:  ON
User Mode:  MULTI
DBEFile0 Name:  PartsF0
DDL Enabled:  YES
No. of Runtime Control Block Pages:  37
No. of Data Buffer Pages:  100
No. of Log Buffer Pages:  24
Max. Transactions:  5

>> E Return
```

Use the E command to exit from SQLUtil and return to your operating system.

The parameters shown above describe the configuration of the sample DBEnvironment immediately after setting it up with SQLSetup. All of them are described in the "DBEnvironment Configuration and Security" chapter of the *ALLBASE/SQL Database Administration Guide*, so they are not described further here. Many of them can be adjusted by the database administrator to meet the needs of a particular system.

Creating a Schema File with SQLGEN

Use option 4 from the SQLSetup menu to create a file that shows all the data definition commands that were used to build PartsDBE. As an alternative, run the SQLGEN program, as follows:

```
sqlgen Return
```

SQLGEN displays its banner (some details can be different on your system):

```
                                   WED, AUG 08, 1990, 11:30 AM
HP36217-02A.07.00.04       SQL Command Generator        HP SQL/UX
(C) COPYRIGHT HEWLETT-PACKARD CO. 1986,1987,1988,1989

>>
```

Figure 4-10. SQLGEN Banner

Respond to the prompts as in the following dialog. For DBEName, enter hpsql/sampledb/PartsDBE in HP-UX, or PartsDBE in MPE XL.

```
>> startdbe Return
DBEnvironment Name >> DBEName Return

DBEnvironment successfully started.

>> gen all Return

HP SQL Command Generator for ALL

Schema File Name or '//' to STOP command >> PartsSch Return
```

Messages are displayed for each of the commands SQLGEN is inserting into the file *PartsSch*. Below are some of the messages:

```
Generating command to START DBE PartsDBE
Generating command to CREATE DBEFILESET ORDERFS
Generating command(s) for DBEFILE ORDERDATAF1
Generating command(s) for DBEFILE ORDERINDXF1
Generating CREATE TABLE PURCHDB.INVENTORY
Generating CREATE TABLE PURCHDB.ORDERITEMS
.
.
.
```

If you created the schema file with SQLSetup, use option 5 to view the file. Otherwise, use an operating system command to examine file PartsSch.

Purging PartsDBE

Use SQLUtil to purge PartsDBE when you no longer need it. Within SQLUtil, you use the PURGEALL command:

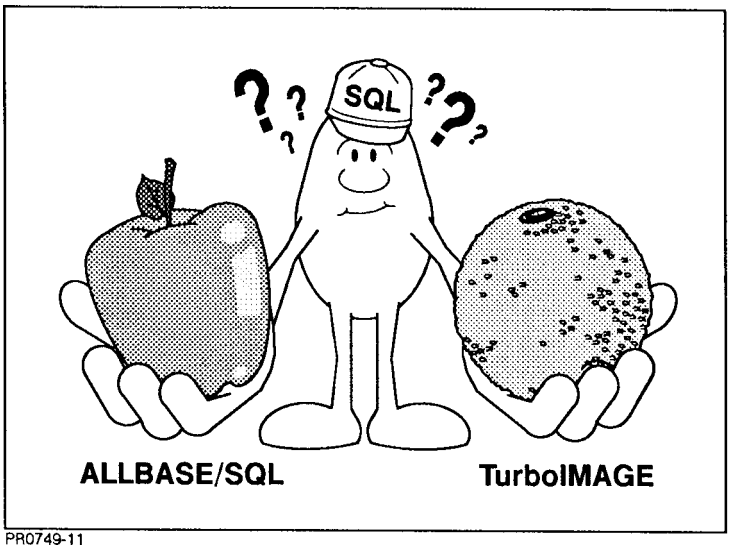
```
>> purgeall (Return)
DBEnvironment Name: PartsDBE (Return)
Purge DBEnvironment and Log Files (y/n)? y (Return)

DBEnvironment and Log Files purged.

