Part II

SQL

SQL is the language for accessing and managing database. Using SQL proficiently is essential for a database programmer. This part introduces fundamentals of SQL, advanced features of SQL, creating database objects, and how to use SQL to manage transactions, control concurrency, and maintain security.

CHAPTER 4 SQL Basics

Objectives

1. To know the history of SQL.
2. To create, modify, and drop tables.
3. To become familiar with SQL data types: varchar2, char, number, and date.
4. To be able to define constraints and know the differences between named constraints and unnamed constraints.
5. To know how to obtain table information and display constraints.
6. To know how to display and enter date values.
7. To learn to use simple SQL statements for queries and update operations.
8. To become familiar with aggregate functions: count, min, max, avg, and sum.
9. To become familiar with Oracle number functions, character functions, date functions, and conversion functions.
10. To know how to use the operators: like, between-and, and is null.
11. To know how to use the order by clause, group by clause, and having clause.

4.1 Introduction

In the preceding chapter, you learned how to analyze data and relationships among data using the ER models, how to convert ER models into the relation schemas, and how to further improve database design using normalization. Now you are ready to create a database and manipulate data in the database using SQL. This chapter and the next two chapters introduce SQL.

4.2 History of SQL

SQL is an acronym for Structured Query Language. It is pronounced "S-Q-L" or "sequel." SQL has its roots in the *System R* – an experimental relational database system developed by IBM in the early 80s. The language used in System R was called *Sequel* on which the SQL language was based. There are three versions of SQL: *SQL1*, *SQL2*, and *SQL3*.

SQL1 was approved by ANSI (American National Standards Institute) in 1986, and adopted by ISO (International Standards Organization) in 1987. It was updated in 1989 with an addendum on integrity enhancements. Almost all current RDBMS (Relational Database Management System) support SQL1.

SQL2, adopted by ISO in 1992, contains many new features. It is an enormous language defined in more than 600 pages. SQL2 is so ambitious that no current database systems can support all its full features. So three levels of SQL2 were proposed: *entry level*, *intermediate level*, and *full level*. The entry level is SQL1 plus the additional improvements such as data types, simple functions, and schema definition statements. The intermediate level is the entry level plus some language support such as embedded SQL, dynamic SQL, and PSM (Persistent Stored Modules). PSM enables the user to create procedures and functions that can be used in the queries. The full level is the entire SQL2. None of the commercial RDBMS supports the SQL2 full level. Most of them support the SQL2 entry level with proprietary extensions. For example, Oracle has its own extensions for stored procedures and functions. Despite of the differences in SQL dialects, the standard core SQL remains the same.

SQL3, adopted in 1999 by ISO, is the latest standard that incorporates object-oriented concept into relational database schemas.

NOTE: The discussion of this chapter is based on SQL2. Some of the new features of SQL3 that are implemented in Oracle will also be introduced.

4.3 Creating, Modifying and Dropping Tables

Tables are the essential objects in the database. To create a table, use the create table statement to specify a table name, attributes and their types, primary keys, foreign keys, and constraints, as in the following example:

```
create table Course(
  courseId char(5),
  subjectId char(4) not null,
  courseNumber number(4),
  title varchar2(50) not null,
  numOfCredits number(1)
    constraint ckNumOfCredits check (numOfCredits >= 1),
  constraint pkCourse primary key (courseId),
  constraint fkSubjectId foreign key (subjectId)
    references Subject(subjectId));
```

This statement creates the Course table with attributes courseId, subjectId, courseNumber, title, and numOfCredits. ckNumOfCredits, pkCourse, and fkSubjectId are the constraint names to denote the constraints on numOfCredits, the primary key and a foreign key, respectively.

> NOTE: SQL is not case-sensitive except the string literals enclosed in the single quotation marks.

> NOTE: The statements for creating all the tables in the sample student information system are included in Appendix A, "Student Information Database Schema and Contents."

4.3.1 Data Types

Each attribute has a data type that specifies the type of data stored in the attribute. Table 4.1 lists some frequently used data types in Oracle.

Table 4.1

Oracle Data Types

Type	**Description**
varchar2(size)	variant-length character data (maximum 4000 characters)
char(size)	fixed-length character data (maximum 2000 characters)
number(p)	integer value of precision p
number(p, s)	floating-point value of precision p and scale s
date	date and time value
clob	character large object
blob	binary large object
XMLType	XML document

The varchar2 type is appropriate for storing strings of variant-length. For example, the course title is a string whose maximum size is 25, thus it can be defined as varchar2(25). Only the actual string is stored in the database. If the size of the string is larger than the specified size, an error occurs.

> NOTE: Oracle also supports the varchar type, which is same as varchar2 in Oracle 9i. In the future version, varchar might be used as a separate data type to define variable-length character strings with different comparison semantics. You should use varchar2 in Oracle 9i.

The char type is appropriate for storing strings of fixed-length. For example, the social security number is a fixed 9-character long string, thus it can be defined as char(9). If a string is shorter than its declared size, spaces are appended to the end of the string to make its length equal to the declared length. If a string is larger than its declared size, an error occurs.

The number type is for storing numerical values. You use number(p) to store an integer with maximum of p digits, number(p, s) to store a fixed-size floating-point number with maximum of p - s digits before the decimal point and maximum of s digits after the decimal point, use number to store floating-point numbers of any size. If a value exceeds the number of digits allowed before the decimal point, Oracle returns an error. If a value exceeds the scale, Oracle rounds it. If the scale is negative, the actual data is rounded to the specified number of places to the left of the decimal point. Here are some examples:

Table 4.2

Number Type Examples

Actual Data	**Data Type Specified As**	**Stored As**
123.456	number(6)	123
123.456	number(6, 1)	123.5
123.456	number(6, 2)	123.47
123.456	number(6, 3)	123.456
123.456	number(6, 4)	exceeds precision
123.456	number(2, -1)	120
123.456	number(2, -2)	100
123.456	number(2, -3)	0

The date type is for storing date and time value. The year, month, day, hour, minute, and second are stored in a date value. Section 4.5, “Entering and Displaying Date Values,” introduces the date type.

The clob type is for storing a large text in the character format. It can be used to store up to 2 GB characters. The blob type is for storing binary data such as image. These two types were introduced in SQL3. The examples of using the blob type will be given in Chapter 9, “Advanced JDBC.”

The XMLType is a new Oracle data type for storing XML documents. XML will be introduced in Chapter 12, “XML.”

4.3.2 Constraints

Constraints are used to define the primary keys, foreign keys, and restrictions on the attributes. Constraints can be defined with or without names. If names are used, they must be distinctive in a table. If a constraint is violated, the constraint name is returned and can be used to identify the violation. If the constraint is not named, it is difficult to identify the violation. Thus, I recommend that you use named constraints. The following are three examples of the named constraints:

```
constraint ckNumOfCredits check (numOfCredits >= 1)
constraint pkCourse primary key (courseId)
constraint fkSubjectId foreign key (subjectId)
  references Subject(subjectId));
```

The first constraint named ckNumOfCredits defines a constraint on the attribute numOfCredits. This type of constraint is known as an *attribute constraint* or a *column constraint*. The second constraint named pkCourse defines a primary key. This is known as the *primary key constraint*. Each table can have only one primary key constraint. The third constraint named fkSubjectId defines a foreign key. Each table may have many foreign keys.

> NOTE: In the relational database theory, no duplicate tuples are allowed. In practice, however, all RDBMS allows duplicates. Therefore, it is permissible to create a table without primary keys. To enforce no duplicates, specify a primary key or a unique attribute in the table.

4.3.2.1 Column-Level and Table-Level Constraints

A constraint can be defined when a table is created, or added later. It can also be dropped or modified. Constraints can be defined at *column-level* or *table-level*.

A column-level constraint specifies a constraint on a single column and it is defined along with the definition of the column. Any constraint that involves a single attribute can be defined at column-level. The following constraints are defined at column-level:

```
courseId char(5) constraint pkCourse primary key,
subjectId char(4)
  references Subject(subjectId),
numOfCredits number(1)
  constraint ckNumOfCredits check (numOfCredits >= 1)
```

A table-level constraint specifies a constraint on one or more columns, and it is defined separately from the definitions of columns. If a constraint references more than one column, it must be defined at table-level. A table-level constraint can be defined after all its referencing columns are defined, and it is normally defined after all columns are defined. The preceding three constraints can be defined at table-level as follows:

```
constraint pkCourse primary key (courseId),
constraint fkSubjectId foreign key (subjectId)
  references Subject(subjectId),
constraint ckNumOfCredits check (numOfCredits >= 1)
```

> NOTE: Every column-level constraint can be defined at table-level. The syntax of column-level constraints is slightly different from the table-level constraints.

4.3.2.2 Attribute Constraints

Three types of attribute constraints can be specified: *not null constraint*, *unique key constraint*, and *check constraint*.

A not null constraint specifies that the attribute value cannot be null. null is a special value in the database, meaning not known or not applicable. If an attribute is part of the primary key, it automatically has the not null constraint. The following clause specifies that subjectId is not null.

```
subjectId char(4) constraint nnSubjectId not null
```

A unique key constraint specifies that every tuple has a distinct value on an attribute or a set of attributes. The unique attribute or the set of attributes is called a *unique key*. The following clause specifies that title is unique.

```
title varchar2(50) constraint ukTitle unique (title)
```

The following clause specifies that subjectId and courseNumber together form a unique key:

```
constraint uk unique (subjectId, courseNumber)
```

A check constraint specifies a rule on a single column. A column can have more than one check constraints. The following clause specifies that title is unique.

```
title varchar2(50) constraint ukTitle unique (title)
```

The following clause specifies that subjectId and courseNumber together form a unique key:

```
constraint uk unique (subjectId, courseNumber)
```

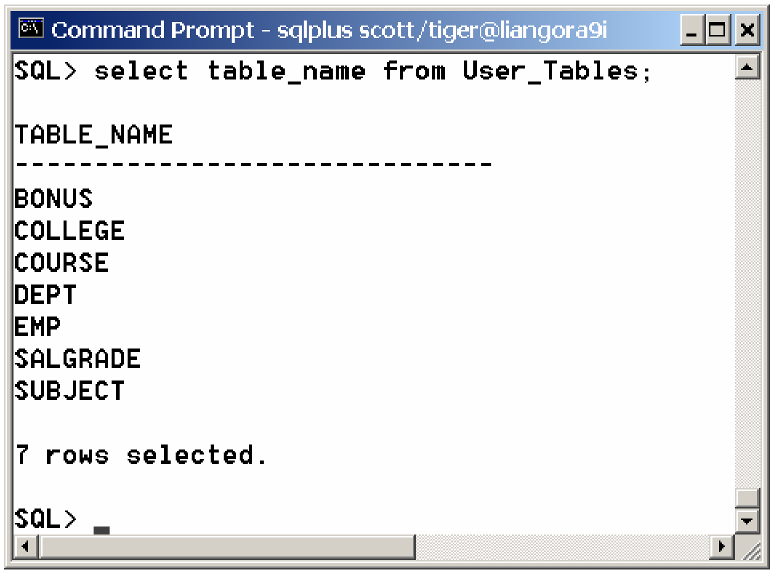
4.3.3 Default Values

You can specify a default value on an attribute. The default value is used when a null value is inserted. The following clause specifies the default value for numOfCredits is 3.

```
numOfCredits number(1) default 3
```

4.3.4 Displaying Table Information

Database not only stores the contents of the tables, but also the definition of the tables. User tables are stored in a system table named User_Tables. Whenever a table is created, the table name is added to the table by the system. To find all tables created by the user, use the command select table_name from User_Tables as shown in Figure 4.1.

