This chapter provides an introduction to the SQL language and two tools for working with it. The first section presents a high-level overview of the SQL language, which will give you an idea of the capabilities of this language. Then some important basic concepts of the SQL language are introduced in the second section, such as constants, literals, variables, expressions, conditions, functions, operators, operands, and so on. Finally, this chapter provides a tour of SQL*Plus and SQL Developer, the two main tools we will use throughout this book to learn the SQL language. In order to maximize the benefits of any tool, you first must learn how to use it and to identify the main features available in that tool.

This is the first chapter with real SQL statement examples. It thus would be beneficial for you to have access to an Oracle database and a schema with the seven case tables introduced in Chapter 1, and described in detail in Appendix A. You can find the scripts to create that schema in the download hosted from this book’s catalog page or the Source Code page on the Apress website (www.apress.com).

We assume that Oracle is running; database (instance) startup and shutdown are normally tasks of a system or database administrator. Specific startup and shutdown procedures might be in place in your environment. However, if you are working with a stand-alone Oracle environment, and you have enough privileges, you can try the SQL*Plus STARTUP command or use the GUI offered by Oracle Enterprise Manager to start up the database.

### 2.1 Overview of SQL

SQL (the abbreviation stands for Structured Query Language) is a language you can use in (at least) two different ways: interactively or embedded. Using SQL interactively means that you enter SQL commands via a keyboard, and you get the command results displayed on a terminal or computer screen. Using embedded SQL involves incorporating SQL commands within a program in a different programming language (such as Java or C). This book deals solely with interactive SQL usage.

Although SQL is called a query language, its possibilities go far beyond simply data retrieval. Normally, the SQL language is divided into the following four command categories:

- Data definition (Data Definition Language, or DDL)
- Data manipulation (Data Manipulation Language, or DML)
- Retrieval
- Security and authorization
Data Definition

The SQL data definition commands allow you to create, modify, and remove components of a database structure. Typical database structure components are tables, views, indexes, constraints, synonyms, sequences, and so on. Chapter 1 introduced tables, columns, and constraints; other database object types (such as views, indexes, synonyms, and sequences) will be introduced in later chapters.

Almost all SQL data definition commands start with one of the following three keywords:

- **CREATE**, to create a new database object
- **ALTER**, to change an aspect of the structure of an existing database object
- **DROP**, to drop (remove) a database object

For example, with the **CREATE VIEW** command, you can create views. With the **ALTER TABLE** command, you can change the structure of a table (for example, by adding, renaming, or dropping a column). With the **DROP INDEX** command, you can drop an index.

One of the strengths of an RDBMS is the fact that you can change the structure of a table without needing to change anything in your existing database application programs. For example, you can easily add a column or change its width with the **ALTER TABLE** command. In modern DBMSs such as Oracle, you can even do this while other database users or applications are connected and working on the database—like changing the wheels of a train at full speed. This property of an RDBMS is known as **logical data independence** (see Ted Codd’s rule 9, discussed in Chapter 1).

Data definition is covered in more detail in Chapters 3 and 7.

Data Manipulation and Transactions

Just as SQL data definition commands allow you to change the **structure** of a database, SQL data manipulation commands allow you to change the **contents** of your database. For this purpose, SQL offers three basic data manipulation commands:

- **INSERT**, to add rows to a table
- **UPDATE**, to change column values of existing rows
- **DELETE**, to remove rows from a table

You can add rows to a table with the **INSERT** command in two ways. One way is to add rows one by one by specifying a list of column values in the **VALUES** clause of the **INSERT** statement. The other is to add one or more rows to a table based on a selection (and manipulation) of existing data in the database (called a **subquery**).

---

**Note** You can also load data into an Oracle database with various tools specifically developed for this purpose—such as Data Pump in Oracle Database 10g, Export and Import in previous Oracle releases, and SQL*Loader. These tools are often used for high-volume data loads.
Data manipulation commands are always treated as being part of a transaction. This means (among other things) that all database changes caused by SQL data manipulation commands get a pending status, until you confirm (commit) or cancel (roll back) the transaction. No one (except the transaction itself) can see the pending changes of a transaction before it is committed. That’s why a transaction is often labeled atomic: it is impossible for other database users to see parts of a transaction in the database. It is “all or nothing,” no matter how many DML operations the transaction comprises.

SQL offers two commands to control your transactions explicitly:

- COMMIT, to confirm all pending changes of the current transaction
- ROLLBACK, to cancel all pending changes and restore the original situation

Sometimes, transactions are committed implicitly; that is, without any explicit request from a user. For example, every data definition command implicitly commits your current transaction.

Note the following important differences between data manipulation and data definition:

- DELETE empties a table; DROP removes a table. TRUNCATE allows you to delete all the rows in a table in an efficient (but irrevocable) way.
- UPDATE changes the contents of a table; ALTER changes its structure.
- You can undo the consequences of data manipulation with ROLLBACK; data definition commands are irrevocable.

Chapter 6 will revisit data manipulation in more detail. Chapter 7 discusses the TRUNCATE command, which is considered a data definition command.

Retrieval

The only SQL command used to query database data is SELECT. This command acts at the set (or table) level, and always produces a set (or table) as its result. If a certain query returns exactly one row, or no rows at all, the result is still a set: a table with one row or the empty table, respectively.

The SELECT command (as defined in the ANSI/ISO SQL standard) has six main components, which implement all SQL retrieval. Figure 2-1 shows a diagram with these six main components of the SELECT command.

![Figure 2-1. The six main components of the SELECT command](image)
The lines in this diagram represent all possibilities of the `SELECT` command, like a railroad map. You can deduce the following three syntax rules from Figure 2-1:

- The order of these six command components is fixed.
- The `SELECT` and `FROM` components are mandatory.
- The remaining components (WHERE, GROUP BY, HAVING, and ORDER BY) are optional.

Table 2-1 gives a high-level description of the roles of these six components of the `SELECT` command.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>Which table(s) is (are) needed for retrieval?</td>
</tr>
<tr>
<td>WHERE</td>
<td>What is the condition to filter the rows?</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>How should the rows be grouped/aggregated?</td>
</tr>
<tr>
<td>HAVING</td>
<td>What is the condition to filter the aggregated groups?</td>
</tr>
<tr>
<td>SELECT</td>
<td>Which columns do you want to see in the result?</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>In which order do you want to see the resulting rows?</td>
</tr>
</tbody>
</table>

**Tip** The order of the `SELECT` command components as displayed in Table 2-1 is also a good order to think about them when writing SQL statements. Notice that the `SELECT` clause is almost the last one.

Components of the `SELECT` command implement three of the relational operators introduced in Chapter 1 (Section 1.6) as follows:

- The `SELECT` component acts as the *projection* operator.
- The `FROM` component implements the *join* operator.
- The *restriction* operator corresponds to the `WHERE` component.

Now that we are on the subject of relational operators, note that the *union*, *intersection*, and *difference* (minus) operators are also implemented in SQL. You can use these three set operators to combine the results of multiple `SELECT` commands into a single result table, as illustrated in Figure 2-2. We will revisit these operators in Chapter 8.
Security

SQL offers several commands to implement data security and to restrict data access. First of all, access to the database must be defined. User authorization is implemented by providing database users a login name and a password, together with some database-wide privileges. These are the most important commands in this area:

- **CREATE USER**, to define new database users
- **ALTER USER**, to change properties (privileges and passwords) of existing database users
- **DROP USER**, to remove user definitions from the database

Privileges and Roles

If users are authorized to access the database, you can implement fine-grained data access by granting specific *privileges*. The Oracle DBMS offers two types of privileges: system privileges and object privileges.

*System privileges* pertain to the right to perform certain (nonobject-related) actions; for example, you can have the **CREATE SESSION** privilege (allows you to log on to the database) and the **CREATE TABLE** privilege. Oracle supports approximately 140 different system privileges.

*Object privileges* involve the right to access a specific database object in a specific way; for example, the right to issue **SELECT**, **INSERT**, and **UPDATE** commands against the **EMPLOYEES** table. Table 2-2 lists the most important Oracle object privileges.

