Module 11

Software System Design

The primary objective of the design phase is to create a design that satisfies the system functional requirements. The graphical user interface prototype developed earlier is gradually refined and extended with all the details required to satisfy the full spectrum of application business functions.

The construction of this preliminary application prototype provides several advantages to the users. Among other things, it allows them to better visualize the application user interface and its major components. The users try out first-hand interactions with the main application windows and their graphics user interface (GUI) controls. They explore the navigational capabilities provided by the interface and thus have the opportunity to comment on the interface's overall usability and effectiveness.

The creation of this live prototype also provides a solid foundation to identify the detailed functional requirements of the application.

From the user's point of view, the GUI is certainly the most important component of the system. After all, through this interface, they will interact with the system to perform their business functions. However, for the developers, the user interface just represents the tip of the iceberg. In fact, the design and integration of several additional internal system components is required to eventually deliver a robust, fully functional system.

Objectives

After studying this module, you should be able to:

1. Determine GUI design standards and guidelines;
2. Create logical database design;
3. Create physical database design;
4. Design database security and audit schemes;
5. Design shared, reusable software components;
6. Design system interfaces;
7. Design ad hoc report and query environment; and
8. Create system documentation.

For large and complex client-server and Web-database applications, the design issues that are associated with the performance, reliability, robustness, scalability, security, and maintainability characteristics of the system are of paramount importance.

The detailed design of the physical database and the application functional components should be engineered in a way that offers as much flexibility and adaptability as possible. This process must take into consideration the existing enterprise technical architecture. It must also take into account the physical constraints that might be imposed by the technology that was selected to implement the system.

You should properly document any deviation from the original application requirements that might be needed to accommodate specific operational criteria. Discuss thoroughly the alternatives, along with their respective trade-offs, with the users.

During the design phase, the initial system data conversion and testing strategies that were elaborated during the analysis phase are refined with more detailed information. Similarly, detailed test cases are designed to support the different testing stages that will be executed during the construction phase.

Although the linear sequence in which some of the design activities might suggest that one specific activity must be completed prior to start the next one, in reality they can be performed in parallel or in different sequences.

Furthermore, several of these activities can be performed by following an iterative approach, gradually refining the deliverables in incremental steps. On the other hand, certain types of activities might not easily lend themselves to an iterative development approach.

At the completion of each major design activity, a formal walkthrough should be scheduled to review the final deliverables.

Determine GUI Design Standards and Guidelines

This section discusses basic conventions and guidelines that relate to the design of graphical components such as application windows and menus.

Overview

The minimum yet useful set of user interface standards, guidelines, or conventions that should be used to design the various application screen layouts must be thoroughly discussed with all development team members.

This review process is a very crucial step toward ensuring the design of a quality system. This statement is especially true when many design teams are called upon to simultaneously develop different sections of a large and complex client-server or Web application. Failure to clearly spell out the most critical GUI conventions might result in reduced application consistency, usability, and maintainability.

If a minimum level of user interface design consistency is achieved across all corporate applications, then the users will be relieved of the painful task of trying to adapt to a puzzle of different GUI. Moreover, the development team will likely increase the overall efficiency of the users by reducing the likelihood of them making costly errors. Application design consistency should also decrease their level of frustration while they interact with these applications.

Lastly, the learning curve that they must go through while they familiarize themselves with different applications should be minimized significantly since a common GUI will be shared across applications and sustain a predictable behavior. The users should become far more effective and efficient when they use applications that share common "look and feel" characteristics.

The following sections discuss some of the different types of GUI standards and conventions that apply to the design and construction of a large-scale client-server application. It also includes advice and commentary on each major set of proposed standards. The two major standards that are covered include:

- Windows conventions
- Menu conventions

Some of these conventions are also relevant to Web database applications.

It also provides specific guidelines on user interface style definition, screen navigation behavior and on product usability concepts.

The types of hardware and software tools that are used to develop the system will likely influence the GUI design guidelines. Above all, the design standards and guidelines should remain at a very pragmatic level so developers really use them.

Window conventions

The GUI conventions that will be used to design the application windows should at least cover the items that are listed in Table 11-1.

Table 11-1. Windows components that require design conventions

- Placement and format of command buttons
- Placement and format of status elements such as date, time, page number
- Placement and format of messages (i.e., warning, informative messages)
- Placement and format of menu items
- Placement and format of tool bars, scroll bars, control palettes
- Placement, format, and data representation of input fields (i.e., text and numbers), headers, and labels
- Use of colors and fonts for window objects and window backgrounds
- Window screen resolution levels
- Placement and format of multimedia elements such as sounds and video clips
- Placement and format of error messages
- Placement and format of pop-up windows

Several good ideas for designing effective, professional-looking user interfaces can be emulated by carefully studying the most popular commercial client-server and Web applications that exist on the market today. In some instances, if the users of the future applications are already familiar with a particular product that they use regularly, then resorting to developing a similar user interface makes sense in order to take full advantage of that familiarity.

Discussing in detail how to design very effective user interfaces is beyond the scope of this guide. Searching for GUI guidelines that are posted on the world-wide-web can be a very effective way of finding some valuable information on this subject.

Table 11-2 lists some practical conventions that are utilized for the effective design of user-friendly windows.

Table 11-2. General windows conventions and guidelines

Subject	Design Guideline
Number of control buttons	The number of control buttons that are displayed on a window should be limited to avoid confusing the user. Use multiple windows as an alternative to overcrowd a single window with too many controls. The tab folder metaphor can be used to display several tabbed pages that in turn display additional controls.
Controls availability	When controls are not available to users, they should be disabled (i.e., dimmed out).
Controls spacing	Normally, controls should not be clustered physically one against another. White spaces should be inserted between them.
Controls borders	3-D lowered borders are often to make control buttons stand out as opposed to static text, which usually does not display any form of border delineation.
Keyboard support	At a bare minimum, the user should be able to activate every window control feature not only with the mouse but also directly from the keyboard.
Fonts	As a rule of thumb, a unique font should be consistently used for the same set of applications windows. Ideally, this font should be a standard font that is already installed on the user workstation. If required, the same font may be used in a limited number of different font sizes.
Colors	Color should be used with parsimony, such as when some portions of the window must stand out. Careful attention should be given to selecting non-offensive colors that, as much as possible, will not adversely affect the user's default settings.
Amodal window usage	Use non-modal (non-response) windows wherever possible to provide the user with as much control latitude as possible over the presentation interface.
Modal window usage	Use modal (response) windows only when the user must be forced to respond to the information that is displayed in the window.

To a large extent, windows standardization in an application can be achieved through the use of inheritance. With inheritance, a basic set of standard windows can be defined once at the ancestor level and subsequently extended or overridden by an application at a descendant level. It provides more in-depth information on the object-oriented concept of inheritance.

Menu conventions

Table 3 lists simple conventions that are commonly used for the effective design of standard menus in a client-server application. The use of a common "look and feel" menu across multiple applications will allow the users to utilize different applications that offer the same predictable behavior.

Table 11-3. General windows menu conventions and guidelines

Subject	**Design Guideline**
Number of menu elements	The number of elements that are displayed on a menu should be limited to avoid confusing the user with too many options.
Depth of menu elements	When using a cascading menu, no more than one or two lower levels should be utilized in order to avoid user confusion.
Menu element s availability	When specific elements of the menu are not available to users, they should be disabled (i.e., dimmed out).
Menu elements wording	Each element shown on a menu should carry a descriptive name that clearly indicates its purpose. A menu element should not be composed with more than two simple words, preferably a single word only. Standard names or verbs should be used whenever possible, such as File, Edit, Window, Help.
Menu elements placement	The menu elements in an application should always be positioned at the same location on the menu bar, preferably using the most common standard window conventions (i.e., File, Edit, Window, Help, etc.).

Table 11-3 continued

Help menu element	Each menu bar should always contain a help menu element that displays the help information that is specific to the current application.
Keyboard support	Shortcut keys should be used to alternately select a menu item. Accelerator keys should be used to allow quick access to a specific menu element.

ADDITIONAL GUI DESIGN GUIDELINES

GUI physical design limitation. While the developers establish the design conventions that should be applied to the layout of the graphical user interfaces, keep in mind that the display capabilities of different types of user workstations might impose some constraints on the available design options. For example, if some users have display monitors that have lower level resolution capabilities than other users, then you might need to use the lowest level of resolution as a common denominator to design the application windows. Although less prevalent today, a few users out there may have monochrome monitors. In such a case, the design of the windows should not emphasize the use of colors to highlight special elements of the user interface.

Another constraining designing factor might be the display size of the monitors that will be utilized by the users. Will the users utilize workstations that all have the same display size or will they use several types of workstations that have different monitor display sizes?

Multi-platform design limitations. If the application is intended to be used by an international audience or in a multilingual setting, then the conventions for the design of the user interface will need to take into account the various cultures and languages of the different countries where it will be used. Cultural differences might be reflected in the choices made while designing the user interfaces.

Help system. The standards that apply to the help system should include guidelines on context sensitivity, different levels of help provided (application level, screen level, field level), and format and general placement of help information.

Create Logical Database Design

This section discusses the detailed steps required to transform the application data model into a sound logical database design. It also provides suggestions and tips on how to best perform this process.

In this activity, the business data model is transformed into a first-cut logical database design that can satisfy the data access requirements of the application.

Once this transformation process is done, the logical database design structures are then gradually refined to satisfy the specific operational requirements of the application that relate to performance, security, and data distribution considerations.

Finally, the initial logical database structures might require specific enhancements to accommodate any physical limitations or constraints that might be imposed by the selected database management software and target technology architecture.

Translating the data model into a first-cut logical database design

Prior to converting the data model into a set of logical database structures, verify that the data model encompasses all the data needs for the application at hand. Similarly, all the unique identifiers for each entity should be clearly identified on the business data model.

Once this verification step is done, then you can derive a first-cut logical database structure from the entity-relationship diagram. The major tasks that are involved in this mapping process are listed in table 11-4.

Table 11-4. Logical database design task

Entities with no subtypes	• For each entity documented in the entity-relationship data model, create a single table that contains one column for each specific attribute of the entity. • For each relationship that is characterized by a one-to-one association, either add the primary key of one of the data entities as a foreign key column in the other entity's table or implement both entities in one table. • For each relationship that is characterized by a one-to-many association, add a foreign key column in the table implementing the entity at the "many" side of the relationship. • For each relationship that is characterized by a many-to-many association, create an intersection table whose primary key is a concatenation of the respective primary keys of the two entities involved in the many-to-many relationship.
Entities with subtypes	• For complex entities that contain several subtypes, design a table structure using various options, such as: • Create a unique table that implements together the entity supertype and all the subtypes • Create separate tables for the entity supertype and each entity subtype • Create separate tables for each entity subtype but not the supertype

The particular characteristics of each column in a table are directly derived from those that were documented with the attributes for each specific entity, such as length, domain, data type, and so forth.

Finally, each unique identifier of an entity is usually transformed directly into a primary key identifier for the corresponding table. However, in some specific instances, you may want to implement an artificial unique key. For example, if you can't be certain that the primary key will ever need to be modified, then add an artificial unique key. Another situation might be when the primary key is a composite of several attributes and the entity itself is referenced by several other entities. In such a context, you might want to implement an artificial unique key to save on disk space and possibly eliminate the need to concatenate foreign keys and related concatenated indexes.

