CHAPTER 1

On Your Marks ... Get Set ... Go!!! An Introduction to the Apache Derby and IBM Cloudscape Community

Introduction

Welcome to the Apache Derby and IBM Cloudscape family! We are glad to have you join our community of thousands, all dedicated to open standards and open source computing.

On August 3, 2004, IBM released a technical preview of IBM Cloudscape Version 10. While this in itself is exciting news, more importantly, IBM also made one of the most significant contributions in the history of the open source database community by contributing the source code for IBM Cloudscape to the Apache Software Foundation as the Apache Derby database. The Apache Software Foundation subsequently accepted Apache Derby as an incubator project. At the time this book was published, Apache Derby and IBM Cloudscape Version 10.1 was released and the Apache Derby project graduated from an incubator project (more on that in a bit) to an Apache sub-project.

The history of the Cloudscape technology is an interesting one indeed. In 1997, Cloudscape Inc. released what was perhaps the world's first real Java-based database, well before Java became the darling of the information technology (IT) community. The Cloudscape database began to carve its reputation as a pure Java-based relational database with just a 2 MB "fingerprint" that simplified application development. It made applications easier to deploy on any platform and represented the first truly relational database for which a DBA was not a requirement! Its significance went well beyond a "lights out" no management database. Cloudscape opened the doors for application developers (who are not generally database savvy) to leverage the benefits of a persistent data store for their applications.

In 1999, Informix bought Cloudscape Inc. and continued to enhance this database server, seeding it with even richer self-managing, ease of deployment, and open standards features—many of which were ahead of their time. IBM acquired the Cloudscape technology in July 2001 by purchasing the assets of Informix Software. As you can see, the IBM Cloudscape (formerly just Cloudscape) database has been around for a long time, which accounts for its maturity, robustness, and functionally rich feature set.

After the Informix acquisition, the IBM Cloudscape development teams have reported into IBM's Database Technology Institute, under the direction of Don Haderle and Dr. Patricia Selinger, two of the founders of relational database technology. In fact, since IBM acquired this technology, it has funded a continually expanding development team through several new versions and added compatibility with the IBM DB2 Universal Database (DB2 UDB) family through a fully compliant SQL application programming interface (API). This means that applications written for IBM Cloudscape (and subsequently Apache Derby—more on that in a bit) can easily be migrated to the DB2 UDB platform if necessary.

You might be surprised to know just how many partners, customers, and software packages use IBM Cloudscape in their technology—the very technology you're about to learn how to program to. In fact, over 80 different IBM products use IBM Cloudscape for many different reasons, including ease of deployment, portability, "hands-free" operation, an open standards-based Java engine, the small footprint, and more. For example, IBM Cloudscape powers products such as WebSphere Application Server, DB2 Content Manager, WebSphere Portal Server, IBM Director, Lotus Workplace, and many others. The IBM Cloudscape engine is a transparent component of all these products, and that is the whole point! Despite its application-transparency, you should be able to sense the power of this database engine because it is trusted to support these enterprise products.

As you probably have figured out by now, IBM has open sourced the IBM Cloudscape technology as Apache Derby and remains solidly committed to its future success. Throughout this book, we will use the term Apache Derby to represent both the IBM Cloudscape and Apache Derby databases because they are identical. There are some add-ons for IBM Cloudscape that you can freely download; where appropriate, we will identify them as such.

Specifically, IBM supports the Apache Derby open source database by:

- Contributing the IBM Cloudscape code to serve as the code base for the Apache database
- Donating resources to host a central Web-based location for add-on code, educational materials, support, the IBM DB2 Universal JDBC Driver, interfaces for ODBC, PHP, Perl, .NET, and more. The IBM Cloudscape Web site is available on IBM's developerWorks at: www.ibm.com/developerworks/cloudscape.
- Funding a dedicated open source project team that includes a complete development group with database development experience. Some members of this team are Apache Derby committers
- Providing hardware resources for nightly builds and testing of the open source code lines
- Running a full set of software and function verification test cases, and creating a fully functional open source test suite for Apache Derby
- Monitoring multiple Apache- and IBM-based forums and mail lists to offer free advice and best practice information to support the Apache Derby community

Why should you be interested in developing on the Apache Derby platform? The Apache Derby platform:

- ***Is easy to deploy.*** This database requires no installation: just copy a 2 MB .JAR file and set the `CLASSPATH` environment variable. Because it is Java-based, the database runs anywhere a standard JVM (J2SE 1.3 or higher) can be installed: from Macintosh to mainframe, and all points in between. In addition to this Apache Derby databases (and their data) are platform-independent and can be moved to any machine by simply zipping up your files and emailing them.
- ***Requires no DBA skills or administration effort.*** When using Apache Derby as an embedded solution, the application automatically starts the relational engine. Upgrades to future versions are done in-place, at connection time. Apache Derby automatically reclaims space, updates statistics, and much more. Quite simply, you do not need to staff your project with a DBA.

To be a player in the open source movement, you must be a proponent of open standards. Arguably, no other large IT company in the world has done more for the open source movement and open standards than IBM. In fact, today IBM backs over 150 different open source projects.

Specifically, Apache Derby uses open standards technology such as the American National Standards Institute's Structured Query Language (ANSI SQL), JDBC, SQLJ, and more. In fact, because Apache Derby is open source, future plans could include distributing Apache Derby with various mainstream Linux distributions as an accompanying database.

The IBM user community provides valuable feedback to this open source project, including bug reports and feature requests. Although IBM Cloudscape is the same database as the Apache Derby database available to the open source community, the IBM Cloudscape product has a separately purchasable service and support contract, and comes with a graphical-based installation and setup wizard to make full platform deployments (which include binary add-on features from IBM, such as an ODBC interface) easier and more consolidated.

Apache Derby addresses a unique market need and is a great fit for many workloads. Although Apache Derby enriches an environment steeped with a heavy Java investment and an open source middleware stack, it isn't just for Java developers. APIs are provided (either directly or through a download) for PHP, Perl, CLI, ODBC, and .NET.

In this chapter, we'll discuss the Apache Derby and IBM Cloudscape platforms, differences between them (again, there is nothing from a code perspective), why IBM is involved, why now, more than ever, you should consider a relational storage engine for your applications if you're not already using one, and more.

If You're Not the Kind of Person Who Reads Introduction Chapters …

We recommend that you read this chapter anyway to get a good level of detail about Apache Derby, the motivation behind what IBM is doing in the open source space (it's a continuation of previous works actually), why open source in the first place, and more.

If you're not going to read this chapter, but you want a one-minute overview of Apache Derby and IBM Cloudscape, the following paragraphs, courtesy of Kathy Saunders (Release and QA Manager for IBM Cloudscape) and Jean Anderson (Architect for IBM Cloudscape) summarize this product quite well:

"Apache Derby is a lightweight, embeddable relational engine in the form of a Java class library. Its native interface is Java Database Connectivity (JDBC), with Java-relational extensions. It implements the SQL92E standard as well as many SQL 99 extensions. The engine provides transactions and crash recovery, and allows multiple connections and multiple threads to use a connection. Apache Derby can be easily embedded into any Java application program or server framework without compromising the Java-ness of the application because it is a Java class library. Derby's support for complex SQL transactions and JDBC allows your applications to migrate to other SQL databases, such as IBM DB2Universal Database (UDB), when they need to grow.