>> exit (Return)
```

You can also use SQLSetup option 6 to purge PartsDBE.

5

Comparing ALLBASE/SQL with TurboIMAGE

If you are coming to ALLBASE/SQL from the world of TurboIMAGE, this chapter should help you make the necessary translations from the terminology of a network DBMS to the terminology of a relational DBMS.



PR0749-11

The following topics are covered:

- Basic structures.
- Procedures for starting up.
- Tables and indexes versus data sets.
- Mapping of data types.
- Differences in security.
- Differences in accessing databases.
- Sample mapping of a TurboIMAGE database to an ALLBASE/SQL DBEnvironment.
- Using ALLBASE/Turbo CONNECT.

The goal in discussing these topics is not to present a complete picture of either system, but rather to suggest some points of correspondence in order to make learning ALLBASE/SQL easier. The discussion is deliberately oversimplified. For definitions of ALLBASE/SQL terms, refer to chapter 1, "Very Basic ... ," and to chapter 6, "Glossary of Terms in ALLBASE/SQL." Complete information about TurboIMAGE is in the *TurboIMAGE/XL Database Management System Reference Manual.*

Basic Structures

Figure 5-1 shows the basic architecture of the TurboIMAGE system.

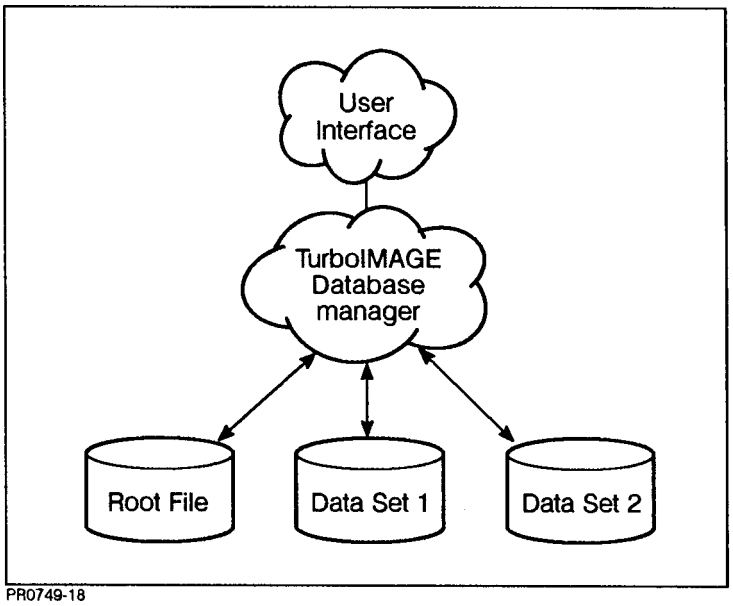


Figure 5-1. TurboIMAGE Architecture

The TurboIMAGE database manager accesses data in each data set as needed, based on information given in the root file.

Figure 5-2 shows the ALLBASE/SQL architecture.

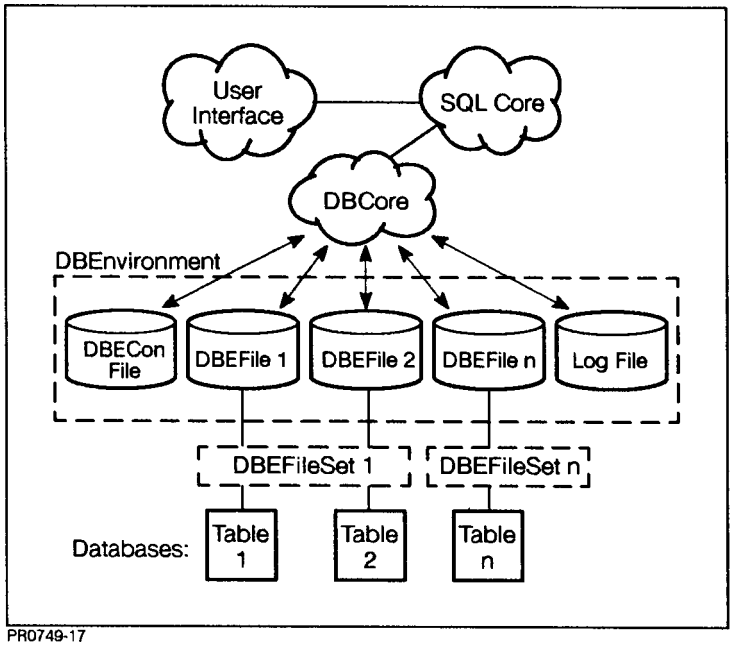


Figure 5-2. ALLBASE/SQL Architecture

ALLBASE/SQL has two components which manage access to databases, SQLCore and DBCore. The data is stored in tables which reside physically in DBEFiles and logically in DBEFileSets, as shown in Figure 5-2. These elements are described further in the next paragraphs.

Procedures for Starting Up

In TurboIMAGE, you start up a **database** with the following steps:

1. Create a schema. A common way of doing this is to enter the schema into a text file created with an editor.
2. Run DBSCHEMA to generate a root file from the schema.
3. Run DBUtil to create data sets based on the root file.

The schema contains definitions of all the data items in your database, and it describes all the data sets in the database. It also specifies the required security for the database by defining passwords for specific users.

In ALLBASE/SQL, you start up a DBEnvironment with a simple SQL command. The command may be issued interactively (through ISQL), or through an application program. The START DBE NEW command is as follows:

```
START DBE ' DBEnvironmentName' NEW
```

The START DBE NEW command creates a file known as a DBECon file, which is similar to the root file in TurboIMAGE.

The DBECon file contains information about startup parameters for the DBEnvironment and its logs. START DBE NEW also creates a structure within the DBEnvironment known as a system catalog, which is a set of information about all the databases in the DBEnvironment.

Use of a Schema

In TurboIMAGE, a schema is required to define a database. Most users create the schema in an ASCII file. Once the database exists, the schema can serve as a record of its contents. The schema can also be used to create the same database structure in different groups and accounts or on different systems.

You can also create a schema for ALLBASE/SQL by entering into an ASCII file all the SQL commands needed to configure the DBEnvironment and create all the objects in it. You can use this file as input to ISQL. However, in ALLBASE/SQL, no schema is required; all that is necessary to configure a DBEnvironment is a START DBE NEW command issued through ISQL or an application program.

Root File versus DBECon File and System Catalog

In TurboIMAGE, the root file, which is generated by DBSCHEMA from the schema, contains security information and definitions of all the data sets in the database, together with the name of the database creator.

The DBECon file in ALLBASE/SQL is created when you issue the START DBE NEW command. The DBECon file, which has the same name as the DBEnvironment, contains the name of the DBECreator and the names of the logs associated with the DBEnvironment. It also indicates startup parameters, such as SINGLE or MULTI user mode, and others (examples are shown in the section "Examining Startup Parameters with SQLUtil" in chapter 4).

In ALLBASE/SQL, structural information is also stored internally in a set of system tables known as the system catalog. This is like an internal schema. The DBECon file does not contain the names of tables or other database objects; these are stored in the system catalog.

Data Files for Data sets versus DBEFiles for Tables

Each data set in a TurboIMAGE database occupies a separate MPE XL file, created by TurboIMAGE as a PRIV file. The file size is determined from the capacity you indicate for the data set in the schema.

In ALLBASE/SQL, you create a table inside a DBEFileSet, to which you have added one or more DBEFiles. These DBEFiles need to be large enough for the amount of data you need to store. Instead of specifying a capacity, you create DBEFiles of whatever size is needed. DBEFiles can hold data and/or indexes for more than one table at a time; no simple correspondence between tables and data files exists. You do not specify the size of a table when you create it; the table size is limited only by the capacity of your system. As

you need additional space for a growing table, you add DBEFiles to the DBEFileSet in which the table was created. This increases the capacity of the table. (An exception to this is tables created as hash structures, described in a later section.)

Naming Conventions

The names of data items and data sets within a TurboIMAGE schema may contain some characters which are not allowed in ALLBASE/SQL. For example, the hyphen is not allowed in ALLBASE/SQL names; so hyphens need to be represented in some other way in ALLBASE/SQL, for example, by an underscore (_).

Tables and Indexes versus Data Sets

The basic unit of storage for data items in TurboIMAGE is the data set, which consists of a set of records containing an ordered series of data items. A data set is either a detail or a master, and, if it is a master, it is either manual or automatic.

In ALLBASE/SQL, the basic unit of storage is the table, which is an unordered set of rows containing data items. Tables are not labeled manual or automatic, master or detail; but the same relationships are possible through the creation of different types of indexes or index-like structures.

Automatic Masters versus Indexes

Each TurboIMAGE automatic master data set has a unique key item, which provides calculated access to the data in the master, and chained access to the data in one or more detail data sets. Like an ALLBASE/SQL index, an automatic master is maintained by the system; that is, when the table or detail data set is updated, the index or automatic master is updated automatically. Also, you cannot have an index without a table, and you cannot have an automatic master without a detail data set. Both indexes and automatic masters contain only key data values.

One difference between the automatic master and the index is that an automatic master may serve up to 16 detail data sets, whereas an index serves only one key within one table. However, separate indexes can be created on similar keys in other tables, and many indexes may exist on the same table. Another difference is that ALLBASE/SQL indexes use a B-tree structure, whereas TurboIMAGE master data sets use calculated (hash) access to key values.

Manual Masters versus Hash Structures

Each TurboIMAGE manual master data set has a unique primary key item, which provides calculated access to the data in the master, and chained access to the data in the detail data set (if one exists). A manual master may contain data items other than just the key item; therefore, it cannot be automatically updated when detail data changes.

When you create an ALLBASE/SQL table as a **hash structure**, it behaves like a manual master data set, in that it has a unique **primary key** with calculated access to the key value. Like manual masters, hash structures have the advantage of speed when the key value is known exactly, but are less efficient than normal indexes when a range of values is required.

Master/Detail versus Referential Integrity

The TurboIMAGE manual master provides the following methods for enforcing data integrity:

- Insisting that key values entered into the detail data set already exist in the manual master.
- Preventing deletions of key values in the manual master without prior deletion of the same key values in the detail data set.

In ALLBASE/SQL, you can achieve the same end by creating a table with a **referential constraint** and specifying the following clause:

```
HASH ON CONSTRAINT
```

This causes a unique hash key in the table (that is, the referenced or master table) to be related to a **foreign key** in another table (that is, the referencing or detail table).

Sort Items versus Indexes

In TurboIMAGE, you can specify sort items that become the basis for the sorted order of the output in queries to the database. You cannot use the search item as a sort item, however, because the search item points to a chain of entries whose order is fixed, and this order is not necessarily the same as the sort order.

In ALLBASE/SQL, you can use the ORDER BY clause in the SELECT command to sort by any column you wish. ALLBASE/SQL tables are essentially unordered sets of rows, so they can be accessed in any order. Sorting is improved markedly, however, by the use of an index on the sort key. Note that the sort key can be and often is the same as the primary key in the table.

Mapping of Data Types

Both TurboIMAGE and ALLBASE/SQL have data types that do not map exactly to a type in the other system. But a satisfactory mapping with appropriate conversions can easily be done for most TurboIMAGE data types.

Basic Mapping

Table 5-1 shows the mapping of the most common data types from TurboIMAGE to ALLBASE/SQL:

Table 5-1. Mapping of TurboIMAGE and ALLBASE/SQL Data Types

TurboIMAGE Data Type	ALLBASE/SQL Data Type	Description
I,J	SMALLINT	16-bit integer
I2,J2	INTEGER	32-bit integer
K1,K2	INTEGER	Requires conversion from binary to integer
P *n*	DECIMAL(*n*-1,0)	Packed decimal
R4	FLOAT	Conversion from HP 3000 real to IEEE real
U(*n*), X(*n*)	CHAR(*n*)	Byte character string
Z *n*	DECIMAL(*n*,0)	Requires conversion from zoned decimal to packed decimal

Compound Items

TurboIMAGE compound items are not compatible with ALLBASE/SQL data types, because ALLBASE/SQL does not accomodate arrays. In ALLBASE/SQL, you need to create a separate column description for each member of the compound.

Null Handling

TurboIMAGE does not support null values. Thus, a null value is often represented as an empty string or as zero (for numeric values).

In ALLBASE/SQL, a special data type NULL can be used to indicate the absence of a value; NULL is distinct from 0 or from an empty string, which are like any other values. If columns are not permitted to contain nulls in ALLBASE/SQL, you must define the column as NOT NULL when you create the table.

Differences in Security

TurboIMAGE and ALLBASE/SQL differ markedly in their implementation of security systems.

TurboIMAGE Security

The security of TurboIMAGE databases is determined partly by passing MPE file system security and partly by the assignment of user classes and passwords within the schema. Externally, database users must be valid users in the account where the root file resides. For internal security, you define a numbered set of classes and assign passwords to them, then you add the classes that have read and/or write access to each data item and data set description in the schema. When accessing the database, you must specify a password, which assigns you to a user class with particular permissions in the database.

Granting and Revoking Authorities

In ALLBASE/SQL, the DBA (database administrator) GRANTs and REVOKEs authorities that relate to the DBEnvironment as a whole or to specific tables within it. If you are the DBEnvironment's creator, you have DBA authority. Users can CONNECT to a DBEnvironment if the DBA grants CONNECT authority to their DBEUserIDs, which are related to login name. It is possible to CONNECT to a DBEnvironment from any account.

If you are the creator of a table, you have OWNER authority over it, which lets you perform any operation on it, including granting authorities to other users. Table authorities include the ability to SELECT, DELETE, INSERT, and INDEX. UPDATE authority can be granted for individual columns in a table or for the table as a whole.

Defining ALLBASE/SQL Groups

In ALLBASE/SQL, you can define authorization groups and then grant authorities to them; then you can add users to the groups, at which point they immediately receive the authorities the group possesses. This makes it possible to create an authorization scheme that is independent of any list of particular users and passwords. An authorization group may be a member of another authorization group.

Defining Views in ALLBASE/SQL

A different approach to security is possible in ALLBASE/SQL through the use of views. For a table that contains some sensitive information and some widely used information, you can create a view that contains only the widely-used information, grant appropriate access on the view to a wide range of users, then restrict the access on the base table to only a few users.

Differences in Accessing Databases

TurboIMAGE and ALLBASE/SQL both offer a variety of tools for accessing databases, and both provide techniques for **concurrency control**, to regulate access by more than one user at a time.

Interactive Access

TurboIMAGE interactive access is through Query/V, which lets you find database entries and report on them using the Query command language. In ALLBASE/SQL, the interactive interface is known as ISQL, which uses Structured Query Language (SQL) to access the database and display query results.

For sophisticated reporting, Business Report Writer supports both TurboIMAGE and ALLBASE/SQL databases.

Programmatic Access

A major difference between TurboIMAGE and ALLBASE/SQL is in the programmatic interface. TurboIMAGE uses a set of intrinsics which you use in application programs to open databases, obtain locks, retrieve data, unlock data items and data sets, and close a database.

ALLBASE/SQL uses **embedded SQL programming**. You insert standard SQL statements in an application program, then you preprocess the program to convert the SQL statements into valid procedure calls in the language you are using. The converted code is compiled and linked with a library of ALLBASE/SQL routines. The use of embedded SQL means that you can prototype and test your queries in ISQL before running them in an application, thus saving development time. Embedded SQL also includes a set of dynamic commands which let your end users perform **ad hoc queries** through your applications.

4GL

You can use ALLBASE/4GL as a programming tool to create applications that can access both TurboIMAGE and ALLBASE/SQL databases—even at the same time. Simply define the appropriate data sets and/or tables in ALLBASE/4GL's dictionary, then create screens and menus. As a fourth-generation tool, ALLBASE/4GL lets you avoid tedious and repetitive coding.

Differences in Concurrency Control

Concurrency control is needed to protect the consistency of a database when it is in multiuser operation. TurboIMAGE permits concurrent access through a mechanism known as access mode. You choose one of the eight modes of access as you open the database. These modes offer a wide range from very restrictive single user exclusive access to multiuser access with updates permitted by different users. Some access modes enforce the application's locking of data sets; others do not.

ALLBASE/SQL uses two DBEnvironment access modes: SINGLE and MULTI user mode, which you set when you create the DBEnvironment. (You can also change modes using SQLUtil.) In

addition, tables have an access mode, which you specify when you create them. Tables may be PUBLIC, PUBLICREAD, or PRIVATE, as follows:

PUBLIC	may be read or updated by anyone who has authority to CONNECT to the DBEnvironment.
PUBLICREAD	may be read by anyone but only updated by one user at a time.
PRIVATE	may only be read or updated by a single user at a time.

Locking Mechanisms

In TurboIMAGE, you use the DBLOCK intrinsic in certain access modes to provide **locking** at the database level, the data set level, or the data item level. Locking must be explicitly requested by the user; it is required for concurrent updates. You can request locks conditionally in TurboIMAGE, which means that the call returns if the lock request fails.

ALLBASE/SQL provides automatic locking for all data manipulation commands—reads and writes. Locking is unconditional, and it applies at the level of the table or the data page; row level locking is not supported. You can also use the explicit LOCK TABLE command. Further, you can specify an **isolation level**, which determines the kinds of locks obtained by ALLBASE/SQL. Isolation level applies to transactions, which are bounded by the SQL BEGIN WORK and COMMIT WORK commands. Four isolation levels are possible:

RR	**Repeatable Read**. The strongest locks are used to assure continuity of data from one read to another within the same transaction.
CS	**Cursor Stability**. Weaker locks are obtained and released as needed during the scan of a particular database table.
RC	**Read Committed**. Weaker locks are obtained but released even sooner during the scan of a particular database table.
RU	**Read Uncommitted**. No locks are obtained.

For complete information about these isolation levels, refer to the chapter "Concurrency Control Through Locks and Isolation Levels" in the *ALLBASE/SQL Reference Manual*.

Concurrency control is complex, and no exact mapping between the two systems is possible.

Sample Mapping of a TurboIMAGE Database to an ALLBASE/SQL DBEnvironment

Consider the following TurboIMAGE schema:

```
BEGIN DATABASE TIPART;

PASSWORDS:
   12 BUYER;
   14 CLERK;
   18 DO-ALL;

ITEMS:
   BINNUMBER      , K     (12,14/18);
   COUNTCYCLE     , 3I    (12/18);
   ITEMCOUNT      , I2    (/14,18);
   LASTCOUNTDATE  , X8    (12/18);
   PARTNAME       , X32   (14/12,18);
   PARTNUMBER     , X16   (14/12,18);
   SALESPRICE     , P12   (/12,18);
   WAREHOUSE      , X32   (/12,18);

SETS:
   NAME: PARTS, MANUAL;
   ENTRY:
      PARTNUMBER(1),
      PARTNAME,
      SALESPRICE;
   CAPACITY: 301;

   NAME: INVENTORY, DETAIL;
   ENTRY:
      PARTNUMBER(PARTS),
      BINNUMBER,
      ITEMCOUNT,
      WAREHOUSE,
      LASTCOUNTDATE,
      COUNTCYCLE;
   CAPACITY: 200;
END.
```

You might implement this design using a set of SQL commands such as the following:

```
START DBE 'TIPARTS' NEW;

CREATE GROUP BUYER;
CREATE GROUP CLERK;
CREATE GROUP DO_ALL;

COMMIT WORK;

CREATE DBEFILESET DATAFS;
CREATE DBEFILE DATAF1 WITH
```

```
   PAGES=200, NAME='DATAF1', TYPE=MIXED;
ADD DBEFILE DATAF1 TO DBEFILESET DATAFS;
CREATE PUBLIC TABLE INVENTORY
  (PARTNUMBER    CHAR(16)    NOT NULL
      REFERENCES PARTS (PARTNUMBER),
   BINNUMBER     INTEGER     NOT NULL,
   ITEMCOUNT     INTEGER     NOT NULL,
   WAREHOUSE     CHAR(32)    NOT NULL,
   LASTCOUNTDATE CHAR(8)     NOT NULL,
   COUNTCYCLE_1  SMALLINT    NOT NULL,
   COUNTCYCLE_2  SMALLINT    NOT NULL,
   COUNTCYCLE_3  SMALLINT    NOT NULL)
   IN DATAFS;
COMMIT WORK;

CREATE DBEFILESET HASHFS;
CREATE DBEFILE HASHF1 WITH
   PAGES=350, NAME= 'HASHF1';
ADD DBEFILE HASHF1 TO DBEFILESET HASHFS;
CREATE PUBLIC TABLE PARTS
  (PARTNUMBER    CHAR(16)      NOT NULL,
   PARTNAME      CHAR(32)      NOT NULL,
   SALESPRICE    DECIMAL(11,0) NOT NULL)
  HASH ON (PARTNUMBER) PAGES=301
  IN HASHFS;
COMMIT WORK;

REVOKE ALL ON INVENTORY FROM PUBLIC;
REVOKE ALL ON PARTS FROM PUBLIC;

