ORACLE

Oracle Press Chapter Sampler

Exclusive to Oracle OpenWorld and JavaOne 2015 Attendees!

Oracle SQL Developer Data Modeler for Database Design Mastery

Oracle Press™

www.OraclePressBooks.com • @OraclePress

CHAPTER 1

Saibot Airport Reaching for the Future

This book introduces you to Oracle SOA Suite 12*c*. It will show in great detail how the many features and functions of this rich set of products can be used and tied together. The book also tells the story of Saibot Airport—an ambitious international airport with a clear business vision about where it is going and a great need for IT solutions to enable these ambitions. Saibot Airport and its requirements constitute the case that provides the backdrop against which the SOA Suite is used. Various aspects of running and evolving the airport are used to illustrate the usage of the functionality offered by the SOA Suite.

This chapter introduces Saibot Airport as an organization with a vision and a business strategy, and one that depends heavily on IT to fulfill the strategy. The IT department itself is confronted by changing industry trends, changing regulations, new technology, and an evolution in the way it organizes its processes. From all of these, architecture consequences are derived. And finally, technology products have to be selected to start the realization of the information and application architecture designed to enable the IT and business objectives. Given the title of this book, it will come as no surprise that this made up Saibot Airport selects many components from the Oracle Fusion Middleware stack including the SOA Suite. The next chapter introduces this stack, the SOA Suite, and the role the SOA Suite plays.

Saibot Airport

Saibot Airport used to be just another mid-size airport near the great city of Lexville. It has been only a domestic hub for a long time, but for the last decade, its role in the region has become more prominent. It is very much the desire of the management team at the airport to continue and extend this trend and turn Saibot Airport into a major international hub, not dissimilar to what Dubai International Airport has achieved. The board of directors and the shareholders enthusiastically support this idea.

Like any airport, Saibot Airport offers facilities that enable airlines to provide services to passengers and logistics companies. The infrastructure includes terminal buildings, runways, parking space for planes, fuel depots, safety equipment, check-in desks and kiosks, various types of vehicles for transport inside the terminal as well as on the platform, luggage processing equipment, bird scaring machinery, and different types of flight information systems.

The airport primarily acts as a broker in services—bringing together parties offering services and parties looking for providers of such services. Examples of such services are cleaning, repairing, fueling and deicing the airplanes, handling luggage, providing security, catering, and safety. In the wake of the core activities at the airport, focused on flying, there is a wide range of other commercial activities. Saibot Airport has shops, restaurants, hotels, car rental companies, car parks, meeting rooms, office space, and entertainment areas—that are used to provide services to passengers and other visitors. Some of these services are offered by subsidiaries of Saibot Airport itself—but the vast majority are delivered by concession holders: companies that have paid for the (sometimes exclusive) right to provide certain services at the airport and rent facilities to do so. As a result, out of the many thousands of people working at the airport, only a minority is employed by Saibot Airport Corporation itself. Most are staff working at one of the hundreds of business partners active on the airport.

The airport may not do any flying itself and only execute a small portion of the activities, it does have a number of important and overarching responsibilities. These include overall safety and cleanliness of all facilities, for example, security on the ground, the technical condition of buildings and equipment, and up-to-date and accurate information with regard to flights and many other aspects of the operations on the airport.

Saibot Airport has interactions with many parties, both local and much further afield. These include the business partners active on the airport, travelers and their friends and relatives. Many interactions take place with local authorities and central government agencies, regarding topics such as environment, taxes, security, and safety.

Business Vision and Strategy

As mentioned before, Saibot Airport wants to expand and become a more important player at the *national* and *international* level. The associated increase in the number of flights to and from the airport and the number of passengers visiting the airport will drive up revenue and profit.

The potential for the growth is there, as extensive research has shown. In order to tap into that potential, the airport has to become more attractive for commercial airlines to use Saibot Airport as their hub. This is to be achieved in several ways, including offering attractive rates for using the airport facilities, ensuring a wide range of high-quality services, enabling airlines to operate very smoothly on the airport and making the airport especially appealing to travelers.

Attractive rates and smooth operations depend on a very efficient organization and implementations of processes and systems at the airport with a high rate of automation.

Getting very favorable traveler ratings is crucial. Traveler satisfaction depends on many qualities—and is not easy to obtain and retain. What matters most to passengers is a quick and painless check-in process and basic comfort in terminals. Other elements are airport accessibility, baggage claim, terminal facilities, security check, and food and retail services. Additionally, airport facilities have to be clean and good looking and within easy reach. Information plays a major role in the traveler's experience: info on how to get to the airport and how to find one's way around the airport, information on the flight and the check-in and boarding process, notifications on (changes in) the status of the flight and the ability to get quick and accurate responses to questions about airport facilities and flight details.

Any growth at Saibot Airport has to be coordinated with local and central government bodies. Safety, security, and logistics on and around the airport are to be orchestrated across the area. Especially, relevant for any growth scenario are environmental issues. Noise pollution, carbon dioxide emissions, energy consumption, and waste production are among the aspects that could constrain the growth in air traffic, unless handled carefully.

Business Objectives

To facilitate the longer term vision and goals, Saibot Airport has identified concrete business objectives it has to pursue. These are to be achieved or at least facilitated by the IT department

Modern interaction channels have to be introduced that allow 24/7 access that enable airlines to (re)schedule flights and to acquire services around these flights. Passengers and their relatives should have round-the-clock access to flight information as well as data on shops, restaurants, parking options, and travel times to and from the airport. Among these channels are to be a B2B service, a web portal, and a mobile application.

In 2018, Saibot Airport should be paperless. All information required, for example, for scheduling a slot, requesting a concession for a restaurant or renting a shop should be submitted in electronic format—both the forms and the supporting documents. Having all incoming information in digital format should reduce the workload on the inbound side of the airport's operation quite dramatically. The lead time from the moment the request is submitted to the moment a staff member can actually start working on it is should be shrunk to almost nothing thanks to the digitization. Furthermore, having multiple staff members work on the same request at the same time will finally be standard procedure—whereas today because of the single copy paper-based file this is only done in exceptional situations.

The learning curve for new staff should be much shorter than today. Saibot Airport's workforce is fairly flexible with a high turnover in several roles. A lot of money can be saved if new staff is productive in a much shorter period of time and will make fewer mistakes (currently caused by user interface unfriendliness). Management at the airport also requires more insight into the actual proceedings. It wants to be aware of delays in process execution, bottlenecks, and other process inefficiencies. Furthermore, it wants to be able to continuously improve business processes—without long lead times, massive development effort and high risks. They have heard the phrase "embrace change" at some agile seminar—and they like it. They desire the flexibility and short time to market promised by the agile evangelist.

Efficiency must be the name of the game. To be able to offer attractive rate to airlines and scale operations to the levels envisioned, the airport needs to make its operations more efficient. Marginal operational costs per flight must be reduced by at least 20 percent. Even though it is yet to be decided exactly where those costs are to be saved, it is obvious that downsizing the manual labor per transaction has to be a major part of the meeting the efficiency demands. By having business partners submitting all information in electronic format and by making all information about the progress of such requests available on line—a large part of the current workload will be taken away. Self-service through portals and mobile applications seems all about customer satisfaction—yet it can also work miracles in terms of cost savings on the part of the airport. Simpler applications with short learning curves and requiring less business understanding and process expertise should allow Saibot Airport to work with temporary workers—creating a flexible layer that can shrink and grow with the actual workload. This not only applies to the Saibot Airport organization itself but also to the many companies active on the airport. That in turn affects the airport—for example, through the frequent and usually urgent processing of security accreditation requests for new staff.

Smooth operations are crucial—from a cost perspective as well as the travelers' experience. Check-in, security, and boarding procedures are all too often hindered by inaccurate or incomplete information about the passengers or some flight details. This also applies to processing of luggage, informing all stakeholders of changes regarding the flight and coordination of catering, fueling, and cleaning the plane prior to departure. Ensuring the rapid electronic exchange of up-to-date and complete data regarding the flight schedule, the passengers and crew and the services to be provided to the air plane is crucial to be able to operate smoothly and efficient and to scale up these operations.

IT Objectives

The CIO—guided by his team of architects and with a clear link to the overall goals set by the board—has made sure that a number of specific IT objectives have been included in the program's design. She wants to ensure that the vital role IT plays for the operation of the airport is recognized and that a clear statement is laid down with regard to IT that serves as the starting point of the IT roadmap.

From the overhead objectives, it is already obvious that IT plays a large part in realizing the desired move toward the future. More specific statements in the program about IT are also included.

An enterprise architecture design has been created and serves as the foundation for all future projects. Modern IT architecture patterns will be adopted to translate the enterprise architecture into IT architecture and subsequently into designs of applications and infrastructure.

The architecture and the technology selected have to allow for flexibility: changing functionality should be possible in a simple, cheap, fast and risk free manner. Saibot Airport wants to go *agile*, both in business and in IT. Furthermore, the evolution of the systems, the transitions to new systems all have to take place while the shop stays open. The airport clearly cannot afford to close down, especially once 24/7 online channels (web, B2B, mobile) have been introduced.

Saibot Airport's IT should be based on current industry standards that are open and promote interaction and reuse. It should not be on the bleeding edge of technology and only use concepts and components that have been proven. At the same time Saibot Airport's IT has to be up to date in order to appeal to both staff and clients and allow the airport to find the right resources to help design, develop, and administrate the infrastructures and systems. There is neither special preference for nor aversion against open source software. However, Saibot Airport has found out the hard way that it should only use software that has a large community around it and one or more large commercial parties backing it.

It is the airport's intention to work with a small number of strategic vendors that have a broad product portfolio, a clear roadmap, ability and intention to keep evolving and a willingness to cooperate and ideally take responsibility for Saibot Airport's success with their products. Slideware won't do: the potential of the vendor's products has to be demonstrated through customer references. A technology (and vendor) selection has been made by the airport, which included consultation with industry analysts.

The airport is opening up to the outside world. In the past, many of its interactions beyond the perimeter of its physical site were on paper, through fax or telephone. Until not too long ago, its main online interaction comprised of email and a read-only website based on a database in the DMZ that was refreshed once a day from a file dump with real-time actual flight information. Now, however, real-time synchronous interactions with the enterprise systems will have to be supported, in the B2B exchange, for the mobile apps and for a much more interactive, real-time web application. Security has to be at the heart of these initiatives. It is imperative for certain information to be only made available to authorized parties—and hence it is crucial to identify any party dealing with the airport in a secure way.

The integrity of the data will also play an even more important role; automated processes do not have the same capacity as humans to cater for inconsistencies or simple typos in data. Furthermore, because the data from internal systems will be published directly to portal and B2B channels, without human checks and filters, the quality of the data has to improve beyond what it is today.

Part of the plans is reducing IT expenditure by consolidating onto a central infrastructure with a single source of truth for each data domain. This should also help with the quality of data—with far less data duplication and replication. This also should have the effect of lower hardware costs, lower software license expenses, and a downsized administration staff. In the initial stages of the program's execution, however, it is envisioned that IT spending will increase because of the many projects that will have to be carried out in order to meet the objectives. Saibot Airport is looking closely into the possibilities of making use of cloud service to achieve not only the consolidation but also the ability to achieve "web scale" IT operations that as a mid-size organization they would never be able to realize on their own. Leveraging cloud services would also allow the airport to graciously handle temporary increases in demand for IT infrastructure capacity without making structural investments.

The intended consolidation of all IT infrastructure and all data into a single instance means on the one hand a relief for the security officer as it means fewer sites, administrators, and environments to worry about. However, this consolidation means that availability, which in the 24/7 world of Saibot Airport is essential, becomes a much harder challenge. The consolidated systems, logically a single instance, are the critical factor in virtually all of the activities. It is a single point of failure—at least logically. Part of the IT roadmap is taking measures to safeguard the availability of all IT components—for example, through clustering and fail-over.

Architecture to Enable the Future

The architecture team at the airport leads the way in terms of technology evolution. This team has drawn up the high level IT architecture, selected and fine-tuned the architecture patterns that are to be applied and worked with developers to design the reference architecture. This reference architecture provides guidelines on how to make use of architectural patterns when designing system components and how to make use of the selected technology to implement these patterns. It also provides a common vocabulary with which to discuss implementations. A crucial architecture product is the roadmap that defines how Saibot Airport can go through the transition from the current to the to-be situation.

Partly based on these architecture designs, the strategic technology and vendor selection is made; it is after all imperative that the vendor and his product portfolio is capable of implementing the architecture as designed by the team.

The legacy at Saibot Airport involved a classic case of application *silos*: stand-alone units that consist of a database, business logic, and a user interface. Each application is implemented through its own silo—using distinct and sometimes very proprietary technologies and maintained by fairly inward facing, somewhat self-absorbed teams. Flexibly sharing resources between these teams is neither common nor easy. Technical integration between the silos for exchanging and ideally truly sharing data is hard to achieve too; frequently files are used to export and import data in an asynchronous batch process that may involve manual actions as well. Data replication is common as is the human task of typing to reenter data: even though data may already exist in one silo that does not mean it is accessible to another. Because data exchange is not readily available, manually keying in that same information is frequently the easy way out.

Breaking up the silos is an absolute requirement in the new architecture. Lasagna style is on the menu: a layered architecture with clear responsibilities assigned to each layer and well-defined interfaces describing the interactions between these layers. Figure 1-1 illustrates this transition.

No single team or application is owner of what essentially is and always should have been treated as enterprise data that can be used in many different processes. Teams and projects are not masters of their own destiny: decisions they make regarding technology, application layout, and implementation patterns are part of the enterprise landscape and have to fit in.

The layers identified in this architecture:

- User interface and programmatic interface layer—the interaction with the outside world through human-oriented as well as system-oriented interfaces
- Business layer—common interface to data and business logic reusable across user and programmatic interfaces
- Data layer—the persistency and integrity of all enterprise information assets including documents and other unstructured data

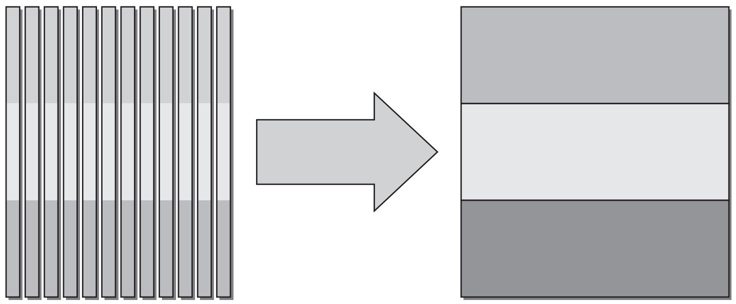


FIGURE 1-1. *Transition from application silos to a layered architecture*

Identifying these layers will help to establish a clear *separation of concerns*. Each layer has its own role using its own design patterns and designated technology. The implementation of each layer is encapsulated to other layers. Layers can only invoke the next lower layer. Layers are unaware of any other layers except for the one directly underneath them. Communication is always started at the top, flowing downward.

Each layer should have clear interfaces defined that describe the interaction that it supports. Part of this interface description is the definition of the operations that can be invoked in the layer, the input they require and the output they return—including exceptions that can be thrown—as well as a description of side effects of the call, such as emails being sent, products being shipped, or data being persisted. Nonfunctional aspects of operations should also be described; these include availability (opening hours), costs, authorization and other security aspects, response time, and accepted volumes.

One of the key decisions made in the early stages of the architecture design was the adoption of many service-oriented architecture (SOA) principles. These principles include decoupling (well, loose coupling at the very least), abstraction and encapsulation, reusability, and location virtualization. Applying these principles will help to implement the layered architecture and will play a large part in the flexibility, short time to market, efficiency through reuse, and risk reduction that the business requires.

The Triangle

An increasingly important role in discussion about Saibot Airport's IT future was played by a very simple illustration. Basically nothing but a triangle, with its base at the top; see Figure 1-2.

This triangle visualizes the distinction between the layers in terms of their reuse potential and generic nature versus their multichannel support and level of customization.

The data layer is characterized by centralization and (logical) consolidation. Data assets have a single source of truth. This layer has a very high reuse potential and generic, enterprise-wide structure. As a core enterprise resource, requirements in quality, integrity, availability, and confidentiality are very high. The rate of change at this level is fairly low—at least at the meta-level. Data is not removed very frequently and the data structures evolve even slower. Note that big data and fast data are a special kind of data—raw data that serves very operational goals or undergoes substantial processing before ending up in this enterprise data layer.

The top layer that exposes interfaces to users and applications is quite different. It sports a large variety, catering to very specific channels, consumers, and user roles—allowing customization and

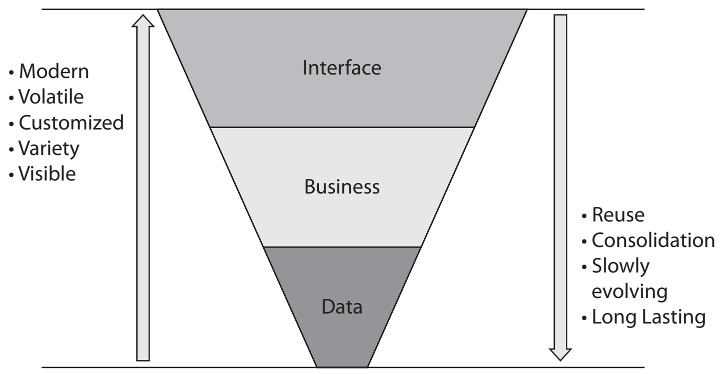


FIGURE 1-2. *The characteristics of the three main layers in the layered architecture*

personalization to meet special requirements. The average life time of components in this layer is fairly short, especially compared to the data layer, and the rate of change is much higher.

The business layer, the man in the middle, is also in the middle in terms of reusability and rate of change. It offers services that are aimed at reuse by various different interface components. This layer brings together various assets from the data layer, implements business logic, validates, processes and enriches data, and runs processes. The rate of change is higher than at the data layer as is the functional variety. Compared to the user and application interface layer, however, this layer evolves much slower, is much more focused at reuse and caters for far fewer specific, one-off needs.

Most business requirements are expected to find the majority of the required effort in the top layer, a sizable portion still in the middle layer and very little effort in the bottom layer. The triangle therefore also represents the work ratio in many development projects and therefore suggests a team composition.

The transition IT at Saibot Airport is undergoing is now characterized at a very abstract level by Figure 1-3.

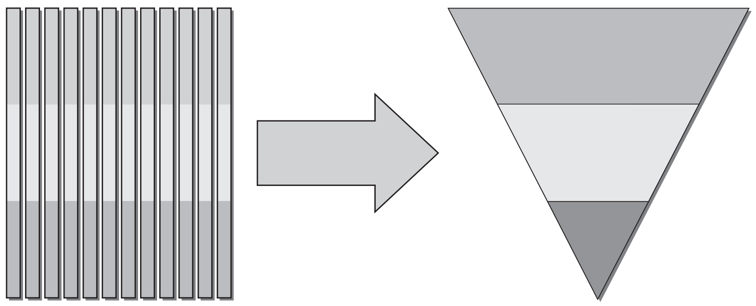


FIGURE 1-3. *From application silos to the layered architecture with varying degrees of consolidation and change*

Domains of Data

The enterprise architecture has identified domains within Saibot Airport—relatively independent areas. Data in each domain and services exposing that data are to be controlled by domain experts. There are no direct relationships between components in different domains. The thinking should be that at any time a domain could be reimplemented using a commercial off-the-shelf offering such as a SaaS CRM system or a third party expertise application.

Some of the domains identified by the architects are: common (reference data), relations, flight slots and schedules, security, concessions, documents, and finance.

Most interactions within and across domains will involve data. A common understanding of and language for interacting in terms of the data is essential. The architects have launched an initiative to create an Enterprise Data Model (EDM). This model describes all business objects that are meaningful to Saibot Airport's operations, including their properties and relationships. Common terminology as well as lists of reference values that are to be used to set the value of certain properties are defined in a standardized way and made available throughout the organization. Note: The model is defined at the business layer and it may not necessarily be fully aligned with the database structures and other technical assets inside each of the domains. The EDM is the common business language across all of Saibot Airport—stretching beyond IT. All interactions between the business layer and the data layer will be in terms of the EDM. Note that operations inside the data layer will probably use existing idiom and structures for quite some time to come—which is unavoidable and perfectly acceptable.

Service-Oriented Architecture

The decision to make service-oriented architecture the leading architecture principle has a number of consequences. Middleware is still fairly new at the airport. Teams used to be organized around applications—around the silos that were discussed earlier on. The business layer at the heart of the layered architecture will be the new focal point for all teams. This layer is made up of a number of different types of services. These services bring together all data from the data layer—structured and unstructured, across databases, document repositories, LDAP instances and mail servers—and expose access to the data in a standardized, unified way. Moreover, these services make business logic available to applications—both user interfaces and programmatic channels.

Part of the foundation of the business layer is the Enterprise Data Model (aka Canonical Data Model) and more specifically: an XML representation of that model, expressed in terms of a heavily annotated XSD (XML Schema Definition). All data structures handled by the services, both input and output, are defined in correspondence with business objects in this canonical XSD. The data domains are recognizable through the namespace structure used in these XSD definitions.

The architecture team came up with a service classification scheme that helps organize the services as well as the teams working on those services. Following this scheme, the business layer is subdivided in these types of services (as illustrated in Figure 1-4).