Apache Derby Network Server provides multi-user connectivity to Apache Derby databases within a single system or over a network. The Apache Derby Network Server receives and replies to queries from clients using the standard Distributed Database Architecture (DRDA) protocol. Databases are accessed through the Apache Derby Network Server using the IBM JDBC driver and the DB2 UDB JDBC universal driver.

There are several technical aspects that differentiate Apache Derby from other database systems:

- *It's easy to administer. When embedded in a client application, a Derby system requires no administrative intervention.*
- *It's embeddable. Applications can embed the Database Management System (DBMS) engine in the application process, eliminating the need to manage a separate database process or service.*
- *It can run as a separate process, using the Network Server framework or a server framework of your choice.*
- *It is a pure Java class library: This is important to Java developers who are trying to maintain the advantages of Java technology, such as platform independence, ease of configuration, and ease of installation.*
- *It needs no proprietary Java Virtual Machine (JVM). Written entirely in the Java language, it runs with any certified JVM.*
- *The engine is lightweight. It is about 2 MB of class files, and it uses as little as 4 MB of Java heap.*
- *It provides the ability to write stored procedures and functions in Java that can run in any tier of an application. Derby does not have a proprietary stored procedure language; it uses JDBC.*

Apache Derby is also like other relational database systems. It has transactions (commit and rollback), supports multiple connections with transactional isolation, and provides crash recovery.

The unique combination of technical capabilities allows application developers to build data-driven applications that are pervasive (run anywhere), deployable (downloadable), manageable, extensible, and connectable."

Let Me Get This Straight, Apache Derby Is IBM Cloudscape?

IBM contributed the IBM Cloudscape database technology to the Apache Software Foundation as *Apache Derby* while at the same time announcing the *IBM Cloudscape Version 10* database (the commercial offering based on the same open source technology).

What does all this mean? As far as the code goes—how you use it, class names, files names, and practically anything else you can think of—you can use the terms Apache Derby and IBM Cloudscape interchangeably (the exception to this is Apache-specific licensing rules that are governed by the Apache Software Foundation). Both Apache Derby and IBM Cloudscape are built from the same open source Apache Derby codebase. For the most part, all packages (class names, command names, stored procedure names, and so on) use the same naming convention under the Apache Derby name.

There are some minor differences between Apache Derby and IBM Cloudscape with respect to surrounding (or add-on) technologies that you should be aware of, and we will cover them later in this chapter.

The Apache Derby database, at the time this book was written, is available from the Incubator page on the Apache Web site (http://incubator.apache.org/projects/index.html) as both source code and compiled binaries. If you download Apache Derby, your usage of Apache Derby is bound by the Apache License Version 2.0. The Apache License is friendly to commercial businesses (which makes this a great product that provides a zero-cost, rich, and robust data runtime). In fact, anyone who binds to it can create his or her own commercial product. You can learn more about the Apache Software Foundation's licensing terms at: www.apache.org/licenses.

As previously mentioned, the Apache Derby database is distributed both in open source code and in compiled binary format on the Apache Web site. To become a full-fledged Apache project, a project (like Apache Derby) must start in the Apache incubator, where it is given the opportunity to demonstrate that it can become part of a viable Apache community composed of individual contributors (you can see a list of the current contributors for Apache Derby at: http://incubator.apache.org/projects/derby.html#Detailed+References%3A) from both within and outside of the contributing company (in this case IBM). To learn more about how Apache incubator projects work, see http://incubator.apache.org/. Apache Derby will not be fully accepted by the Apache Software Foundation until it graduates from the incubator project. Incidentally, during the production phase of this book, Apache Derby graduated from the incubator phase to an official Apache sub-project.

At the time this book was written, the Apache Derby contribution was in a "Podling" stage and therefore requires the accompanying disclaimer (from http://incubator.apache.org/derby):

"Apache Derby is an effort undergoing incubation at the Apache Software Foundation (ASF), sponsored by IBM. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is

> *not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.”*

In contrast, the IBM Cloudscape database is only available in binary format; however, you do not need to pay for this license either. You can freely download and use the IBM Cloudscape database from the IBM developerWorks Web site at: http://www.ibm.com/developerworks/cloudscape.

Whether you’re developing an Apache Derby, IBM Cloudscape, .NET, open source (Python, PHP, or Perl), or Java application, bookmark this site. It is the most comprehensive developer resource for the IBM databases.

This Web site provides resources for developers contributed by Apache Derby community members from inside and outside IBM, such as technical articles, sample code, user forums, documentation, and links to the IBM Cloudscape add-ons.

One difference between Apache Derby and IBM Cloudscape is that you can purchase a support contract for IBM Cloudscape. A support contract delivers a service level agreement (SLA) from IBM that comes from its award winning, 24 × 7 ‘best-of-breed’ service support team. Most of the personnel on this support team also contribute time and effort to the Apache Derby project by monitoring forums and newsgroups. However, a support contract enables businesses to rely on a paid support structure for critical application deployments in the same way they would receive support for proprietary products—this means defect repudiation, committed response times, severity classifications, and so on.

Other differences between IBM Cloudscape and Apache Derby are that IBM Cloudscape has:

- A graphical installation wizard that lets you optionally select add-on IBM Cloudscape functional components like the ones detailed in this list. It is very well suited for developer workstation deployments.
- The DB2 Universal JDBC driver, commonly referred to as the Java Common Client (JCC). This JCC driver is available for download from the IBM Cloudscape Web site. The JCC driver can perform Type 2 and Type 4 JDBC connections, and it shares a common codebase that can be used for JDBC connections across the entire DB2 and IBM Cloudscape family (there are licensing considerations for some of these connections). The benefit is portability if the need ever arises for a more enterprise-class database like DB2 UDB. Because the same code manages the connection and application flow between the hosting application and the database, you can be assured that migration efforts are trivial.
- The IBM DB2 Plug-In for Eclipse (available as a free download from the IBM developerWorks Web site). This download includes:
  - The base Eclipse software developer’s kit (SDK).
  - An Eclipse-based plug-in for Apache Derby, IBM Cloudscape, DB2, or Informix schema and SQL statement development. This tool enriches functions previously delivered by the old Cloudscape `Cview` utility.

  - An Eclipse-based plug-in for schema and data migration (with validity checking) from Apache Derby to DB2 for Linux, UNIX, and Windows.

- Additional data access APIs like ODBC, PHP, Perl, and .NET interfaces. These interfaces can all be downloaded from developerWorks (although we've included links to them on the book's Web site [www.ibmpressbooks.com/title/0131855255] to make it easier to get going). White papers and other supporting collateral will accompany them as they become available. For example, the ODBC collateral will explain how to interface with Perl, PHP, and .NET applications using the ODBC driver.
- Preloaded sample databases, including the traditional Cloudscape TOURSDB database and the DB2 SAMPLE database (these are included in the installation package).
- Scripts to help you get up and running more quickly by setting your `CLASSPATH` environment variable, or by running some of the data utilities, and so on (these are included in the installation package).
- Sample programs that test your database server.
- An installation program that includes an IBM Java Runtime Environment (JRE) which is automatically installed with IBM Cloudscape for Windows and Linux (IBM Cloudscape also ships a Java-based installer for other platforms, but you have to ensure that a JRE is installed and configured on your server.)
- Documentation in PDF format, and online access to an information center built on the Eclipse help system framework. For Apache Derby, because you will have access to the documentation source files, you could use Apache Forrest to build and publish the documentation in HTML format to your own Web site or help system.

