Understanding the Enterprise Information System Technology of BaanERP*

Gerd Wagner

http://www.inf.fu-berlin.de/~wagnerg
Institut für Informatik, Freie Universität Berlin
Takustr. 9, D-14195 Berlin, Germany
E-mail: gw@inf.fu-berlin.de

April 25, 2000

*Tutorial at the Fourth IEEE International Baltic Workshop on Databases and Information Systems, May 2000, Vilnius (Lithuania).

Contents

List of Figures

List of Tables

1. Introduction

Enterprise resource planning (ERP) systems are generic and comprehensive business software systems based on a distributed computing platform including one or more database management systems. They combine a global enterprise information system covering large parts of the information needs of an enterprise with a large number of application programs implementing all kinds of business processes that are vital for the operation of an enterprise. These systems help organizations to deal with basic business functions such as purchase/sales/inventory ('distribution') management, financial accounting and controlling, and human resources management, as well as with advanced business functions such as project management, production planning, supply chain management, and sales force automation.

First generation ERP systems now run the complete back office functions of the worlds largest corporations. The ERP market rose at 50% per year to $8.6 billion in 1998 with 22,000 installations of the market leader, SAP R/3. The benefits of a properly implemented ERP system can be significant.

Typically, ERP systems run in a three-tier client/server architecture. They provide multi-instance database management as well as configuration and version (or 'customization') management for the underlying database schema, the user interface, and the numerous application programs associated with them. Since ERP systems are designed for multinational companies, they have to support multiple languages and currencies as well as country-specific business practices. The sheer size and the tremendous complexity of these systems make them difficult to deploy and maintain. Despite the worldwide success of systems like SAP R/3 and BaanERP, the underlying architectures, data models, transaction mechanisms and programming techniques are to a large degree unknown to computer scientists.

The goal of this tutorial is to present the information technology of BaanERP, as a representative of the ERP system paradigm, from a computer science (rather than from a business management) perspective, relating it to established database and distributed systems concepts and techniques. A critical assessment of BaanERP will point out some of its merits and weaknesses. The tutorial will help attendees to understand the potential of ERP system technology in general, and of Baan ERP system technology in particular, and how it relates to their own research and development work

2. Overview

Baan—The Company

Founded:	1978
Head quarter:	Barneveld (The Netherlands) and Herndon (Virginia, USA)
Customers:	13,000 customer sites worldwide (biggest customer: *Boing*)
Employees:	approx. 4,700
Revenues 1998:	736 Mio USD

Terminology

In Table 2.1, we list the most important terms where the Baan jargon departs from usual terminology.

Baan Jargon	*Meaning*
'Company'	Client (having its own set of tables)
'Session'	Application program with associated screen form
'Package'	application program package
'Package VRC'	package version
'Package Combination'	configures a specific version of the BaanERP database schema together with a specific collection of (application program) package versions ('Baan environment')

Table 2.1.: Baan-Jargon.

Customization/Programming with BaanERP Tools

BaanERP Tools consist of three parts:

1. **Application Administration**: users, clients, database management, SQL queries, etc.
2. **User interface customization**: version management, menus, forms, reports, sessions, etc.
3. **Programming**

Some application administration functions, such as *Software Installation*, *Device Management* or *Audit Management*, are only of interest to Baan system administrators.

Customization/Programming with BaanERP Tools

BaanERP Tools consist of three parts:

1. **Application Administration**: users, clients, database management, SQL queries, etc.
2. **User interface customization**: version management, menus, forms, reports, sessions, etc.
3. **Programming**

Some application administration functions, such as *Software Installation*, *Device Management* or *Audit Management*, are only of interest to Baan system administrators.

Application Administration

The following application administration functions are of general interest:

Maintain Companies 'Companies' (clients) are defined by company number, name, currency, and a *package combination* that associates the corresponding database schema with the company.

User Management The name of a Baan user is normally the same as the system login name. To each user, a start menu, a program package configuration ('package combination'), and a client is assigned. The client determines the data set on which the user will work. See Figure 2.1.

Text Management Internal texts that have to be edited with the built-in (rather old fashioned) text editor, and that are being stored line by line (and optionally by language), are used for delivery conditions, item descriptions, invoice footers, and the like.

Job Management A job consists of a configurable sequence of print and processing programs which are being executed periodically (for instance, every two hours).

Database Management comprises the following functions:

- separating the data into (normally) two databases: one for the 'critical' tables (such as operating parameters and authorizations) for which all user accesses are written in a security log, and one for the remaining tables;
- exporting, importing, deleting, checking and reorganizing tables (under *Miscellaneous*);

- inspecting and possibly modifying the data of certain tables (under *General Table Maintenance*).

SQL Queries A database query can be created interactively by using forms or by directly entering SQL SELECT statements in the Baan text editor. The resulting answer set is displayed on screen or printed on paper by means of a Baan report.

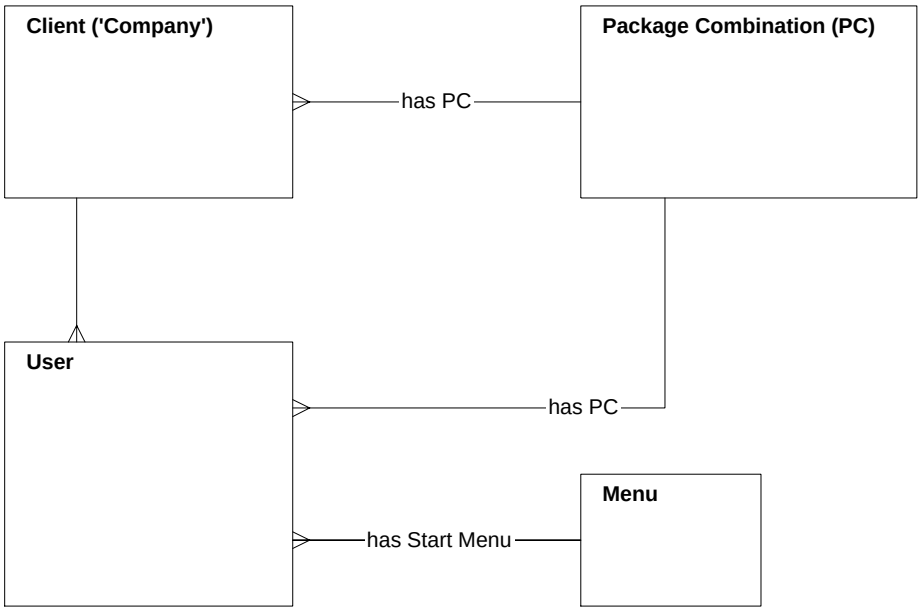

Figure 2.1.: The default client of a user must have the same package combination as this user.

Customizing the User Interface

Various components of the user interface can be created and modified interactively without programming. This includes:

Menus consist of a list of choice options leading to application programs or to submenus.

Labels are named short texts used to label form fields and report columns.

Reports are defined by a number of layout elements and their data fields and labels. They are used to create complex documents like bills of lading or invoices, and also for displaying or printing the results of SQL queries.

Forms consist of form fields for displaying and allowing to modify data, and of pull-down menus and push buttons to execute actions and call application programs.

Messages and Questions are named short texts used to display messages or ask questions during the execution of an application program.

Programming

BaanERP programming consists of version management and of the creation and maintenance of program scripts. Version management includes:

1. the creation and maintenance of package versions (PVRCs) and
2. the configuration of a set of package versions in a package combination.

The real programming refers to *Program Scripts*, include modules (called '*Functions*') and DLL modules (*Libraries*):

Program Scripts are being written in a Pascal-like procedural programming language, called *'Baan 3GL'*. For event-based programming in connection with forms, there is a *'4GL'* extension by the addition of special key words in order to refer to form and database events.

'Functions' are in fact *include modules* allowing to re-use variable declarations, functions and procedures.

Libraries allow to maintain re-usable function and procedure code. As opposed to include modules ('Functions'), DLL library functions do not blow up the source code generated before compilation and the resulting object code.

Limits

Application Server ('Bshell')

- maximum string buffer size is 4K
- maximum function stack depth is 75

Tables

- maximum record length is 3072 bytes
- maximum number of fields is 1024
- maximum field length is 3072 bytes

Indexes

- maximum index length is 120 bytes
- maximum number of fields is 32 (no combined fields)

3. System Architecture

BaanERP supports a three-tier architecture consisting of a user interface tier, an application tier, and a database tier. The user interface tier provides presentation and input services for user interaction. The application tier consists of the BaanERP application server and the application programs. The database tier includes the BaanERP database driver and a third party RDBMS product that acts as the database server. Figure 1 depicts the BaanERP architecture. The emphasis of this document is the BaanERP database driver. The database driver is the interface between the BaanERP applications and the RDBMS server. The database driver translates database requests from the BaanERP application server to RDBMS specific SQL requests that it sends to the database server. After the database server retrieves the requested information, the database driver then passes the data back to the BaanERP application server.

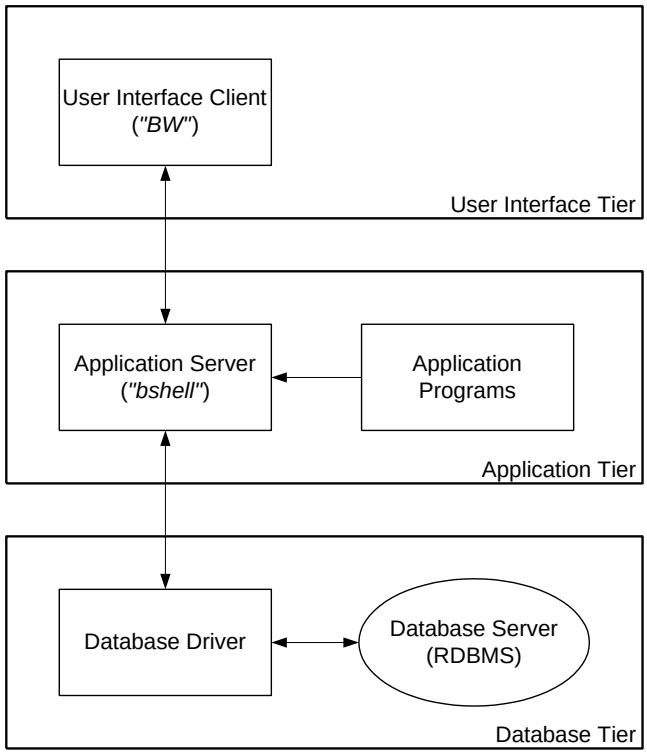

Figure 3.1.: BaanERP three-tier architecture.

3.1. User Interface Tier

The user interface tier consists of the BaanERP user interface for Microsoft Windows (called 'BW') and for Internet browsers (called 'BI'). Data input from the user through BW or BI is relayed to the BaanERP application server; data returned from the BaanERP application server is displayed to the user in graphical form by the user interface.

3.2. Application Tier

The application tier includes both the application programs and the BaanERP application server. Together, the application programs and the application server provide much of the functionality of BaanERP. The application programs include the compiled BaanERP applications and the data dictionary. The BaanERP applications are written in the Baan 4GL programming language with embedded SQL using the development environment provided by the BaanERP Tools package.