---

**Note** Granting and revoking system privileges is typically a task for database administrators. See *Oracle SQL Reference*, part of the official documentation set for the Oracle Database, for more details on both system and object privileges.
**Table 2-2. Important Oracle Object Privileges**

<table>
<thead>
<tr>
<th>Object Privilege</th>
<th>Allowable Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>Change the table structure (with ALTER TABLE)</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete rows</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Execute stored functions or procedures</td>
</tr>
<tr>
<td>FLASHBACK</td>
<td>Go back in time (with FLASHBACK TABLE)</td>
</tr>
<tr>
<td>INDEX</td>
<td>Create indexes on the table</td>
</tr>
<tr>
<td>INSERT</td>
<td>Insert new rows</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Create foreign key constraints to the table</td>
</tr>
<tr>
<td>SELECT</td>
<td>Query the table (or view)</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Change column values of existing rows</td>
</tr>
</tbody>
</table>

The Oracle DBMS allows you to group privileges into *roles*. Roles make user management much easier, more flexible, and also more manageable. The following are the corresponding SQL commands used to administer these privileges and roles:

- **GRANT**, to grant certain privileges or roles to users or roles
- **REVOKE**, to revoke certain privileges or roles from users or roles

A typical scenario is the following:

```
CREATE ROLE <role name>
GRANT privileges TO <role name>
GRANT <role name> TO user(s)
```

The first step creates a new (empty) role. The second step (which can be repeated as many times as you like) populates the role with a mix of object and system privileges. The third step grants the role (and thereby all its privileges) to a user in a single step.

Roles have several useful and powerful properties:

- Roles are dynamic; further changes to the role contents automatically affect all users previously granted that role.
- Roles can be enabled or disabled during a session.
- You can protect roles with a password. In that case, only users who know the role password can enable the role.
- The most important advantage of roles is their manageability.
**GRANT and REVOKE**

Each table has an owner, the user who created the table. Table owners are able to grant privileges on their tables to other database users using the `GRANT` command. As soon as you create a table, you implicitly get all object privileges on that table, with `GRANT OPTION`, as illustrated in Figure 2-3, which shows the syntax of the `GRANT` command.

---

Note: System privileges and roles are not considered in Figure 2-3, so the syntax diagram is incomplete.

---

**Figure 2-3. The GRANT command syntax diagram**

Here are some comments about the `GRANT` command:

- Table owners cannot grant the right to remove a table (DROP TABLE) to other database users. Note, however, that Oracle supports a (rather dangerous) DROP ANY TABLE system privilege.

- If you want to grant all object privileges to someone else, you can use the keyword ALL (see Figure 2-3). (Instead of ALL PRIVILEGES, the Oracle DBMS also allows you to specify ALL.)

- With a single `GRANT` command, you can grant privileges to a single user, a list of users, a role, or all database users. You can address all database users with the pseudo-user PUBLIC (see Figure 2-3).

- The UPDATE privilege supports an optional refinement: this privilege can also be granted for specific columns, by specifying column names between parentheses.

- In principle, there is no difference between tables and views when granting object privileges; however, the privileges ALTER, INDEX, and REFERENCES are meaningless in the context of views.

- The `GRANT OPTION` not only grants certain object privileges, but also grants the right to the grantees to spread these privileges further.
The counterpart of GRANT is the REVOKE command. Figure 2-4 shows the syntax diagram for REVOKE.

![Figure 2-4. The REVOKE command syntax diagram](image)

Besides the two standard SQL commands mentioned in this section (GRANT and REVOKE), Oracle supports several additional commands in the security and data access area; for example, to influence the locking behavior of the DBMS, to implement auditing, and to set up more detailed user authorization.

### 2.2 Basic SQL Concepts and Terminology

This section discusses the following topics:

- Constants (literals)
- Variables
- Operators, operands, conditions, and expressions
- Functions
- Database object names
- Comments
- Reserved words

#### Constants (Literals)

A constant (or literal) is something with a fixed value. We distinguish numbers (numeric constants) and text (alphanumeric constants). In database jargon, alphanumeric constants are also referred to as strings.

In the SQL language, alphanumeric constants (strings) must be placed between single quotation marks (quotes). Numbers are also relatively straightforward in SQL; however, don't put them between quotes or they will be interpreted as strings. If you like, you can explicitly indicate that you want SQL to interpret numeric values as floating point numbers by adding the suffixes `f` or `d` to indicate single (float) or double precision, respectively. Be careful with the decimal period and group separators (commas) in
numbers, because the correct interpretation of these characters depends on the value of a session parameter (NLS_NUMERIC_CHARACTERS), and there are some cultural differences in this area.

In SQL, dates and time durations (intervals) are special cases. They are typically specified and represented as alphanumeric constants, but they need something else to distinguish them from regular strings. In other words, you must help the DBMS to interpret the strings correctly as date or time-interval constants. Probably the most straightforward (and elegant) method is to prefix the strings with a keyword (DATE, TIMESTAMP, or INTERVAL) and to adhere to a well-defined notation convention. (See the examples in Table 2-3 and the third option in the following list.) These are the three options to specify date and time-related constants in SQL:

- Specify them as alphanumeric constants (strings) and rely on implicit interpretation and conversion by the Oracle DBMS. This is dangerous, because things can go wrong if the actual format parameter for that session is different from the format of the string.

- Specify them as alphanumeric constants (strings) and use a CAST or TO_DATE conversion function to specify explicitly how the strings must be interpreted (see Chapter 5).

- Specify them as alphanumeric constants (strings), prefixed with DATE, TIMESTAMP, or INTERVAL. If you use INTERVAL, you also need a suffix to indicate a dimension, such as DAY, MONTH, or YEAR.

Table 2-3 shows examples of using SQL constants.

<table>
<thead>
<tr>
<th>Table 2-3. Examples of SQL Constants (Literals)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>Numeric</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Alphanumeric</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Dates and intervals</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Note the subtle difference between 132 and '132'. The difference between numbers and strings becomes apparent when considering the operators they support. For example, numbers can be added or multiplied, but you cannot do that with strings. The only operator you can apply to strings is the concatenation operator.

In general, the SQL language is case-insensitive. However, there is one important exception: alphanumeric constants (strings) are case-sensitive. For example, 'JOneS' is not equal to 'Jones'. This is sometimes the explanation of getting the message “no rows selected” in cases where you were expecting to see rows in the result.
Variables

A variable is something that may have a varying value over time, or even an unknown value. A variable always has a name, so you can refer to it.

SQL supports two types of variables:

- **Column name variables**: The name of a column stays the same, but its value typically varies from row to row while scanning a table.
- **System variables**: These have nothing to do with tables; nevertheless, they can play an important role in SQL. They are commonly referred to as pseudo columns. See Table 2-4 for some examples of Oracle system variables.

### Table 2-4. Examples of Oracle System Variables (Pseudo columns)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDATE</td>
<td>The current system date in the database</td>
</tr>
<tr>
<td>CURRENT_DATE</td>
<td>The current date at the client application side</td>
</tr>
<tr>
<td>SYSTIMESTAMP</td>
<td>The system date and exact time, with time zone information</td>
</tr>
<tr>
<td>LOCALTIMESTAMP</td>
<td>The system date and exact time, with time zone information, at the client application side</td>
</tr>
<tr>
<td>USER</td>
<td>The name used to connect to the database</td>
</tr>
</tbody>
</table>

The difference between dates (and timestamps) at the database side and those at the client application side can be relevant if you are connected over a network connection with a database in a remote location.

Users commonly make mistakes by forgetting to include quotes in SQL statements. Consider the following SQL statement fragment:

```sql
...WHERE LOCATION = UTRECHT...
```

`LOCATION` and `UTRECHT` are both interpreted by Oracle as variable names (column names), although the following was probably the real intention:

```sql
...WHERE LOCATION = 'UTRECHT'...
```

Operators, Operands, Conditions, and Expressions

An operator does something. Operands are the “victims” of operations; that is, operands serve as input for operators. Sometimes, operators need only a single operand (in which case, they are also referred to as monadic operators), but most operators need two or more operands.
The SQL operators are divided in four categories, where the differentiating factor is the operand datatype:

- Arithmetic operators
- Alphanumeric operators
- Comparison operators
- Logical operators

### Arithmetic Operators

The SQL language supports four arithmetic operators, as shown in Table 2-5.

