



Introduction to Caché

Version 5.2
01 September 2006

Introduction to Caché

Caché Version 5.2 01 September 2006

Copyright © 2006 InterSystems Corporation.

All rights reserved.

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at www.w3c.org. The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.



The Caché product and its logos are registered trademarks of InterSystems Corporation.



The Ensemble product and its logos are registered trademarks of InterSystems Corporation.



The InterSystems name and logo are trademarks of InterSystems Corporation.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

Caché, InterSystems Caché, Caché SQL, Caché ObjectScript, Caché Object, Ensemble, InterSystems Ensemble, Ensemble Object, and Ensemble Production are trademarks of InterSystems Corporation. All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Customer Support

Tel: +1 617 621-0700

Fax: +1 617 374-9391

Email: support@InterSystems.com

Table of Contents

1 What is Caché?	1
1.1 A Unique Architecture	2
1.2 What Do You Mean Post-Relational?	3
1.3 Caché in Action	4
1.4 Contacting InterSystems	4
2 The Caché Database Engine	7
2.1 Transactional Multidimensional Storage	7
2.1.1 Objects and Multidimensional Storage	8
2.1.2 Flexibility	9
2.2 Process Management	9
2.3 Distributed Data Management	10
2.4 Journal Management	11
2.5 Lock Management	12
2.6 Device Management	12
2.7 Portability	12
2.8 Deployment Options	13
2.8.1 Basic Client/Server Configuration	13
2.8.2 Shadow Server Configuration	14
2.8.3 Multi-tier Configuration	14
3 Objects, SQL, and the Unified Data Architecture	17
3.1 Unified Data Dictionary	17
3.1.1 Flexible Storage	19
3.2 Objects	19
3.2.1 Defining Classes	20
3.3 SQL	21
3.3.1 The Object / Relational Connection	22
3.3.2 Inheritance and SQL	23
3.3.3 Object Extensions to SQL	24
4 Development Tools and Database Utilities	25
5 Connectivity	27

List of Figures

Storage of Persistent Objects	8
Storage of Indices	8
Process Management	9
Enterprise Cache Protocol	11
Client/Server Configuration	13
Shadow Server Configuration	14
Multi-tier Configuration	14
Unified Data Architecture	18

List of Tables

- Relational View of Object Features 22
- SQL View of the Person class: `SELECT * FROM Person` 23
- SQL View of the Employee class: `SELECT * FROM Employee` 24
- Revised SQL View of the Person class: `SELECT * FROM Person` 24

1

What is Caché?



Welcome to Caché, the post-relational database.

This introduction provides an overview of the major components and technologies that make up Caché. These components include:

- A powerful, multidimensional transaction engine that includes the ability to create distributed databases.
- A unified data architecture uniting the power of objects with high performance SQL.
- A suite of technologies and tools that provide rapid development for database and web applications.
- Native, object-based XML and Web Services support. For more information on Caché's XML features, refer to [Using XML with Caché](#).

- Automatic interoperability via Java, EJB, JDBC, ActiveX, .net, C++, ODBC, XML, SOAP, and more.

Please read the rest of this document to learn more about Caché. You may also find that the *Caché Technology Guide* provides a good overview of the features and architecture of Caché. For more details on a specific topic, refer to one of the other books available from the online documentation home page. In addition, Caché includes a number of online tutorials on various development and system administration topics.

1.1 A Unique Architecture

Caché derives much of its power from its unique architecture. At the core, the Caché Database Engine provides the complete set of services—including data storage, concurrency management, transactions, and process management—needed to build complex database management systems. You can think of the Caché engine as a powerful database toolkit. Using this toolkit, Caché implements a complete object and relational database management system.

The benefits of this architecture are manifold:

- The object and relational database systems talk directly to the database engine for extremely efficient operation; there is no object-relational middleware or SQL-to-object bridge technology.
- The logical separation of the database from its physical implementation makes it possible to radically reconfigure application deployments with no change to application logic.
- Because the Database Engine interface is open, you can make direct use of its features where needed. This can range from building your own customized database management system to adding targeted optimizations to performance critical applications.
- A platform for the future: The Caché architecture makes future Database Engine enhancements possible without impact on existing applications. For example, Caché v4.1 introduced a brand-new physical data structure, with dramatically improved scalability and performance, that not only required no change to existing applications but also required no change to the Caché object or relational systems. As new technologies emerge, such as XML, Caché can add support for them as native, high-performance components with little impact to existing applications.