GRANT SELECT ON INVENTORY, PARTS TO
      BUYER, CLERK, DO_ALL;
GRANT INSERT, UPDATE, DELETE ON
      INVENTORY, PARTS TO DO_ALL;
GRANT UPDATE ON INVENTORY
      (PARTNUMBER, ITEMCOUNT) TO CLERK;
GRANT UPDATE ON PARTS (PARTNUMBER, PARTNAME,
      SALESPRICE, WAREHOUSE) TO BUYER;
COMMIT WORK;
```

This mapping is intended as illustrative only, not an exact migration of the database. To create an exact mapping, you would create views to define specific subsets of the tables, grant authorities on the views to appropriate users, then revoke access to the base tables.

Using ALLBASE/Turbo CONNECT

ALLBASE/Turbo CONNECT allows you to select TurboIMAGE/XL data using ALLBASE/SQL. Using the utility program ATCUtil, you attach the TurboIMAGE database to an ALLBASE/SQL DBEnvironment as shown in Figure 5-3. The attachment process creates a mapping of data sets to tables like the one shown previously in this chapter.

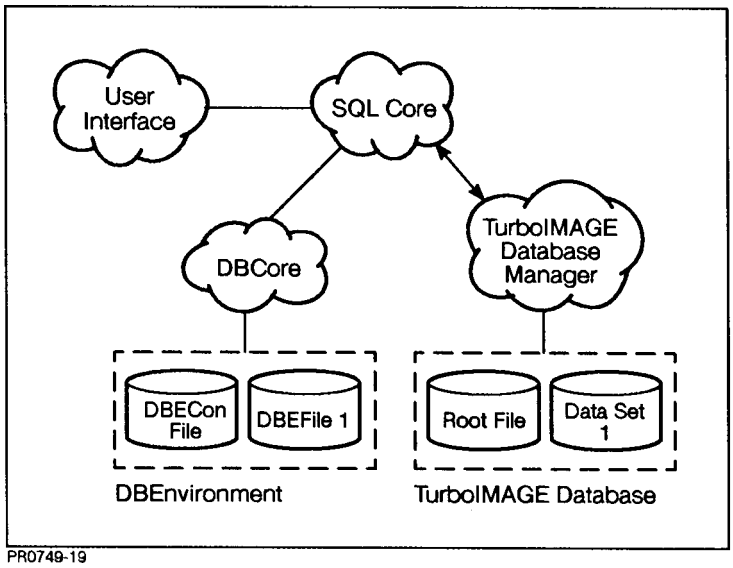


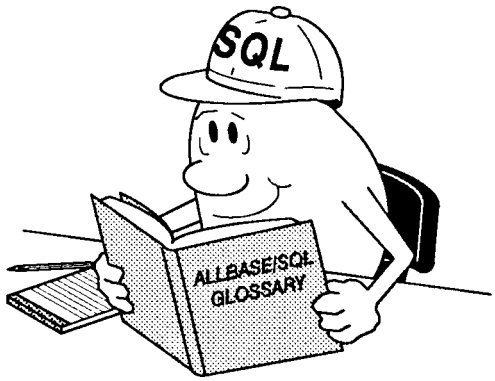
Figure 5-3. Using ALLBASE/Turbo CONNECT

After attaching, you can use ALLBASE/Turbo CONNECT for read-only access of TurboIMAGE data from an ALLBASE/SQL DBEnvironment—interactively or in application programs.

For complete information about ALLBASE/Turbo CONNECT, refer to the *ALLBASE/Turbo CONNECT Administrator's Guide.*

6

Glossary of Terms in ALLBASE/SQL



PR0749-13

Ad Hoc Query Type of query that is issued for the needs of a particular moment. It is usually not stored for later use or built into an application. Ad hoc queries are important in the use of relational databases for decision support.

Archive Logging Logging method that uses log files to roll back incomplete transactions after a system failure and to roll forward from an earlier DBEnvironment backup. Uses a relatively large file or set of files to record all activity that modifies databases from the point at which you do a backup of the entire DBEvironment. If the logs are intact following a hard crash, you can recover a DBEnvironment from an earlier saved version.

Attribute A characteristic of a data element considered during database design. As you organize your data, you arrange it into categories that possess similar attributes. The categories are known as entities.

Authority Permission to access specific objects for specific purposes within an ALLBASE/SQL DBEnvironment. Three major types are **SPECIAL authority**, **TABLE authority**, and **RUN authority**.

Authorization Group See Group.

Base Table Table upon which a view is based.

Class Special category of ALLBASE/SQL owner that is neither a particular DBEUserID nor a group. You do not explicitly create a class; you create it implicitly by creating objects owned by it. A class does not have members like a group. Objects owned by classes can be dropped or modified only by a DBA. A class does not have a password associated with it.

Clustering Index An index which attempts to locate new rows in physical proximity to other rows with similar key values. Valuable when a large number of inserts follows a similar large number of delete operations.

Column Vertical division within a database table. Analogous to a field in a file.

Column Authorization Permission to update a specific column within a table.

Column List One or more columns specified as part of a query result.

Concurrency The ability of multiple users to access the same database files simultaneously. Concurrency is regulated by locking, which controls the degree of concurrent access permitted—from exclusive read or write access to shared read with concurrent updates.

Constraint A condition placed upon a column or table that requires values in the column or table to meet certain conditions before a row can be inserted or deleted. Two types supported by ALLBASE/SQL are unique constraints and referential constraints.

Cursor Stability (CS) An isolation level that guarantees that any data on the page you are currently accessing cannot be updated by other users until you move off that page. This offers a greater degree of concurrency than Repeatable Read, which is the default isolation level.

Data Analysis Study of raw data before building a database. Concerns the kind of data that is to be stored and how the data is to be used.

Database A structured arrangement of data elements designed for the easy selection of information. In ALLBASE/SQL, a database is a collection of tables, views, and indexes having the same ownership in a DBEnvironment. A DBEnvironment may contain several databases.

Database Administrator (DBA) The individual with DBA authority who creates and maintains objects in a DBEnvironment. DBA authority permits the use of certain restricted SQL and SQLUtil commands or options, and also confers co-ownership of all the objects in a DBEnvironment.

Database Design The creation of a specific arrangement of data in tables or data sets with an appropriate security structure.

Data Control Language The set of SQL commands that control access to data. This includes the ADD, REMOVE, GRANT, and REVOKE commands, as well as the commands to create, manage, and drop authorization groups. Also known as DCL.

Data Definition The process of creating and dropping database objects.

Data Definition Language The set of SQL commands that create and drop database objects. This includes the commands to create and remove DBEFileSets, DBEFiles, tables, views, and indexes. Also known as DDL.

Data Manipulation The process of access data within a database.

Data Manipulation Language The set of SQL commands that access data. This includes the actions of selecting data, inserting rows, updating columns, and deleting rows. Also known as DML.

Data Type A kind of data that can be stored in database tables. Valid types are CHARACTER, VARCHAR, INTEGER, DECIMAL, FLOAT, DATE, TIME, DATETIME, INTERVAL, BINARY, and VARBINARY. LONG varieties of BINARY and VARBINARY are also available.

DBA Authority The most powerful authority within an ALLBASE/SQL DBEnvironment. Includes the authority to create new objects, drop all existing objects, and grant or revoke all authorities for other users. DBA authority implies co-ownership of all objects within the DBEnvironment.

DBCore A central component of ALLBASE/SQL that performs physical file access and logging. DBCore also provides concurrency control through the use of isolation levels and locking.

DBECon File DBEnvironment Configuration File. This contains startup parameters for the DBEnvironment. The contents of this file are initially determined at the time you issue the START DBE NEW command. You can modify some of these parameters using SQLUtil, and you can override some of them with the START DBE command.

DBECreator The individual who issues the START DBE NEW command. Some maintenance operations require you to be the DBECreator.

DBEFile File containing data or indexes or both. A DBEFile of type TABLE can only contain table data; a DBEFile of type INDEX can only contain index data; a DBEFile of type MIXED can contain both table and index data. DBEFiles are operating system files and are named according to the conventions of the operating system.

DBEFileSet Logical grouping of DBEFiles. You associate newly created DBEFiles with a DBEFileSet, and you specify a DBEFileSet when you create a table.

DBEnvironment A collection of files containing one or more databases. Files include the DBECon file (which holds startup parameters and log file names); DBEFile0, which contains the system catalog; and log files. A DBenvironment may also contain additional DBEFiles for table and index data. The DBEnvironment is the maximum scope of a transaction within ALLBASE/SQL.

DBEUserID In HP-UX, a login name. In MPE XL, a login name and account name joined with the character '@'. One type of owner of database objects.