Determining the initial dataviews

A data view allows the database designer to create a different representation of the data that is stored in the database tables. Views can be used for different purposes, such as to restrict accesses to a predetermined set of rows and/or columns in a table or to facilitate the work of the programmers by allowing them to select data from several tables without forcing them to do a join.

The initial requirements for data views can be determined by carefully examining the data access requirements of the application that are documented in the data access matrices that were developed earlier, during the analysis phase.

Augmenting the Initial Logical Database Structures with Additional Database Objects

The logical databases structures can be refined by introducing supplementary database objects that might be required to accommodate specific design requirements, such as different application reference code tables or artificial keys, for example. You might also need to create additional tables or columns for data journalizing needs, such as the addition of date and time data as well as user IDs. Another example might be the addition of various flags to indicate that the records in a table have been processed in some manner. Lastly, you might consider adding special fields that contain summarized data.

Denormalization

As discussed in earlier module, the normalization of the data model helps to eliminate the major difficulties that are associated with updating redundant information and helps to improve the integrity of the data that is stored in the databases.

On the other end, the normalization process can generate a very large number of individual tables for a large database, some of which might need to be joined together to satisfy specific cross-table queries, hence producing a non-negligible amount of overhead in terms of CPU cycles.

Besides simplicity and maintainability, good database design is often measured by its ability to provide good performance. Consequently, some very specific situations might warrant the need to denormalize some of the logical database structures to build a major application that is more responsive, such as the potential cases indicated in the table below.

In all instances, be sure to document the database structural changes that are caused by a denormalization process.

Table 11-5. Denormalization considerations

Derived data	Derived data entails storing in the database specific data that can be produced from other data, such as the derived field, for instance total_invoice_amount.
Duplicated data	Sometimes you might need to duplicate some specific data from one table to another, to improve performance, for example.
Summary data	Sometimes, you might need to store various types of summary or historical data in the database to ensure a faster response time on certain queries. A typical example would be queries that very frequently access detailed data to calculate a sum. In such a context, it is recommended storing computed vales immediately when the data is created at a higher level in the database.

Refining and analyzing the critical data access patterns

The data access matrices that were created during the analysis phase should be revisited and refined with up-to-date volumetric information. The detailed data access paths of the most critical online transactions should be carefully analyzed to ensure that the logical database structures can accommodate them. In the case of a critical batch program, its overall job turnaround time might become critical, especially when the run time of the batch program is constrained by a limited time window.

A critical factor often used to identify the most efficient data access design solutions is estimating the number of rows to be retrieved as a result of each call. From there, the number of I/Os required to access those rows is estimated. Then the associated estimates for CPU time are often compared to identify the best design alternative. Simply plotting the estimated number of I/Os for each data access in a spreadsheet can be a good approach. Another option is to use analytical modeling software tools provided by database or third party vendors. In some cases, a relatively simple prototype may be required to issue SQL calls against real physical databases that contain realistic but artificial data, in order to assess and evaluate the best design alternatives. For very large databases, full-scale prototypes may be essential to obtain realistic estimates of program execution times.

Several alternative designs might be available to improve the response time of the most critical system transactions. In some instances, the logical database structures might need to be modified to accommodate some specific system transactions and associated data access patterns. Sometimes the system transactions and associated data accesses themselves might need to be redesigned to meet the stated response time objectives.

The database designer, with the application developers and even the users if necessary, should also revise the cardinality figures that were documented between the relations shown in the entity-relationship diagram.

The overall data volume estimates should be refined by taking into consideration the enterprise data archival policies that apply to the information contained in the logical database structures. How many years will production data be accessible online, and under which conditions will they be archived? Furthermore, the volume estimates should not only include average occurrences for each specific entity but also the estimated minimum and maximum number of occurrences for an entity. The revised data sizing estimates are used to verify the hardware disk space requirements that were forecasted during the analysis phase.

In some cases, you might also want to verify how often the maximum number of entity occurrences can be encountered while processing the application data. Obtaining this information might be imperative, in order to handle unforeseen situations in large databases. For example, the database structure might be able to support the processing of an average number of entity occurrences but can fail to adequately support the processing of a few occasional but very large numbers of entity occurrences stored in a given table.

Create Physical Database Design

This section describes the detailed steps involved in creating the physical database design. It also provides tips and guidelines on how to perform this highly iterative process.

Overview

The logical database structures are transformed into a set of physical database structures. Then each physical database structure is progressively refined and optimized to satisfy the complete data access and service-level performance requirements that were identified for the application.

If a preliminary version of the physical database structures was already constructed during the analysis phase in order to support the creation of alive application prototype, then the physical database structures are simply refined to satisfy the additional database requirements that have been identified since then.

Database design factors

Several application requirements can affect the important decisions that must be weighed by the database administrator to derive an optimal database structure. Some of the most common design factors, which will likely influence the construction and fine-tuning of the physical database are listed in table 11-6.

The various database SQL constructs that are illustrated throughout the next sections apply primarily to the design of an SQL-based database.

Database design factors

Most relational databases are usually partitioned into several logical storage units. In some commercial SQL-based databases, these logical storage units are called table spaces. The system table space contains the data dictionary. It also contains the names and addresses of all the other table spaces, tables, indexes, and clusters. A table space has a fixed size that can be expanded by the database administrator when it gets full.

Table 11-6. Physical database design factors

- The volume of live data that must be stored in the database
- The volume of online and batch database transactions that must be processed per unit of time
- The various audit, security, and integrity requirements that must be enforced by the database
- The database backup and recovery requirements, which must be completed in the time windows available
- The database files archival requirements
- The database availability requirements
- The number and different groupings of data objects, data attributes, and relationships that must be supported by the physical database structure and their complexity
- The number of business rules that must be enforced by the database and their complexity
- The proposed application data access sequences
- Efficient database loading

One of the first design tasks that the database administrator must perform is to simply define and uniquely name the database and then set up the required database table space.

The database name is created using the 'CREATE DATABASE' DDL command, as shown here. For simplification purposes, this example uses defaults for all the different types of arguments that can be appended at the end of the CREATE command statement.

```
CREATE DATABASE Database_1
```

The CREATE TABLESPACE DDL command is used to assign one or more files to a data table space, as shown in the following example:

```
CREATE TABLESPACE tabspace_1
DATAFILE 'disk' SIZE 25M
DEFAULTS STORAGE (INITIAL 10K NEXT 40K
                                    MINEXTENTS 1 MAXEXTENTS 100
                                    PCTINCREASE 15);
```

The amount of work involved in estimating the sizing of the table spaces and in the functional grouping of the database tables, indexes, and roll-back segments to a given table space will vary from one database to another.

Sufficient free space should also be allocated for each table space based on the percentage growth of the data they store.

In all cases, the size of the table spaces should be validated against the database backup and recovery plan. Although this statement might sound subjective, a table space is too large if the amount of time needed to recover it is unacceptable (i.e., it does not fit within the backup and recovery window that has been defined as acceptable for recovery).

Creation of the database tables

For the vast majority of relational database management systems, the table is the most fundamental unit of data storage. The database tables house all the data that can be directly accessed by the users. A database table is defined with a unique table name and contains a certain number of columns. The following characteristics must be assigned to each specific column, in an appropriate fashion:

- column name
- data type (varchar, date, number, etc.)
- data column size or width
- whether the column value is mandatory (i.e., NULL or NOT NULL value)
- whether restrictions are imposed on the permissible values or range of values that are allowed for the column (i.e., CHECK constraints)

Given that the table spaces have been created for the databases, the various applications database tables are now affiliated with their corresponding table space. The DDL command 'CREATE TABLE' is used for this purpose, as illustrated in the following example, where an 'Employee' table is created:

```
CREATE TABLE Employee (
Emp_numbernumber,
Emp_namevarchar (30),
Emp_hired_datedate,
Emp_addressvarchar (150))
TABLESPACE tabspace_1;
```

An important factor that must be taken into consideration by the database designer is the association of tables to table spaces. Following are some basic rules of thumb that might help to optimize the general performance of the database:

- Store index and data components in table spaces on different disks
- Segregate tables with a high insert/delete rate from more stable tables in size
- Place tables with high access rates in table spaces on different disk drives

Creation of the foreign key(s)

A foreign key is a column or combination of columns that contain some primary key value(s) from another database table. A foreign key constraint, also referred to as a referential integrity constraint, indicates that the values of the foreign key must match the actual values of the primary key in another table. The next example illustrates how to define a foreign key constraint in the *'Employee'* table:

```
CREATE TABLE Employee (
Emp_numbernumber CONSTRAINT pk_employee PRIMARY KEY,
Emp_namevarchar (30),
Emp_hired_datedate,
Emp_addressvarchar (150),
Emp_dept_noNUMBER (4),
CONSTRAINT fk_dept_no
    FOREIGN KEY (Dept_no)
    REFERENCES Dept(Dept_no))
TABLESPACE tablespace_1;
```

The referential integrity constraint *fk_dept_no* enforces the rule stating that any employee in the *Employee* table must work in a department defined in the department table identified as '*Dept.*'

Creation of the table indexes

An index is an object that holds an entry for each value that occurs in an indexed column of a table. Database indexes can improve significantly the performance of database searches against specific table columns. They provide fast and direct access to table rows when the rows are searched in index column sequence, using DML commands such as SELECT, UPDATE, and DELETE.

The data access specifications that were documented in the data access matrices should be scrutinized carefully to identify queries that are performed on data fields that are not primary keys– for example, selecting a customer based on its name and address because the customer number is known.

Indexes can be created on one or multiple table columns. However, they cannot be created on a view. They are automatically generated and maintained by the database management system. Consequently, indexes are completely transparent to users and applications. The next example illustrates the creation of a database index against the *Emp_name* column in the *Employeetable*:

```
CREATE INDEX idx_emp_name ON Employee (Emp_name);
```

Table 11-7 provides some tips on database indexes.

To conclude, the database administrator should carefully weigh the advantages and disadvantages of using database indexes. Generally speaking, database indexes should be built only for non-key table columns that are searched regularly or when storage and processing overhead is not an issue.

Table 11-7. Observations on database indexes

- For a large database, first creating the table and inserting all the required rows in it and then creating the database index is generally faster. If the index is created prior to the insertion of the rows, most relational databases will update the index each time a row is inserted into the table. This situation can cause some performance problems when several tables are populated with a significant number of records.

- The use of database indexes will gradually become more and more attractive as the size of the database tables increases significantly. For instance, the average time needed to locate a particular employee name within a huge table will be dramatically reduced via searching through a database index. On the other hand, small tables that contain 10 to 30 rows might not need to be indexed.

- Indexes can take up a significant amount of space for very large indexed database tables. Hence, you need to index only specific non-primary key target columns that are frequently accessed by several database queries.

- Applications with frequent and massive INSERT, DELETE, and UPDATE operations can be slowed down by the abuse of indexes, due to the fact that the execution of each of these database operations against the indexed columns in a table will force the database management system to maintain both the index pointers and the data itself.

- If a query accesses more than 25 to 35 percent of all rows in a table, then performing a full table scan might sometimes be faster then using indexes. In such a case, further investigation might be required to determine which approach provides the best results.