1.2 What Do You Mean Post-Relational?

Caché is designed to transcend the limitations of the relational model while providing an evolutionary upgrade path for the thousands of existing relational database applications as well as support for the many SQL-based reporting tools on the market.

The “relational” part of “post-relational” refers to the fact that Caché is a full-featured relational database. All the data within a Caché database is available as true relational tables and can be queried and modified using standard SQL via ODBC, JDBC, or object methods. Because of the power of the underlying Caché database engine, we believe that Caché is the fastest, most reliable, and most scalable relational database available today.

The “post” part of “post-relational” refers to the fact that Caché offers a range of features that go beyond the limits of relational databases, while still supporting a standard relational view of data. These features include:

- The ability to model data as objects (each with an automatically created and synchronized native relational representation) while eliminating both the impedance mismatch between databases and object-oriented application environments as well as reducing the complexity of relational modeling.
- A simpler, object-based concurrency model.
- User-defined data types.
- The ability to take advantage of methods and inheritance, including polymorphism, within the database engine.
- Object-extensions for SQL to handle object identity and relationships.
- The ability to intermix SQL and object-based access within a single application, using each for what they are best suited.
- Control over the physical layout and clustering used to store data in order to ensure the maximum performance for applications.

While most databases with both object and relational access provide one form of access on top of the other, the SQL and object aspects of Caché both go directly to the data—this dual, direct access gives Caché its post-relational power.

1.3 Caché in Action

Caché is used around the world for a wide variety of applications ranging from single-user embedded systems to enterprise-wide multi-server installations with tens of thousands of concurrent users.

A small sample of applications built with Caché includes:

- As the application platform for a large health-care network running hundreds of patient-critical applications. The network includes a set of Caché systems acting as data and application servers and has over 30,000 client machines.
- As the data server for a Java-based enterprise messaging system for large financial institutions. Caché was chosen both for its performance and its ability to carry out customized tasks not possible within a traditional relational database.
- As an SQL-based OLTP (online transaction processing) system for a large government organization with over 1400 concurrent users. Caché was a drop-in (no application changes) replacement when other relational products failed to perform.
- As an object database and Web application framework for an online educational system used by a leading technical university. Caché was chosen for its rapid development (the application had to be built in three months), its object capabilities, as well as its ability to scale without application reworking.
- As an object database used to track real-time position and velocity of professional athletes during a world championship. Caché was chosen for its performance (compared with the leading object and relational databases) and its native C++ interface.
- As a distributed SQL data engine for a major Web site with millions of users. This site uses a set of cost-effective Linux-based servers and uses the Caché distributed data management to provide a scalable, personalized site with no middleware or Web caching infrastructure. The hardware costs of this system (four off-the-shelf Linux machines) were less than 10% of those quoted by a “leading database for internet applications” .

1.4 Contacting InterSystems

For support questions about any InterSystems products, please contact the InterSystems Worldwide Support Center:

- Telephone: +1 617 621-0700
- Fax: +1 617 374-9391
- Email: support@intersystems.com
- Web: <http://www.intersystems.com>

2

The Caché Database Engine

At the heart of Caché lies the Caché Database Engine. The database engine is highly optimized for performance, concurrency, scalability, and reliability. There is a high degree of platform-specific optimization to attain maximum performance on each supported platform.

Caché is a full-featured database system; it includes all the features needed for running mission-critical applications (including journaling, backup and recovery, and system administration tools). To help reduce operating costs, Caché is designed to require significantly less database administration than other database products. The majority of deployed Caché systems have no database administrators.

The following sections describe the major features of the database engine.

2.1 Transactional Multidimensional Storage

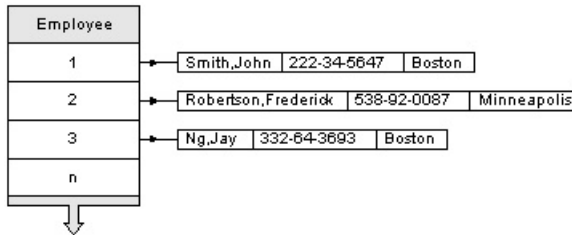
All data within Caché is stored within sparse, multidimensional arrays. Unlike the multidimensional arrays used by typical OLAP (online analytic processing) products, Caché supports transaction processing operations (inserts, updates, locking, transactions) within its multidimensional structures. Also, unlike most OLAP engines, these multidimensional structures are not limited in size to available memory. Instead, Caché includes a sophisticated, efficient data cache.