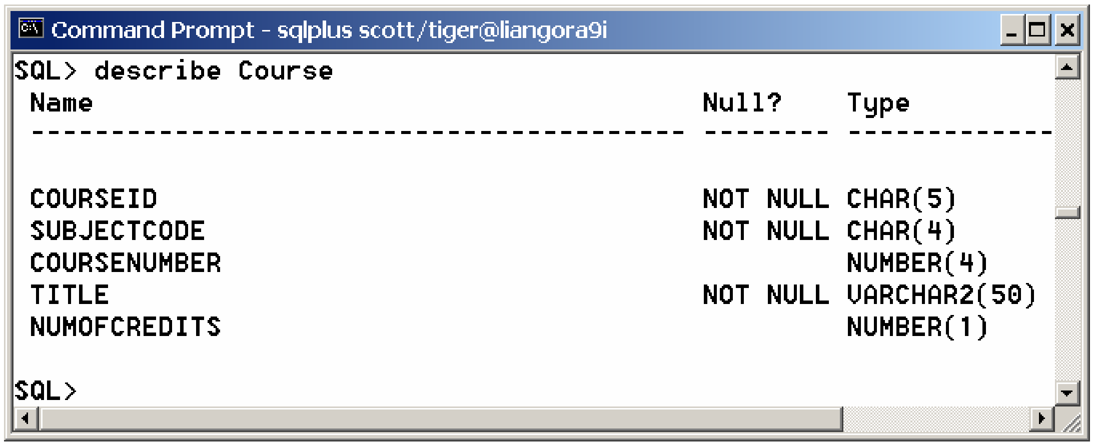


```
SQL> select table_name from User_Tables;

TABLE_NAME
------------------------------
BONUS
COLLEGE
COURSE
DEPT
EMP
SALGRADE
SUBJECT

7 rows selected.

SQL> _
```

Figure 4.1

You can retrieve table names from the User_Tables table.

To display the column definition in a table, use the describe command in Oracle. Figure 4.2 shows the information for the Course table. The command displays table column names, data types and whether columns can be null.



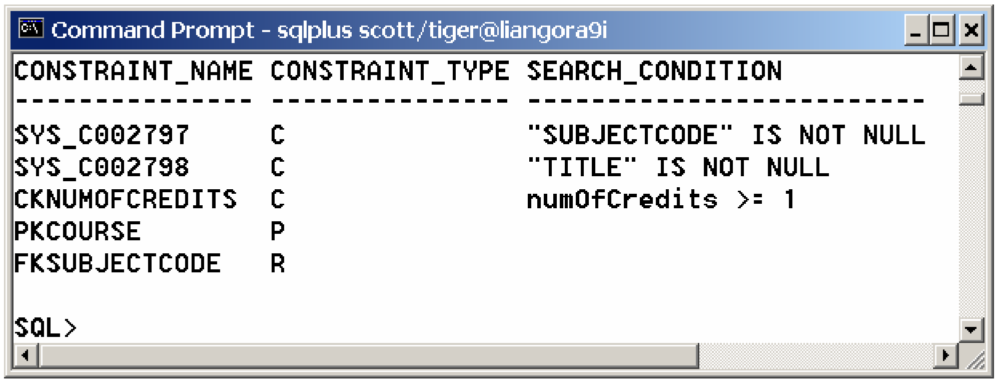
```
SQL> describe Course
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------

 COURSEID                                  NOT NULL CHAR(5)
 SUBJECTCODE                               NOT NULL CHAR(4)
 COURSENUMBER                                       NUMBER(4)
 TITLE                                     NOT NULL VARCHAR2(50)
 NUMOFCREDITS                                       NUMBER(1)

SQL>
```

Figure 4.2

You can use the describe command to display table information.

The information on constraints is stored in a system table named User_Constraints. The following statements display the constraint names, constraint types, and search conditions for all constraints in the Course table, as shown in Figure 4.3.

```
column constraint_name format a15;
column constraint_type format a15;
column search_condition format a25;
select constraint_name, constraint_type, search_condition
from User_Constraints
where table_name = 'COURSE';
```



```
CONSTRAINT_NAME CONSTRAINT_TYPE SEARCH_CONDITION
--------------- --------------- -------------------------
SYS_C002797     C               "SUBJECTCODE" IS NOT NULL
SYS_C002798     C               "TITLE" IS NOT NULL
CKNUMOFCREDITS  C               numOfCredits >= 1
PKCOURSE        P
FKSUBJECTCODE   R

SQL>
```

Figure 4.3

You can view constraints information from the User_Constraints table.

The statement

```
column constraint_name format a15;
```

is an Oracle-proprietary command that specifies that the constraint_name should be displayed in width of 15 characters. The column command will be further discussed in Section 4.?, "Formatting Columns."

If a constraint is not named, Oracle automatically assigns a name internally. For example, SYS_C002797 is assigned automatically by Oracle for constraints "SUBJECTCODE" IS NOT NULL. Constraint type C stands for column constraints, P stands for primary key constraints, and R stands for foreign key constraints.

> NOTE: All the database meta data is stored in upper case. Thus, table Course is stored as COURSE in the User_Constraints table. Therefore, you have to use the string 'COURSE' in the query.

4.3.5 Dropping, Renaming and Truncating Tables

If a table is no longer needed, it can be dropped permanently using the drop table command. For example, the following statement drops the Course table.

```
drop table Course;
```

If a table to be dropped is referenced by other tables, you can use the cascade constraints clause in the drop table command to force the table to be drop. For example, the following statement drops the Subject table and all referential integrity constraints that refer to the primary key in the Subject table.

```
drop table Subject cascade constraints;
```

If you omit this clause, Oracle returns an error and does not drop the table, since the Subject table is referenced by other tables.

You can remove all rows in a table using the truncate table statement. For example, the following statement removes all rows from the Course table.

```
truncate table Course;
```

NOTE: Removing rows with the truncate statement can be more efficient than dropping and re-creating a table. Dropping and re-creating a table invalidates the table's dependent objects, requires you to re-grant object privileges on the table, and requires you to re-create the table's indexes, integrity constraints, and triggers and re-specify its storage parameters. Truncating has none of these effects.

You can rename a table using the rename table statement. For example, the following statement renames Course to NewCourse.

```
rename Course to NewCourse;
```

All the referential integrity constraints that are dependent on the renamed table are automatically modified to reference the new table.

4.3.6 Altering Table Definitions

Occasionally, you need to alter table definitions to accommodate changes in the database. You can add or drop a column, add or drop a constraint, enable or disable a constraint, and increase the size of an attribute.

The following command adds a new column into the Course table.

```
alter table Course add newTitle varchar2(50);
```

You can drop a column provided it is not the only column in the table. The following command drops an existing column named newTitle from the Course table.

```
alter table Course drop column newTitle;
```

The following command adds a new constraint to the Course table.

```
alter table Course
  add constraint uk unique (subjectId, courseNumber);
```

The following command drops an existing constraint from the Course table.

```
alter table Course drop constraint uk;
```

The following command modifies the data type of numOfCredits to number(5) with default value 3.

```
alter table Course modify numOfCredits number(5) default 3;
```

A constraint can be disabled or enabled, as shown in the following examples:

```
alter table Course disable constraint fkSubjectId;
alter table Course enable constraint fkSubjectId;
```

NOTE: If an altering table command affects the constraints in other tables, Oracle will generate an error and no change is done. To force the change, use the cascade constraints clause. For example, the following command forces the primary key column subjectId to be dropped from the Subject table and the foreign key constraint in the Course table that references subjectId is also dropped.

```
alter table Subject drop column subjectId cascade constraints;
```

End of NOTE

TIP: If two tables T1 and T2 reference each other in the foreign keys. You can create the table without specifying foreign keys and later add the foreign key constraints using the alter table command.

4.4 Simple Insert, Update and Delete

Once a table is created, you can insert records into the table. You can also update and delete records. This section introduces simple insert, update and delete statements. More advanced features will be introduced in Chapter 5, “Advanced SQL.”

The general syntax to insert a record into a table is

```
insert into tableName [(column1, column2, …, column]]
  values (value1, value2, …, valuen);
```

For example, the following statement inserts a record into the Course table. The new record has the courseId '11113', subjectId 'CSCI', courseNumber 3720, title 'Database Systems', and creditHours 3.

```
insert into Course (courseId, subjectId, courseNumber, title)
  values ('11113', 'CSCI', '3720', 'Database Systems', 3);
```

The column names are optional. If the column names are omitted, all column values for the record must be entered even though the columns have default values. String values are case-sensitive and enclosed inside single quotation marks.

The general syntax to update a table is

```
update tableName
  set column1 = newValue1 [, column2 = newValue2, ...]
  [where condition];
```

For example, the following statement changes the numOfCredits for the course whose title is Database Systems to 4.

```
update Course
  set numOfCredits = 4
  where title = 'Database Systems';
```

The general syntax to delete the records in a table is

```
delete [from] tableName
  [where condition];
```

For example, the following statement deletes the Database Systems course from the Course table:

```
delete Course
  where title = 'Database System';
```

The following statement deletes all records from the Course table:

```
delete Course;
```

NOTE: You need to issue the commit command to end the current transaction and make permanent all changes by the insert, update, and delete operations. The truncate command is a DDL command. All DDL commands are committed once they are executed.

NOTE: The ampersand (&) is a special Oracle character. It can be used to define variables in SQL. If you enter a string that consists of an ampersand (&) character, use the set define off command to suspend the special character from SQL*Plus. The special character cannot be turned off from iSQL*Plus.

4.5 Entering and Displaying Date Values

You can enter a date value using the default format or a custom format. The default date format for entering date is 'DD-MON-YY' or 'DD-MON-YYYY', where DD represents the day of month, MON represents the first three letters of the month capitalized, YY represents the last two digits of the year, and YYYY represents the full year value. By default, the time is 12:00:00 AM. The following statement inserts birth date October 9, 1969 for a student:

```
insert into Student (ssn, birthDate)
  values ('444112222', '9-OCT-69');
```

NOTE: When you use the format 'DD-MON-YY' to enter a date, if YY is less than 50, YY is in the current century; if YY is greater or equal to 50, YY is in the previous century. So, '9-OCT-49' represents 9-OCT-2049, but '9-OCT-69' represents '9-OCT-1969'.