The BaanERP application server schedules and runs the application programs, sends and receives information to and from the user interface server, and initiates an instance of the database driver as necessary for communication with the database server. A running database driver can support multiple connections to a single RDBMS instance. If a BaanERP installation stores data tables in multiple databases, the application server must start one instance of the database driver for each database with which it must communicate. The BaanERP application server has traditionally been called 'Baan shell' or simply 'bshell'. Throughout the remainder of this document, it is referred to as the BaanERP application server or the application server.

3.3. Database Tier

The database tier consists of the BaanERP database driver and the database server. The database driver provides a common interface between the BaanERP application server and the database server. Communication between the application server and the database driver is the same, no matter which RDBMS product is used as the database server. There is one database driver for each of the RDBMS products that BaanERP supports. Communication between the database driver and the database server is tailored to the RDBMS being used. The database driver communicates with the RDBMS through structured query language (SQL) statements and the native application programming interface (API) of the RDBMS.

The database server consists of one of five third party RDBMS products: Oracle, Informix, Sybase, DB2, or Microsoft SQL Server. All BaanERP application data is stored in a relational database managed by an RDBMS. It is possible to have multiple RDBMS products in one BaanERP installation, with some data residing in one database server and other data residing in another.

3.4. Data Flow through the BaanERP Architecture

Note that the database driver provides an interface between the BaanERP application server and the specific RDBMS server being used. The flow of data through the system is described below.

When a user performs an operation at a GUI workstation, the user interface server interprets the input and sends the information to the BaanERP application virtual machine. Based on the information it receives, the application server causes the appropriate application object to be executed.

When a running application object requires information that is stored in the database, the application server sends the request to the database driver. Data requests from the client applications are RDBMS independent and are made using BaanERP SQL, an RDBMS independent SQL language.

When the application server executes a database query from an application program, it first determines whether or not there is a running database driver available to process the query. If there is no database driver running, or if the running database driver instances are communicating with a database server other than the one storing the needed data, the application server starts a new instance of the database driver. The application server parses the BaanERP SQL database query it receives from the application object and sends an internal representation of the query to the database driver. The internal representation of the query that the database driver receives is still RDBMS independent.

The database driver translates the database query into an appropriate query using SQL statements compatible with the specific RDBMS being used. Each database driver takes advantage of the design of the particular RDBMS that it supports so that the resulting SQL statements are valid for the RDBMS and provide the best possible performance. The RDBMS specific SQL statements are then submitted to the RDBMS server, which processes the data request. When the RDBMS has processed the query, it returns the data to the database driver. Any error conditions are caught and handled by the database driver. The database driver then returns the data and status information to the application server, where it provides the information to the application that requested it. The application server may also send a message to the user interface server, which displays an appropriate message on the users workstation.

3.5. BaanERP Hardware Configurations

Several hardware configurations are supported for a BaanERP implementation. These configurations include standalone mode and many variations of client/server mode. Available hardware, data storage requirements, and performance expectations determine the most appropriate hardware configuration.

Standalone mode refers to a configuration where all components of the BaanERP architecture run on a single machine. In standalone mode, an end user can work from the host machine or from a thin client machine, such as an X-Terminal running BI. The

standalone mode configuration is illustrated in Figure 2.

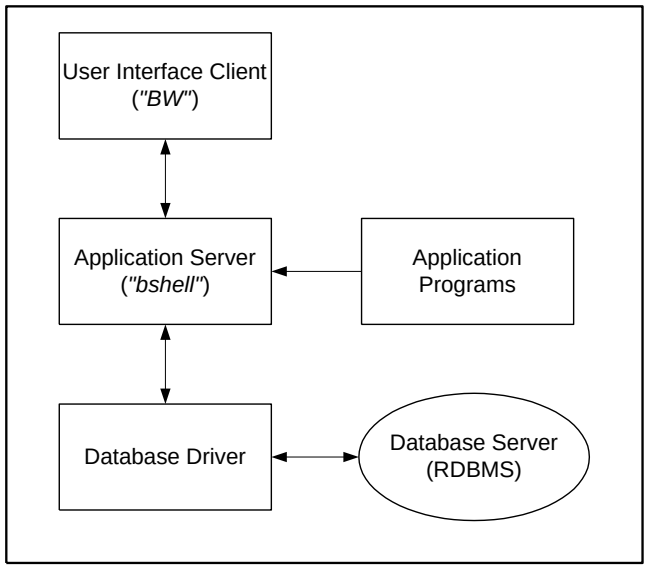

Figure 3.2.: Standalone mode configuration.

In a client/server configuration, the components of the BaanERP architecture are distributed over two or more machines. There are many client/server configurations. The most common configurations are described here. The simplest client/server configuration is sometimes thought of as a variation of standalone mode. In this configuration, the application tier, database driver and RDBMS are on one machine, while the display drivers are distributed among the user workstations. An instance of the application server and at least one instance of the database driver is started for each user. All users have access to the same application programs and database servers. This configuration is illustrated in Figure 3.3.

When two machines are available to be used as servers, the application tier is placed on one server, while the database driver and the database server are placed on another. As with the previous configuration, an instance of the application server and at least one instance of the database driver is started for each user. All users have access to the same application programs and database servers. This client/server configuration is illustrated in Figure 3.4. This configuration uses the BaanERP method of client/server access between the application server and the database server.

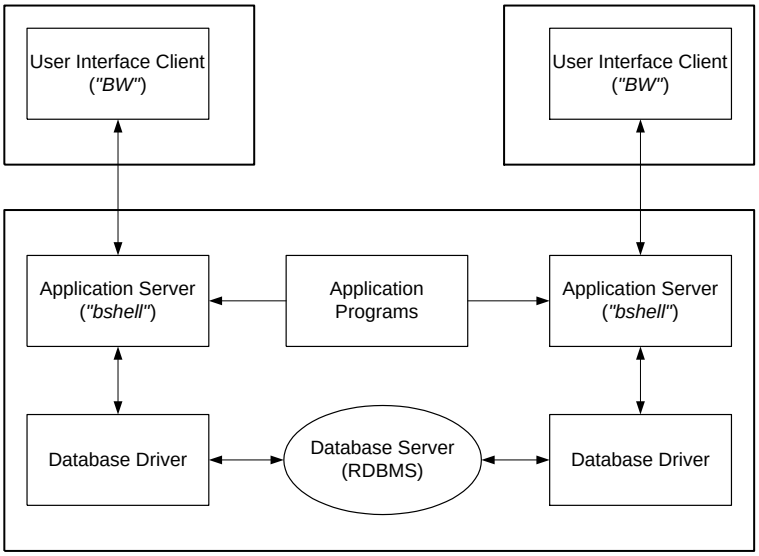

Figure 3.3.: Application and Database Server on the same computer.

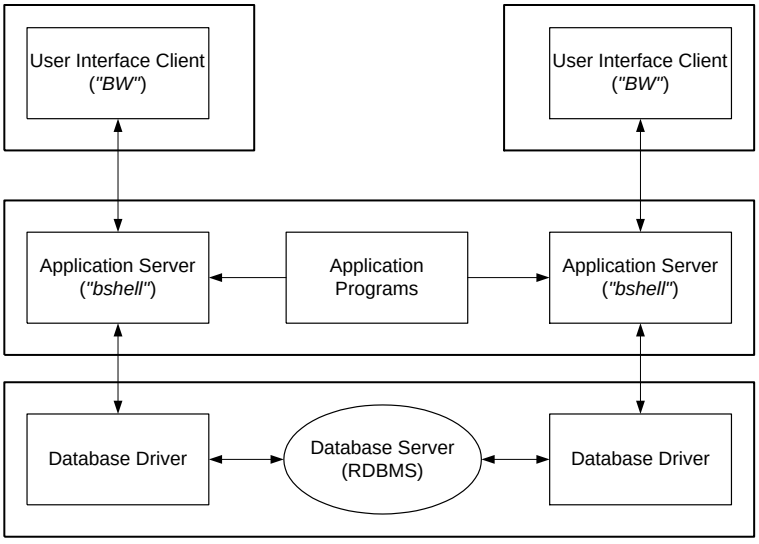

Figure 3.4.: Application and Database Server on different computers.

4. Software Architecture

Both tables as well as application programs ('sessions') and their software components are assigned to *modules* grouping related business functions.

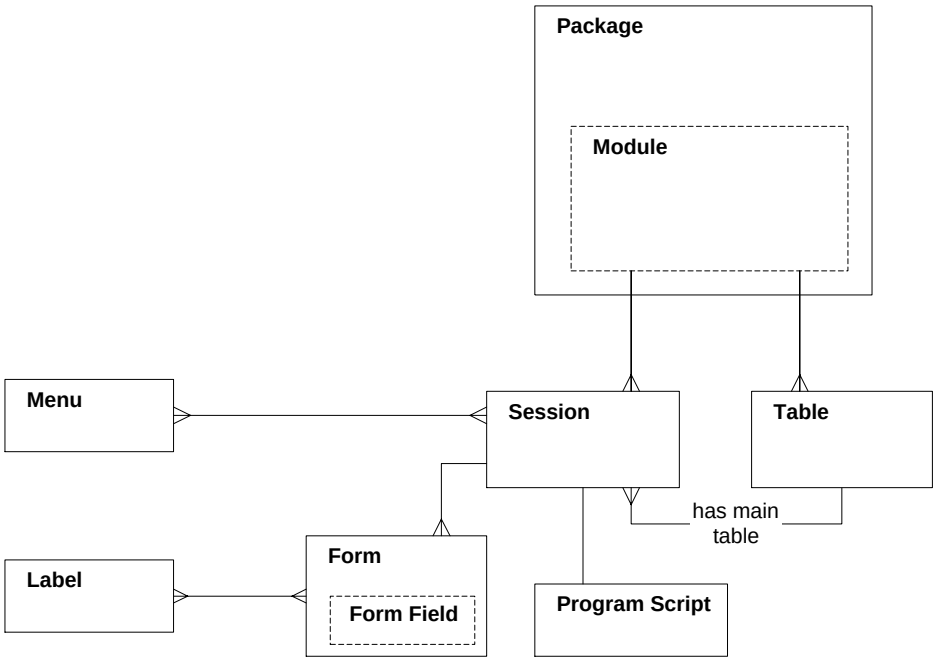

Figure 4.1.: Packages, modules, sessions.

4.1. Packages

Business areas such as distribution, manufacturing, and finance are mapped into **program packages** that have a two character identification code, as listed in Table 4.1. The most important ones are: tc/Common, td/Distribution, ti/Manufacturing, and tf/Finance.

cp	Constraint Planning	
ct	Controlling	
ps	Process	*process-based production*
tc	**Common**	*master data*
td	**Distribution**	*purchase/sales/inventory control*
ti	**Manufacturing**	
tf	**Finance**	*financial accounting*
tp	Project	*project management*
tr	Transport	*transport industry*
ts	Service	*service industry*
tt	**Tools**	*Baan administration and programming*
tu	Utilities	*data exchange interfaces, etc.*

Table 4.1.: BaanERP packages. The most important ones are emphasized.

4.2. Modules

Business functions are grouped in **modules** with a three-character code. For instance, the package *Distribution (td)* consists of the modules

Distribution:Purchase	td pur
Distribution:Sales	td sls
Distribution:Inventory	td inv

4.3. Sessions

All business processes of a business function are realized by means of **'Sessions'**. A BaanERP session is an application program that is associated with a screen *form* and normally with a base *table*, and possibly with a *report*. A session ID is formed using the package/module IDs as a prefix. For instance, the session ID "ti itm 0101m000" displays that the session belongs to the module *Item Control (itm)* in the package *Manufacturing (ti)*.