Keep in mind that because Apache Derby and IBM Cloudscape are the same product, any of the IBM Cloudscape add-ons mentioned in the previous list can be downloaded from IBM for use with Apache Derby.

Development of the Apache Derby Database—Who Can Contribute and How?

In this section, we will briefly discuss who can contribute to Apache Derby and how contributions are made. The goal of this section is not to make you an expert in how the Apache Software Foundation process works; rather, it will briefly describe how your work could end up in Apache Derby. For complete details on the concepts outlined in this section, see http://www.apache.org/foundation/how-it-works.

While you read this section, keep in mind that because Apache Derby and IBM Cloudscape share the same codebase, any changes to the host Apache Derby code are automatically reflected in the IBM Cloudscape binaries. The add-ons provided by IBM with IBM Cloudscape are proprietary, and their source code is not being made available; however, they can be freely used with Apache Derby. For example, IBM's Java Common Client is not open source code.

Because Apache Derby is an open source database project, its success relies on an ecosystem to enrich and mature the quality of its code base. The development of Apache Derby is peer-based, and these development peers come from an ecosystem that includes personnel from IBM, current Cloudscape customers, and a host of other developers who have previous Cloudscape experience or who choose to become experts in this field because of its obvious advantages. The introduction of new features into the base source code must strictly follow Apache guidelines.

Anyone can be included in the Apache mailing lists and submit code. These developers are known as Apache Derby *contributors*. The actual code for Apache Derby is maintained in a Subversion (SVN) repository that is restricted to a smaller set of *committers*. (You can learn more about SVN at http://subversion.tigris.org/.)

The Apache meritocracy determines which members of the Apache Derby community can become committers. This ascension path is the nucleus of the Apache organization and its projects. The Apache Software Foundation has five different levels, as shown in Figure 1-1.

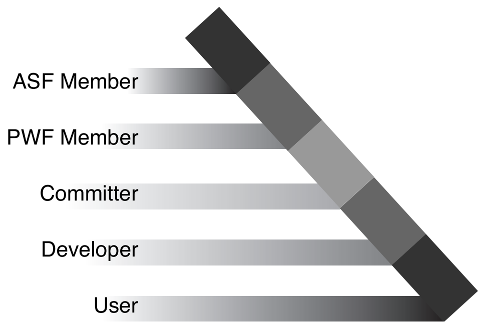


Figure 1-1 The Apache Software Foundation hierarchy

After contributors "earn their stripes" through meritable contributions and overall technical leadership for the direction of the source code, they can be promoted to committers. When a developer is considered a committer, he or she is granted access to the SVN repository. On their Web site, the Apache group calls this process *meritocracy;* literally, "govern of merit." You earn rights in accordance to your worth and efforts.

Committers get to vote on what contributed features make it into the build tree (again, the source code for the product is stored in the SVN repository). In this way, an experienced committee directs future innovations for the Apache Derby project (this is all part of the Apache process). Of course, because Apache Derby is open source, you are free to develop your own derivative works and use them for yourself, but we encourage all developers to become active in the development of Apache Derby—we are a community after all!

Each committee member can vote for a new feature, abstain from voting, or vote against a new feature for Apache Derby. Any negative votes must include an alternative proposal or a detailed explanation for the negative vote. (Bug fixes do not go through this process, but they do require review at check-in.) In essence, the community tries to come to agreement before moving forward with any changes. The Apache Software Foundation refers to this process as "consensus gathering," and it is a key construct in any Apache community.

IBM assigned six committers to the Apache Derby project when it released this product to the open source community. These committers are all steeped in IBM Cloudscape skills. In fact, they also have a deep understanding of other databases, such as DB2, Informix, Sybase, and SQL Server. Some of these committers are the original architects of the Cloudscape database. For Apache Derby to graduate from an Apache incubator project to a full-fledged Apache project (the project currently sits as an official Apache sub-project only one year before it is accepted as an incubator project), the committer ratio of IBMers to non-IBMers needs to "strike balance." Who will be among the committers that will join the Apache Derby project and help it become a full-fledged Apache project? It could be you!

How Can IBM Sell a Product for Profit and Contribute the Same Product to the Open Source Community?

Some members of the community might be skeptical about how IBM could possibly open source the code base for a product that is a source of revenue. Remember, if you choose to use IBM Cloudscape instead of Apache Derby, you are not paying for the right to use the database; you are only paying for a support contract (if you want one).

IBM has long been a strong contributor of open source software and open standards, especially in the areas of XML, Linux, Web Services, HTML, IP, HTTP, J2EE, true grid computing, and more. For example, IBM invented and donated the Distributed Relational Database Architecture (DRDA) standard for client/server communications.

The XML4J processor is another example of software that was developed in an IBM proprietary manner and later successfully open sourced. If you go to the Apache Web site, you will find the Xerces XML projects in the project list—those came from IBM. IBM actually helped start the Apache Software Foundation, so Apache Derby and Apache Xerces are great examples of the synergy that can be leveraged between open source and the for-profit business sector. This can work as long as the for-profit side of the equation is committed to open standards.

One of the best known contributions from IBM to the open source community is the Eclipse project (www.eclipse.org), a very successful open source integrated development environment (IDE) that was donated by IBM after they invested over 40 million dollars into it, and it not only lives on today, but has developed into a thriving community into the millions.

Other examples include IBM's extensive contributions to the Globus Alliance Grid specification (http://www.globus.org/ogsa/), the millions of dollars contributed toward the Linux movement, and the fact that IBM has led or co-led the creation of Web Service foundation technologies. IBM has also made significant contributions to Java; in fact, by contributing the IBM Cloudscape technology to the open source community as Apache Derby, IBM is enabling Java to expand into new market segments. This is not market posturing; this is real code that has changed (and is changing) the IT landscape forever.

The Apache Derby contribution represents over 500,000 lines of code, estimated to be worth over 85 million dollars!

So, in a nutshell, IBM is a significant force supporting open source innovation and collaboration. The company participated in more than 120 collaborative projects that have been contributed to the open source community. IBM contributed 500 patents into a "patent commons" for development and innovation. IBM has invested more than $1 billion in Linux development. IBM drives the adoption of and migrations to Linux among the development community, and expands enterprise-class tools for open source developers. So indeed, one can say that IBM is involved in open source!

How an Open Source Database Like Apache Derby Can Help

Several vendors are open sourcing their databases these days. Some have yet to publish their source code. The Apache Derby initiative is unique in that the entire source code is published. IBM believes that delivering this code to the Apache Derby community enables a richer and more robust feedback mechanism by which more functions, features, and benefits can be driven into the Apache Derby database.