Because Caché data is of inherently variable length and is stored in sparse arrays, Caché often requires less than half the space needed by a relational database. In addition to reducing disk requirements, compact data storage enhances performance because more data can be read or written with a single I/O operation, and data can be cached more efficiently.

2.1.1 Objects and Multidimensional Storage

Caché Objects uses multidimensional storage as the foundation of its object persistence technology. For example, suppose you have a simple Employee class that represents employee data—say an employee's name, ID number, and location. Instances of Employee object could be stored in a multidimensional array like the one pictured here.

Storage of Persistent Objects



In this case, the array is subscripted by an object identifier value and the data for each instance is stored compactly within the nodes of the array. Caché Objects automatically creates the optimal storage structure for persistent classes.

If a cross-index is needed, say on the Location property, the persistence engine would use a different multidimensional array, similar to the one pictured below, to associate location values with corresponding object identifiers (typically a two-dimensional structure is used for this purpose).

Storage of Indices



Such indices are automatically maintained as changes are made to the database. The Caché SQL engine automatically uses such an index to satisfy queries for finding Employees by location.

2.1.2 Flexibility

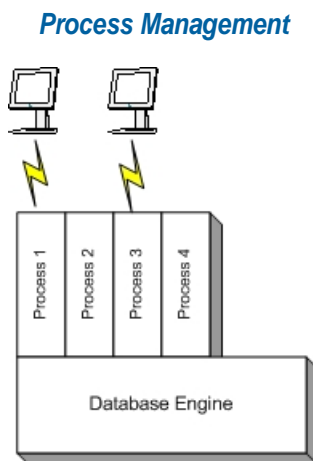
Multidimensional arrays give applications a great degree of flexibility in how they store their data. For example, a set of closely related objects, say an Invoice object and its corresponding LineItem objects, can easily be configured so that the LineItem objects are physically clustered with a Invoice object for highly efficient access.

Using a unique feature known as “subscriber mapping,” you can specify how the data within one or more arrays is mapped to a physical database file. Such mapping is a database administration task and requires no change to class/table definitions or application logic. Moreover, mapping can be done within a specific sparse array; you can map one range of values to one physical location while mapping another to another file, disk drive, or even to another database server. This makes it possible to reconfigure Caché applications (such as for scaling) with little effort.

The flexibility of transactional multidimensional storage gives Caché a significant advantage over the two-dimensional structure used by traditional relational databases: it is this flexibility that allows Caché to be a high-performance SQL, object, and XML database without compromise. It also means that Caché applications are better prepared for future changes in technology.

2.2 Process Management

Caché provides the ability to execute database operations as well as any degree of business logic within Caché processes.



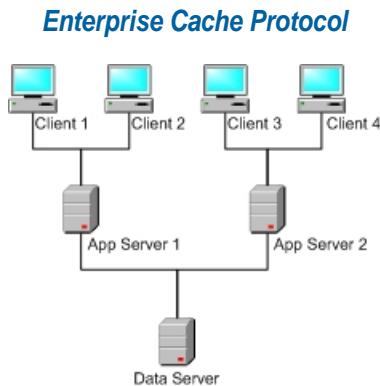
A process is an instance of a Caché virtual machine running on a Caché server. A typical Caché server can run thousands of simultaneous processes depending on hardware and operating system. Each process has direct, efficient access to the multidimensional storage system.

The Caché virtual machine executes instructions, referred to as “P-code” , that is highly optimized for the database, I/O, and logic operations typically required by transaction processing and data warehousing applications. Virtual machine code can be created in the following ways:

- **SQL**—SQL queries submitted to Caché are processed by the Caché SQL optimizer which, in turn, converts them to efficient, executable P-code (making use of any indices that may be present).
- **Object Behavior**—The Caché Objects technology provides a high degree of server-side object behavior (such as object persistence) by automatically generating executable P-code (using method generators—object methods that can generate code according to predetermined rules).
- **[Caché ObjectScript](#)**—Applications can include any needed logic within a Caché data or application server using the Caché ObjectScript scripting language. Such code can take the form of object methods (analogous but much more powerful than stored procedures within the relational world as they can make full use of object-oriented features) or complete “routines” (small programs that run within a Caché server).
- **[Caché Basic](#)**—Object methods can also be implemented using the Basic programming language. Caché includes a powerful, object-based version of the popular Basic programming language that is familiar to a large portion of the world's software developers. Caché's Basic runs on every supported platform and is completely interoperable with Caché ObjectScript.