Embedded SQL Program An application program incorporating SQL statements for programmatic access to ALLBASE/SQL databases. Each embedded SQL statement begins with the keywords EXEC SQL. Embedded SQL programs are preprocessed, then compiled before execution. For most SQL commands, the preprocessor stores a section, or runtime version of the command, in the DBEnvironment.

Entity Basic subdivision of data elements in database design. Each entity is a thing or event about which information is kept in the database. For each entity, there is at least one attribute that uniquely identifies a data element as belonging to the entity.

Explicit Locking Locking of tables in transactions by the use of the LOCK TABLE command.

Expression specifies a value. The most common sources of values are columns in a table or host variables in an application program. Expressions are used to identify columns or rows or to define new values for columns.

Foreign Key A column or columns in a table which have a relationship to a primary column or columns in a different table such that the value must exist in the primary key column before it can be inserted into the foreign key column, and it must be deleted from all foreign key columns before it can be deleted from the primary key column.

Group Authorization group. Membership in a group is used to confer common ownership or common authorization for other objects in the DBE. You create a group explicitly, using the CREATE GROUP command, then you add users to it. You can then grant authorizations to the group or revoke authorizations from the group. You can also use the group name for the ownership of database objects.

Hash Structure ALLBASE/SQL table containing rows that are stored in such a way as to permit fast access to specific tuples by means of a hash function. A hash structure provides a method for quickly finding a row by calculating its location based on the value of the hash key, which you specify when you create the table.

Host Variable A variable in an application program that receives data from an ALLBASE/SQL database (output host variable) or passes data to the database from the program (input host variable).

Implicit Locking Locking of tables in transactions according to table type and isolation level. For example, PRIVATE tables are locked exclusively for all access; PUBLIC tables are locked exclusively only for write operations.

Index A data structure that potentially speeds access to table data through the use of an index scan. The four types of index are: unique, **clustering**, unique and clustering, and non-unique and non-clustering. An index is created for one or more key columns in the table.

Index Scan A methoe of looking up each row in an index to find its location in the data file, then accessing the row in the table. This kind of access requires the existence of a B-tree index, which you must create. You do not explicitly request an index scan. Instead, SQLCore makes this choice if the query optimizer decides that the use of an index is the best way to access the data.

Integrity Constraint A device that ensures that a database contains only valid data. Two types are the referential constraint and the **unique constraint**.

ISQL The interactive interface to ALLBASE/SQL. ISQL is the tool you use for ad hoc queries as well as for loading and unloading data and other database administration tasks.

Isolation Level The degree of separation enforced between the transactions of different users. There are four levels: Repeatable Read (RR), Cursor Stability (CS), Read Committed (RC), and Read Uncommitted (RU). You specify an isolation level in the BEGIN WORK command.

Join A query that accesses data from two or more ALLBASE/SQL tables at a time. A join column is a column that occurs in both tables of a join (often it is a key column) and contains similar values in both tables.

Key One or more columns on which an index, hash structure, or integrity constraint are based.

Key Column A column which is indexed, or a column which participates in integrity constraints as all or part of a PRIMARY or FOREIGN key.

Key Value The value contained in the columns of a key. Key values are stored in index pages along with pointers to the location of rows in data pages.

Locking A technique for concurrency control through which ALLBASE/SQL restricts access to data by one individual when the data is being used by another. Locks are of three types: shared, exclusive, or shared with intent to become exclusive. Lock type is determined by the type of table being accessed and by the kind of operation the user is performing. Locks are released when a transaction ends with a COMMIT WORK command.

Logging The use of log files to record operations that modify database files. Logging is of two kinds: **nonarchive logging**, and **archive logging**. Both kinds permit you to roll back incomplete transactions following a system failure. This maintains data integrity by backing out changes to the database that were not committed. Only archive logging allows you to roll forward from an earlier version of a DBEnvironment by reapplying all committed transactions up to a specific recovery time.

Message Catalog A file containing ALLBASE/SQL error and warning messages. When a message is displayed, its text comes from this file. You can look up the meaning of the message in the *ALLBASE/SQL Message Manual*.

Message File An error and warning file known as SQLMSG generated by a preprocessor session. It contains any errors generated during the preprocessing of an embedded SQL program.

Modified Source File The file that results from using the preprocessor on an embedded SQL source file. The modified source file can then be compiled into an executable program.

Module A group of **sections** stored in the DBEnvironment when an embedded SQL program is preprocessed or when you use the PREPARE command in ISQL. The sections are activated when the program is run or when the EXECUTE command is issued in ISQL.

Native Language The language of the DBEnvironment or of specific CHAR and VARCHAR columns in a table. You specify the DBEnvironment's language in the START DBE NEW command using the LANG= clause. In table creation, you can use the LANG= clause as part of a character column description. The default language is known in HP-UX as n-computer; in MPE XL, it is NATIVE-3000.

Nonarchive Logging The default logging method. Uses log files to roll back (that is, undo) incomplete transactions that were not committed at the time of a system failure.

Normalization A formal process of adjusting table design in relational databases by examining and adjusting the relationships among columns.

Object A structure created and stored in an ALLBASE/SQL DBEnvironment. The most common objects are tables, views, indexes, and groups.

Optimizer Component of SQLCore which chooses the access path in processing a query. In optimization, ALLBASE/SQL chooses whether to use serial access to the data, or whether to use an index or hash structure if they exist. If there is a choice among indexes, the optimizer calculates the best access path.

Owner A DBEUserID, a group name, or a **class** name. Ownership applies to database objects such as tables, views, indexes, and authorization groups. The owner may drop the object or transfer it to some other owner.

Predicate Part of query syntax that specifies a subset of rows to be returned in the query result. Predicates are introduced by the keyword WHERE, so they are sometimes called WHERE clauses.

Predicates let you specify a range of values. The comparison predicate lets you compare a column value with a constant or **host variable**; the LIKE predicate lets you compare a column value with a portion of a character string; the BETWEEN predicate specifies a range of values for a comparison. Special predicates of various kinds let you search for rows in more complex ways, including the use of subqueries.

Preprocessor A component of ALLBASE/SQL that converts an embedded SQL program into a modified source file for input to a compiler in one of several programming languages: C, COBOL, FORTRAN, and Pascal.