Another unique and differentiating feature of Apache Derby is that this database was born out of a very successful proprietary initiative and consumer adoption lifecycle. Many features in this product resulted from specific business needs, and customers paid for the right to have these features. Since the Informix acquisition, IBM has continued to invest in this database, and it now powers much of the IBM software middleware stack. An open source Apache Derby database is interesting from both business and technical perspectives.

Open Source Software from a Business Perspective

From a business perspective, the Apache Derby database provides a way for individuals and companies to collaborate on projects that result in a greater sum than each could accomplish on their own: in short, it is synergistic.

It also flattens the "time to value" curve for projects by allocating fixed and variable costs across a greater sum of resources (most of which are not yours, which is a good thing from a cost allocation perspective). This effect drives down the marginal (per unit) cost of new features that are continuously driven into a product and would otherwise cost the recipient more money to obtain and leverage.

For example, consider maintenance programs such as Microsoft's Software Assurance (SA), which are used to deliver technology updates with new features to Microsoft software. Agreements like this come with their own usage terms but no promise with respect to the delivery of new features. However, new features are prepaid in advance, without credit. For example, SQL Server has a built-in five-year gap between SQL Server 2000 and SQL Server 2005. During this time, Microsoft did not release a significant amount of new functionality because of the business unit's release schedule, in spite of a user base that required a technology update. (They did deliver some new features. For example, their 64-bit engine was delivered outside of a scheduled release to remain competitive in the database space.) The point here isn't to besmirch Microsoft, but rather to point out that the delivery of new technology had nothing to do with technological innovation. In fact, some would argue that there has been limited technological innovation with this product (because there wasn't a release) in the last four years (until the release of SQL Server 2005)—yet not only was the right for these new features prepaid for, but there was also an opportunity cost of not having access to them.

On the other hand, the Apache Derby formula gives access to innovations from their earliest inception, as soon as they become part of the stable build stream. The point is that users of Apache Derby can take advantage of a new feature independently of its location in the product lifecycle, individual business unit goals, corporate funding, and so on. Rather, the decision to take advantage of a feature is based on need—early adopters will gravitate to new features earlier, whereas others might choose to wait until new features are more mature and tested—with a database like Apache Derby, you can do this.

Because this book is written with the Apache Derby application developer in mind, it is beyond our scope to discuss in detail the business benefits of the open source movement. There is a host of information on the Web about the business value of open source software. A good place to start is at www.ibm.com/developerworks/rational/library/1303.html.

Open Source Software from a Technical Perspective

From a technical perspective, because the source code for Apache Derby is readily and fully available to an "interested" and participatory community, the open source model has the advantage of transforming application developers into Apache Derby co-developers. Code bugs have a better chance of being detected, and the community serves as a quality assurance tier beyond the test buckets engineered by function verification testing (FVT) and system verification testing (SVT) units in the proprietary model. Beyond the mere detection of problem code, the fix for any issue can be delivered when the code is ready, outside of the complexities of a release cycle, costly hot fix schedules, or emergency rating systems.

The open source model also helps drive key technical features into the product more quickly because more developers are working on solutions to today's problems, and they get these improvements into the code at the same time (and not in correlation to profit center results). Quite simply, when software like the Apache Derby database is open source, it is free from business conditions and acts as a propellant for new features. In addition, the open source model enables end users to leverage features in a granulized and "as-needed" fashion (some applications will need urgent access to a performance feature, whereas others can wait), with full and open access to the build tree.

When discussing the benefits of open source software, we found the following quote quite interesting: "Source-level debugging saves valuable time throughout the embedded development process by eliminating the guesswork about how the code is interacting with the kernel or other low level operating system code." This quote (from Microsoft: http://msdn.microsoft.com/embedded/prevver/ce3/download/source/default.aspx) captures one of the key technical advantages of open source software. It's our point exactly.

The Apache Derby project delivers where other companies' marketing collateral tailors off.

Why the Need for a Local Data Store?

There are many reasons why more and more application developers and architects are turning to embeddable or lightweight databases. Before answering "why use a relational engine underneath an application?" which draws you to the Apache Derby platform, let's first discuss some of the benefits of just having a localized engine to store and manage your data.

Applications sometimes need to persist data, yet operate as an occasionally connected client (OCC). Such applications require intermittent access to the corporate network to refresh, upload, or pull data changes. For example, a traveling insurance agent might fill out reports throughout the day and want to manage the data locally until a time when the data can be uploaded to the corporate server. During the day, however, that user will need fast and efficient local access to data that pertains to his or her region.

Local data stores have become more popular recently for security reasons as well. Having a database available to an application, but not over a network, minimizes many security concerns. Today's economy is governed by more and more regulatory compliance rules concerning data persistence, such as the Health Insurance Portability and Accountability Act (HIPAA), Sarbanes-Oxley (SOX), the Basel II accord, and others. Another example of this is the challenge that many lines of business are facing with the new corporate reporting requirements in the United States. A lot of reporting is handled and stored in Microsoft Excel files, which might trigger compliance issues. Apache Derby can be used to provide a secure and robust repository for this data (alleviating the issues surrounding file repositories), while providing transparent access for end users who won't really know that the data isn't in a file.

In the past, data storage engines were under the complete control of IT departments, which had a number of implications. These engines had to be resourced from IT (which meant jumping through hoops to get a DBA's attention or resource); you consumed corporate CPU cycles, storage, and more. A self-managing and self-contained database such as Apache Derby has a "rounding error" of significance with respect to IT budgets and skill allocations, so you can avoid that lengthy process altogether.

Geographic isolation might also drive the requirement for a local data store. For example, many of today's cars have built-in computers that gather information that will subsequently be dumped to a certified mechanic's computer. These databases are being enhanced in many ways. Computers that collect driving characteristics data will run algorithms and queries that will adjust the performance characteristics of the car (suspension, torque, air/fuel ratios, and so on) to match the owner's driving patterns in real time.

Another example is Rolls-Royce. Although Rolls-Royce may be well known for its spectacular cars and artisanship (though the reality is that Rolls-Royce hasn't been making Rolls-Royces since the early 1980s), its real business is producing aero-engines, along with marine engines, power generators, nuclear submarine power plants, and so on. Rolls-Royce attaches drives to some of these engines that literally record gigabytes of information during operation. This data has to be stored somewhere. Sounds like a great opportunity for a database! Take for example an airplane whose engines record data through an entire flight pattern, which then is dumped for analysis to the safety engineer at the destination airport—it makes us feel safe when someone sending our plane on to the next destination knows how that engine performed that whole way!

Having data managed locally can increase performance, deliver better and more convenient access to vital data, and provide many other benefits that we will not cover here.

Estimates point to data collection rates of around 450 megabytes for every man, woman, and child per year, and in the three years prior to 2004, the world collected more data than it did in the last ten thousand years. There are a multitude of factors driving unprecedented data storage requirements, and these data requirements can only be handled by a relational database management system (RDBMS).

Why Use a Relational Database?

Now that it is apparent how applications and business processes can benefit from a functional place to persist and manage data, it is interesting to briefly comment on why a relational database is the best choice (as opposed to other approaches, like a file system, etc.).