2.3 Distributed Data Management

One of the most powerful features of Caché is its ability to link servers together to form a distributed data network. In such a network, machines that primarily serve data are known as Data Servers while those that mainly host processes, but little to no data, are known as Application Servers.



Servers can share data (as well as locks) using the Caché Enterprise Cache Protocol (ECP). ECP is effective because data is transported in packages. When information is requested across the network, the reply data package includes the desired data, and additional, related data as well. The natural data relationships inherent to objects and the Caché multidimensional data model make it possible to identify and include information that is related to the originally requested data. This “associated” information is cached locally either at the client or on the application server. Usually, subsequent requests for data can be satisfied from a local cache, thus avoiding additional trips across the network. If the client changes any data, only the updates are propagated back to the database server.

ECP makes it possible for applications to support a wide variety of runtime configurations including multi-tier and peer-to-peer.

2.4 Journal Management

To provide database integrity and reliability, Caché includes a number of journaling subsystems that keep track of physical and logical database updates. The journal management technology is also used to provide transaction support (a journal is used to perform transaction rollback operations) as well as database shadowing (a journal is used to synchronize a shadow server with a primary data server). As with the rest of the system, Caché lets you configure its journaling system to meet your application's needs.

2.5 Lock Management

To support concurrent database access, Caché includes a powerful Lock Management System.

In systems with thousands of users, reducing conflicts between competing processes is critical to providing high performance. One of the biggest conflicts is between transactions wishing to access the same data. Caché lock management offers the following features to alleviate such conflicts:

- **Atomic Operations**—To eliminate typical performance hot spots, Caché supports a number of atomic operations, that is with no need for application level locks. An example of this is the ability to atomically allocate unique values for object/row identity (a common bottleneck in relational applications).
- **Logical Locks**—Caché does not lock entire pages of data while performing updates. Because most transactions require frequent access or changes to small quantities of data, Caché supports granular logical locks that can be taken out on a per-object (row) basis.
- **Distributed Locks**—in distributed database configurations, Caché automatically supports distributed locks.

2.6 Device Management

Caché provides portable support for myriad devices (such as files, TCP/IP, printers) making it possible for Caché applications to interoperate with a host of other technologies. The interconnectivity options available with Caché (including CSP, ODBC, SOAP, and Java) are built on top of this underlying support.

2.7 Portability

Caché runs on, and is optimized for, a variety of hardware platforms and operating systems including Windows (98, NT, 2000, XP, etc.), OpenVMS, Linux, and every major version of UNIX.

You can easily port applications developed with Caché as well as data from one platform to another. This can be as easy as installing Caché on the new platform and moving the database

files to new system. When moving between some systems, you may need to run an in-place data conversion utility (to convert one endian representation to another).

2.8 Deployment Options

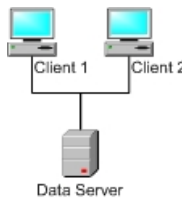
Caché supports a variety of different runtime configurations giving you maximum flexibility when you deploy your applications. You can switch between different deployment options by changing Caché system settings; typically there is no need to change your application logic.

Some basic deployment options are listed below.

2.8.1 Basic Client/Server Configuration

In the simplest client/server configuration, a single Caché data server services a number of clients (from one to many thousands, depending on the application and platform).

Client/Server Configuration



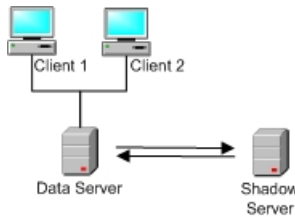
The client systems can be any of the following:

- Stand-alone desktop systems running a client application that connects via a client/server protocol (such as ODBC, ActiveX, JDBC, Java).
- Web server processes talking to Caché via [CSP](#) (Caché Server Pages), [SOAP](#), or some other connectivity option (such as ODBC, JDBC). Each Web server process may then service a number of browser-based or machine-to-machine sessions.
- Middleware processes (such as an Enterprise Java Bean application server) that connect to Caché via ODBC, JDBC, etc.
- Devices, such as terminals or lab equipment, that connect to Caché using one of many supported protocols (including TELNET and TCP/IP).
- Some combination of the above.

