CERTIFICATION OBJECTIVES

1.01 Position the Server Technologies
1.02 Understand Relational Structures
1.03 Summarize the SQL Language
1.04 Use the Client Tools
1.05 Create the Demonstration Schemas
   ✓ Two-Minute Drill
   Q&A Self Test
The content of this chapter is not directly tested by the Oracle certification examination, but it is vital in understanding the purpose of SQL. This is considered prerequisite knowledge that every student should have, beginning with an appreciation of how the Oracle server technologies fit together and the positioning of each product.

The Oracle server technologies product set is more than a database. There are also the Oracle WebLogic Server and the Oracle Enterprise Manager. Taken together, these are the server technologies that make up the Oracle Cloud. Cloud computing is an emerging environment for managing the complete IT environment and providing services to users on demand.

Databases fundamentally provide the infrastructure for the organization, storage, and retrieval of data in an efficient manner. Oracle Database 12c has evolved from a Relational Database Management System (RDBMS) to an Object RDBMS supporting the organization of virtually any type of information with no practical limit on the volume of data that may be stored. The vast data volumes generated by Amazon.com, the Large Hadron Collider (LHC) at CERN in Europe, and many financial institutions and governments are organized and managed in Oracle databases. Although Oracle databases have features addressing demands for scalability, high availability, and superb performance, this guide will focus on the data organization problem. Structured Query Language (SQL, pronounced “sequel”) is an international standard for managing data stored in relational databases. Oracle Database 12c offers an implementation of SQL that is generally compliant with the current standard, which is Core SQL:2011. Full details of the compliancy are in Appendix C of the SQL Language Reference, which is part of the Oracle Database Documentation Library. As a rule, compliancy can be assumed.

Throughout this book, two tools are used extensively for exercises: SQL*Plus and SQL Developer. These are tools that developers use every day in their work. The exercises and many of the examples are based on two demonstration sets of data, known as the HR and OE schemas. There are instructions on how to launch the tools and create the demonstration schemas, though you may need assistance from your local database administrator to get started.

This chapter consists of summarized descriptions of the Oracle server technologies, the SQL language, the client tools, and the demonstration schemas. Several real-world data organization scenarios are considered while discussing the concepts behind the relational paradigm and normalizing of data into relational structures.
CERTIFICATION OBJECTIVE 1.01

Position the Server Technologies

There is a family of products that makes up the Oracle server technologies:

- The Oracle database
- The Oracle WebLogic Server
- The Oracle Enterprise Manager
- Various application development tools and languages

These products each have a position in the Oracle product set. The database is the repository for data and the engine that manages access to it. The Oracle WebLogic Server runs software that generates the web user interfaces that submit calls for data retrieval and modification to the database for execution. The Oracle Enterprise Manager is a comprehensive administration tool for monitoring, managing, and tuning the Oracle processes and also (through plug-ins) third-party products. Lastly, there are tools and languages for developing applications; either applications that run on end users’ machines in the client-server model or applications that run centrally on application servers.

The combination of the server technologies and the development tools make up a platform for application development and delivery that enables the cloud. The cloud is an approach to the delivery of IT services that maximizes the cost efficiency of the whole environment by delivering computing power from a pool of available resources to wherever it is needed, on demand.

The Oracle Server Architecture

An Oracle database is a set of files on disk. It exists until these files are deliberately deleted. There are no practical limits to the size and number of these files, and therefore no practical limits to the size of a database. Access to the database is through the Oracle instance. The instance is a set of processes and memory structures: it exists on the CPU(s) and in the memory of the server node, and this existence is temporary. An instance can be started and stopped. Users of the database establish sessions against the instance, and the instance then manages all access to the database. It is absolutely impossible in the Oracle environment for any user to have direct contact with the database. An Oracle instance with an Oracle database makes up an Oracle server.
The processing model implemented by the Oracle server is that of client-server processing, often referred to as two-tier. In the client-server model, the generation of the user interface and much of the application logic is separated from the management of the data. For an application developed using SQL (as all relational database applications will be), this means that the client tier generates the SQL commands, and the server tier executes them. This is the basic client-server split, with (as a general rule) a local area network between the two sides. The network communications protocol used between the user process and the server process is Oracle’s proprietary protocol, Oracle Net.

The client tier consists of two components: the users and the user processes. The server tier has three components: the server processes that execute the SQL, the instance, and the database itself. Each user interacts with a user process. Each user process interacts with a server process, usually across a local area network. The server processes interact with the instance, and the instance with the database.

Figure 1-1 shows this relationship diagrammatically. A session is a user process in communication with a server process. There will usually be one user process per user and one server process per user process. The user and server processes that make up sessions are launched on demand by users and terminated when no longer required; this is the log-on and log-off cycle. The instance processes and memory structures are launched by the database administrator and persist until the administrator deliberately terminates them; this is the database startup and shutdown cycle.

The user process can be any client-side software that is capable of connecting to an Oracle server process. Throughout this book, two user processes will be used extensively: SQL*Plus and SQL Developer. These are simple processes provided by Oracle for establishing sessions against an Oracle server and issuing ad hoc SQL. A widely used alternative is TOAD (the Tool for Application Developers) from Quest Software, though this is licensed software. End-user applications will need to be written with something more sophisticated than these tools, something capable of...
managing windows, menus, proper onscreen dialogs, and so on. Such an application could be written with the Oracle Development Tools, with Microsoft Access linked to the Oracle ODBC drivers, with any third-generation language (such as C or Java) for which Oracle has provided a library of function calls that will let it interact with the server, or with any number of Oracle-compatible third-party tools. What the user process actually is does not matter to the Oracle server at all. When an end user fills in a form and clicks a Submit button, the user process will be generating an INSERT statement (detailed in Chapter 10) and sending it to a server process for execution against the instance and the database. As far as the server is concerned, the INSERT statement might just as well have been typed into SQL*Plus as what is known as ad hoc SQL.

Never forget that all communication with an Oracle server follows this client-server model. The separation of user code from server code dates back to the earliest releases of the database and is unavoidable. Even if the user process is running on the same machine as the server (as is the case if, for example, one is running a database on one's own laptop for development or training purposes), the client-server split is still enforced. Applications running in an application server environment (described in the next section) also follow the client-server model for their database access.

The simplest form of the database server is one instance connected to one database, but in a more complex environment one database can be opened by many instances concurrently. This is known as a RAC (Real Application Cluster). RAC can bring many potential benefits, which may include scalability, performance, and zero downtime. The ability to dynamically add further instances running on supplementary nodes to a database is a major part of the database’s contribution to the cloud.

**The Oracle WebLogic Server**

With the emergence of the Web as the standard communications platform for delivering applications to end users has come the need for application servers. An application server replaces the client-side software traditionally installed on end-user terminals; it runs applications centrally, presenting them to users in windows displayed locally in web browsers. The applications make use of data stored in one or more database servers.

The Oracle WebLogic Server is a platform for developing, deploying, and managing web applications. A web application can be defined as any application with which users communicate with HTTP. Web applications usually run in at least three tiers: a database tier manages access to the data, the client tier (often implemented as a web
Chapter 1: Relational Database Design Using Oracle

Web applications can be developed with a number of technologies, among which Java is predominant. Applications written in Java should conform to the Java EE (Java Enterprise Edition) standard, which defines how such applications should be packaged and deployed. JEE and related standards are controlled by Oracle and accepted by virtually all software developers. Oracle WebLogic Server is a JEE-compliant application server. Oracle's implementation of the standards allows for automatic load balancing and fault tolerance across multiple application servers on multiple machines through JEE clustering. Clustering virtualizes the provision of the application service; users ask for an application that might be available from a number of locations, and the cluster works out from where any one session or request can best be serviced. If one location fails, others will take up the load, and more resources can be made available to an application as necessary. The ability to separate the request for a service from the location of its provision and to add or remove JEE servers from a cluster dynamically is a major part of the Oracle WebLogic Server's contribution to the cloud.

It is important to note that Oracle's commitment to international standards is very strong. Applications running in the Oracle WebLogic Server environment can connect to any database for which there are Java-compliant drivers; it is not necessary to use an Oracle database. Applications developed with the Oracle WebLogic Server toolkits can be deployed to any third-party JEE-compliant application server.

The simplest processing model of web applications is three-tier: a client tier that manages the user interface, a middle tier that generates the interface and issues SQL statements to the data tier, and a data tier that manages the data itself. In the Oracle environment, the client tier will be a browser (such as Mozilla Firefox or Windows Internet Explorer) that handles local window management, controls the keyboard, and tracks mouse movements. The middle tier will be an Oracle WebLogic Server running the software (probably written in Java) that generates the windows sent to the client tier for display and the SQL statements sent to the data tier for execution. The data tier will be an Oracle server: an instance and a database. In this three-tier environment, there are two types of sessions: end-user sessions from the client tier to the middle tier, and database sessions from the middle tier to the data tier. The end-user sessions will be established with HTTP. The database sessions are client-server sessions consisting of a user process and a server process, as described in the previous section.
It is possible for an application to use a one-for-one mapping of end-user session to database session: each user, from their browser, will establish a session against the application server, and the application server will then establish a session against the database server on the user’s behalf. However, this model has been proven to be very inefficient when compared to the connection pooling model. With connection pooling, the application server establishes a relatively small number of persistent database sessions and makes them available on demand (queuing requests if necessary) to a relatively large number of end-user sessions against the application server. Figure 1-2 illustrates the three-tier architecture using connection pooling.