Sessions are being called interactively via menu options by the user. There are sessions for master data management, like *Maintain Items* (tiitm0101m000), or for executing certain business processes, such as *Update Cost Prices* (ticpr2220m000).

Sessions can also be defined as subprograms ("subsessions") that can be programmatically invoked from other sessions. Instead of an "m" (for 'main') the session ID contains an "s", like the subsession *Update Cost Prices* (ticpr2220s000).

5. Version Management

Complex application software systems, like ERP systems, are subject to **continuous change**. Both the discovery of programming errors and new technical as well as legal requirements call for modifications of the ERP system software. In addition, various parts of the standard software have to be customized for supporting country-, branch- and company-specific business processes. In order to be able to perform all these changes in a systematic way, different versions of software components (notably tables and sessions) can be maintained and configured into **customized versions** of a BaanERP system. Therefore, all components exist on a **logical naming level** and on the **level of special instances** or versions, like shown in Figure 5.1.

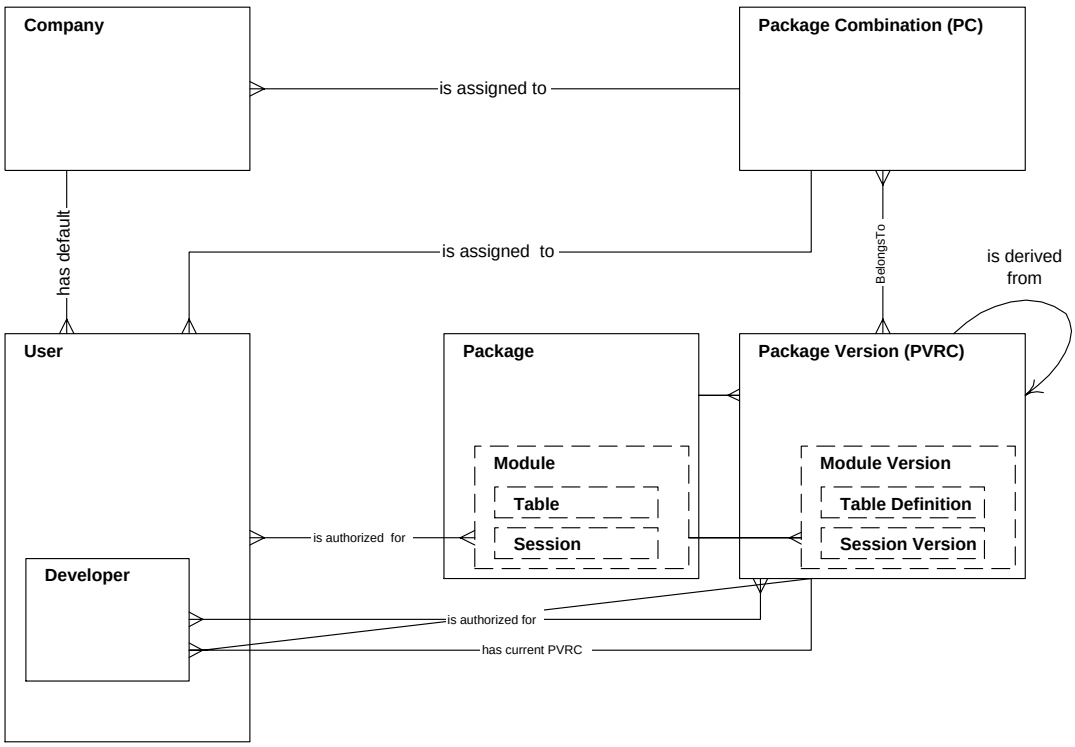

Figure 5.1.: Version management in BaanERP.

5.1. Package Versions (PVRCs)

Version management in BaanERP is hierarchical: a new version of a package can be derived from an already existing one. The new version has only to contain modified and newly created components. All other components are automatically executed from the predecessor version. The actual version of an application program (i.e. a *session*) is determined by the *package combination* selected by the login. For instance, the version of the session *Maintain Item Data* (tiitm0101m000) to be run is determined by the version of the *ti*-package contained in the *package combination* selected by the login. A package version is also called **Package VRC**, or 'PVRC' in short, where 'VRC' stands for *Version-Release-Customization*. This naming reflects the different levels of versioning in BaanERP. Through the continuous improvement and extension of the Baan ERP software, new upgrade versions are provided by Baan at regular time intervals. They are named using the two-level *Version-Release* schema. For instance, from the *version* name *B40S* and the *release* name *c3* the *Package VRC tiB40Sc3* is formed for the *ti* package within release *c3* of the standard version *B40S* of Baan IV.

The two-level versioning of the BaanERP standard software is supplemented by a third version naming level for enterprise-specific modifications (or *customization*) of a standard BaanERP version. For instance, the particular version of the *ti*-package derived from the Baan IV standard version c3 and customized for *Wagner World-Wide Enterprises* could be named *tiB40Cc3wag*. Baan strongly recommends that all customizations comply with its 'coding standards', a set of naming conventions. One such convention requires that the version code of an enterprise-specific customization of BaanERP be **B40C** instead of **B40S**, thus indicating the customization by the letter 'C'. The version indicators to be used are:

S	standard version
U	update version (service pack)
L	country-specific 'localisation'
B	branch-specific version
C	enterprise-specific customization

Baan has to provide many country-specific additions/modifications of its standard system in order sell BaanERP to countries all over the world (see Table 5.1).

A specific package version (called *"current PVRC"*) is assigned to each Baan developer as her current programming environment. It can be changed on the fly if the developer has the *general developer* authorization.

5.2. Derivation of Package Versions

A new package version (PVRC) can be derived from an already existing one. This mechanism allows to put only those components that are new or that deviate from their predecessor versions in the new PVRC. All other unchanged components do not have to be duplicated in the new PVRC, but are automatically executed from the first

Country/Region	Localisation Items
Argentina	central invoicing, Argentinean VAT
Brazilia	central invoicing, fiscal receipts, receipt schedule
Italy	Libro Giornale, withholding tax, LIFO by Year, BAM, VAT book
Japan	statement of accounts, customer approvals flow
Nordic countries	flow for interest invoice
Switzerland	central invoicing
India	excise invoice, MODVAT, personal ledger account, tax deduction at source
Australia	sales tax, prescribed payments system

Table 5.1.: Examples of country-specific customization.

predecessor PVRC in the derivation chain where they are found. A PVRC derivation chain represents a kind of specialization hierarchy where the most specific (or most recent) versions of software components are found at the end of the chain.

5.3. Package Combinations

A package combination is a specific configuration of various PVRCs, forming a comprehensive application software package. Each package combination is associated with a specific database schema, called *data dictionary*, that defines the structure of all involved tables.

Each BaanERP user is assigned a package combination that determines the application programs available to her. Since these programs work only for a particular database schema, the user can only access tables from clients that are associated with the same package combination.

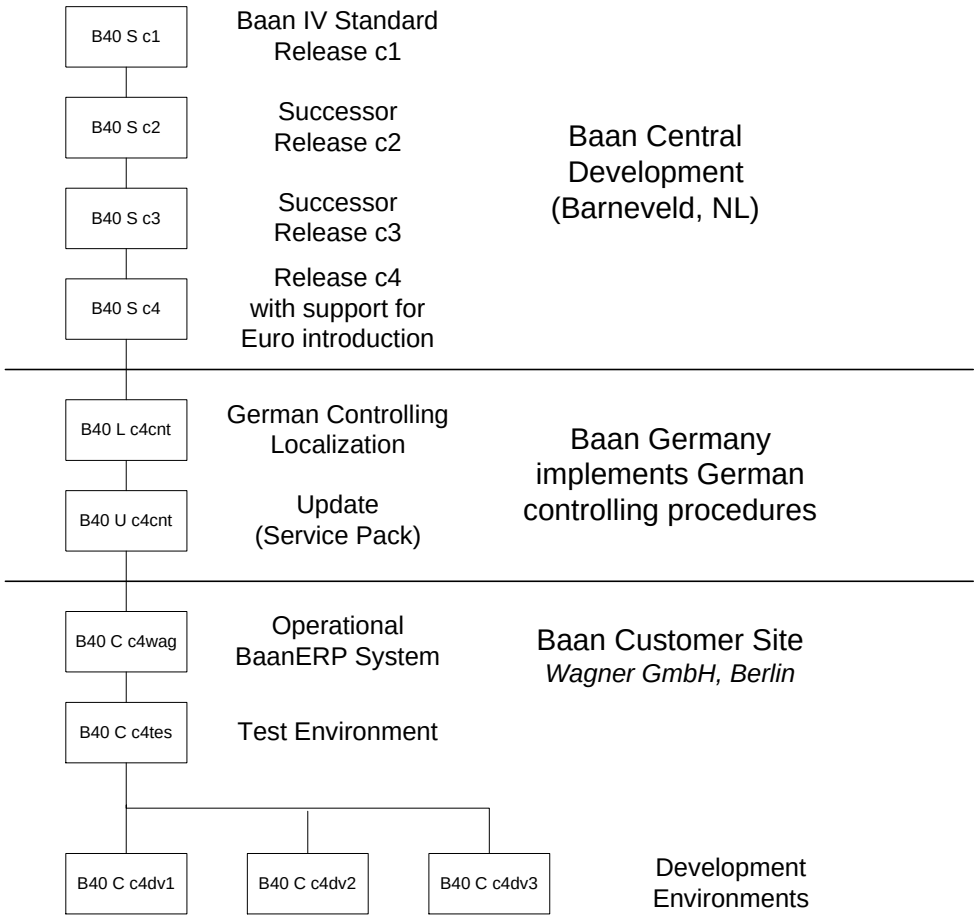

Figure 5.2.: How different versions are derived.

6. Companies and Users

BaanERP companies are defined by company number, name, currency, and a *package combination* that associates a specific database schema with the company.

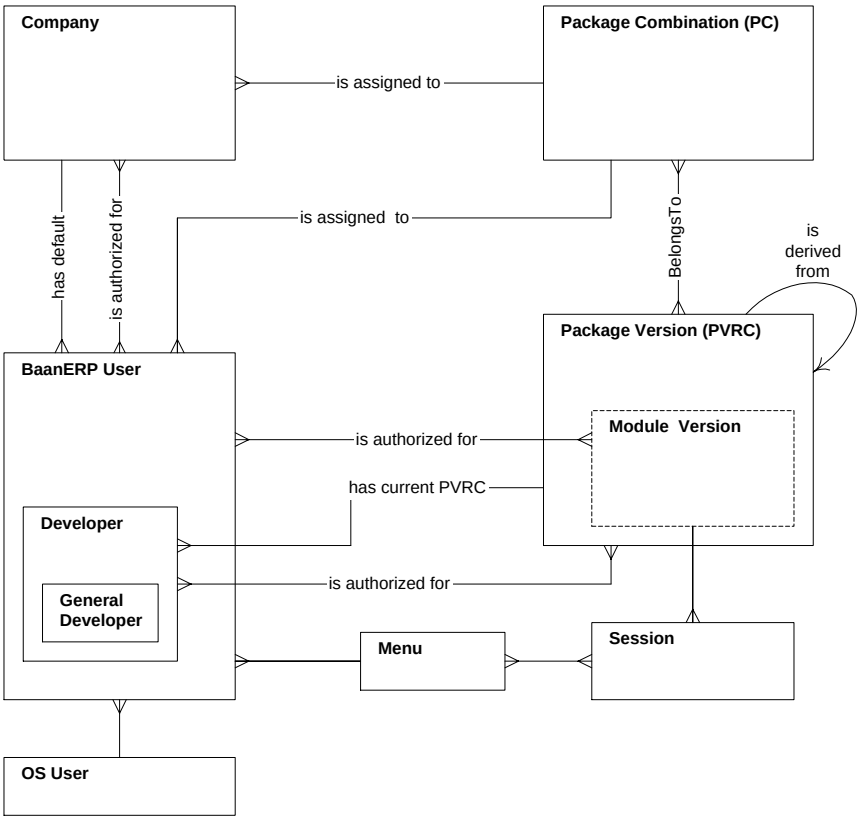

Figure 6.1.: Users and Developers.