- As a rule of thumb, the database indexes should be physically stored separate from the data in order to optimize database performance.

Creation of database clusters

Clusters provide an alternative method of physically storing table data. A cluster can contain one or multiple tables that together share one or more columns. Clusters, like indexes, are transparent to the users and application developers. The data that is stored in a cluster are accessed in the same manner as the data stored in a non-clustered table.

The judicious use of clusters can favorably increase system performance, especially in situations where the clustered tables are frequently queried in joins. This improvement is due to the fact that the rows that are common to the joined tables are retrieved with a minimum amount of Input-Output (I/O) operations.

On the other hand, the use of clusters is likely to decrease the performance of full table searches. Hence, the potential advantages and disadvantages associated with the use of clusters must be carefully weighed in light of the type of operations that must be performed on the data tables.

Creation of database views

A view is customized, logical representation of some of the data that is contained in one or more database tables. Unlike a table, a database view does not actually store physical data. Rather, the view derives its data from the multiple tables that support it. The definition of the database view is permanent, whereas the data content supported by view is always re-created at execution time.

In many aspects, views can be manipulated very similarly to permanent tables, such as:

- access authorization can be granted to a view just like for a permanent table
- queries (i.e., SELECT) can be performed on views
- operations such as UPDATE, DELETE, and INSERT commands can be performed on a view (with some restrictions)
- views can be created based on other views

Views also have some limitations when compared to permanent tables, such as:

- indexes cannot be created on views
- integrity constraints and keys cannot be assigned to views
- views are re-created each time they are invoked by an application, since they are derived from permanent tables and do not retain copies of that information
- operations such as UPDATE, DELETE, and INSERT commands can be performed on a view but with some restrictions

A view can be created to unify data that comes from several tables. Views are used often to precisely tailor the access to the data in the database. The construction of database views might serve several purposes, such as those listed in the table below.

Table 11-8. Various utilizations of database views

- To provide improved security by not allowing access to database information that is not relevant to specified users
- To improve usability by hiding data complexity and presenting to users and developers data in form that is simpler to comprehend and manipulate
- To allow users and developers to retrieve data stored in multiple tables, without needing to perform a join operation
- To present data in different perspectives from the permanent table
- To improve consistency by centralizing in the database the definition of common queries that perform extensive calculations with the data that is stored in the tables. By saving the query as a view, the calculations can be executed automatically every time the view is invoked

Automated database design tools

Several automated tools exist on the market to facilitate the creation and enhancement of multiple types of logical and physical database models. These automated data modeling tools often support a large variety of commercial relational database management systems by generating the appropriate data definition language (DDL) statements. They can also manage various database objects such as stored procedures, views, and triggers.

Furthermore, several of these tools can also reverse engineer several types of physical relational databases directly from their data definition language (DDL) or from the database files that use the Open Database Connectivity (ODBC) protocol. Following are the addresses of vendors' Web sites that offer automated data modeling and database designing tools.

http://www.logicworks.com(Erwin/ERX)
http://www.oracle.com(Oracle Database Designer)
http://www.sybase.com(PowerDesigner)
http://www.popkin.com(SA/Data Architect)
http://www.visible.com(Visual Analyst Workbench)
http://www.infomodelers.com(InfoModeler)

Distributed database technology and design considerations

This section describes some high-level concepts that relate to distributed databases, including:

- Different database distribution techniques
- Database replication mechanisms
- Synchronous or asynchronous replication mechanisms
- Database replication timing mechanisms
- Database replication conflict resolution mechanisms
- Full or partial data refresh mechanisms

It also discusses some basic guidelines that you should consider when designing distributed physical databases. Finally, presented are different case scenarios where the database replication technology can be advantageously used.

Different database distribution techniques

Table 11-9 describes three different techniques that can be used to distribute data across different locations.

Table 11-9. Various database distribution techniques

Segmentation	Database segmentation is the physical partitioning of a database into a set of separate physical database tables, which then can be stored at different sites. The database tables are not duplicated but only implemented at different sites.
Extraction	For many years, custom database extraction programs have been created by developers to read the data that is stored in databases and create copies of that data either in a flat file format or other formats, for further processing.
Database replication	Database replication allows the automatic generation of several copies of either full or partial database tables. The data replication process is usually performed with built-in database management system (DBMS) utilities. Most relational DBMS vendors also offer sophisticated database replication technologies that support various database objects such as triggers, stored procedures, views, and indexes.

Out of the three techniques that were described in the table above, the database replication technique is becoming increasingly popular and is supported by all the large relational database management system vendors. The following section briefly discusses the database replication concept.

Database replication

Database replication is a technique that is frequently used to distribute data from a master database to additional copies that typically are stored at different local sites.

Table 11-10 shows some of the most common benefits that are associated with the database replication technology.

Table 11-10. Database replication benefits

- Multiple copies of data can be downloaded to branch offices, where users can access the data locally in a more efficient manner
- Multiple copies of data can help load-balance the traffic on heavily used networks across different locations
- Multiple copies of data can be used for disaster recovery situations
- Database replication can be used to data warehouses or data marts

Fundamentally, two basic types of database replication exist: synchronous and asynchronous replication.

Synchronous replication

Synchronous replication is often used in high-end distributed client-server applications where data consistency, availability, and concurrency are of the utmost importance, such as for a sophisticated airline reservation systems, for example. Also known as "real-time" replication, synchronous replication ensures that the application data is always up-to-date and in-sync at all locations that house a database copy.

Synchronous replication relies on a special two-stage hand-shaking technology protocol called two-phase commit. In the first phase, all the database servers that need to be updated simultaneously proceed with a synchronized locking of data (i.e., "get ready" and "I'm ready" confirmation tasks). In the second phase, all the slave database servers process the update request and do a commit as soon as the master database server instructs them to proceed. The rule is simple: either all the databases must be updated at once or none of them are updated. A failure at any particular node in the distributed network will force a rollback of the update transaction at all nodes.

Asynchronous replication

With asynchronous replication, the different database copies are allowed to become slightly out of sync with each other, usually for what the business area might consider to be an acceptable period of lag time. Over time, though, all the different database copies eventually converge to the same data values at all the local sites where the database replicas reside.

Asynchronous database replication is not recommended when the business application absolutely needs to know whether a database update has been successfully completed at all the different sites before it can resume its operations.

Different propagation timing mechanisms

With the asynchronous propagation mechanism, the database updates are stored in a deferred transaction queue. At an appropriate time, the changes are subsequently transmitted to each remote location that must receive the data. The selected RDBMS should have a mechanism that ensures that the database transactions are not eliminated from the deferred transaction queue until they were successfully replicated to each particular remote site. With such an approach, no transactions are ever lost, even in situations in which the network is temporarily down.

The database administrator controls the time at which the data will be transferred to the other database remote locations. For business applications that require almost a continuous data replication scheme, the database administrator can choose a time interval that will propagate the data very close to a real-time mode, usually within a few seconds.

For business applications that do not require a near real-time mode of operation, the database administrator can schedule the replication events at different time intervals, such as every minute, hour, day, or even week, depending on the specific need of the application.

If the application needs allow it, launching the propagation of data during the night might be convenient, at a time when the network is not overloaded and when the communication costs are usually at their lowest possible rates.

The data replication process can also be triggered by an event-based scheme, such as data value changes, for example, instead of a regular time interval scheme.

Finally, another option is to kick off the replication of data on demand. The on-demand (a.k.a. administrator-initiated) replication scheme might be particularly attractive for applications that support multiple small remote locations that regularly connect to a central database to refresh their database copies. The same approach could also be used for mobile users who are equipped with laptops.

Conflict resolution

In a distributed database environment where all the updates need to be simultaneously applied to the same data across all database copies, update conflicts can sometimes occur. Most distributed RDBMSes offer several options to detect and help resolve these potential conflicts.

Conflict resolution routines are important in a distributed database environment to ensure data convergence. The table below enumerates some of the standard pre-defined conflict resolution routines that are typically used by most commercially available distributed RDBMSes. The database administrator should opt as much as possible for conflict detection and resolution methods that minimize the need for manual interventions, either from the database administrators or the users themselves.

Table 11-11. Simultaneous update conflict detection and resolution routines

• Site priority	• Priority group
• Latest timestamp	• Overwrite
• Earliest timestamp	• Discard

Also, the database administrators should be able to set up their own conflict resolution routines, in situations where they need to address complex application-related business rules.

Full data refresh versus partial data refresh

Basically, you have two options to replicate data, namely: full or partial refreshes.

A full refresh is performed when the entire content of the database is transferred from the master database to a local copy. Full database refreshes can be suitable for small database applications. However, full database refreshes might not be suitable for large databases, since the network might not be able to support the high volume of data that must be transferred between the different sites.

A partial refresh is performed when only the rows that were updated in the master database are replicated over to the local database copies.

Some of the factors that will influence the selection of a full data refresh approach versus a partial refresh approach appear in the table below.

Table 11-12. Factors influencing the distributed database refresh strategy

- The size of the database tables must be replicated
- The volume of data that is modified and its frequency
- The network bandwidth capacity
- The frequency of the replication process
- The need to maintain associated tables all in sync, timewise, to maintain business integrity

Distributed database considerations

The physical placement of data at different locations can cause several design-related challenges that must be satisfactorily addressed prior to moving forward with the implementation of this strategy.

Table 11-13 lists some of the major points that should be considered by the database administrator and the application development team during the final design stage of a distributed database.

Ideally, the database tables should be stored as closely as possible to where the data is created or where the data is used the most often. If the same data is heavily used at more than one location, then a database replication scheme can be used to create additional local database copies.

Table 11-13. Distributed database design considerations

- What are the types and amount of data that need to be moved across the network to other sites?
- Will only the database changes be propagated or the entire database tables?
- Will the data move one way or in a bi-directional manner?
- What are the peak loads?
- Who owns the data?
- Where are the user groups located that will access the data?
- How many database servers need to be updated?
- What are the system transactions that create the data? Update the data? Where will they be used?
- What volume of transactions is performed at each different site per unit of time and against which database tables?
- What type of replication mechanism will be used? Synchronous or asynchronous?
  - o If asynchronous database replication is used, how up-to-date do the remote sites' database copies need to be?
  - o Might multiple time zone constraints affect the scheduling of database updates between the various local sites? What are the data usage patterns by time zones?
  - o Can the users work with the data is slightly out-of-date?
  - o Can the enterprise network bandwidth sustain the required data transfer rates?
  - o What mechanism is available to detect the unavailability of the network or the server at a remote site?
- If the network is down between different sites, how will the replicated data be stored for later transmission when the communication facilities are restored?
- In the case of a communication or server failure, how will the initiating application acknowledge and appropriately respond to these failure events?
- What type of processing will be allowed at the local sites when the network is not available?
- How will the database backup and recovery strategy be affected by the distributed data strategy when:
  - o The data resides in different geographical sites?
  - o The data is stored on different computer platforms?
  - o Will the database replication process be performed between databases from different vendors (i.e., cross-platform replication) ?
- Will the database replication process involve complex data types such as image, sound, or video data?

An overview of different types of database replication applications

This section presents a variety of case scenarios in which the database replication technology can be advantageously used to satisfy the business needs for data distribution that were identified earlier during the analysis phase.