From the point of view of the database, it makes no difference whether a SQL statement comes from a client-side process such as SQL*Plus or Microsoft Access or from a pooled session to an application server. In the former case, the user process all happens on one machine; in the latter, the user process has been divided into two tiers: an application tier that generates the user interface and a client tier that displays it.

**Oracle Enterprise Manager**

The increasing size and complexity of IT installations makes management a challenging task. This is hardly surprising: no one ever said that managing a powerful environment should necessarily be simple. However, management tools can make the task easier and the administration staff more productive.
Chapter 1: Relational Database Design Using Oracle

Oracle Enterprise Manager comes in three forms:

- Database Express
- Fusion Middleware Control
- Cloud Control

Oracle Enterprise Manager Database Express is a graphical tool for managing one database, which may be a RAC clustered database. It consists of a Java process running on the database server machine. Administrators connect to Database Express from a browser, and Database Express then connects to the database server. Database Express has facilities for real-time management, performance monitoring, and running scheduled jobs. Oracle Enterprise Manager Fusion Middleware Control is a graphical tool for managing a Fusion Middleware deployment. These deployments typically include Oracle WebLogic Server, an industry-leading application server which provides the container Java virtual machines (JVMs) that host Oracle or custom Java applications.

Oracle Enterprise Manager Cloud Control globalizes the management environment. A management repository (residing in an Oracle database) and one or more management servers manage the complete environment: all the databases and application servers, wherever they may be. Cloud Control can also manage the nodes, or machines, on which the servers run, as well as (through plug-ins) a wide range of third-party products. Each managed node runs an agent process, which is responsible for monitoring the managed target on the node: executing jobs against them and reporting status, activity levels, and alert conditions back to the management server(s).

Cloud Control provides a holistic view of the environment and, if well configured, makes administration staff far more productive than they are without it. It becomes possible for one administrator to effectively manage hundreds of targets.

Cloud Computing

Critical to the concept of cloud computing is service virtualization. This means that at all levels there is a layer of abstraction between what is requested and what is provided. End users ask for an application service and let the cloud work out which clustered JEE application server can best provide it. Application servers ask for a database service and let the cloud work out from which RAC node the data can best be served. Within the cloud there is a mapping of possible services to available service providers, and there are algorithms for assigning the workload and
resources appropriately. The result is that end users have neither the need nor the capacity to know from where their computing resources are actually being provided. The analogy often drawn is with delivery of domestic electricity: it is supplied on demand, and the home owner has no way of telling which power station is currently supplying him.

The cloud is not exclusive to Oracle. At the physical level, some operating system and hardware vendors are providing cloud-like capabilities. These include the ability to partition servers into virtual machines and dynamically add or remove CPU(s) and RAM from the virtual machines according to demand. This is conceptually similar to Oracle’s approach of dynamically assigning application server and database server resources to logical services. There is no reason why the two approaches cannot be combined. Both are working toward the same goal and can work together. The result should be an environment where adequate resources are always available on demand, without facing the issues of excess capacity at some times and under-performance at others. It should also be possible to design a cloud environment with no single point of failure, thus achieving the goal of 100 percent uptime that is being demanded by many users.

The SQL application developer need not know how the cloud has been implemented. The SQL will be invoked from an application server and executed by an instance against a database: the cloud will take care of making sure that at any moment pools of application servers and instances sized appropriately for the current workload are available.

EXERCISE 1-1
Investigate Your Database and Application Environment

This is a paper-based exercise, with no specific solution.

Attempt to identify the user processes, application servers, and database servers used in your environment. Try to work out where the SQL is being generated and where it is being executed. Bear in mind that usually the user processes used by end users will be graphical and will frequently go through application servers; the database administration and development staff will often prefer to use client-server tools that connect to the database server directly.
Development Tools and Languages

The Oracle server technologies include various facilities for developing applications, some existing within the database, others external to it.

Within the database, it is possible to use three languages. The one that is unavoidable, and the subject of this book, is SQL. SQL is used for data access, but it cannot be used for developing complete applications. It has no real facilities for developing user interfaces, and it also lacks the procedural structures needed for manipulating rows individually. The other two languages available within the database fill these gaps. They are PL/SQL and Java, although Java may also be used outside the database. PL/SQL is a third-generation language (3GL) proprietary to Oracle. It has the usual procedural constructs (such as if-then-else and looping) and facilities for user interface design. In the PL/SQL code, one can embed calls to SQL. Thus, a PL/SQL application might use SQL to retrieve one or more rows from the database, then perform various actions based on their content, and then issue more SQL to write rows back to the database. Java offers a similar capability to embed SQL calls within the Java code. This is industry standard technology: any Java programmer should be able write code that will work with an Oracle database (or indeed with any other Java-compliant database).

Other languages are available for developing client-server applications that run externally to the database. The most commonly used are C and Java, but it is possible to use most of the mainstream 3GLs. For all these languages, Oracle Corporation provides OCI (Oracle Call Interface) libraries that let code written in these languages establish sessions against an Oracle database and invoke SQL commands.

Many organizations will not want to use a 3GL to develop database applications. Oracle Corporation provides rapid application development tools such as Oracle Application Express, JDeveloper, ADF, and many other products. These can make programmers far more productive than if they were working with a 3GL. Like the languages, all these application development tools end up doing the same thing: constructing SQL statements that are sent to the database server for execution.

All developers and administrators working in the Oracle environment must know PL/SQL. C and Java are not necessary, unless the project specifically uses them.
Understand Relational Structures

CERTIFICATION OBJECTIVE 1.02

Several real-world data organization scenarios are introduced to discuss the relational paradigm and introduce some practical modeling techniques. Critical to an understanding of SQL is an understanding of the relational paradigm and the ability to normalize data into relational structures. Normalization is the work of systems analysts, as they model business data into a form suitable for storing in relational tables. It is a science that can be studied for years, and there are many schools of thought that have developed their own methods and notations.

Real-World Scenarios

This guide uses several hypothetical scenarios, including the two canned scenarios called HR and OE provided by Oracle and frequently used as the context for exam questions to illustrate various SQL concepts. The following scenarios evolve further as new concepts are discussed.

Car Dealership

Sid runs a car dealership and needs a system to keep track of the cars that she buys and sells. She has noticed business taking a dive and wants to move into the twenty-first century and create a website that advertises available stock. She needs a system to keep records of the cars she has bought and sold and the details of these transactions.

Geological Cores

Core samples of the earth have been collected by your local geological survey agency. To ensure scientific rigor, the developers at GeoCore have determined that the system must track the exact geographical location, the elemental content of the core samples, and the dates of collection.

Order Entry

The Order Entry (OE) scenario provided as a sample by Oracle contains information for a fictitious commercial system that tracks products, customers, and the sales orders that have been placed.
Human Resources

The Human Resources (HR) scenario provided as a sample by Oracle records employees, departments, office locations, and job-related information for a typical HR department.

Although the hypothetical scenarios described above vary in complexity, they share several characteristics, including a potential data growth that may eventually overwhelm a paper-based or spreadsheet-based data organization solution, as well as a requirement for data to be manipulated (inserted, updated, and deleted) and retrieved in an efficient manner. The challenge of producing an efficient data organization design (also known as a data model) may be overcome with both an understanding of how the data being organized is likely to be utilized and a few basic data modeling techniques. The goal is to achieve an optimal balance between data storage and data access which will provide long-term downstream cost-saving benefits.

Data Modeling

Various formalized data modeling approaches are available, such as the Zachman framework and the Rational Unified Process, that ultimately seek to provide a systematic, standards-based approach to representing objects in an enterprise. There are a multitude of notations available to model entities and their relationships. A popular notation adopted by Oracle in its CASE tools (Computer Aided Software Engineering) and more recently in SQL Developer is the crow’s foot notation, which will be discussed below. Other notations, such as Relational Schema notation and UML (Universal Markup Language), are also popular, but you must choose a notation that is comfortable and sensible for you.

Logical modeling is based on conceptualizing objects of interest as entities and their interactions with each other as relationships. There are many approaches to entity-relationship diagrams, each with their benefits and limitations. A brief discussion of entity-relationship diagrams and their notation follows.

Entities and Relations

Many Oracle professionals have adopted a framework that consists of three modeling stages for relational database modeling. A logical model is conceived when high-level constructs called entities, comprising various attributes and their relationships, are typically represented together in a diagram. Entities in logical models are usually depicted as
rectangles with rounded corners, which comprise attributes or identifiers sometimes denoted by an “o” symbol. Attributes that uniquely identify an instance of an entity are designated as primary keys and are sometimes denoted by “#*”. Data typing the attributes may be done at this stage, but it is generally not reflected in the design.

The logical model is then turned into a relational model by translating the entities into relations, commonly referred to as tables. The idea here is that sets of instances of the entities are collectively modeled as a table. The attributes are turned into table columns. Each instance of an entity is reflected as a tuple or row of data, each having values for its different attributes or columns. The number of “rows in the table” is the “cardinality of the tuples.” Usually the attributes that are unique for each row are called unique keys, and typically a unique key is chosen to be the primary key (which is discussed later). The relationships between the entities are often modeled as foreign keys, which will also be explored below.

Relations in relational models are usually depicted as rectangles. At this stage there is typically more detail in terms of data typing for the attributes, and primary and foreign key attributes are also reflected with a “P” and an “F,” respectively, in the relational model. Finally, the relational model is engineered into a physical model by implementing the design in a relational database.

Crow’s foot notation is often used to depict relationships in logical and relational data models. The relationships between the entities can be one of the following and will be explored in the context of the Car Dealership scenario.