**Table 2-5. SQL Arithmetic Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
</tbody>
</table>

You can apply arithmetic operators only on **NUMBER** values; however, there are some exceptions:

- If you subtract two **DATE** values, you get the difference between those two dates, expressed in days.
- You can add a **DATE** and an **INTERVAL** value, which results in another date.
- If you add a **DATE** and a **NUMBER**, the number is interpreted as an interval expressed in days.

### The Alphanumeric Operator: Concatenation

SQL offers only one alphanumeric operator, allowing you to concatenate string expressions: `||`. This modest number of operators is compensated for by the overwhelming number of alphanumeric functions in SQL, which are discussed in Chapter 5. For an example of the use of the concatenation operator, see Table 2-8, later in this chapter.

### Comparison Operators

The comparison operators allow you to formulate conditions in SQL. Table 2-6 shows the comparison operators available in SQL.
### Table 2-6. SQL Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
</tr>
<tr>
<td><code>=</code></td>
<td>Equal to</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;&gt;</code> or <code>!=</code></td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

Expressions with comparison operators are also referred to as *predicates* or *Boolean expressions*. These expressions evaluate to **TRUE** or **FALSE**. Sometimes, the outcome is **UNKNOWN**, such as when you have rows with missing information. We will revisit this topic in more detail in Chapter 4, when we discuss null values.

### Logical Operators

SQL also offers three operators whose operands are conditions: the logical (or Boolean) operators. Table 2-7 lists these operators.

### Table 2-7. SQL Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>Logical AND</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>Logical OR (the <strong>inclusive</strong> OR)</td>
</tr>
<tr>
<td><strong>NOT</strong></td>
<td>Logical negation</td>
</tr>
</tbody>
</table>

### Expressions

An *expression* is a well-formed string containing variables, constants, operators, or functions. Just like constants, expressions always have a certain datatype. See Table 2-8 for some examples of expressions.
Table 2-8. SQL Expression Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 + 4</td>
<td>Numeric</td>
</tr>
<tr>
<td>ENAME</td>
<td></td>
</tr>
<tr>
<td>LOCATION = 'Utrecht'</td>
<td>Boolean</td>
</tr>
<tr>
<td>12*MSAL &gt; 20000 AND COMM &gt;= 100</td>
<td>Boolean</td>
</tr>
<tr>
<td>BDATE + INTERVAL '16' YEAR</td>
<td>Date</td>
</tr>
<tr>
<td>999</td>
<td>Numeric</td>
</tr>
</tbody>
</table>

The last example in Table 2-8 shows that the simplest expression is just a constant.

When SQL expressions get more complex, operator precedence can become an issue; in other words: what are the operator priority rules? Of course, SQL has some precedence rules. For example, arithmetic operators always have precedence over comparison operators, and comparison operators have precedence over logical operators. However, it is highly recommended that you use parentheses in your complex SQL expressions to force a certain expression evaluation order, just as you would do in regular mathematics.

Functions

Oracle has added a lot of functionality to the SQL standard in the area of functions. This is definitely one of the reasons why Oracle SQL is so powerful. You can recognize SQL functions by their signature: they have a name, followed by one or more arguments (between parentheses) in a comma-separated list. You can use functions in expressions, in the same way that you can use operators.

These are the six SQL function categories, based on their operand types:

- Numeric functions
- Alphanumeric functions
- Group functions
- Date functions
- Conversion functions
- Other functions

Table 2-9 shows some examples of SQL functions.
Table 2-9. Examples of SQL Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG(MSAL)</td>
<td>The average monthly salary</td>
</tr>
<tr>
<td>SQRT(16)</td>
<td>The square root of 16</td>
</tr>
<tr>
<td>LENGTH(INIT)</td>
<td>The number of characters in the INIT column value</td>
</tr>
<tr>
<td>LOWER(ENAME)</td>
<td>ENAME column value, in lowercase</td>
</tr>
<tr>
<td>SUBSTR(ENDDATE,4,3)</td>
<td>Three characters of the ENDDATE column value, from the fourth position</td>
</tr>
</tbody>
</table>

Oracle even allows you to create your own SQL functions by using the PL/SQL or Java languages. Chapter 5 will show a simple example of a user-defined function.

Database Object Naming

All objects in a database need names. This applies to tables, columns, views, indexes, synonyms, sequences, users, roles, constraints, functions, and so on. In general, to enhance the readability of your SQL code, it is highly recommended that you restrict yourself to using the characters A through Z, the digits 0 through 9, and optionally the underscore (_).

Note In Oracle, object names are case-insensitive; that is, internally all database object names are converted to uppercase, regardless of how you enter those names.

You may use digits in database object names; however, database object names should always start with a letter. Oracle object names have a maximum length of 30 characters.

Database objects need different names to be able to distinguish them, obviously. To be more precise, database objects need unique names within their namespace. On the other hand, different database users may use the same names for their own objects if they like, because the owner/object name combination is used to uniquely identify an object in the database.

If you insist on creating your own object names in Oracle SQL using any characters you like (including, for example, spaces and other strange characters), and you also want your object names to be case-sensitive, you can include those names within double quotes. The only restriction that remains is the maximum name length: 30 characters. Using this “feature” is discouraged, because you will always need to include those names in double quotes again in every interactive SQL statement you want to execute against those objects. On the other hand, you can use this technique in written applications to prevent conflicts with reserved words, including reserved words of future DBMS versions not known to you at application development time. Actually, several Oracle database utilities use this technique under the hood for precisely this reason.
Comments

You can add comments to SQL commands in order to clarify their intent or to enhance their maintainability. In other words, you can add text that does not formally belong to the SQL statements themselves, and as such should be ignored by the Oracle DBMS. You can add such comments in two ways: between /* and */ or after two consecutive minus signs. Comments after two minus signs are implicitly ended by a newline character; comments between /* and */ can span multiple lines. See Listing 2-1 for two examples.

**Listing 2-1. SQL Comments Examples**

```sql
/* this text will be considered a comment, 
   so the Oracle DBMS will ignore it ... */
-- and this text too, until the end of this line.
```

Listing 2-1 shows how you can add comments to SQL commands. Note that you can also add comments to database objects with the **COMMENT** command. See Chapter 7 for details.

Reserved Words

Just like any other language, SQL has a list of reserved words. These are words you are not allowed to use, for example, as database object names. If you insist on using a reserved word as an object name, you must enclose the name within double quotes, as explained earlier in the “Database Object Naming” section.


**Tip** The Oracle data dictionary contains a **V$RESERVED_WORDS** view. You can check your object names against this view to avoid using reserved words.

See Appendix A of this book, and also the *Oracle SQL Reference* for more details about naming rules for database objects and a more complete listing of SQL reserved words.

### 2.3 Introduction to SQL*Plus

SQL*Plus is a tool used to enter SQL commands and display the output. It is provided with every Oracle installation, whether on Windows or Unix. It is a command line interface and supports editing, user input, and report formatting.
Note In 11g, SQL*Plus for Windows (sqlplusw.exe) is no longer part of the client or database install. The command line version (sqlplus.exe) is still available. You can use an older version of SQL*Plus for Windows to connect to an 11g database, but some functionality may not be supported. SQL Developer, which we will cover later in this chapter, is a GUI interface that is shipped with 11g and should be considered the replacement for SQL*Plus for Windows.

To start SQL*Plus, simply type 'sqlplus' at the command prompt or after starting a DOS command session in Windows. Under normal circumstances, SQL*Plus prompts you for a username and corresponding password. If you are able to provide a valid username/password combination, the SQL> prompt appears on your screen to indicate that you have successfully established a session.

You can also start SQL*Plus with the username and password at the command line, as shown in Figure 2-5. In this case, if the username/password are valid, the SQL> prompt will appear. If not, you will be asked to enter a valid username and password.