- Elementary services that provide atomic functionality within a domain; their reuse potential is high, their added value usually is low
- Composite services that combine two or more elementary services into business functions with higher added value; composite services come in two flavors:
  - Within a domain
  - Across domains—typically introducing the need for either global transactions or transaction compensation; even such cross-domain services should have a single owner—perhaps a designated domain

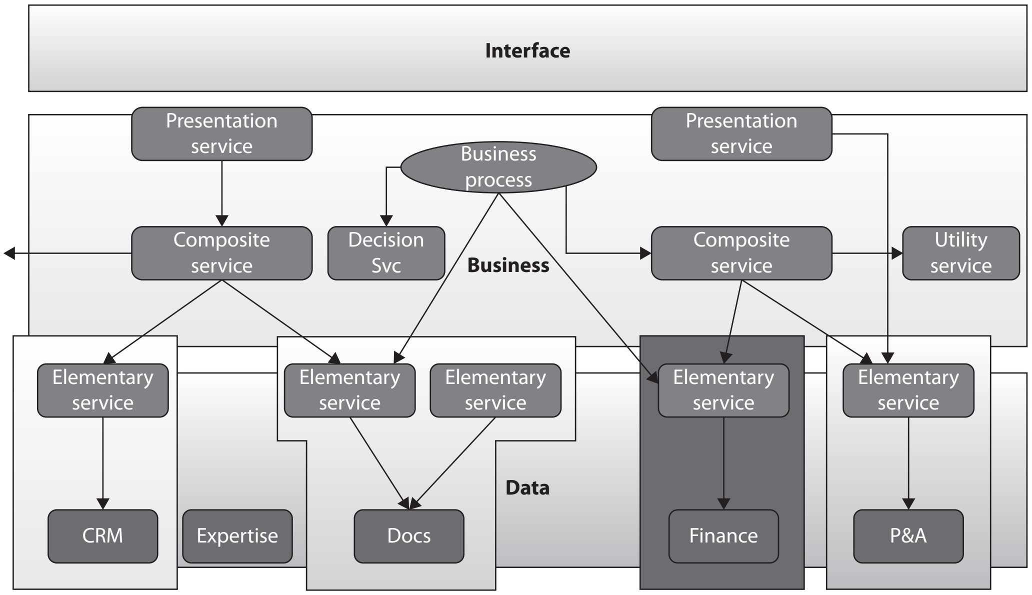


FIGURE 1-4. *Domains and service types at Saibot Airport*

- Process services that are often asynchronous and longer running (up to minutes, hours, or even days) and that will usually contain state while running
- Presentation services that are usually not meant for general reuse; instead they cater for specific needs of an application—either a user interface or a programmatic interface, speaking the language of that application as closely as possible
- Utility services—generic, domain independent, highly reusable services, frequently of an almost infrastructural nature; for example, logging, sending emails, value translation, and geocoding

Service design and implementation guidelines are created per service category. Governance, ownership, testing and many other aspects of the services also depend on the type of service. Saibot Airport ended up closely aligning its team structure with these categories of services—as we will see in a later section.

NOTE
Except for presentation services, the service interfaces are expressed in terms of the canonical data model. They are all recorded in a central service catalog where potential consumers will find information about the service including functionality, contract, nonfunctional aspects and status. At Saibot Airport, this service catalog started life as a simple Wiki that references the live WSDL (Web Service Definition Language) and XSD specifications of services.

Event-Driven Architecture

For the architecture layers it was stated that a layer cannot invoke a higher layer—or even be aware of it. The same applies to the service categories: a service is unaware—and therefore completely independent—of higher level services. It cannot directly initiate communications with higher level services. This means, for example, that an elementary service cannot invoke a composite service or process service; for all intents and purposes, it may not even know such higher level services exist.

This does not mean however that a lower level layer or service will never have something to tell that could be of interest to a higher level service. It means there has to be another way of communicating that information then telling it in a direct call. To address this challenge, the architects have adopted elements from Event-Driven Architecture (EDA). Events are used as the very decoupled vehicle to convey information without direct any dependencies between the source and the recipient(s) of the information.

In addition to defining the canonical data model and identifying the services, Saibot Airport's information analysts are working on discovering business events. A business event is a condition or situation that may come about somewhere in the airport's daily operations that is potentially of interest to other parties. Events of interest to a component within the Saibot Airport landscape can of course also take place in the outside world; these too classify as business event.

A business event is described by a name or type, a timestamp and a payload—data that clarifies what the event entails. Some examples of business events at the airport are: weather alert, (outbound) flight has been cancelled, airline has filed for bankruptcy, deadline has expired in some business process, business rule has been changed.

The reference architecture describes an event handling infrastructure—a generic facility that is available to all application components and services alike—as shown in Figure 1-5. Anyone can

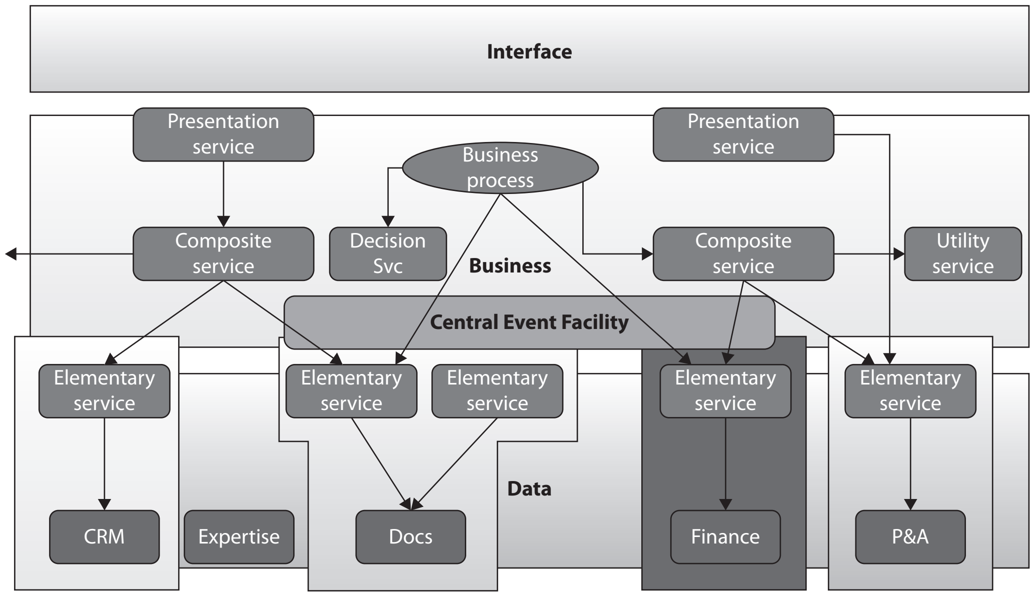


FIGURE 1-5. *The Central Event Facility handles events in an extremely decoupled approach*

publish a business event to this event handler—provided the event type is predefined and the payload has the predefined structure. After publishing the event, the publisher is not involved in any way with the delivery of the event and does not even know if the event is consumed by any party at all. Any component at any layer in the architecture can subscribe to selected types of business events. The event handler will push any published event to all subscribers to the type of this published event. Consumers of the event will receive the event with its payload and can do with it as they see fit. They are not aware of the component that published the event.

The perfect decoupling achieved through the events makes it extremely simple to add consumers of a particular type of event or to introduce new publishers of some event type. Removing subscribers to an event is another zero impact procedure, as is losing one or more publishers of events.

Through events, elementary services and even components in the data layer can tell their story that may be of great interest to composite or process services or even to user interface components—without ever knowing about them. The interaction can take place, but in an entirely decoupled fashion.

On Technology and Vendor Selection

The business objectives and the derived layered architecture as well as the more detailed architecture principles result in a clear image of the technology components required by Saibot Airport. Figure 1-6 shows the most important components that need to be implemented through technology products that will have to be selected and acquired from one or more vendors.

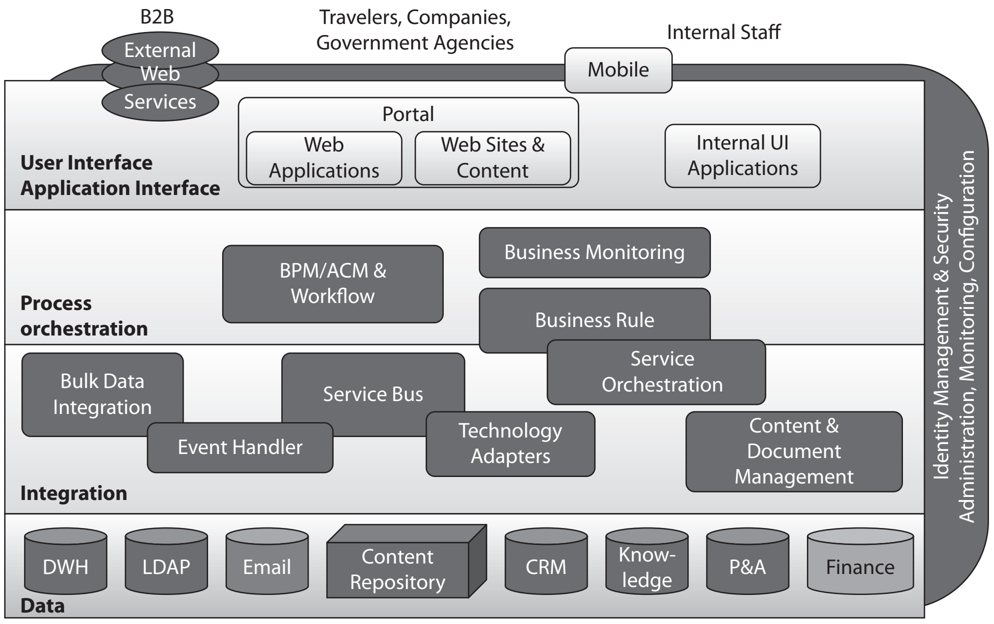


FIGURE 1-6. *An overview of the key technology components that Saibot Airport needs to select and acquire*

Saibot Airport has stated from the outset that it does not want to buy into a single integrated suite of products only because it is a single integrated suite. It wants the best of breed: the best product available in each category. It requires all products to be open, standards based and easily integratable.

A number of additional criteria were determined, based upon which products would be evaluated.

The airport does not have IT as its core activity. It wants to use proven and supported technology and products, backed by verifiable references. It needs a product to have a clear strategy as well as a strong community and an abundance of resources. The latter refers both to qualified IT professionals and to books, internet forums and blogs, training material, etc. The learning curve for a product should be clear and justifiable—judged against the existing workforce.

Any product should have strategic importance to its vendor. Besides, the vendor (or open source project) should be stable and future proof. Analyst reviews are retrieved and taken into consideration.

Saibot Airport does not want to have a large variety of technologies and platforms that require different skills. Even though it is a substantial organization, resources are limited. It wants to focus on a small number of major industry platforms—hardware, O/S, virtualization, database and middleware—as to prevent a nightmare for the administrators. All products selected should provide sufficient options for monitoring and configuration and they should allow for automated build and deployment.

Of course the cost of the software is an important part of the decision. Here, the selection process will look at a number of aspects. What is the license fee per pricing unit—user, CPU core—and what is the estimated number of pricing units. What is the yearly support fee and what are pricing elements play a role? What discounts can be negotiated? How long are the licenses valid for? What alternative constructions are available—such as a subscription fee? Furthermore, what assurances about the software behavior is the prepared to make? What SLAs will it enter? Is the vendor prepared to take some form of responsibility for the successful implementation of the software—possibly a no cure no pay construction or a fee that is partly based on the return on the investment. What are the options for using the software *from the cloud* to complement an on premises investment with a pay-per-use, scalable capacity?

Selection Approach

Saibot Airport had to make two selections that cannot be separated. It had to select one or more vendors and it needs to select the products that will implement the functions identified in the architecture.

It published a *Request for Information* in which it invited any vendor to answer questions for each of the components identified in the target architecture. This RFI focused on checklist on features, technical characteristics, implementation requirements, training, and order of magnitude pricing. Responses were received from vendors of niche products, specialists in very specific areas as well as reactions from vendors of suites of products that covered wide ranges of products. It also invited a number of parties to represent open source products.

In parallel with the RFI process, the airport gathered information from peer organizations—largely governmental—about product and vendor experiences. Saibot Airport also consulted market analysts—both online and in person.

The information received in this first round from the RFI and the parallel explorations was screened and evaluated, based on the criteria listed overhead. This resulted in a short list of both products and vendors. The vendors on this short list were invited for the next stage: *request for proposal*.

In this stage, vendors were asked to present a plan for how their product(s) could best be used by the airport—including the infrastructure topology, the licenses, the migration of systems and transition of processes and staff. The proposal needed to cover the overall price as well as any alternative compensation proposals such as deferred payment, result-based payment, subscription-based fee, and usage-based fee.

Each vendor had to present relevant customer reference cases that could be contacted and visited. It also had to lay down the product roadmap and longer term vision and strategy. Saibot Airport wanted to be convinced that the products presented were indeed future proof.

By this time, the airport had decided to take a slightly different approach. It defined several technology clusters that it wanted to select in somewhat separate stages. Roughly like this:

- Hardware—no immediate investment; virtualization using VM Ware was the short-term way forward; investigations are started into a multisite, disaster proof, very high availability data center setup.
- Database—consolidation on Oracle Database 12*c*—using the multitenancy option for consolidation (which means upgrading some databases and replacing some MySQL and SQL Server databases).
- Mobile—no strategic selection is made for now; given the volatility in the mobile market and the fact that mobile applications are on the very outside of the enterprise with little impact and a short lifespan, it was deemed unnecessary to make a strategic selection for mobile technology; being able to support mobile applications by exposing relevant (REST) services was deemed much more important.
- Internal user interface applications—for quite some time to come, some Oracle Forms applications will be used and maintained; one application has been migrated to (or rather rebuilt in) Oracle ADF 11*g*. The airport has evaluated its experiences with ADF and decided that for now it has no reason to select a different technology—only switch to the latest version of the framework (12*c*). It did of course verify the strategic importance of ADF for Oracle, the strength of the community, the roadmap and vision and the pricing, and found them satisfactory. The availability of external resources is somewhat worrying, although the realization that ADF is a Java EE framework that any Java Web Developer can quickly embrace by and large took care of that worry.

For the following clusters, a separate Request for Proposal is conducted:

- Service Oriented Middleware—Saibot Airport needs products to implement an enterprise service bus, service orchestration, Event-Driven Architecture, technology adapters; these products have to work together well and ideally use the same platform.
- Process and human workflow management—in this cluster, Saibot Airport identified the need for decision rules (aka business rules), business process orchestration, human tasks and workflow management, and business activity monitoring (BAM); these products have to be able to closely work together.
- Enterprise content management—one of the business objectives is the complete eradication of paper; working with digital documents is crucial for the organization. It requires

products to store, search and publish, convert, tag and track, archive and protect digital documents in various formats. These products should of course fit in the service-oriented architecture that will be established.

- Portals and external user interface applications—because of the decoupling achieved in the layered architecture and the use of services, the dependencies between portals and other external interfaces on the one hand and the products in the business layer on the other are minimal; that means that a decision on the products used for these portals can be made independently of the other product selections.
- Identity and access management—Saibot Airport is opening up its enterprise applications to users outside the organization; this new situations calls for a new approach to management of identities, the implementation of authentication and the authorization based on the identity; the airport currently uses Active Directory for its internal staff and is not keen on abandoning that platform (as it is integrated into the overall office automation). It wants to introduce products that will handle identities and authorization for external users. It also requires products that handle encryption, digital signatures, and other security techniques that are to be applied to certain services.

Selections

Most of the products in the clusters *Service-Oriented Middleware* and *Process and Human Workflow Management* that made it onto the short list were based on the Java EE platform. Added to this, the fact that the technology for the internal applications was set as ADF 12*c*, another Java EE–based technology, and good old Oracle Forms—also running on the Java EE platform, Saibot Airport decided to choose Java EE as the platform for all its middleware. It also selected Oracle WebLogic 12*c* as the Java EE application server of choice. Only when a best of breed product would be selected with superior functionality that would be unable to run on WebLogic could another application server be considered. Note that the team at Saibot Airport very specifically kept open the possibility to support a different platform in the User and Application Interface layer. Some internal politics may have been part of that decision; there was some resistance against going Java all the way from some of the .Net-oriented teams.

The product selection for enterprise service bus and service orchestration (the latter quickly translated to BPEL) evaluated among other Microsoft BizTalk and various Tibco products as well as some open source offerings and then decided on Oracle SOA Suite including the Service Bus. This combination also brought in the required technology adapters and support for event handling—through the SOA Suite Event Delivery Network, as well as support for JMS and AQ (Advanced Queuing).

The support in SOA Suite for Decision (Business) Rules and Human Workflow as well as the strong integration with Oracle BPM Suite at both design time and run time weighed strongly in favor of the latter when the product selection was made for process orchestration and human workflow. The SOA Suite license includes most of the required functionality with rich (enough) functionality and a track record of many years. The BPM Suite adds support for true BPMN process modeling and execution and comes with a range of run time tools that help design, track and monitor, manage and collaborate on process instances. Saibot Airport ended up selecting BPM Suite because of its best of breed quality with the huge added bonus of perfect integration with SOA Suite.

NOTE
*An on-site visit at St. Matthew's Hospital where SOA Suite 11*g *had been in use since early 2010 proved extremely helpful. The experiences at the hospital with almost all aspects of implementing the layered architecture using Oracle Fusion Middleware and in particular SOA Suite 11*g *were very valuable to the Saibot Airport staff. They were after all about to embark on a very similar journey that the hospital already had been on for the previous five years.*

The product selection for content management did not go very smooth at all. This area is quite new at the airport and there is not a lot of grasp of the subject matter. An external consultant was hired—and quickly let go off again when he turned out to be quite biased (without making that clear up front). Then the definitions of what constitutes content—and what does not—in this selection process were contested. Some people had a vision of the static content of websites whereas others were thinking about all documents—or even all unstructured data—passing through the organization. Even the naming of the Oracle product—WebCenter Content—made some eyebrows go up; it sounded like that static website content thing that they had been able to get off the table after much debate. Only the reassurance that WebCenter Content was in fact the Universal Content Manager restored peace and quiet. In the end, it was decided to give WebCenter Content a go—not because it was such a clear winner but because of the integration in the Oracle Fusion Middleware Platform and the perceived lower risk and smaller effort resulting from that integration.

This next illustration, Figure 1-7, shows the products that were selected. The choice of the portal technology is yet to be made; for the time being, existing .Net and Sharepoint teams

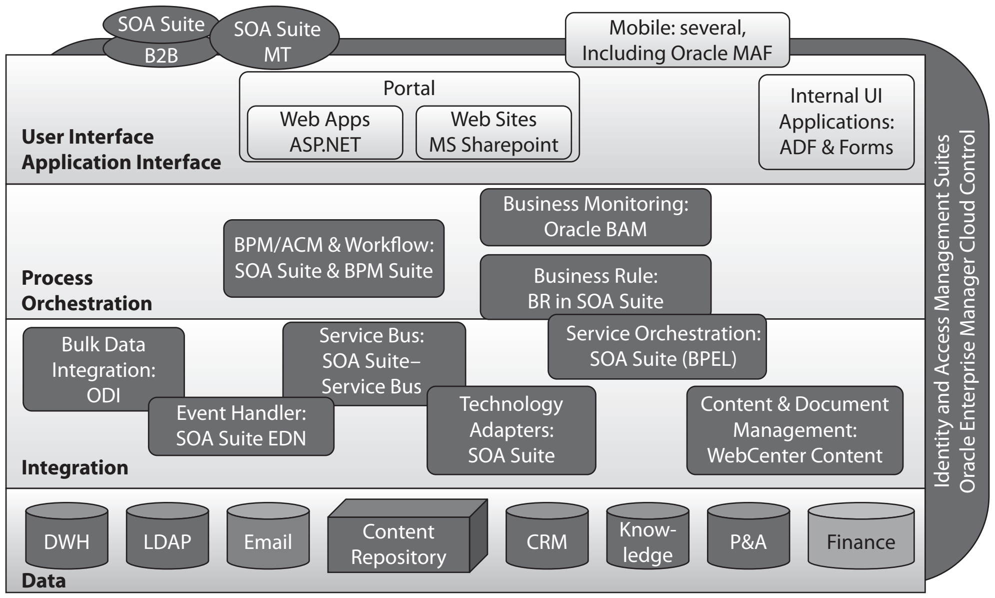

SOA Suite B2B
SOA Suite MT
Mobile: several, Including Oracle MAF
Portal
Web Apps ASP.NET
Web Sites MS Sharepoint
Internal UI Applications: ADF & Forms
User Interface Application Interface
Business Monitoring: Oracle BAM
BPM/ACM & Workflow: SOA Suite & BPM Suite
Business Rule: BR in SOA Suite
Process Orchestration
Bulk Data Integration: ODI
Service Bus: SOA Suite– Service Bus
Service Orchestration: SOA Suite (BPEL)
Event Handler: SOA Suite EDN
Technology Adapters: SOA Suite
Content & Document Management: WebCenter Content
Integration
DWH
LDAP
Email
Content Repository
CRM
Know-ledge
P&A
Finance
Data
Identity and Access Management Suites
Oracle Enterprise Manager Cloud Control

FIGURE 1-7. *Mapping the required components to actual products*

continue development work with their proven technology—connecting to the services offered from the business layer.

The decision on Identity Management and Security is proving difficult. Part of the complexity is the licensing conditions regarding all potential users of Saibot Airport's systems: anyone traveling through the airport is after all a potential user. The airport is negotiating with various vendors of IdM products on how to handle that particular situation in license terms. No final selection has been made. Fortunately, the Oracle Platform Security Services in the WebLogic Platform insulate applications running on the platform—such as ADF applications, SOA Suite Composite applications and Service Bus services—from the actual IdM technology. Any development work being done right now will not be impacted by a later decision on the specific products for identity management and authentication.

After the selections and the associated contracts have been approved by the board of directors, things moved forward and once the infrastructure was prepared, the software was actually installed and staff members educated. The business objectives that had lived on paper for several years now and the architecture design that decorated many a whiteboard were on the verge of getting real.

Processes and Organization

Saibot Airport has traveled part of the road mapped in their business plan. The first few of many projects for transitions of existing systems or the introduction of new business functionality through IT have been executed or are in progress. New technology has been introduced as well as a fundamentally changed way of approaching software development. While the road is bumpy, successes are far more abundant than failures. Reluctance was pretty widespread when the first steps with service orientation and layered architecture were taken and now the mood has changed. Many IT staff members are getting convinced of the approach in general and their ability to play a part in it. Enthusiasm is growing with self-confidence.

The way the airport is now perceived by its own staff, its many business partners, the travelers, liaisons in government bodies, etc. has improved substantially over the past few years. The modernization of applications appeals to all involved and an increase of the quality of service and information has been reported across the board. Shorter waiting times, fewer mistakes, and a better experience are among the results found in a recent survey. An internal evaluation under employees yielded similar outcomes with a higher satisfaction and a much higher score at the standard survey question "do you intend to work at Saibot Airport one year from now."