- **1:N** One-to-many
- **N:1** Many-to-one
- **1:1** One-to-one
- **M:N** Many-to-many

Consider the scenario of Sid’s car dealership introduced earlier. You could model the likely data as an entity consisting of the following car-related attributes: Make, Model, Engine Capacity, and Color. Information regarding the buying and selling of cars is also required, so you could add Purchase Date, Sold Date, Sellers Name, Sellers SSN (Social Security number), Sellers Company, the same details for the buyer, and finally the Purchase Price and Selling Price, as in Figure 1-3.
Sample transactional data stored in a table based on this entity may look like Figure 1-4, which shows three rows of data in a table comprising fourteen columns called CAR_DEALERSHIP. The commands to create tables and populate them with data will be discussed later in this book. For now, there are several more fundamental important things to notice. Tables store data in rows, also called records. Each data element is found at the intersection of a row and a column, also called a cell. It is fairly intuitive and much like a spreadsheet.

The first two records in the CAR_DEALERSHIP table include the following information:

- A silver Mercedes A160 with a 1600cc engine capacity that belonged to Coda, a private seller with SSN 12345, was bought by Sid with SSN 12346 from Sid’s Cars, for $10,000 on 1 June 2013.
- A silver Mercedes A160 with a 1600cc engine capacity that belonged to Sid, with SSN 12346, from Sid’s Cars, was bought by Wags, with SSN 12347, from Wags Auto, for $12,000 on 1 August 2013.
Notice the repetition of data. Each record contains duplicate information for
the cars being bought or sold and for the customer doing the buying or selling.
Unnecessary duplication of data usually indicates poor design since it is wasteful and
often requires needless maintenance. If this maintenance is not carefully done, this
design allows errors (sometimes referred to as insert update and deletion anomalies) to
creep in and reduces the overall integrity of the data.

Database normalization refers to modeling data using multiple entities with
relationships between them, which may reduce or entirely eliminate data redundancy.
There are many types of normal forms that have been defined theoretically, but
relational database design primarily focuses on the following three:

- First normal form (1NF) deals with the issue of eliminating unnecessary
  repeating groups of data. An example of a repeating group in Figure 1-4
  would be the first four columns on the first two rows where descriptive
  information about the car is repeated. You could define a new Cars entity
  that uniquely identifies a specific car using the Car ID primary key attribute
  as well as the Make, Model, Engine Capacity, and Color attributes. The Car
  ID identifier is then used in the related Transactions entity to avoid repeating
  groups of data.

- Second normal form removes attributes from the entity (1NF) that are not
  dependent on the primary key. In the proposed Cars entity described above,
  the Color attribute is not dependent on a specific car. You could define a
  new Colors entity that uniquely identifies a specific color using the Color
  ID primary key attribute. The Color ID can then be referenced by the Cars
  entity.

- Third normal form removes all interdependent attributes from a 2NF entity.
The buyers and sellers of cars each have a uniquely identifying Social Security
  number (SSN). Their names, however, are interdependent on the SSN
  attribute. You could define a new Customers entity that uniquely identified a
customer using the Customer ID primary key attribute, where interdependent
  information like the customer’s name and company are stored.

There are often several possible normalized models for an application. It
is important to use the most appropriate—if the systems analyst gets this
wrong, the implications can be serious for performance, storage needs, and
development effort.
Note: In the context of performance tuning, it is intentional and acceptable to duplicate data in entities. When data is normalized across multiple entities instantiated as multiple tables that must be joined together, the Oracle server processes need to physically fetch data from multiple tables and join them in memory buffers to produce the required results set. The extra IO required to query or manipulate normalized data sometimes justifies denormalizing data models to reduce disk IO operations and hence increase performance. This is common in Data Warehouse (DWH) and Decision Support Systems (DSS) but is an exception rather than the rule in Online Transaction Processing (OLTP) systems.

Consider the logical data model in Figure 1-5. The car-related data has been modeled as the Cars entity. The customer's (buyers and sellers) information is essentially the same, so they have been modeled as the Customers entity with the
Customer Type attribute to differentiate between Purchasers and Sellers. The sales and purchases are recorded in the Transactions entity, while a lookup entity called Colors keeps track of different colors.

There are several advantages to conceptualizing this design as four interrelated entities. Firstly, the data has been normalized and there is no duplication of data. A practical benefit of multiple entities, each tracking a single construct like Cars, Customers, Colors, and even Transactions, is the ease of data maintenance. New colors can be added, each with a unique code, and as new cars are purchased, these colors, defined and maintained in one place, can be used to describe multiple cars with the same color. You could improve the sophistication of this model by defining entities for Tires, Security Systems, Tracking Devices, or Audio Visual add-ons. You could equally enhance the details collected for each car, such as VIN and engine numbers; or for each customer, such as address and bank details; but this hypothetical scenario serves to illustrate several concepts and obviously cannot be used in a production application scenario without further enhancements.

**Primary Keys**

Each entity in Figure 1-5 has a primary key attribute that uniquely identifies a tuple or row of data denoted by a #* adjacent to the attribute name. Each value of the Car ID primary key is unique in the entity. Multiple rows cannot share the same primary key value. Similarly, Color ID uniquely identifies each row in the Colors entity, as do Customer ID and Transaction ID in the Customers and Transactions entities, respectively.

**Relationships**

The lines in Figure 1-5 linking the various entities are known as relations. The crow’s foot notation expresses the cardinality of the relationships between the entities—one-to-one, one-to-many, many-to-one, and many-to-many. The crow’s foot notation explicitly illustrates the entity with the many side of the relationship with multiple “feet,” while the entity on the one side has one foot. Attributes in a one-to-one relationship are identical, while many-to-many relationships indicate that multiple tuples in entity A have the same attribute values as many tuples in entity B. Both one-to-one and many-to-many relationships are not very common and sometimes point to flaws in the relational model. One-to-many and many-to-one relationships occur frequently when modeling relational entities. They relate attributes in two entities in a master-detail relationship. From the point of view of the relationship between the Cars and Colors entities (the order is significant), for
example, many records in the Cars entity will be one Color. Many cars could have
the same single Color ID attribute indicating they are the same color. The Colors
entity is the master or lookup entity, while the Cars entity is the detail entity in this
relationship. From the point of view of the relationship between Colors and Cars,
one Color can be associated with many Cars. So it is just a matter of perspective
whether a relationship is one-to-many or many-to-one; it all depends on which
direction of the relationship you consider. The other relationships indicated by the
crow’s feet show that a single car can be bought and sold multiple times, hence the
one-to-many relationship between the Cars and Transactions entities and finally that
one customer can perform many transactions (like buying and selling many cars).

Referential Integrity and Foreign Keys
These relationships introduce the concept of referential integrity, which ensures
data consistency and integrity by guaranteeing that an attribute (say attribute A)
belonging to the entity on the one side of the relationship must be unique, while
the attribute (say attribute B) on the entity on the many side must have a value
that is in the set of unique values described by attribute A. Attribute B is called
a foreign key since it has a referential dependency on attribute A. Consider the
Colors-Cars relationship based on the Color ID attribute. Referential integrity
ensures that the Color ID attribute in each tuple in the Cars entity must have a
value that is identical to exactly one instance of the Color ID attribute in the Colors
entity. This guarantee is central to relational modeling since the joining of the Cars
and Colors entities on the Color ID attribute allows the Colors.Color (this is dot
notation) attribute to be matched with a related tuple in the Cars entity. The Color
ID attribute in the Cars entity is the foreign key that is related to the unique key,
which is the Color ID attribute in the Colors entity, where it also happens to be the
primary key. It is very common that foreign keys in an entity are based on primary
keys in a related entity, but this is not the rule.

Note: Foreign keys in an entity are based on unique keys in a related entity,
but those unique keys do not have to be the primary key; they just have to
be unique.

The logical model in Figure 1-5 would typically evolve into a relational model
typically with more data-typing details and clearer primary and foreign keys, as in
Figure 1-6.
The relational model could be engineered into a physical model where actual
tables and other database constructs (discussed later in this chapter) are created.
The sample data in Figure 1-4 transferred into the physical model built from the relational model above would produce four datasets, as in Figure 1-7.

The first two rows of data in the Transactions dataset can be interpreted as follows:

- A transaction with TX ID 100 describes the purchase of a car with Car ID 1 by Sid's dealership from a customer with Customer ID 2 for $10,000 on 1 June 2013. You lookup Customer ID 2 and see it was a sale from Coda, a private seller with SSN 12345. You lookup Car ID 1 and determine that it was a 2001-A160 Mercedes with Color ID 1, which you further resolve to be Silver.
- A transaction with TX ID 101 describes the sale of Car ID 1 to Customer ID 4, which you resolve to be a dealer called Wags from Wags Auto with SSN 12347, for $12,000 on 1 August 2013.
Based on the earlier descriptions offered in a single-entity design, nothing has been lost by organizing the sample data across the four-entity design. However, much has been gained. There is no duplication of data. There is a clarity and elegance that will facilitate ease of maintenance of this data as new cars are bought and sold and as new customers transact with Sid's dealership.

EXERCISE 1-2

Design an Entity-Relationship Diagram for the Geological Cores Scenario

In this exercise, you will gain familiarity with basic relational modeling by completing an entity-relationship diagram for the Geological Core scenario introduced earlier. To recap: Core samples of the earth have been collected by
your local geological agency. To ensure scientific rigor, the developers at GeoCore have determined that the system must track the exact geographical location, the elemental contents of the core samples, and the date of collection.