2.8.2 Shadow Server Configuration

The Shadow Server configuration builds upon the basic client/server setup by adding one or more shadow servers. Each shadow server synchronizes itself with the data within the main data server by connecting to and monitoring its transaction journal.

Shadow Server Configuration

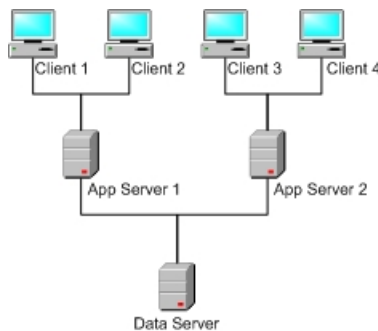


Shadow servers are typically used to service ad hoc queries, large reports, and batch processes to limit their impact on the main transaction system. They can also be used to provide fail-over systems.

2.8.3 Multi-tier Configuration

The multi-tier configuration uses the Caché distributed database technology—the Enterprise Cache Protocol (ECP)—to make it possible for a greater number of clients to connect to the system.

Multi-tier Configuration



In the simplest multi-tier setup, one or more Caché systems (acting as application servers) are placed between the central data server and the various client systems. In this case, the application servers do not store any data; instead they host processes that perform work for the client's benefit, off-loading the CPU of the data server. This type of configuration scales

best for applications that exhibit good “locality of reference;” that is, most transactions involve reasonably related data so that locking across application servers is limited. Such applications, as well as those with a fair amount of read access (like most typical Web applications), work extremely well in this model.

More complex configurations, with multiple data servers as well as data stored on application server machines, are also possible.

Typically applications use the multi-tier configuration for scaling as well as for providing high-availability (with applications servers serving as hot standby systems).

3

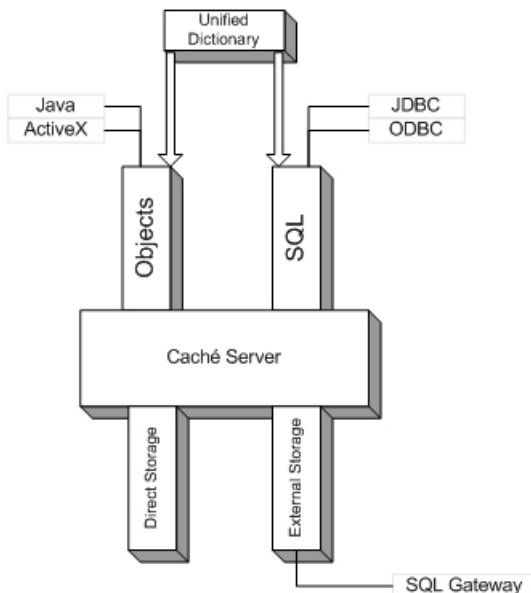
Objects, SQL, and the Unified Data Architecture

A powerful and unique feature of Caché is its unique Unified Data Architecture that provides simultaneous, high-performance object and relational access to data stored within Caché.

3.1 Unified Data Dictionary

Within Caché, you can model your application components as objects. Objects are organized by classes that define the object's data (properties) and behavior (methods).

Unified Data Architecture



The meta-information, or definition, of each class is stored within a common repository referred to as the Caché Class Dictionary. The Class Dictionary is itself an object database, stored within Caché, whose contents can be accessed using objects. The Class Dictionary, by means of a Class Compiler, defines the storage structure needed by persistent objects and converts class definitions into parallel sets of executable code that provide both object and relational access to this storage structure. By means of this architecture, the object and relational code paths are efficient and automatically synchronized with one another.

Class definitions can be added to the Class Dictionary in a number of ways:

- Interactively, using the [Caché Studio](#) development environment.
- Relationally, using DDL. Caché accepts standard SQL DDL statements and automatically creates corresponding class and table definitions.
- Textually, using XML. Caché supports an external, XML representation of class definitions. Typically this is used for source code management, deployment, automatic code generation, and interoperation with other tools.
- Programmatically, using objects. Using the Caché set of Class Definition objects, you can create programs that communicate directly with the Class Dictionary and create new classes at application runtime.