Among the very first steps in the program has been the consolidation of data into a single database and the centralization of all infrastructure in a single data center (which has two physical sites). All administration activities have been centralized as well. This has turned out to be much more efficient and also allows administrators to specialize. Instead of having to manage small pieces of everything—from network and storage all the way up to operating system and database—they now have responsibility for a much larger piece of one special area. External help is needed much less frequently because of this specialization. That in turn means costs are down and mean times to resolution are shorter.

The data consolidation is proceeding in several steps. First the data was stored in a single database—with the data still scattered across application schemas. Gradually, data becomes truly shared, reducing the duplication and resulting inconsistencies. This is not just a technical process—it also requires putting some pressure on the business representatives who are somewhat wary of sharing their data.

The transition from Oracle Classic [Forms, Data (CRUD-style) and Database oriented, SQL and PL/SQL, Discoverer, Designer] and waterfall approach to the new world of agile and service oriented along with Java and web user interfaces, Mobile, and BPM is scary and overwhelming. Guidance, reassurance, explanations, and almost spiritual support are absolutely necessary to motivate and enable staff in almost every role. For a long period, constant attention is required to engage people who are suspicious and frightened about what is happening. Communication is essential for the success of the kind of rapid evolution the airport is going through.

All roles, from business owner to administrator, need to be involved with the initial transition and with every new initiative involving software development. Support and involvement from C-level is crucial.

Embrace change is the adage of the agile approach to software development that Saibot Airport has adopted. That has been and still is a challenge for many of the parties involved. Even though it is acknowledged rationally that *change* is part of the game, changes in direction and planning are still frequently experienced as very disruptive. Involving the members of the Scrum-team early on whenever such changes occur is important in order to preserve their engagement and motivation.

The Scrum methodology has rapidly taken hold. The initial skepticism was quickly overcome for most and some of the early skeptics are now true Scrum believers—sometimes even bordering on the fanatical side. The commitment of team-members to the team and of the teams to the business objectives has increased dramatically. The constant focus on near rather than distant deadlines and rapid feedback from the business representative turns out to be very motivating for the teams and the quick feedback from the teams to the business also helps the business to adjust requirements. IT staff are usually pretty smart if somewhat overly focused people and their input is used to improve the plans and designs.

The responsibility the business needs to take on in order to gain agility and continuous delivery is substantial. In order for the Scrum approach to work well, the product owner needs to be available very frequently, needs to be thoroughly engaged and be able and authorized to make decisions on priorities and functional design choices. This role can make or break the software development endeavor.

The new way of working—both in terms of architecture and technology and also in terms of software development methodology and process—impacts all roles involved. Some roles, however, were far more changed in the transition. There is the role of the business owner—who has become far more actively involved throughout the project. Other roles undergoing substantial change are tester and administrator. Both are far earlier engaged in development initiatives and closer collaborating with or even embedded within the Scrum teams. New technologies and a broader scope of responsibilities make these roles more interesting as well as challenging.

One aspect of the Scrum way of working that Saibot Airport is still somewhat struggling with is the tension that frequently arises between the short-term focus of the sprints and the longer term architecture objectives. Technical debt is a term coined to describe the corners cut during sprints when functionality wins over coding standards and architectural guidelines. Technical debt should be settled before too long—and all parties agree on that principle—however, it proves a constant struggle to actually claim part of the team's capacity away from functional business value to longer term software quality and maintainability.

The database and SQL and PL/SQL continue to be incredibly important to the overall success of the application architecture. Performance, scalability, integrity can benefit from good or suffer from poor usage of the database capabilities.

It has proven extremely helpful to bring in real-world experience with as many aspects as possible of the transition and the target architecture and way of working; this helped prevent costly (and time-consuming) mistakes that would also cause frustration and undermine confidence in what is already a fragile environment. Experience helps to select and apply proven best practices. Experienced outsiders also allowed knowledge transfer in a hands-on situation—instead of just a classroom—and gives the trainer the opportunity to prove his salt.

Outsiders were brought in to initially show how things should be done, then later on to collaborate with the internal staff on equal footing and finally to act as part time coaches and quality assurers when the internal staff took over for real. The long-term objective is to build up the capabilities of the internal staff and in the short-term experiences outsiders can help boost productivity.

Service-Oriented Architecture

Even though the service orientation yields substantial returns in the long run—through reuse and agile development on top of existing services—there will be initial investments to put the architecture and the infrastructure in place. Some of the cost precedes the gain. Embracing the layered architecture and doing this SOA-thing has to be strategic decision and requires some steadfastness. Ideally of course, there is some low hanging fruit or an urgent business requirement that can lead to a first project that can be the carrier for the launch of the SOA implementation and have a quick and visible business value.

Encapsulation is a powerful notion. The implementation of a service is not relevant to its consumers. The contracts, both the functional interface and the nonfunctional usage aspects, are what consumers rely on. This means among other things that legacy applications can very well be embedded in the new architecture. As long as they can be wrapped inside standards-compliant interfaces that are mediated from to adapters on top of those legacy applications, they fit in quite well in the new world. Encapsulation makes it possible to then replace the legacy implementation of the service, along with its adapter, with a custom implementation using the technology of choice or perhaps with a COTS product.

The decoupled architecture approach also allows for specialized teams—working on isolated, clearly identified areas within the system landscape; that also means that not every developer needs to acquire skills for all tools and technologies involved (at once).

The layered architecture and the decoupled software design make it possible to outsource chunks of works to external partners. For example, once all service interfaces have been agreed upon, a nearby or even a far-away company can build a website or mobile application on top of these interfaces. Rapid development and flexible software engineering capacity—especially when a cloud-based development environment is used—are at the fingertips of the IT manager.

Governance is a critical element, especially in the mid to longer range. Governance is the approach for managing services and related assets (such as the canonical model) through their life cycle. It also includes identifying new services and changes to existing ones, cataloguing services in a way that makes them findable and ultimately reusable, and performing QA on design and implementation to ensure consistency and adherence to the standards specified as part of the governance initiative.

Tools

The short release cycles and frequent, near continuous, delivery can only be realized with automated build and deployment procedures that ideally include testing as well. These procedures for building and rolling out software deliverables should be coordinated and monitored.

Something similar applies to the environments in which the artifacts are deployed—the combination of platform and frameworks that make up the infrastructure in which the developed software runs. The environment consists both of standard software from third party vendors and the specific configuration this software as it applies to Saibot Airport. Controlling the creation, roll out of and updates to these environments is another process that requires automated execution and control in order to achieve an agile and managed delivery process.

The airport has selected a wide assortment of tools, most of them from open source projects, to aid with software engineering. Among these are tools that help with a variety of tasks, most in the area of DevOps. Bringing together the worlds of preparation (or development) and operations (aka ops) is tagged with the popular term DevOps. It is closely associated with an agile way of working that strives for quick time to market for updates in software, platform, and infrastructure. This can only be achieved by smooth communication, close collaboration, and a high degree of automation with regard to build, test, and deploy.

For orchestrating and monitoring the overall build process, Hudson was selected—and stuck to when the Jenkins fork occurred. The build actions themselves are largely done using Maven. In conjunction with Maven, the tool Artifactory is used as a repository for code artifacts. For controlled deployment, a tool called DeployIt has been acquired (this product is not open source). Rolling out environments has been tackled with Puppet.

For various types of testing activities, the following tools were selected:

- Web services: SoapUI (functional), JMeter, and LoadUI (load and stress)
- Java and ADF Business Components: jUnit (function and load)
- Web applications: JMeter (functional and primarily load); Selenium (functional); Oracle Application Testing Suite (under evaluation for both functional and load testing)

In addition, some of the built in testing features of the SOA Suite are used as well. Some attempts have been made to create unit tests for web components and composite services through the use of mock objects. EasyMock (for Java) and SoapUI with Jetty (for mock web services) were used for this.

Management of tickets—issues, requirements, enhancement requests—is done using Jira. This tool is also used for coordinating the Backlog (Product and Team Sprint) and the Scrum board.

Source code control is done using Subversion—although some investigation has been done into Git. No final decision has been made yet as to whether a migration to Git should be initiated.

With regard to the quality control of the code, a number of tools are applied—by and large only for Java code and not yet taking Service Bus, SOA Suite and BPM artifacts into account. Some of the frameworks used in this area are Sonar (for integrating the findings from various other QA tools), Checkstyle (for verifying adherence to coding conventions), PMD (spotting bad practices), and FindBugs (for finding potential bugs).

MediaWiki has been set up to allow teams and guilds (see below) to collaborate and share. The Wiki is used for checklists, how-to documents, configuration instructions, the catalog of services, records of design considerations, and motivated decisions. Also on the Wiki are architecture guidelines, coding conventions, descriptions of DTAP environments along with all URLs for the consoles and composers and associated tools for Service Bus, SOA Suite, WebCenter Content, and WebLogic Server.

Organization and Roles

The distinction between projects doing initial development and maintenance teams doing corrective and adaptive management is removed. With very short release cycles, the need for dedicated teams with special focus on maintenance working against a higher frequency release cycle than the projects teams largely disappeared. It seemed a waste to spread expertise in technology and application thin across development and maintenance. And it is considered a good idea that developers and teams take responsibility for evolving, which includes correcting, their own deliverables.

At the same time, the notion of project teams was dropped. Development teams would be created that focus on certain areas of functionality through development skills on a subset of all technology in use at the airport. These teams could be enlisted for user stories (features) defined by several business projects. They also were mandated to spend between 20 and 30 percent of their time on corrective and adaptive maintenance. Up to 10 percent of the time was free to use on "small effort, big gain" activities: business requests that are not really part of a formal user story or project back log but that can add substantial business value with very little risk and effort. Given the short release cycles, this flexibility easily provides happy faces among end users and developers.

The testers used to have their stage in the waterfall approach where—after plenty of preparations, based on detailed design documents—they would torture the software artifacts in order to verify its fitness. Testing was typically done in isolation—the only contact with the development team consisting of the issue tracking system. Testing focused almost exclusively on the user interface—by and large ignoring the internal components.

"Testing 2.0" in a Scrum way of working is very different. Testers are embedded in the Scrum teams. Testing is done as part of every sprint. Design documents are high level if at all available. Testing needs to be geared toward user expectations and interface designs. Testing involves a large chunk of nonuser interface services—such as web services and including database APIs. The use of tooling for automated (regression) testing is rapidly increasing. Communication and collaboration between testers and other team members such as analysts and developers grows from next to nothing to pretty intense. It is not uncommon for testers to engage in other activities in the early stages of a sprint—and for analysts and developers to pick up testing tasks near the end of a sprint. Feedback on software quality is much faster than in the old way of working. At the same time, testing may not be as rigorous as it used to be, although the gradual buildup of automatic test sets eventually brings testing to a fairly high level.

Similarly drastic are the changes for the administrators. For many years, administrators either did hardware and systems administration or database administration. By and large, activities were performed decoupled from development projects and only geared toward production systems. In a few years, a new world descended on the administrators. The introduction of virtualization made a huge difference to the administrators. New skills were required and new options became available, for quick deployment of new environments for example or easy rollback of an environment to a previous snapshot. Planning machines no longer meant the same thing as planning the hardware.

The next major wave of newness consisted of the introduction of middleware. The application server is the core element and then several engines run on the application server—each performing different functions that each required administrative effort in terms of installation, configuration, and monitoring. Expertise on WebLogic Server as well as administration for the Service Bus, the SOA Suite, WebCenter Content, and BPM Suite had to be built up.

Platform administration and infrastructure administration were identified as two individual and complementary disciplines and some steps are being taken to organize and offer the platform and infrastructure as private cloud services. Additionally, investigations have started into using external cloud facilities for testing purposes such as load and stress testing.

DevOps

Gradually the world at large as well as the parties involved at Saibot Airport realized that the gap between the project teams focusing on development and the administrators worried primarily about the running production systems was both artificial and unproductive. Upon closer inspection, it was realized that the stages visited for software development—design, build, test—also apply to platform development and infrastructure construction. For these two layers too, based on requirements, a design is created that is then implemented through installation and configuration and subsequently tested—as shown in Figure 1-8. Functional requirements are the main driver for the application development and nonfunctional requirements are the primary force in platform and infrastructure development. The three layers are closely related—and the way the work on them is performed and organized should reflect this close relation.

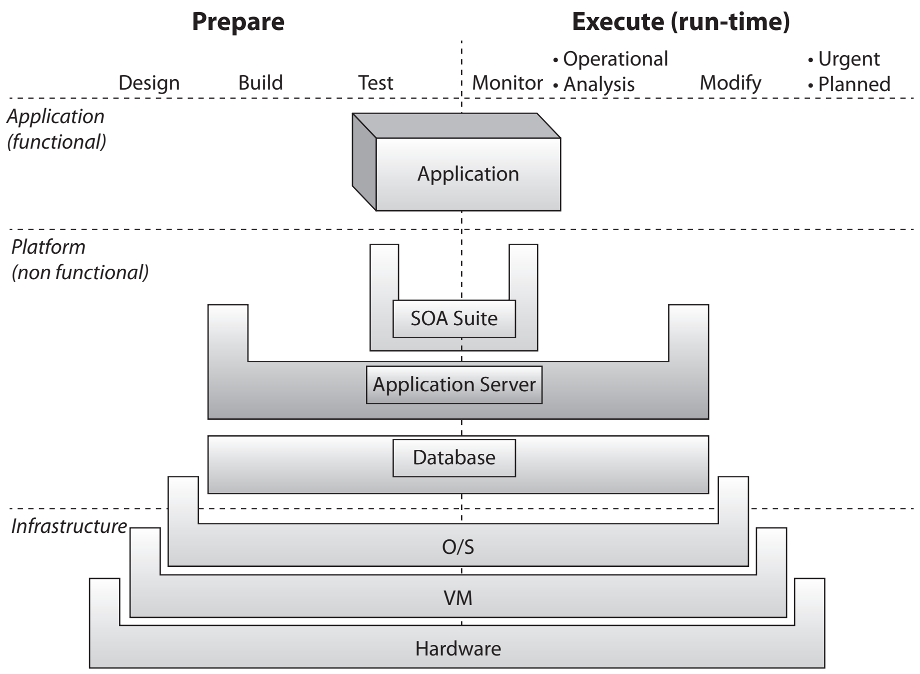


FIGURE 1-8. *The main stages in preparation and execution across the three main IT layers*

On the right side in Figure 1-8 is the run time phase—when custom software and COTS is deployed on the run time platform running on the production infrastructure. In this execution phase all preparations should come to fruition and actual value is delivered to the business. The entire stack is required to generate the business value—there can after all be no meaningful business value with any piece of the stack lacking or underperforming. Across the stack, the performance must be monitored to ensure the SLAs are met. Run time metrics are used for real time—and even predictive—assessment of the operations. These metrics are also analyzed for longer term trends.

Based on these operational findings as well as the longer term analysis and also fed by the ongoing activities in the preparation phase, modifications at various layers in the stack will be required. Some are urgent—instant responses to infrastructure or platform failures and urgent bug fixes in custom software—and some can be planned with more lead time.

As a result of the analysis overhead, some platform and infrastructure specialists were associated with and even embedded in software development teams and their activities were organized in a similar way. Other teams were set up to take on the right side of Figure 1-8—with members covering all layers of the stack. And both processes and tooling were instated to ensure easy communication and handovers between the worlds of preparation and execution. The agile approach with the frequent release of changes to applications, platform, and infrastructure made it imperative to close this traditional gap between the worlds of design time and run time and forced a much more integrated way of looking at the IT department at Saibot Airport.

Looking Forward

In the near future, Saibot Airport will continue executing the business program with the associated IT projects. A major new focus in addition to the introduction of SOA is going to be the introduction of BPM. Focus on business processes is strong—to ensure proper execution of procedures and provide management information about current affairs as well as the ability to optimize and partially automate the processes. It is hoped that new staff will require less training when they are guided to the processes. And some of the actions will be turned into self service activities to be executed by external parties.

Attracting new IT staff and building up internal expertise to reduce the dependency on external consultants is very important to Saibot Airport. It is actively approaching local specialist, trying to hire them away from their current employers. For its current staff, the airport is organizing development programs that will help them build up skills around the newly introduced technologies and methodologies. Staff can attend international conferences and workshops in order to meet with peers and learn the latest information. Renowned speakers are invited to conduct sessions at Saibot Airport itself—for internal staff as well as invitees that the airport hopes to hire. The opportunity to play a key role in this major technology evolution in combination with the facilities for personal development has already attracted a number of new Oracle Fusion Middleware specialists to the ranks of the airport's IT department.

In the slightly longer term, Saibot Airport hopes to do more with the data it is sitting on. It seems that this data may be interesting for external parties. And the airport also hopes to learn more itself about peak volumes and trends in order to streamline its processes and prepare its staff.

The development of mobile apps has only just started. Opening up multiple channels in earnest is among the near future plans. Saibot Airport expects to leverage the layered architecture and the reusable services to be able to quickly roll out new ways to interact.

Summary

The challenges for Saibot Airport are substantial—and very similar to those for many organizations around the world. The mission and business objectives may vary—the steps to go through in terms of IT close to identical across enterprises, industries, and countries. Agility and flexibility, digital, paperless interactions and processes, self-service, and 24/7 availability through multiple channels and near real-time operational insight are goals set by most organizations. At that broad level, the means to these ends are quite similar as well, in terms of IT architecture and the required technology components. Of course, many different vendors provide various products to address the challenges—and different combinations of these products could be used to do the job. Saibot Airport picked Oracle Fusion Middleware as its technology of choice—which makes it the perfect case study for this book. Starting in Chapter 3, we will see *how* the airport uses the Oracle SOA Suite in its quest for success. The next chapter first introduces the Oracle technology portfolio and zooms in on Fusion Middleware. Then it describes the SOA Suite in particular—as to set the scene for the remainder of the book.

CHAPTER 3

Planning and Managing Tablespaces

How a DBA configures the layout of the tablespaces in a database directly affects the performance and manageability of the database. In this chapter, we'll review the different types of tablespaces as well as how temporary tablespace usage can drive the size and number of tablespaces in a database leveraging the temporary tablespace group feature introduced in Oracle 10*g*.

I'll also show how Oracle's Optimal Flexible Architecture (OFA), supported since Oracle 7, helps to standardize the directory structure for both Oracle executables and the database files themselves; Oracle Database 12*c* further enhances OFA to complement its original role of improving performance to enhancing security and simplifying cloning and upgrade tasks.

A default installation of Oracle provides the DBA with a good starting point, not only creating an OFA-compliant directory structure but also segregating segments into a number of tablespaces based on their function. We'll review the space requirements for each of these tablespaces and provide some tips on how to fine-tune the characteristics of these tablespaces.

Using Oracle Automatic Storage Management (ASM) as your logical volume manager makes tablespace maintenance easier and more efficient by automatically spreading out the segments within a tablespace across all disks of an ASM disk group. Adding datafiles to a tablespace is almost trivial when using ASM; using bigfile tablespaces means you only have to allocate a single datafile for the tablespace. In both cases, you don't need to specify, or even need to know, the name of the datafile itself within the ASM directory structure.

In Oracle Database 12*c*, container databases (CDBs) and pluggable databases (PDBs) in a multitenant database architecture change how some tablespaces are used and managed in a pluggable database. All permanent tablespaces can be associated with one and only one database—either the CDB or one PDB. In contrast, temporary tablespaces or temporary tablespace groups are managed at the CDB level and are used by all PDBs within the CDB. See Chapter 10 for an in-depth discussion of the Oracle Database 12*c* multitenant architecture.

At the end of the chapter, I'll provide some guidelines to help you place segments into different tablespaces based on their type, size, and frequency of access, as well as ways to identify hotspots in one or more tablespaces.

Tablespace Architecture

A prerequisite to competently setting up the tablespaces in your database is understanding the different types of tablespaces and how they are used in an Oracle database. In this section, we'll review the different types of tablespaces and give some examples of how they are managed. In addition, I'll review the types of tablespaces by category—permanent tablespaces (SYSTEM, SYSAUX, and so on), temporary tablespaces, undo tablespaces, and bigfile tablespaces—and describe their function. Finally, I'll also discuss Oracle's Optimal Flexible Architecture (OFA) and how it can ease maintenance tasks.

Tablespace Types

The primary types of tablespaces in an Oracle database are permanent, undo, and temporary. *Permanent* tablespaces contain segments that persist beyond the duration of a session or a transaction.

Although the undo tablespace may have segments that are retained beyond the end of a session or a transaction, it provides read consistency for SELECT statements that access tables being modified as well as provides undo data for a number of the Oracle Flashback features of the database. Primarily, however, undo segments store the previous values of columns being updated or deleted. This ensures that if a user's session fails before the user issues a COMMIT or a ROLLBACK, the UPDATEs, INSERTs, and DELETEs will be removed and will never be accessible by other sessions. Undo segments are never directly accessible by a user session, and undo tablespaces may only have undo segments.

As the name implies, temporary tablespaces contain transient data that exists only for the duration of the session, such as space to complete a sort operation that will not fit in memory.

Bigfile tablespaces can be used for any of these three types of tablespaces, and they simplify tablespace management by moving the maintenance point from the datafile to the tablespace. Bigfile tablespaces consist of one and only one datafile. There are a couple of downsides to bigfile tablespaces, however, and they will be presented later in this chapter.

Permanent

The SYSTEM and SYSAUX tablespaces are two examples of permanent tablespaces. In addition, any segments that need to be retained by a user or an application beyond the boundaries of a session or transaction should be stored in a permanent tablespace.

SYSTEM Tablespace User segments should never reside in the SYSTEM or SYSAUX tablespace, period. If you do not specify a default permanent or temporary tablespace when creating users, the database-level default permanent and temporary tablespaces are used.