Primary Key A column in a table defined so as to permit reference by foreign keys in other tables. A primary key also enforces uniqueness within the column.

Projection Relational operation that extracts a subset of columns from a table.

Query Request for information from database tables. A typical example is a SELECT statement.

Query Language A set of operators, expressions, and commands that let you manipulate a database. The query language of ALLBASE/SQL is SQL.

Query Result The rows retrieved by a SELECT statement. Query results are also known as result tables.

Read Committed (RC) An isolation level that guarantees only that data you read in a transaction has been committed by some earlier transaction; that is, it is not currently in the process of update by some other transaction at the time you are reading it. In practical terms, this means that another transaction can update or delete the same row before your transaction is over. However, concurrency is greatly improved.

Read Uncommitted (RU) An isolation level that enforces no separation between your transaction and those of others, because no locks are obtained for reads. This level permits dirty reads, that is, reading data from the data buffers that has not and may never be written to the database at all.

Referential Constraint An integrity constraint that enforces a relationship between the rows of two tables. Any value you attempt to insert into a table that has a referential constraint must either be NULL or be the same as a value in the referenced table.

Relation See Table.

Relational Operations Ways of extracting data from relational tables. The three primary relational operations are selection, projection, and joining.

Relationship The meaningful interaction of entities in database design. Relationships may be one-to-one, one-to-many, or many-to-many.

Repeatable Read (RR) An isolation level that enforces the highest level of separation between the transactions of different users. This level guarantees that when you re-read any data you have read previously in the same transaction, the value seen in the second read will be the same as the value seen in the first read. In practical terms, this means that other users may not update any data you have read at this isolation level until you COMMIT WORK.

Result Table See Query Result.

Rollback Recovery A process by which ALLBASE/SQL backs out of incomplete transactions using a log file. If a DBEnvironment stops while some transactions are still in progress, they must be undone the next time the DBEnvironment starts up.

Rollforward Recovery A process by which ALLBASE/SQL reapplies transactions to a DBEnvironment from a log file. Rollforward recovery requires the use of archive logging.

Row Horizontal division within a database table. Analogous to a record in a file.

Run Authority Permission to execute stored sections that perform ALLBASE/SQL queries or other operations from an application program. Required in addition to permission at the operating system level to execute the application itself.

Schema An ISQL command file containing commands to create a DBEnvironment and the objects within it, such as DBEFileSets, DBEFiles, tables, views, indexes, and authorities. You can create a schema file with an editor, or you can generate one from an existing DBEnvironment by using SQLGEN.

Also, a TurboIMAGE database definition which is the input to the TurboIMAGE DBSCHEMA program.

Section An SQL command stored in the DBEnvironment for use at run time by an application program. When sections are valid, they can be executed immediately by ALLBASE/SQL. When they are invalid, they must be revalidated at run time before execution.

Serial Scan A method of reading sequentially from the start of a table until the row is found. Also called table or relation scan. This is the default scan method used to access rows in a table when indexes do not exist. If indexes do exist on a table, the optimizer chooses whether to perform an index scan or a serial scan.

Special Authority Permission to use the DBEnvironment in particular ways. CONNECT authority lets you establish a user session. RESOURCE authority lets you create and drop objects such as tables, views, DBEFiles, etc. DBA authority gives you permission to perform all SQL and SQLUtil commands, and it grants co-ownership of all objects in a DBEnvironment.

SQL See Structured Query Language.

SQLCore A central component of ALLBASE/SQL. SQLCore checks the syntax of commands and prepares them for processing. SQLCore also optimizes queries, that is, chooses the best access path to the data.

SQLGEN A utility program for database administrators that generates the SQL commands necessary to re-create all or part of a DBEnvironment. The output from SQLGEN is a command file (sometimes called a schema) that can be used as input to ISQL in re-creating database objects.

SQLMigrate A utility program for database administrators that assists in migrating a DBEnvironment from one version of ALLBASE/SQL to another without unloading and reloading data.

SQLUtil A utility program for database administrators that assists with DBEnvironment maintenance, backup, and recovery. SQLUtil also lets you modify the startup parameters for a DBEnvironment.

Structured Query Language A standard query language syntax defined by ANSI standards in the United States nad X/OPEN standards in Europe. The relational database query language used by ALLBASE/SQL.

Subquery A query within another query. An example is a subquery embedded in the predicate of another query. The result of the inner query is used to evaluate the outer query.

SYSTEM A DBEFileSet created by ALLBASE/SQL when you issue the START DBE NEW command. The DBEFile known as DBEFile0 is associated with SYSTEM, which is the DBEFileSet containing the system catalog. You can add DBEFiles to SYSTEM as you would to any other DBEFileSet.

Also, a special user associated with the system views in the system catalog.

System Catalog A system-maintained database of tables and views owned by the special user SYSTEM and containing information about all the objects in the DBEnvironment. Contains data about all the objects created in the DBEnvironment. Differs from the DBECon file, which contains startup parameters, not object definitions.

System Table See System View.

System View A component view within the system catalog. You can issue queries on the views in the system catalog just as you would on ordinary database tables to display information about the DBEnvironment.

Table Basic unit of data storage in a relational database. Also known as a relation. Tables consist of rows and columns. A result table is a query result displayed in tabular form.

Table Authority Permission to use specific SQL commands on particular tables. There are several kinds of TABLE authority: SELECT, INSERT, DELETE, UPDATE, and INDEX. SELECT, INSERT, and DELETE let you operate on rows or sets of rows in a table; UPDATE lets you modify specific rows or columns in a table; and INDEX lets you create indexes on a table.

Transaction A unit of work in ALLBASE/SQL. Also, a unit of DBEnvironment logging and recovery. A transaction is started with a BEGIN WORK command and is ended by a COMMIT WORK command. The BEGIN WORK statement may be implicitly issued by ALLBASE/SQL if no other transaction is current.

Unique Constraint An integrity constraint which requires that no two rows in a table have the same values in a specified column or columns.

Unique Index An index which requires that no two rows in a table have the same key value.

Validation The process by which ALLBASE/SQL marks a section valid in the system catalog. A section is marked valid if all the objects it refers to exist, and if it has been optimized. A valid section can be executed immediately at run time with no further preparation.

View a table derived by placing a "window" over one or more tables. The derivation of a view is a SELECT command. View names are governed by the same rules as table names.