A BaanERP user is defined by a BaanERP login name that must be associated with an operating system login. The most important attributes of a BaanERP user are: package combination, default company number, and startup menu. In order to allow one person to work with different BaanERP package combinations, multiple user accounts can be associated with one operating system login name.

Every BaanERP user is classified as either a *'normal user'* or a *'super user'*. A super user has unrestricted access to all sessions, all tables, and all companies with her package

combination. In the case of a normal user, the authorizations for sessions, companies and tables have to be explicitly defined by assigning roles and authorization templates.

A user can also be classified as a BaanERP developer who is authorized to customize user interface components, sessions, and tables. A developer has a *current PVRC* that defines her default programming environment, that is the package version whose components she can customize.

7. Database Handling

All of the application data used by BaanERP is stored in database tables in the underlying RDBMS. To keep the majority of the BaanERP processing independent of the RDBMS, BaanERP uses its own **data dictionary**. The data dictionary includes domain, schema, and referential integrity information that is stored in a database independent manner.

The BaanERP system provides a RDBMS interface, called 'database driver', to the major RDBMSs (Oracle, Informix, Sybase, DB2, and Microsoft SQL Server). The BaanERP database driver has a built-in mechanism for preserving **referential integrity**; it does not depend on the underlying RDBMS for maintaining referential integrity.

7.1. BaanERP Database Concepts

7.1.1. Database tables

A relational database presents information to the user in the form of tables. In a table, data is organized in columns and rows. Each column (also referred to as a field) represents a category of data. Each row (also referred to as a record) represents a unique instance of data for the categories defined by the columns. A field always refers to a domain, which defines a set of values from which one or more fields can draw their actual values. For example, the tcweek domain is the set of all integers greater than zero and less than or equal to 53.

7.1.2. Primary keys

Every database table has a field, or a combination of fields, which uniquely identify each record in the table. This unique identifier is referred to as the primary key. Primary keys are fundamental to database operations, as they provide the only record-level addressing mechanism in the relational model. Primary keys act as references to the records in a table.

7.1.3. References

With a relational database, you can store data across multiple tables and you can define relationships between the tables. This means that individual tables can be kept small and data redundancy can be minimized. A relationship exists between two tables when they have one or more fields in common. So, for example, a Customer Detail table can

be linked to an Order table by including a Customer ID field in both tables. In the Customer Detail table, the Customer ID field is the primary key. In the Order table, it is referred to as a foreign key. By linking the two tables in this way, there is no need for the Order table to include customer details such as name and address. Note that references from one table to another must always use the primary key.

7.1.4. Indexes

Indexes facilitate speedy searching and sorting of database tables. An index is a special kind of file (or part of a file) in which each entry consists of two values, a data value and a pointer. The data value is a value for some field in the indexed table. The pointer identifies the record that contains this value in the particular field. This is analogous to a conventional book index, where the index consists of entries with pointers (the page numbers) that facilitate the retrieval of information from the body of the book. Note that it is also possible to construct an index based on the values of a combination of two or more fields. Every table must have at least one index, which is an index on the primary key field(s). This is referred to as the primary index. An index on any other field(s) is referred to as a secondary index.

7.1.5. Transaction handling

With respect to database actions, a transaction is a sequence of related actions that are treated as a unit. The actions that make up a transaction are processed in their entirety, or not at all.

A transaction ends with the function `commit.transaction()` (all changes made during the transaction are stored in the database) or with the function `abort.transaction()` (no changes are stored in the database). A transaction starts either at the beginning of a process, with the function `set.transaction.readonly()`, with the function `db.lock.table()`, or after the preceding transaction has ended. A transaction is automatically rolled back (that is, it is undone) when a process is canceled and if a program ends without a `commit.transaction()` or `abort.transaction()` after the last database call. Undoing a transaction is only possible if the underlying database system supports this.

Certain database actions cannot be placed within a transaction, because they cannot be rolled back. These actions are: `db.create.table()`, `db.drop.table()`, and `set.transaction.readonly()`. These functions can be called only at the start of a program or after the end of the preceding transaction.

You can set a **retry point** immediately before a transaction. In case of an error, the system returns to this point and re-executes the transaction from there.

A read-only transaction is a transaction in which you are permitted only to read records (without lock) from the database. You retain read consistency during the entire transaction. This means that during the transaction your view of the database does not change, even if other users update the records. A read-only transaction starts with the function `set.transaction.readonly()` (this must be called after ending the preceding

transaction or at the beginning of the program) and ends with a `commit.transaction()` or `abort.transaction()`. A consistent view consumes a large amount of memory, so a read-only transaction must be as short as possible; user interaction during the transaction is not recommended.

7.1.6. Locking

Database inconsistencies can arise when two or more processes attempt to update or delete the same record or table. Read inconsistencies can arise when changes made during a transaction are visible to other processes before the transaction has been completed for example, the transaction might subsequently be abandoned.

To avoid such inconsistencies, BaanERP supports the following locking mechanisms: record/page locking, table locking, and application locking.

To ensure that only one process at a time can modify a record, the database driver locks the record when the first process attempts to modify it. Other processes cannot then update or delete the record until the lock has been released. However, they can still read the record. While one process is updating a table, it is important that other processes retain read consistency on the table. Read consistency means that a process does not see uncommitted changes. Updates become visible to other processes only when the transaction has been successfully committed. Some database systems do not support read consistency, and so a dirty read is possible. A dirty read occurs when one process updates a record and another process views the record before the modifications have been committed. If the modifications are rolled back, the information read by the second process becomes invalid. Some databases, such as SYBASE and Microsoft SQL Server 6.5, use page locking instead of record locking. That is, they lock an entire page in a table instead of an individual record. A page is a predefined block size (that is, number of bytes). The number of records locked partly depends on the record size.

7.1.7. Delayed locks

Locking a record for longer than required can result in unnecessarily long waiting times. The use of delayed locks solves this problem to a great extent. A delayed lock is applied to a record immediately before changes are committed to the database and not earlier. When the record is initially read, it is temporarily stored. Immediately before updating the database, the system reads the value of the record again, this time placing a lock on it. If the record is already locked, the system goes back to the retry point and retries the transaction. If the record is not locked, the system compares the content of the record from the first read with the content from the second read. If changes have been made to the record by another process since the first read, the error `ROWCHANGED` is returned and the transaction is undone. If no changes have occurred, the update is committed to the database.

You place a delayed lock by adding the keyword FOR UPDATE to the SELECT statement. For example:

```
table tpctst999
```

```
db.retry.point()
SELECT pctst999.* FROM pctst999 FOR UPDATE
SELECTDO
   pctst999.dsca = "...."
   ....
   db.update(tpctst999, DB.RETRY)
ENDSELECT
```

A **retry point** is a position in a program script to which the program returns if an error occurs within a transaction. The transaction is then retried. There are a number of situations where retry points are useful:

- During the time that a delayed lock is applied to a record/page, an error can occur that causes the system to execute an abort.transaction(). In such cases, all that BaanERP can do is inform the program that the transaction has been aborted. However, if retry points are used, the system can automatically retry the transaction without the user being aware of this.
- Some database systems generate an abort.transaction() when a dirty record is read (that is, a record that has been changed but not yet committed). An abort.transaction() may also be generated when two or more processes simultaneously attempt to change, delete, or add the same record. In all these situations, BaanERP Tools can conceal the problem from the user by using retry points. It simply retries the transaction. If there is no retry point, the transaction is aborted and the session is terminated.
- In BaanERP, updates are buffered, so the success or failure of an update is not known until commit.transaction() is called. If an update fails, the commit of the transaction also fails, and the entire transaction must be repeated. If retry points are used, the system automatically retries the transaction.
- Retry points can also resolve potential deadlock problems. If, for example, the system is unable to lock a record, it rolls the transaction back and tries again.

It is vital that retry points are included in all update programs. The retry point for a transaction must be placed at the start of a transaction. The following example illustrates how you program retry points:

```
db.retry.point() | set retry point
if db.retry.hit() then
   ...... | code to execute when the system
          | goes back to retry point
else
   ...... | initialization of retry point
endif
```

The function `db.retry.hit()` returns 0 when the retry point is generated that is, the first time the code is executed. It returns a value unequal to 0 when the system returns to the retry point through the database layer.

When the system goes back to a retry point, it clears the internal stack of functions, local variables, and so on that were called during the transaction. The program continues from where the retry point was generated. The value of global variables is NOT reset.

When a commit fails, the database automatically returns to its state at the start of the transaction; the program is set back to the last retry point. It is vital, therefore, that the retry point is situated at the start of the transaction. The `db.retry.hit()` call must follow the `db.retry.point()` call. Do not place it in the SQL loop itself as this makes the code very untransparent. When a retry point is placed within a transaction, the system produces a message and terminates the session.

7.1.8. Table locks

BaanERP provides a table locking mechanism, which enables you to lock all the records in a specified table. A table lock prevents other processes from modifying or locking records in the table but not from reading them. This is useful when a particular transaction would otherwise require a large number of record locks. You use the `db.lock.table()` function to apply a table lock.

7.1.9. Application locks

An application lock prevents other applications and users from reading and/or modifying an applications data during critical operations. It is not part of a transaction and so is not automatically removed when a transaction is committed. Instead, an application lock is removed when the application ends or when `appl.detete()` is called.

7.2. Microsoft SQL Server Database Driver

This section describes the RDBMS interface issues with respect to Microsoft SQL Server.

Because so many tables are needed, a convention is used for naming tables, columns within tables, and indexes to data within the tables. This chapter describes the data dictionary and the naming conventions used by the BaanERP database drivers to access data stored in the RDBMS. It also discusses how BaanERP data types are mapped to SQL Server data types.

The BaanERP data dictionary maps BaanERP data types, domains, schemas, and referential integrity information to the appropriate information in the RDBMS. When storing or retrieving data in the RDBMS, the database driver maps data dictionary information to database table definitions.

BaanERP data dictionary information can be kept in shared memory where it will be available to all running BaanERP application servers. The data dictionary information is shared among all the sessions open within a single database driver.

The BaanERP data types cannot be used directly by the database driver to create SQL Server tables. This is because not all BaanERP data types exactly match SQL Server data types. To create valid SQL Server tables, the driver must perform some mapping or translation. When mapping the BaanERP data dictionary to tables in SQL Server, conventions are used for the table names, column names, and index names.

7.2.1. Table naming convention

The external name of a BaanERP table stored in SQL Server has the following format:

t⟨PackageCode⟩⟨TableName⟩⟨CompanyNumber⟩

The components of the external table name are:

PackageCode A two-letter code referring to the BaanERP package the table belongs to. For example, a table defined by the Tools package has the package code tt.