![Figure 2-5. SQL*Plus screen after a successful connection using the username/password at the command line](image)

You can leave SQL*Plus with the commands EXIT or QUIT.

**Entering Commands**

SQL*Plus not only “understands” the SQL language, but it also supports and recognizes several tool-specific SQL*Plus commands. You must make sure to distinguish these SQL*Plus commands from SQL commands, because SQL*Plus treats these two command types differently, as you will see.

Let’s start by entering an arbitrary (and rather simple) SQL command in SQL*Plus, as shown in Listing 2-2.
Listing 2-2. A Basic SQL SELECT Command

SQL> select *
   2   from   employees;

Notice that SQL commands are often spread over multiple lines and, by default, SQL*Plus automatically displays line numbers during SQL command entry. If your SQL command is fully entered and you want SQL*Plus to execute it for you, you should finish the last line with a semicolon (;) as a delimiter. If you forget the semicolon (this will probably happen quite often, initially), you can still enter that semicolon on the next (empty) line, as shown here:

SQL> select *
   2   from   employees
   3   ;

Either way, the command will execute. SQL*Plus will return all columns and all rows of the EMPLOYEES table, since the asterisk character (*) means to show all columns of this table.

Using the SQL Buffer

SQL*Plus stores your most recent SQL command in an area called the SQL buffer. The SQL buffer is an important SQL*Plus concept. You can display the contents of the SQL buffer using a SQL*Plus command called LIST, as shown in Listing 2-3.

Listing 2-3. The SQL*Plus LIST Command

SQL> l
    1 select *
    2* from employees

SQL>
The ability to retrieve the last SQL statement from the SQL buffer is often very useful when you need to correct errors and re-execute the SQL statement. You will see how to do this in the subsequent sections, where we’ll also discuss some other SQL*Plus commands related to the SQL buffer.

If you enter a second SQL command, the SQL buffer is overwritten, and you lose the previous SQL command. In the “Saving Commands” section later in this chapter, you will see an easy method to save SQL commands for reuse in SQL*Plus.

Note from the example in Listing 2-3 that the SQL command returned from the SQL buffer did not include a semicolon at the end of it. The semicolon is not part of the SQL command itself, and it does not end up in the SQL buffer. If you enter a SQL command (or even a portion of a SQL command) and press the Enter key twice, without first adding a semicolon, the command will not be executed, but it will be saved in the SQL buffer.

The SQL*Plus commands you enter are not stored in the SQL buffer. You can run as many SQL*Plus commands as you like, but another SQL*Plus LIST command will display the same SQL command.

From the example in Listing 2-3, you can also note several other things about SQL*Plus commands:

- They are normally executed on a single line, unlike most SQL commands.
- You don’t need to enter a semicolon to execute SQL*Plus commands. They execute immediately when you press the Enter key.
- SQL*Plus commands can be abbreviated (L stands for LIST), whereas SQL commands cannot.

Rather than just see what is in the buffer, it is often useful to be able to edit its contents and then re-execute the SQL, so let’s now move on to discuss how to do that.

Using an External Editor

You can edit the contents of the SQL buffer in two ways:

- Use an external editor of your choice
- Use the built-in SQL*Plus editor

The main advantage of the SQL*Plus editor is that its functionality is always available in SQL*Plus, and the editor is totally independent of the underlying platform. The disadvantage of the SQL*Plus editor is its lack of user-friendliness and its very limited capabilities. This section explains how to use an external editor to edit your SQL commands. The next section will discuss the built-in SQL*Plus editor.

The default external editor under Microsoft Windows is Notepad.

You can also change or display the SQL*Plus external editor preference from the command line by using the DEFINE command, as shown in Listing 2-4.

Listing 2-4. Displaying and Changing the External Editor Preference

```
SQL> define _editor=Notepad
SQL> define _editor
DEFINE _EDITOR = "Notepad" (CHAR)
SQL>
```
Note The SQL*Plus variable that holds the name of the external editor is \_editor, with a leading underscore in its name.

You can invoke the external editor to change the contents of the SQL buffer. For this purpose, the SQL*Plus command is EDIT. You can invoke the external editor only when your SQL buffer is not empty. An empty buffer results in the error message “nothing to save.”

Invoking the external editor starts a subprocess, which means that you cannot return to SQL*Plus until you have closed the external editor window. Alternatively, you may want to start a separate editor session from the operating system (that is, not from SQL*Plus) so you can switch between two windows. In that case, you must make sure to save the changes in your editor window before executing the changed SQL command in SQL*Plus.

Using the SQL*Plus Editor

Learning to use the SQL*Plus editing commands is key to being more proficient and efficient in scripting. Instead of starting over if you make a mistake entering a statement, you can make a quick edit and then execute the statement. The editing commands are the same in all versions of SQL*Plus on all platforms.

To explore the SQL*Plus editor, we begin with the same simple SQL SELECT command in the SQL buffer (from the “Entering Commands” section earlier in the chapter):

```
SQL> select *
  2  from   employees;
```

Note Please follow all instructions in this section verbatim, even when you think there are some mistakes, because any mistakes are intentional.

It is important to realize that the SQL*Plus editor is line-oriented; that is, there is only one current line at any point in time. You can make changes only to that current line. (Perhaps you remember the good old EDLIN editor under MS-DOS?)

SQL*Plus marks the current line on screen with an asterisk (*) after the line number. Normally, it is the line you entered last; in our example, it is the second line.

If you want to change something on the first line, you must first activate that line with the L1 command. Let’s try to change the asterisk into two column names. C is an abbreviation for the SQL*Plus command CHANGE. Listing 2-5 shows how to use the LIST and CHANGE commands to make this change. SQL*Plus searches the current line for the first occurrence of an asterisk (*) and changes that character into eename, bdate.
**Listing 2-5. Using the SQL*Plus LIST and CHANGE Commands**

```sql
SQL> L
  1  select *
  2* from   employees

SQL> L1
  1* select *

SQL> c/*/eename, bdate/
  1* select eename, bdate

SQL> L
  1  select eename, bdate
  2* from   employees

SQL>
```

Instead of slashes (/), you can use any arbitrary character for the string delimiter (separator) in the `CHANGE` command. Also, a space character between the `C` and the first separator is not mandatory, and you can omit the last string delimiter too.

Now, let’s try to execute the SQL command in the buffer again. The SQL*Plus command to execute the contents of the SQL buffer is `RUN`, abbreviated to `R`. Apparently we made a mistake; we get an Oracle error message, as shown in Listing 2-6. Observe the error message. First, it shows a line number indication (`ERROR at line 1`), and within that line, an asterisk (*) indicates the position where the error was detected. Listing 2-6 also shows a first attempt to correct the error and the erroneous result of our `CHANGE` command.

**Listing 2-6. Fixing Typos with the SQL*Plus CHANGE Command**

```sql
SQL> R
  1  select eename, bdate
  2* from   employees
    select eename, bdate
  *
ERROR at line 1:
ORA-00904: "EENAME": invalid identifier

SQL> c/e//
  1* select eename, bdate

SQL>
```

We removed the first occurrence of an `e` on the first line, instead of the `e` in `eename`. This is the default (and only) way the `CHANGE` command works. This means that you must be careful with this command and be sure to specify appropriate search strings for replacement. In this case, it would have been better to issue the `c/ee/e/` command instead.

You can also add text at the end of the current line using the SQL*Plus `APPEND` command, which is abbreviated `A`. Listing 2-7 shows how we can first fix the mistake, and then add one more column to the `SELECT` expression.
Listing 2-7. Appending Text with the SQL*Plus APPEND Command

SQL> L1
1* select ename, bdate

SQL> c/slect ee/select e/
1* select ename, bdate

SQL> A , deptno
1* select ename, bdate, deptno

SQL> L
1  select ename, bdate, deptno
2* from   employees

SQL>

Note that the SQL*Plus APPEND command does not insert a space by default. In this case, we don’t need a space, but otherwise you should specify a second space character after the APPEND command.