The default date format for displaying date is 'DD-MON-YY'. You can enter or display date in a custom format using the date functions to_date and to_char.

The to_date function converts a string to a date equivalent. For example, the following statement inserts a complete date and time using the to_date function with the format 'YYYY MM DD HH24:MI:SS':

```
insert into Student (ssn, birthDate)
  values ('444112222',
    to_date('1969 10 9 17:35:24', 'YYYY MM DD HH24:MI:SS'));
```

The to_char function converts a date to a string. For example, the following statement displays date in the format 'YYYY MM DD HH24:MI:SS':

```
select ssn, to_char(birthDate, 'YYYY MM DD HH24:MI:SS')
from Student;
```

The format string consists of the date and time elements supported in Oracle. Section 4.7.2.4, “Conversion Functions,” discusses the date format in details.

NOTE: You can set the default format using the alter session command. For example, the following Oracle statement sets the default format to 'YYYY MM DD HH24:MI:SS'.

```
alter session
  set nls_date_format = 'YYYY MM DD HH24:MI:SS';
```

Now you can insert a date using the default format as follows:

```
insert into Student (ssn, birthDate)
  values ('444112222', '1969 10 9 17:35:24');
```

End of NOTE

4.6 Simple Queries

To retrieve information from tables, use the select statement with following syntax:

```
select column-list
  from table-list
  [where condition];
```

The select clause lists the columns to be selected. The from clause refers to the tables involved in the query. The where clause specifies the conditions for the selected rows.

There are many options and flavors in a select statement. This chapter introduces simple queries. Complex queries will be introduced in the next chapter.

4.6.1 Comparison and Boolean Operators

SQL has the six comparison operators, as shown in Table 4.3, and three Boolean operators, as shown in Table 4.4.

Table 4.3

Comparison Operators

Operator	**Description**
=	Equal to
<> or !=	Not equal to
<	Less than
<=	Less or equal to
>	Greater than
>=	Greater than

Table 4.4

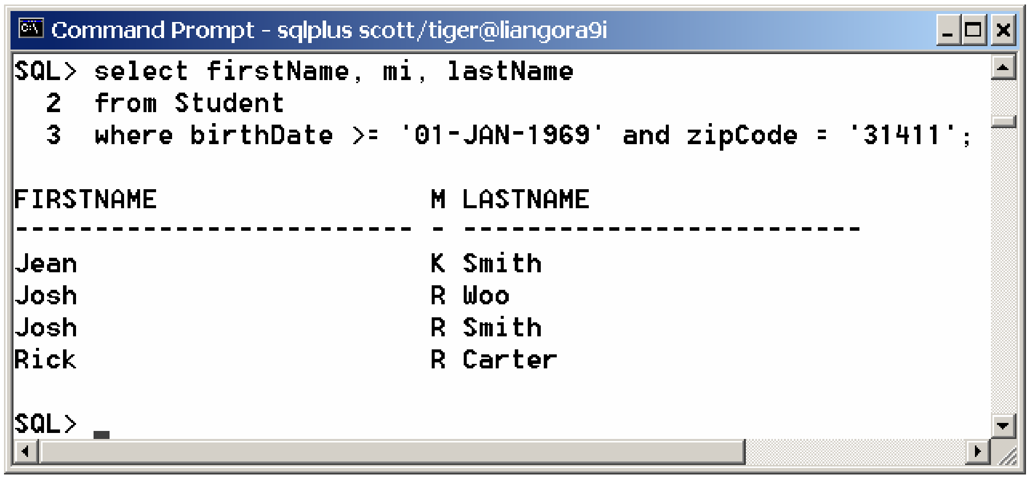
Boolean Operators

Operator	**Description**
not	logical negation
and	logical conjunction
or	logical disjunction

> NOTE: The comparison and Boolean operators in SQL have the same meaning as in Java. In SQL the equals to operator is =, but it is == in Java. In SQL the not equal to operator is <> or !=, but it is != in Java. The not, and, and or operators are !, && (&), and || (|) in Java.

Example 4.1: Get the names of the students who were born after 1969 and live in the zip code 31411. The query result is shown in Figure 4.4.

```
select firstName, mi, lastName
from Student
where birthDate >= '01-JAN-1969' and zipCode = '31411';
```


Command Prompt - sqlplus scott/tiger@liangora9i

```
SQL> select firstName, mi, lastName
  2  from Student
  3  where birthDate >= '01-JAN-1969' and zipCode = '31411';

FIRSTNAME                 M LASTNAME
------------------------- - -------------------------
Jean                      K Smith
Josh                      R Woo
Josh                      R Smith
Rick                      R Carter

SQL> _
```

Figure 4.4

Example 4.1 demonstrates using comparison and Boolean operators.

birthDate is compared with '01-JAN-1969'. '01-JAN-1969' is not a string, but a date with the default format 'DD-MON-YYYY'. zipCode is a string, which is compared with '31411'.

> NOTE: To select all attributes from a table, you don't have to list all attribute names in the select clause. You can just specify an *asterisk* (*), which stands for all attributes. For example, the following query displays all attributes of the students who were born after 1969 and live in the zip code 31411:
>
> ```
> select *
> from Student
> where birthDate >= '01-JAN-1969' and zipCode = '31411';
> ```

******End of NOTE***

4.6.2 The like, between-and, and is null Operators

SQL has the like operator that can be used for pattern matching. The syntax to check if a string s has a pattern p is

s like p or s not like p

You can use wild card characters % (percent symbol) and _ (underline symbol) in the pattern p. % matches zero or more characters and _ matches any single character in s. For example, lastName like '_mi%' matches any string with the second and third letters being m and i. lastName not like '_mi%' excludes any string whose the second and third letters are m and i.

The between-and operator checks whether a value v is between two other values v1 and v2 using the following syntax:

v between v1 and v2 or v not between v1 and v2

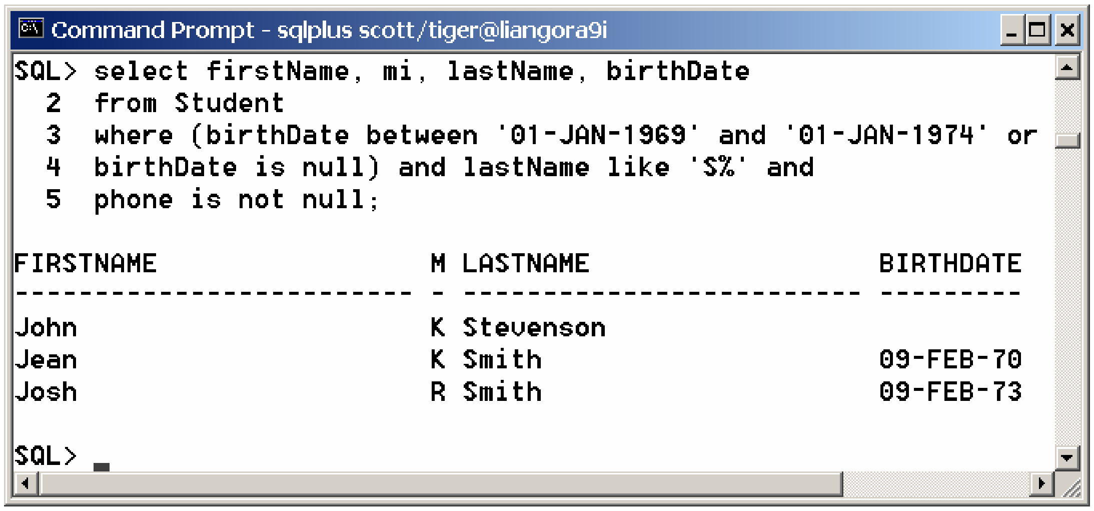
v between v1 and v2 is equivalent to v >= v1 and v <= v2 and v not between v1 and v2 is equivalent to v < v1 and v > v2.

The is null operator checks whether a value v is null using the following syntax:

v is null or v is not null or

Example 4.2: Get the name and birth date of the students who were born between 1969 and 1974 or unknown, last name begins with the letter 'S', and phone is not null. The query result is shown in Figure 4.5.

```
select firstName, mi, lastName, birthDate
from Student
where (birthDate between '01-JAN-1969' and '01-JAN-1974' or
birthDate is null) and lastName like 'S%' and phone is not null;
```



```
SQL> select firstName, mi, lastName, birthDate
  2  from Student
  3  where (birthDate between '01-JAN-1969' and '01-JAN-1974' or
  4  birthDate is null) and lastName like 'S%' and
  5  phone is not null;

FIRSTNAME                 M LASTNAME                  BIRTHDATE
------------------------- - ------------------------- ---------
John                      K Stevenson
Jean                      K Smith                     09-FEB-70
Josh                      R Smith                     09-FEB-73

SQL> _
```

Figure 4.5

Example 4.2 demonstrates the like, between-and and is null operators.

4.6.3 Column Alias

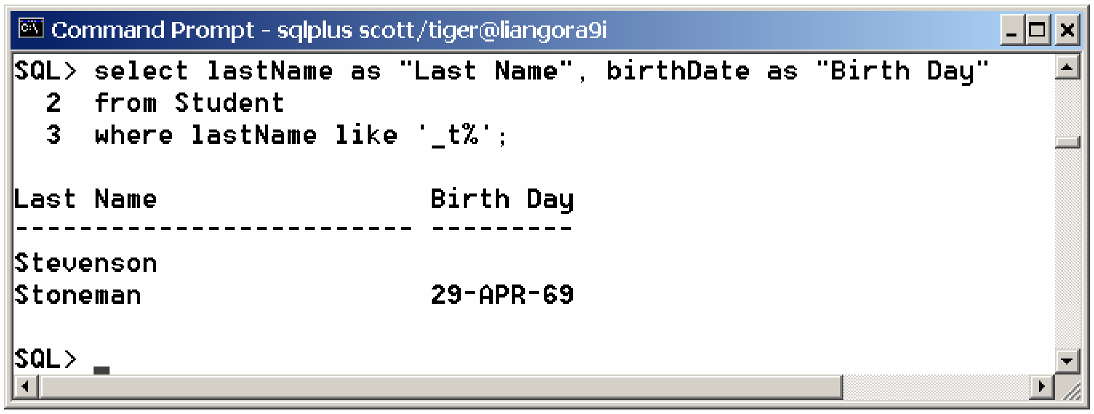
When a query result is displayed, SQL uses the column names as the column heading. Usually the user gives abbreviated names for the columns and the columns cannot have spaces when the table is created. Sometime, it is desirable to give more descriptive names in the result heading. You can use the column aliases with the following syntax:

```
select columnName [as] alias
```

Example 4.3: Get the last name and birth date of the students whose last name’s second letter is t. Display the column heading as Last Name and Birth Day. The query result is shown in Figure 4.6.

```
select lastName as "Last Name", birthDate as "Birth Day"
```

```
from Student
where lastName like '_t%';
```



Command Prompt - sqlplus scott/tiger@liangora9i

```
SQL> select lastName as "Last Name", birthDate as "Birth Day"
  2  from Student
  3  where lastName like '_t%';

Last Name                 Birth Day
------------------------- ---------
Stevenson
Stoneman                  29-APR-69

SQL> _
```

Figure 4.6

Example 4.3 demonstrates the column alias.