There are many applications out there today that use some sort of local storage format to keep data collocated with the application, or to address any of the issues mentioned in the previous section. However, without a database, most of the proposed solutions cannot handle the atomicity of multiple changes (all or nothing, and the management of transactions that may impact each other), and there is no built-in recovery mechanisms in case of a failure during the transaction, no parallelism, no set-oriented data access API, no ability to share the same data at the same time between applications or users based on the business rules, and so on.

The most common format—perhaps due to legacy decisions and the lack of an embeddable, small-footprint database like Apache Derby in the past—is the flat file. Some databases even build on a flat file system, like FileMaker Pro. In the past, developers might have used a flat file system for their data storage requirements because it was perceived to be simple (for operational and management reasons, not from a coding standpoint) and fast. However, this architecture comes with its disadvantages: namely, it requires a lot of hand coding and is prone to errors.

Another common approach is XML; most programming languages implement standard programming interfaces for reading and writing XML (the XML specification itself is an open standard maintained by the World Wide Web Consortium). Like flat files, XML files can be easy to understand when you open them in an XML editor; however, as your data and the relationships between your data elements become more complex, designing a single XML file to store your data becomes a non-trivial exercise.

Storing data in XML files forces you to implement locking mechanisms to ensure that the files are always in a consistent state. In addition, as the amount of data you have to store in the file grows, your application will either require more system resources to store the entire XML file in memory, or have to reread and parse the XML file every time it has to retrieve data.

XML files are a great means of communicating between applications who don't know each other. In fact, XML is great for a lot of reasons. But as a data store, it requires a lot of effort for anything more than trivial data persistence scenarios. Of course advanced databases like DB2

UDB have the capability to provide XML repositories in combination with all the benefits that accrue from years of experience in the relational world.

One of the most significant benefits of using a database such as Apache Derby is that it delivers *atomicity, consistency, isolation, and durability (ACID)* compliance. Not all open source or proprietary databases do that. Databases exist for the purpose of providing a reliable and permanent storage mechanism that encompasses very strict properties embodied by these ACID characteristics. Although it is outside the scope of this chapter to discuss relational theory, you can learn more about ACID transactional processing and why it is so important from the "bible" of database theory: *An Introduction to Database Systems*, by C. J. Date (Addison-Wesley, 2004).

Another benefit of a relational data store is the concept of *relationships*. In fact, that is what relational databases (as the name would imply) are all about: set relationships. Whereas mathematicians can use algebra and set theory to no end for theoretical proofs, what a relational database delivers is a mechanism by which entities can be easily traversed and assembled, in any direction, with minimal coding effort. Need to know all the accounts with a last name that contains a specific group of characters? What about folks in the Eastern region? What happens to your application logic and pointer files when you move this data to another storage device in a flat file system? All this is transparently retrievable and navigable using a relational database, without code changes.

How is this possible? What is this 'magic' language by which you can interact with your data in an abstracted way without caring about its location? *Structured Query Language (SQL):* it can be used (in one form or another) to access all sorts of relational databases. SQL is a declarative language in that it does not contain any variables, loops, or other programming constructs (though there are procedural cousins that incorporate these entities). It is easy to learn, even for the non-programming kind.

Using a data store that can process SQL makes the data model more dynamic and flexible than it could ever hope to be when using a flat file system or some other approach. For example, if new information is deemed to be important, a query can be quickly written and implemented into the application. The methods by which the data needs to be accessed, or where it resides, are not issues. Typically, with a flat file system, this information was only well known to the original application developer; it's just not an issue with a relational database system. With a relational database system, access to the data and how to get to that data is abstracted from the developer.

With a flat file system, adding a new structure to the data model requires a new file (or the editing of an existing one), the registration of (or pointer updates to) the file, and so on. With an XML file system, you would have to modify your existing XML file structure and all of the methods that make assumptions based on the existing structure, or add another XML file to contain the new structure, and then implement a complex (and customized) join operation between the two XML files. Today, with Apache Derby relational database technology, it is as simple as a `CREATE TABLE` statement to add a new structure and a `SELECT` statement to return the new data.

The use of SQL also opens up a database to a wide variety of interfaces and access methods that would all have to be hand-coded when working with a flat file system. These interfaces mean a shortened and less expensive development cycle for applications, along with a readily available talent pool of developers and administrators.

Today's computers are orders of magnitude more *powerful* than those of the previous two generations. Many decisions were made to use flat files as opposed to a relational database some years ago due to the cost/power ratio for computers. When an Intel x486 processor with 64 MB of RAM was the most powerful (and expensive) computer around, you can see why economics would dictate the use of a flat file system. Today, computers and personal devices have moved far from the early adopters' phase of their respective product lifecycle curves such that they are capable of hosting a lot of data at relatively inexpensive costs.

Relational databases are very *scalable* as well. You can start with a database that resides on a tablet for a single user and move it to a large symmetric multiprocessor (SMP) machine that is used to support thousands of users and literally millions of transactions per minute.

Finally, the *manageability* of a relational database was thought to require an expert skill set, and thus it was deemed inappropriate for many applications. Databases like Apache Derby do not require users to perform table reorganizations or know anything about heap-related tuning parameters (if you do not know what these are, don't worry—that is the point).

How the Apache Derby Platform Can Help Your Business

There are many reasons why an Apache Derby platform decision can benefit your company, project, or infrastructure. In this section, we will give you some examples of how Apache Derby can be used in today's marketplace.

From a market perspective, not all client/server or Web applications require the muscle of an enterprise-class infrastructure database. In fact, 20–30% of applications might only need basic relational database capabilities, and their hosting environments might not have the system requirements to grab IT allocations (or attention for that matter) for these full-fledged data engines. However, they still require the scalability and robustness for data integrity that are associated with such databases, and Apache Derby gives them a way to do that.

In addition, as the amount of data being collected literally grows logarithmically and developers learn to leverage the benefits of a relational storage model, there are more and more demands for relational engines to be easy to use, easy to deploy, and embeddable. Developers will use databases, but they have overwhelmingly communicated that they are not interested in becoming DBAs.

Java has become a mainstay of the IT community; it is the write once, run anywhere imperative that helps to encourage platform independence and drives down the cost of computing. In fact, over 40% of all developers have Java-based skills. Coupled with other open source movements like Linux, the market screams for a true Java-oriented database from implementation through to the API. (Apache Derby can also support other programming languages and interfaces like .NET and ODBC, too, so it isn't just for Java developers.)

Finally, more than ever, the open source community is literally changing the landscape of information technology. Open source communities drive timely innovation into operating systems; Web servers can deliver this innovation in a rapidly adoptable model at minimal costs. As other proprietary software companies deliver an infrastructure under predatory pricing schemes, the market is turning to the open source community to “break the chains that bind.”

The Apache Derby database delivers the true database technology that the market has been asking for on many fronts. Apache Derby is a Java-based embeddable database. Because it is written in Java, it can be embedded within your Java applications and deployed anywhere. This means that moving your application from a mainframe to a Linux- or Windows-based server is as simple as deploying a JAR file: “just drop-and-go.” Further adding to its portability, Apache Derby uses an SQL92 and SQL99 entry level compliant SQL API (in fact, its SQL API is 100% compliant with the DB2 family).