You can also add one or more additional lines to the SQL buffer with the SQL*Plus INPUT command (abbreviated I), as shown in Listing 2-8. The lines you enter are added below the current line. If the current line is the last line in the buffer, the new lines are added at the end of the statement. This also means you need a “special trick” to add lines before the first line, as you’ll learn in the next section. Notice the line numbering; SQL*Plus automatically generates appropriate line numbers while entering text. You can stop entering additional lines by pressing the Enter key twice, or by entering a semicolon when you are adding lines at the end of the buffer.

Listing 2-8. Inserting Text with the SQL*Plus INPUT Command

1  select ename, bdate, deptno
2* from   employees

SQL> I
3  where  deptno = 30;

ENAME    BDATE         DEPTNO
-------- ----------- --------
ALLEN    20-FEB-1961       30
WARD     22-FEB-1962       30
MARTIN   28-SEP-1956       30
BLAKE    01-NOV-1963       30
TURNER   28-SEP-1968       30
JONES    03-DEC-1969       30

SQL>
Note The I is an abbreviation for **INPUT**, not for **INSERT**. **INSERT** is a SQL command (to add rows to a table in the database).

The SQL*Plus **DEL** command deletes the current line from the SQL buffer. You can optionally specify a line number with the **DEL** command to remove a certain line from the SQL buffer without making that line the current line first, or a range of line numbers to remove several lines with a single **DEL** command. See Listing 2-9 for an example.

**Listing 2-9. Deleting Lines with the SQL*Plus DEL Command**

```
SQL> L
  1  select ename, bdate, deptno
  2  from   employees
  3* where  deptno = 30
SQL> DEL
SQL> L
  1  select ename, bdate, deptno
  2* from   employees
```  

Note **DEL** is not an abbreviation for **DELETE**, because **DELETE** is a SQL command (to remove rows from a table in the database.)

Using SQL Buffer Line Numbers

You can make any line the current one by just entering the line number, without the **L** (**LIST**) command, as shown in Listing 2-10.

**Listing 2-10. Using Line Numbers to Change the Current Line**

```
SQL> L
  1  select code, description
  2  from   courses
  3* where  category = 'DSG'
SQL> 2
  2* from   courses
SQL> 42
```
Using line numbers, you can also replace any line in the SQL buffer without needing to use the SQL*Plus \texttt{DEL} command followed by a SQL*Plus \texttt{INPUT} command. Instead, simply enter the desired new line preceded by its line number. Listing 2-11 shows how to replace the first line and add a line at the end of the SQL buffer. Notice that the high line number (42) does not generate an error message, as it does in the example in Listing 2-10.

\textbf{Listing 2-11. Using Line Numbers to Change the SQL Buffer}

\begin{verbatim}
SQL> 1 select *
SQL> L
1  select *
2  from  courses
3* where  category = 'DSG'
SQL> 42 order  by code
SQL> L
1  select *
2  from  courses
3  where  category = 'DSG'
4* order  by code

SQL>
\end{verbatim}

As explained earlier, the SQL*Plus \texttt{INPUT} command always inserts lines \textit{below} the current line. The trick to insert extra lines \textit{before} the first line is to “overwrite” the artificial line zero, as demonstrated in Listing 2-12. This is a rather trivial example; however, this trick can be quite useful when creating views. Views are discussed in Chapter 10.

\textbf{Listing 2-12. Inserting Text Before the First Line of the SQL Buffer}

\begin{verbatim}
1 select *
2 from  courses
3 where  category = 'DSG'
4* order  by code

SQL> 0 /* this is just a comment */
SQL> L
1 /* this is just a comment */
2 select *
3 from  courses
4 where  category = 'DSG'
5* order  by code

SQL>
\end{verbatim}
Using the Ellipsis

If you are using the SQL*Plus CHANGE command, you might benefit from using three consecutive period characters, also known as the ellipsis. The examples in Listings 2-13 and 2-14 demonstrate the effect of using the ellipsis. First, we enter a new SQL command into the buffer and deliberately make a mistake.

Listing 2-13. Entering a SQL Command with a Deliberate Error

```
SQL> select mgr, department_name
2  from   departments
3  where  location = 'SCHIERMONNIKOOG';
select mgr, department_name
*
ERROR at line 1:
ORA-00904: "DEPARTMENT_NAME": invalid identifier
SQL>
```

Normally, the last command line you entered into the SQL buffer is automatically the current line. However, if an error condition occurs (such as in Listing 2-13), the line where the error is found becomes the current line. This allows you to correct any mistakes with the SQL*Plus CHANGE command immediately, without activating any line with the SQL*Plus LIST command. Listing 2-14 shows this phenomenon; the asterisk in the L* command means to show the current line.

Listing 2-14. Using the SQL*Plus L* Command and the Ellipsis (...)

```
SQL> L*
1* select mgr, department_name

SQL> c/d.../dname
1* select mgr, dname

SQL> 3
3* where  location = 'SCHIERMONNIKOOG'

SQL> c/s...g/BOSTON
3* where  location = 'BOSTON'

SQL>
```

The last example in Listing 2-14 shows that all CHANGE command searches are case-insensitive. As you can see, the ellipsis is powerful, but it’s also dangerous. For example, the command c/d.../dname searches for the first occurrence of a d on the first line, and then replaces everything to the end of the line.

SQL*Plus Editor Command Review

The SQL*Plus editor is a rather simple editor; nevertheless, it makes sense to spend some time to explore its possibilities. It might come in handy when you need to work with the Oracle DBMS in an
environment that is completely unknown to you, or where you are not allowed to launch an external editor from the underlying operating system. The SQL*Plus editor is always available, and it’s identical on all platforms supported by Oracle.

Table 2-10 summarizes all the SQL*Plus editor commands covered in this chapter.

**Table 2-10. Some SQL*Plus Editor-Related Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>Show the complete SQL buffer</td>
</tr>
<tr>
<td>LIST n (or just n)</td>
<td>Make line n the current line</td>
</tr>
<tr>
<td>CHANGE/old/new/</td>
<td>Change the first occurrence of old into new on the current line</td>
</tr>
<tr>
<td>APPEND txt</td>
<td>Append txt to the end of the current line</td>
</tr>
<tr>
<td>INPUT</td>
<td>Insert line(s) below the current line</td>
</tr>
<tr>
<td>DEL [x [y]]</td>
<td>Without arguments: remove current line. One argument: remove that line. Two arguments: remove range of lines (x and y can be line numbers, *, or LAST)</td>
</tr>
<tr>
<td>RUN (or /)</td>
<td>Execute the contents of the SQL buffer</td>
</tr>
<tr>
<td>EDIT</td>
<td>Start an external editor on the current buffer contents</td>
</tr>
<tr>
<td>DEFINE _EDITOR</td>
<td>Define your preferred external editor</td>
</tr>
</tbody>
</table>

As Table 2-10 shows, you can use the slash (/) command as an alternative for the SQL*Plus RUN command. The difference between the two is that RUN always displays the SQL command and the results, whereas the slash (/) command shows the results only.

**Saving Commands**

As explained earlier in the chapter, the SQL buffer is overwritten with every new SQL command you enter in SQL*Plus. If you want to save the contents of the SQL buffer, you can use the SQL*Plus SAVE command. The SAVE command creates a script file containing the contents of the SQL buffer.

If a script file already exists, you can specify (with the options APPEND or REPLACE) what you want the SAVE command to do in that case. The APPEND option is useful if you want to save all your SQL commands in one single file; for example, to print that file later.

Under Microsoft Windows, the options for saving the contents of the SQL buffer are also available via the File pull-down menu of SQL*Plus, as shown in Figure 2-6.
As an example of saving SQL commands, enter the commands shown in Listing 2-15.