-
- Using an XML Schema Wizard, included within [Caché Studio](#), that can create class definitions from most XML schema files.

3.1.1 Flexible Storage

The Caché object model differs from those of programming languages in that in addition to properties and methods, you can specify storage-related behavior such as indices, constraints, and storage structure.

The storage structure used by persistent objects is independent of the logical definition of a class and is quite flexible: developers can use the default structures provided by the Class Compiler or they can tune the structures for specific cases. It is even possible, using the [Caché SQL Gateway](#), to have classes stored within external relational databases.

3.2 Objects

Caché includes a full-featured, next-generation object database specifically designed to meet the needs of complex, transaction-oriented applications. The Caché Object Model includes the following features:

- **Classes**—You can define classes that represent the state (data) and behavior (code) of your application components. Classes are used to create instances of objects as both runtime components and as items stored within the database.
- **Properties**—Classes can include properties, which specify the data associated with each object instance. Properties can be simple literals (such as strings or integers), user-defined types (defined using data type classes), complex (or embedded) objects, collections, or references to other objects.
- **Relationships**—Classes can define how instances of objects are related to one another. The system automatically provides navigational methods for relationships as well as referential integrity within the database.
- **Methods**—Classes can define behavior by means of methods: executable code associated with an object. Object methods run within a Caché server process (though they can be invoked from a remote client). Object methods can be scripted using Caché ObjectScript or SQL, or they can be generated using method generators, which are code that automatically creates customized methods according to user-defined rules.

- Object persistence—Persistent objects have the ability to automatically store and retrieve themselves to a database. The persistence support includes complete database functionality including automatic transaction management, concurrency control, index maintenance, and data validation. Persistent objects are automatically visible through SQL queries.
- Inheritance—By deriving new classes from existing ones, you can reuse previously written code as well as create specialized versions of classes.
- Polymorphism—Caché supports complete object polymorphism. This means that applications can use a well-defined interface (a set of methods and properties provided by a superclass) and the system will automatically invoke the correct interface implementation based on the type of each object. This makes it much easier to develop flexible database applications.
- Swizzling (also known as “lazy loading”)—Caché automatically swizzles (brings into memory from disk) any related persistent objects when they are referenced from other objects. This greatly simplifies working with complex data models.

3.2.1 Defining Classes

The simplest and most common way to define classes within Caché is to use the [Caché Studio](#) development environment. The Studio lets you define classes either using a simple text format within a syntax-coloring editor or by using a graphical point-and-click interface. These two views are interchangeable and are automatically synchronized.

Here is the definition of an extremely simple persistent object, Component, as seen within Caché Studio:

```
Class MyApp.Component Extends %Persistent [ClassType = persistent]
{
Property TheName As %String;
Property TheValue As %Integer;
}
```

This class is defined as a persistent class (that is, it can store itself within a database). In this case, the Caché-provided, %Persistent class (system class names start with a “%” character to distinguish them from application classes) provides all the needed persistence code via inheritance. The class belongs to the package, “MyApp”. Packages group related classes together and greatly simplify development of large applications. The class defines two properties: TheName, which has a string value, and TheValue, which has an integer value.

From within [Caché ObjectScript](#) code, such as within a method, you can use this object syntax to manipulate instances of Component object:

Using [Basic](#), you can define a method to manipulate instances of the Component object:

```
' Create a new component
component = New Component()
component.TheName = "Widget"
component.TheValue = 22

' Save the new Component to the database
component.%Save()
```

At this point a new instance of `Component` is stored within the database with a system-assigned unique object identifier. You can later retrieve this object by opening it (using its object identifier):

```
' Open an instance and double its value:
component = OpenId Component(id)

component.TheValue = component.TheValue * 2
component.%Save()
```

You can perform the exact same operations using native [Java](#), [C++](#), [ActiveX](#), or [.net](#) by means of the various Caché client bindings. The Class Compiler can generate, and synchronize, any additional code required to access objects externally. For example, if you are using Caché with Java, you can specify that the Class Compiler automatically generate and maintain Java proxy classes that provide remote access to persistent database classes. Within a Java program you can use this object naturally:

```
// Get an instance of Component from the database
component = (MyApp.Component)MyApp.Component._open(database, new Id(id));

// Inspect some properties of this object
System.out.println("Name: " + component.getName());
System.out.println("Value: " + component.getValue());
```

3.3 SQL

[Caché SQL](#) is a full-featured relational database engine that is fully integrated with the Caché object technology. In addition to standard SQL-92 features, Caché SQL offers:

- Support for streams (known in SQL as Binary Large Objects, or BLOBS).
- Support for stored procedures (implemented as object methods).
- A set of object-based extensions.
- User-definable data types.
- Support for Transactional Bitmap Indices.