There are no DBA requirements for managing Apache Derby. It could truly be hosted on those irritating late-night infomercials that claim to take the hassle out of various day-to-day chores. What little management there is (or that you can take advantage of for more granular tuning) can be controlled programmatically from the application itself. For example, a database backup can be executed through a native API call.

Finally, heavy investments in tooling create a feature-rich open source Eclipse environment for Apache Derby application development. This allows Java developers to remain productive in a comfortable open source environment, yet still leverage the benefits of a database.

Specifically, Apache Derby can be used for workloads such as:

- ***Small business client database applications:*** Many small businesses do not require a database that can support thousands of simultaneous users. For these businesses, Apache Derby may be a perfect fit, especially because it does not require a DBA—a luxury many small businesses cannot afford.
- ***Embedded applications:*** Because the database is only about 2 MB in size and is implemented as a Java .JAR file, it simply becomes a part of the application that is running it.
- ***Client database applications*:** Many client applications need the power and reliability of a fully transactional (ACID compliant) SQL database, without the unwanted overhead of a DBA. Java applications built on the Apache Derby platform will pretty much run on any platform that supports a JVM.
- ***Local registries and repositories:*** Application Servers use Apache Derby to store dynamic registries such as UDDI or meta-data stores. Because these databases are fully transactional engines, developers do not have to worry about data corruption and system crashes destroying their critical configuration data.
- ***Small business client/server and Web-based applications:*** Small businesses need Web sites that are low on maintenance and high on reliability, with plenty of head room for seasonal business spikes. Apache Derby delivers a robust performance engine suitable for many small business needs; for example, Web shopping carts, recommendation engines, and so on.
- ***Java development databases:*** Apache Derby makes it easy to put a relational database on every developer’s desktop. Apache Derby is just 2 MB, requires no DBA, and the support

for JDBC allows for an easy switch to an open standards enterprise database during the later stages of development and testing. This means that you can leverage this database for development at no cost and write applications to be deployed on more industrial-strength databases such as DB2, Informix, and so on. In the past, many shops addressed the development database requirement with a client/server database. However, this introduced the need for a DBA and opened up the possibility of one developer's "bad code" compromising the entire development team's productivity by accidentally erasing test databases or corrupting the database. It also forced system administrators to develop software deployment strategies and to obtain the bandwidth to support them. With Apache Derby, developers can simply copy the Apache Derby .JAR file to a local directory, set their `CLASSPATH` environment variable so that it points to the .JAR file, and they are ready for development. Even moving data around is a snap.

- ***Demonstration or proof-of-concept applications:*** Apache Derby is an ideal database to use for demonstrations of your software (when a database is required) and proofs of concept. Because of its small size, Apache Derby adds very little to the download size of your demo, and it can be invisibly embedded into your application. Your potential customers and evaluators can run your application just as they would against an enterprise database without having to download, configure, and run one. Think of all those applications that run on a multitude of databases but require a significant investment for a simple proof-of-concept or demonstration. Apache Derby can solve these problems.

A High-Level View of the Apache Derby Database

Apache Derby is a relational database management system that is written in Java. Java and the open source movement go hand-in-hand. Java is an attractive language in which to write such software because its "write once, run anywhere" mantra allows for seamless portability. People that like Java will love Apache Derby because the engine is very portable: again, from Mac to mainframe, and all parts in between. For example, Apache Derby is heavily leveraged on mainframes for applications that require a lightweight meta-data storage engine.

The moment many application developers or project managers hear the word "database," they start looking for a database administrator (DBA), and questions along the line of "how do I implement an incremental backup" and so forth arise. The Apache Derby database is not for DBAs (and they're likely not to like it very much because they'll be looking for the wrong things). In fact, Apache Derby is unique in that it does not require professional DBAs, or DBA skills for that matter, to run. It is truly a "lights-out" database management system. Think of Apache Derby as that "black box" in the corner. But don't let the "behind the scenes" characteristics of the Apache Derby platform fool you: there are customers today with Cloudscape databases that hold tens of gigabytes of data.

Apache Derby delivers an efficient "next to no compromise" database engine in a "fingerprint-sized" engine of only 2 MB. Along with its tiny size, it has generally minimized resource requirements too. For example, in some environments, it can run with as little as 4 MB of RAM on a J2SE/J2EE 1.3 or later platform. In addition, the number and size of tables are pretty much limited only by the disk space available to them. (The maximum number of indexes per table is 32,767, and the maximum number of columns in a table is 1,012.) This makes Apache Derby not only easy to deploy but also easy to embed within applications as well. In fact, it literally disappears into its hosting application.

Apache Derby packs a lot of punch in its tiny 2 MB allocation. It provides a fully relational database engine, including complete support for SQL92E and JDBC (2.0 and 3.0), and partial support for SQL99. It has many advanced features that experienced database developers may be accustomed to with larger databases, such as:

- Identity columns (sometimes referred to as auto-incremented columns) for key generation
- Fast query compilation through a cost-based optimizer that supports hash joins, sort avoidance, and row- or table-level locking based on a percentage of data selected
- Automated statistics collection, automatic table reorganization and space reclamation, programmatic backups (no DBA required), and more
- Binary large objects (BLOBs) for complex non-traditional data handling
- Referential constraints (primary and foreign key), general constraints (unique and check), and default values for business rules enforcement
- Transactional processing (it's ACID-compliant; see the "Why Use a Relational Database?" section for more details) with isolation levels such as repeatable-read and uncommitted-read
- Objects to encapsulate business logic and promote code reuse and best practices, such as stored procedures, user-defined functions (UDFs), and triggers
- Views
- Bulk load utility
- Scrollable cursors for more efficient result set processing
- Concurrent access by any number of program threads or processes
- Multi-user access
- JDBC (both an embedded driver and an external driver that's the same one used with DB2 UDB for z/OS and DB2 UDB for Linux, Unix, and Windows—called the Java Common Client), ODBC, CLI, .NET, PHP, and Perl connectivity (add-on interfaces are available from the IBM Cloudscape Web site). In fact, IBM contributed the Apache Derby Network Client during the writing of this book—learn more at: http://incubator.apache.org/derby/faq.html#netclient.
- Rich security features, such as signed Java Archive (.JAR) files, encryption of stored data (through the Java Cryptography Extension, or JCE, and other providers), ability to run with a Java 2 security manager enabled, optional LDAP authentication, and more

- Unlimited table size (though many operating system's 2 GB file limits could theoretically come into play because Cloudscape stores each table as a separate file)
- The IBM Integration Plug-in for Apache Derby; a free Eclipse-based plug-in to implement Apache Derby functionality as user interface component (http://www-106.ibm.com/developerworks/db2/library/techarticle/dm-0501cline/)
- National language support for program integrated information (for example, error messages, program output, and so on), and documentation for the following languages: Spanish (es), German (de_DE), French (fr), Italian (it), Brazilian Portuguese (pt_BR), Korean (ko_KR), Japanese (ja_JP), Traditional Chinese (zh_TW), and Simplified Chinese (zh_CN)

Officially, Apache Derby supports any standard JVM (J2SE 1.3 or higher). Whereas most applications detail operating system support prerequisites, the JVM is really the run-time platform for this database.