TableName The data dictionary table name consists of a three-letter module identifier followed by a three-digit number. The module identifier refers to the module the table belongs to; the number is just a sequence number.

CompanyNumber Within BaanERP, three-digit numbers are used to identify different instances of the BaanERP application database, called 'companies'. Company number 000 denotes the meta-database containing various system data common to all companies (including currencies and languages used). In addition to company 000, there may be several other companies in a BaanERP system, each with its own set of tables for application data.

For example, the data dictionary table *adv999* with company number 000 is created in SQL Server as `tttadv999000`.

7.2.2. Column naming convention

Each column in the BaanERP data dictionary corresponds to one or more columns in a SQL Server table. The rules for column names are as follows:

General When a BaanERP column name is created in SQL Server, it is preceded by the string `t_`. For example, the BaanERP column with the name *cpac* is created in SQL Server with the name `t_cpac`. If a BaanERP column name contains a period, it is replaced by the underscore character.

Long string columns BaanERP columns of type string can exceed the maximum length of character columns in SQL Server. The SQL Server data type CHAR has a limit of 254 characters. When a BaanERP string column exceeds this limit, the column is split into segments with up to 254 characters each. The first 254 characters are mapped to a column where the name of the column is extended with `_1`. The

BaanERP data types	MSQL data types
INT	Smallint
LONG	Integer
FLOAT	Real
DOUBLE	Float
CHAR	Binary(1)
STRING(N)	Char(n)
TIME	DateTime
DATE	DateTime
TEXT	Integer
BITSET	Integer
ENUM	Binary(1)

Table 7.1.: Mapping between BaanERP and MSQL data types.

next 254 characters are mapped to a column with a name extended by _2, and so on, until all the characters of the string are mapped to a column. For example, if a BaanERP string column called *desc* contains 300 characters, the following two SQL Server columns are created: `t_desc_1` with size 254, and `t_desc_2` with size 46.

Array columns In the BaanERP data dictionary, array columns can be defined. An array column is a column with multiple elements. The number of elements is called the depth. For example, a column containing a date can be defined as an array of three elements: a day, a month, and a year. In SQL Server, the three elements of the array column are placed in separate columns. The names of these columns include a suffix with the element number. For example, an array column called *date* will be transformed to: `t_date_1` for element 1, `t_date_2` for element 2, and `t_date_3` for element 3.

7.2.3. Data type mapping

Table 7.1 shows the mapping between BaanERP data types and their SQL Server counterparts.

Note that the MSQL driver uses the SQL Server CHAR data type since ANSI-compliant behavior is expected for character data, such as with the BaanERP string type. Since BaanERP SQL expects ANSI-compliant string comparison semantics, the SQL Server CHAR data type is used instead of VARCHAR. This SQL Server data type is used because a BaanERP string data type has characteristics that conform to the ANSI specification for character data. When the CHAR data type is used, operations such as comparison and concatenation can be done in a predefined manner with predictable results.

In addition to the above naming conventions and data types, the following rules apply when mapping BaanERP data to SQL Server data:

- Since the binary sort order is selected during the installation, SQL Server treats object names with case sensitivity.
- All columns created by the BaanERP database driver have the `NOT NULL` constraint. BaanERP does not support the NULL value concept of SQL.
- The date range supported by the BaanERP application server is not the same as the range for SQL Server (SQL Server is more restrictive), so some BaanERP dates are not valid when stored with the MSQL driver. The BaanERP date number 0 is mapped to the earliest possible date in SQL Server (01-Jan-1753). The earliest possible BaanERP date is then 02-Jan-1753 and the latest is 31-Dec-9999.

7.2.4. The ODBC interface

ODBC is an application programming interface (API) used to communicate with the database server. It is made up of a function library that can be called from an application program to execute SQL statements and communicate with the data source. The ODBC functions called by the MSQL database driver perform the following actions:

- Connect to Microsoft SQL Server (open session)
- Allocate a statement handle
- Parse a SQL statement
- Bind input variables
- Define result variables
- Execute a SQL statement
- Fetch the resulting rows
- Commit or abort a transaction
- Close, unbind and drop a cursor
- Disconnect from MSQL (close session)

8. Programming

8.1. Introduction

The development tools supplied by the BaanERP Tools package enable developers to program additional functionality for existing BaanERP applications or to build entirely new applications. The features provided by the development tools include:

- The BaanERP 3GL programming language.
- 4GL language features that enable you to add to or modify the default behaviour of sessions, reports, and the Data Access Layer.
- Baan SQL, which enables you to retrieve database data.
- Support for Dynamic Link Libraries (DLLs).
- A debugger that enables you to control and test the execution of your programs.

3GL scripts are program scripts that are either linked to sessions without forms or not linked to sessions at all. They do not have any relationship with the 4GL Engine (previously known as the Standard Program). When creating such scripts, you must specify the entire program flow, including the main function. You cannot use 4GL event sections or functions.

In BaanERP applications, the 4GL engine provides much of the default functionality for a session. You can add to or modify the default functionality of a session by creating a 4GL script that is linked to the session. 4GL scripts are event-oriented. They consist of one or more event sections in which you program actions to be performed at particular states of execution of the 4GL engine. In previous versions of the software, changes or additions to the default functionality of a session were programmed in a single script that was associated with the session. In BaanERP, user interface actions and database actions have been separated. The *Data Access Layer (DAL)* now handles database interaction. Programmers create a user interface (UI) script to change the default behavior of a session. They create a DAL script to program all the logical integrity rules for a particular table. The statements programmed in event sections can be a combination of 3GL/4GL functions and 3GL language statements.

Report scripts are 4GL scripts that are linked to a report in order to add to or modify its output. In a report script, you can program actions that you want to be performed at particular stages of the report execution.

You can use Baan SQL (Structured Query Language) in 4GL scripts to retrieve data from database tables. There are two ways to use Baan SQL in a 4GL program. You can embed it in the language (embedded SQL), or you can use BaanERP 4GL functions (dynamic SQL).

The *bshell* (the BaanERP application server) provides a multitasking execution environment for BaanERP applications. Each bshell can execute and schedule multiple parallel processes.

8.2. 3GL programming language features

8.2.1. Data types

There are five types of variables: long, double, string, table, and domain variables.

Long variables can contain any whole number from -2147483648 to 2147483647. For numbers beyond this range, use double variables instead. Physically, four bytes are reserved for each long variable.

Double variables are used for any number containing a decimal point, with a maximum of 15 significant digits (8 bytes).

String variables are used for symbolic names, descriptions, and short text. The maximum length of a string is 1024 characters. A string variable can be declared as a multibyte string, in order to handle multibyte or bidirectional characters. In a single-byte string, each byte contains a single character. But in a multibyte string, characters can occupy from one to four bytes.

Table declarations are used for accessing database tables in a program. The table must be defined in the data dictionary.

Domain variables assume the characteristics of a database domain defined in the data dictionary. A domain may be based on any of the following data types: long, byte, integer, date, enumerate, set, float, double, string, text. Each domain defined in the data dictionary can be used in a declaration of your program.

When declaring temporary variables for storing values of database fields, you can use normal variables of type long, double, or string. But this can cause problems if the length or type of the table field has been changed in the data dictionary. It is preferable to use domain declarations for storing the value of a database table field. When you use a domain declaration, the type and length for the declared variable are read from the data dictionary. Variables of type enumerate or set must always be declared with a domain declaration. It is also possible to declare an array by using a domain.

8.2.2. Programming Statements

Baan 3GL comprises the usual imperative programming statements:

- Variable assignment
- `IF ...THEN ...ELSE ...ENDIF`
- `ON CASE` statement
- `WHILE`-, `REPEAT`-, and `FOR`-Loops with `BREAK` and `CONTINUE` options
- Procedure calls (procedures are called 'functions')

In addition to the standard loop constructs there is an embedded SQL loop statement consisting of a SQL query expression (by means of the standard `SELECT ...FROM ...WHERE ...` statement) followed by `SELECTDO ...SELECTEMPTY ...ENDSELECT`. The statements in the `SELECTDO` block are executed iteratively for each row of the query result set, whereas the statements in the `SELECTEMPTY` block are executed only if the result set is empty.

8.2.3. Example

```
FUNCTION LONG compnr_check( LONG new_compnr )
{
 TABLE tpctst999

 SELECT pctst999.* FROM pctst999
 WHERE pctst999.compnr = :new_compnr
 ORDER BY pctst999.compnr
 SELECTDO
    compnr = pctst999.compnr
 SELECTEMPTY
    RETURN(FALSE)
 ENDSELECT
 RETURN(TRUE)
}
```

8.3. 4GL programming language features

When you create a session, the session generator generates a standard source. This standard source provides default session functionality. If the required functionality of the session is not fully implemented by the standard source, you can program the additional functionality in a 4GL program script.

In Baan IV, changes or additions to the default functionality for a session were programmed in a single script that was associated with the session. Both user interface actions and database actions were programmed in this script. In BaanERP, user interface actions and database actions have been separated. The Data Access Layer (DAL) now handles database interaction. Programmers create a user interface (UI) script to change

the default behavior of a session. They create a DAL script to program all the logical integrity rules for a particular table.

8.3.1. 4GL script types

There are four types of 4GL program scripts:

Type 1 single-occurrence (details) session, operating on a main table.

Type 2 multi-occurrence (overview) session with group fields, and operating on a main table.

Type 3 multi-occurrence (overview) session, without group fields, operating on a main table.

Type 4 print/processing sessions, without main table.

4GL scripts are event oriented. They consist of one or more event sections in which you program actions to be performed at particular states of execution of the standard program - for example, when a session is started, before a form is activated, before input to a field, after input to a field, and so on. It is not necessary to program all sections, only those for which the standard program does not provide the required functionality. The statements programmed in a section can be a combination of 3GL/4GL functions and 3GL statements.

8.3.2. Example

```
field.pctst099.item:
check.input:
   SELECT pctst001.* FROM pctst001
   WHERE pctst001.item = :pctst099.item
   AS SET WITH 1 ROWS
   SELECTDO
      ....
   SELECTEMPTY
      pctst001.dsca = "*****"
      set.input.error(".....")
   ENDSELECT
before.input:
   if ..... then
      attr.input = false
   endif
after.display:
   SELECT pctst001.* FROM pctst001
   WHERE pctst001.item = :pctst099.item
   AS SET WITH 1 ROWS
```

```
    SELECTEMPTY
       pctst001.dsca = "*****"
    ENDSELECT

field.pctst099.date:
before.input:
    attr.oformat$ = "%D002,2"
```

8.4. Report scripts

BaanERP reports are used to output data from the database to a variety of devices (for example, printers, displays, and files). The contents and layouts of reports are defined in the data dictionary. In addition, you can link a report script to a report. In a report script, you can program actions that you want to be performed at particular stages of the report execution. For example, you can create a script to perform calculations on the report data or to read records from related tables that are not automatically available to the report. You program report scripts in the same way as you program 4GL program scripts, except that report scripts use different event sections and some special functions.

A report script consists of one or more event sections in which you program actions to be performed at particular states of execution of the report to which the report script is linked. The statements programmed in a report script section consist of a combination of 3GL language statements and report script functions. Report scripts support the following event sections: program sections, report sections, and text field sections.