If you use the Oracle Universal Installer (OUI) to create a database for you, a separate tablespace other than SYSTEM is created for both permanent and temporary segments. If you create a database manually, be sure to specify both a default permanent tablespace and a default temporary tablespace, as in the sample CREATE DATABASE command that follows.

```
CREATE DATABASE rjbdb
   USER SYS IDENTIFIED BY melsm25
   USER SYSTEM IDENTIFIED BY welisa45
   LOGFILE GROUP 1 ('/u02/oracle11g/oradata/rjbdb/redo01.log') SIZE 100M,
           GROUP 2 ('/u04/oracle11g/oradata/rjbdb/redo02.log') SIZE 100M,
           GROUP 3 ('/u06/oracle11g/oradata/rjbdb/redo03.log') SIZE 100M
   MAXLOGFILES 5
   MAXLOGMEMBERS 5
   MAXLOGHISTORY 1
   MAXDATAFILES 100
   MAXINSTANCES 1
   CHARACTER SET US7ASCII
   NATIONAL CHARACTER SET AL16UTF16
   DATAFILE '/u01/oracle11g/oradata/rjbdb/system01.dbf' SIZE 2G REUSE
   EXTENT MANAGEMENT LOCAL
   SYSAUX DATAFILE '/u01/oracle11g/oradata/rjbdb/sysaux01.dbf'
      SIZE 800M REUSE
   DEFAULT TABLESPACE USERS
      DATAFILE '/u03/oracle11g/oradata/rjbdb/users01.dbf'
      SIZE 4G REUSE
   DEFAULT TEMPORARY TABLESPACE TEMPTS1
      TEMPFILE '/u01/oracle11g/oradata/rjbdb/temp01.dbf'
      SIZE 500M REUSE
   UNDO TABLESPACE undotbs
      DATAFILE '/u02/oracle11g/oradata/rjbdb/undotbs01.dbf'
      SIZE 400M REUSE AUTOEXTEND ON MAXSIZE 2G;
```

As of Oracle 10*g*, the SYSTEM tablespace is locally managed by default; in other words, all space usage is managed by a bitmap segment in the first part of the first datafile for the tablespace. In a database where the SYSTEM tablespace is locally managed, the other tablespaces in the database must also be locally managed or they must be read-only. Using locally managed tablespaces takes some of the contention off the SYSTEM tablespace because space allocation and deallocation operations for a tablespace do not need to use data dictionary tables. More details on locally managed tablespaces can be found in Chapter 6. Other than to support the import of a transportable tablespace that is dictionary managed from a legacy database, there are no advantages to having a dictionary-managed tablespace in your database.

SYSAUX Tablespace Like the SYSTEM tablespace, the SYSAUX tablespace should not have any user segments. The contents of the SYSAUX tablespace, broken down by application, can be reviewed using Oracle Enterprise Manager Database Express (EM Express) or Cloud Control 12*c*. You can edit the SYSAUX tablespace in Cloud Control 12*c* by choosing Administration | Storage | Tablespaces and clicking the SYSAUX link in the tablespace list. Figure 3-1 shows a graphical representation of the space usage within SYSAUX.

If the space usage for a particular application that resides in the SYSAUX tablespace becomes too high or creates an I/O bottleneck through high contention with other applications that use the SYSAUX tablespace, you can move one or more of these applications to a different tablespace. Any SYSAUX occupant listed in Figure 3-1 that has a Change Tablespace link available can be moved by clicking the link and then choosing a destination tablespace in the field shown in Figure 3-2. The XDB objects will be moved to the SYSAUX2 tablespace. An example of moving a SYSAUX occupant to a different tablespace using the command line interface can be found in Chapter 6.

The SYSAUX tablespace can be monitored just like any other tablespace; later in this chapter, I'll show how EM Cloud Control can help us to identify hotspots in a tablespace.

Undo

Multiple undo tablespaces can exist in a database, but only one undo tablespace can be active at any given time for a single database instance. Undo tablespaces are used for rolling back transactions, for providing read consistency for SELECT statements that run concurrently with DML statements on the same table or set of tables, and for supporting a number of Oracle Flashback features, such as Flashback Query.

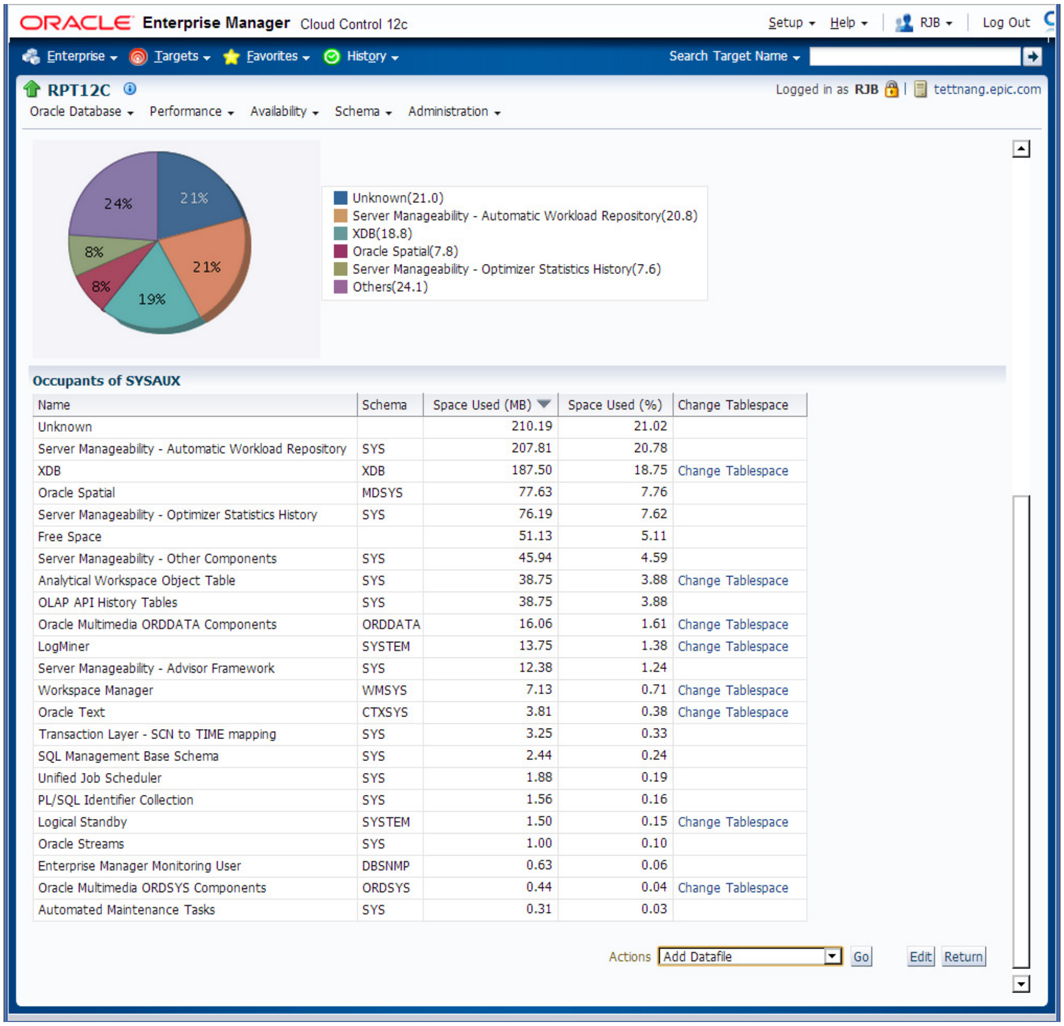
The undo tablespace needs to be sized correctly to prevent ORA-01555 "Snapshot too old" errors and to provide enough space to support initialization parameters such as UNDO_RETENTION. More information on how to monitor, size, and create undo tablespaces can be found in Chapter 7.

Temporary

More than one temporary tablespace can be online and active in the database, but until Oracle 10*g*, multiple sessions by the same user would use the same temporary tablespace because only one default temporary tablespace could be assigned to a user. To solve this potential performance bottleneck, Oracle supports *temporary tablespace groups*. A temporary tablespace group is a synonym for a list of temporary tablespaces.

A temporary tablespace group must consist of at least one temporary tablespace; it cannot be empty. Once a temporary tablespace group has no members, it no longer exists.

One of the big advantages of using temporary tablespace groups is to provide a single user with multiple sessions with the ability to use a different actual temporary tablespace for each session. In the diagram shown in Figure 3-3, the user OE has two active sessions that need temporary space for performing sort operations.



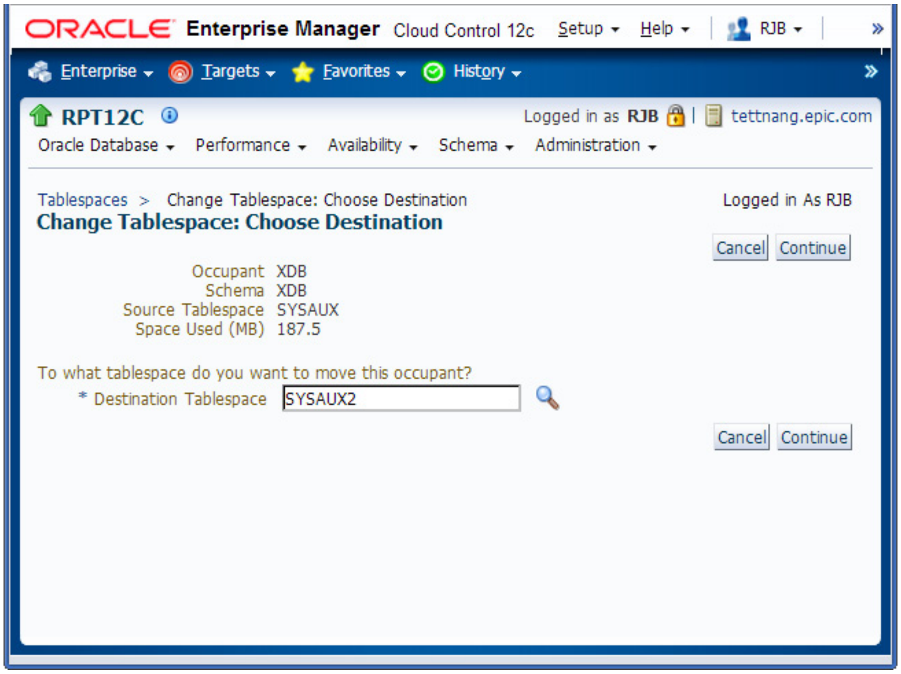
🟦 Unknown(21.0)
🟧 Server Manageability - Automatic Workload Repository(20.8)
🟩 XDB(18.8)
🟥 Oracle Spatial(7.8)
🟫 Server Manageability - Optimizer Statistics History(7.6)
🟪 Others(24.1)

Occupants of SYSAUX

Name	Schema	Space Used (MB)	Space Used (%)	Change Tablespace
Unknown		210.19	21.02	
Server Manageability - Automatic Workload Repository	SYS	207.81	20.78	
XDB	XDB	187.50	18.75	Change Tablespace
Oracle Spatial	MDSYS	77.63	7.76	
Server Manageability - Optimizer Statistics History	SYS	76.19	7.62	
Free Space		51.13	5.11	
Server Manageability - Other Components	SYS	45.94	4.59	
Analytical Workspace Object Table	SYS	38.75	3.88	Change Tablespace
OLAP API History Tables	SYS	38.75	3.88	
Oracle Multimedia ORDDATA Components	ORDDATA	16.06	1.61	Change Tablespace
LogMiner	SYSTEM	13.75	1.38	Change Tablespace
Server Manageability - Advisor Framework	SYS	12.38	1.24	
Workspace Manager	WMSYS	7.13	0.71	Change Tablespace
Oracle Text	CTXSYS	3.81	0.38	Change Tablespace
Transaction Layer - SCN to TIME mapping	SYS	3.25	0.33	
SQL Management Base Schema	SYS	2.44	0.24	
Unified Job Scheduler	SYS	1.88	0.19	
PL/SQL Identifier Collection	SYS	1.56	0.16	
Logical Standby	SYSTEM	1.50	0.15	Change Tablespace
Oracle Streams	SYS	1.00	0.10	
Enterprise Manager Monitoring User	DBSNMP	0.63	0.06	
Oracle Multimedia ORDSYS Components	ORDSYS	0.44	0.04	Change Tablespace
Automated Maintenance Tasks	SYS	0.31	0.03	

FIGURE 3-1. *EM Cloud Control 12c SYSAUX tablespace contents*

Instead of a single temporary tablespace being assigned to a user, the temporary tablespace group is assigned; in this example, the temporary tablespace group TEMPGRP has been assigned to OE. However, because there are three actual temporary tablespaces within the TEMPGRP temporary tablespace group, the first OE session may use temporary tablespace TEMP1, and the SELECT statement executed by the second OE session may use the other two temporary tablespaces, TEMP2 and TEMP3, in parallel. Before Oracle 10*g*, both sessions would use the same temporary tablespace, potentially causing a performance issue.



Change Tablespace: Choose Destination

Occupant XDB
Schema XDB
Source Tablespace SYSAUX
Space Used (MB) 187.5

To what tablespace do you want to move this occupant?
* Destination Tablespace SYSAUX2

FIGURE 3-2. *Using EM Cloud Control 12c to move a SYSAUX occupant*

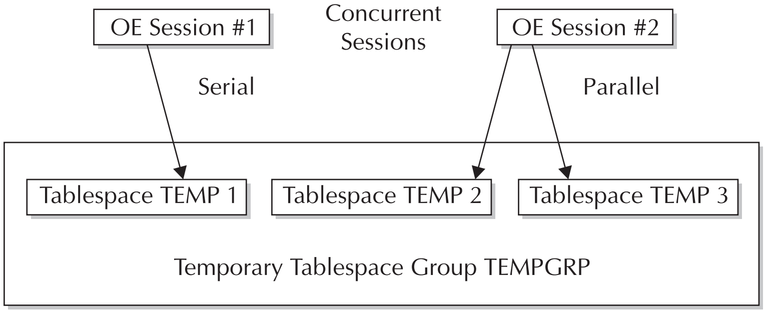


FIGURE 3-3. *Temporary tablespace group TEMPGRP*

Creating a temporary tablespace group is very straightforward. After creating the individual tablespaces TEMP1, TEMP2, and TEMP3, we can create a temporary tablespace group named TEMPGRP as follows:

```
SQL> alter tablespace temp1 tablespace group tempgrp;
Tablespace altered.
SQL> alter tablespace temp2 tablespace group tempgrp;
Tablespace altered.
SQL> alter tablespace temp3 tablespace group tempgrp;
Tablespace altered.
```

Changing the database's default temporary tablespace to TEMPGRP uses the same command as assigning an actual temporary tablespace as the default; temporary tablespace groups are treated logically the same as a temporary tablespace:

```
SQL> alter database default temporary tablespace tempgrp;
Database altered.
```

To drop a tablespace group, we must first drop all its members. Dropping a member of a tablespace group is accomplished by assigning the temporary tablespace to a group with an empty string (in other words, removing the tablespace from the group):

```
SQL> alter tablespace temp3 tablespace group '';
Tablespace altered.
```

As you might expect, assigning a temporary tablespace group to a user is identical to assigning a temporary tablespace to a user; this assignment can happen either when the user is created or at some point in the future. In the following example, the new user JENWEB is assigned the temporary tablespace TEMPGRP:

```
SQL> create user jenweb identified by pi4001
  2      default tablespace users
  3      temporary tablespace tempgrp;
User created.
```

Note that if we did not assign the tablespace during user creation, the user JENWEB would still be assigned TEMPGRP as the temporary tablespace because it is the database default from our previous CREATE DATABASE example.

A couple of changes were made to the data dictionary views in Oracle Database 10*g* and Oracle Database 11*g* to support temporary tablespace groups. The data dictionary view DBA_USERS still has the column TEMPORARY_TABLESPACE, as in previous versions of Oracle, but this column may now contain either the name of the temporary tablespace assigned to the user or the name of a temporary tablespace group:

```
SQL> select username, default_tablespace, temporary_tablespace
  2       from dba_users where username = 'JENWEB';

USERNAME             DEFAULT_TABLESPACE TEMPORARY_TABLESPACE
-------------------- ------------------ --------------------
JENWEB               USERS              TEMPGRP

1 row selected.
```

The new data dictionary view DBA_TABLESPACE_GROUPS shows the members of each temporary tablespace group:

```
SQL> select group_name, tablespace_name from dba_tablespace_groups;

GROUP_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------
TEMPGRP                        TEMP1
TEMPGRP                        TEMP2
TEMPGRP                        TEMP3

3 rows selected.
```

As with most every other feature of Oracle that can be accomplished with the command line, assigning members to temporary tablespace groups or removing members from temporary tablespace groups can be performed using EM Cloud Control 12*c*. In Figure 3-4, we can add or remove members from a temporary tablespace group.

Bigfile

A bigfile tablespace eases database administration because it consists of only one datafile. The single datafile can be up to 128TB (terabytes) in size if the tablespace block size is 32KB; if you use the more common 8KB block size, 32TB is the maximum size of a bigfile tablespace. Many of the commands previously available only for maintaining datafiles can now be used at the tablespace level if the tablespace is a bigfile tablespace. Chapter 6 reviews how BIGFILE tablespaces are created and maintained.

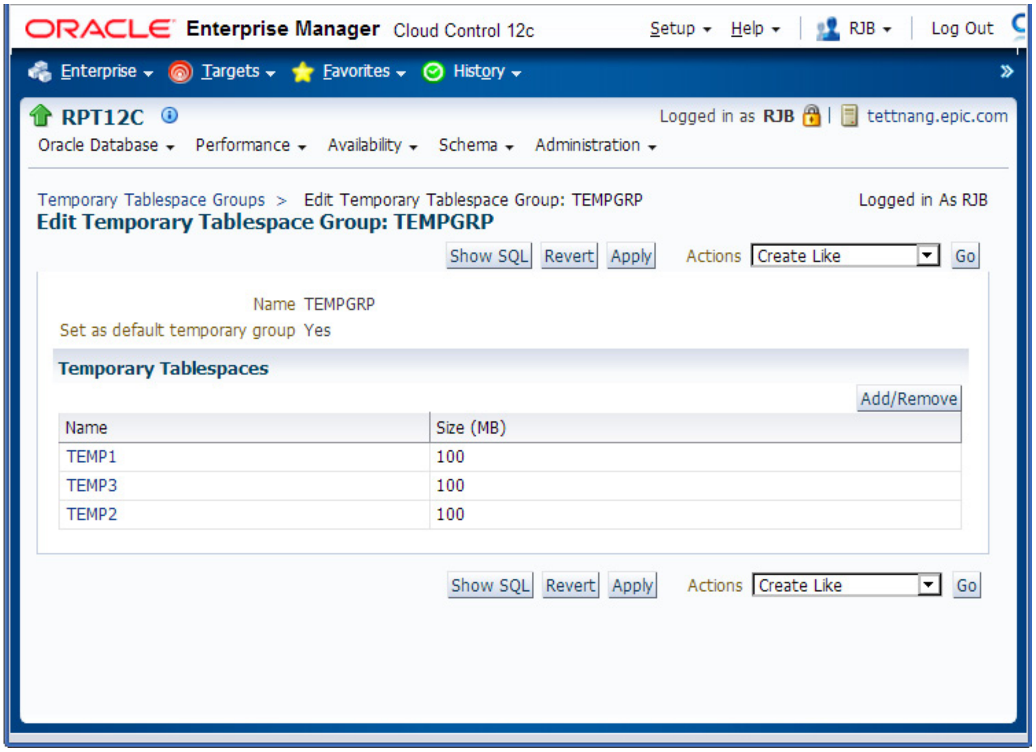
The maintenance convenience of bigfile tablespaces can be offset by some potential disadvantages. Because a bigfile tablespace is a single datafile, a full backup of a single large datafile will take significantly longer than a full backup of several smaller datafiles (with the same total size as the single bigfile tablespace) even when Oracle uses multiple slave processes per datafile. If your bigfile tablespaces are read-only or if only changed blocks are backed up on a regular basis, the backup issue may not be critical in your environment. If you use the SECTION SIZE option in RMAN, available as of Oracle Database 11*g*, then an entire bigfile tablespace (and therefore the entire datafile) can be backed up in parallel.

Optimal Flexible Architecture

Oracle's Optimal Flexible Architecture (OFA) provides guidelines to ease the maintenance of the Oracle software and database files as well as improve the performance of the database by placing the database files such that I/O bottlenecks are minimized.

Although using OFA is not strictly enforced when you're installing or maintaining an Oracle environment, using OFA makes it easy for someone to understand how your database is organized on disk, preventing that phone call in the middle of the night during the week you're on vacation!

OFA is slightly different depending on the type of storage options you use: either an ASM environment or a standard operating system file system that may or may not be using a third-party logical volume manager or RAID-enabled disk subsystem. In either case, the Database Configuration Assistant can create an OFA-compliant datafile directory structure for you.



Temporary Tablespace Groups > Edit Temporary Tablespace Group: TEMPGRP

Logged in As RJB

Edit Temporary Tablespace Group: TEMPGRP

Name TEMPGRP

Set as default temporary group Yes

Temporary Tablespaces

Name	Size (MB)
TEMP1	100
TEMP3	100
TEMP2	100

FIGURE 3-4. *Using EM Cloud Control 12c to edit temporary tablespace groups*

Non-ASM Environment

In a non-ASM environment on a Unix server, at least three file systems on separate physical devices are required to implement OFA recommendations. Starting at the top, the recommended format for a mount point is /<*string const*><*numeric key*>, where <*string const*> can be one or several letters and <*numeric key*> is either two or three digits. For example, on one system we may have mount points **/u01**, **/u02**, **/u03**, and **/u04**, with room to expand to an additional 96 mount points without changing the file-naming convention. Figure 3-5 shows a typical Unix file system layout with an OFA-compliant Oracle directory structure.

There are two instances on this server: an ASM instance to manage disk groups and a standard RDBMS instance (**dw**).