Bitmap indices, typically used in large data warehousing and OLAP systems, offer the ability to perform high-speed searches based on complex combinations of conditions.

Such bitmap indices cannot be updated in real-time, however and are typically updated as a batch process. Caché SQL supports bitmap indices that offer high-performance searching power combined with no loss in insert/update performance. This gives transaction processing applications the ability to perform data warehouse-style queries and gives data warehouse applications the ability to perform real-time updates. For more information, refer to the [Caché SQL](#) documentation.

3.3.1 The Object / Relational Connection

All components within the Caché Dictionary are defined as classes. The Caché Class Compiler automatically projects persistent classes as relational tables. For every object feature, there is a corresponding relational equivalent, as illustrated in the following table:

Relational View of Object Features

Object Feature	Relational Equivalent
Package	Schema
Class	Table
Object instance	Row within a table
Property	Column
Relationship	Foreign key
Embedded object	Multiple columns
Method	Stored procedure
Index	Index

When Caché loads SQL DDL (Data Definition Language) statements, it uses the inverse of this projection to create classes that correspond to relational tables.

To demonstrate the object to relational projection, let's consider a simple example. Here is the definition of a simple, persistent Person class (part of a package called “MyApp”) containing two properties, Name and Home:

```
Class MyApp.Person Extends %Persistent [ClassType = persistent]
{
Property Name As %String(MAXLEN=100);
Property Home As Address;
}
```

The `Person` class gets its persistent behavior from the `%Persistent` superclass provided with Caché. The `Name` property is defined as a simple `String` of up to 100 characters.

The `Home` property illustrates the use of complex, user-defined data types, in this case the `Address` class, which is defined as:

```
Class MyApp.Address Extends %SerialObject [ClassType = serial]
{
Property City As %String;
Property State As %String;
}
```

The `Address` class is derived from the `%SerialObject` superclass. This class provides the ability to serialize itself (convert itself to a single-string representation) and embed itself within another containing class (as with the `Person` class).

When viewed via SQL, the `Person` class has the following structure:

SQL View of the `Person` class: `SELECT * FROM Person`

ID	Name	Home_City	Home_State
1	Smith,John	Cambridge	MA
2	Doe,Jane	Dallas	TX

Note that the object identifier is visible as a column. In addition, the fields of the embedded `Address` object are projected as separate fields. These fields are given the synthetic names `Home_City` and `Home_State` and behave exactly as if they were defined as two individual fields.

3.3.2 Inheritance and SQL

Inheritance is an important feature within object-based systems and is completely lacking within relational databases. Caché SQL makes it possible to use the power of inheritance using standard relational constructs. For example, we can derive a new `Employee` class from the `Person` class used in the previous example:

```
Class MyApp.Employee Extends Person [ClassType = persistent]
{
Property Salary As %Integer(MINVAL=0,MAXVAL=100000);
}
```

This new class extends the `Person` class by adding an additional property, `Salary`.

When viewed via SQL the `Employee` class has the following structure:

SQL View of the Employee class: *SELECT * FROM Employee*

ID	Name	Home_City	Home_State	Salary
3	Divad, Gino	Irvine	CA	22000

Notice that all of the inherited properties are available as columns. Also note that only rows that are actual instances of Employee are included. If we again ask for all Person instances:

Revised SQL View of the Person class: *SELECT * FROM Person*

ID	Name	Home_City	Home_State
1	Smith,John	Cambridge	MA
2	Doe,Jane	Dallas	TX
3	Divad, Gino	Irvine	CA

In this case, we see that all the rows are returned since every Employee is defined to be an instance of Person. In this case, however, only the properties defined by Person are displayed.