8.5. Data Access Layer (DAL) Functions

8.5.1. Overview

In BAAN applications, the standard program provides much of the default functionality for a session. In previous versions of the software, changes or additions to the default functionality for a session were programmed in a single script that was associated with the session. Both user interface actions and database actions were programmed in this script. In BaanERP, user interface actions and database actions have been separated. The Data Access Layer (DAL) now handles database interaction. Programmers create a user interface (UI) script to change the default behavior of a session. They create a DAL script to program all the logical integrity rules for a particular table. So the DAL ensures the logical integrity of the database. As in previous versions of the software, the database server ensures the referential integrity of the database.

The DAL script for a particular table has the same name as that table. It is implemented as a DLL that can be accessed by user interface scripts (via the standard program), by other DALs, and by external programs (via the Common Data Access Server (CDAS). The following diagram illustrates the overall relationship of these components.

8.5.2. Database integrity checks

The following are examples of some logical integrity rules that could be programmed in a DAL script:

- When customers have reached their credit limit, they cannot order further items.
- If the invoice for an order has been printed, the order cannot be changed.
- If the current PVRC of a user is not equal to the PVRC of the program script, the script cannot be compiled.

Programming database integrity checks in a separate DAL script has two main advantages:

- Code reuse: The integrity rules do not have to be replicated in each session that uses a particular table.
- External access: External applications can access the database via the CDAS and the DAL.

For an overview of the interaction between the user interface, the standard program, and the DAL, see DAL, UI, and standard program interaction.

8.5.3. Business methods

In addition to performing data integrity checks, the DAL provides business methods for handling non-interactive database modifications such as printing sales orders or posting all orders to history. A business method is a function that performs a task that involves manipulating and/or checking one or more tables in the database. The function is programmed in a DAL script and can be called directly from a UI script. It must be programmed in the DAL script of the most relevant table.

Users can activate a business method with a single command. There is then no further user interaction. The UI script calls the relevant function in the DAL script. The function performs all operations to complete the required task. The user must wait until the business method finishes before continuing with other tasks. However, if a progress window is provided, the user can cancel the business method by clicking on the Cancel button. The UI script starts a business method by calling the `dal.start.business.method()` function.

8.5.4. Interaction between UI, DAL, and standard program

A DAL script contains all the logic integrity rules for a particular object set. These rules are referred to as hooks and they can be programmed for every possible manipulation of an object in the object set. For each session with a main table, the standard program ensures that the integrity rules for the table are checked each time an update, delete, insert, or read operation is performed on an object of the table. If there is no DAL

script for the particular object set being accessed, no logic integrity checks are performed (unless they are programmed in the UI script itself).

A DAL script can contain hooks that prevent access to the database and hooks that prevent data being passed back to the user interface. The former are executed before the database action. The latter are performed after the database action.

If a user changes the address of a customer on a form, the standard program changes the address for that customer in the database via the DAL. If you want certain restrictions to apply to the update action, you can program a property hook in the DAL of the sessions main table to impose these restrictions.

8.5.5. UI function calls

The UI script can use either the database write functions or the DAL Data Access Methods (DAM) to manipulate the database. The database write functions are direct database calls. They do not use the DAL. In this case, any logic integrity checks required must be programmed in the UI script itself. This means that they cannot be reused by other sessions. In preference, use the DAL Data Access Methods. These access the database via the DAL, so all necessary checks are automatically performed.

When the DAL is used to manipulate the database, the main steps involved are as follows:

1. The user issues a command through the user interface to access a record in the database.
2. The standard program loads the appropriate DAL and calls the hooks in the DAL to check the integrity rules for the object.
3. If the particular database action is permitted, the standard program issues the appropriate database call.
4. After the database has been updated, the DAL can perform further checks to determine whether or not data is passed back to the user interface.
5. The standard program passes data back to the user interface (provided that the integrity rules permit this).

8.5.6. DAL hooks

A hook is a function, with a predefined name, that the DAL programmer programs in a DAL script. The function is used to program logic integrity rules for database access. The DAL script is compiled into a DLL.

When a user issues a command to access the database, the standard program loads the DAL DLL for the object being accessed and calls the hooks to perform the integrity checks. Provided that a DAL script exists for an object set being accessed, the standard program always ensures that the hooks in the DAL are called at the appropriate times. A DAL script can contain two types of hooks: property hooks and object hooks.

9. Enterprise Modeler

The Enterprise Modeler assists in developing a vision on:

- How to functionally structure an organization
- How to organize the business processes of the organization
- How to integrate the information system with these business processes

In addition, the Enterprise Modeler can also be used to assist in the actual implementation of the Baan applications by:

- Setting parameters in BaanERP.
- Creating BaanERP users, based on the role and employee data of the Enterprise Modeler.
- Creating the menus and session authorizations for users.

A **reference model** represents a line of business or business typology and can be built from libraries (repositories) of the following enterprise-modeler components:

- Business control diagrams
- Business function models
- Business process models
- Organization models

A reference model can serve as a basis for an organization-specific project model.

A **project model** is an organization-specific model that can also be built from a library (repository) of Enterprise Modeler components, and can be based on a reference model.

An enterprise-structure model is graphically represented by a map, on which all the participants (enterprise units) in the supply chain are graphically represented at organization/enterprise level. An enterprise-structure model can be used for determining currencies and transaction types between enterprise units.

Unlike business functions and business processes, organization diagrams are not a mandatory part of DEM. There is no enforced interdependency between business processes or business functions on the one hand and organization diagrams on the other

hand. Nevertheless, it may be useful to define an organization diagram to visualize the organization structure and the roles and responsibilities of departments and employees. In the project model, employees can be linked to the roles that have been defined for the reference-model specific organization diagram. The link between a role and an employee transforms a line-of-business oriented diagram into a company-specific (organization-specific) diagram.

9.1. Roles and Authorization Types

You can link roles and responsibilities to the following components:

- Reference models
- Project models
- Business processes
- Business process activities
- Organization units

You must first define roles as part of a reference model or of a project model before you can link those roles to any of the other components above. Assume that you link roles and responsibilities to business processes and activities. If for a certain process one role carries all the responsibilities for all activities, it is sufficient to link the role and the responsibilities to the process.

Examples of roles are: Manager and Secretary. Examples of responsibilities are: maintain data, inform manager, check data.

9.2. Rules

There are four different types of rules.

9.2.1. Consistency rule

An expression containing a combination of business functions on the basis of which one or more other business functions must also be included in a reference or project model.

Example: `IF <BF,5> AND <BF,12> THEN <BF,9>`

Explanation: If business function 5 and business function 12 are part of a reference model or a project model, business function 9 must also be part of the reference model or the project model.

9.2.2. Parameter-setting rule

An expression that determines the value of one or more parameters in a reference model or project model. The value is determined on the basis of a combination of business functions, business processes, and/or static conditions.

Example:

```
IF <BF,1> OR <BF,3> THEN
Parameter:  tttld000.user User jjohnson
```

Explanation: If business function 1 or business function 3 is part of the reference model or the project model, the value of the User parameter is jjohnson.

9.2.3. Transformation rule

An expression imposing a unilateral dependency of business processes in relation to business functions. If specific business functions are present in a reference or project model, business processes are automatically incorporated in that reference model or project model.

Example:

```
IF <BF,1> AND <BF,3> THEN
DPL081 MRP Purchase Orders
```

Explanation: If business functions 1 and 3 are part of the reference model or the project model, business process DPL081 must also be part of the reference model or the project model and can therefore be incorporated automatically.

9.2.4. Static-condition

An expression, containing a combination of business functions and/or business processes on the basis of which the value of static conditions is set in a reference or project model.

Example:

```
IF <BF,1> OR <BF,3> THEN
LTC LTC implemented Yes
```

Explanation: If business function 1 or 3 is part of the reference model or project model, the value of the LTC static condition is Yes.

9.3. Enterprise-structure modeling

An enterprise-structure model is graphically represented by an enterprise-structure diagram, in which the supply chain at enterprise level is modeled. All agents in the supply chain are graphically represented on a map. Example of agents are:

- Customers
- Sales offices
- Distribution centers
- Assembly sites
- Manufacturing sites
- Suppliers
- Central planning/purchasing sites

Between enterprise units all kinds of relationships can exist in term of material flows, financial flows, and information flows. The enterprise units from this specific enterprise-structure model are the direct link between the BaanERP application data, such as a warehouse or an entity on the one hand and the model on the other hand. As opposed to business processes, and business-function diagrams and so on, an enterprise-structure diagram is not part of a business model (reference model or project model). A business model, on the contrary, is linked to an enterprise unit, which is part of an enterprise-structure diagram.

The enterprise-structure model is the top level of an enterprise model. One level below the enterprise-structure model you find business models (reference or project models). Business models can be linked to enterprise units. This implies that the enterprise unit is the indirect link between an enterprise-structure model and multiple business models.

Because a business model is used to set parameters at BaanERP company level, only one business model must be used to represent one BaanERP company. Therefore, different enterprise units must be associated with different (logistic) companies.

You cannot directly create an operational enterprise-structure model. First you must create one or more enterprise-structure models, after which you can specify one of those models as the operational model. At runtime the operational enterprise-structure model is used by the BaanERP applications to determine prices and currencies when goods are transferred from one enterprise unit to another.

There are two different types of enterprise-structure models:

Reference enterprise-structure model A generic enterprise-structure model that represents a certain branch of industry.

Enterprise-structure model A model that is built for one specific company.

You can directly create a (specific) enterprise-structure model or base it on a (more generic) reference enterprise-structure model.

A. Glossary

ACID Transactions satisfy the properties of Atomicity, Consistency, Isolation, and Durability. As complex update operations on possibly distributed data, transactions are subject to various failure conditions. For instance, some step of a transaction may violate an integrity constraint, or the connection to a remote database may get lost, or the server running the transaction application may crash during the execution of a transaction. *Atomicity* denotes the requirement that a transaction needs to be all-or-nothing: either it executes completely or not at all. If all steps of a transaction have been performed successfully, the transaction *commits*, making all involved updates permanent. If some step fails, the transaction *aborts*, rolling back all involved updates. A transaction program should maintain the *consistency* of the databases it updates. Since the internal consistency of a database is defined by its integrity constraints, this means that all integrity constraints have to be satisfied when the transaction is completed. For performance reasons, transactions are executed concurrently by interleaving the execution of their single steps. Special techniques (such as locking mechanisms) are needed to avoid interference between transactions accessing the same database objects. Referring to concurrent transactions, one says that the *isolation* property is satisfied if their effects are the same as if they were run one at a time in some order, or, in other words, if they are *serializable*. Finally, a transaction is *durable* if all of its updates are stored on a permanent storage medium when it commits. This is usually achieved by writing a copy of all the updates of a transaction to a log file. If the system fails after the transaction commits and before the updates go to the database, then after the system recovers it rereads the log and checks that each update actually made it to the database; if not, it re-applies the update to the database.

Notice that unlike atomicity, isolation and durability which are guaranteed by the transaction processing system, maintaining consistency is the responsibility of the application programmer.

Agent A computer program that can accept tasks from its human user, can figure out which actions to perform in order to solve these tasks and can actually perform these actions without user supervision, is an example of a *software agent*. More generally, any system that is capable of perceiving events in its environment, of representing information about the current state of affairs, and of acting in its environment guided by perceptions and stored information, is called an agent. If the environment is virtual, such as the Internet, we deal with software agents. If

the environment is physical, we deal either with natural agents such as human beings and animals, or with artificial physical agents such as robots and embedded systems. The term agent denotes an abstraction that subsumes all these different cases.