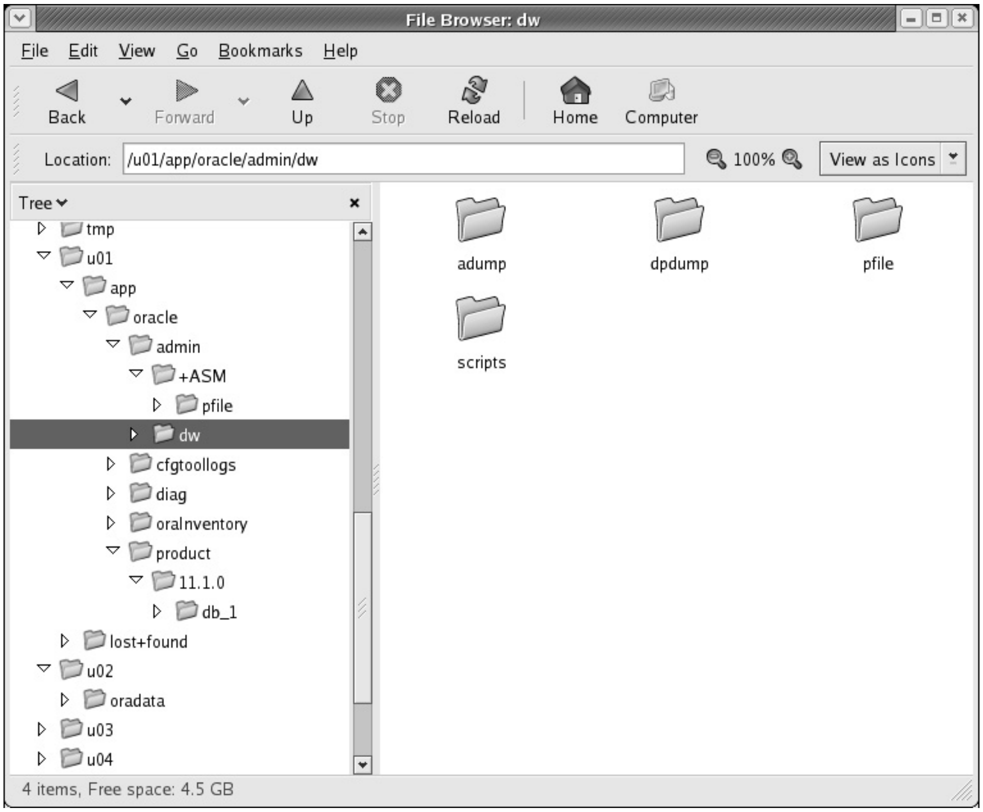


FIGURE 3-5. *OFA-compliant Unix directory structure*

Software Executables The software executables for each distinct product name reside in the directory /*<string const><numeric key>*/*<directory type>*/*<product owner>*, where *<string const>* and *<numeric key>* are defined previously, *<directory type>* implies the type of files installed in this directory, and *<product owner>* is the name of the user that owns and installs the files in this directory. For example, **/u01/app/oracle** would contain application-related files (executables) installed by the user **oracle** on the server. The directory **/u01/app/apache** would contain the executables for the middleware web server installed from a previous version of Oracle.

As of Oracle 10*g*, the OFA standard makes it easy for the DBA to install multiple versions of the database and client software within the same high-level directory. The OFA-compliant Oracle home path, corresponding to the environment variable ORACLE_HOME, contains a suffix that corresponds to the type and incarnation of the installation. For example, one installation of

Oracle 12*c*, one installation of Oracle 11*g*, two different installations of Oracle 10*g*, and one installation of Oracle9*i* may reside in the following three directories:

```
/u01/app/oracle/product/9.2.0.1
/u01/app/oracle/product/10.1.0/db_1
/u01/app/oracle/product/10.1.0/db_2
/u01/app/oracle/product/11.1.0/db_1
/u01/app/oracle/product/12.1.0/dbhome_1
```

At the same time, the Oracle client executables and configuration may be stored in the same parent directory as the database executables:

```
/u01/app/oracle/product/12.1.0/client_1
```

Some installation directories will never have more than one instance for a given product; for example, Oracle Grid Infrastructure (one installation per server) will be installed in the following directory given the previous installations:

```
/u01/app/oracle/product/12.1.0/grid
```

Because Grid Infrastructure can be installed only once on a system, it does not have an incrementing numeric suffix.

Database Files Any non-ASM Oracle datafiles reside in /<*mount point*>/oradata/<*database name*>, where <*mount point*> is one of the mount points we discussed earlier, and <*database name*> is the value of the initialization parameter DB_NAME. For example, **/u02/oradata/rac0** and **/u03/oradata/rac0** would contain the non-ASM control files, redo log files, and datafiles for the instance **rac0**, whereas **/u05/oradata/dev1** would contain the same files for the **dev1** instance on the same server. The naming convention for the different file types under the **oradata** directory are detailed in Table 3-1.

Although Oracle tablespace names can be as long as 30 characters, it is advisable to keep the tablespace names eight characters or less in a Unix environment. Because portable Unix filenames are restricted to 14 characters, and the suffix of an OFA datafile name is <*n*>.dbf, where *n* is two digits, a total of six characters are needed for the suffix in the file system. This leaves eight characters for the tablespace name itself.

File Type	Filename Format	Variables
Control files	control.ctl	None.
Redo log files	redo<*n*>.log	*n* is a two-digit number.
Datafiles	<*tn*>.dbf	*t* is an Oracle tablespace name, and *n* is a two-digit number.

TABLE 3-1. *OFA-Compliant Control File, Redo Log File, and Datafile Naming Conventions*

Only control files, redo log files, and datafiles associated with the database <*database name*> should be stored in the directory /<*mount point*>/oradata/<*database name*>. For the database **ord** managed without ASM, the datafile names are as follows:

```
SQL> select file#, name from v$datafile;

     FILE# NAME
---------- -----------------------------------
         1 /u05/oradata/ord/system01.dbf
         2 /u05/oradata/ord/undotbs01.dbf
         3 /u05/oradata/ord/sysaux01.dbf
         4 /u05/oradata/ord/users01.dbf
         5 /u09/oradata/ord/example01.dbf
         6 /u09/oradata/ord/oe_trans01.dbf
         7 /u05/oradata/ord/users02.dbf
         8 /u06/oradata/ord/logmnr_rep01.dbf
         9 /u09/oradata/ord/big_users.dbf
        10 /u08/oradata/ord/idx01.dbf
        11 /u08/oradata/ord/idx02.dbf
        12 /u08/oradata/ord/idx03.dbf
        13 /u08/oradata/ord/idx04.dbf
        14 /u08/oradata/ord/idx05.dbf
        15 /u08/oradata/ord/idx06.dbf
        16 /u08/oradata/ord/idx07.dbf
        17 /u08/oradata/ord/idx08.dbf
17 rows selected.
```

Other than file numbers 8 and 9, all the datafiles in the **ord** database are OFA compliant and are spread out over four different mount points. The tablespace name in file number 8 is too long, and file number 9 does not have a numeric two-digit counter to represent new datafiles for the same tablespace.

ASM Environment

In an ASM environment, the executables are stored in the directory structure presented previously; however, if you browsed the directory **/u02/oradata** in Figure 3-5, you would see no files. All the control files, redo log files, and datafiles for the instance **dw** are managed by the ASM instance +ASM on this server.

The actual datafile names are not needed for most administrative functions because ASM files are all Oracle Managed Files (OMF). This eases the overall administrative effort required for the database. Within the ASM storage structure, an OFA-like syntax is used to subdivide the file types even further:

```
SQL> select file#, name from v$datafile;

     FILE# NAME
---------- ----------------------------------------
         1 +DATA/dw/datafile/system.256.622426913
         2 +DATA/dw/datafile/sysaux.257.622426915
         3 +DATA/dw/datafile/undotbs1.258.622426919
         4 +DATA/dw/datafile/users.259.622426921
         5 +DATA/dw/datafile/example.265.622427181
5 rows selected.

SQL> select name from v$controlfile;

NAME
----------------------------------------
+DATA/dw/controlfile/current.260.622427059
+RECOV/dw/controlfile/current.256.622427123
2 rows selected.

SQL> select member from v$logfile;

MEMBER
----------------------------------------
+DATA/dw/onlinelog/group_3.263.622427143
+RECOV/dw/onlinelog/group_3.259.622427145
+DATA/dw/onlinelog/group_2.262.622427135
+RECOV/dw/onlinelog/group_2.258.622427137
+DATA/dw/onlinelog/group_1.261.622427127
+RECOV/dw/onlinelog/group_1.257.622427131
6 rows selected.
```

Within the disk groups +DATA and +RECOV, we see that each of the database file types, such as datafiles, control files, and online log files, has its own directory. Fully qualified ASM filenames have the format

```
+<group>/<dbname>/<file type>/<tag>.<file>.<incarnation>
```

where *<group>* is the disk group name, *<dbname>* is the database to which the file belongs, *<file type>* is the Oracle file type, *<tag>* is information specific to the file type, and the pair *<file>.<incarnation>* ensures uniqueness within the disk group.

Automatic Storage Management is covered in Chapter 6.

Oracle Installation Tablespaces

Table 3-2 lists the tablespaces created with a standard Oracle 12*c* installation using the Oracle Universal Installer (OUI); the EXAMPLE tablespace is optional; it is installed if you specify that you want the sample schemas created during the installation.

Tablespace	Type	Segment Space Management	Approx. Initial Allocated Size (MB)
SYSTEM	Permanent	Manual	790
SYSAUX	Permanent	Auto	1000
TEMP	Temporary	Manual	160
UNDOTBS1	Permanent	Manual	180
USERS	Permanent	Auto	255
EXAMPLE	Permanent	Auto	358

TABLE 3-2. *Standard Oracle Installation Tablespaces*

SYSTEM

As mentioned previously in this chapter, no user segments should ever be stored in the SYSTEM tablespace. The clause DEFAULT TABLESPACE in the CREATE DATABASE command helps to prevent this occurrence by automatically assigning a permanent tablespace for all users that have not explicitly been assigned a permanent tablespace. An Oracle installation performed using the OUI will automatically assign the USERS tablespace as the default permanent tablespace.

The SYSTEM tablespace will grow more quickly the more you use procedural objects such as functions, procedures, triggers, and so forth, because these objects must reside in the data dictionary. This also applies to abstract datatypes and Oracle's other object-oriented features.

SYSAUX

As with the SYSTEM tablespace, user segments should never be stored in the SYSAUX tablespace. If one particular occupant of the SYSAUX tablespace takes up too much of the available space or significantly affects the performance of other applications that use the SYSAUX tablespace, you should consider moving the occupant to another tablespace.

TEMP

Instead of one very large temporary tablespace, consider using several smaller temporary tablespaces and creating a temporary tablespace group to hold them. As you found out earlier in this chapter, this can improve the response time for applications that create many sessions with the same username. For Oracle container databases and pluggable databases (in Oracle's multitenant architecture, new to Oracle Database 12*c*), the container database owns the temporary tablespace used by all plugged-in databases.

UNDOTBS1

Even though a database may have more than one undo tablespace, only one undo tablespace can be active at any given time for a given instance. If more space is needed for an undo tablespace, and AUTOEXTEND is not enabled, another datafile can be added. One undo tablespace must be available for each node in a Real Application Clusters (RAC) environment because each instance manages its own undo.

USERS

The USERS tablespace is intended for miscellaneous segments created by each database user, and it's not appropriate for any production applications. A separate tablespace should be created for each application and segment type; later in this chapter I'll present some additional criteria you can use to decide when to segregate segments into their own tablespace.

EXAMPLE

In a production environment, the EXAMPLE tablespace should be dropped; it takes up hundreds of megabytes of disk space and has examples of all types of Oracle segments and data structures. A separate database should be created for training purposes with these sample schemas; for an existing training database, the sample schemas can be installed into the tablespace of your choice by using the scripts in **$ORACLE_HOME/demo/schema**.

Segment Segregation

As a general rule of thumb, you want to divide segments into different tablespaces based on their type, size, and frequency of access. Furthermore, each of these tablespaces would benefit from being on its own disk group or disk device; in practice, however, most shops will not have the luxury of storing each tablespace on its own device. The following list identifies some of the conditions you might use to determine how segments should be segregated among tablespaces. The list is not prioritized because the priority depends on your particular environment. Using ASM eliminates many of the contention issues listed with no additional effort by the DBA. ASM is discussed in detail in Chapter 4. In most of these scenarios the recommendations primarily enhance manageability over performance to enhance availability.

- Big segments and small segments should be in separate tablespaces, especially for manageability and reclaiming empty space from a large table.
- Table segments and their corresponding index segments should be in separate tablespaces (if you are not using ASM and each tablespace is stored in its own set of disks).
- A separate tablespace should be used for each application.
- Segments with low usage and segments with high usage should be in different tablespaces.
- Static segments should be separated from high DML segments.
- Read-only tables should be in their own tablespace.
- Staging tables for a data warehouse should be in their own tablespace.
- Tablespaces should be created with the appropriate block size, depending on whether segments are accessed row by row or in full table scans.
- Tablespaces should be allocated for different types of activity, such as primarily UPDATEs, primarily read-only, or temporary segment usage.
- Materialized views should be in a separate tablespace from the base table.
- For partitioned tables and indexes, each partition should be in its own tablespace.

Using EM Cloud Control 12*c*, you can identify overall contention on any tablespace by identifying hotspots, either at the file level or at the object level. We'll cover performance tuning, including resolving I/O contention issues, in Chapter 8.

Summary

The basic logical building block of a database is the tablespace. It consists of one or more physical datafiles, only one datafile if you create a bigfile tablespace. Whether you're creating a permanent, undo, or temporary tablespace you can create those tablespaces as bigfile tablespaces for ease of management.

When you create tablespaces or other objects, you can use Optimal Flexible Architecture (OFA) to automatically create an appropriate OS file name and directory location. This is even more useful in an ASM environment where you only need to specify the disk group name; Oracle puts it in the right directory location automatically and you may never need to know where in the ASM file structure Oracle places the object.

In a default Oracle database installation, Oracle creates five required tablespaces: SYSTEM, SYSAUX, TEMP, UNDOTBS1, and USERS; if you choose to install the sample schemas they will exist in the EXAMPLE tablespace. You will most likely create many more tablespaces in your environment to segregate applications to their own tablespace or to restrict how much disk space a tablespace may use for that application.

CHAPTER 1

Data Integrations Overview

Data is an extremely valuable frontier for the enterprise. Some companies have pulled away from their competition because of their use of data. They understand the value the information brings in making things more efficient, knowing their customers, and advancing their products based on known needs. Data can be a powerful tool in driving business decisions and providing tremendous value. The question is how to harness this data and the power of the information.

There are two ways of approaching the data: a technical one, which will be covered in this book, and a business one. The business approach means that each company has to apply its own expertise of its respective industry and field of business in order to decide how to gain and exploit additional insights from the data. Leveraging the technology will provide a way to harness the data using those insights. Companies that leverage their data and know how to use the information realize that the efforts needed to keep the data updated are well worth it. In this sense, data should be handled as assets. The internal data has to be protected and managed to provide the details across the enterprise, and not just focused on one line of business or product. Depending on the industry, the type of data being collected will require different regulations around the data, which adds complexity to how the data can be used and shared. This adds governance around the data and reasons for business rules to manage and maintain it. The amount of data being collected might even increase questions around what data filters are important. This might also involve the need for analytics and other analysis before the data is used in conjunction with other systems.

The Big Data technologies all over the news these days might give the impression that filtering just some of the data available is a thing of the past. However, the volume of data being generated grows so fast that filtering—at least for analysis of significant data sets—remains an important processing step. This filtering of data provides the data sets needed to integrate with company-specific data for an even more powerful analysis.

The starting point for building systems that utilize the value of data is to begin asking questions. Questions should be formulated to determine the problems the company is trying to solve with the data, and a data strategy can be developed based on these questions. Data strategies are based on business needs and where to get value, and they determine what type of data is collected and integrated. It is amazing what can be done with data—from actually saving lives to anticipating customer needs. As an example, a healthcare company can become more efficient in its processes of gathering medical information and then correlating that data with medical issues to provide a more comprehensive patient history, which leads to better patient care and valuable repository of information about the patients. Another example is an IT department understanding how the various environments, systems, and applications interface within an enterprise; for example, which database supports what applications and on what servers and networks. The support of these systems requires not only issuing upgrades and patches, but also the ability to communicate with the right teams and application owners whenever an issue arises or maintenance needs to be completed. Having an effective data strategy in place helps streamline the communication issues involving maintenance and test plans. These are the types of issues that can be addressed with the analysis of the data.

Internal company data presents excellent information about the enterprise and can be used to answer several questions. Merging data from multiple sets and sources gives new and better insights than just analyzing single-source data sets. Internal systems can be integrated to provide comprehensive data for various departments. Add into that external data sources, and the data can be enriched and probably becomes more complete. The external sources can be standard data sets that are common for industries, governments, regulations, markets, and any other source of data one can think of. It is also possible to subscribe to outside data sources—and with these data sources integrating with the internal sources, the enhanced data and analytics provide more than just an internal picture, but a larger industry picture.

The questions can then be greater and further reaching to provide a deeper look and expand beyond the initial thoughts and research. Now, as the questions are flying and the company is thinking of the possibilities of how to use the data, a plan has to come together—a master plan of the data from master data management, data governance and quality, and of course data integrations.

What Is a Data Integration?

At this point, it might be fine to continue by discussing tools and step-by-step directions for data integrations, but we'll start with a discussion around what data integrations are. Data can be in various formats as well as different databases and systems. Applications that collect and use data can keep the data contained. It is when the application needs an outside source of information or has valuable information to share with other systems that the data is considered for integrations.

Taking data from one system and combining it with data from another system, based on a common identifier, for use as a data set is the underlying idea of data integrations. Numerous forms of data integration are available. It can be as simple as having a table in an Oracle database and running a query against another table in a different Oracle database. The combined query has to provide the method for the integration. This data can then be loaded into another database or a data warehouse. It can even just be joined in a materialized view (that is, a view that is a snapshot of the data) for reporting.

Data integrations most likely come into the equation when new questions are asked. There is normally an existing system, and answering a question means pulling data from another place and enhancing the data.

Data integrations mean that data can come from almost any platform and format—especially with today's information being collected from everywhere. Data is being tracked about when we sleep, exercise, and eat. The number of people going into brick-and-mortar stores versus shopping online is counted, and information is gathered about what the weather was on a particular day. Devices and systems collecting high volumes of data in nonrelational data stores can still allow data to be integrated with relational databases.

It all comes back to the data that is available for processing and the information that can be integrated into various data sets. Data warehouses load data from various sources to integrate it into a consolidated location for reporting and as a source of data for other applications to use.

Integrating data can be in done in several steps—from simple statements to complex sources of data that need business rules and controls around them to be able to consume the data for the other purposes. Tools are needed to perform the integrations, a plan is needed to understand the data and the quality of the data, and governance is needed to maintain the system to continuously provide consistent data. More importantly, domain know-how is required to formulate meaningful analysis and interpretations of the data.

History of Data Integration

Data integration used to be simpler. Most of the data would be housed in just a couple of systems. There were not as many options to get data, and not as much information was yet being provided from the Internet. There were fewer sensors and data collection processes in production lines and at consumer sources. Smartphones were not yet around either. Maybe the right questions were not being asked to see the need for different data sources. The applications collecting the data also fed data warehouses to provide the consolidated data.

A data warehouse can have historical and current data, but the main purpose is to centrally store the data. Having this centralized repository allows the business to point to the data warehouse for reporting and analysis. A data mart is a focused and simpler form of a data warehouse; it is used for one subject or functional area. Populating the data warehouses and data marts requires the processes for data integrations, as data warehouses continue to be an important way to store the integrated data.

The amount of data is continuously growing. Today, more sensors and objects provide constant data, and some of these can even talk to smartphones. In addition, the Internet continues to provide more and more information. In other words, there is not going to be any less data going forward. Hopefully, though, it is *smarter* data—that is, the right information being gathered, along with the right questions being asked.

In the past, it was also typical that data integrations were an integral part of migrations to a new application. Either one company purchased another company or a new system was implemented. This caused a need for the existing systems to be migrated into one for the new entity. Migrations are a whole project in themselves, but they usually have data integration aspects as part of their execution. When combining the two systems, it would make sense for the company to integrate the data as part of the migration plan.

Outside of the data warehouses and migrations, many of the systems were built as stand-alone environments. It was not always expected that another system would ever use the data in these databases. There was no plan or thought given to what would happen if these data sources were combined to provide additional information.

Data integrations have taken on the following forms:

- **Manual integrations** These integrations use queries to pull out the needed information.
- **Common and uniform access** The data has the same look and feel and a single access point for the client.

- **Application integrations** Applications pull data from different sources to feed the results to the user.
- **Common data** Provides a single access point for the data that is gathered and pulled together for the user. This involves the use of the following:
    - Queries and materialized views
    - Data warehouses
    - Common data stores

Common data can also mean common data storage. Queries, materialized views, and portals are used to provide a common access point for common data. It is a way for the data to be combined, to make the database uniform, and use queries and reporting access. Using a data warehouse is another way to integrate data to a common source. There might also be additional data stores for a common place to access and store data.

A group of databases is an example of a model used for relational databases to be logically joined together for integration. The data can be in different databases, and still a query across databases could join the data because they share a similar query language and structure. This doesn't necessarily mean bringing the data into one physical location; different sources can be used to create views or reports and provide the integration of the sources of data. This illustrates a simpler method because the data types are similar; it isn't until NoSQL that unstructured data sources start being added in. Peer-to-peer methods enable a peer application access to another. Again, the databases are dispersed, but access can be consistent and have a single access point.

With these ways of integrating data, there are still several issues with data integrations, and no single way to solve these problems.

Integrations Today

Issues that exist today with data integrations mainly involve large sets of data that keep getting larger and need integrations and automated processes. Performing data integrations is not easy, but the value of the data is worth the effort.