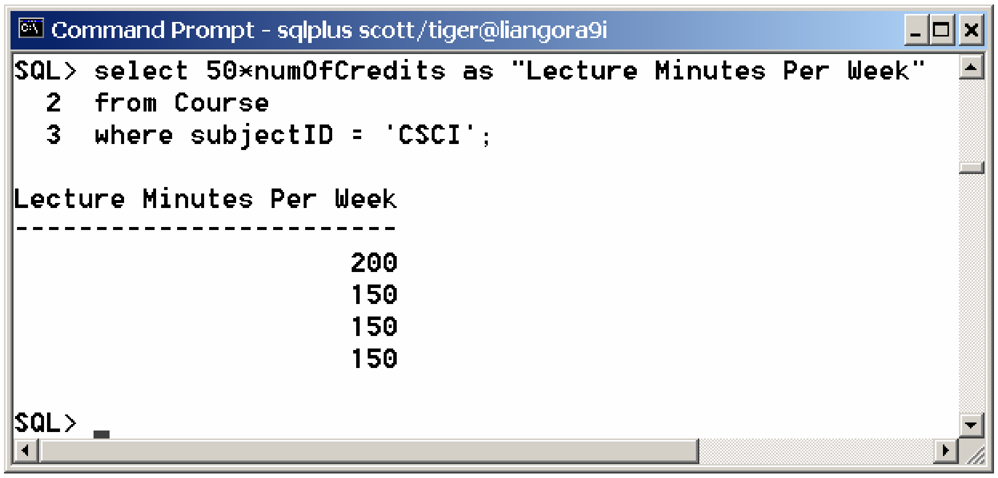
> NOTE: The as keyword is optional. An alias can also appear without the quotation marks. For clarity, I recommend to use the as keyword with the alias in the quotation marks.

4.6.4 The Arithmetic Operators

You can use the arithmetic operators * (multiplication), / (division), + (addition), and - (subtraction) in SQL.

Example 4.4: Assume each credit hour is 50 minutes of lectures, get the total minutes for each course with the subject CSCI. The query result is shown in Figure 4.7.

```
select title, 50 * numOfCredits as "Lecture Minutes Per Week"
from Course
where subjectId = 'CSCI';
```



Command Prompt - sqlplus scott/tiger@liangora9i

```
SQL> select 50*numOfCredits as "Lecture Minutes Per Week"
  2  from Course
  3  where subjectID = 'CSCI';

Lecture Minutes Per Week
------------------------
                     200
                     150
                     150
                     150

SQL> _
```

Figure 4.7

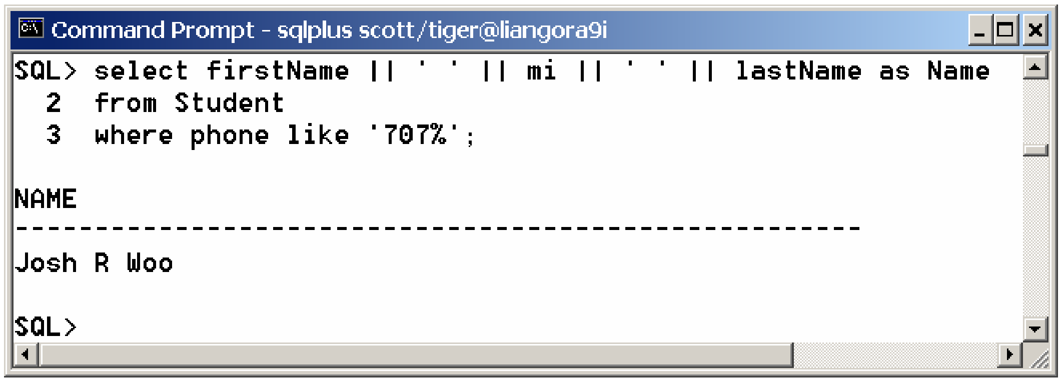
Example 4.4 demonstrates the arithmetic operators.

4.6.5 The Concatenation Operator (||)

SQL provides the concatenation operator (||) that can be used to combine columns with numbers and strings. For example, you can display last name, mi, and first name together in one string, instead of three strings.

Example 4.5: List the full name of the students whose area code of the phone number is 707. The query result is shown in Figure 4.7.

```
select firstName || ' ' ||  mi || ' ' || lastName as Name
from Student
where phone like '707%';
```



```
SQL> select firstName || ' ' || mi || ' ' || lastName as Name
  2  from Student
  3  where phone like '707%';

NAME
-----------------------------------------------------
Josh R Woo

SQL>
```

Figure 4.7

Example 4.5 demonstrates the concatenation operator.

4.6.6 Displaying Distinct Tuples

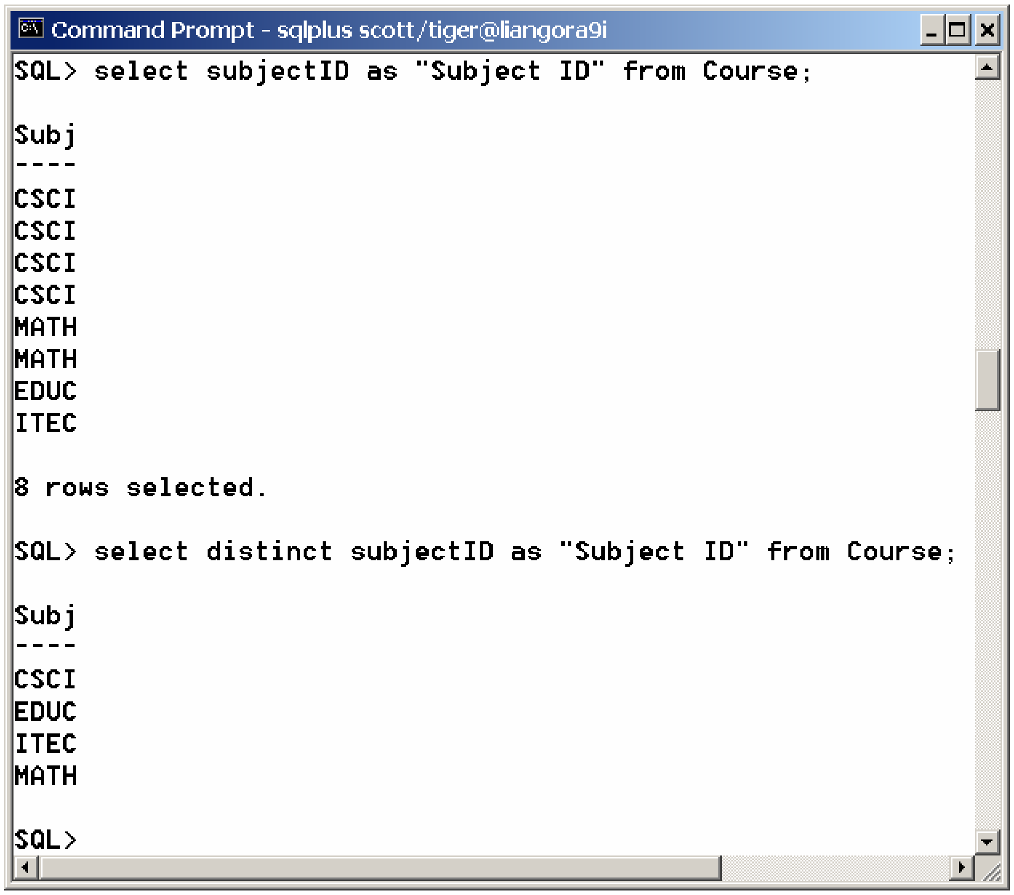
SQL provides the distinct keyword that can be used to suppress duplicate tuples in the output. For example, the following statement displays all subject IDs that are used by the courses.

```
select subjectId as "Subject ID"
from Course;
```

This statement displays all subject ID, as shown in Figure 4.8. To display distinct tuples, add the distinct keyword in the select clause as follows:

```
select distinct subjectId as "Subject ID"
  from Course;
```

This statement displays distinct subject ID, as shown in Figure 4.8.

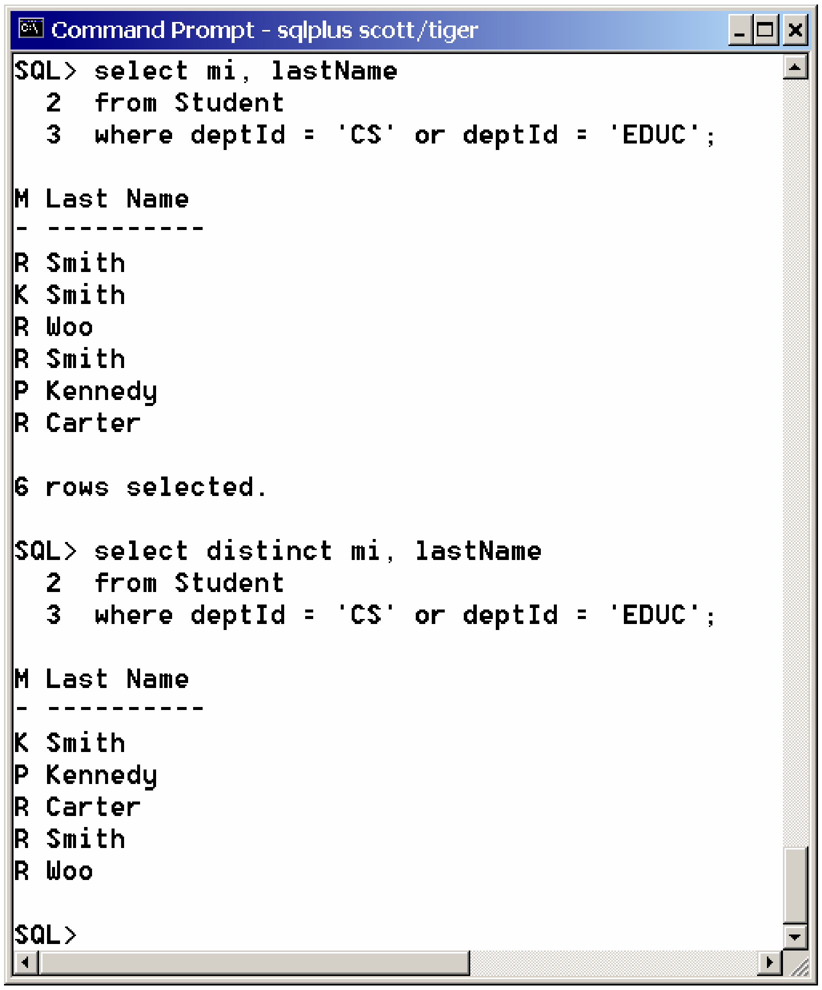


```
SQL> select subjectID as "Subject ID" from Course;

Subj
----
CSCI
CSCI
CSCI
CSCI
MATH
MATH
EDUC
ITEC

8 rows selected.

SQL> select distinct subjectID as "Subject ID" from Course;

Subj
----
CSCI
EDUC
ITEC
MATH

SQL>
```

Figure 4.8

You can use the distinct keyword to suppress duplicates.