1. The GeoCore developers have proposed four entities with their respective attributes as per the following table. Draw a diagram with these entities reflecting the primary keys.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>Atomic Number (Primary Key)</td>
</tr>
<tr>
<td></td>
<td>Element Description</td>
</tr>
<tr>
<td></td>
<td>Symbol</td>
</tr>
<tr>
<td></td>
<td>Atomic Mass</td>
</tr>
<tr>
<td>Contents</td>
<td>Content ID (Primary Key)</td>
</tr>
<tr>
<td></td>
<td>Atomic Number</td>
</tr>
<tr>
<td></td>
<td>Quantity</td>
</tr>
<tr>
<td>Depth</td>
<td>Depth ID (Primary Key)</td>
</tr>
<tr>
<td></td>
<td>Start Depth</td>
</tr>
<tr>
<td></td>
<td>End Depth</td>
</tr>
<tr>
<td>Cores</td>
<td>Core ID (Primary Key)</td>
</tr>
<tr>
<td></td>
<td>Depth ID</td>
</tr>
<tr>
<td></td>
<td>Collection Date</td>
</tr>
<tr>
<td></td>
<td>Longitude</td>
</tr>
<tr>
<td></td>
<td>Latitude</td>
</tr>
<tr>
<td></td>
<td>Content ID</td>
</tr>
</tbody>
</table>

2. The Elements and Contents entities share a many-to-one relationship through the Atomic Number attribute. The Cores entity shares a many-to-one relationship with the Depth entity through the Depth ID attribute and a many-to-one relationship with the Contents entity through the Content ID attribute. Your task is to update the entity diagram to reflect these
relationships. Your completed Entity-Relationship diagram should closely resemble this illustration:

![Entity-Relationship diagram]

## Rows and Tables

The relational paradigm models data as two-dimensional tables. A table consists of a number of rows, each consisting of a set of columns. Within a table, all the rows have the same column structure, though it is possible that in some rows some columns may have nothing in them. An example of a table would be a list of one’s employees, each employee being represented by one row. The columns might be employee number, name, and a code for the department in which the employee works. Any employees not currently assigned to a department would have that column blank. Another table could represent the departments: one row per department, with columns for the department’s code and the department’s name.
Relational tables conform to certain rules that constrain and define the data. At the column level, each column must be of a certain data type, such as numeric, date-time, or character. The character data type is the most general, in that it can accept any type of data. At the row level, usually each row must have some uniquely identifying characteristic: this could be the value of one column, such as the employee number and department number in the preceding examples, which cannot be repeated in different rows. There may also be rules that define links between the tables, such as a rule that every employee must be assigned a department code that can be matched to a row in the departments table. Tables 1-1 through 1-4 are examples of the tabulated data definitions (a subset of data and structures from the sample schema known as SCOTT provided by Oracle):

**Table 1-1**
The DEPT Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPTNO</td>
<td>Department number</td>
<td>Numeric</td>
<td>2</td>
</tr>
<tr>
<td>DNAME</td>
<td>Department name</td>
<td>Character</td>
<td>14</td>
</tr>
</tbody>
</table>

**Table 1-2**
The EMP Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPNO</td>
<td>Employee number</td>
<td>Numeric</td>
<td>4</td>
</tr>
<tr>
<td>ENAME</td>
<td>Employee name</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>Department number</td>
<td>Numeric</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 1-3**
Row Data from the DEPT Table

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
</tr>
<tr>
<td>20</td>
<td>RESEARCH</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
</tr>
<tr>
<td>40</td>
<td>OPERATIONS</td>
</tr>
</tbody>
</table>
Looking at the layout of the DEPT and EMP Tables 1-1 and 1-2, the two-dimensional structure is clear. Each row is of fixed length, each column is of fixed length (padded with spaces when necessary), and the rows are delimited with a new line. Table 1-3 shows the rows in the DEPT table stored in DEPTNO order, but this is a matter of chance, not design: relational tables do not impose any particular ordering on their rows. Table 1-4 shows that department number 10 has one employee, and department number 40 has none. Changes to data are usually very efficient with the relational model. New employees can be appended to the employees table, or they can be moved from one department to another simply by changing the DEPTNO value in their row.

Consider an alternative structure, where the data is stored according to the hierarchical paradigm. The hierarchical model was developed before the relational model, for technology reasons. In the early days of computing, storage devices lacked the capability for maintaining the many separate files that were needed for the many relational tables. Note that this problem is avoided in the Oracle database by abstracting the physical storage (files) from the logical storage (tables): there is no direct connection between tables and files and certainly not a one-to-one mapping. In effect, many tables can be stored in a very few files.

A hierarchical structure stores all related data in one unit. For example, the record for a department would include all that department’s employees. The hierarchical paradigm can be very fast and very space efficient. One file access may
be all that is needed to retrieve all the data needed to satisfy a query. The employees and departments listed previously could be stored hierarchically as follows:

<table>
<thead>
<tr>
<th></th>
<th>ACCOUNTING</th>
<th>7782</th>
<th>CLARK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RESEARCH</td>
<td>7369</td>
<td>SMITH, JONES, 7566, SCOTT</td>
</tr>
<tr>
<td></td>
<td>SALES</td>
<td>7499</td>
<td>ALLEN, WARD, 7521, MARTIN, 7654, BLAKE</td>
</tr>
<tr>
<td></td>
<td>OPERATIONS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this example layout, the rows and columns are of variable length. Columns are delimited with a comma, rows with a new line. Data retrieval is typically very efficient if the query can navigate the hierarchy: if one knows an employee’s department, the employee can be found quickly. If one doesn’t, the retrieval may be slow. Changes to data can be a problem if the change necessitates movement. For example, to move employee 7566, JONES from RESEARCH to SALES would involve considerable effort on the part of the database because the move has to be implemented as a removal from one line and an insertion into another. Note that in this example, while it is possible to have a department with no employees (the OPERATIONS department), it is absolutely impossible to have an employee without a department: there is nowhere to put him or her. This is excellent if there is a business rule stating that all employees must be in a department but not so good if that is not the case.

The relational paradigm is highly efficient in many respects for many types of data, but it is not appropriate for all applications. As a general rule, a relational analysis should be the first approach taken when modeling a system. Only if it proves inappropriate should one resort to nonrelational structures. Applications where the relational model has proven highly effective include virtually all OLTP and DSS systems. The relational paradigm can be demanding in its hardware requirements and in the skill needed to develop applications around it, but if the data fits, it has proved to be the most versatile model. There can be, for example, problems caused by the need to maintain the indexes that maintain the links between tables and the space requirements of maintaining multiple copies of the indexed data in the indexes themselves and in the tables in which the columns reside. Nonetheless, relational design is in most circumstances the optimal model.

A number of software publishers have produced database management systems that conform (with varying degrees of accuracy) to the relational paradigm; Oracle is only one. IBM was perhaps the first company to commit major resources to it,
but their product (which later developed into DB2) was not ported to non-IBM platforms for many years. Microsoft’s SQL Server is another relational database that has been limited by the platforms on which it runs. Oracle databases, by contrast, have always been ported to every major platform from the first release. It may be this that gave Oracle the edge in the RDBMS market place.

A note on terminology: Confusion can arise when discussing relational databases with people used to working with Microsoft products. SQL is a language and SQL Server is a database, but in the Microsoft world, the term SQL is often used to refer to either.

**CERTIFICATION OBJECTIVE 1.03**

**Summarize the SQL Language**

SQL is defined, developed, and controlled by international bodies. Oracle Corporation does not have to conform to the SQL standard but chooses to do so. The language itself can be thought of as being very simple (there are only 16 commands), but in practice SQL coding can be phenomenally complicated. That is why a whole book is needed to cover the bare fundamentals.
SQL Standards

Structured Query Language (SQL) was first invented by an IBM research group in the '70s, but in fact Oracle Corporation (then trading as Relational Software, Inc.) claims to have beaten IBM to market by a few weeks with the first commercial implementation: Oracle 2, released in 1979. Since then the language has evolved enormously and is no longer driven by any one organization. SQL is now an international standard. It is managed by committees from ISO and ANSI. ISO is the Organisation Internationale de Normalisation, based in Geneva; ANSI is the American National Standards Institute, based in Washington, DC. The two bodies cooperate, and their SQL standards are identical.

Earlier releases of the Oracle database used an implementation of SQL that had some significant deviations from the standard. This was not because Oracle was being deliberately different: it was usually because Oracle implemented features that were ahead of the standard, and when the standard caught up, it used different syntax. An example is the outer join (detailed in Chapter 7), which Oracle implemented long before standard SQL; when standard SQL introduced an outer join, Oracle added support for the new join syntax while retaining support for its own proprietary syntax. Oracle Corporation ensures future compliance by inserting personnel onto the various ISO and ANSI committees and is now assisting with driving the SQL standard forward.

SQL Commands

These are the 16 primary SQL commands, separated into commonly used groups:

The Data Manipulation Language (DML) commands:
- SELECT
- INSERT
- UPDATE
- DELETE
- MERGE

The Data Definition Language (DDL) commands:
- CREATE
- ALTER
- DROP
Chapter 1: Relational Database Design Using Oracle

- RENAME
- TRUNCATE
- COMMENT

The Data Control Language (DCL) commands:

- GRANT
- REVOKE

The Transaction Control Language (TCL) commands:

- COMMIT
- ROLLBACK
- SAVEPOINT

The first command, SELECT, is the main subject of Chapters 2 through 9. The remaining DML commands are covered in Chapter 10, along with the TCL commands. DDL is detailed in Chapter 11. DCL, which has to do with security, is only briefly mentioned: it falls more into the domain of the database administrator than the developer.