Case 1: Central headquarters to branch offices read-only database replication

In this scenario, the data contained in a main database is downloaded from the central headquarters office to multiple branch offices. The data replication process is performed at regular intervals. The users at the branch offices can access the local database replicas only in a read-only mode.

Case 2: Branch offices to central headquarters database replication

In this scenario, the database data is replicated from each branch office back to the central headquarters. The replication process is performed at regular intervals. The data that is uploaded by each branch office to the headquarters office is consolidated in a central database. The consolidated data is then analyzed and processed by the central office for different financial trend and statistical analysis purposes.

Case 3: Database replication to transfer operational data to a data warehouse environment

In this scenario, the data that are stored in the production operational databases are periodically replicated in the data warehouse databases. The databases in the data warehouse environment hold summarized information such as daily, weekly, monthly, or yearly summarized or historical data.

Case 4: Database replication to support mobile users

In this scenario, the mobile users can download data from the corporate databases with the use of a database replication scheme. Later on, the mobile users connect to the central corporate databases and upload the modified data that was stored in their local database replicas back to the corporate databases.

You can think of several database replication scenarios for mobile users, depending on the users' needs. The one example provided here illustrates a potential use of database replication.

Case 5: Database replication for disaster recovery

In this scenario, complete copies of the corporate databases are periodically replicated to a distinct, secure site, at pre-defined intervals. In the event that a major disaster occurs at the central site, the database replicas can then be used to restore the corporate databases with the data that was previously copied the last time the replication process was done prior to the disaster.

Design Database Security and Audit Schemes

A preliminary investigation of the different classes of data that must be manipulated by the client-server or Web database system was conducted with the users earlier during the analysis phase. The end result of this study was a broad assessment of the general security measures that should be enforced to control the accesses to each distinct category of data stored in the database and who should have access to what functions.

In this current activity, specific users or group of users will be provided with different types of access rights to a variety of database objects.

During the design phase, the specific data confidentiality requirements can be addressed further by elaborating multiple database access controls schemes. Most often than not, two major categories of data access controls can be used to satisfy the database security needs in a client-server environment, namely: discretionary or mandatory access controls.

Discretionary access controls

The database administrator uses the discretionary access controls to regulate all user accesses to the data that is stored in the database. Typically, granting database accesses is done by means of implementing some or all of the client-server database security features that are indicated in table 11-14.

Table 11-14. Database security features

- Privileges
- Roles
- Views
- Stored Procedures
- Triggers

The following section discusses the particular merits of each database-related security feature indicated in the table above.

Privileges

A privilege is a permission to perform a particular database operation, such as querying, updating, deleting, or inserting data in a relational database table. Without privileges, a user cannot access the information that is stored in the relational database. The database access privileges are controlled with GRANT statements. The most elementary database security rule is fairly simple: grant the users only the strict minimum set of privileges that they require to accomplish their normal day-to-day business tasks.

Consequently, you need to ensure that the selected relational database system can implement this "only on a need-to-know basis" security policy, by default. With such a policy, nobody can have database access privileges unless he/she is specifically granted permission to use database privileges.

This approach ensures that the database system retains full control over who accesses the data stored in the database. It eliminates the possible scenario in which a user might attempt to bypass the application and its related security mechanisms by accessing the data directly, with an ad hoc database query tool, for instance.

The database administrator should review the security requirements that were defined during the analysis phase, along with any change that might have occurred since then. Then the different security rules that any particular individual or any group of users needs are defined by the database administrator. Table 11-15 lists the different access types that can be assigned to an SQL-Based database object.

Table 11-15. Database object access rules

- Select
- Insert, update, delete
- Execute (for stored procedures)
- Index, reference, alter, grant
- All

Typically, the users are allowed only select, insert, update, and delete accesses to a database object.

Roles

Privileges can be explicitly granted to specific users, such as granting John or Mary a read-only access to the table "Personnel." However, this approach might prove to be not very practical when the client-server application is used by a large number of users. In such situations, the concept of roles can be used to ease the administration of database privileges.

A role is a collection of pre-defined privileges that can be granted either to users or to other roles. Instead of explicitly granting the same privileges to all the users one after the other, the privileges for a group of users who perform a common business function can be granted to a specific role. Once a specific role has been created, each user in that group can be then granted this specific role. Once a specific role has been created, each user in that group can be then granted this specific role. In the following example, a role identified as Human_Resource. Rep has the authority to execute SELECT, UPDATE, INSERT, and DELETE operations against the Eng_Dept table:

```
CREATE ROLE Human_Resource.Rep;
GRANT SELECT, UPDATE,INSERT,
DELETE
ON Eng_Dept TO Human_Resource. Rep;
```

Having this single role defined, you can grant it to several users, with the following instructions:

```
GRANT Human_Resource.Rep to Paul, Mary;
```

With most relational database servers, a role may be assigned a password. Which can provide an additional layer of security:

```
CREATE ROLE Human_Resource.Rep
IDENTIFIED BY “Password xyz”;
```

The user is prompted for a password whenever the role is invoked. If the password is invalid, the user will not be able to use the set of privileges that were defined for this particular role.

A role can also be activated directly within an application to constrain the users to exercising their privileges only within the boundaries of that application. As soon as the users quit the application, their privileges are revoked immediately. By enforcing this additional precautionary database security measure, the database administrator can prevent a potentially hazardous situation in which users can execute destructive SQL statements against database tables while utilizing online ad hoc database query tools with the privileges that were granted to them via a user role.

When defining various user roles, take into consideration real-life situations that frequently occur in the workplace. For example, a user may temporarily replace the supervisor for a short period of time. Hence, this particular individual may require some additional database access capabilities for the duration of that temporary assignment.

Database views

Database views can be utilized advantageously as another technique to exert more control on the user’s ability to access the information that is stored in the database. As an example, a simple database view can be created to display only relevant columns in a database, as illustrated in the following example:

```
CREATE VIEWEmp_Supv AS
SELECT emp_name, emp_id, emp_supv
FROMemp_table;
```

With this view, the users can access only the ‘emp_name_emp_id_ and emp_supv’ fields from the ‘emp_table’ and nothing else.

Stored procedures

Another method for controlling security at the database level is to code the data access rules inside a stored procedure. A stored procedure can be created to perform a function with all the necessary database access privileges. Then the permission to execute this stored procedure is granted to certain users. This authorization is done, though, without granting the users any direct access rights to the tables and SQL operations that are used by the stored procedure.

With this technique, the users are granted EXECUTE privileges on the stored procedure itself, and the only possible way for them to access the data in the database is through the stored procedure. In other words, the data accesses are encapsulated at the level of the stored procedure. This technique should be considered only when security cannot be implemented through roles.

Database triggers

Database triggers can also be used to exert tighter access control against the database. The trigger execution follows a pattern similar to a stored procedure, with the exception that the trigger is fired automatically as soon as a delete, insert, or update operation is performed against a database table.

Contrary to a stored procedure, the user who executes the database operation has no control whatsoever over the execution of the database trigger. With this technique, enterprise-wide business rules can be enforced across all corporate applications at the level of the database trigger, based on predetermined business events.

Application-level security

An alternative method to limit access to data is to include the access rules within the application code itself. With this scenario, the application itself controls the data access. While this technique enforces a certain level of security when the users operate within the internal boundaries of the application, it might present serious shortcomings when the users attempt to bypass the application and access the data directly via an ad hoc query tool, for instance.

Mandatory access controls

In specific situations, the standard discretionary database access control mechanisms may simply be insufficient to secure highly sensitive information. Typically, organizations that process highly sensitive information might require more stringent levels of security to control the user accesses to the database system. Quite often, such a situation might be the case for applications that are developed by government, commercial, and intelligence agencies that use highly sensitive data.

In a robust multi-level secure database server, all the database objects, such as the tables, rows, and procedures, are labeled according to their level of sensitivity. These labels cannot be altered by the users of the database. Furthermore, the users can get to just the data that match the range of labels they are authorized to access, according to special security clearance tables.

In a typical multilevel secure database server environment that utilizes mandatory access controls, only the security administrator can manage and administer the data accesses in the database system. This arrangement is quite a departure from discretionary access controls, in which the owners of the database objects might be able to grant access to the data they control to other users.

Database encryption

Encryption is a special technique that is used to convert readable data into an unreadable format. Those people who know the algorithmic key that was used to originally encrypt them can decrypt the data. If the users do not have access to the proper key, then they cannot read the encrypted data.

Encryption is primarily utilized to scramble information used in digital transmission but can also be used to secure the information contained in a database or other types of data storage facilities. Encrypting an entire database can adversely affect its overall performance because of the overhead associated with encrypting or decrypting information every time it is modified by the users. For this reason, this technique is not used frequently for an ordinary database application, unless some very stringent security requirements demand such as drastic security measure.

Nevertheless, the organization might decide to encrypt selective segments for information that are stored on the databases and that are highly sensitive. In such a situation, only a very small number of carefully chosen individuals are provided with the capability of decrypting the encrypted data.

Although most client-server database systems usually do not provide industrial-strength encrypting functionality in their software, several third-party vendors offer robust commercial encryption algorithms. In certain cases, though, specific encrypting technologies might be subject to rigorous export restrictions.

The following Web site addresses are provided for those readers who want more information on some vendors that offer different encryption software solutions.

http://www.axent.com(Omniguard/Enterprise Access Control)
http://www.entrust.com(Entrust)
http://www.worldtalk.com(World-Secure Server)
http://www.redcreek.com(Ravlin 4)

Database auditing requirements

For certain categories of systems, such as financial applications for instance, keeping a record of all physical database activities might be important to ensure that the users are held responsible for their actions.

If keeping database modification traces is the case for the application at hand, then the production database server must be able to provide some form of traceability for all the database operations that are performed by the users.

Most relational database server systems provide at least some basic form of audit trail capabilities. Some of the most common built-in database auditing features that can be used to design and implement an automated database audit trail subsystem are listed in the table.

Table 11-16. Database audit facilities

Database Audit Feature	Description
Database audit reporting facility	Some general database reporting facilities are provided to produce various database audit reports and also to support ad-hoc queries against the audit information that is centrally accumulated and stored in a system database.
DML/DDL auditing	Standard auditing facilities are provided to allow the automatic auditing of various data manipulation language (DML) and data definition language (DDL) statements, for all types of database objects and structures.
Database user connections auditing	Standard auditing facilities are provided to allow the auditing of all the attempts that are made to connect with or disconnect from the database.
Database user activity auditing	A standard auditing facility is provided to allow the auditing of all database activity by operation, by utilization of database system privileges, by objects, or by the user.

The granularity and scope of the database audit functions can be as generic as auditing all the database accesses that are made by the users across the enterprise or as confined as auditing the database activities of a particular individual.

However, you want to ensure that the implementation of the database auditing functions does not adversely affect the performance of the production databases. For this reason, the potential database overhead that can be generated by implementing database auditing functions must be assessed when designing the application databases audit trails.

Database triggers can sometimes be utilized to replace or supplement the built-in-database audit features. For instance, database triggers can be used to audit specific database activities against particular database operations, such as triggering automatically an audit trail process every time an employee updates the employee salary database table.