When there is more than one item in the select clause, the distinct keyword applies to all the items to find the distinct tuples. As shown in Figure 4.9, the first statement displays mi and lastName for all students in the CS or EDUC departments with duplicates and the second statement displays the same tuples without duplicates.



```
SQL> select mi, lastName
  2  from Student
  3  where deptId = 'CS' or deptId = 'EDUC';

M Last Name
- ----------
R Smith
K Smith
R Woo
R Smith
P Kennedy
R Carter

6 rows selected.

SQL> select distinct mi, lastName
  2  from Student
  3  where deptId = 'CS' or deptId = 'EDUC';

M Last Name
- ----------
K Smith
P Kennedy
R Carter
R Smith
R Woo

SQL>
```

Figure 4.9

The distinct keyword applies to all items in the select clause.

> NOTE: By definition null values are unknown and cannot be compared with each other, but distinct treats null values are the same. Therefore, if multiple selected tuples are null, only one is displayed.

4.6.7 Displaying Sorted Tuples

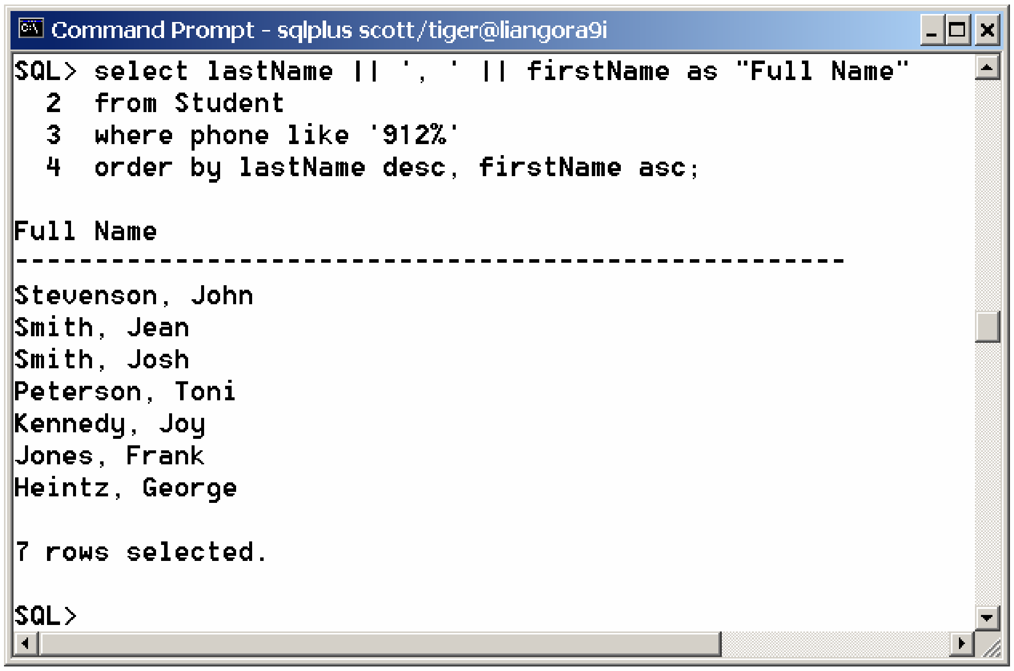
SQL provides the order by clause to sort the output using the following general syntax:

```
select column-list
from table-list
[where condition]
[order by columns-to-be-sorted];
```

In the syntax, columns-to-be-sorted specifies a column or a list of columns to be sorted. By default, the order is ascending. To sort in descending order, append the desc keyword. Similarly, you can append the asc keyword, but it is not necessary. When multiple columns are specified, the rows are sorted based on the first column, then the rows with the same values on the second column are sorted based on the second column, and so on.

Example 4.6: List the full name of the students whose area code of the phone number is 912, ordered primarily on the last name in descending order and secondarily on the first name in ascending order. The query result is shown in Figure 4.10.

```
select lastName || ', ' || firstName as "Full Name"
from Student
where phone like '912%'
order by lastName desc, firstName asc;
```



```
SQL> select lastName || ', ' || firstName as "Full Name"
  2  from Student
  3  where phone like '912%'
  4  order by lastName desc, firstName asc;

Full Name
----------------------------------------------------
Stevenson, John
Smith, Jean
Smith, Josh
Peterson, Toni
Kennedy, Joy
Jones, Frank
Heintz, George

7 rows selected.

SQL>
```

Figure 4.10

Example 4.6 demonstrates the order by clause.

NOTE: The *columns-to-be-sorted* list may contain any columns in the table, not necessarily to be in the selected *column-list*.

NOTE: When null values are sorted, they are ordered last in Oracle.

4.7 SQL Functions

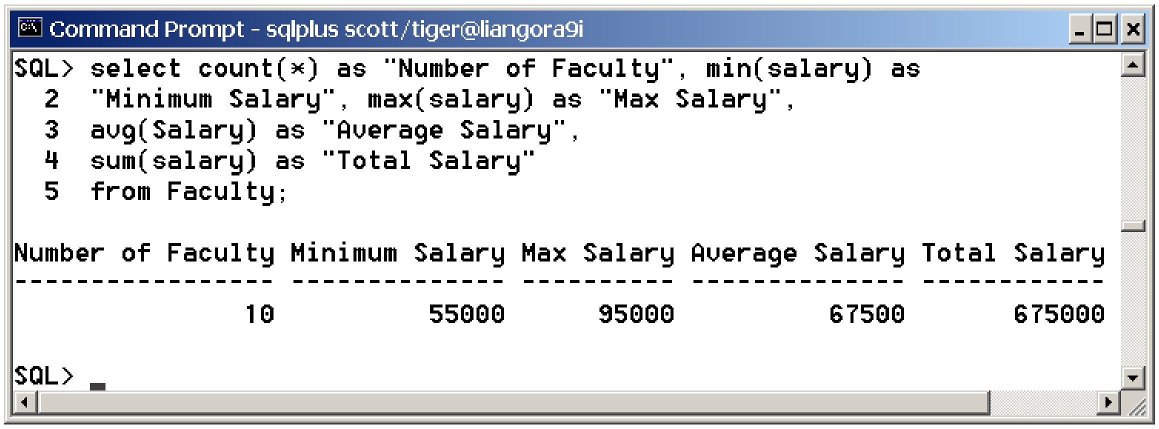
SQL functions can be used in SQL query to extend the power of SQL. A SQL function takes zero or more arguments and returns a single value. There are five standard SQL functions supported by all RDBMS. Oracle supports many other useful functions.

4.7.1 Standard SQL Functions

Five standard SQL functions are count, max, min, avg, and sum. count returns the number of rows, max, min, avg, and sum find the maximum, minimum, average, and sum of all values in a column ignore null values. These functions are known as *aggregate functions* or *group functions* because they operate on a group of rows and return a single value.

Example 4.7: List the number of faculty, minimum salary, maximum salary, average salary, and total salary. The query result is shown in Figure 4.11.

```
select count(*) as "Number of Faculty", min(salary) as
  "Minimum Salary", max(salary) as "Max Salary", avg(salary)
  as "Average Salary",
  sum(salary) as "Total Salary"
from Faculty;
```



```
SQL> select count(*) as "Number of Faculty", min(salary) as
  2  "Minimum Salary", max(salary) as "Max Salary",
  3  avg(Salary) as "Average Salary",
  4  sum(salary) as "Total Salary"
  5  from Faculty;

Number of Faculty Minimum Salary Max Salary Average Salary Total Salary
----------------- -------------- ---------- -------------- ------------
               10          55000      95000          67500       675000

SQL> _
```

Figure 4.11

Example 4.7 demonstrates standard SQL aggregate functions.

NOTE: If no row meets the query condition, count returns 0, and the other functions all return null.

NOTE: You can use distinct with any aggregate functions except count(*). For example,

```
select sum(distinct salary)
from Faculty;
```

finds the average of distinct salary. Note there is no need to use distinct with min and max functions because the maximum distinct salary is the same as the maximum salary.

******End of NOTE***

NOTE: The null values are ignored in the aggregate functions except <u>count(*)</u> that counts each row including the row with null values.

NOTE: The aggregate functions cannot be used in the <u>where</u> clause. For example, the following statement is erroneous.

```
select lastName
from Faculty
where salary = max(salary);
```

End of NOTE

4.7.2 Oracle SQL Functions

Oracle provides many proprietary functions to extend the power of SQL. These functions can be classified into number functions, character functions, date functions, conversion functions, and miscellaneous functions. These functions are known as *single-row functions* because they operate on a single value or a single column in a row.

4.7.2.1 Number Functions

A number function takes a number or a column of a numeric type as an argument and returns a numeric value. Table 4.5 lists some frequently used number functions.

Table 4.5 Number Functions

Function Name	Description	Example
abs(x)	absolute value of x	abs(-5.5) is 5.5
ceil(x)	smallest integer greater than or equal to x	ceil(-5.5) is -5 ceil(5.5) is 6
cos(x)	trigonometric cosine of x (x in radians)	cos(0) is 1 cos(3.14195/2) is 0
exp(x)	e^x (x in radians)	exp(2) is 7.38906
floor(x)	largest integer less than or equal to x	floor(5.5) is 5 floor(-5.5) is -6
ln(x)	natural logarithm of x (base *e*)	ln(2.718282) is 1
log(m, n)	$log_m n$	log(10, 100) is 2
mod(m, n)	remainder of m divided by n	mod(4, 2) is 0 mod(4, 3) is 1
power(m, n)	m^n	power(2, 3) is 8
round(x)	round x to an integer	round(5.5) is 6
round(x, d)	round x to d places right of the decimal point.	round(5.567, 1) is 5.6 round(5.567, 2) is 5.57
sign(x)	returns 1 if x > 0, 0 if x is 0, and -1 if x < 0	sign(-3) is -1 sign(3) is 1
sin(x)	trigonometric sine of x (x in radians)	sin(0) is 0 sin(3.14195/2) is 1
sqrt(x)	square root of x	sqrt(4) is 2
tan(x)	trigonometric tangent of x (x in radians)	tan(0) is 0 cos(3.14195/2) is 1
trunc(x)	truncate x to an integer	trunc(5.5) is 5