All Apache Derby components and future enhancements will adhere to the *Apache Derby Charter*. This charter is a statement of direction or purpose. It dictates a course of action that everyone involved in this community will adhere to in order to create the best possible underlying data store for their applications. Specifically, it decrees that any contributions to the Apache Derby code base must be completely written in Java, easy to use, have a small footprint, and be secure.

Details on SQL Support in Apache Derby and IBM Cloudscape

Apache Derby and IBM Cloudscape implement the SQL92E language standard and many features that are found in the SQL99 specification (with extensions for Java). Although not complete, the following list details most of the features that you're likely to leverage when building your own application:

- Basic database types, such as: `CHAR, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT`
- Datetime data types (from SQL92T): `DATE, TIME, TIMESTAMP` (with JDBC date/time escape syntax)
- Other types: `BIGINT, VARCHAR, CHAR FOR BIT DATA, VARCHAR FOR BIT DATA, LONG VARCHAR, LONG VARCHAR FOR BIT DATA, BLOB, CLOB`
- Basic operations: +,*,-,/,unary +,unary -
- Basic comparisons: <,>,<=,>=,<>,=
- Datetime literals
- Built-in functions: `ABS` or `ABSVAL`, `CAST`, `LENGTH`, `CONCATENATION` (||), `NULLIF` and `CASE` expressions, `CURRENT_DATE, CURRENT_ISOLATION, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER, DATE, DAY, HOUR, IDENTITY_VAL_LOCAL, LOCATE, LCASE or LOWER, LTRIM, MINUTE,`

MOD, MONTH, RTRIM, SECOND, SESSION_USER, SQRT, SUBSTR, TIME, TIMESTAMP, UCASE or UPPER, USER, YEAR

- Basic predicates: BETWEEN, LIKE, NULL
- Quantified predicates: IN, ALL, ANY or SOME, EXISTS
- CREATE and DROP SCHEMA
- CREATE and DROP TABLE
- Check constraints
- ALTER TABLE: ADD COLUMN and ADD or DROP CONSTRAINT
- CREATE and DROP VIEW
- Constraints: NOT NULL, UNIQUE, PRIMARY KEY, CHECK, FOREIGN KEY
- Cascade delete
- Column defaults
- Delimited identifiers
- Updatable cursors (through JDBC)
- Dynamic SQL (through JDBC)
- INSERT, UPDATE, and DELETE statements
- Positioned updates and deletes
- WHERE qualifications
- GROUP BY
- HAVING
- ORDER BY
- UNION and UNION ALL
- Subqueries as expressions (from SQL92F)
- Joins in the WHERE clause
- Joins (SQL92T): INNER, RIGHT OUTER, LEFT OUTER, named column join, conditional join
- Aggregate functions (with DISTINCT): AVG, COUNT, MAX, MIN, SUM
- SELECT *, SELECT table.* (SQL92T), SELECT DISTINCT, select expressions
- Named select columns
- SQLSTATE
- UNION in views (SQL92T)
- CAST (SQL92T)
- INSERT expressions (SQL92T): insert into T2 (COL) select col from T1

- `VALUES` expressions: select * from (values (1, 2)) as foo(x, y), and so on
- Triggers

The Apache Derby Components

Developers, line of business (LOB) managers, and IT professionals have chosen and will continue to choose the Apache Derby database in countless scenarios and for numerous vertical applications. Apache Derby can be entirely embedded within an application, used in a client/server architecture, or even embedded in an application server.

Three major components are shipped with Apache Derby: the Apache Derby database engine, the Apache Derby network server, and the Apache Derby `ij` JDBC scripting tool (there are some other tools, but this is the main one). As previously mentioned, the Apache Derby Network Client is a recent IBM contribution that will become the mainstay for many of your applications that leverage a traditional client/server environment. You can get it at: http://incubator.apache.org/derby/faq.html#netclient.

The Apache Derby Database Engine

The core of an Apache Derby solution lies within the *Apache Derby database engine*. As you have seen, the Apache Derby database engine is a full-featured, fully functional, embeddable relational database engine. Having been part of IBM for three years, the Apache Derby SQL dialect is a subset of that used for DB2. This means that Apache Derby applications are, for the most part, fully portable to DB2. In addition, an Apache Derby database can interact with the DB2 Everyplace SyncServer (both as a client and as a backend data store), which allows it to be part of a mobile solution that includes synchronization logic.

Apache Derby can be easily embedded within a Java application because it is implemented as a Java class library. For example, Apache Derby is usually delivered as part of an application's .JAR file in which the Apache Derby .JAR file has been included.

The database is started when the Java virtual machine (JVM) that hosts the application is started, and it is subsequently shut down when the JVM hosting the application is terminated. Because a JVM can be multi-threaded, any threads that are spawned within the JVM hosting the Apache Derby database engine can access the database. Concurrent access to this database, without the use of the Apache Derby network server, *must* be made from within the same JVM.

It is possible to have multiple JVMs running on a machine and even to have separate Apache Derby databases in each JVM. In this case, the environments are considered to be completely separate.

Apache Derby Network Server

The Apache Derby network server increases the reach of the Apache Derby database engine by providing traditional client/server functionality across JVMs, expanded data access interface options, as well as concurrent access between JVMs on the same machine. The Apache Derby network server allows clients to connect over TCP/IP.

When developers refer to Apache Derby as an embeddable database, they are referring to the fact that the Apache Derby database runs within a JVM process. Without the Apache Derby network server, there would be no networking services, data access outside of the embedded JDBC driver in the database engine, or other infrastructure requirements; this accounts for its small footprint.

Understanding what the embedded concept entails is critical when developing applications. For example, one common misconception that developers have when they work with Apache Derby as a standalone database is that it's only a single-user database and does not have communication capabilities. They believe that it is a single-user, single-connection, single-threaded system and develop their applications accordingly. This is not true. Apache Derby as a standalone database can support as many connections as desired, so long as they are established from the same JVM hosting the Apache Derby engine.

For an Apache Derby database to be accessed from a process that resides outside the hosting JVM that loaded the Apache Derby database initially (even if the JVM process resides on the same server), you need to load the Apache Derby network server. Read that last sentence twice to ensure you understand it because it is often a source of confusion for Apache Derby developers when multiple JVMs reside on the same machine. The Apache Derby network server allows for communications between JVM processes. This means that this communication infrastructure isn't solely required to communicate between machines; it is needed even if two different JVM processes reside *on the same machine* and want to talk to the same database.

Using the Apache Derby network server, you could have a number of Java programs or JVMs interacting with a number of Apache Derby databases (or a single database for that matter). The Apache Derby network server is also required to support interfaces other than JDBC, like ODBC or .NET. The Apache Derby database engine comes with an embedded JDBC driver (this is not the JCC driver that we talked about earlier)—so you will not need the network server for intra-JVM JDBC connections. The Apache Derby Network Client works with the Apache Derby Network Server.

ij—The Apache Derby JDBC Scripting Tool

The `ij` tool is a JDBC-based scripting tool that allows SQL scripts to be executed against an Apache Derby database. This is one of the few tools used with Apache Derby. We will use this tool throughout the book to create a database and some tables, populate those tables, and retrieve data from the Apache Derby database.