Design Shared, Reusable Software Components

Several types of software components can be reused in large software applications or across multiple applications. Some of the most common types of software components that can be shared in the context of an enterprise-wide application architecture are the database stored procedures, the database triggers, and the remote procedure subroutines. Reusable code can drastically ease application maintenance later on, simply by avoiding code duplication.

Consequently, the shared application components should always be among the first software objects to be designed, coded, and tested, so they can then be shared among all application developers, preferably at the level of the enterprise.

The next three technical activities cover the design of the database stored procedures, triggers, and remote procedures.

Design database stored procedures

A stored procedure is a collection of pre-compiled and pre-optimized SQL statements that are centrally stored on the server database rather than in each application. Any application can then execute the stored procedure, simply by invoking it. Besides the usual SQL constructs, most relational database software management systems allow the inclusion of additional control statements within the stored procedure. These additional control statements support the usual repetition and conditional programming language constructs such as LOOP, IF THEN ELSE, and CASE.

The execution of the stored procedure is triggered via a CALL statement from the client program. This CALL statement is usually invoked by an application, a trigger, or another stored procedure. Parameters are ordinarily passed along via the CALL statement and return values are sent back to the caller.

The execution of a stored procedure is far more efficient than a dynamic SQL query.

At creation time, the stored procedure is verified for correct syntax and subsequently saved in the system tables. When the stored procedure is invoked for the first time by an application, it is retrieved from the system tables, compiled, and stored into the virtual memory of the server database. From there, it is finally executed. Since the stored procedure is compiled and optimized only once and then stored permanently on the database server, subsequent executions of the procedure that are requested by any client application will be performed immediately by the database server.

On the other hand, when a dynamic SQL query is sent from the client workstation to the database server, the server parses the command, verifies the syntax and names utilized, verifies the protection levels, optimize the search, creates an execution plan, and then compiles and executes it. This process must be repeated entirely every time a query is fired from the client workstation.

Some experts have claimed that stored procedures can sometimes be executed as much as 50 to 100% more quickly than the equivalent set of dynamic SQL statements.

The advantages and potential disadvantages of using stored procedures in a client-server system are summarized in Table 11-17.

Stored procedures can be used quite favorably in several instances during the development of a large client-server or Web database application, as indicated in Table 11-18.

Based on all the information provided on stored procedures, developers should carefully analyze the data access specifications that are documented in the data access matrices to identify the critical system transactions that are characterized with high frequency usage and/or high data access. These system transactions are likely candidates for stored procedures, which can also be reused for other business applications in the corporations.

Table 11-17. Stored procedures

Advantages	Potential Drawbacks
• Usually executes more quickly than dynamic SQL • Promotes code reusability • Reduces network traffic by returning to the calling application (normally the client) only the required rows • Promotes standardization and modularity since the stored procedure is physically stored on the database server and therefore is independent of any particular application system • Can be utilized to enhance security and data integrity when used for database updates • Reduces maintenance; if a stored procedure is modified, all changes are made in one central locations, and all applications using it will automatically get the new functionality	• Can limit the application independence toward the database software since stored procedures can tie you to a proprietary database language • Incorrect modifications that can be applied inadvertently to a stored procedure can affect adversely several applications or several programs within the same applications • Improper use or inefficient codification of SQL calls in the stored procedures can result in system performance degradation

Table 11-18. When to use stored procedures

• To reduce the overall network traffic between the client and the server caused by the frequent execution of large database transactions • To code only once the set of SQL functions that are commonly used and shared across multiple applications or by certain programs within the same application; stored procedures can be used in the context of a reusable code enterprise program • To standardize any set of SQL actions that are executed by more than one program within a client-server application • To improve security, since the users can be allowed to execute a stored procedure even if they were not granted permissions to access directly the database tables or views that are referenced in it • To centralize and enforce business rules at a single location such as the server level

Design database triggers

A trigger is a special form of stored procedure that is invoked by the database server when an attempt is made at updating the data that is stored in the relational tables.

Contrary to a stored procedure, though, triggers cannot be called directly by an application or a user. Rather, the database server automatically executes them only when the data they are associated with are modified. In that respect, we can say that triggers are data-driven events.

The vast majority of the relational database servers support the use of triggers. However, their implementation might differ slightly from one relational DBMS vendor to another.

Triggers are automatically fired whenever an insert, update, or delete SQL operation is performed on the table identified in the trigger.

Once triggers are created, they are stored on the database server and stay there until they are explicitly deleted. Most RDBMSs usually support the common types of triggers that are shown in the table below.

Table 11-19. Trigger types

Type of Trigger	**Invocation Event**
INSERT	Invoked when a new row is inserted into the table associated with the trigger
DELETE	Invoked when a row is deleted in the table associated with the trigger
UPDATE	Invoked when a row in the table associated with the trigger is updated
UPDATE OF Column-List	Invoked when a row in the table associated with the trigger is updated and a column in the column list has been changed

Each type of trigger shown in the table can be set for automatic execution either before or after the insert, update, or delete SQL request is performed.

Whenever the referential integrity features supported by a given RDBMS are not sufficient, triggers can always be used to enforce specific or additional constraints on data.

As a rule of thumb, the set of application business rules that is directly associated with the creation, update, or deletion of data can often benefit from the implementation of triggers. Similarly, potential candidates for triggers can also be found by carefully examining the business events (external, internal and temporal) that were documented in the business process model, during the analysis phase. Whenever a situation arises in which a specific action must be accomplished automatically in response to some modifications made to the database, triggers should be considered seriously as one of the best solutions at hand.

In such instances, utilizing triggers is far more efficient instead of executing an equivalent set of procedural logic at the client workstation. If the procedure is executed at the client workstation, then at least one more call to the database becomes necessary. The net result adds up to an increase in network traffic combined with a potential diminution of performance. Furthermore, the SQL statements included in a trigger often can be optimized in such a way that they will execute far more efficiently than if they were submitted from the client workstation.

A trigger can also be seen as a form of reusable code, since the logic it executes does not have to be replicated by different business applications.

Triggers can be used in numerous pre-defined situations in a client-server or Web database application, such as those indicated in Table 11-20.

Table 11-20. When to use database triggers

• To log the major activities/events performed against all database tables, independently of the different applications accessing them • To support complex, detailed auditing measures; for example, you can monitor value changes by accumulating detailed update information in specific audit tables • To enforce elaborate security authorizations to verify specific data changes • To perform complex validations on data or various types of calculations such as summarization of data • To automatically update data values in other tables based on changes performed against a specific table • To automatically enforce referential integrity across the several database servers; you can also enforce complex constraints that cannot be implemented through the regular integrity features that are supported by the database • To automatically produce derived column values; for example, you can create values for derived columns automatically when you modify the columns on which they are constructed • To circumvent invalid transactions • To enforce enterprise-wide compliance to fundamental corporate business rules • To notify users through electronic mail messages when specific database change events are triggered

Table 11-21 summarizes the common advantages and potential disadvantages of using database triggers.

Table 11-21. Database triggers advantages and disadvantages

Advantages	Disadvantages
• Complex business rules, integrity constraints, and procedural logic can be segregated from the applications and independently enforced at the database server level; useful in the context of an enterprise-wide application architecture • Easier to maintain • Ability to link a data-driven event to the specific data that triggers the event	• Improper testing and implementation can adversely affect several applications by generating unexpected results • Can increase consumption of system resources and cause performance deterioration if not well optimized

Design remote procedures

In a traditional mainframe environment, the communications between a main program and its subroutine modules are handled within the confines of a single computer.

In a client-server environment, the procedural logic of a single program can be implemented partially in a client module and partially in a server module. Furthermore, you can have both routines run on a different physical platform machine. The subroutine that runs on a different physical computer calling procedure is called a remote procedure.

Remote procedure calls (RPCs) provide a method of communicating in a distributed environment. Functions can be invoked and executed with processors that can run on different operating systems and computer platforms. The RPC tool provides the mechanism for the client to invoke the server module as if both modules were physically part of the same piece of code logic.

RPCs hide from the developers all the intricacies and complexities of network protocols. In fact, from a programmer's standpoint, RPCs are transparent and operate in the same manner as a traditional function call, with the distinction, though, that the function is executed on a different machine from the one calling it. Since the applications interfaces with the RPC API layer and not the network, modifications at the network level do not adversely affect the application.

The rationale behind the use of RPCs is to distribute the work and data over the network to maximize resources. It also enables the design and construction of modular code, with the use of subroutines that are stored on a machine where everyone can access them. RPCs can help standardize the way programmers are creating their subroutine calls in a way that remote procedures can recognize them and execute the requests.

For the most part, the technology supporting RPCs relies heavily on direct synchronous connections between two machines. Because of this architectural particularity, RPCs performance for industrial-strength applications can be very dependent on the robustness of the network supporting them. Furthermore, if the application attempts to invoke a remote function located on an application server that is down or very busy, then the application must wait until the call is satisfied. No other tasks on the client machine can be performed until the remote procedure call is satisfied.

The interfacing between the client and the server in a remote procedure call is generally implemented via a framework that was standardized by the Open Software Foundation (OSF) with their specification for the Distributed Computing Environment standard (DCE). Remote procedures can be advantageously used in the context of an enterprise- wide application architecture, since they represent a form of reusable code.

Design System Interfaces

In large and complex environments, you frequently encounter client-server or Web database applications that must interface with different types of systems which reside either on the mainframe or on different client-server computing platforms.

Depending on the technology architecture that is currently in place in the enterprise, the client-server system might have to create some files that, at certain point in time during the day, will be uploaded to the mainframe for further processing. The reverse might also be true. Files might be downloaded from the mainframe system for further processing by the client-server system. Performance issues might constrain the designers to resorting to some form of batch processing or database replication, using a gateway middleware.

Interface design specifications

If interfaces must be constructed between the client-server application software and other systems, then be sure to review the initial interface strategy that was developed during the analysis phase. Can new information cause a change of strategy?

Table 11-22 enumerates a list of simple questions that should help to validate the interface design strategy.

Table 11-22. Application interface design strategy checklist

- Does a need exist for database middleware software that allows the client-server or Web system to access files and databases that reside on a mainframe platform? On a different client-server platform such as from a UNIX to an Intel-based computer platform?
- Did the estimated volume of data for the interface change since the last development phase? Was the potentially large volume of data peak and low variations accounted for, based on daily, weekly, monthly, or yearly processing cycles? What about the estimated volume of data growth figures per unit of time?
- Did the frequency (i.e., hourly, daily, weekly, monthly, yearly, or on-demand) at which the interface must be processed changed?
- Will some form of file reconciliation mechanism exist between the system donor and the system receiver?
- Does a need exist for some recovery/restart mechanism if the batch interface program "abends" right in the middle of the job? What about checkpoints or rerun procedures?
- Must security measures be enforced to prevent unauthorized access to or submissions of the batch interfaces files/programs?

Design the Ad Hoc Report and Query Environment

The developers might need to address two broad categories of reports and queries during the system development cycle to satisfy the user needs for turning data into useful information.

The first category consists of reports and queries that are fairly static and well structured. These reports are the regular, pre-formatted type of reports for which the detailed information requirements are well defined and complete. These reports are usually created by the developers and exhibit some, if not most, of the characteristics that you see in Table 11-23.