The systems that need the data integration might only require a filter of read-only data to form a new set based on the integration. This filtering process might actually exclude data from the integration because no one knew the right questions to ask or maybe even the wrong questions were being asked. Therefore, data might be missing because of this exclusion if different questions are asked later or other data is required. Missing data or data that's not provided is something that is seen regularly in technology systems. There are discussions concerning servers, the applications and databases on these servers, what other servers have dependencies, and the owners for all of these items. With constant changes to configurations, applications, and owners, it is difficult to keep the data up to date, so it might be left out of the integration. There might also be other systems that keep part of the data, such as the configurations. This data is not always included in the integration through the filtering because it is not regularly part of the requirements—that is, until a question is asked or new reasons arise to keep the data. The process is just updating information in a system, but even this list of servers, applications, and owners could provide key details for automated systems or operational processes to create efficiencies. These systems might have different sources of data for all of these pieces of information, which are normally integrated from these different systems to pull together a system of record for the server. There seems to be several operational processes and procedures that can be solved by having consistent configuration management data and information. The right filtering or the realization that there are other uses for data needs to be taken into consideration for the data integration process. Filtering data too soon or not asking all of the questions up front can cause sources to be restricted and limited.

This leads into another issue: the fact that systems were not designed for integrations. If there was no initial requirement for another application to need the same information, it was not built into or planned as part of the application. New systems should be designed for others to consume the data. The company should assume that data is not being used for just one thing and that at some point someone is going to want to use that information for another purpose. During the development and data workflow stages, designs should include ways to extract the data and make it usable by others and different applications. Also, APIs should be provided through code, data stores, views, and so on. Quite a few options are available, so including this in the design of the application or just planning for others to use the data will be advantageous for future data integrations.

Figure 1-1 shows the components needed for data integrations to happen at the enterprise level.

The components listed in Figure 1-1 will be discussed in more detail throughout the rest of the book. Notice that there are some initial architecture and data governance steps with data classification and master data management that will make the data easier to integrate. Developing these strategies leads to better data and better integrations. Using common formats and understanding what the data is are both keys to being able to consume the completed data set. The business needs can flow into these steps, but it's still important to ask the right questions.

In order to look into how to perform data integrations, we'll discuss these topics of the data components first. Understanding how the environment is set up and how data requirements and issues are a part of the development of an integration strategy. The components are shown in no particular order, and some environments are going to have more advanced definitions and processes in these various areas. To have effective processes for the data so that it is providing value, these areas need to be addressed.

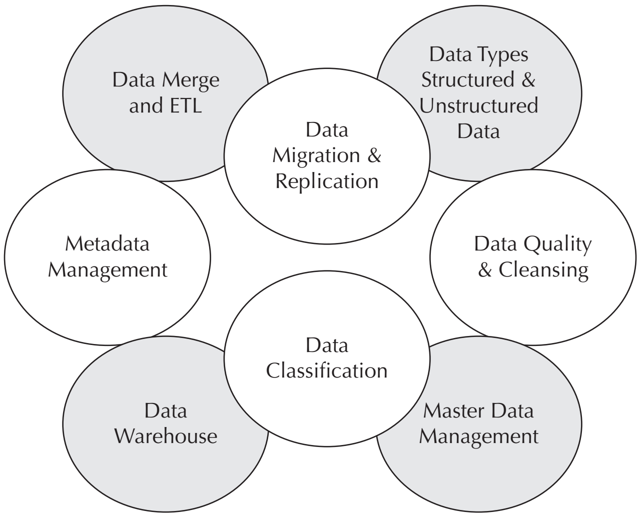


FIGURE 1-1. *Data integration components*

To begin, master data management (MDM) provides the specifics about the data and data sources. The sources of data are defined and documented. With external sources of data, the source of the truth and data formats will be part of an MDM strategy. Knowing and understanding the data and where it is coming from allows for reuse in other systems. For proficiency, master data management requires business and upper management support. It also ties into data governance and how the data is used in the company. You can see why this ties into data integrations. Not only can the data integrations be used in the MDM strategy, but the understanding of the data is essential for some of the data integrations.

Data classification is a component that can be used in the data integrations workflow. The data can be sensitive data, a source of truth, or another classification. The sensitivity of the data is important to know if the data needs a certain level of security and therefore might be restricted in use with integrations. The classification of being the true source of data is integral for where to pull data from in order to match it up and use it for mappings. Data classification tags the data with the needed attributes so that the company knows when it can and can't be used, or with what systems it should be used. This can be part of the development strategy in planning for future integrations or for current processes to use the right source of data.

Data merge and ETL (Extract, Transform, and Load) are processes that use the data source to match and map the needed records and fields. The transformation process can use data integrations for the proper mapping in order to load the data into a data store, data warehouse, or other application. The data merge process is really performing the data integration, in that it merges data sets together. These tools perform the merger or transformation of the data to load the results as needed. Chapters 2 and 3 discuss how queries and other database processes perform these steps.

Data migration and replication can be done in various ways as well. Normally, a mapping of the information is needed to be able to migrate the data to a new system. The migration might involve the same data moving from one system to another to be used by a different application, or it might be two or more applications combining their data to be migrated to a new data store or a whole new application. Nevertheless, these processes perform data migrations to move the data. Replication also duplicates the data to be used in a different place. These components are discussed in more detail as part of Chapter 4.

Because data comes in all shapes and sizes—from external and internal sources—there needs to be a way to bring the data together. Integration of similar types of data provides for more straightforward processes and similar data stores. Unstructured data and data from outside the realm of relational databases require different ways of integrating. Chapter 8 focuses on Big Data and how to pull it. Enhancing existing structured data with unstructured data is where the analytics get interesting and opens up a whole new set of questions and possibilities. These challenges are where the future of data integrations lies because of all the new data coming in now.

Even though data quality and cleansing are really strategies that belong with master data management and data governance, poor quality data will create havoc on any data integration. The testing of the data quality as well as the data cleanup processes is vital to the systems. Because of this, the topics of data quality and cleansing receive a complete chapter: Chapter 7. Not only can data integrations help with the data cleanup, but having the quality of the data tested and confirmed will provide consistent and reliable data integrations. Data quality brings with it a whole set of challenges and needs to be addressed before a data integration for strategic data use is attempted.

Even though the use of a data warehouse is a past way of bringing data together, it is not a dying breed, and will continue to be vital going forward. Data warehouses might transform into different platforms and have different types of data available, but they provide a stable set of data that uses standard processes to have the centralized repository of data. Data integrations are used every step of the way for data warehouses—from loading to reporting. Every data store can be a small part in a bigger data warehouse picture. Simplifying an already complex set of data coming from all over the place to a centralized location can be an asset for several lines of business. They might be pulling data from different sources and integrating it with nonrelational sources on the fly, but for performance in reporting and analytics, having the combined data in a warehouse will provide a way to decrease the access time with the right set of information. The steps along the way might not store the integrated data because of the APIs or the way the data can be pulled, but that doesn't mean the data can't end up in a data warehouse.

Metadata management defines the columns and what the data actually means; it allows for proper data mappings. Without the metadata being defined and communicated, different lines of business might be using a certain column or table for something it wasn't intended for, and thus creating integrations that produce incorrect sources. The characterization of the data is also part of the data cleansing, quality, and master data management efforts. Are you starting to see a pattern here? These components are all tied nicely together; they are integrated just as the data inside of them needs to be integrated. Being lopsided in components or not spending the time to manage the metadata or the quality of the data will lead to bad information. Decisions that are made from poor data will have the opposite effect on the business as using the right information with the right data and asking the right questions.

It is not easy to align all these components. However, if the right questions are being asked now, and the data is available to answer them, it will be well worth the effort to plan, manage, and maintain a system that has integrated data and tools to report and visualize the data, thus resulting in tremendous business value.

Decision Flow Chart

After the discussion of the various components of data integration, decisions need to be made about how to handle the data flow—decisions about the technologies, tools, and how to match the business needs.

These decisions involve what data is needed, frequency, management tools, and how to handle changes. The stakeholders need to be involved in helping make these decisions because they have a vested interest in the data or the processes. Technologists cannot make decisions in isolation about what tools to use because the business and data owners will be the ones using them. Data owners can set the data definitions and the frequency required, but they need help in deciding what tools to use to accomplish what they are looking for.

The problem is that not everyone is communicating. Perhaps the data owners are working in isolation because they don’t believe they need other data and are not sure that the data should be shared. The first decision that actually has to be made is who is going to own the data and if the data, or what parts of the data, are going to be shared.

A lot depends on making good decisions and communicating with the various teams. Successful data integrations have proper data requirements. As shown in Figure 1-2, data requirements comprise the next couple of decisions. Defining the owner of the data will help with what data is accessible because that will provide a responsible party for making the data that should be available and ready for use. Not all data will be available for consumption for data integrations, but as previously discussed, there are steps surrounding metadata information to explain what the data is and what workflows it

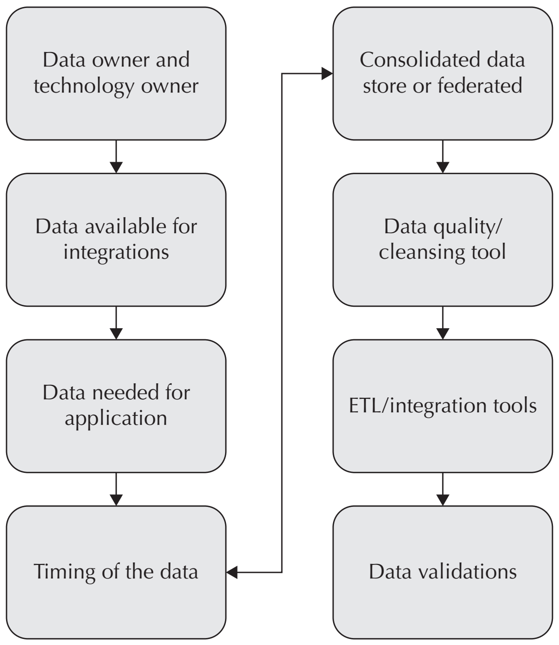

Data owner and technology owner
Data available for integrations
Data needed for application
Timing of the data
Consolidated data store or federated
Data quality/ cleansing tool
ETL/integration tools
Data validations

FIGURE 1-2. *Decisions for data integrations*

came from. When planning an application or data system for integrations, the company must build in APIs and other ways to use the data. Also, communication with other teams is necessary for understanding these pieces and learning what the other teams want. You can communicate what is being made available and how to use the data through the workflows. Technology talking with the business will drive the integrations together. These discussions drive the flow of the data and get the necessary data available. Deciding together what data means and working through the needs is definitely a better way to get the right tools in place. This also sets the direction for data quality, reporting, and successful data integrations.

The timing of the delivery of the data might be another area where the business wants something that might not be possible technically. There might be different understandings of what the timing of the data delivery is. If the data is only delivered from an external source on a daily or weekly basis, how does that change the timing of the data integration? The timing might change based on when data is available and usable. Real-time data might not even be possible because of workflow processes. If manual checks and processes are in place, these need to be reviewed so they can be automated before data can even be expected any earlier. This can be included as part of the metadata and definitions surrounding the data for timing and details.

Once it is decided what data is needed, the appropriate teams need to be made aware of its timing and what data is available; then the data can be loaded into the consolidated data store or data warehouse. If a federated system is being used, the data can be placed in different areas to be pulled together with processes.

The data can be cleansed at this point. The cleansing of the data will depend on the rules and business logic surrounding it. Data cleansing is discussed more in Chapter 7. At this stage, the data is either corrected or completed, and can now be used for ETL processing or for data integrations.

Data validation is an area that is significant for the entire process because it determines if the data that is delivered is what was expected. Testing against the data will verify that the workflows are working and that the data is defined properly. Data validation is an on-going procedure to confirm several steps in the overall process. Not only does it confirm that the data integrations are working properly, but it also confirms the steps to get to the integrations. There should be checks along the way for data validation, especially when data quality steps and master data management are used. The business rules should be verified against this, and if changes need to be made, the process can circle back through the workflow to validate the data and the changes.

Decisions concerning data integrations start out with communication and having the data owner be responsible for the data. Understanding what the data is currently being used for and the intent of the data will then allow the process for the data validations and data integrations to be designed and created.

Tools for Harnessing Data

Data integrations are more than just technology and mappings of data in a database or environment. There is obviously value in incorporating the data into a common source for analysis. Process work and decisions need to be made with the data owner and applications owners about the data and how to integrate it. Even though some of this process is just data owners discussing the data, there are reasons to pull in tools to make the other work flows with data integrations consistent and repeatable.

Tools such as Oracle Data Integrator and GoldenGate can help with integrations and migrations. In addition, they help automate and track information concerning what has been done and business rules. Oracle Data Integrator is discussed more in Chapter 4. It can be used to set up mappings and business rules and to run through data integrations and data validations.

GoldenGate is for replication and is also discussed in Chapter 4, which provides you with guidance on how to use GoldenGate for data migrations and data integrations. Thinking of replication as part of a data integration strategy is not always intuitive, but using it to provide the source of data in another system and keeping it synchronized without manual intervention is critical for the right data to be replicated over.

Other tools such as SQL*Plus, database features such as materialized views, and external tables can also used for integrating the data. Whether joining tables together or loading an external source through stored procedures, relational databases provide solutions for integrating data. Even though the sources might not be in relational databases, extensions for databases can be used to combine the data in a view or to load the data into a data store. Using features from the database allows for automation and data checking. You'll find more of a discussion about using queries and these features in Chapters 2 and 3.

The database is a powerful tool, for both structured and unstructured data. Storing and integrating data can be part of the design of the application, which then also provides a way to allow others to access it. The security around data integration can match that of the database as well.

The tools are key to managing and implementing effective data integrations. They provide a way to automate, test, and track what is being done. Integrations are already difficult enough without the tools to maintain the process. The rest of the book will discuss in more detail how to use these tools effectively to have a system for managing data integrations.

Summary

Data used for analytics and reporting is very valuable to companies. This data can give a company a competitive advantage and allow for more efficiencies in improving technology processes and providing cost savings. Combining systems internally as well as with external sources of data can be a very involved effort. A workflow process and checks and validations need to be built in. Applications should be designed with data integrations in mind by providing APIs and other ways to access their data.

There is no easy way to provide all the data integrations needed. The data needs to be reviewed for quality, and it doesn't work without data owners, governance, and communication. There is not always a technology or tool that can help provide the solution needed. Rather, a decision needs to be made about what data is needed and what that data means.

Questions need to be asked—questions that drive the analytics in figuring out what is needed to move the business forward, provide the right solutions for customers, and result in a competitive edge. Data that is pulled from the various available sources, maintained in a secure system, and available at the right time for those who need it can be used to answer these questions. Tools to help harness the data are available, and the combination of these tools with business processes and workflows will provide the data integrations.

CHAPTER 1

Introduction

"I see what you mean." We understand and interpret the world through our sense of vision. If we hope to share understanding in large organizations, we have to find ways to communicate a consistent, coherent message to hundreds or even thousands of individuals across large distances, time zones, and even cultures. The fastest and most effective way to do this is through the presentation of data-driven insights displayed as graphs, tables, maps, simple statements, and patterned visuals. Business intelligence dashboards and reports are exactly this—attempts at visual communication. If we are to communicate effectively, however, we must pay close attention to the visuals we present to each other.

About Oracle Business Intelligence 11*g*

Oracle Business Intelligence 11*g* is one of the most capable and comprehensive business intelligence platforms in the marketplace. The average user size for an OBI 11*g* implementation is more than 2,000 users. These are very large, very complex implementations. Building an OBIEE implementation is much like constructing a 40-story office building for several thousand employees. Many of the tools, techniques, and data structures are necessarily geared to a very large scale. In contrast, many smaller business intelligence systems operate at a decidedly smaller scale. This is particularly important to the discussion of data visualization, or, if you prefer, design. The approach one takes to designing a functional modern skyscraper and making it "beautiful" is somewhat different in terms of the materials, tools, and techniques that are used when contrasted with designing a modern house and making it beautiful. Much of the "beauty" that lies in a modern office building exists in the functional environment of moving people physically through the structure and providing them expected services (such as plumbing, heat, air, light, and so on). There is a fundamental difference between designing something practical that is expected to be used simultaneously by thousands of people and designing something customized for a single family.

Business Intelligence System Goals

One of the most important attributes of a large enterprise business intelligence system is its ability to drive a common understanding of an organization's business situation. This situation can be characterized differently. We often organize analysis in three ways:

- **Position analysis** looks at the "state" of the organization at a point in time. You can think of it as a "snapshot." That snapshot can use a "wide-angle" lens and capture a very broad landscape from great distances or heights, or it can be highly focused and extremely detailed.

- **Performance analysis** characterizes what has happened over a period time, with specific attention paid to the end position. This typically involves summaries and "slices and dices" of categorized information.
- **Flow analysis** evaluates a particular type of data or account and how additions and subtractions to it change over a period of time. Although most people are familiar with (or have heard of) cash flow, there are several other types of flow, such as inventory flow, customer flow, data flow, and so on.

There are almost always multiple ways to visualize data, just as there are multiple ways to characterize analysis. There is not a "defined hierarchy" of value in which we can say "this is better than that, which is better than the other." There are always multiple perspectives and methodologies, and they all have both advantages and disadvantages.

NOTE
We will stay focused on the topic of data visualization and not address the inner workings of OBIEE software and the complexities of its environment. For instruction on how the software works, several excellent titles on Oracle Business Intelligence are available—in particular the Oracle Press book Oracle Business Intelligence 11g Developers Guide by Mark Rittman.

Understanding visual perception and the representation of quantitative information is a life-long study, and far more content has been collected on these subjects than can be presented in this book. Reports, dashboards, and interactive BI displays all share the same issues of the most optimal way to present information so that it informs users and supports decision making. The need has never been greater to translate vast amounts of data into information that provides evidence for choices between alternative actions and promotes a shared understanding of business situations and situational dynamics.

This brief overview highlights three key concepts:

- BI reports and dashboards should be viewed primarily as communication devices, and both the principles of human cognition and the needs of the individual user should help guide their proper use.
- BI reports and dashboards are used either in the exploration of data or in the explanation of data.
- It's much easier to misuse BI tools than to use them well.

Humans Evolved to Sense the World, Not to "Do Numbers"

Computers are very powerful tools for manipulating large sets of data and performing all kinds of mathematical operations, including aggregation, division, correlation, regression, K-means attribute clustering, and Markov Logic Network construction. However, it turns out that as human beings, we're not terribly good at seeing objects and translating them into numbers. Indeed, once there are more than about seven of something, we have a hard time counting exactly how many there are at a glance, and we settle for knowing that there are "a whole bunch."

We're even worse at visualizing basic mathematical operations such as addition, multiplication, and division. Visualizing complex mathematics takes a tremendous amount of time and practice, and like juggling while riding a unicycle, the average person can't do it easily. We humans are good, however, at other things, such as finding patterns in raw visual data and constructing three-dimensional schemas; we dynamically interpret colors and light levels and the size and angle relationship of lines. We're good at understanding moving objects and motion in general; we're good at navigating landscapes; we're superb at recognizing patterns. In fact, we're so good at recognizing patterns that we insist on seeing them even when they're not there, and we often refuse to acknowledge a new pattern that violates an existing pattern. Our brains are optimized for helping us survive in the wild, but not for deciphering BI dashboards and reports.

We all know that BI systems provide value to organizations only when they are used. Calvin Mooers coined his famous Mooers' Law and its corollary in 1959:

> *An information retrieval system will tend not to be used whenever it is more painful and troublesome for a customer to have information than for him not to have it.*
>
> *Where an information retrieval system tends not to be used, a more capable information retrieval system may tend to be used even less.*

This reminds us that there may be a natural resistance to using BI systems in many situations. BI systems may point out situations that managers don't want to address. Compounding this, when BI systems poorly present or distort data, they ultimately lead to misuse, mistrust, or abandonment of the system. Proper visualizations and data presentation lead to business insights and build trust in the system. As executives and managers begin to rely on them, they improve their decision-making abilities. Effective BI interfaces also build a more coherent and consistent view of the business and its operational environment.

Basic Principles of BI Dashboards

The effective implementation of BI systems requires both knowing the basic principles of data communication and thinking critically about who is using a BI system, how they are using it, and what their needs and goals are. In his seminal work, *The Visual Display of Quantitative Information,* Edward Tufte emphasizes five key principles:

- Above all else, show the data.
- Maximize the data-ink ratio.
- Erase non-data-ink.
- Erase redundant data-ink.
- Revise and edit.

If Tufte's advice is to be followed, only information that is absolutely necessary for the contextual understanding of the data will be depicted. The general rule for BI displays is "less is more." Eliminate as much visual clutter as possible and let the data present itself as simply as possible. Drop shadows, 3-D effects, and extra graphic elements should be avoided because they draw attention away from the data. The purpose of business intelligence systems is to relate a clear message about data that is easily understood and interpreted consistently across the highest percentage of users. It is not about entertainment or visual interest for the sake of decoration. Designers of business intelligence reports, graphs, and dashboards should approach data visualization the way Strunk and White approached writing in *The Elements of Style,* by stating their case with "cleanliness, accuracy, and brevity."

Many of the built-in data-visualization tools such as graphs suffered as computers became more powerful and additional "visual effects" were added—not for the sake of communicating a message more effectively, but rather for the sake of "eye candy" or simply because the effects had become possible. Software designers forget that data visualization is a representation or a visual metaphor, and the emphasis should be on making it as easy as possible for people to interpret and understand the information consistently and accurately. Instead, they get sidetracked by trying to represent physical objects, by replicating cockpits and physical dashboards designed for very different purposes, such as flying a plane, and by adding unnecessary design elements unrelated to analytic communication needs. The best example of this is the use of three-dimensional renderings of pie charts, bar graphs, and line graphs. Three-dimensional renderings do not add any quantitative content that is not present in two-dimensional renderings, and they misrepresent and distort values in order to add the illusion of depth. Software designers contribute to this problem by showcasing new features in a product that implementers then copy in an attempt to appear "fresh" or "cool."

Two books in particular offer clear and accessible information on human cognition and visual processing: *Visual Intelligence: How We Create What We See,* by Donald Hoffman, and Information *Visualization: Perception for Design,* by Colin Ware. These works provide the scientific justification for the summary statements in this book.