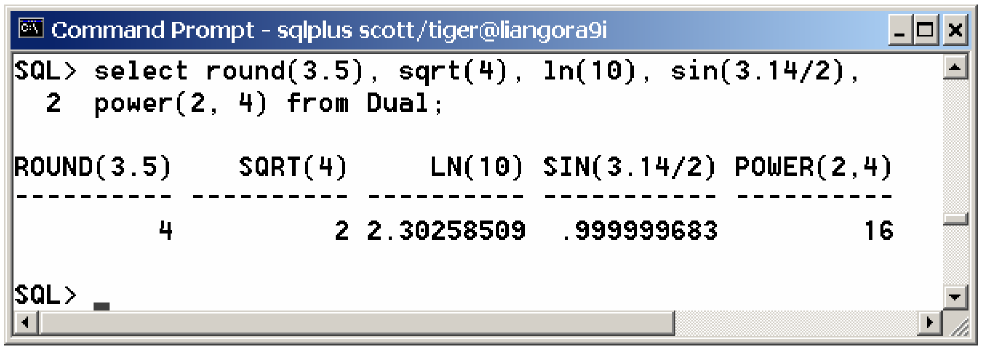
trunc(x, d)	truncate x to d decimal places	trunc(5.546, 2) is 5.54

You can use the functions in an expression. To display the result of an expression, use a select statement as follows:

```
select exp
from tableName;
```

You can use any tables. The result is displayed as many times as the number of the rows in the table. Oracle has a dummy table named Dual, which is automatically created along with the data dictionary. Dual is owned by the user SYS, but is accessible by the name Dual to all users. It has one column named dummy, defined to be varchar2(1), and contains one row with a value 'X'. Selecting from the Dual table is useful for computing an expression with a select statement. Because Dual has only one row, the result of an expression is returned only once. The following statement (shown in Figure 4.12) shows several examples of using number functions.

```
select round(3.5), sqrt(4), ln(10), sin(3.14/2), power(2, 4) from Dual;
```



```
SQL> select round(3.5), sqrt(4), ln(10), sin(3.14/2),
  2  power(2, 4) from Dual;

ROUND(3.5)    SQRT(4)     LN(10) SIN(3.14/2) POWER(2,4)
---------- ---------- ---------- ----------- ----------
         4          2 2.30258509  .999999683         16

SQL>
```

Figure 4.12

You can use the Dual table to display the result of expressions.

4.7.2.2 Character Functions

A character function takes a number, a string or a column of a string type or numeric type as an argument and returns a string or a numeric value. Tables 4.6 and 4.7 list the character functions that return character values and return numerical values, respectively.

Table 4.6 Character Functions Returning Character Values

Function Name	Description	Example
chr(n)	returns a character whose code is n based on the machine's native coding system	On ASCII-based machine, chr(67) is 'C' and on EBCDIC-based machine, chr(195) is 'C'
concat(s1, s2)	combines string s1 with s2. This function is the same as the || operator	concat('Java ', 'Oracle ') is 'Java Oracle'
initCap(s)	converts the first letter	initCap('java oracle') is

	of each word to uppercase, all other letters to lowercase.	'Java Oracle'
lower(s)	converts each letter to lowercase.	lower('Java Oracle') is java oracle
lpad(s1, n, s2)	left-pad s1 with s2 to total width of n	lpad('Java', 8, 'Ja') is 'JaJaJava'
ltrim(s)	removes left blanks	ltrim(' Java ') is 'Java '
replace(s1, s, r)	replaces s with r in s1	replace('Java Oracle', 'a', 'b') is 'Jbvb Orbcle'
lpad(s1, n, s2)	right-pad s1 with s2 to total width of n	rpad('Java', 8, 'Ja') is 'JavaJaJa'
rtrim(s)	removes right blanks	rtrim(' Java ') is ' Java'
trim(s)	removes left and right blanks	trim(' Java ') is 'Java'
upper(s)	converts each letter to uppercase.	upper('Java Oracle') is 'JAVA ORACLE'

Table 4.7 Character Functions Returning Numeric Values

Function Name	Description	Example
ascii(c)	returns an ASCII code for	ascii('a') is 97
instr(s1, s2)	if s2 is a substring of s1, return the index of the first character where a matching occurs. Otherwise, return 0	instr('Java', 'a') is 2 and instr('Java', 'b') is 0
length(s)	returns the length of string s	length('Java Oracle') is 11

4.7.2.3 Date Functions

A date function operates on values of the date type. All date functions return a date or interval value of date type, except the months_between function, which returns a number. Some frequently used date functions are listed in Table 4.8.

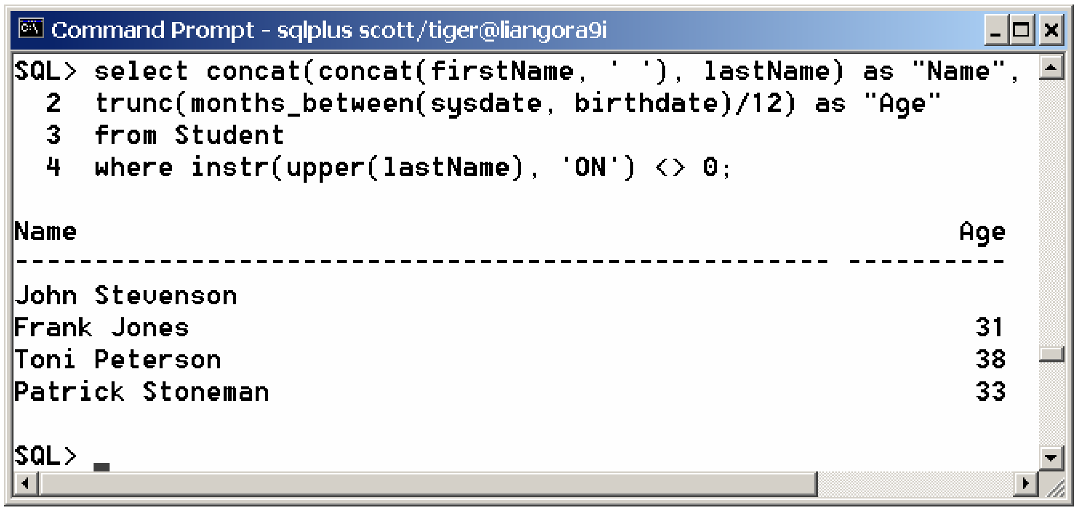
Table 4.8 Date Functions

Function Name	Description	Example
add_months(d, m)	adds number of months m	add_months('9-DEC-2002', 1) returns '9-JAN-03'
current_date	returns the current date	

last_day(d)	returns the last date of the month in date d	last_day('9-DEC-2002') is '31-DEC-2002'
months_between(d1, d2)	returns the number of months between dates d1 and d2	months_between('9-DEC-2001', '9-DEC-2002') is -12 months_between('9-DEC-2002', '9-DEC-2001') is 12
sysdate	returns the current date	
trunc(d, format)	truncates a date d into a specified format	

Example 4.8: Display the ages of all students whose last name has a string 'ON' in lowercase or uppercase. The query result is shown in Figure 4.13.

```
select concat(concat(firstName, ' '), lastName) as "Name",
  trunc(months_between(sysdate, birthdate)/12) as "Age"
from Student
where instr(upper(lastName), 'ON') <> 0;
```


Command Prompt - sqlplus scott/tiger@liangora9i

```
SQL> select concat(concat(firstName, ' '), lastName) as "Name",
  2  trunc(months_between(sysdate, birthdate)/12) as "Age"
  3  from Student
  4  where instr(upper(lastName), 'ON') <> 0;

Name                                                   Age
--------------------------------------------------- ----------
John Stevenson
Frank Jones                                             31
Toni Peterson                                           38
Patrick Stoneman                                        33

SQL> _
```

Figure 4.13

Example 4.8 demonstrates Oracle SQL character functions.

4.7.2.4 Conversion Functions

A conversion function converts a value from one data type to another. Table 4.9 lists some frequently used conversion methods.

Table 4.9 Conversion Functions

Function Name	**Description**	**Example**
to_char(d, fmt)	converts a date value d into a string with the specified format fmt	to_char(current_date, 'MON DD, YYYY')
to_char(n, fmt)	converts a number n into a string with the specified	to_char(-10000, 'L99G999D99MI') is

	format fmt	10,000.00-
to_date(s, fmt)	converts a string s into a number with the specified format fmt	to_date('Feb 14, 2002', 'MON DD, YYYY')

Tables 4.10 and 4.11 list the some frequently used number and date formats.

Table 4.10 Number Formats

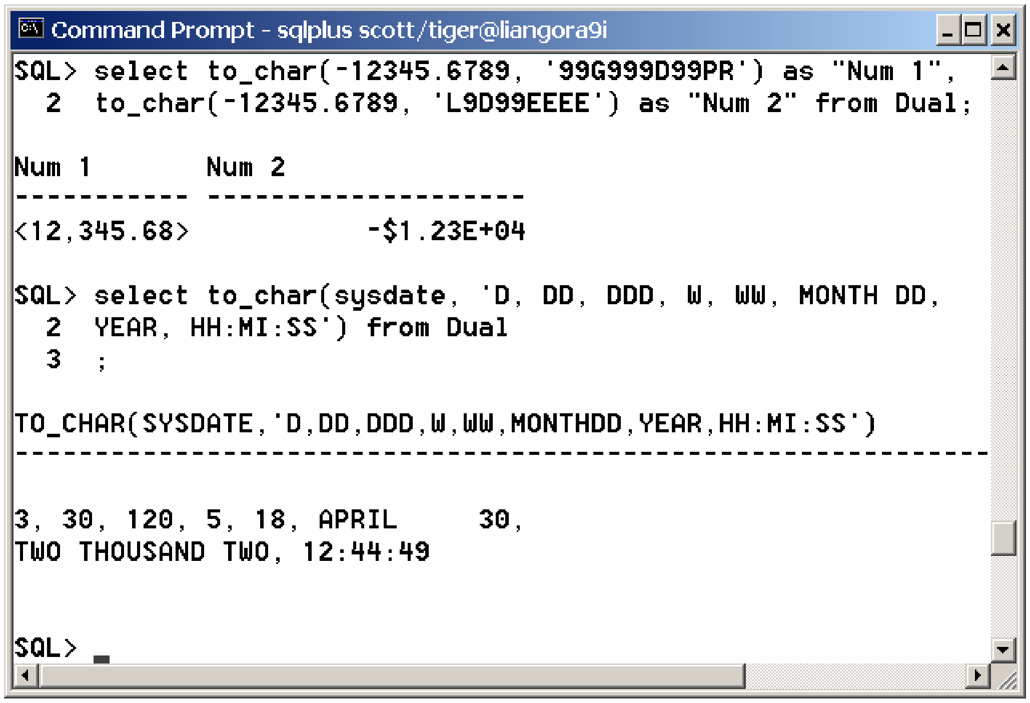
Format	Description	Example
9	a number	9999
0	a leading zero	0009999
EEEE	scientific notation	999EEEE
D	decimal point (.)	9D99
G	group separator (,)	9G99
L	currency symbol	L999
MI	minus sign	999MI
PR	puts negative number in	999PR