Table 11-23. Pre-defined reports characteristics

- The types of data that go in the reports are well known by the users
- The reporting data requirements are fairly static as opposed to volatile
- The production of the reports is scheduled on a regular basis or on demand
- Some of the reports can be high-volume reports
- The integrity of the data shown on the reports must be very high
- The report presentation layout may range from very simple to very complex
- The report data can be extracted directly from the operational production databases
- The reports are managed with an emphasis on efficient report distribution and control
- The application programs that produce the reports are fine-tuned to optimize the database accesses and not adversely affect the overall performance of the online transactions

The second category of reports and queries are those for which the users never know exactly what type of information will be required from one day to another. Very often, a sophisticated data reporting and querying environment must be set up beside the operational database environment to satisfy the user needs for ad hoc information. This special reporting and querying environment contains mostly static, historical data that is refreshed on a regular basis, such as daily, weekly, quarterly and annually.

Interface design

As described in the activity interface design, the primary objective of the current activity is to design and construct the interface that bridges the application operational databases to the enterprise ad hoc database report and query environment.

The frequency at which data will be transferred from the operational application databases to the separate ad hoc report and query environment is determined by the users, along with the identification of the specific data fields that must be migrated for this purpose.

The developers should also ensure that the users can live with the fact that the ad hoc report and query environment might contain data that are not necessarily as current as the data that are housed in the operational databases.

The interface should be designed so that the required data gets transferred automatically at periodical intervals, preferably without any human intervention. If necessary, brand-new programs can be created to migrate the data between the two environments, or you can refine and extend existing business transactions specifically for this purpose.

Database replication facilities can also be used to copy the data from the operational databases to the data warehouse databases, at predetermined time intervals.

Create System Documentation

The table below provides a list and brief description of the most common type of guides that can be produced for the new system.

Table 11-24. Brief description of different types of system guides

User Guide	Documents the set of procedures that the users can use to interact with the system
Online Help Guide	Provides the users with an online description of the system functions and facilities, including suggestions on how to best use the graphical user interface, description of the fields contained on each specific screen or form, how to deal with errors, generic search facilities, and so forth
System Operations Guide	Documents the set of technical procedure that are required to operate the system in the production environment, such as the backup and recovery procedures, the performance monitoring procedures, and the security procedures
Technical Reference Guide	Documents the technical components of the system for those individuals who will maintain and evolve the system, once it is transferred into its final production environment

As shown in Table 11-24, the User and Online Help Guides target the user community as their prime audience. At the opposite end, the System Operations and Technical Reference Guides primarily target the Data Center and Systems Maintenance groups as their main customers. The User and Online Help Guides are business-oriented, whereas the System Operations and technical Reference Guides are technical.

The User Guide describes, in non-technical language, the various tasks that the users must follow to interface with the new software system while doing their regular jobs. It contains a set of operational instructions on how to best use the new system, from an end-user perspective. The User Guide can also be used to train users long after the system has been transferred into its production environment, if it is kept up-to-date.

The Online Help Guide contains system-related documentation that the users can interactively reference while using the actual system. Some organizations might decide to create an Online Help Guide and then print it to be used as a User Guide in a paper format.

The System Operations Guide describes the tasks that are required to operate the system in its production environment. Typically, the intended audience for this guide is the Operations Systems personnel who will run and support the system in the production environment.

The Technical Reference Guide describes the internal architecture of the system. It is primarily targeted at the systems professionals who will maintain the production system.

At this stage, the developers should discuss with the different user groups who will eventually utilize the system or those technical groups who will operate or support it, what their essential requirements for system documentation are.

The need or extent to which all these application documentation guides should be (or should not be) produced might vary from one project to another and even from one organization to another.

For instance, if the same people who are developing the new system will also be called upon to maintain it, then the technical reference guide might not be necessary at all. Or if it is still required, then it might contain high-level technical information instead of containing information at a very detailed level. On the other hand, if the people who will support the system were not those who developed it, then the technical reference guide will likely require a more detailed description of the most critical technical aspects of the new system and its maintenance environment.

Once the basic requirements for the system documentation have been determined, along with the brief explanation of how they will be used, the proposed format and overall content of each guide should be discussed with each involved party. Also, the type of medium that will be used to deliver the system guides should be decided at this stage. For example, will the guides be provided in soft or hard copy, or both? Will they be available in an intranet environment?

As much as possible, the format and content of the various system documentation guides should not be re-created from scratch. If enterprise documentation standards exist, the development team should use them or even recycle existing documents that were produced from previous projects as much as possible.

Quickly developing an initial prototype of the Online Help Guide is probably one of the most effective ways of demonstrating to the users the "look and feel" of this particular system documentation component. Similarly, show some representative samples of the other guides to the users or data center personnel to firm up the system documentation requirements and manage expectations.

At the bare minimum, produce a draft table of content for each type of documentation guide and review it with some representatives of the targeted audience. The table of contents should provide a list of the major chapters that will be included in the guide and the general organization of the guide.

The detailed sections and content of each guide can be developed later during the next system development phase. For very large projects, a slight variation to this approach might consist of developing a complete sample of a representative chapter for each guide and then reviewing these samples with users or operations systems staff.

Tables 11-25 and 11-26 show an example of table of contents for a User Guide and a System Operations Guide. These 'table of contents' may be adapted and tailored to satisfy the specific needs at any particular project.

Table 11-25. Table of contents for the user guide

INTRODUCTION
- Purpose of the guide
- Intended Audience
- Assumptions
- How to use the guide
- General Conventions

SYSTEM OVERVIEW
- System mission
- Situation of the system within the organization
- System context diagram
- Summary description of major system functional components
  - o Online transactions
  - o Online reports
  - o Batch reports
  - o Ad hoc reports
  - o Batch job descriptions

GRAPHICAL USER INTERFACE
- Introduction
- General menu/toolbar options
- Online help facility
- Detailed screen(s) or report layout(s) description
- Data fields and control descriptions
- General task flow diagram
- Detailed task(s) description
  - o Why?
  - o When?
  - o Pre-requisites
  - o How?
  - o Error handling

MISCELLANEOUS
- Appendix
- Glossary of terms
- Index
- Reader's comment form

Table 11-26. Table of contents for the system operations guide

INTRODUCTION
- Purpose of the guide
- Intended audience
- Assumptions
- How to use the guide
- General conventions

SYSTEM OVERVIEW
- System technology architecture
- General description of the major system technical components
- General description of the system operating procedures
  - Hardware maintenance procedures
  - Software maintenance procedures
  - Networking maintenance procedures
- Detailed system operations procedures
- Detailed task(s) description
  - Why?
  - When?
  - Pre-requisites
  - How?
  - Error handling

MISCELLANEOUS
- Appendix
- Glossary of terms
- Index
- Reader's comment form

Summary

In this module, you learned how to design a database software system. The design did not only focus on the logical and physical database design but also tackled the creation of GUI, database security considerations, use of reusable components, system interfaces, ad hoc report and query, and system documentation.

Module 12
Database Trends

Our course has concentrated mainly on designing and implementing a relational database system. However, it should be noted that the relational model is no panacea, and has never claimed to be. It has shortcomings in dealing with recursive queries, missing information, "complex object" support, and so on and these shortcomings should be properly addressed.

Over the years, many enhancements have been done to improve database systems. In this last module, we briefly examine the question "What are the trends in database system design?" There are several research topics that can be considered but only three of these areas will be touched on lightly in this module.

Objectives

At the end of this module, you should be able to:

1. Describe the features of Object-Oriented Databases, Deductive Databases and Distributed Database Systems; and
2. List the advantages and disadvantages of each system.

By nature of the subject matter, this module somewhat differs in its structure and intent from all previous modules. This module is divided into three main parts. The first part will discuss basic Object-Oriented concepts and their incorporation into databases. The second part will introduce basic concepts of Deductive Databases (DDBs), and finally, the last part will discuss concepts of Distributed Database Systems (DDBSs).

Object-Oriented Databases (OODBs)

The term **object-oriented (OO)** has its roots in OO programming languages. OO concepts are now applied to the areas of databases, software engineering, knowledge bases, artificial intelligence, and computer systems in general.

To help us understandOO concepts faster, consider the "equivalence" of OO terms to traditional programming terms below.

OO Term	**Programming Term**
object	variable
class	type
method	function
message	call
class hierarchy	type hierarchy

The term **object** in OO parlance corresponds to a variable in the programming sense. In an OO programming language, an object exists only during program execution. In contrast, an OO database provides capabilities so that objects can be created to exist permanently, or **persist** and be shared by numerous programs. Hence, OO databases store **persistent object***s* permanently on secondary storage, and allow the sharing of these objects among multiple programs and applications. This requires the incorporation of other well-known features of database management systems you learned in Unit 2, such as cataloging and indexing, concurrency control, and crash recovery. An OO database system interfaces with one or more OO programming languages to provide persistent and shared object capabilities.

One aim of OO databases is to maintain a direct correspondence between database objects and real-world entities so that the objects do not lose their integrity and can easily be identified and operated upon. Hence, OO databases provide a unique system-generated **object-identifier** (OID) for each object that is guaranteed never to change. We can compare this OID to a primary key attribute in the relational model. Remember, each relation must have a primary key attribute which can uniquely identify each tuple. If the value of the primary key is changed, the tuple will have a new identity, even though it represents the same real-world entity. Alternatively, a real-world object may possess different keys in different relations, making it difficult to ascertain that the keys do indeed stand for the same object (e.g., the object identifier may be represented as EMP_ID in one relation and as SSS# in another relation). It should be stressed that OIDs should not be thought of as object properties in the same sense as attributes in a relation. They may be regarded as **hidden attributes**, in the sense that they are not visible to users, nor can they be directly manipulated.

Objects can be simple or complex. An OO system will provide a set of built-in primitive object classes, such as INTEGER or FLOAT. The user can define individual objects of these primitive classes. More complex objects can then be constructed from combinations of existing objects. This brings to the fore another feature of OO databases. Objects can be defined as an **object structure** of arbitrary complexity which allows objects to contain all of the significant information that describes the real-world entity. In contrast, in traditional database systems, information about a complex object is often scattered over many relations or records, leading to loss of direct corres-spondence between real-world object and its database representation.

Example 1:

An object corresponding to an EMPLOYEE relation can be constructed as a set of tuples. Each EMPLOYEE tuple will be an object consisting of a set of scalars (e.g., AGE, ENAME, ADDRESS, SALARY). And each scalar will be an object consisting of one of the built-in classes (i.e., INTEGER, FLOAT).

The internal structure of an object includes the specification of **instance variables,** which hold the values that define the internal state of the object. Hence, an instance variable is similar to the concept of an **attribute**, except that instance variables may be encapsulated within the object and thus are not necessarily visible to external users. Instance variables may also be of arbitrarily complex data types.

The OO term **class** corresponds to the notion of a data type or more accurately - to an abstract data type (ADT). In OO programming, the users can define their own classes, and all classes, whether built-in or user-defined, includes definition of operators that apply to the objects of that class. Thus, the only way to operate on an object is by means of the operators defined for that object's class. Object-oriented systems allow definition of the operations or functions that can be applied to objects of a particular type. In fact, some OO models insist that all operations a user can apply to an object must be predefined. This forces a complete **encapsulation** of objects. This rigid approach has since been relaxed in most OO data models, since it implies that any simple retrieval requires predefined operation.