Listing 2-15. The SQL*Plus SAVE Command

```
SQL> save BLA

SQL> select * from departments;

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DNAME</th>
<th>LOCATION</th>
<th>MGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ACCOUNTING</td>
<td>NEW YORK</td>
<td>7782</td>
</tr>
<tr>
<td>20</td>
<td>TRAINING</td>
<td>DALLAS</td>
<td>7566</td>
</tr>
<tr>
<td>30</td>
<td>SALES</td>
<td>CHICAGO</td>
<td>7698</td>
</tr>
<tr>
<td>40</td>
<td>HR</td>
<td>BOSTON</td>
<td>7839</td>
</tr>
</tbody>
</table>
```

Created file BLI.sql

```
SQL> save BLI

SQL> select * from courses;

<table>
<thead>
<tr>
<th>CODE</th>
<th>DESCRIPTION</th>
<th>CAT</th>
<th>DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>Introduction to SQL</td>
<td>GEN</td>
<td>4</td>
</tr>
<tr>
<td>OAU</td>
<td>Oracle for application users</td>
<td>GEN</td>
<td>1</td>
</tr>
<tr>
<td>JAV</td>
<td>Java for Oracle developers</td>
<td>BLD</td>
<td>4</td>
</tr>
<tr>
<td>PLS</td>
<td>Introduction to PL/SQL</td>
<td>BLD</td>
<td>1</td>
</tr>
<tr>
<td>XML</td>
<td>XML for Oracle developers</td>
<td>BLD</td>
<td>2</td>
</tr>
<tr>
<td>ERM</td>
<td>Data modeling with ERM</td>
<td>DSG</td>
<td>3</td>
</tr>
<tr>
<td>PMT</td>
<td>Process modeling techniques</td>
<td>DSG</td>
<td>1</td>
</tr>
<tr>
<td>RSD</td>
<td>Relational system design</td>
<td>DSG</td>
<td>2</td>
</tr>
<tr>
<td>PRO</td>
<td>Prototyping</td>
<td>DSG</td>
<td>5</td>
</tr>
<tr>
<td>GEN</td>
<td>System generation</td>
<td>DSG</td>
<td>4</td>
</tr>
</tbody>
</table>
```

10 rows selected.
SQL> save BLA
SP2-0540: File "BLA.sql" already exists.
Use "SAVE filename[.ext] REPLACE".

SQL> save BLA replace
Created file BLA.sql

SQL>

Note the error message after the second SAVE BLA attempt; REPLACE (or APPEND) is mandatory if a file already exists.

We have created two script files. These script files get the extension .SQL by default. If you prefer to use a different file name extension, you can change it with the SQL*Plus SUFFIX setting.

Running SQL*Plus Scripts

You can load script files saved with the SAVE command back into the SQL buffer with the GET command, followed by the name of the script. For example, you might reload a script and then edit it. If you want to load a script file and immediately execute it, you can use the START command (to get and run the script), as shown in Listing 2-16.

Listing 2-16. Using the SQL*Plus GET and START Commands

```
SQL> GET BLA
   1* select * from courses

SQL> START BLI

DEPTNO DNAME      LOCATION   MGR
------ ---------- -------- -----      
   10 ACCOUNTING NEW YORK  7782
   20 TRAINING   DALLAS    7566
   30 SALES      CHICAGO   7698
   40 HR         BOSTON    7839

SQL>
```

Listing 2-17 shows that you can also use the @ shortcut for the SQL*Plus START command.

Listing 2-17. Using the SQL*Plus @ Command

```
SQL> L
   1* select * from departments

SQL> @BLA

CODE DESCRIPTION                    CAT DURATION
---- ------------------------------ --- --------
SQL  Introduction to SQL            GEN        4
```

51
Specifying Directory Path Specifications

The SQL*Plus commands SAVE, GET, and START can handle full file name specifications, with directory paths. In the absence of a directory path, these commands default to the current directory. In a Microsoft Windows environment, it is relatively simple to define the directory (or folder) in which you want SQL*Plus to start. This is one of the shortcut properties, which you can set in the Start In field of the Properties dialog box, shown in Figure 2-7. Right-click the SQL*Plus icon and select Properties to open this dialog box.

![Figure 2-7. SQL*Plus shortcut properties](image)

10 rows selected.

SQL>
Chapter 2: Introduction to SQL, SQL*Plus, and SQL Developer

Through the Properties dialog box, you can also simplify the process to start SQL*Plus by specifying your username and password (such as book/book) in the Target field. In that case, the standard log on dialog will be skipped. However, this is a security risk, because anyone with access to your keyboard for more than two seconds will find out your database name and password.

Tip Under Microsoft Windows, you can also set the SQLPATH Registry setting to define a default search path for all files that cannot be found in the current directory. For example, you could have this Registry setting point to a central directory where you maintain all your generic SQL scripts. Just open the Registry Editor with the REGEDIT command and search for SQLPATH. Under other operating systems, check out the SQLPATH environment variable.

Adjusting SQL*Plus Settings

You can modify the behavior of SQL*Plus in numerous ways, based on SQL*Plus variables or settings. This section provides some simple examples to give you an idea of how this works. Chapter 11 covers the topic in more detail.

Listing 2-18 demonstrates using the SET command to change some SQL*Plus settings.

Listing 2-18. Changing SQL*Plus Settings with the SET Command

```
SQL> set pagesize 22
SQL> set pause "Hit [Enter]... 
SQL> set pause on

SQL> run
  1* select * from courses

Hit [Enter]...
```

The effect of changing the PAUSE and PAGESIZE settings as shown in Listing 2-18 is that SQL*Plus now produces screen output per page, in this case, 22 lines at a time. The PAUSE setting is useful if the results of your SQL commands don’t fit on your screen.

Tip When using the PAUSE setting, don’t just switch it on or off; make sure to specify a prompt string, too. Otherwise, SQL*Plus will just wait until you press the Enter key.

You can display the current values of SQL*Plus settings with the SHOW command, and you can revert to the default behavior with the SET command. Listing 2-19 shows examples of using these commands.
Listing 2-19. Displaying SQL*Plus Settings with the SHOW Command

SQL> show pages
pagesize 22

SQL> show pause
PAUSE is ON and set to "Hit [Enter]... 

SQL> set pause off

SQL> show pause
PAUSE is OFF

Although we are discussing the SQL*Plus tool in this section, there is also another (client tool-independent) way to influence your database session behavior: by using the SQL command ALTER SESSION. With this command, you can set several NLS (National Language Support) session parameters, a selection of which are shown in Table 2-11.

Table 2-11. Examples of NLS Session Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS_DATE_FORMAT</td>
<td>Default format to display dates</td>
</tr>
<tr>
<td>NLS_TIME_FORMAT</td>
<td>Default format to display timestamps</td>
</tr>
<tr>
<td>NLS_LANGUAGE</td>
<td>The language for SQL*Plus feedback and messages</td>
</tr>
<tr>
<td>NLS_NUMERIC_CHARACTERS</td>
<td>The decimal point and group separator characters</td>
</tr>
<tr>
<td>NLS_CURRENCY</td>
<td>The currency symbol</td>
</tr>
</tbody>
</table>

The most important parameter in this list is probably NLS_DATE_FORMAT, because this parameter influences the way date values are interpreted and displayed by your session, which is often a source of confusion. Listing 2-20 shows an example of using the ALTER SESSION command to set some NLS session parameters.

Listing 2-20. Changing NLS Parameters with ALTER SESSION

SQL> alter session
2  set nls_date_format='dd-mm-yyyy'
3       nls_language=Dutch
4       nls_currency='Eur';

Sessie is gewijzigd.

SQL>
If you change settings with the `ALTER SESSION` command, or if you change certain SQL*Plus settings with the SQL*Plus `SET` command, you lose these changes as soon as you log off. On startup, SQL*Plus will use the default values again. If you want to avoid the hassle of applying the same changes over and over again, you can store these SQL and SQL*Plus commands in a file with the special name `login.sql`. This file is automatically executed when you start SQL*Plus, or even when you change connections within a SQL*Plus session with the `CONNECT` command. Note that SQL*Plus must be able to find this file in the directory it starts in or via the `SQLPATH` Registry setting. `login.sql` is an example of a SQL*Plus script. We will revisit this type of file in more detail in Chapter 11.

If the rows of a result table don't fit on a single line on your screen (and the line wrapping makes the result rather ugly), a solution might be to narrow the display of one or more columns with the SQL*Plus `COLUMN` command. By default, SQL*Plus displays all columns on the screen with a width derived from the corresponding column definitions found in the data dictionary. Listing 2-21 shows how you can narrow (or widen) the display of alphanumeric columns on your screen by using the `FORMAT` option of the `COLUMN` command.