Typical examples of software agents are web shopping assistants and life-like characters (artificial creatures) in computer games. Typical examples of artificial physical agents are the entertainment robot *Aibo* by Sony and the unmanned NASA space vehicle *Deep Space One*. It is expected that software agents capable to assist their users to cope with the increasing complexities caused by the accelerating and virtually uncontrolled growth of the World Wide Web will play a major role in the future.

The term *agent* is sometimes used as a synonym for *intelligent system*. But, in general, agents do not have to be 'intelligent'. In software engineering, for instance, the ability of an agent to communicate and cooperate with other systems in a flexible manner, and the ability of a mobile agent to migrate to another computer providing more resources via suitable network links, are considered more fundamental than any form of 'intelligence'.

The philosophical basis for the agent paradigm in computer science is the concept of *intentional systems* introduced by Daniel Dennett in [Den71] to characterize systems whose behavior can be best explained and forecasted by ascribing them beliefs, goals and intentions. Following Dennett, Yoav Shoham proposed in [Sho93] a mentalistic approach to model and program agents, called *Agent-Oriented Programming*. In this approach, the data structures of an agent program reflect basic mental components such as beliefs, commitments, and goals, while the agent's behavior is determined by reaction rules that refer to its mental state and are triggered by events, and possibly by its planning capabilities for pro-actively achieving its goals.

An important feature of agents is their ability to communicate and interact with each other. For artificial agents, communication is normally implemented by an asynchronous message passing mechanism. Agents created by different designers must speak the same agent communication language for expressing the type of communication act, and must refer to shared ontologies for being able to understand the contents of the messages of each other. Conversations between agents often follow a certain protocol that defines the admissible patterns of message sequences.

Similarly to the notion of *objects* in software engineering, the term *agent* denotes an abstraction that leads to more natural and more modular software concepts. While the state of an object is just a collection of attribute values without any generic structure, the state of an agent has a mentalistic structure comprising perceptions and beliefs. Messages in object-oriented programming are coded in an application-specific ad-hoc manner, whereas messages in agent-oriented programming are based on an application-independent agent communication language. An

agent may exhibit pro-active behavior with some degree of autonomy, while the behavior of an object is purely reactive and under full control of those other objects that invoke its methods.

Agent-Oriented Information Systems (AOIS) represent a new information system paradigm where communication between different (software-controlled) systems and between systems and humans is understood as communication between agents whose state consists of mental components (such as beliefs, perceptions, memory, commitments, etc.). In enterprise information systems, for instance, the AOIS paradigm implies that *business agents* are treated as first class citizens along with business objects.

There is also an AOIS workshop series.

Business Rules are statements that express a *business policy*, defining or constraining some aspect of a business, in a declarative manner (not describing/prescribing every detail of their implementation). Business rules may be strict or defeasible (allowing exceptions). They can be formalized as integrity constraints, derivation rules, or reaction rules.

Business Transaction A sequence of actions performed by two or more agents, involving a flow of information and a flow of money, and normally also a flow of material or certain other physical effects. Usually, it requires some bookkeeping to record what happened. Today, this bookkeeping is done by the computer-based information systems of the involved business partners. Since an enterprise may participate in a great number of business transactions at the same time, this requires sophisticated information system technologies for guaranteeing high performance and consistency.

CORBA The *Common Object Request Broker Architecture* is an established standard allowing object-oriented distributed systems to communicate through the remote invocation of object methods.

Database Management System (DBMS) The main purpose of a DBMS is to store and retrieve information given in an explicit linguistic format (using various symbols). As opposed to certain other types of information that are also processed in agents, this type of information is essentially propositional, that is, it can be expressed as a set of propositions in a formal language. In the sixties and seventies, pushed by the need to store and process large data sets, powerful database management systems extending the file system technology have been developed. These systems have been named *hierarchical* and *network* databases, referring to the respective type of file organization. Although they were able to process large amounts of data efficiently, their limitations in terms of flexibility and ease of use were severe. Those difficulties were caused by the unnatural character of the conceptual user-interface of hierarchical and network databases consisting of the rather low-level data access operations dictated by their way of implementing storage and retrieval. Thus, both

database models have later on turned out to be cognitively inadequate. The formal conceptualization of relational databases by Codd in the early seventies rendered it possible to overcome the inadequacy of the first generation database technology. The logic-based formal concepts of the relational database model have led to more cognitive adequacy, and have thus constituted the conceptual basis for further progress (towards object-relational, temporal, deductive, etc. databases). Driven by the success of the object-oriented paradigm, and by the desire to improve the relational database model, object-relational databases are now increasingly regarded the successor to relational databases. This development is being reflected in the progression of SQL, the established standard language for database manipulation, from SQL-89 via SQL-92 to SQL-99.

Data Warehouse A very large database that stores historical and up-to-date information from a variety of sources and is optimized for fast query answering. It is involved in three continuous processes: 1) at regular intervals, it extracts data from its information sources, loads it into auxiliary tables, and subsequently cleans and transforms the loaded data in order to make it suitable for the data warehouse schema; 2) it processes queries from users and from data analysis applications; and 3) it archives the data that is no longer needed by means of tertiary storage technology.

Most enterprises today employ computer-based information systems for financial accounting, purchase, sales and inventory management, production planning and control. In order to efficiently use the vast amount of information that these operational systems have been collecting over the years for planning and decision making purposes, the various kinds of information from all relevant sources have to be merged and consolidated in a data warehouse.

While an operational database is mainly accessed by OLTP applications that update its content, a data warehouse is mainly accessed by ad hoc user queries and by special data analysis programs, also called *Online Analytical Processing* (OLAP) applications. For instance, in a banking environment, there may be an OLTP application for controlling the banks's automated teller machines (ATMs). This application performs frequent updates to tables storing current account information in a detailed format. On the other hand, there may be an OLAP application for analyzing the behavior of bank customers. A typical query that could be answered by such a system would be to calculate the average amount that customers of a certain age withdraw from their account by using ATMs in a certain region. In order to attain quick response times for such complex queries, the bank would maintain a data warehouse into which all the relevant information (including historical account data) from other databases is loaded and suitably aggregated.

Typically, queries in data warehouses refer to business events, such as sales transactions or online shop visits, that are recorded in event history tables (also called 'fact tables') with designated columns for storing the time point and the location at which the event occurred. Usually, an event record has certain numerical

parameters such as an amount, a quantity, or a duration, and certain additional parameters such as references to the agents and objects involved in the event. While the numerical parameters are the basis for forming statistical queries, the time, the location and certain reference parameters are used as the *dimensions* of the requested statistics. There are special data management techniques, also called *multidimensional databases*, for representing and processing this type of multidimensional data. For further research, see [Cod94, AM97, IWK97].

Derivation Rules (or *deduction rules*) are used for defining intensional predicates and for representing heuristic knowledge, e.g. in *deductive databases* and in *logic programs*. Intensional predicates express properties of, and relationships between, entities on the basis of other (intensional and extensional) predicates. Heuristic knowledge is often represented in the form of *default rules* which may be naturally expressed using the weak and strong negation from partial logic (like in the formalism of 'extended logic programs'). While relational databases allow to define non-recursive intensional predicates with the help of *views*, they do not support default rules or any other form of heuristic knowledge.

EDI Electronic Data Interchange, denotes the traditional computer-to-computer exchange of standard messages representing normal business transactions including payments, information exchange and purchase order requests. Besides the two main international standards for EDI messages, UN/EDIFACT and ANSI X.12, there are several vertical EDI standards. EDIFACT is administered by a working party (WP.4) of the United Nations Economic Commission for Europe (UN/ECE). The EDIFACT syntax rules have been published by the ISO as ISO9735. In [Moo99], it is shown that current EDI standards have the message structure proposed by speech act theory. The current EDI standards are beeing criticized because of a number of problems such as underspecified meaning, idiosyncratic use and inflexibility. In 1999, a major initiative has been launched to replace the outdated EDI message syntax by a more flexible XML-based framework called *ebXML*.

Enterprise Application Integration (EAI) refers to the problem of how to integrate the increasing number of different application systems and islands of information an enterprise has built up over many years. The EAI problem also arises through the formation of a virtual enterprise or from merging two companies. While the integration of various islands of information including databases, sequential files, and spreadsheets, may be achieved through *data federation systems*, the interoperation between different application systems requires an asynchronous message exchange technology, also called *message-oriented middleware (MOM)*. In addition, a message translation service is needed to transform the messages sent by one application to the message language of another application. An application-independent EAI message language, called *Business Object Documents*, is proposed by the Open Application Group. The integration of applications across enterprise boundaries is also called 'Enterprise Relationship Management'.

Enterprise Resource Planning (ERP) systems are generic and comprehensive business software systems based on a distributed computing platform including one or more database management systems. They combine a global enterprise information system covering large parts of the information needs of an enterprise with a large number of application programs implementing all kinds of business processes that are vital for the operation of an enterprise. These systems help organizations to deal with basic business functions such as purchase/sales/inventory management, financial accounting and controlling, and human resources management, as well as with advanced business functions such as project management, production planning, supply chain management, and sales force automation. First generation ERP systems now run the complete back office functions of the worlds largest corporations. The ERP market rose at 50% per year to $8.6 billion in 1998 with 22,000 installations of the market leader, SAP R/3. Typically, ERP systems run in a three-tier client/server architecture. They provide multi-instance database management as well as configuration and version (or 'customization') management for the underlying database schema, the user interface, and the numerous application programs associated with them. Since ERP systems are designed for multinational companies, they have to support multiple languages and currencies as well as country-specific business practices. The sheer size and the tremendous complexity of these systems make them difficult to deploy and maintain.

Entity-Relationship (ER) Modeling A conceptual modeling method and diagram language based on a small number of ontological principles: an information system has to represent information about *entities* that occur in the *universe of discourse* associated with its application domain, and that can be uniquely identified and distinguished from other entities; entities have *properties* and participate in *relationships* with other entities; in order to represent entities in an information system, they are classified by means of *entity types*; each entity type defines a list of (stored and virtual) *attributes* that are used to represent the relevant properties of the entities associated with it; together, the values of all attributes of an entity form the *state* of it; in order to represent ordinary domain relationships (or *associations*) between entities, they are classified by means of *relationship types*; there are two designated relationships between entity types that are independent of the application domain: *specialization* (subclass) and *composition* (component class).

ER modeling was introduced in [Che76]. In its original form, it included the *primary key* concept as its standard naming technique, but did not include specialization and composition. The primary key standard naming technique proved to be inadequate since a standard name should be a unique identifier which is associated with an entity throughout its entire life cycle implying that it must be immutable. However, the basic idea of ER modeling does not depend on the primary key concept. It is also compatible with the *object identifier* concept of OO systems and ORDBs. This implies that ER modeling does not preclude the possibility of two distinct entities having the same state. It is therefore justified to view OO information modeling, such as UML class diagrams, as inessential ex-

tensions of ER modeling, and to regard ER modeling as the proper foundation of information modeling.

Information System (IS) An IS is an artifact (or technical arrangement) for efficiently managing, manipulating, and evaluating information-bearing items such as paper documents, ASCII text files, or physical objects. Today, especially in enterprises and other large organizations, there are more and more computerized ISs implemented by means of DBMS technology. One may distinguish between private, organizational and public ISs. Typical examples of a private IS are personal address databases and diaries. The major paradigms of an organizational IS are transaction-oriented database (OLTP) applications (such as ERP systems) and query-answering-oriented data warehouse (OLAP) applications. Typical examples of a public IS are libraries, museums, zoos, and web-based community ISs.