Example 2:

If a user-defined class called DEPARTMENT (with operators CUT_BUDGET, ADD_EMPLOYEE, DELETE_EMPLOYEE and CHANGE_MANAGER) was created, then individual objects defined as a DEPARTMENT can have these operations (only) applied to them.

To encourage encapsulation, an operation is defined in two parts. The first part, called the **signature** or interface of the operation, specifies the operation name and parameters. The second part, called the **method** or **body**, specifies the **implementation** of the operation. Operations can be invoked by passing a **message** to an object, which includes the operation name and the parameters. The object then executes the method for that operation.

Example 3:

An example of a built-in operation is the method NEW, which is understood by every class. Applying NEW to the class of EMPLOYEE, will cause a new empty EMPLOYEE tuple to be created. By empty, we mean that AGE, ENAME, ADDRESS, and SALARY instance variables within that tuple will be initialized to a **null** value.

Another key concept in OO systems is that of **type and class hierarchies** and inheritance. This permits specification of new types and classes that inherit much of their structure and operation from previously defined types or classes. Hence, specification of the object types can proceed systematically. This makes it easier to develop the data types of a system incrementally, and to reuse existing type definitions when creating new types of objects.

An OODB drawback

One problem in OO database systems involves representing **relationships** among objects. The insistence on complete encapsulation in early OO data models led to the argument that relationships should not be explicitly represented, but instead should be described by defining appropriate methods that locate related objects. However, this approach does not work very well for complex databases with multiple relationships, because it is useful to identify relationships and make them visible to users. Many OO data models now allow the representation of relationships via **references** - that is, by placing the OIDs of related objects within an object itself.

Some OO systems provide capabilities for dealing with **multiple versions** of the same object–a feature that is essential in design and engineering applications. For example, an old version of an object that represents a tested and verified design should be retained until the new version is tested and verified. A new version of a complex object may include only a few new versions of its component objects, whereas other components remain unchanged.

There are a lot more OO concepts that we haven't covered. If you feel there's a need for more discussions on OO, you can read more on this in the references cited.

SAQ 12-1

For each of the three numbers below, encircle the number that corresponds to a TRUE statement. If the statement is FALSE, encircle the word/phrase that makes the statement false and write on top of the encircled word/phrase the correct word/phrase to make the statement true. Feel free to review the text before you put marks on the sentences below.

1. Object-oriented databases have adopted many concepts that were developed for object-oriented programming languages.
2. A key feature of object oriented databases is the power they give the designer to specify both the structure of complex objects and the operations that can be applied to these objects.
3. Object-oriented data models use multiple versioning to represent multiple relationships in complex databases.

Deductive Databases (DDBs)

A **deductive database system** is a database system that includes capabilities to define rules, which can deduce or infer additional information from facts that are stored in a database. Because part of the theoretical foundation for some deductive database systems is mathematical logic, they are often referred to as **logic databases**.

In a deductive database system, a declarative language is used to specify rules. By a **declarative language**, we mean a language that defines what a program wants to achieve rather than one that specifies the details of how to achieve it. An **inference** engine (or **deduction mechanism**) within the system can deduce new facts from the database by interpreting these rules. The model used for deductive databases is closely related to relational data model. It is also related to the field of **logic programming** and the **Prolog** language.

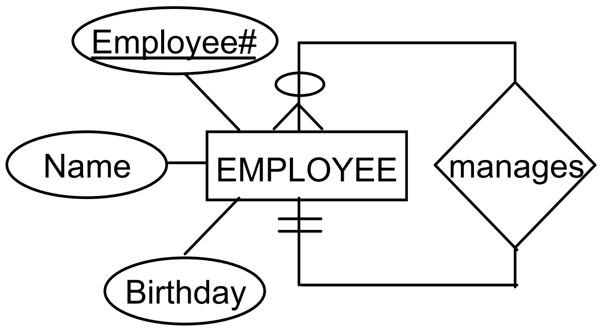
A deductive database uses two main types of specifications: facts and rules. **Facts** are specified in a manner similar to the way relations are specified, except that it is not necessary to include attribute names. Recall that a tuple in a relation describes some real-world fact whose meaning is partly determined by the attribute names. In a deductive database, the meaning of an attribute value in a tuple is determined solely by its *position* within the tuple. **Rules** specify virtual relations that are not actually stored but can be formed from the facts by applying inference mechanisms based on the rule specifications.

We discussed object-oriented databases (OODBs) earlier. It is instructive to put deductive databases (DDBs) in a proper context with respect to OODBs. The emphasis in OODBs has been on providing a natural modeling mechanism for real-world objects by encapsulating their structure with behavior. The emphasis with DDBs, in contrast, has been on deriving new know-ledge from existing data by supplying real-world relationships in the form of rules. OODBs have traditionally left optimization of navigational queries up to the programmer, whereas DDBs use internal mechanisms for evaluation and optimization. Signs of marriage of these two different enhancements to traditional databases have started appearing in the form of adding deductive capabilities to OODBs and adding programming language interfaces like C^{++} to DDBs.

Representation of facts

Like Prolog, most deductive programming languages represent facts using **predicates.** A predicate has an implicit meaning, which is suggested by its unique name, and a fixed number of **arguments**. If the arguments are all constant values, the predicate simply states that a certain fact is true. If, on the other hand, the predicate contains variables as arguments, it is either considered as a query or as part of a rule or constant.

Consider the logical model below. This suggests that an employee may possibly manage other employees and that an employee is managed by at most one manager.



A corresponding relational database for this model possibly contains the EMPLOYEE relation shown below.

EMPLOYEE

Employee#	Name	Birthday	Supervisor#
12-345	John	09-JAN-55	33-344
33-344	Frank	08-DEC-45	88-866
99-988	Simon	19-JUL-58	98-765
98-765	Jennifer	20-JUN-31	88-866
66-688	Joy	15-SEP-52	33-344
45-345	James	31-JUL-62	33-344
98-798	Cathy	29-MAR-59	98-765
88-866	Lee	10-NOV-27	null

Given the EMPLOYEE relation, we can form three predicates named: **supervise, superior** and **subordinate**. The supervise predicate can be defined using the facts in the relation, each of which has two arguments: a supervisor name, followed by the name of a subordinate of that supervisor.

Example 1:

Using Prolog notation, we have the following facts:

```
supervise (Frank, John)..   supervise (Jennifer, Simon)..   supervise (Lee, Frank)..
supervise (Frank, Joy)..    supervise (Jennifer, Cathy)..   supervise (Lee, Jennifer)..
supervise (Frank, James)..
```

Thus, supervise (X,Y) states the fact that "X supervises Y." Notice the omission of the attribute names in the Prolog notation. Attribute names are only represented by virtue of the position of each argument in a predicate; the first argument corresponds to the supervisor, and the second represents the direct subordinate.

The main contribution of deductive databases is the ability to specify recursive rules. A rule is of the form **head :- body.**

Example 2:

The other two predicates, superior and subordinate, are defined by rules. Based on the facts that we have, the rules are:

1. superior (X,Y) :- supervise (X,Y).
2. superior (X,Y) :- supervise (X,Z), superior (Z,Y).
3. subordinate (X,Y) :- superior (Y,X).

The predicate superior, whose first argument is an employee name and whose second argument is an employee who is either a direct or an indirect subordinate of the first employee. By indirect subordinate, we mean the subordinate of some subordinate down to any number of levels. Thus, superior (X,Y) stands for the fact that "X is a superior of Y" through direct or indirect supervision.

The first rule states that for every value of X and Y, if supervise (X,Y), the rule body, is true, then superior (X,Y), the rule head, is also true, since Y would be a direct subordinate of X. This rule can be used to generate all direct superior/subordinate relationships from the facts that define the supervise predicate. The second recursive rule states that if supervise (X,Z) and superior (Z,Y) are both true, then superior (X,Y) is also true. This is an example of a **recursive rule**, where one of the rule body predicates is the same as the rule head predicate. In general, the rule body defines a number of premises such that, if they are all true, we can deduce that the conclusion in the rule head is also true.

A **query** typically involves a predicate symbol with some variable arguments, and its meaning (or "answer") is to deduce the different constant combinations that, when assigned to the variables, can make the predicate true.

Example 3:

For example, the query "superior (Lee, Y)?" requests the names of all subordinates of "Lee" at any level.

A different type of query which has only constant symbols as arguments, returns either a true or false result, depending on whether the arguments provided can be deduced from the facts or rules. For example, "superior (Lee, Joy)?" returns true if superior (Lee, Joy) can be deduced, and returns false otherwise.

SAQ 12-2

Try this test without going back to the text. You can review the text later after you have finished answering the items. Now, for each of the five items below, match the items in column A with the definitions/descriptions in column B. Write the corresponding letters of the items in column B in the box provided in column A.

Column A	**Column B**
☐ 1. inference engine	a. a predicate with some variable arguments
☐ 2. predicate	b. interprets rules to gather new facts
☐ 3. query	c. specifies a relation which is implied
☐ 4. rule	d. the head predicate is part of the body predicate
☐ 5. recursive rule	e. used for representing facts

Distributed Database Systems (DDBSs)

A **distributed database system (DDBS)** can be defined as a collection of multiple, logically interrelated databases distributed over a computer network. The software system that permits the management of the DDBS and makes the distribution transparent to the users is called a **distributed database management system (distributed DBMS).**

A DDBS is **NOT**, I repeat **NOT** a "collection of files" that can be individually stored at each node of a computer network. To form a DDBS, files should not only be logically related, but there should be structure among the files, and access should be via a common interface. It should be emphasized that the physical distribution of data is very important. It creates problems that are not encountered when the databases reside in the same computer. Note that physical distribution does not necessarily imply that the computer systems be geographically far apart; they could actually be in the same room. It implies that the communication between them is done over a network instead of through a shared memory, with the network as the only shared resource.

A DDBS is not a system where, despite the existence of a network, the database resides at only one node of the network (see Figure 12-1). In this case, the problems of database management is no different from the problems encountered in a centralized database environment. The database is centrally managed by one computer system and all the requests are routed to that site. The only additional consideration has to do with transmission delays. It is obvious that the existence of a computer network or a collection of files is not sufficient to form a distributed database system.