**Listing 2-21. Changing the Width of Alphanumeric Columns**

```sql
SQL> select * from courses
  2  where category = 'BLD';

  CODE DESCRIPTION                    CAT DURATION
  ---- ------------------------------ --- --------
     JAV  Java for Oracle developers   BLD        4
     PLS  Introduction to PL/SQL      BLD        1
     XML  XML for Oracle developers   BLD        2

SQL> COLUMN description FORMAT a26
SQL> /

  CODE DESCRIPTION                CAT DURATION
  ---- -------------------------- --- --------
     JAV  Java for Oracle developers BLD        4
     PLS  Introduction to PL/SQL     BLD        1
     XML  XML for Oracle developers  BLD        2

SQL>
```

All SQL*Plus commands (and their optional components) can be abbreviated, as long as the abbreviation is unique. For example, the `COLUMN` command can be abbreviated to `COL`, and `FORMAT` can be abbreviated to `FOR` (see Listing 2-22).

You can influence the width of numeric columns in a similar way, as you can see in Listing 2-22.

**Listing 2-22. Changing the Display of Numeric Columns**

```sql
SQL> select * from salgrades
  2  where grade > 3;

  CODE DESCRIPTION                    CAT DURATION
  ---- ------------------------------ --- --------
     JAV  Java for Oracle developers   BLD        4
     PLS  Introduction to PL/SQL      BLD        1
     XML  XML for Oracle developers   BLD        2

SQL>
```
### CHAPTER 2  ■ INTRODUCTION TO SQL, SQL*PLUS, AND SQL DEVELOPER

<table>
<thead>
<tr>
<th>GRADE</th>
<th>LOWERLIMIT</th>
<th>UPPERLIMIT</th>
<th>BONUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2001</td>
<td>3000</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>3001</td>
<td>9999</td>
<td>500</td>
</tr>
</tbody>
</table>

SQL> COL bonus FOR 9999.99
SQL> /

<table>
<thead>
<tr>
<th>GRADE</th>
<th>LOWERLIMIT</th>
<th>UPPERLIMIT</th>
<th>BONUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2001</td>
<td>3000</td>
<td>200.00</td>
</tr>
<tr>
<td>5</td>
<td>3001</td>
<td>9999</td>
<td>500.00</td>
</tr>
</tbody>
</table>

SQL>

If you want to save all your current SQL*Plus settings in a file (a SQL*Plus script file), use the `STORE SET` command. See Listing 2-23 for the syntax of this command.

**Listing 2-23. SQL*Plus STORE SET Command Syntax**

```sql
SQL> STORE SET <filename>[.sql] [REPLACE|APPEND]
```

The brackets in Listing 2-23 (around `.sql` and `REPLACE|APPEND`) are part of a common syntax notation convention to denote optional command clauses. This convention is also used in Appendix A of this book. In this convention, a vertical bar (|) can be used to separate optional choices, as in `[REPLACE|APPEND]`. Uppercase components such as `SET` and `APPEND` should be entered verbatim; lowercase components (such as `<filename>`) should be replaced (in this case) by a file name of your own choice. See Appendix A for more details.

If you have saved all SQL*Plus settings in a script file by using the `STORE SET` command, you can restore those settings at any time using the `START` (or `@`) command. This allows you to write SQL*Plus scripts that capture all SQL*Plus settings at the beginning, change various settings during script execution, and then restore the original settings at the end of the script.

### Spooling a SQL*Plus Session

You can record the complete results (as displayed on your screen) of a SQL*Plus session in an operating system file, using the SQL*Plus `SPOOL` command. Listing 2-24 shows an example.

**Listing 2-24. Using the SQL*Plus SPOOL Command**

```sql
SQL> spool BLA.TXT [create|replace|append]
SQL> select * from employees;
...  
SQL> select * from departments;
...  
SQL> spool off
```

The `BLA.TXT` file, created in the same directory or folder where the `SAVE` command stores its script files, now contains a complete copy of all screen output. As Listing 2-24 shows, you can influence the
behavior of the **SPOOL** command by specifying one of the following keywords: **CREATE**, **REPLACE**, or **APPEND**. With these three options, you can specify which behavior you want in case the specified file already exists. Just try these options for yourself; the error messages are self-explanatory.

### Describing Database Objects

When formulating SQL commands, it is sometimes convenient to get a quick overview of the structure of a table; for example, to see the column names and the datatypes. In such cases, the SQL*Plus **DESCRIBE** command is what you need. See Listing 2-25 for an example.

**Listing 2-25. The SQL*Plus DESCRIBE Command**

```sql
SQL> descr employees

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPNO</td>
<td>NOT NULL</td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>ENAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(8)</td>
</tr>
<tr>
<td>INIT</td>
<td>NOT NULL</td>
<td>VARCHAR2(5)</td>
</tr>
<tr>
<td>JOB</td>
<td>NULL</td>
<td>VARCHAR2(8)</td>
</tr>
<tr>
<td>MGR</td>
<td>NULL</td>
<td>NUMBER(4)</td>
</tr>
<tr>
<td>BDATE</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td>MSAL</td>
<td>NOT NULL</td>
<td>NUMBER(6,2)</td>
</tr>
<tr>
<td>COMM</td>
<td>NOT NULL</td>
<td>NUMBER(6,2)</td>
</tr>
<tr>
<td>DEPTNO</td>
<td></td>
<td>NUMBER(2)</td>
</tr>
</tbody>
</table>
```

### Executing Commands from the Operating System

The **HOST** command (most implementations support a platform-specific shortcut, such as $ or !) allows you to execute commands at the underlying operating system; for example, on a Microsoft Windows system, a command window is opened. Depending on the underlying operating system, you can finish the subsession and return to your SQL*Plus session with **EXIT**, **LOGOUT**, or a similar command.

### Clearing the Buffer and the Screen

With the **CLEAR BUFFER** command, you can empty the SQL buffer in SQL*Plus. This is something you won’t need to do too often, because the SQL buffer is overwritten each time by consecutive commands.

With the **CLEAR SCREEN** command, you can start at the top of a new, empty SQL*Plus screen.

### SQL*Plus Command Review

Table 2-12 shows an overview of all SQL*Plus commands covered in this chapter (including the SQL*Plus editor commands already listed in Table 2-10).
Table 2-12. Some SQL*Plus Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE</td>
<td>Save the SQL buffer contents in a script file</td>
</tr>
<tr>
<td>GET</td>
<td>Read a saved script file back into the SQL buffer</td>
</tr>
<tr>
<td>START or @</td>
<td>Execute the contents of a script file</td>
</tr>
<tr>
<td>SPOOL</td>
<td>Copy all screen output to a file</td>
</tr>
<tr>
<td>SET</td>
<td>Change a SQL*Plus setting</td>
</tr>
<tr>
<td>SHOW</td>
<td>Show the current value of SQL*Plus settings</td>
</tr>
<tr>
<td>COLUMN ... FORMAT</td>
<td>Change screen display attributes of a column</td>
</tr>
<tr>
<td>STORE SET</td>
<td>Save the current SQL*Plus settings in a script file</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>Provide a description of a database object</td>
</tr>
<tr>
<td>HOST or $</td>
<td>Start a subsession at the operating system level</td>
</tr>
<tr>
<td>CLEAR BUFFER</td>
<td>Empty the SQL buffer</td>
</tr>
<tr>
<td>CLEAR SCREEN</td>
<td>Start with an empty SQL*Plus screen</td>
</tr>
</tbody>
</table>

We also introduced the following SQL command in this section:

- **ALTER SESSION** changes various settings for your session, such as NLS settings.

### 2.4 Introduction to SQL Developer

SQL Developer is the Graphical User Interface (GUI) tool that Oracle supplies to query the database, explore objects, run reports, and run scripts. It runs on Windows, Linux and Mac OS X. It can be used to access Oracle databases 9i, 10g, and 11g, as well as other databases such as Times Ten, Microsoft Access, MySQL and SQL Server.