Every schoolchild is exposed to optical illusions and understands that magicians trick us. However, adults (particularly in large organizations) sometime forget that the presentation of information must be designed carefully according to the way it is perceived. This involvement in and active guidance of the visualization process is sometimes less than ideal. Too many people will simply accept the system defaults set at the time of installation, but these are seldom reflective of fundamental data visualization best practices. Of course, this does beg the question of whether an organization should set system defaults and establish an organizational style guide so that those who are less inclined to edit and improve the presentation or who are simply in a hurry do not produce poor results. This important topic is addressed more fully in Chapter 12.

BI Systems Need Training

BI implementations typically require tremendous time and money, but also offer the potential for significant returns in comparison with the investment in developing and deploying the system. Just as most developers benefit tremendously from training, not only in the functional aspects of software systems ("this button does that") but also in basic system architecture strategy and data flows, users become far more effective in reading and understanding a BI system when they are shown both the basics of "how" and "why."

Most executives and managers have not had training in visualizing data, and many may also have not had training in analysis techniques and are therefore unlikely to do either properly by chance. The most successful BI implementations "finish the project" by including a training budget that is not spent within a compressed amount of time at the end of implementation when everyone is exhausted. Rather, a relatively modest portion of the total project budget should be allocated to training and workshops and should be spread over the first year of implementation. A series of classes on visualization and data analysis with executive users in combination with follow-up sessions (often one-on-one with highly placed executives) reinforce the information and ensure that the BI system is fully leveraged by the organization. What people can learn in initial training is limited because they can absorb only so much information at a time, so these follow-up sessions allow those who will rely on the BI system to expand their use of it more completely. As they gain experience, they are able to learn more and leverage the tools in a more sophisticated and complete manner.

Dashboard Best Practices

What is the most important part of your dashboard? If you want to draw attention to certain areas of your dashboard, you need to know what draws the eye. The three most powerful ways to draw attention are motion, color, and alignment/position.

Motion Demands Attention and Cannot Be Ignored

Motion draws the human eye more effectively than size, shape, color, pattern, or any other visual characteristic. It is now possible in many dashboard systems to embed scrolling messages and incorporate moving displays of data. These displays will command attention, and if the user requires constant monitoring of changing data, such displays can be extremely effective. However, these displays can also be extremely annoying. Using motion can be distracting and often calls attention away from other important features of the dashboard interface. Make certain that motion is used sparingly so that the dashboard doesn't become distracting and annoying to the user community.

Color Is Powerful

Color is a powerful visual clue and should be used consciously and sparingly. Colors will stand out immediately against a plain background but can easily be missed when bright and overly garish colors dominate the screen. The overreliance on bright colors is a major drawback of many BI dashboards and reports. Bright colors should only be used in exceptional situations to call attention to unusual circumstances.

Keep in mind that approximately 10 percent of men and 1 to 2 percent of women have some form of color blindness. Red/green is the most common form of color blindness. Therefore, designs requiring the distinction between red and green are best avoided for general use. Also, the more color is used, the less effective it is. Soft, muted colors are recommended for the vast majority of visualizations. The online tool ColorBrewer 2.0 (colorbrewer2.org) offers several selections of color palettes that are professionally designed. Although ColorBrewer was designed with map interfaces in mind, its color palettes are also good for most dashboard designs. See Chapter 11 for more information about color choices.

Alignment and Position

Humans are relatively good at comparing and seeing alignment (or lack thereof), which is why we're so quick to understand and interpret basic bar graphs. People can immediately see fine distinctions between adjacent bars and whether they're higher or lower. We tend to form patterns so that we see "wholes" before we see "parts." Most people using business dashboards read from left to right and from top to bottom, so choosing where you place things and how you organize your overall layout is very important.

As good as we are at seeing alignment, we're actually not so good at judging relative sizes. If you want people to see that something is bigger than something else, it has to be significantly bigger. Size can indicate importance on dashboards, but only in the sense that "this is excessively, unusually large so that you'll look at it."

A Little Bit about Tables

When precise values are required, it's generally better to show numbers in text rather than as a graph or some other complex visualization. Eliminate grid lines in tables or render them in a light gray. Basic tables are best used for data lookup, not for data comparison. Other visualizations, including charts and graphs, are useful in comparisons and pattern recognition.

Most tables can be immediately improved through the removal of unnecessary gridlines. When tables were hand-drawn, gridlines enabled people to keep their columns and rows straight. If tables are properly designed, gridlines are generally unnecessary. Place related information in close proximity and provide space between unrelated data. This will help the user understand the layout of tables more than trying to separate information through the use of lines. It can also be effective to use highly contrasted display styles with different tables to help differentiate between various data sets. One of the real strengths of OBIEE is its ability to combine data from different sources for simultaneous presentation. One of the most basic methods for communicating "hey, we want you to see these data sets at the same time, but you should be aware that they are different" is to use different formatting and styles for them. Of course, this only works if you are otherwise consistent in your use of formatting and styles. Differences should always be a conscious choice to communicate to the audience, not a result of haphazard development or design.

Although massive tables can be displayed, requiring users to scroll excessively should be avoided. If scrolling is unavoidable, make sure the titles and headers are locked so that users can immediately see what an entry is associated with. Many tables suffer from the display of too much detail. Particularly for budgets and forecasts, where future values are estimates, excessive detail not only clutters the interface, it implies a level of precision that does not exist.

Conditional formatting asks the system to apply a format such as a background color to a table cell based on the results. This can vastly improve the user's ability to recognize a significant value because color draws the eye very effectively. However, a screen of blaring colors does little to impart meaning. The sparing use of soft colors can more easily attract attention to a particular value than can a screen of bright colors. Conditional formatting is especially powerful for data exploration when users are looking for anomalies or for patterns in the data. Regular reports can often be improved by removing colors that do not highlight extraordinary information or are not communicating a pattern directly (as they are in "heat map" styled tables). It is best to avoid putting any text in color because colored text is more difficult to read.

We often sees dashboards with a large selection of prompts where users can assemble very large tables containing dozens if not hundreds of columns. Although the desire for some executives and managers to "have everything" available for inclusion on a dashboard is understandable, organizations should not encourage these "one table to rule them all" strategies. Every element (table, graph, text, icon, and so on) that is placed on a business intelligence dashboard should have a primary purpose and then be designed to best accomplish that purpose. Broadly speaking, dashboard prompts and selection mechanisms should not function as unlimited query design tools. Users who want to perform ad hoc analysis on large, complex data sets should generally use OBIEE's "Analyses" interface (also known as "Answers") and learn how to appropriately filter and form their queries. Of course, exceptions can typically be made for highly placed executives who lack an interest in learning how to create and edit their own analyses but still possess a strong desire to define large tables of numbers.

Chapter 2 covers these points in greater depth and gives other tips specifically on using tables.

Background Thoughts on Graphs

When we design a graph, we have to carefully think about what it is we want to convey. Thoughtful consideration of choices between alternatives is the key to designing effective graphs. All graphs have a primary message or purpose. Sometimes that message is determined in advance, and the graph is designed to communicate that primary message to a broad audience. Sometimes graphs do not have a predetermined message, but rather are designed to uncover or reveal patterns and relationships in data there were previously unknown. It should be noted that data analysis and perception are individual activities, like reading a book, and are not a shared experience such as attending a concert. Although some may argue that the search for new insights is the primary purpose of business intelligence systems, for many large organizations the primary value of business intelligence systems lies in the creation of a shared understanding of business situations and dynamics and fostering a sense of strategic coherence often is difficult if not impossible without a shared foundational view of organizational data. These shared and common presentations of business information should be designed to present an objective, agnostic view of business situations.

Organizational Dashboards Typically Feature Explanation Views, Whereas Individual or Departmental Dashboards Typically Feature Exploration Views

Exploration involves individuals or small teams discovering new, previously unknown or unrecognized insights. Think of exploration as a process of "finding." Newly found insights can often inform a decision that is taken by the discoverer, but often these findings must be shared with others in the organization who are also involved in making decisions and would benefit from the newly discovered information. Explanation is communicating a common message to a group or organization. This ability to accurately convey information or evidence to a large, diverse group of people in an organization helps build coherence in decision making. Think of explanation as a process of "communicating." Insights discovered during exploration need to be shared in a consistent, effective manner. Dashboards designed for exploration are often necessarily different than dashboards designed for explanation.

A carefully designed visual presentation of a major point does not mean the view is distorted or biased. To the contrary, visualizations have to be designed carefully in order to avoid bias, distortion, and confusion arising from inconsistent interpretations. Indeed, the worst kinds of distortions are those unintentional or unconscious ones that arise because of a lack of care in the design process. Just as someone needs skill and practice to prepare excellent-quality meals, conscious decisions regarding details are necessary to prepare excellent data visualizations. Although it's possible to get lucky and fix something tasty for a big crowd without much prep, making carefully considered decisions each step of the way greatly increases the chance for success.

Data Visualization Graph Views

There are four common data visualization graph views:

- **Line graphs** Line graphs are best used to depict a pattern over a continuous range (such as time). Unlike bar graphs, line graphs can be valued within a range to highlight more granular detail without distorting the meaning of the chart. Any time a different data range is used, it should clearly marked. Line graphs should maintain a rectangular shape (roughly according to the Golden Proportion, or approximately 5:8). If the graph is excessively tall and narrow, the data will show an excessive amount of change. If the graph is short and wide, the change will be minimized.

- **Bar graphs** Bar graphs depict the value of nominal data. Bar graphs should start with zero and use a clear scale. Bar graphs are often used for comparison of the value of data items in a group with one another. Bars should be depicted as two-dimensional objects.
- **Pie graphs** Pie graphs are used for the comparison of the size of individual data items in a set with the size of the whole set (most typically as percentages totaling 100 percent). Pie graphs are not effective when too many items are included (more than seven or eight) and are best used for approximate relationships. Data visualization guru Stephen Few recommends avoiding the use of pie charts altogether. Pie graphs should never be depicted as three-dimensional objects, because the relative size of the pieces of a pie are distorted to achieve the illusion of perspective.
- **Scatter plots** Scatter plots depict combinations of two measurements—one on the x-axis and one on the y-axis. They are most useful for visually displaying the relationship between those two measurements. Scatter plots can represent hundreds of individual data points and are useful for seeing overall patterns in the comparison of two variables.

Chapter 3 covers these points in greater depth and gives other tips specifically on using graphs.

Map Views Communicate Effectively

The new inclusion of map views as a native view type in OBI 11*g* adds greater value than almost any other addition. People intuitively recognize and know how to navigate landscapes and easily make the abstraction to geographical representations of location. Spatial representations of data make sense to most people and provide an extremely dense visualization. The interactive capabilities of maps further promote the involvement of users and offer an ideal interface for master detail linking and other interaction effects.

Chapter 4 covers these points in greater depth and gives other tips specifically on using maps.

Dashboard Design Examples

Let's now look at some of those general principles in a sample dashboard. Think of this as a "sneak preview" of what lies ahead in other chapters.

Oracle's OBIEE SampleApp

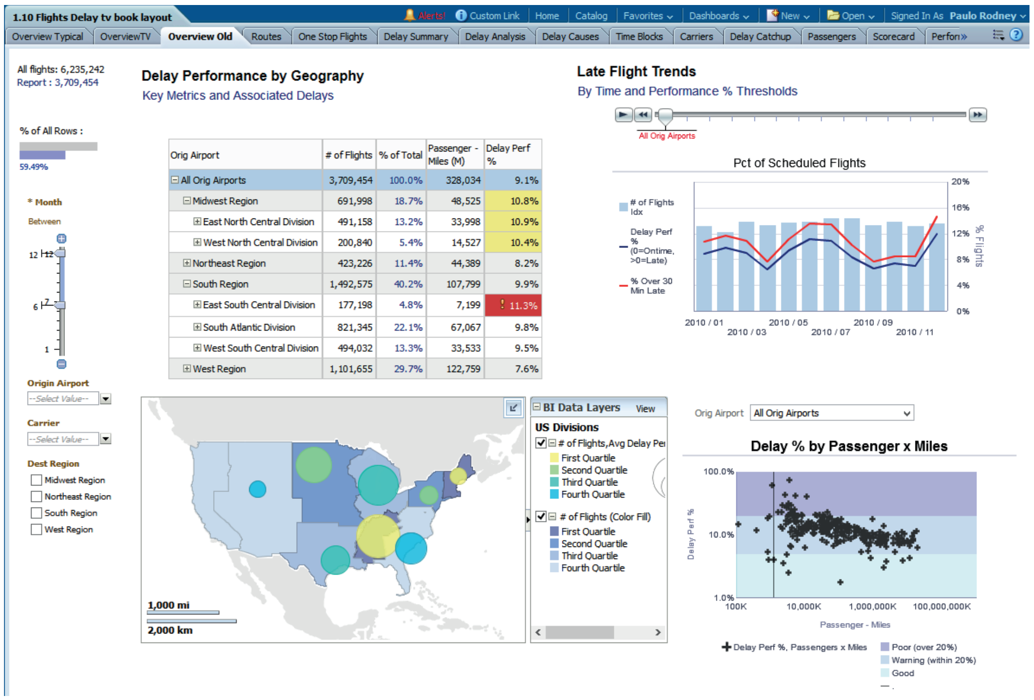
Throughout this book we will be using Oracle's OBI SampleApp Virtual Machine as a source for information and inspiration. Most of the examples are pulled from SampleApp V406. You can download the SampleApp virtual machine at the SampleApp home page at:

> http://www.oracle.com/technetwork/middleware/bi-foundation/obiee-samples-167534.html

The OBIEE SampleApp is a standalone VirtualBox VM for creating a comprehensive collection of examples and integrations designed to demonstrate Oracle BI capabilities and product integrations.

The Sample Dashboard Is a Good Start

Let's look at the 11.10 Flights Delay overview dashboard page, pictured in Figure 1-1, from Oracle's SampleApp V406. This dashboard has several attributes that make it a



Delay Performance by Geography
Key Metrics and Associated Delays

Orig Airport	# of Flights	% of Total	Passenger - Miles (M)	Delay Perf %
⊟ All Orig Airports	3,709,454	100.0%	328,034	9.1%
⊟ Midwest Region	691,998	18.7%	48,525	10.8%
⊞ East North Central Division	491,158	13.2%	33,998	10.9%
⊞ West North Central Division	200,840	5.4%	14,527	10.4%
⊞ Northeast Region	423,226	11.4%	44,389	8.2%
⊟ South Region	1,492,575	40.2%	107,799	9.9%
⊞ East South Central Division	177,198	4.8%	7,199	! 11.3%
⊞ South Atlantic Division	821,345	22.1%	67,067	9.8%
⊞ West South Central Division	494,032	13.3%	33,533	9.5%
⊞ West Region	1,101,655	29.7%	122,759	7.6%

FIGURE 1-1. *The Flights Delay overview dashboard from Oracle's SampleApp V406*

significant improvement over the typical dashboard of large tables seen every day in large corporations and government agencies. After we review some of the key features of this dashboard, we'll look at some suggested improvements and a slightly different version that should set the stage for the rest of the book.

The Flights Delay dashboard summarizes and presents publically available information regarding flight departure and arrival information for several years. Information regarding delays and their causes is also included.

The Flights Delay overview dashboard has more visuals than tables. Typically, a minimum of 60 percent of the dashboard should be composed of graph views rather than table views. The ratio of three graphs to one table is about right. The prompts are organized on the leftmost column. Placing the prompts in that position or along the top of the dashboard provides a consistent location for users to easily find them, and they do not move depending on the content presented in the dashboard (OBIEE dynamically adjusts the position of content based on the data returned).

At the top-left corner, you can see a small two-cell table and a small summary bar chart underneath it, as shown in Figure 1-2. This "contextual" information regarding the current data selections and what is being represented in the table and graph views is valuable to users. The raw numbers tell the user that out of the 6,235,242 flights in the data set, only 3,709,454 are being reported. This table is actually created via a narrative view. Small tables are typically more useful than large tables. One of the most common data visualization "mistakes" is an overreliance on big tables. The meaning and purpose of this table is clear, and it's extremely effective. The small bar chart presents the same information, but allows the user to perceive at a glance how many of the flights are being represented by the current data selection. The bar chart and the table are repeated on several pages of the dashboard and offer consistent contextual information about more detailed and involved views.

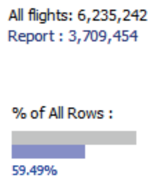
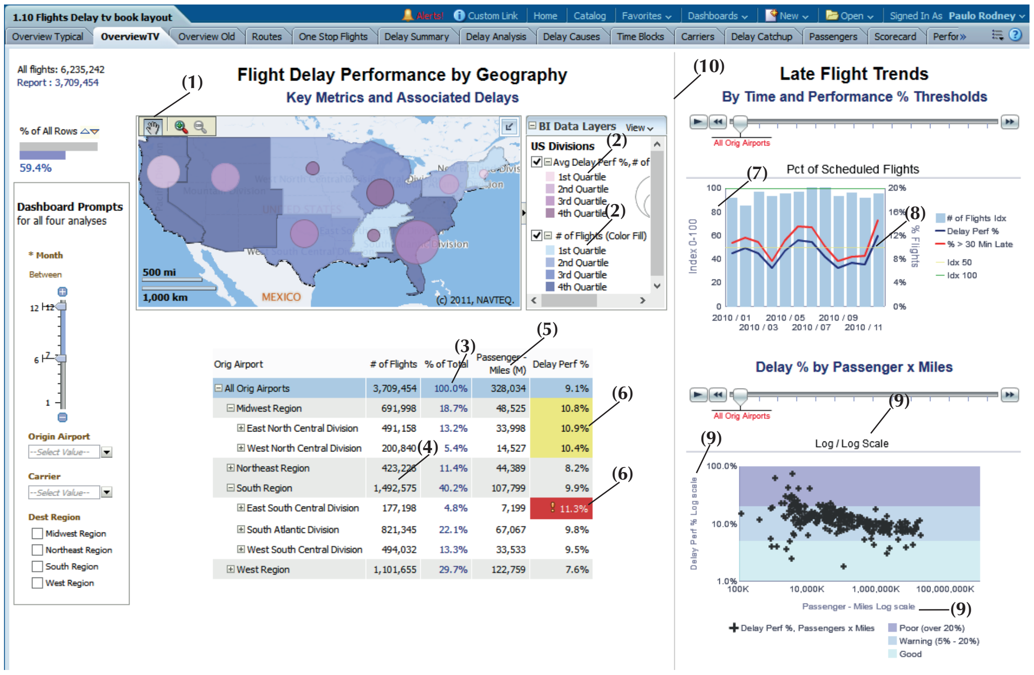

All flights: 6,235,242
Report : 3,709,454

FIGURE 1-2. *Small tables and graphs are big communicators.*

All four featured views are strong visualizations. The pivot table features yellow and red conditionally formatted cells calling attention to the results. The Line and Bar Combo graph utilizes an indexed measure, ensuring a normalized presentation of the number of flights for a hierarchy of airports (displayed as a slider prompt with animation). Map views are always a preferred methodology for displaying data that has a geographical component. Scatter plots (particularly when they employ background data range bars, as this one does) can show the relationship for hundreds or even thousands of individual data points across two dimensions.

Improving a Dashboard from SampleApp

Although we are in deep admiration of the Flights Delay overview dashboard, a few visualization "tweaks" can be made that can strengthen it even more (see Figure 1-3). This discussion will preview some of the topics we delve into later in this book.



Orig Airport	# of Flights	% of Total	Passenger - Miles (M)	Delay Perf %
⊟ All Orig Airports	3,709,454	100.0%	328,034	9.1%
⊟ Midwest Region	691,998	18.7%	48,525	10.8%
⊞ East North Central Division	491,158	13.2%	33,998	10.9%
⊞ West North Central Division	200,840	5.4%	14,527	10.4%
⊞ Northeast Region	423,226	11.4%	44,389	8.2%
⊟ South Region	1,492,575	40.2%	107,799	9.9%
⊞ East South Central Division	177,198	4.8%	7,199	! 11.3%
⊞ South Atlantic Division	821,345	22.1%	67,067	9.8%
⊞ West South Central Division	494,032	13.3%	33,533	9.5%
⊞ West Region	1,101,655	29.7%	122,759	7.6%

FIGURE 1-3. *The revised Flights Delay overview dashboard*

Several changes are immediately apparent. The first is the placement of the map in the upper-left quadrant (1). Maps communicate data faster and more intuitively than any other visualization method. In the revised dashboard, the map is placed in the most visually dominant space on the dashboard and the pivot table is moved below it. Placing maps in the upper-left position and tables toward the bottom (and right) of dashboards is a preferred arrangement for the following reasons:

- Tables are ideal for looking up precise values and act in support of overall conclusions, which are more succinctly communicated in maps and graphs.
- Graph views show patterns and typically have a main point. Graphs better summarize a major insight than do tables and deserve a more prominent placement.

- Tables and pivot tables can often be expanded both horizontally and vertically in OBIEE dashboards and affect other views below and to the left of the table.

Specifically in the map, notice that the color ramps have been changed in the revised dashboard (2). In the original, the color-fill for the region started with a dark blue for the fewest number of flights and progressed to a light blue for the highest number of flights. However, it is more intuitive to use the light blue to reflect fewer flights and the dark blue to reflect more flights. Additionally, the color ramp progression for the variable-shaped circles is changed to a “sequential” color scheme that more accurately reflects progression. (Throughout the book, Dr. Cynthia Brewer’s Colorbrewer2.org website is used to specify preferred color schemes for data visualization.)

The grid lines in the pivot table have been changed to a less intrusive white color (3). (Note that grid lines can often be eliminated completely.) Also, spaces or “padding” was added to the columns (4) to help organize the data and make the table more readable. In addition, the column headers were aligned to the right for numeric columns and to the left for text columns (5). Note that the yellow and red conditional formatting (6) for cells exceeding the threshold value has been retained because the information is important and deserves to be so visually prominent. Indeed, it could be argued that the conditional formatting is more pronounced in the revised dashboard than in the original, despite the less prominent placement, simply because there is less saturated color in the revised dashboard and therefore the yellow and red cells stand out more.

The scale of the Line Bar Combo graph was changed to be exactly 100 points for the indexed value, and the scale is shown (7). Also, a scale marker was added at an index value of 50% for context purposes (8).