Table 4.11 Date Formats

Format	Description
MM	two-digit month
MON	first three letters of the month
MONTH	month name using 9 characters padded with blanks on the left
RM	month in Roman numerals
Q	quarter of the year (1, 2, 3, 4)
YEAR	year spelled out completely
Y	one-digit year
YY	two-digit year
YYY	three-digit year
YYYY	four-digit year
W	week number of the month
WW	week number of the year
D	day of week
DD	day of month
DDD	day of year
DAY	name of day using nine characters padded with blanks on the left
HH or HH12	hour of day (0-12)
HH24	hour of day (0-23)
MI	minute (0-59)
SS	second (0-59)

Here are two examples of using the number and date formats. Their output is shown in Figure 4.14.

```
select to_char(-12345.6789, '99G999D99PR') as "Num 1",
  to_char(-12345.6789, 'L9D99EEEE') as "Num 2" from Dual;

select to_char(sysdate, 'D, DD, DDD, W, WW, MONTH DD, YEAR,
  HH:MI:SS')
  from Dual;
```



Command Prompt - sqlplus scott/tiger@liangora9i

```
SQL> select to_char(-12345.6789, '99G999D99PR') as "Num 1",
  2  to_char(-12345.6789, 'L9D99EEEE') as "Num 2" from Dual;

Num 1       Num 2
----------- --------------------
<12,345.68>            -$1.23E+04

SQL> select to_char(sysdate, 'D, DD, DDD, W, WW, MONTH DD,
  2  YEAR, HH:MI:SS') from Dual
  3  ;

TO_CHAR(SYSDATE,'D,DD,DDD,W,WW,MONTHDD,YEAR,HH:MI:SS')
-------------------------------------------------------------

3, 30, 120, 5, 18, APRIL     30,
TWO THOUSAND TWO, 12:44:49


SQL> _
```

Figure 4.14

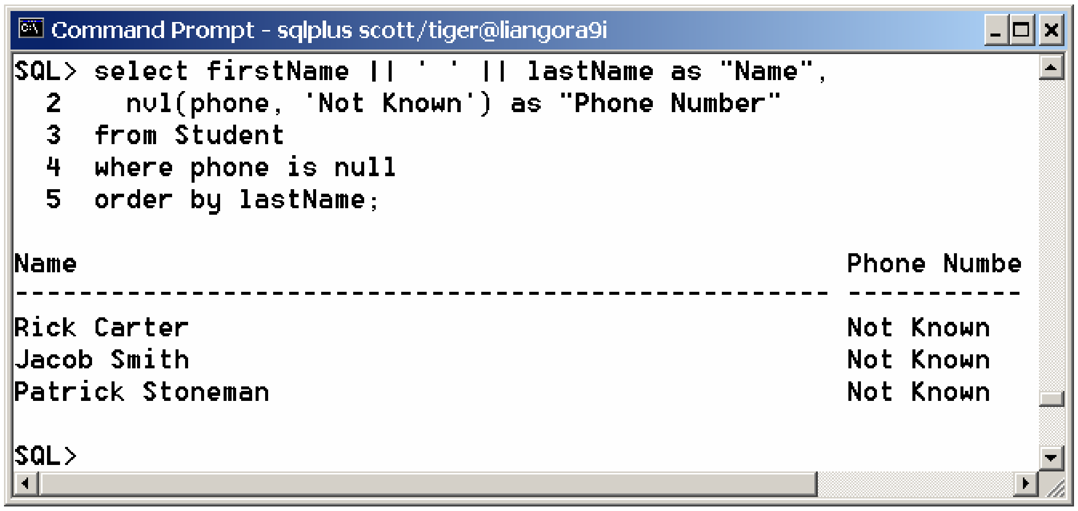
Numbers and dates are displayed in the specified formats.

4.7.2.5 Miscellaneous Functions

Oracle has several other functions that do not fall into the category in the preceding sections. They are called *miscellaneous functions*. Two useful miscellaneous functions are nvl and decode. nvl(exp1, exp2) returns exp2 if exp1's value is null, otherwise exp1's value is returned. This function is useful to give descriptive names for null values.

Example 4.9: Display the students whose phone number is null in increasing order of their last name. Display "Not Known" for null values. The query result is shown in Figure 4.15.

```
select firstName || ' ' || lastName as "Name",
  nvl(phone, 'Not Known') as "Phone Number"
from Student
where phone is null
order by lastName;
```



```
SQL> select firstName || ' ' || lastName as "Name",
  2    nvl(phone, 'Not Known') as "Phone Number"
  3  from Student
  4  where phone is null
  5  order by lastName;

Name                                                 Phone Numbe
---------------------------------------------------- -----------
Rick Carter                                          Not Known
Jacob Smith                                          Not Known
Patrick Stoneman                                     Not Known

SQL>
```

Figure 4.15

Example 4.9 demonstrates the nvl function.

NOTE: By default, null value is displayed as an empty string in SQL*Plus and iSQL*Plus.

The decode function is like a Java switch statement. decode(exp, value1, result1, [value2, result2, ...], [, default]) compares exp to each value value1, value2, ... If exp is equal to a value, Oracle returns the corresponding result. If no match is found, Oracle returns default, or, if default is omitted, returns null.

Example 4.10: Increase the salary for MATH faculty by 10%, for CS faculty by 20%, and for others by 15%.

```
update Faculty
set salary =
  decode(deptCode, 'MATH', salary * 1.1,
     'CS',   salary * 1.2,  salary * 1.15);
```

4.8 The group by clause

Rows can be divided into groups. You can apply aggregate functions in each group. For example, to find the total number of faculty in each department, you need to group faculty into departments and count faculty in each department. SQL provides the group by clause to group rows using the following syntax:

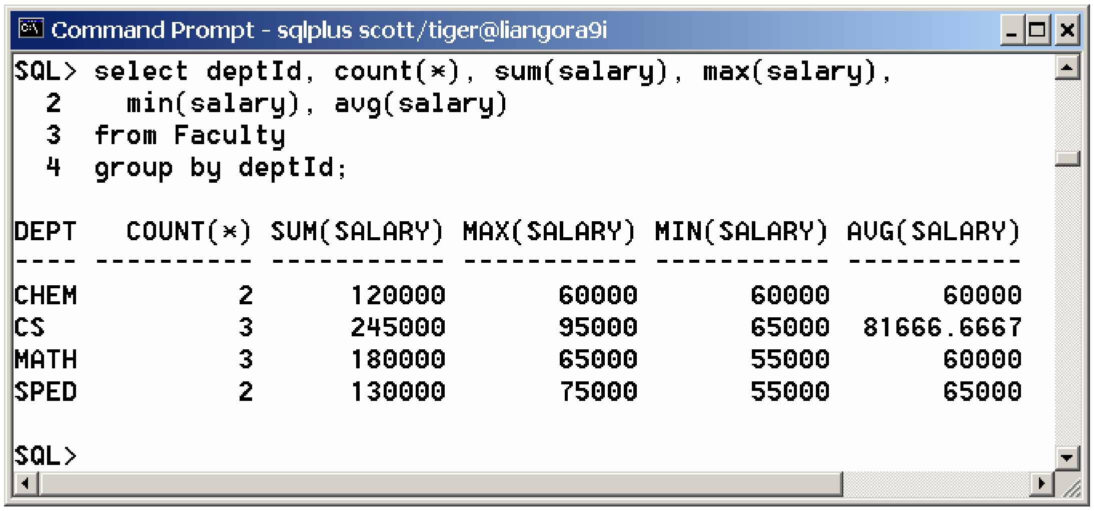
```
select column-list
from table-list
[where condition]
[group by columns-to-be-grouped]
```

```
[order by columns-to-be-sorted];
```

The group by clause specifies how the rows are grouped. The group by clause is often used with the aggregate functions.

Example 4.11: Get the number of faculty, total salary, maximum salary, minimum salary, and average salary in each department. The query result is shown in Figure 4.16.

```
select deptId, count(*), sum(salary), max(salary),
  min(salary), avg(salary)
from Faculty
group by deptId;
```


Command Prompt - sqlplus scott/tiger@liangora9i

```
SQL> select deptId, count(*), sum(salary), max(salary),
  2    min(salary), avg(salary)
  3  from Faculty
  4  group by deptId;

DEPT   COUNT(*) SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)
---- ---------- ----------- ----------- ----------- -----------
CHEM          2      120000       60000       60000       60000
CS            3      245000       95000       65000  81666.6667
MATH          3      180000       65000       55000       60000
SPED          2      130000       75000       55000       65000

SQL>
```

Figure 4.16

Example 4.11 demonstrates the group by clause.

> NOTE: The columns to be grouped don't have to appear in the select clause. However, if you use an aggregate function in the select clause, all the individual columns in the select clause must be in the group by clause.

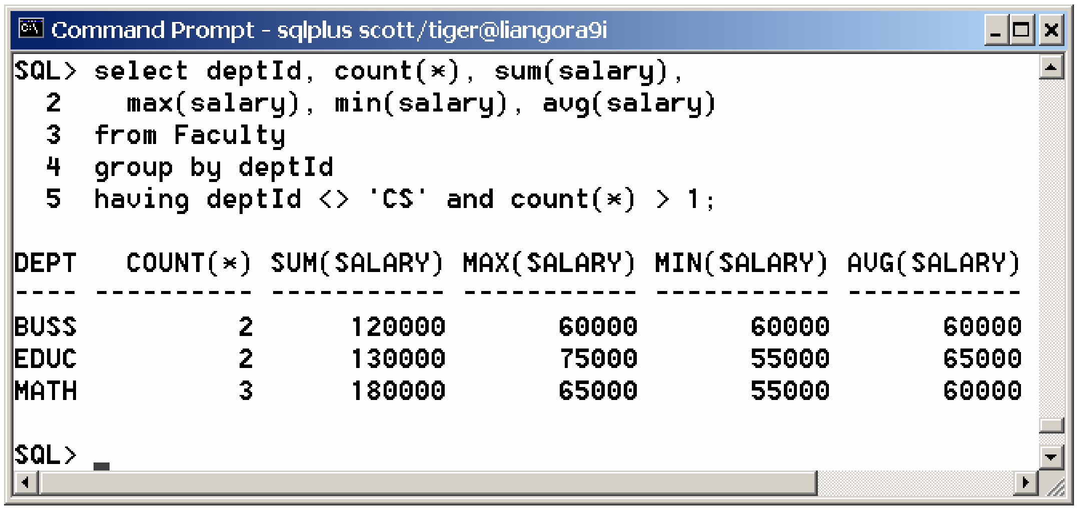
4.9 The having Clause

Sometimes, you don't need all the groups. You can use the having clause to restrict groups. The syntax is as follows:

```
select column-list
from table-list
[where condition]
[group by columns-to-be-grouped]
[having condition]
[order by columns-to-be-sorted];
```

Example 4.12: Get the number of faculty, total salary, maximum salary, minimum salary, and average salary in each department except the CS department and the department with only one faculty. The query result is shown in Figure 4.17.

```
select deptId, count(*), sum(salary),
  max(salary), min(salary), avg(salary)
from Faculty
group by deptId
having deptId <> 'CS' and count(*) > 1;
```



```
SQL> select deptId, count(*), sum(salary),
  2    max(salary), min(salary), avg(salary)
  3  from Faculty
  4  group by deptId
  5  having deptId <> 'CS' and count(*) > 1;

DEPT   COUNT(*) SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)
---- ---------- ----------- ----------- ----------- -----------
BUSS          2      120000       60000       60000       60000
EDUC          2      130000       75000       55000       65000
MATH          3      180000       65000       55000       60000

SQL> _
```

Figure 4.17

Example 4.12 demonstrates the having clause.