#### Installing and Configuring SQL Developer

SQL Developer is included as part of Oracle Database 11g. You can also download it from the following URL:

Once you save the downloaded archive and extract it to a directory, double click on sqldeveloper.exe to start SQL Developer.

**Note** SQL Developer for Windows does not create any menu shortcuts or icons on the desktop. You need to create these manually if you want them. Create a desktop shortcut by right clicking on the file and selecting **Send To Desktop (create shortcut)**. SQL Developer also does not create any registry entries. Thus, uninstalling SQL Developer is as simple as deleting the SQL Developer directory that you created when you unpacked the archive.

One of the first tasks that you may be prompted to do when you start SQL Developer for the first time is to locate the Java Development Kit (JDK). If you selected the option to download SQL Developer with the JDK, then java.exe will be included. In this example, SQL Developer is installed in C:\oracle\product\sqldeveloper and the location of the JDK will be in the subdirectory structure show in Figure 2-8.

![Figure 2-8. SQL Developer java.exe location](image)

When SQL Developer first starts, the Start Page shown in Figure 2-9 opens. This page includes links to documentation, to tutorials, and to the SQL Developer Forum.
Note As SQL Developer is a non-licensed (free) product, support is not obtained through Oracle’s Metalink site. The SQL Developer Forum on Oracle Technet (http://www.oracle.com/technology/index.html) is the location for support and questions. When you have questions or issues, look there for assistance.

Figure 2-9. The SQL Developer start page

There is not a great deal of basic configuration for SQL Developer that you need to do at this time. The ‘out of the box’ settings are fairly good for most users, but there are a couple of items that are worth considering: setting the default script file location and disabling default extensions.

It is usually a good idea to specify the default location for saving and running scripts. One minor annoyance with SQL Developer is that the settings for the file locations are spread among several different dialogs. Select Tools > Preferences to bring up the Preferences dialog box, as shown in Figure 2-10. To set the Script location, select Databases > Worksheet and enter the preferred location for scripts in the ‘Select default path to look for scripts’ box.
A second task is to disable some of the extensions that you do not need at this time. The advantage is reducing the start time for SQL Developer. For this book, you don’t need any of the extensions, so unselect them all. You will be prompted for a restart. You should notice that SQL Developer restarts considerably faster than it did when you first started it.

Connecting to a Database

Unlike SQL*Plus, you do not have to enter your username, password, and database name every time you connect. With SQL Developer you can have multiple connections that can be saved and organized. If you are using multiple accounts for a single database, you can have a connection created for each of those accounts.

Note You can have multiple connections open at one time, but be careful when one of those connections is to a production database. Two common problems leading to the need for database recovery are when a table is accidently dropped and when data is mistakenly modified in production.

To create a new connection, click on the Connections tab to make it active and then click on the large green cross (+) in the upper left corner. You can also right click on the Connections icon and select...
New Connection. This will bring up the New / Select Database Connection dialog as seen in Figure 2-11. In this example, the connection is the book user to a local database.

![New / Select Database Connection](image)

Figure 2-11. Creating a database connection

To organize your connections, you can create folders and add them to folders. You could organize by database name, type, and location, or any meaningful criteria. There is no option to create a new folder, so you add a connection to a new folder. Right click on the connection, select Add to Folder, and if there aren’t any folders defined you will only have the New Folder option. Enter a folder name in the dialog box. If folders have already been defined, you have the option to add to an existing folder or create a new folder. For existing folders, you can drag and drop the connection onto a folder name to assign it to that folder.

Exploring Objects

SQL Developer includes an Object Browser, which enables you to see the tables, indexes, procedures that you own and have access to query or execute. Figure 2-12 shows how to look at the table definition.

The tabs on the table object window enable you to see additional details about the object. There are two tabs that deserve special mention, Data and SQL. The Data tab will display the actual data in the table, which is like doing a select * from table_name. The Data tab is also part of the View object window. The SQL tab, which is in every object window, displays the actual SQL calls to create the object. Figure 2-13 shows the data in the employees table that is displayed by clicking the Data tab.

You can also explore the objects owned by others that you are able to access. At the very bottom of the object list, the Other Users entry can be expanded to show all of the objects you can access. All the users in the database are displayed, even if you cannot see any of their objects.
CHAPTER 2 INTRODUCTION TO SQL, SQL*PLUS, AND SQL DEVELOPER

Figure 2-12. Browsing a table

Entering Commands

The SQL Worksheet is where you enter commands to query and modify data. Like SQL*Plus, you can enter SQL and PL/SQL commands. Some SQL*Plus commands are supported, such as COLUMN, DESCRIBE and SPOOL. For a full list of supported and unsupported SQL*Plus commands, please refer to the Oracle SQL Developers User’s Guide.

The Worksheet is automatically opened when you connect to a database. If you need to open another worksheet or have closed the only one open, click on the SQL Worksheet icon or select the Tools » SQL Worksheet menu option.

Note If the Worksheet contains more than one statement, the statements must be terminated with a ; or / (on a separate line). If they are not properly terminated, the session will return an error message “ORA-00933: SQL command not properly ended”.
Run Statement

Unlike SQL*Plus, a statement is not automatically run when you enter a ; or /. The Run Statement (F9) command or the large green triangle icon is used to run a single command. If the worksheet contains more than one command, Run Statement will run the command immediately after the selected line, assuming that the previous statement(s) have been terminated with a ; or /.

Let’s start by entering the following, simple statement:

```
SELECT * FROM EMPLOYEES;
```

There are two things worth noting: First, the SQL statement reserved words are highlighted; second, EMPLOYEES is suggested as the table after you type FROM E. The syntax highlighting is handy when you accidentally type FORM instead of FROM. The auto-complete feature is also a time saver as it can suggest table or view and column names.

Click on the Run Statement button or press F9 to execute the query and display the data in the Query Result window, as seen in Figure 2-14.
To change the sort order of the data, double click on a column heading in the Query Result window.

**Run Script**

The **Run Script** command will run all the statements and/or SQL*Plus commands in the worksheet. This is the command to use when you have multiple statements or want to format the output using supported SQL*Plus commands.

Below the `SELECT * FROM EMPLOYEES;` we entered in the worksheet, enter `SELECT * FROM DEPARTMENTS;` and then click the Run Script button or press F5. The output will be displayed in the Script Output window alongside the Query Result window. Notice that the output is almost identical to what you have seen in SQL*Plus and is displayed below in Figure 2-15.
Figure 2-15. Querying EMPLOYEES and DEPARTMENTS tables

When running scripts, the output is appended to the Script Output window. To clear the window so that only new output is displayed, click on the Clear button (the picture of the pencil eraser).

Note Not all supported SQL*Plus commands are properly interpreted for Run Script. For example, the COLUMN command did not change the column headings, but SET FEEDBACK OFF worked as expected.

Saving Commands to a Script

After taking time to create a complex statement, it is wise to save that command to a script that you can run later. After entering the commands and statement(s), select File > Save, press CTL+S, or click on the disk button to bring up the File Save dialog box. The directory that it opens should be the same one you set in the Configuration section. The File Save dialog box is shown in Figure 2-16.
Running a Script

To run the script we just saved, there are two ways to load and run. The SQL*Plus standard of using @ is supported. To use the @ command, type @employees.sql in the worksheet and select Run Script (F5). This is demonstrated in Figure 2-17.

The second option is to select File > Open and pick the employees.sql file you just saved. The commands contained in that file will be loaded into the worksheet. Select the database connection you want to use in the Choose db Connection drop down box in the upper right of the employees.sql window. Until you select the connection, the other buttons will remain grayed out. After you select the connection, press the Run Script button to see the output, as seen in Figure 2-18.
We have just touched on the features of SQL Developer. For further information, visit the SQL Developer home page at:

In Chapter 11, we will revisit SQL*Plus to cover some more advanced features that are useful in writing scripts to automate your work. In case you are curious about more SQL*Plus features, feel free to visit the Oracle online documentation or refer to the quick reference in Appendix A of this book.
Figure 2-18. Running employees.sql using File Load