According to all the documentation, SELECT is a DML statement. In practice, no one includes it when they refer to DML—they talk about it as though it were a language in its own right (it almost is) and use DML to mean only the commands that change data.

A Set-Oriented Language

Most 3GLs are procedural languages. Programmers working in procedural languages specify what to do with data, one row at a time. Programmers working in a set-oriented language say what they want to do to a group (a “set”) of rows and let the database work out how to do it to all the rows in the set.

Procedural languages are usually less efficient than set-oriented languages at managing data, as regards both development and execution. A procedural routine for looping through a group of rows and updating them one by one will involve many lines of code, where SQL might do the whole operation with one command. The result: programmers’ productivity increases. During program execution, procedural code gives the database no options; it must run the code as it has been written. With SQL, the programmer states what he or she wants to do but not how to do it; the
The database has the freedom to work out how best to carry out the operation. This will usually give better results.

Where SQL fails to provide a complete solution is that it is purely a data access language. Most applications will need procedural constructs, such as flow control: conditional branching and iteration. They will also usually need screen control, user interface facilities, and variables. SQL has none of these. SQL is a set-oriented language capable of nothing other than data access. For application development, one will therefore need a procedural language that can invoke SQL calls. It is therefore necessary for SQL to work with a procedural language.

Consider an application that prompts a user for a name, retrieves all the people with that name from a table, prompts the user to choose one of them, and then deletes the chosen person. The procedural language will draw a screen and generate a prompt for a name. The user will enter the name. The procedural language will construct a SQL SELECT statement using the name and submit the statement through a database session to the database server for execution. The server will return a set of rows (all the people with that name) to the procedural language, which will format the set for display to the user and prompt him to choose one (or more) of them. The identifier for the chosen person (or people) will then be used to construct a SQL DELETE statement for the server to execute. If the identifier is a unique identifier (the primary key), then the set of rows to be deleted will be a set of just one row; if the identifier is nonunique, then the set selected for deletion would be larger. The procedural code will know nothing about the likely size of the sets retrieved or deleted.

CERTIFICATION OBJECTIVE 1.04

Use the Client Tools

There are numerous tools that can be used to connect to an Oracle database. Two of the most basic are SQL*Plus and SQL Developer. These are provided by Oracle Corporation and are perfectly adequate for much of the work that a developer or a database administrator needs to do. The choice between them is partly a matter of personal preference, partly to do with the environment and partly to do with functionality. SQL Developer undoubtedly offers far more functionality than SQL*Plus, but it is more demanding in that it needs a graphical terminal, whereas SQL*Plus can be used on character mode devices.
The tool that has lasted longest is SQL*Plus, and even though Oracle Corporation is promoting SQL Developer very strongly as a replacement, all people working in the Oracle environment will be well advised to become familiar with it.

Many experienced developers and database administrators (perhaps including the author of this book) treat the newer tools with a certain degree of skepticism—though this may be nothing more than an indication that these people are somewhat old-fashioned. Throughout this book both tools will be used.

SQL*Plus

SQL*Plus is a client-server tool for connecting to a database and issuing ad hoc SQL commands. It can also be used for creating PL/SQL code and has facilities for formatting results. It is available on all platforms to which the database has been ported—the sections that follow give some detail on using SQL*Plus on Linux and Windows. There are no significant differences with using SQL*Plus on any other platform.

In terms of architecture, SQL*Plus is a user process written in C. It establishes a session against an instance and a database over the Oracle Net protocol. The platforms for the client and the server can be different. For example, there is no reason not to use SQL*Plus on a Windows PC to connect to a database running on a Unix server (or the other way round) provided that Oracle Net has been configured to make the connection.

SQL*Plus on Linux

The SQL*Plus executable file on a Linux installation is sqlplus. The location of this file will be installation specific but will typically be something like:

/u01/app/oracle/product/12.1.0/db_1/bin/sqlplus

Your Linux account should be set up appropriately to run SQL*Plus. There are some environment variables that will need to be set. These are

- ORACLE_HOME
- PATH
- LD_LIBRARY_PATH

The ORACLE_HOME variable points to the Oracle Home. An Oracle Home is the Oracle software installation: the set of files and directories containing the
Use the Client Tools

executable code and some of the configuration files. The PATH must include the `bin` directory contained within the Oracle Home. The `LD_LIBRARY_PATH` should include the `lib` directory also contained within the Oracle Home, but in practice you may get away without setting this. Figure 1-8 shows a Linux terminal window and some tests to see if the environment is correct.

In Figure 1-8, first the `echo` command checks whether the three variables have been set up correctly: there is an ORACLE_HOME, and the `bin` and `lib` directories in it have been set as the last element of the PATH and the first element of the `LD_LIBRARY_PATH` variables, respectively. Then `which` confirms that the SQL*Plus executable file really is available, in the PATH. Finally, SQL*Plus is launched with a username, a password, and a connect identifier passed to it on the command line. If the tests do not return acceptable results and SQL*Plus fails to launch, this should be discussed with your system administrator and your database administrator. Some common errors with the logon itself are described in the section “Creating and Testing a Database Connection” later in this chapter.

The format of the logon string is the database username followed by a forward slash character as a delimiter, then a password followed by an `@` symbol as a delimiter, and finally an Oracle Net connect identifier. In this example, the username is `system`, whose password is `admin123`, and the database is identified by `coda`.

![FIGURE 1-8](image-url)

```
[oracle@hades ~]$ echo ORACLE_HOME
/ora1/app/oracle/product/12.1.0/dbhome_1
[oracle@hades ~]$ echo PATH
/usr/local/bin:/usr/bin:/bin
[oracle@hades ~]$ echo SQLPATH
/ora1/app/oracle/product/12.1.0/dbhome_1/lib
[oracle@hades ~]$ echo LD_LIBRARY_PATH
/usr/local/lib:/usr/lib:/lib:/usr/lib:
/ora1/app/oracle/product/12.1.0/dbhome_1/lib
[oracle@hades ~]$ echo ORACLE_HOME
/ora1/app/oracle/product/12.1.0/dbhome_1
[oracle@hades ~]$ echo ORACLE_SID
sock
[oracle@hades ~]$ which sqlplus
/usr/local/bin/sqlplus
[oracle@hades ~]$ sqlplus system/admin123@coda
SQL*Plus: Release 12.1.0.0.2 Beta on Sat Jun 8 22:39:18 2013
Copyright (c) 1982, 2013, Oracle.  All rights reserved.
Last Successful login time: Sat Jun 2013 22:39:18 +02:00
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - 64bit Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL*Plus 8.1.0.6.0 Beta
Following the logon, the next lines of text display the version of SQL*Plus being used, which is 12.1.0.0.2, the version of the database to which the connection has been made (which happens to be the same as the version of the SQL*Plus tool), and which options have been installed within the database. The last line is the prompt to the user, SQL>, at which point the user can enter any SQL*Plus or SQL command. If the logon does not succeed with whatever username (probably not system) you have been allocated, this should be discussed with your database administrator.

**SQL*Plus on Windows**

Historically, there were always two versions of SQL*Plus for Microsoft Windows: the character version and the graphical version. The character version is the executable file sqlplus.exe, and the graphical version was sqlplusw.exe. With the current release and 11g, the graphical version no longer exists, but many developers will prefer to use it, and the versions shipped with earlier releases are perfectly good tools for working with a 12c database. There are no problems with mixing versions: a 12c SQL*Plus client can connect to a 10g database, and a 10g SQL*Plus client can connect to a 12c database. Following a default installation of either the Oracle database or just the Oracle client on Windows, SQL*Plus will be available as a shortcut on the Windows Start menu. The location of the executable file launched by the shortcut will, typically, be something like the following:

D:\app\oracle\product\12.1.0\dbhome_1\BIN\sqlplus.exe

However, the exact path will be installation specific. Figure 1-9 shows a logon to a database with SQL*Plus, launched from the shortcut. The first line of text shows the version of SQL*Plus, which is the 12.1.0.0.2 release, and the time the program was launched. The third line of text is a logon prompt:

Enter user-name:

followed by the logon string entered manually, which was

system/admin123@coda

A change some people like to make to the shortcut that launches SQL*Plus is to prevent it from immediately presenting a logon prompt. To do this, add the NOLOG switch to the end of the command:

sqlplus /nolog
There is no reason not to launch SQL*Plus from an operating system prompt rather than from the Start menu shortcut: simply open a command window and run it. The program will immediately prompt for a logon, unless you invoke it with the NOLOG switch described above.

The tests of the environment and the need to set the variables if they are not correct, previously described for a Linux installation, are not usually necessary on a Windows installation. This is because the variables are set in the Windows Registry by the Oracle Universal Installer when the software is installed. If SQL*Plus does not launch successfully, check the Registry variables. Figure 1-10 shows the relevant section of the Registry, viewed with the Windows regedit.exe Registry Editor utility. Within the Registry Editor, navigate to the key:

```
HKEY_LOCAL_MACHINE
SOFTWARE
ORACLE
KEY_OraDb12c_home1
```

The final element of this navigation path will have a different name if there have been several 12c installations on the machine.
Note the values of the Registry variables ORACLE_HOME and ORACLE_HOME_NAME. These will relate to the location of the sqlplus.exe executable and the Start menu navigation path to reach the shortcut that will launch it.

Creating and Testing a Database Connection

SQL Plus does not have any way of storing database connection details. Each time a user wishes to connect to a database, the user must tell SQL Plus who they are and where the database is. There are variations depending on site-specific security facilities, but the most common means of identifying oneself to the database is by presenting a username and a case-sensitive password. There are two typically used forms of connect identifier for identifying the database: either by providing an alias that is resolved into the full connect details or by entering the full details.

From an operating system prompt, these commands will launch SQL Plus and connect as database user SCOTT, whose password is tiger using each technique:

```
sqlplus scott/tiger@orcl
sqlplus scott/tiger@ocpl2c.oracle.com:1521/orcl.oracle.com
```

The first example uses an alias, orcl, to identify the database. This must be resolved into the full connect details. This resolution can be done in a number of
ways, but one way or another it must be accomplished. The usual techniques for
this are to use a locally stored text file called the tnsnames.ora file (typically
contained within the network/admin subdirectory of the ORACLE_HOME) or to
contact an LDAP directory such as Microsoft’s Active Directory or Oracle’s Oracle
Internet Directory (OID).

The second example provides all the connect details in line. The connect details
needed are the hostname of the computer on which the database instance is running,
the TCP port on which the Oracle Net database listener can be contacted, and the
database service to which the user wishes the database listener to connect him.
The first technique, where the user need only enter an alias, requires the database
administrator to configure a name resolution mechanism; the second technique can
only work if the user knows the details himself.

There are a number of circumstances that will cause a SQL*Plus connection attempt
to fail. Figure 1-11 illustrates some of the more common problems.

First, the user launches SQL*Plus from a Windows operating system prompt, using
the NOLOG switch to prevent the immediate logon prompt. No problem so far.
Second, from the SQL> prompt, the user issues a connection request, which fails with a well-known error:

ORA-12154: TNS: could not resolve the connect identifier specified

This error is because the connect identifier given, wrongalias, cannot be resolved into database connection details by the TNS (Transparent Network Substrate—not an acronym particularly worth remembering) layer of Oracle Net. The name resolution method to be used and its configuration is a matter for the database administrator. In this case, the error is obvious: the user entered the wrong connect identifier.

The second connect attempt gives the correct identifier, orcl. This fails with

ORA-12541: TNS: no listener

This indicates that the connect identifier has resolved correctly into the address of a database listener, but that the listener is not actually running. Note that another possibility would be that the address resolution is faulty and is sending SQL*Plus to the wrong address. Following this error, the user should contact the database administrator and ask him or her to start the listener. Then try again.

The third connect request fails with

ORA-12514: TNS: listener does not currently know of service requested in connect descriptor

This error is generated by the database listener. SQL*Plus has found the listener with no problems, but the listener cannot make the onward connection to the database service. The most likely reason for this is that the database instance has not been started, so the user should ask the database administrator to start it and then try again.

The fourth connect request fails with

ORA-01017: invalid username/password; logon denied

To receive this message, the user must have contacted the database. The user has got through all the possible network problems, the database instance is running, and the database itself has been opened by the instance. The user just has the password or username wrong. Note that the message does not state whether it is the password or the username that is wrong; if it were to do so, it would be giving out information to the effect that the other one was right.

Finally, the fifth connect attempt succeeds.

The preceding example demonstrates a problem-solving technique you will use frequently. If something fails, work through what it is doing step by step. Read every error message.
SQL Developer

SQL Developer is a tool for connecting to an Oracle database (or, in fact, some non-Oracle databases, too) and issuing ad hoc SQL commands. It can also manage PL/SQL objects. Unlike SQL*Plus, it is a graphical tool with wizards for commonly needed actions. SQL Developer is written in Java and requires a Java Runtime Environment (JRE) to run.

Being written in Java, SQL Developer is available on all platforms that support the appropriate version of the JRE. There are no significant differences between platforms.

Installing and Launching SQL Developer

SQL Developer is not installed with the Oracle Universal Installer, which is used to install all other Oracle products. It does not exist in an Oracle Home but is a completely self-contained product. The latest version can be downloaded from Oracle Corporation’s website.

An installation of the 12c database will include a copy of SQL Developer, but it will not be the current version. Even if you happen to have an installation of the database, you will usually want to install the current version of SQL Developer as well.

To install SQL Developer, unzip the ZIP file. That’s all. It does require a JDK, the Java Runtime Environment, to be available; this comes from Oracle. But if an appropriate JDK is not already available on the machine being used, there are downloadable versions of SQL Developer for Windows that include it. For platforms other than Windows, the JDK must be preinstalled. Download it from Oracle’s website and install according to the platform-specific directions. To check that the JDK is available with the correct version, from an operating system prompt run the following command:

```java
java -version
```

This should return something like the following:

```java
java version "1.7.0_21"
Java(TM) SE Runtime Environment (build 1.7.0_21-b11)
Java HotSpot(TM) 64-Bit Server VM (build 23.21-b01, mixed mode)
```

If it does not, using `which java` may help identify the problem: the search path could be locating an incorrect version.
Once SQL Developer has been unzipped, change your current directory to the directory in which SQL Developer was unzipped and launch it. On Windows, the executable file is `sqldeveloper.exe`. On Linux, it is the `sqldeveloper.sh` shell script. Remember to check that the DISPLAY environment variable has been set to a suitable value (such as `127.0.0.1:0.0`, if SQL Developer is being run on the system console) before running the shell script.

Any problems with installing the JRE and launching SQL Developer should be referred to your system administrator.

**The SQL Developer User Interface**

Figure 1-12 shows the SQL Developer User Interface after connecting to a database.

The general layout of the SQL Developer window is a left pane for navigation around objects, and a right pane to display and enter information.

In the figure, the left pane shows that a connection has been made to a database. The connection is called `orcl_sys`. This name is just a label chosen when the connection was defined, but most developers will use some sort of naming convention—in this case, the name chosen is the database identifier, which is `orcl`, and the name of the user the connection was made as, which was `sys`. The
branches beneath list all the possible object types that can be managed. Expanding the branches would list the objects themselves. The right pane has an upper part prompting the user to enter a SQL statement and a lower part that will display the result of the statement. The layout of the panes and the tabs visible on them are highly customizable.

The menu buttons across the top menu bar give access to standard facilities:

■ **File**  A normal Windows-like file menu, from which one can save work and exit from the tool.

■ **Edit**  A normal Windows-like edit menu, from which one can undo, redo, copy, paste, find, and so on.

■ **View**  The options for customizing the SQL Developer user interface.

■ **Navigate**  Facilities for moving between panes and for moving around code that is being edited.

■ **Run**  Enables execution and debugging of the SQL statements, SQL script, or PL/SQL block that is being worked on. Debugging facilitates stepping through code line by line rather than running the entire block of code.

■ **Versioning**  Supports the creation of a code versioning repository to track different versions of your SQL programs.

■ **Tools**  Links to external programs, including Data Modeler and SQL Worksheet

■ **Help**  It’s pretty good.

SQL Developer can be a very useful tool, as it is highly customizable. Experiment with it, read the Help, and set up the user interface the way that works best for you.

**Creating a Database Connection**

Database connections can be created and saved for reuse. Figure 1-13 shows the window where connections can be defined. To reach this window, click the “+” symbol visible on the Connections tab shown previously in Figure 1-12.

The name for the connection is arbitrary. In this example, the name chosen is the name of the database connect identifier (orcl) suffixed with the username (hr) that will be embedded in the connection.

The username and password must both be supplied, but only the username will be saved unless the Save Password check box is selected. Saving a password means that future connections can be made without any password prompt. This is convenient
but highly dangerous if there is any possibility that the computer you are working on is not secure. In effect, you are delegating the authentication to your local operating system: if you can log on to that, you can log on to the database.

Assuming that you are using SQL Developer to connect to an Oracle database rather than to a third-party database, select the Oracle tab.

The Role drop-down box gives you the option to connect as sysdba. A sysdba connection is required before certain particularly serious operations (such as database startup and shutdown) can be carried out. It will never be needed for the exercises covered in this book.

The Connection Type radio buttons let you choose among five options:

- **Basic** This prompts for the machine name of the database server, the port on which the database listener will accept connection requests, and the instance (the SID) or the service to which the connection will be made.

- **TNS** If a name resolution method has been configured, then an alias for the database can be selected (from the local tnsnames.ora file) or entered, rather than the full details needed by the Basic option.

- **LDAP** Database service definitions and their connection details that are stored in a LDAP directory service may be queried by specifying the LDAP Server details.
Use the Client Tools

- **Advanced**: This allows entry of a full JDBC (Java Database Connectivity) connect string. This is completely Oracle independent and could be used to connect to any database that conforms to the JDBC standard.

- **Local/Bequeath**: If the database is running on the same machine as your SQL Developer client, this option allows you to connect directly to a server process bypassing the network listener.

Selecting Basic requires the user to know how to connect to the database; selecting TNS requires some configuration to have been done on the client machine by the database administrator, in order that the alias can be resolved into the full connection details.

After you enter the details, the Test button will force SQL Developer to attempt a logon. If this returns an error, then either the connection details are wrong, or there is a problem on the server side. Typical server-side problems are that the database listener is not running or that the database has not been started. Whatever the error is, it will be prefixed with an error number—some of the common errors were described in the preceding section, which described the use of SQL*Plus.

<table>
<thead>
<tr>
<th>SCENARIO &amp; SOLUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>If a connection fails, what can you do? How should you try to fix the problem?</strong></td>
</tr>
<tr>
<td><strong>If you can't fix the problem yourself, where can you go for help?</strong></td>
</tr>
</tbody>
</table>
Certification Objective 1.05

Create the Demonstration Schemas

Throughout this book, there are hundreds of examples of running SQL code against tables of data. For the most part, the examples use tables in two demonstration schemas provided by Oracle: the HR schema, which is sample data that simulates a simple human resources application, and the OE schema, which simulates a more complicated order entry application.

These schemas can be created when the database is created; it is an option presented by the Database Configuration Assistant (DBCA). If they do not exist, they can be created later by running some scripts that will exist in the database Oracle Home.

An earlier demonstration schema was SCOTT (password tiger). This schema is simpler than HR or OE. Many people with long experience of Oracle will prefer to use this. The creation script is still supplied; it is utlsampl.sql.

Users and Schemas

First, two definitions. In Oracle parlance, a database user is a person who can log on to the database. A database schema is all the objects in the database owned by one user. The two terms can often be used interchangeably, as there is a one-to-one relationship between users and schemas. Note that while there is in fact a CREATE SCHEMA command, this does not actually create a schema—it is only a quick way of creating objects in a schema. A schema is initially created empty, when a user is created with the CREATE USER command.

Schemas are used for storing objects. These may be data objects such as tables or programmatic objects such as PL/SQL stored procedures. User logons are used to connect to the database and access these objects. By default, users have access to the objects in their own schema and to no others, but most applications change this. Typically, one schema may be used for storing data that is accessed by other users who have been given permission to use the objects, even though they do not own them. In practice, very few users will ever have objects in their own schema, or permission to create them: they will have access rights (which will be strictly controlled) only to objects in another schema. These objects will be used by all users who run the application whose data that schema stores. Conversely, the users who own the data-storing schemas may never in fact log on: the only purpose of their schemas is to contain data used by others.
It is impossible for a data object to exist independently of a schema. Or in other words, all tables must have an owner. The owner is the user in whose schema the table resides. The unique identifier for a table (or any other schema object) is the username, followed by the object name. It follows that it is not possible for two tables with the same name to exist in the same schema, but that two tables with the same name (though possibly different structures or contents) can exist in different schemas. If an object does not exist in one’s own schema, to access it one must qualify its name with the name of the schema in which it resides. For example, HR.EMPLOYEES is the table called EMPLOYEES in user HR’s schema. Unless synonyms are available, only a user connected as HR could get to the table by referring to EMPLOYEES without a schema name qualifier. A synonym is a construct that makes an object accessible to other users without requiring its schema name as a prefix.

The HR and OE Schemas

The HR demonstration schema consists of seven tables, linked by primary key to foreign key relationships. Figure 1-14 illustrates the relationships between the tables, as an entity relationship diagram.
Two of the relationships shown in Figure 1-14 may not be immediately comprehensible. First, there is a many-to-one relationship from EMPLOYEES to EMPLOYEES. This is what is known as a self-referencing foreign key. This means that many employees can be connected to one employee, and it’s based on the fact that many employees may have one manager, but the manager is also an employee. The relationship is implemented by the column manager_id being a foreign key to employee_id, which is the table’s primary key.

The second relationship that may require explanation is between DEPARTMENTS and EMPLOYEES, which is bidirectional. The one-department-to-many-employees relationship simply states that there may be many staff members in each department, based on the EMPLOYEES dept_id column being a foreign key to the DEPARTMENTS primary key dept_id column. The one-employee-to-many-departments relationship shows that one employee could be the manager of several departments and is implemented by the manager_id column in DEPARTMENTS being a foreign key to the primary key employee_id column in EMPLOYEES.

Table 1-5 shows the columns of each table in the HR schema, using the notation described in the earlier section “Data Normalization” to indicate primary keys (#), foreign keys (\), and whether columns are optional (o) or mandatory (*).

The tables are:

- **REGIONS** has rows for major geographical areas.
- **COUNTRIES** has rows for each country, which are optionally assigned to a region.
- **LOCATIONS** includes individual addresses, which are optionally assigned to a country.
- **DEPARTMENTS** has a row for each department, optionally assigned to a location and optionally with a manager (who must exist as an employee).
- **EMPLOYEES** has a row for every employee, each of whom must be assigned to a job and optionally to a department and to a manager. The managers must themselves be employees.
- **JOBS** lists all possible jobs in the organization. It is possible for many employees to have the same job.
- **JOB_HISTORY** lists previous jobs held by employees, uniquely identified by employee_id and start_date; it is not possible for an employee to hold two jobs concurrently. Each job history record will refer to one employee, who will have had one job at that time and may have been a member of one department.
<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGIONS</td>
<td>*= region_id</td>
</tr>
<tr>
<td></td>
<td>o region_name</td>
</tr>
<tr>
<td>COUNTRIES</td>
<td>*= country_id</td>
</tr>
<tr>
<td></td>
<td>o country_name</td>
</tr>
<tr>
<td></td>
<td>o region_id</td>
</tr>
<tr>
<td>LOCATIONS</td>
<td>*= location_id</td>
</tr>
<tr>
<td></td>
<td>o street_address</td>
</tr>
<tr>
<td></td>
<td>o postal_code</td>
</tr>
<tr>
<td></td>
<td>* city</td>
</tr>
<tr>
<td></td>
<td>o state_province</td>
</tr>
<tr>
<td></td>
<td>o country_id</td>
</tr>
<tr>
<td>DEPARTMENTS</td>
<td>*= department_id</td>
</tr>
<tr>
<td></td>
<td>* department_name</td>
</tr>
<tr>
<td></td>
<td>o manager_id</td>
</tr>
<tr>
<td></td>
<td>o location_id</td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td>*= employee_id</td>
</tr>
<tr>
<td></td>
<td>o first_name</td>
</tr>
<tr>
<td></td>
<td>* last_name</td>
</tr>
<tr>
<td></td>
<td>* e-mail</td>
</tr>
<tr>
<td></td>
<td>o phone_number</td>
</tr>
<tr>
<td></td>
<td>* hire_date</td>
</tr>
<tr>
<td></td>
<td>*= job_id</td>
</tr>
<tr>
<td></td>
<td>o salary</td>
</tr>
<tr>
<td></td>
<td>o commission_pct</td>
</tr>
<tr>
<td></td>
<td>o manager_id</td>
</tr>
<tr>
<td></td>
<td>o department_id</td>
</tr>
<tr>
<td>JOBS</td>
<td>*= job_id</td>
</tr>
<tr>
<td></td>
<td>* job_title</td>
</tr>
<tr>
<td></td>
<td>o min_salary</td>
</tr>
<tr>
<td></td>
<td>o max_salary</td>
</tr>
<tr>
<td>JOB_HISTORY</td>
<td>*= employee_id</td>
</tr>
<tr>
<td></td>
<td>*= start_date</td>
</tr>
<tr>
<td></td>
<td>* end_date</td>
</tr>
<tr>
<td></td>
<td>*= job_id</td>
</tr>
<tr>
<td></td>
<td>o department_id</td>
</tr>
</tbody>
</table>
This HR schema is used for most of the exercises and many of the examples embedded in the chapters of this book and does need to be available.

There are rows in EMPLOYEES that do not have a matching parent row in DEPARTMENTS. This could be by design but might well be a design mistake that is possible because the DEPARTMENT_ID column in EMPLOYEES is not mandatory. There are similar possible errors in the REGIONS—COUNTRIES—LOCATIONS hierarchy, which really does not make a lot of sense.

The OE schema is considerably more complex than the HR schema. The table structures are much more complicated: they include columns defined as nested tables, user-defined data types, and XML data types. There are a number of optional exercises at the end of each chapter that are usually based on the OE schema. The objects referred to are described as they are used.

### Demonstration Schema Creation

If the database you are using was created specifically for studying for the SQL examination, the demonstration schemas should have been created already. They are an option presented by the Database Configuration Assistant when it creates a database. After database creation, the schemas may have to be unlocked and their passwords set; by default the accounts are locked, which means you cannot log on to them. These commands, which could be issued from SQL*Plus or SQL Developer, will make it possible to log on as users HR and OE using the passwords hr and oe:

```
ALTER USER hr ACCOUNT UNLOCK IDENTIFIED BY hr;
ALTER USER oe ACCOUNT UNLOCK IDENTIFIED BY oe;
```

These `ALTER USER` commands can only be issued when connected to the database as a user with DBA privileges, such as the user SYSTEM.

If the schemas were not created at database creation time, they can be created later by running scripts installed into the Oracle Home of the database. These scripts will need to be run from SQL*Plus or SQL Developer as a user with SYSDBA privileges. The script will prompt for certain values as it runs. For example, on Linux, first launch SQL*Plus from an operating system prompt:

```
sqlplus / as sysdba
```

There are various options for this connection, but the preceding syntax will usually work if the database is running on the same machine where you are running SQL*Plus. Then invoke the script from the `SQL>` prompt:

```
SQL> @?/demo/schema/human_resources/hr_main.sql
```
Create the Demonstration Schemas

The “?” character is a variable that SQL’Plus will expand into the path to the Oracle Home directory. The script will prompt for HR’s password, default tablespace, and temporary tablespace; the SYS password; and a destination for a logfile of the script’s execution. Typical values for the default tablespace and temporary tablespace are USERS and TEMP, but these must already exist. After completion, you will be connected to the database as the new HR user. To verify this, run these statements.

SQL> show user;
You will see that you are currently connected as HR; then run:

SQL> select table_name from user_tables;
You will see a list of the seven tables in the HR schema.

To create the OE schema, follow the same process, nominating the script:

`/?/demo/schema/order_entry/oe_main.sql`

The process for creating the schemas on Windows is identical, except for the path delimiters—where most operating systems use forward slashes, Windows uses back slashes. So the path to the Windows HR creation script becomes:

`@/?\demo\schema\human_resources\hr_main.sql`

Note that running these schema creation scripts will drop the schemas first if they already exist. Dropping a schema means removing every item in it and then removing the user. This should not be a problem, unless the schema has been used for some development work that needs to be kept.

If the demonstration creation schema scripts do not exist as just described, this will be because the Oracle Home installation has not been completed. Installing a database Oracle Home can be done from one CD, but there is a second CD (called the companion or example CD) that includes a number of components that are, strictly speaking, optional. These include the demonstration schemas. The companion CD should normally be installed; if this has not been done, this must be discussed with the database administrator.

*The demonstration schemas should not exist in production databases. It is not good, for security reasons, to have unnecessary schemas in a database that have well known usernames, capabilities, and (possibly) passwords.*
Chapter 1: Relational Database Design Using Oracle

CERTIFICATION SUMMARY

SQL is a language for managing access to normalized data stored in relational databases. It is not an application development language, but is invoked by such languages when they need to access data. The Oracle server technologies provide a platform for developing and deploying such applications. The combination of the Oracle server technologies and SQL result in an environment conforming to the relational database paradigm that is an enabling technology for Cloud computing.

Numerous client tools can be used to connect to an Oracle database. Two provided by Oracle Corporation are SQL*Plus and SQL Developer: SQL*Plus is installed as part of every Oracle client and Oracle database install, but SQL Developer can be installed as a separate product. Both tools can be used for preparing for the OCP examinations, and students should be familiar with both.

The demonstration schemas store example data that is used to illustrate the use of SQL, and also of more advanced Oracle development facilities.
TWO-MINUTE DRILL

Position the Server Technologies
- The Oracle database stores and manages access to user data.
- The Oracle WebLogic Server runs applications that connect users to the database.
- Oracle Enterprise Manager is a tool for managing databases, application servers, and, if desired, the entire computing environment.
- Languages built into the database for application development are SQL, PL/SQL, and Java.

Understand Relational Structures
- Data must be normalized into two-dimensional tables.
- Tables are linked through primary and foreign keys.
- Entity-relationship diagrams represent the tables graphically.

Summarize the SQL Language
- The DML commands are SELECT, INSERT, UPDATE, DELETE, and MERGE.
- The DDL commands are CREATE, ALTER, DROP, RENAME, TRUNCATE, and COMMENT.
- The DCL commands are GRANT and REVOKE.
- The TCL commands are COMMIT, ROLLBACK, and SAVEPOINT.

Use the Client Tools
- SQL*Plus is a command-line utility installed into the Oracle Home.
- SQL Developer is a graphical tool installed into its own directory.
- Both tools require a database connection, consisting of a username, a password, and a connect identifier.

Create the Demonstration Schemas
- The demonstration schemas are provided by Oracle to facilitate learning but must be created before they can be used.
Chapter 1: Relational Database Design Using Oracle

SELF TEST

Position the Server Technologies

1. What components of the IT environment can Oracle Enterprise Manager Cloud Control manage? (Choose the best answer.)
   A. Oracle databases
   B. Oracle application servers
   C. Third-party products
   D. The server machines
   E. All of the above

2. What languages can run within the database? (Choose all that apply.)
   A. SQL
   B. C
   C. PL/SQL
   D. Java
   E. Any other language linked to the OCI libraries

Understand Relational Structures

3. Data that is modeled into a form suitable for processing in a relational database may be described as being (Choose the best answer.)
   A. First normal form
   B. Third normal form
   C. Abnormal form
   D. Paranormal form

4. An entity-relationship diagram shows data modeled into (Choose the best answer.)
   A. Two-dimensional tables
   B. Multidimensional tables
   C. Hierarchical structures
   D. Object-oriented structures
Summarize the SQL Language

5. SQL is a set-oriented language. Which of these features is a consequence of this? (Choose the best answer.)
   A. Individual rows must have a unique identifier.
   B. Sets of users can be managed in groups.
   C. SQL statements can be placed within blocks of code in other languages, such as Java and PL/SQL.
   D. One statement can affect multiple rows.

6. Which of these constructs is not part of the SQL language? (Choose all that apply.)
   A. Iteration, based on WHILE..
   B. Iteration, based on FOR..DO
   C. Branching, based on IF..THEN..ELSE
   D. Transaction control, based on COMMIT
   E. Transaction control, based on ROLLBACK

Use the Client Tools

7. Which of these statements regarding SQL Developer are correct? (Choose two answers.)
   A. SQL Developer cannot connect to databases earlier than release 10g.
   B. SQL Developer can be installed outside an Oracle Home.
   C. SQL Developer can store passwords.
   D. SQL Developer relies on an LDAP directory for name resolution.

8. Which of the following are requirements for using SQL Developer? (Choose two correct answers.)
   A. A Java Runtime Environment
   B. The OCI libraries
   C. A name resolution method such as LDAP or a TNSNAMES.ORA file
   D. The SQL*Plus libraries
   E. A graphical terminal
Create the Demonstration Schemas

9. Where may the demonstration schemas be created? (Choose the best answer.)
   A. The demonstration schemas must be created in a demonstration database.
   B. The demonstration schemas cannot be created in a production database.
   C. The demonstration schemas can be created in any database.
   D. The demonstration schemas can be created in any database if the demonstration user is created first.

10. How can you move a schema from one user to another? (Choose the best answer.)
   A. Use the ALTER SCHEMA MOVE... command.
   B. You cannot move a schema from one user to another.
   C. A schema can only be moved if it is empty (or if all objects within it have been dropped).
   D. Attach the new user to the schema, then detach the old user from the schema.

LAB QUESTION

The OE schema includes these tables:

- CUSTOMERS
- INVENTORIES
- ORDERS
- ORDER_ITEMS
- PRODUCT_DESCRIPTIONS
- PRODUCT_INFORMATION
- WAREHOUSES

A CUSTOMER can place many ORDERS, and an order can have many ORDER_ITEMS. Each item will be of one product, described by its PRODUCT_INFORMATION, and each product may have several PRODUCT_DESCRIPTIONS, in different languages. There are a number of WAREHOUSES, each of which can store many products; one product may be stored in many warehouses. An INVENTORIES entry relates products to warehouses, showing how much of each product is in each warehouse.

Sketch out this schema as an entity-relationship diagram, showing the many-to-one connections between the tables and ensuring that there are no many-to-many connections.
SELF TEST ANSWERS

Position the Server Technologies

1. ☑ E. Cloud Control can manage the complete environment (according to Oracle Corporation).
   ✗ A, B, C, and D are incorrect. All of these can be managed by Cloud Control.

2. ☑ A, C, and D. SQL, PL/SQL, and Java can all run in the database.
   ✗ B and E are incorrect. C cannot run inside the database, and OCI is used by external processes to connect to the database; it does not run within it.

Understand Relational Structures

3. ☑ B. Third normal form is the usual form aimed for by systems analysts when they normalize data into relational structures.
   ✗ A, C, and D are incorrect. A is incorrect because first normal form is only the first stage of data normalization. C and D would be more suitable to the X-Files than to a database.

4. ☑ A. The relational model uses two-dimensional tables.
   ✗ B, C, and D are incorrect. B is incorrect because two dimensions is the limit for relational structures. C and D are incorrect because they refer to nonrelational structures (though there are facilities within the Oracle database for simulating them).

Summarize the SQL Language

5. ☑ D. In a set-oriented language, one command can affect many rows (a set), whereas a procedural language processes rows one by one.
   ✗ A, B, and C are incorrect. A is incorrect because while rows should have a unique identifier in a well designed application, this is not actually a requirement. B is incorrect because users cannot be grouped in the Oracle environment. C is incorrect because (even though the statement is correct) it is not relevant to the question.

6. ☑ A, B, and C. These are all procedural constructions, which are not part of a set-oriented language. They are all used in PL/SQL.
   ✗ D and E are incorrect. These are SQL’s transaction control statements.

Use the Client Tools

7. ☑ B and C. B is correct because SQL Developer can be installed in its own directory. C is correct because passwords can be saved as part of a connection definition (though this may not be a good idea).
   ✗ A and D are incorrect. A is incorrect because the Oracle Net protocol lets SQL Developer connect to a number of versions of the database. D is incorrect because LDAP is only one of several techniques for name resolution.
8. **A** and **E**. A is correct because SQL Developer is written in Java and therefore requires a Java Runtime Environment. E is correct because SQL Developer needs a graphics terminal to display windows.
   **B, C, and D** are incorrect. B is incorrect because SQL Developer uses JDBC to connect to databases, not OCL. C is incorrect because, while SQL Developer can use LDAP or a TNSNAMES.ORA file, it can also use and store the basic connection details. D is incorrect because SQL Developer is a completely independent product.

**Create the Demonstration Schemas**

9. **C**. The demonstration schemas can be created in any database, either at database creation time or by running scripts later.
   **A, B, and D** are incorrect. A and B are incorrect because, while they may be good practice, they are not a technical requirement. D is incorrect because it fails to understand that a schema can only be (and always is) created with a user.

10. **B**. A schema and a user are inseparable.
    **A, C, and D** are incorrect. A is incorrect because there is no such command. C and D are incorrect because they assume the impossible: that you can separate a user from his or her schema.

**LAB ANSWER**

Figure 1-15 shows a solution.

![Entity-relationship diagram](image)