It is important to remember the role of the Apache Derby network server when using this tool because many application developers end up with a connection error when using it for the first time. For example, let's assume a developer has created an application and is invoking that application to test it. Starting the application starts a JVM where the application resides and where the Apache Derby database engine runs. When testing the application, the developer might want to add some data or work with the database schema, so what better tool with which to connect to the database than the `ij` tool? If you do not have the Apache Derby network server loaded, you are going to get an error because the `ij` tool is a Java tool that runs in its own JVM. For the most part, if you are a developer, we recommend that you include the Apache Derby network server in your environment. Otherwise, you will have to concern yourself with these sorts of issues, which can sneak up on you if your integrated development environment (IDE) makes multiple connections to the target database.

Developing Apache Derby Applications

Because Apache Derby is a Java-based database engine, most application developers will choose to program their applications using the JDBC API. Cloudscape supports Java applications written in JDBC 2.0 and 3.0.

An Apache Derby application follows the same general methodology as other Java-based applications in that a proper JDBC driver is loaded, a connection is made to the required database, a statement object is created, and finally the statement object is used to execute some business logic (generally an SQL statement or a stored procedure). Prepared statements are also supported in Apache Derby, as well as parameter markers, which deliver an excellent performance boost for your applications. In addition, the Apache Derby database leverages the Java just-in-time (JIT) compiler for better performance through native operational profiling.

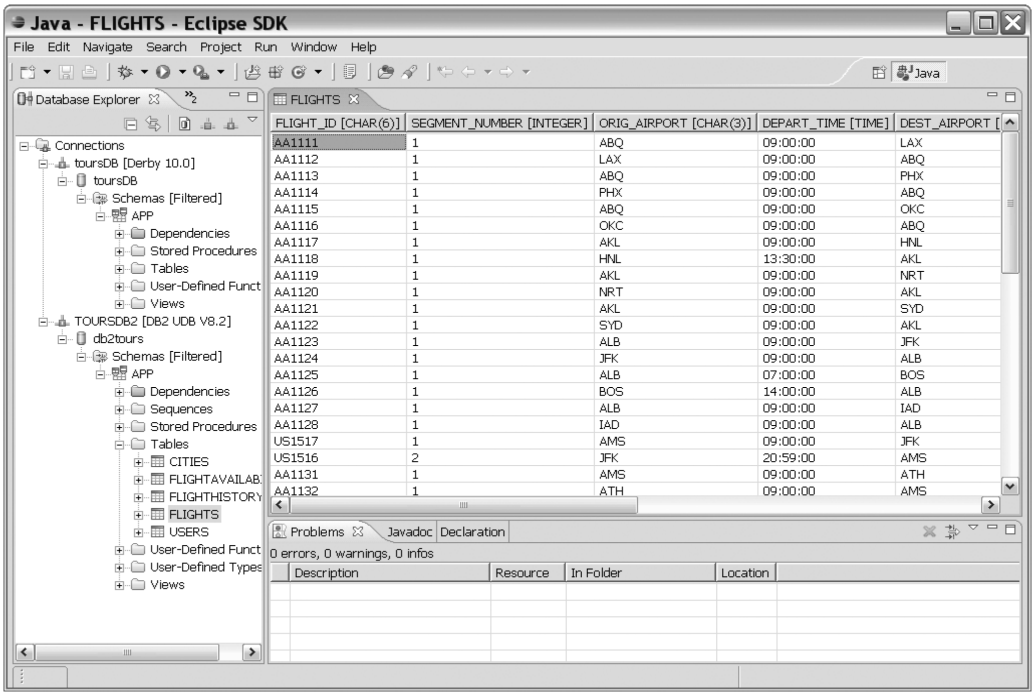
The only Apache Derby-specific code outlined in the previous paragraph is the loading of the JDBC driver and the uniform resource locator (URL) you would specify in the connection string. This means your Apache Derby applications can be easily altered to use other JDBC-compliant databases, such as DB2, Informix, and so on, by loading the appropriate driver class.

One very strong feature of Apache Derby is that the SQL API is 100% compatible with DB2, so if you ever decide to trade up for a full-fledged database, the migration to DB2 would be effortless. (There is also a free Eclipse-based migration toolkit for Apache Derby-to-DB2 migrations!) We bet you did not know that being an Apache Derby developer meant you were a DB2 developer too. Investing your skills in Apache Derby is definitely a resume-builder!

You can use almost any Java-based IDE to develop your Apache Derby-based applications. For example, you can use WebSphere Studio Application Developer (WSAD), Borland JBuilder, netBeans, and so on.

In keeping with the open source tradition, a plug-in is available for the Eclipse framework to equip application developers with a rich toolset for developing Apache Derby applications. Of course, you can also choose to write your applications the old-fashioned way, using your favorite editor such as `emacs` or `vi`—whatever suits your needs. When using a separate tool to help with application development, remember that you should probably have the Apache Derby Network Server installed as well.

The IBM DB2 plug-in for Eclipse lets you explore and create schemas for your DB2, Apache Derby, IBM Cloudscape, and Informix databases. An example of the Eclipse IDE with this plug-in is shown in Figure 1-2.


Java - FLIGHTS - Eclipse SDK

FLIGHT_ID [CHAR(6)]	SEGMENT_NUMBER [INTEGER]	ORIG_AIRPORT [CHAR(3)]	DEPART_TIME [TIME]	DEST_AIRPORT [
AA1111	1	ABQ	09:00:00	LAX
AA1112	1	LAX	09:00:00	ABQ
AA1113	1	ABQ	09:00:00	PHX
AA1114	1	PHX	09:00:00	ABQ
AA1115	1	ABQ	09:00:00	OKC
AA1116	1	OKC	09:00:00	ABQ
AA1117	1	AKL	09:00:00	HNL
AA1118	1	HNL	13:30:00	AKL
AA1119	1	AKL	09:00:00	NRT
AA1120	1	NRT	09:00:00	AKL
AA1121	1	AKL	09:00:00	SYD
AA1122	1	SYD	09:00:00	AKL
AA1123	1	ALB	09:00:00	JFK
AA1124	1	JFK	09:00:00	ALB
AA1125	1	ALB	07:00:00	BOS
AA1126	1	BOS	14:00:00	ALB
AA1127	1	ALB	09:00:00	IAD
AA1128	1	IAD	09:00:00	ALB
US1517	1	AMS	09:00:00	JFK
US1516	2	JFK	20:59:00	AMS
AA1131	1	AMS	09:00:00	ATH
AA1132	1	ATH	09:00:00	AMS

0 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Location

Figure 1-2 Using free open source Eclipse-based IDEs to work with Apache Derby databases

This plug-in also comes with a migration toolkit that allows you to migrate your Apache Derby schema to a DB2 database. Over time, expect to see more and more features driven into the plug-in and the Eclipse framework. (It is open source after all, so we are counting on some of you to contribute!)

Of course, there are other programming interfaces that you can use to develop applications and the tooling that goes along with it: for example, ODBC, CLI, .NET, Perl, and PHP will all be supported. You're going to learn a whole lot about application development with Apache Derby from this book.