Explanatory text was added to the Scatter Plot graph to indicate that a Log/Log scale has been used (9). Although the relationship between the variables is more perceptible with the Log/Log scale, its use should generally be avoided for dashboards intended for a broad, general audience, and it should always be labeled specifically when it is used.

The column structure in the dashboard layout has been changed from two columns (one for prompts and one for visualizations) to three columns (10). The visualizations are organized into Flight Delay Performance by Geography and Late Flight Trends. Aligning the visualizations and separating the columns with a light rule better organizes the dashboard and makes it easier to see the relationships between the visualizations. There are other slight “tweaks” that have been made, and there is no doubt that plenty of reasonable edits remain.

Tradeoffs are always involved in making choices when you’re designing visualizations and dashboards. One of the key decisions that must be made is to determine how much time will be invested in editing and tweaking visualizations and dashboards. The cost in terms of time must be balanced against the return of better understanding and improved consistency in interpretation. This is covered in more depth in Chapter 7. However, many organizations are often too quick to accept the default settings and therefore suffer from having less optimal visualizations for years.

Where the World of Business Intelligence Data Visualization Is Headed

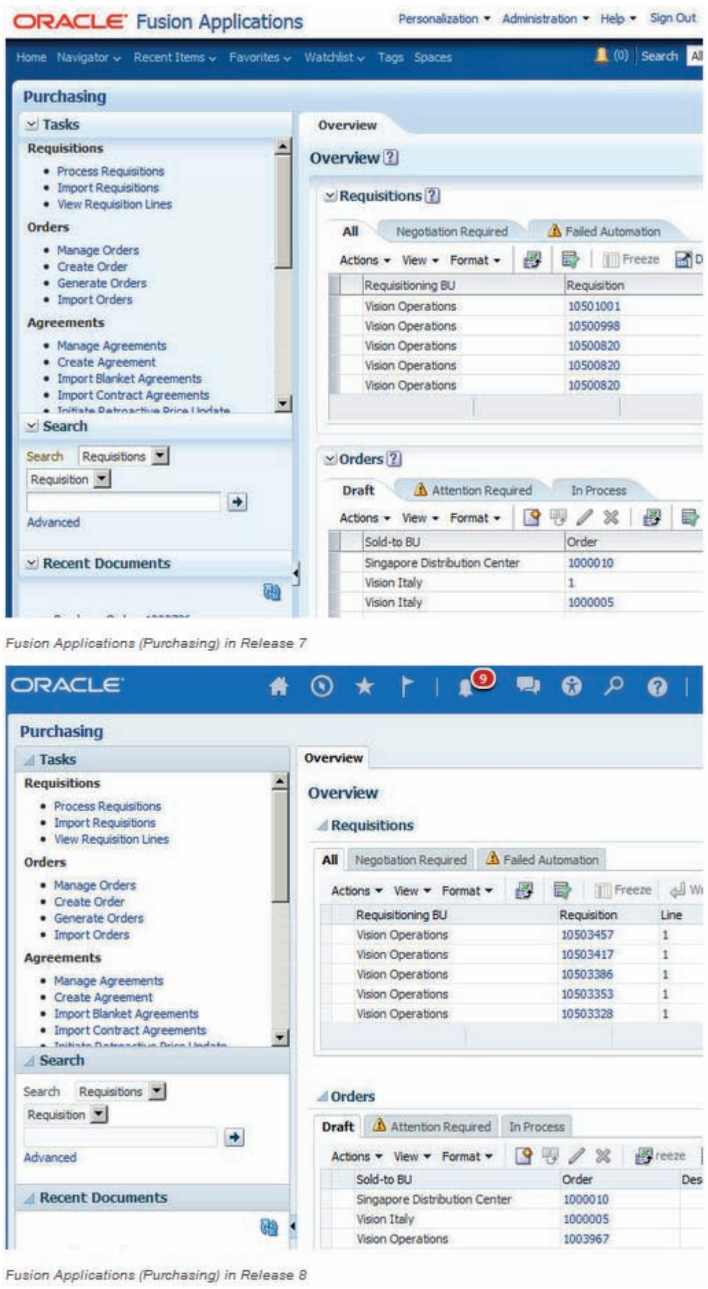
Many of the latest trends for data visualization overall, and for Oracle specifically, mirror the discussion in the earlier part of this chapter. Two trends in particular are the use of a cleaner look and the adaption of a common "grammar of graphics" methodology.

There is a strong movement toward a "cleaner" interface with fewer visual gimmicks and extraneous graphics. As of the writing of this chapter, Oracle's latest "skin" release is called Skyros (named after the Greek island). Here is a quote from the Skyros release document:

> "Skyros…embodies a fresh, lighter weight and cleaner appearance…. Specific design changes includes a focus on current UI visual design trends, such as a flatter, cleaner display. It uses light and/or white color themes, with a few touches of well-placed color. In addition reduced use of gradients and borders replaces background images, enhancing the lighter weight feel."

This fits extremely well with a strategy of deemphasizing the use of gradients, 3-D effects, and bright colors in data visualization graphs. The sparing use of color will make its placement more important and more effective in drawing the eye and highlighting important evidence and database insights. Even Apple Computers, long held in high esteem for their sense of design, is abandoning their preference for the graphic representation of real-life items (called "skeuomorphism") in favor of a flatter, cleaner look. This is likely a long-term trend that will continue to see the emphasis placed on the accurate visual representation of data along with a de-emphasis on visual decoration and embellishment. One might say that as business intelligence systems have grown in size and scale, we are moving toward a "Miesian" aesthetic, where less is more and clean lines and balance are more treasured than garish flourishes and screams for attention. You can see this Skyros style reflected in Figure 1-4.

The second major trend is movement toward a "grammar of graphics" approach to data visualization. We are already seeing some fantastic extensions of OBIEE with JavaScript, D3, R (ggplot2 package), and other "open" scripting languages. The primary paradigm is to define objects and attach attributes to them, which includes



Requisitions
- Process Requisitions
- Import Requisitions
- View Requisition Lines

Orders
- Manage Orders
- Create Order
- Generate Orders
- Import Orders

Agreements
- Manage Agreements
- Create Agreement
- Import Blanket Agreements
- Import Contract Agreements
- Initiate Retroactive Price Update

Requisitioning BU	Requisition
Vision Operations	10501001
Vision Operations	10500998
Vision Operations	10500820
Vision Operations	10500820
Vision Operations	10500820

Sold-to BU	Order
Singapore Distribution Center	1000010
Vision Italy	1
Vision Italy	1000005

Fusion Applications (Purchasing) in Release 7

Requisitions
- Process Requisitions
- Import Requisitions
- View Requisition Lines

Orders
- Manage Orders
- Create Order
- Generate Orders
- Import Orders

Agreements
- Manage Agreements
- Create Agreement
- Import Blanket Agreements
- Import Contract Agreements
- Initiate Retroactive Price Update

Requisitioning BU	Requisition	Line
Vision Operations	10503457	1
Vision Operations	10503417	1
Vision Operations	10503386	1
Vision Operations	10503353	1
Vision Operations	10503328	1

Sold-to BU	Order	Des
Singapore Distribution Center	1000010	
Vision Italy	1000005	
Vision Operations	1003967	

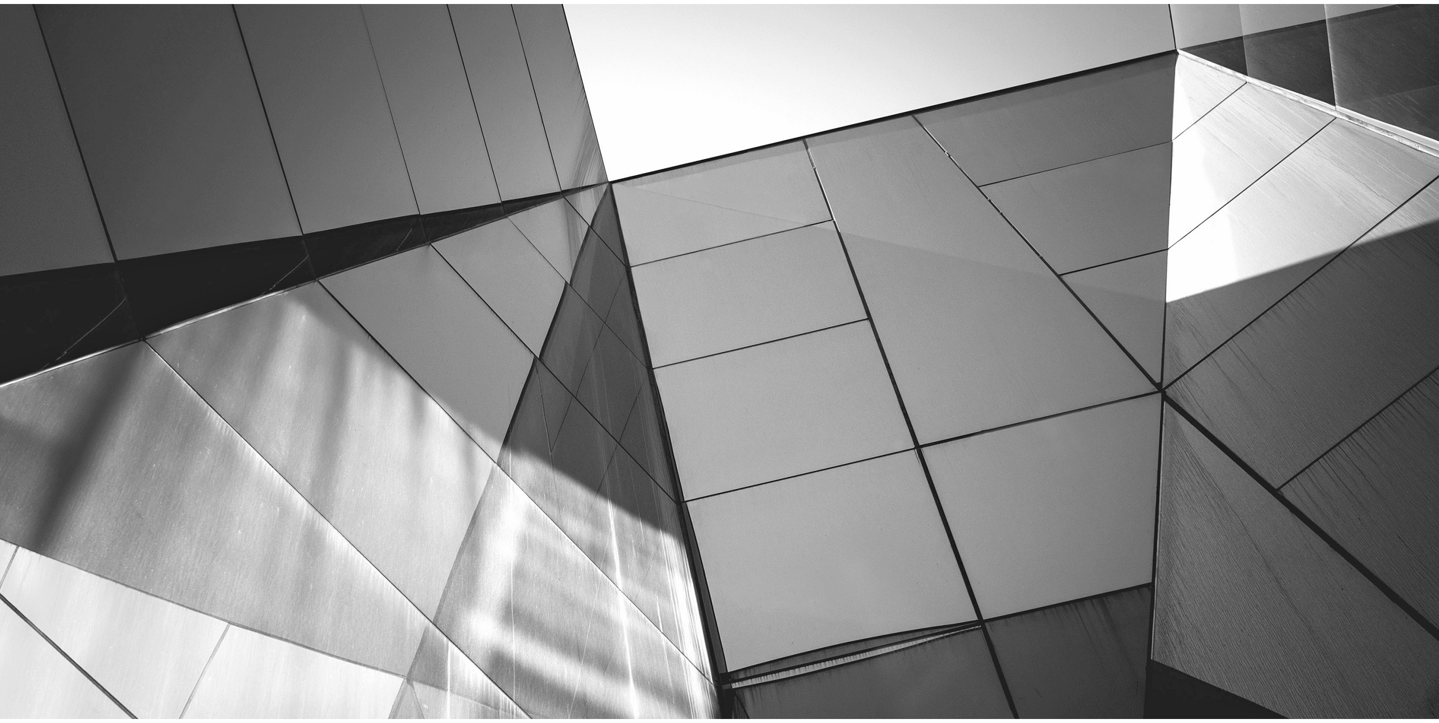
Fusion Applications (Purchasing) in Release 8

FIGURE 1-4. *Screenshot from Oracle's press release announcing their new "Skyros" CSS, which has a cleaner look than the older "FusionFX" style*

the maturation of web interfaces toward HTML5 and away from Flash. This approach deals with graphics more as combinations of components (and structures). You can think of this as "data poetry," where structure and syntax (in short, "composition") all become essential elements of a thoughtful communication. Just as the patterns and "rules" of grammar guide how we formulate sentences and combine and organize them to form larger works, the patterns and rules of graphics guide the formulation of graphs and visualizations. This is addressed more in Chapter 5. A finer integration of "grammar of graphics" style methods can be anticipated in future releases of OBIEE.

Summary

Editing and improving business intelligence visualizations and dashboards takes a certain amount of time and effort. We should be guided not by "taste" or opinion, but rather by understanding the fundamentals of human visual perception and cognition. Our job is to present data accurately and clearly. We must understand that visualizations, which are presented as communications to broad audiences to explain certain business situations, are different from exploratory dashboards, which are designed to reveal previously unknown results to an individual. There is a great emphasis in many data visualization circles on "discovery," and several parts of this book are dedicated to this subject. However, there is also a need to leverage the power of business intelligence systems and dashboards to communicate a shared, coherent understanding of business information across large organizations—that is, to explain organizational position and performance. Much of this book deals with the strong need to understand the implications of design choices for queries, views, and dashboards as they relate to communicating to a large, diverse audience.



CHAPTER 1

Introducing Database Design and Oracle SQL Developer Data Modeler

Database design is the process of producing detailed entity-relationship (ER) diagrams and data flow diagrams (DFDs) in order to produce the data definition language (DDL) scripts that will create the objects needed for the database. Database design consists of requirements analysis, conceptual design, logical design, physical design, and, depending on who you ask, transaction design. (This book will not discuss transaction design.) The process is incremental and iterative, meaning all these phases will be done repeatedly. The backbones of database design are logic theory and relational theory.

Database design is all about the data, namely, how to save the data and how to retrieve it. Data integrity and data quality should always be high priorities when designing a database, and you must consider future needs as well. Even though an application user interface might change every five to ten years, the database behind it must continue to perform well for years to come.

The process of database design is changing as application development processes are getting more agile and iterative. Management demands fast results, so IT projects must be completed faster than ever before. Database designers often do not have time to analyze everything well before starting to design, and sometimes systems are launched into production to be completed later in increments, without having the analysis completed. In fact, sometimes databases are created with no time spent on design and with no thought to the principles of relational theory. Even with the world seemingly getting faster every day, when designing a database, you need to know the full picture of what the database is for. That is what makes database design difficult. The only way to survive is to use a tool that meets all of today's needs, helping you create databases quickly but with the "big picture" in mind. Without a tool, you cannot be as agile as needed.

Though I've mentioned application development processes, I want to be clear that database design is not the same as application design. The database should not be designed as a side product of an application design. When trying to save time and money, people think they will design only either the ER model or the Unified Modeling Language (UML) model and then generate the other. Although it is good they realize they need two models (one for the database and the other for the application), it is not just a question of which notation to choose; the perspectives are very different and so are the goals. For example, let's look at code tables versus code files. For the application designer, it might be easier to have all the lookup information in files, but the database designer definitely will want them in tables. Why? The database person is also in charge of the data integrity, which cannot be controlled if some of the important data is in files somewhere out of the reach of the database.

I often hear people arguing about which is better, ER or UML. To me this question is irrelevant. ER diagrams are for designing databases, and UML is for designing user interfaces. If you try to design a database with UML, you can get easily distracted and want to start designing the user interface. My recommendation is that while the database designer designs the database, the application designer designs the application in cooperation with the database designer. And before the database designer moves to the logical design, the database designer and the application designer should sit down and compare their models to be sure that they really have all the requirements implemented in both the designs. There might be information in the UML model that the ER model does not have or should not have. For instance, in the UML model, there might be an attribute named AGE, but in the data model (ER), there might be an attribute called DATE OF BIRTH. There can also be some technical attributes in the data model that do not need to be in the UML model. For instance, every table might have the columns Creator, Created_Date, Modifier, and Modified_Date. The two models (UML and ER) do not have to be the same and actually rarely are, but creating and maintaining both models will guarantee a better result. But this is true only if you have a tool for both purposes; if this work is done without a tool, the dual processes take too much time and money. You want to use a tool to create designs in cooperation and take advantage of everybody's special skills and knowledge.

NOTE
The UML model and the ER model do not need to be the same and rarely are.

The importance of database design increases on agile projects. In that case, the process involves not just designing but also finding the right questions to ask and having pictures (ER models and DFDs) to use when talking to end users. You need as much information as possible from end users and business owners. You need to understand the big picture, lest you get a database totally different than you wanted. It's as simple as thinking before doing.

It's important that you understand the main concepts (*entities*) of the database and their relationships correctly because it is easy to add entities and attributes later, but it is not easy to divide them later or correct the relationships modeled wrongly. Always design the database for the right purpose and model only what is needed, starting with the most difficult task.

In Figure 1-1, you can see my version of agile database design. It starts with requirements analysis and finding the main concepts and their relationships. Next you try to model the whole conceptual model as well as you can. Then you design the conceptual model for iteration *n*, making it as detailed as possible; continue to the logical design of iteration *n*; and finally move to physical design and creating the database objects with the DDL scripts. Then you perform the whole round again for iteration *n*+1, and so on. The process is the same as it is in other projects; the only difference in an agile project is that you move from phase to phase faster, and you design in pieces, rather than as a whole.

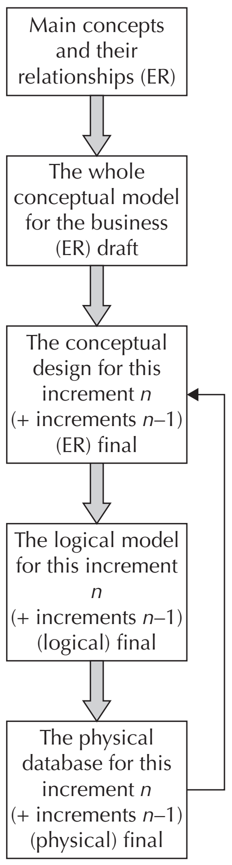

Main concepts and their relationships (ER)
The whole conceptual model for the business (ER) draft
The conceptual design for this increment *n* (+ increments *n*–1) (ER) final
The logical model for this increment *n* (+ increments *n*–1) (logical) final
The physical database for this increment *n* (+ increments *n*–1) (physical) final

FIGURE 1-1. *Agile database design process*

If a database will hold valuable data, the database must be designed by someone who understands how the database works and knows how the data should be modeled. When designing the database, you may need all different types of subject-matter experts to give you the information you need to make decisions about the design. Additionally, if your deadline is tight, you need even more information to be sure you are making the right decisions; you don't want to have to change everything later. Prioritization is important for everybody (even the end users). Nothing is more stressful than too much work with too little time to do it. Database design means teamwork, and that's why you need a tool to do database design right in today's environment.

When selecting the tool for database design, you'll want one that supports these features: ER notation, an automatic transformation process from the conceptual design to the logical design, the ability to work in a multiuser environment, version control, reporting capabilities, scripts for generating the database objects automatically (preferably adjustable), and strong documentation tools. It would be a bonus if the tool also has support for the standardization of naming, processes, and design rules; the ability to alter scripts for changing the database to be like the design; and the ability to compare designs to each other and to compare a design to a database.

What Is Oracle SQL Developer Data Modeler?

Oracle SQL Developer Data Modeler (referred to as Data Modeler in this book) is a free tool for designing and documenting databases and data architecture. It supports not only Oracle databases but also DB2 and Microsoft SQL Server databases and, at a certain level, any standards-based database that has a Java Database Connectivity (JDBC) connector. Data Modeler supports all the steps in database design and includes easy forward and backward engineering. After you have designed your database and have a physical model for it, you can export the scripts to create the database objects. Data Modeler also supports different kinds of compares and multidimensional models. Data Modeler helps you keep your databases documented and enables you to be agile. The tool is available as a stand-alone product, but it is also integrated into Oracle SQL Developer, so you can decide which way is the best for you to use the tool. Installing the tool is simple, and support is provided by Oracle if you have a database support contract.

Data Modeler offers the following features for database designers:

- **Database design tools** A collection of metadata about a database is called a *design* in Data Modeler. A design consists of the logical models, multidimensional models, relational models, domains, data type models, process models, business information, and change requests, as well as all the objects those models need. Every object (entity, table, diagram, and so on) is a single Extensible Markup Language (XML) file in a hierarchy that the tool creates automatically. The design itself is saved with the extension .dmd, and the .dmd file contains pointers to individual XML files.
- **Customization** You can tweak Data Modeler to your liking. In Preferences, you can, for instance, define where to keep your working copy of designs.
- **Version control** Data Modeler is integrated into a version control tool called Subversion. This integration allows you to have multiple users

changing the model at the same time. It also gives you version control functionalities. When working with version control, the latest official version of your design is always on version control, and the one you are working with is in your local saved working copy directory.

- **Documenting existing databases** You can import designs to Data Modeler from existing databases, from other designing tools (for example, Oracle Designer or ERwin), or from DDL scripts.
- **Reporting capabilities** Data Modeler has built-in reporting functionalities, but you can also create your own reports and templates and use the Search functionality as a base for a report. It is also possible to use a reporting repository if you want to have reports across all your designs and use SQL to query that information. You can also print the design layouts.
- **Documentation tools, improving quality and efficency** Data Modeler helps you standardize the design and data documentation in your company. You can use naming standards, domains, glossaries, and design rules to achieve better quality in your database design. You can also compare models and designs to each other, and you can compare a design to a database. Different compares, transformations, and notations will give you a more cost-efficient working environment with better quality.

Designing Databases with Oracle SQL Developer Data Modeler

The database design process when using Data Modeler starts with designing a logical model. In the logical model, you define entities, attributes, and relationships. The next step is to create a relational model based on the logical model. You do this simply by clicking the Engineer To Relational Model icon. When you are done with the relational model, it is time to create the physical model. You do this simply by right-clicking Physical Model in the Browser pane and selecting New. When creating a physical model, you must know what product you will use for your database (Oracle, SQL Server, or DB2) as well as its version. All the properties for the physical model depend on the chosen technology. After you have created a physical model, you should define the properties for the physical objects. After you've done that, you are ready to generate the DDLs (which are the SQL scripts for creating your database objects). You can create DDLs by selecting File | Export | DDL File. Then just run these DDLs on your database to create the objects. And all this can be done in a multiuser environment and while using version control.

You can also use Data Modeler to document existing databases (Oracle, SQL Server, DB2). You can reverse engineer the documentation from a data dictionary or existing DDLs, or you can import it from another design tool (Oracle Designer, ERwin, or a VAR file). Or, you can combine all these techniques, for example, by bringing some of the descriptions from another design tool and adding it to the physical information from the data dictionary. You can find these features by selecting File | Import.

Since an important part of database design is reporting, you might want to use Data Modeler to create your own templates and create reports based on those templates. You can also create a reporting repository; in addition to the templates, you can use SQL to query the information from there. You can also print the diagrams or use the powerful search functionality to search the information in the report.

It is also important to be able to document all the information related to the database in just one place. In Data Modeler you can document all the information needed for the database design as well as change requests, business information, and much more.

Summary

Going through the database design process is vital if you are storing important data in your database. Database design is the process of producing detailed entity-relationship diagrams and data flow diagrams to produce the DDL scripts for creating the objects for the database. Database design consists of requirements analysis, conceptual design, logical design, physical design, and, depending on who you ask, transaction design. To be able to design a database, especially in an agile system, you need Oracle SQL Developer Data Modeler. It is a free tool that supports all the needs of database designers plus some extra.