Integrity Constraints are sentences which have to be satisfied in all evolving states of a database (or knowledge base). They stipulate meaningful domain-specific restrictions on the class of admissible databases (or knowledge bases). Updates are only accepted if they respect all integrity constraints. The most fundamental integrity constraints are value restrictions, keys and foreign keys (or referential integrity constraints).

Interoperability denotes the ability of two or more systems to collaborate. At a lower level, this concerns the ability to exchange data and to allow for remote procedure calls from one system to another. At a higher level, it requires the ability to participate in the asynchronous exchange of messages based on an application-independent language (such as KQML, or FIPA-ACL).

Message-Oriented Middleware (MOM) denotes a type of software systems for managing transactional message queues as the basis of asynchronous message passing. Well-known products include IBM MQSeries and Sun JMQ. A standard MOM application programming interface for Java, called *Java Messaging Service (JMS)* has been proposed by Sun.

Message Transport An abstract service provided by a MOM system in the case of EAI, or by the agent management platform to which the agent is (currently) attached in the case of a FIPA-compliant interoperability solution. The message transport service provides for the reliable and timely delivery of messages to their destination agents, and also provides a mapping from logical names to physical transport addresses

OLAP Application Online Analytical Processing applications allow to evaluate large data sets by means of sophisticated techniques, such as statistical methods and *data mining* techniques. They typically run on top of a data warehouse system.

OLTP System Online Transaction Processing systems are able to process a large number of concurrent database query and update requests in realtime. The information

technology part of a business transaction is called an *online transaction*, or simply 'transaction'. It is performed through the execution of an application program that accesses one or more shared databases within the business information system. A transaction is a complex update operation consisting of a structured sequence of read and write operations. Ideally, a transaction satisfies the ACID properties. Business information systems are primarily OLTP systems. In almost every sector – manufacturing, education, health care, government, and large and small businesses - OLTP application systems are relied upon for everyday administrative work, communication, information gathering, and decision making. The first OLTP application in widespread use was the airline reservation system SABRE developed in the early 1960s as a joint venture between IBM and American Airlines. This system connects several hundred thousand nodes (user interface devices) and has to handle several thousand update request messages per second

Object-Relational Databases (ORDBs) have evolved from relational databases by adding several extensions derived from conceptual modeling requirements and from object-oriented programming concepts. One can view the evolution of relational to object-relational databases in two steps. First, the addition of abstract data types (ADTs) allows *complex-valued tables*. ADTs include user-defined base types and complex types together with user-defined functions and type predicates, and the possibility to form a type hierarchy where a subtype of a tuple type inherits all attributes defined for it. Second, the addition of object identity, object references and the possibility to define subtables within an extensional subclass hierarchy allows *object tables*. There are two notable differences between object-relational databases and object-oriented programming. First, object IDs in ORDBs are logical pointers. They are not bound to a physical location (like C++ pointers). Second, in addition to the intensional subtype hierarchy of the type system, ORDBs have an extensional subclass (or subtable) hierarchy that respects the subtype relationships defined in their type system. ORDBs allow the seamless integration of multimedia data types and large application objects such as text documents, spreadsheets and maps, with the fundamental concept of database tables. Many object-relational extensions have been included in SQL-99.

Object-Oriented Database (OODB) Historically, the successful application of object-oriented programming languages such as Smalltalk, C++ and Java, has led to the development of a number of so-called 'object-oriented database systems' which support the storage and manipulation of persistent objects. These systems have been designed as programming tools to facilitate the development of object-oriented application programs. However, although they are called database systems, their emphasis is not on representing information by means of tables but rather on persistent object management. Any database concept which is intended as an implementation platform for information systems and knowledge representation must support tables as its basic representation concept on which query answering is based. Tables correspond to extensional predicates, and each table row corre-

sponds to a proposition. This correspondence is a fundamental requirement for true database systems. If it is violated, like in the case of OODBs, one deals with a new notion of database system, and it would be less confusing to use another term instead (e.g. persistent object management system) as proposed by [Kim95].

Ontology An ontology explicitly specifies the terms for expressing queries and assertions about a domain in a way that is formal, objective, and unambiguous. This includes the stipulation of terminological relationships and constraints in order to capture key aspects of the intended meaning of the specified terms. An ontology is implicitly defined by a conceptual model (such as an ER or UML model). Communication between agents can only be successful if it is based on a common (or shared) ontology.

Protocol A protocol defines the admissible patterns of a particular type of conversation or interaction between agents. Notice that an interaction protocol refers to the communication acts and high-level actions available to agents, whereas a networking protocol refers to message transport mechanisms such as TCP/IP.

Reaction Rule SQL databases support a restricted form of reaction rules, called *triggers*. Triggers are bound to update events. Depending on some condition on the database state, they may lead to an update action and to system-specific procedure calls. In [Wag98] a general form of reaction rules, subsuming production rules and database triggers (or 'event-condition-action rules') as special cases, was proposed. Reaction rules can be used to specify the communication in multidatabases and, more generally, the interoperation between communication-enabled application systems.

Relational Database (RDB) Already in 1970, Edgar F. Codd published his pioneering article "A Relational Model of Data for Large Shared Data Banks" in the *Communications of the ACM*, where he defined the principles of the relational database model. This was the first convincing conceptualization of a general purpose database model, and it is not an accident that it relies on formal logic providing a clear separation of the conceptual user interface and the underlying implementation techniques. In the mid-eighties, IBM presented *DB2*, the first industrial-strength implementation of the relational model, which continues to be one of the most successful systems today. There are now numerous other relational DBMSs that are commercially available. The most popular ones include Informix, Oracle, Sybase and Microsoft SQL Server. To a great extent, the overwhelming success of these systems is due to the standardization of the database manipulation language SQL originally developed at IBM in the seventies.

While most well-established information processing systems and tools such as programming languages, operating systems or word processors have evolved from practical prototypes, the unprecedented success story of the relational database model is one of the rare examples where a well-established and widely used major software system is based on a formal model derived from a mathematical theory (in

this case set theory and mathematical logic). Conceptually, a relational database is a finite set of finite set-theoretic relations (called 'tables') over elementary data types, corresponding to a finite set of atomic propositions. Such a collection of atomic sentences can also be viewed as a finite interpretation of the formal language associated with the database in the sense of first order predicate logic model theory.

The information represented in a relational database is updated by inserting or deleting atomic sentences corresponding to table rows (or tuples of some set-theoretic relation). Since a relational database is assumed to have complete information about the domain represented in its tables, if-queries are answered either by yes or by no. There is no third type of answer such as unknown. Open queries (with free variables) are answered by returning the set of all answer substitutions satisfying the query formula.

Rule There are various types of rules: business rules, legal rules, calculation rules, derivation rules, production rules, rules of thumb, reaction rules, and many more.

SQL is a declarative language for defining, modifying and querying database tables. A table schema is defined with the command `CREATE TABLE`... and modified with `ALTER TABLE`... The content of a table can be modified by either adding, deleting, or changing rows using the commands `INSERT INTO`..., `DELETE FROM`... and `UPDATE`... Simple queries are formed with the expression `SELECT` *columns* `FROM` *tables* `WHERE` *condition*. Such a query combines the cross product of *tables* with the selection defined by *condition* and the final projection to the attributes occurring in *columns*. More complex queries can be formed by nesting such `SELECT` statements (using subqueries in the `WHERE` clause), and by combining them with algebraic operators such as `JOIN`, `UNION`, `EXCEPT`. SQL queries correspond to relational algebra expressions and to predicate logic formulas: projection corresponds to existential quantification, join to conjunction, union to disjunction, and difference (`EXCEPT`) to negation. The most recent version of SQL, SQL-99, includes many object-relational extensions, such as user-defined types for attributes, object references, and subtable definitions by means of `CREATE TABLE` *subtable* `UNDER` *supertable*.

TCP/IP is a networking protocol used to establish connections and transmit data between hosts.

Unified Modeling Language (UML) is an established object-oriented modeling standard defined by an industry initiative organized and funded by Rational and led by three prominent figures of the OO modeling community: Booch, Jacobson, and Rumbaugh. UML recognizes five distinct modeling views: the *use-case* view for requirements analysis, the *logical view* for describing the static structure and the behavior of a system, and three implementation views regarding *components*, *concurrency*, and *deployment*. Each of these views is composed of several diagrams. A *use-case diagram* depicts a complete sequence of related transactions between an external actor and the system. The idea is that, by going through all of the actors

associated with a system, and defining everything they are able to do with it, the complete functionality of the system can be defined. UML *class diagrams* are a straight-forward extension of ER diagrams. In addition to conventional (stored) attributes, class diagrams also list the operations of a class which may be functions (derived attributes) or service procedures associated with the class. The behavior of a system is modeled by means of four types of diagram: *sequence diagrams* depict the message exchange between objects arranged in time sequence, where the direction of time is down the page; an alternative way of visualizing the message exchange between objects is offered by *collaboration diagrams* emphasizing the associations among objects instead of the time sequence; *activity diagrams* are used for describing concurrent, asynchronous processing; finally, *state charts* allow to represent the state transitions of a system.

B. About the instructor

Dr. Gerd Wagner is currently a research scientist in the informatics department at the Free University of Berlin. He received an MSc in mathematics and a PhD in philosophy from the Free University of Berlin, and a German Habilitation degree in computer science from the University of Leipzig. He was visiting researcher at the Institute de Recherche en Informatique de Toulouse in 1994 and at Universidade Nova de Lisboa in 1995. His research interests are focused on the foundations of information and knowledge systems and of multiagent systems. Dr. Wagner is the guest editor of a 1997 special issue of the Journal of Applied Nonclassical Logic on *Handling Inconsistency in Knowledge Systems*, and the author of the book *Foundations of Knowledge Systems with Applications to Databases and Agents* (Kluwer Academic Publishers, 1998). In 1999, he initiated and co-organized the first International Workshop on Agent-Oriented Information Systems (http://www.AOIS.org). In recent years, Dr. Wagner has also worked with the enterprise resource planning system BaanERP. He has been involved in several BaanERP customization projects, and has taught *BaanERP Tools* to programmers and consultants.

Bibliography

[AM97] S. Anahory and D. Murray. *Data Warehousing in the Real World.* Addison Wesley, 1997.

[Che76] P. Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[Cod94] E.F. Codd. Adding value to relational and legacy dbms: The olap mandate. *Business Intelligence*, 1994.

[Den71] D.C. Dennett. Intentional systems. *The Journal of Philosophy*, 68, 1971.

[IWK97] W.H. Inmon, J.D. Welch, and G.L. Katherine. *Managing the Data Warehouse.* Wiley & Sons, New York, 1997.

[Kim95] W. Kim. Introduction to part 1. In W. Kim, editor, *Modern Database Systems*, pages 5–17. ACM Press, New York, 1995.

[Moo99] S.A. Moore. Categorizing automated messages. *Decision Support Systems*, 1999.

[Sho93] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[Wag98] G. Wagner. *Foundations of Knowledge Systems – with Applications to Databases and Agents*, volume 13 of *Advances in Database Systems.* Kluwer Academic Publishers, 1998. See http://www.inf.fu-berlin.de/∼wagnerg/ks.html.