3.3.3 Object Extensions to SQL

To make it easier to use SQL within object applications, Caché includes a number of object extensions to SQL.


One of the most interesting of these extensions is ability to follow object references using the reference (“->”) operator. For example, suppose you have a Vendor class that refers to two other classes: Contact and Region. You can refer to properties of the related classes using the reference operator:

```
SELECT ID,Name,ContactInfo->Name
FROM Vendor
WHERE Vendor->Region->Name = 'Antarctica'
```

Of course, you can also express the same query using SQL JOIN syntax. The advantage of the reference operator syntax is that it is succinct and easy to understand at a glance.

4

Development Tools and Database Utilities

Caché includes a number of application development tools as well as database administration utilities. On a Windows system, these are accessible from the Caché cube icon  within the Windows task bar. (Right-click on the icon. If you do not see this icon, open the Windows **Start** menu, find the Caché folder under Programs, and click on the Caché entry.) The various graphical utilities are client-server applications whose clients run on Windows systems but can be used with remote Windows, Linux, UNIX, and OpenVMS server systems.

The Caché development tools and utilities include:

Caché Studio

[Caché Studio](#) is the primary development environment for Caché. The Studio lets you create class definitions, Caché Server Pages, and routines using a full-featured editor. Studio includes sophisticated syntax checking and coloring, graphical debugging, and a point-and-click class inspector. Caché Studio works directly with the Caché database (it is a Caché application) and offers multi-developer support as well as source control system integration. You can customize Studio through the use of Studio templates.

Caché System Management Portal

The System Management portal provides a Web-based interface for managing a Caché site. The management portal includes tools for system administrators, security managers, database operators, and any others needing access to Caché. Its administrative features allow you to set up a Caché system, view and alter its configuration

parameters, adjust system settings, and create and edit databases and namespaces. Its database-browsing features include the ability to browse the contents of a Caché database, to examine data (globals) in the data engine; and to browse classes and routines — as well as monitoring database activity and performing backup operations. Its security-related features allow you to add, alter, and remove users, roles, resources, privileges, and to perform other security management tasks. Its SQL-related features provide a graphical, SQL-based view of a Caché database; you can use it to manage SQL roles and permissions, browse table and view definitions, execute ad hoc SQL queries, manage the query cache, and import and export data.

Caché Terminal

Caché Terminal provides an interactive, command-line interface to Caché. You can use Terminal for direct interactions with the Caché database engine. The Terminal is an important tool for testing and troubleshooting applications.

Caché DocBook

[Caché DocBook](#) provides the documentation for Caché as a Web-based, fully-searchable, database application.

5

Connectivity

Caché includes a variety of technologies for efficiently and easily connecting and interoperating with most every other software architecture:

Caché Server Pages

Caché Server Pages (CSP) encompasses a wealth of technologies that offer the ability to quickly build high performance, scalable database applications for the Web. It works with all the major Web servers on a number of platforms. For details, see [Using Caché Server Pages \(CSP\)](#).

Caché ODBC

Caché includes a high performance, native ODBC driver to provide access to relational applications and tools. The Caché ODBC driver is available on a number of UNIX platforms as well as Windows.

Caché JDBC

Caché includes the world's highest performance, native JDBC driver to provide access to relational Java applications and tools.

Caché XML and Web Services

Caché includes a number of ways to interoperate with XML and SOAP. For details, see [Using XML with Caché](#) as well as [Using SOAP and Web Services with Caché](#).

Caché Object Server for Java and EJB

The Caché Java Binding makes persistent Caché classes available for use within Java and EJB (Enterprise Java Bean) applications. For details, see [Using Java with Caché](#).

Caché Object Server for ActiveX

The Caché Object Server for ActiveX makes persistent Caché classes available for use within ActiveX environments such as Visual Basic or .net. For details, see [Using ActiveX with Caché](#).

Caché Object Server for C++

The Caché C++ binding makes persistent Caché classes available for use within C++ applications. For details, see [Using C++ with Caché](#).

Caché SQL Gateway

The Caché SQL Gateway gives Caché applications object access to third party relational databases. For details, see [Using the Caché SQL Gateway](#).

Caché ActiveX Gateway

The Caché ActiveX Gateway gives Caché applications direct access to ActiveX/COM/.net components. For details, see the [Using the Caché ActiveX Gateway](#).