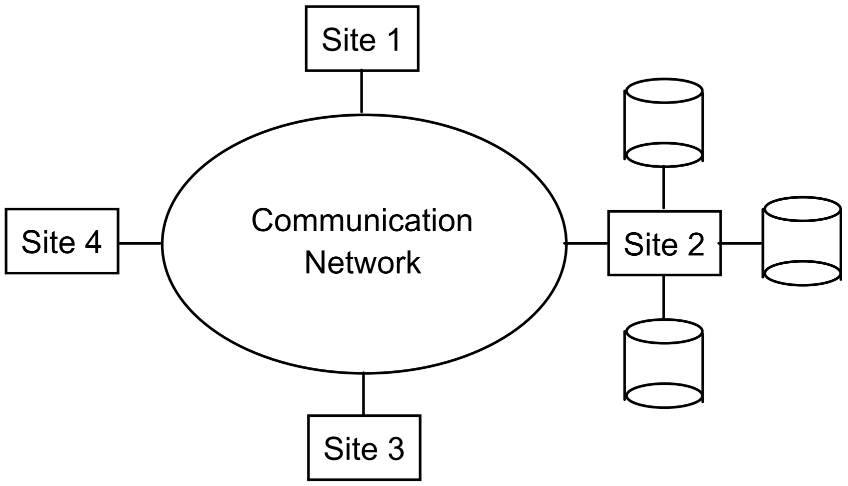


Figure 12-1

Example:

Consider a multinational manufacturing company with its world headquarters in New York, manufacturing plants in Chicago and in Montreal, regional warehouses in Phoenix and Edmonton, European headquarters in Paris, and a research and development facility in San Francisco. Such an organization will want to keep records of its employees, engineering operations, inventory levels, research projects, marketing operations, and much more. Under such circumstances, it is much more logical to manage the data and information flow as follows:

1. Let each location keep local records about the employees working at the location.
2. Let the research and development facility keep track of the information on projects that are going on in that facility.

3. Let the manufacturing plants keep data related to their engineering operations and give them means to access the information at the research facility.
4. Let the manufacturing plants keep track of their in-plant inventories. If necessary, they can access the inventory data at the warehouse locations.
5. Let the warehouses maintain their own inventory management systems with the necessary record keeping. The manufacturing plants should be allowed to access information regarding inventory levels.
6. The headquarters (both world and European) keep the marketing and sales data related to their regions. They can share these data and can access the inventory data at the plants and at the warehouses.

Assuming that all record keeping is done by computers, such an operation is an ideal candidate for a distributed database application like what the diagram in Figure 2 suggests.

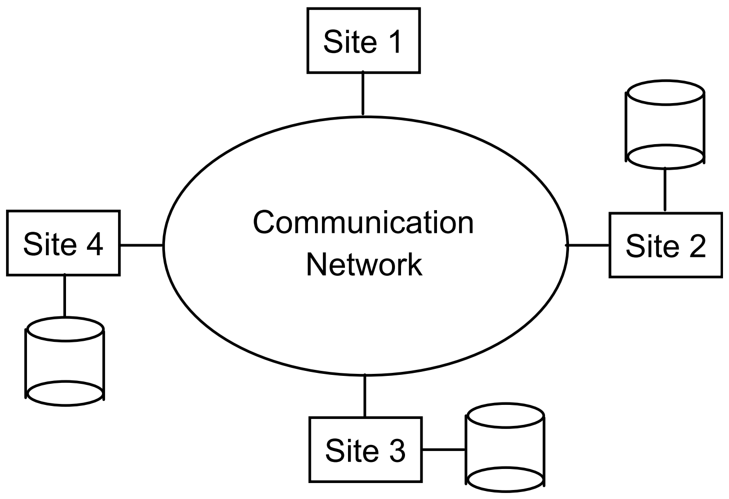


Figure 12-2.

Advantages of DDBS

The distribution of data and applications has promising potential advantages. Note that these are potential advantages which the individual DDBSs aim to achieve. As such, they can also be considered as the objectives of DDBSs.

Local Autonomy. Since data are distributed, a group of users that commonly share data can have them placed at the site where they work, and thus have local control. This permits setting and enforcing local policies regarding the use of data.

I**mproved Performance**. Again, because the regularly used data are readily available to the users, and given the parallelism inherent in distributed systems, it may be possible to improve the performance of database access. On the one hand, since each site handles only a portion of the database, contention for CPU and I/O services is not as severe as for centralized databases. On the other hand, data retrieved by a transaction may be stored at a number of sites, making it possible to execute the transaction in parallel.

Improved Reliability/Availability. If data are replicated so that they exist at more than one site, a crash of one of the sites, or failure of a communication link making some of the sites inaccessible, does not necessarily make the data impossible to reach. Furthermore, system crashes or link failures do not make the system totally inoperable. Even though some of the data may be inaccessible, the system can still provide limited service.

Economics. It is possible to view this from two perspectives. The first is in terms of communication costs. If databases are geographically dispersed and the applications running against them exhibit strong interaction of dispersed data, it may be much more economical to partition the application and do the processing locally at each site. Here the trade-off is between telecommunication costs and data communication costs. The second viewpoint is that it normally costs much less to put together a system of smaller computers with the equivalent power of a single big machine.

Expandability. In a distributed environment, it is much easier to accommodate increasing database sizes. Major system overhauls are seldom necessary; expansion can usually be handled by adding processing power and storage capacity to the network. Obviously, it may not be possible to obtain a linear increase in "power" since this also depends on the overhead of distribution. However, significant improvements are still possible.

Shareability. Organizations that have geographically distributed operations normally store data in a distributed fashion as well. However, if the information system is not distributed, it is usually impossible to share these data and resources. A distributed system therefore makes this sharing feasible.

Disadvantages of DDBS

However, these advantages are offset by several problems arising from the distribution of the database.

Lack of Experience. General-purpose distributed systems are not yet commonly used. What we have are either prototype systems or systems that are tailored to one application. This has serious consequences because the solutions that have been proposed for various problems have not been tested in actual operating environments.

Complexity. DDBS problem are inherently more complex than centralized database management ones, as they include not only problems found in centralized environment, but also a new set of unresolved problems.

Cost. Distributed systems require additional hardware, thus have increased hardware costs. However, the trend toward decreasing hardware costs does not make this a significant factor. A more important fraction of the cost lies in the fact that additional and more complex software and communication may be necessary to solve some of the technical problems.

Distribution of Control. This point was stated previously as an advantage of DDBSs. Unfortunately, distribution creates problems of synchronization and coordination. Distributed control can therefore easily become a liability if care is not taken to adopt adequate policies to deal with these issues.

Security. One of the major benefits of centralized databases has been the control over the access of data. Security can be controlled in one central location, with the DBMS enforcing the rules. However, in a distributed database system, a network is involved which is a medium that has its own security requirements. It is well known that there are serious problems in maintaining adequate security over computer networks. Thus, the security problems in distributed systems are by nature more complicated than in centralized ones.

Difficulty of Change. Most businesses have already invested heavily in their database systems, which are not distributed. Currently, no tools or methodologies exist to help these users convert their centralized databases into a DDBS. Research in heterogeneous databases and database integration is expected to overcome these difficulties.

SAQ 12-3

You are down to your last SAQ. Take a deep breath and relax. Let's see if you can get a perfect score in these review questions. For each of the numbered items below, encircle the letter of the best answer.

1. A distributed database system (DDBS) is a collection of multiple, logically interrelated databases distributed over what kind of network?

a. computer	c. satellite
b. neutral	d. television

2. If the database resides at only one node of the network, the problems of database management are ________ the problems encountered in a centralized database environment.

a. dependent on	c. more complex than
b. less complex than	d. the same as

3. Organizations usually store data in a distributed fashion for geographically distributed operations. The use of a distributed information system allows sharing of these data and resources. This DDBS advantage is called

a. expandability	c. local autonomy
b. improved availability	d. shareability

4. Although DDBSs inherently possess several problems, research in heterogeneous databases and database integration is expected to overcome specifically what disadvantage?

a. complexity	c. difficulty of control
b. difficulty of change	d. lack of experience

Summary

In this module, object-oriented programming concepts were tackled and the corresponding assimilation of these topics to database systems were introduced. The main point considered in this area is that the user should not have to wrestle with computer-oriented constructs like records and fields, but rather should be able to deal with objects and operations that more closely resemble their counterparts in the real world. The discussion basically concentrated on how object-oriented concepts were incorporated into databases.

Next, we discussed a relatively new branch of database management called deductive database systems. This field has been influenced much by logic programming languages, particularly by Prolog. The text explained what deductive systems are all about from the viewpoint of someone who is familiar with database technology but not necessarily with logic programming. Thus, I explained the basic concepts and gave some examples that illustrate the representation of facts and rules, the two most important elements in deductive databases.

Lastly, we looked at some features of distributed databases. We discussed the reasons for distribution and the potential advantages of distributed systems over centralized systems. Not to be biased for distributed systems, we also covered the disadvantages of using DDBSs.

You have reached the end of this course. clAP! cLaP! CLaP! <canned applause>. I hope you enjoyed studying database management systems. I presented to you merely the concepts; there is a lot more to learn out there. Carpe diem!

References

Date, C J. (1990). *An introduction to database systems.* Reading, Massachusetts: Addison-Wesley Publishing Co., Inc. 5th ed.

Elmasri, R., & Navathe, S.B. (1994). *Fundamentals of database systems.* Redwood City, CA: The Benjamin/Cummings Publishing Co., Inc.

Ozsu, MT., & Valduriez, P. (1991). *Principles of distributed database systems.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Silberschatz, A., Korth, HF. & Sudarshan, S. (1997). *Database system concepts.* Singapore: The McGraw-Hill Companies, Inc. International ed.

Answers to Self-Assessment Questions

ASAQ 12-1

The correct markings are given below:

1.

2.

3. $\overset{\textit{references}}{\text{multiple versioning}}$

Statements 1 and 2 are both TRUE. In fact, as support to statement 1, it has been stated many times that the very foundation of OODBs is OO programming. For statement 2, it has been implied that the improvement of OODBs over traditional databases is its capability to meet the needs of more complex database applications. And the features that enables OODBs to handle complex databases better are found in its ability to represent complex objects and all operations.

Statement 3 on the other hand is FALSE. The phrase "multiple versioning" makes the statement false and to make the statement true, this phrase should be replaced by the word "references." Review the text for a refresher.

Remember how many correct answers you got from this SAQ. We'll add those with the correct answers you'll get on the two remaining SAQs. Later, at the end of this module, you'll see how you fared in a chart I've prepared specifically for this purpose.

ASAQ 12-2

The correct matches are 1. b, 2. e, 3. a, 4. c, and 5. d. Did you get all five correct answers? If so, then that's superb! Three to four correct answers is still impressive. However, if you got one or none, I think reading the text once more is a good idea. Add your score to the score you got in SAQ 12-1. Jot it down if you must; just don't forget your accumulated score.

ASAQ 12-3

Easy as 1, 2, 3 and a, b, c. The answers are 1. a, 2. d, 3. d, and 4. b.

Four out of four calls for a celebration. Three out of four is still very good. If you got a lower score, you might want to review the material again.

A perfect score in all of the SAQs will give you a total score of 12. I prepared a chart below that you may find interesting. If we consider all the SAQs as a qualifying exam for Mensa (an international organization whose sole requirement for membership is an IQ in the "genius" category), then how would you fair? Let's find out.

IF YOU SCORED:

11 or 12 points	: You are exceptionally intelligent - a perfect Mensa candidate!
9 or 10 points	: This should put you in the upper 2 percent of the population, brainwise - definitely Mensa material.
7 or 8 points	: An honorable score. You might want to try Mensa.
Less than 7 points	: Forget about joining Mensa. But you're still in good company - many world-famous figures (writers, artists, actors, politicians, business tycoons, etc.) don't have exceptional IQs either! And whatever your score, don't take this chart too seriously. The SAQs are not credited by Mensa anyway. But a review of the topics is called for. You might want to do extra work on the different topics discussed in this module. References are listed at the end of this module.