> NOTE: The having clause must always follow the group by clause and the conditions are related to the groups not individual rows.

4.10 Formatting Query Results

Standard SQL does not allow you to format query results except using the alias. Oracle SQL*Plus allows you to change column headings, specify column width, and format column data.

4.10.1 Changing Column Headings

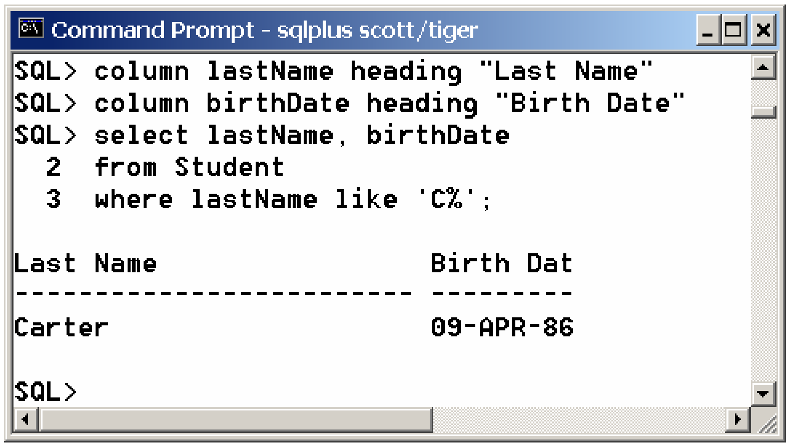
SQL*Plus uses column or expression names as default column headings when displaying query results. You may use the alias in the select clause to change the column heading, or use the following command:

```
column column_name heading column_heading
```

For example, to produce a report with new headings specified for lastName and birthDate for students whose last name begins with letter C, enter the following command:

```
column lastName heading "Last Name"
column birthDate heading "Birth Date"
select lastName, birthDate
from Student
where lastName like 'C%';
```

The output is shown in Figure 4.18


Command Prompt - sqlplus scott/tiger

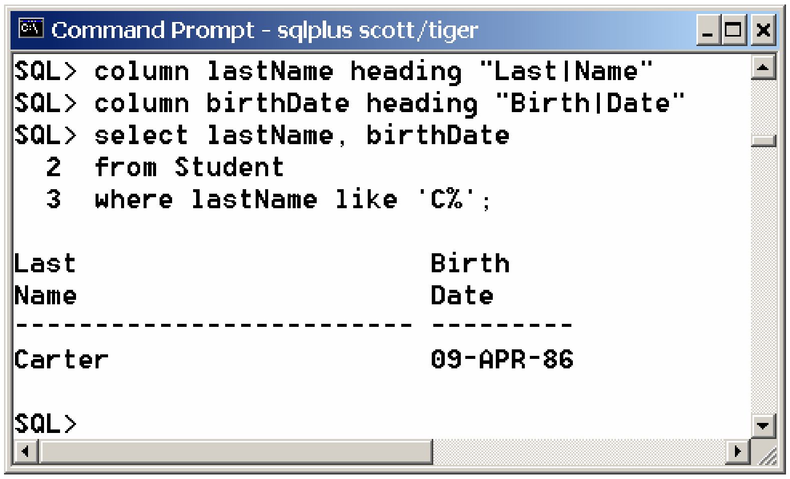
```
SQL> column lastName heading "Last Name"
SQL> column birthDate heading "Birth Date"
SQL> select lastName, birthDate
  2  from Student
  3  where lastName like 'C%';

Last Name                 Birth Dat
------------------------- ---------
Carter                    09-APR-86

SQL>
```

Figure 4.18

You can use the column command to specify a new heading.

You can display column headings in multiple lines by using the vertical bar symbol (|) to separate headings. For example, to display heading "Last Name" and "Birth date" in two separate lines, use the following command (See Figure 4.19):

```
column lastName heading "Last|Name"
column birthDate heading "Birth|Date"
```


Command Prompt - sqlplus scott/tiger

```
SQL> column lastName heading "Last|Name"
SQL> column birthDate heading "Birth|Date"
SQL> select lastName, birthDate
  2  from Student
  3  where lastName like 'C%';

Last                      Birth
Name                      Date
------------------------- ---------
Carter                    09-APR-86

SQL>
```

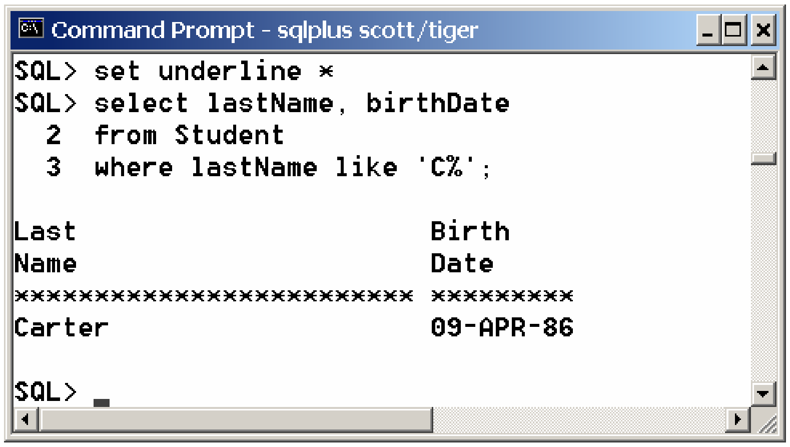
Figure 4.19

You can use the vertical bar to separate column headings.

To change the character used to underline each column heading, set the UNDERLINE variable of the set command to the desired character. For example, the following command changes the underline character to the equal sign (=).

```
set underline =
```

Figure 4.20 shows an output using with a new underline character *.



```
SQL> set underline *
SQL> select lastName, birthDate
  2  from Student
  3  where lastName like 'C%';

Last                      Birth
Name                      Date
************************* *********
Carter                    09-APR-86

SQL> _
```

Figure 4.20

You can use a different underline character.

4.10.2 Formatting Number Columns

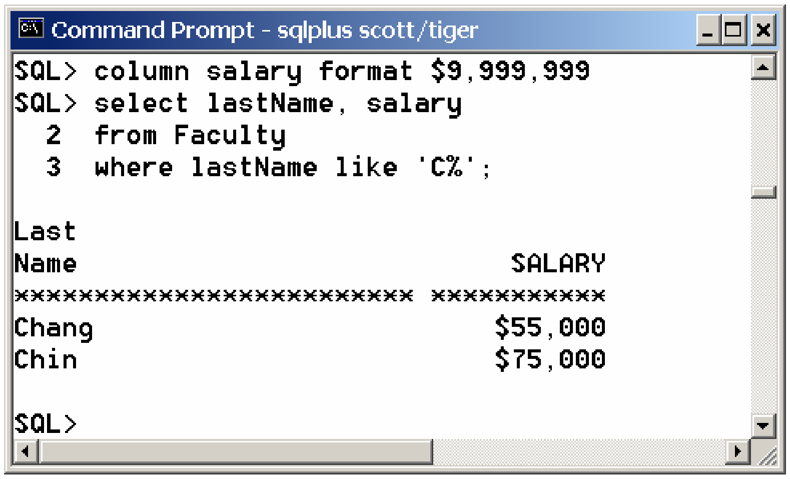
You can display numbers in the specified format using the following command:

```
column columnName format formatStyle
```

The formatStyle can contain currency symbols, 9s and commas. For example, the following command displays salary in US currency.

```
column salary format $9,999,999
```

The output is shown in Figure 4.21.



```
SQL> column salary format $9,999,999
SQL> select lastName, salary
  2  from Faculty
  3  where lastName like 'C%';

Last
Name                              SALARY
************************* ***********
Chang                            $55,000
Chin                             $75,000

SQL>
```

Figure 4.21

You can format numbers.

4.10.3 Formatting Character Types

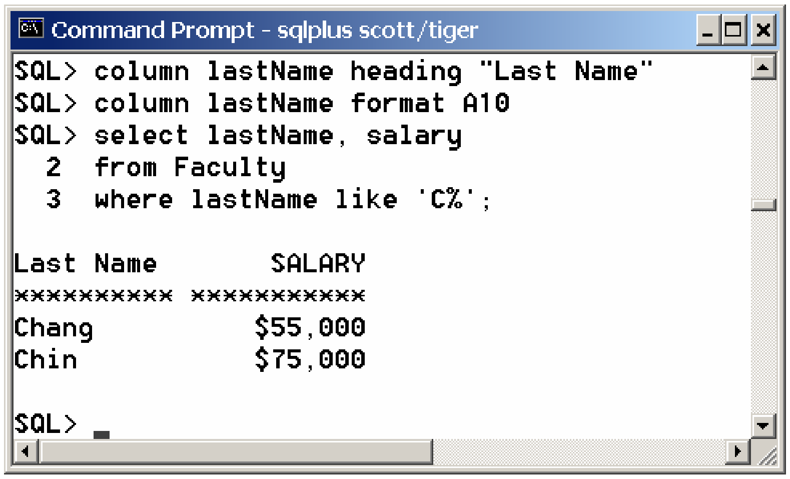
The default width of a column is the width of the column in the database. To specify a width, use the following command:

<u>column *columnName* format *width*</u>

The <u>width</u> parameter specifies a number for the width of the column displayed. For example, the following command displays last name in 10 characters width.

<u>column lastName format A10</u>

The output is shown in Figure 4.22.



```
SQL> column lastName heading "Last Name"
SQL> column lastName format A10
SQL> select lastName, salary
  2  from Faculty
  3  where lastName like 'C%';

Last Name       SALARY
********** ***********
Chang          $55,000
Chin           $75,000

SQL> _
```

Figure 4.22

You can specify the width for character data.

> NOTE: The specified format will stay in effect until you enter a new one; reset the column's format with
>
> column lastName clear
>
> or exit from SQL*Plus.

******End of NOTE***

Chapter Summary

This chapter introduced SQL basics. You learned how to create, modify, and drop tables. You learned how to use SQL data types, how to define constraints tables, and how to use simple statements for queries and update operations. You learned how to use SQL aggregate functions (count, min, max, avg, and sum), and Oracle number, character functions, date, and conversion functions. You learned how to use the order by clause to sort output, the group by clause to group rows, and the having clause to restrict rows. You also learned to format display using the column command in Oracle SQL*Plus.

Review Questions

4.1 Describe SQL1, SQL2, and SQL3.

4.2 What is the maximum size for the varchar2 type? What is the maximum size for the char type?

4.3 After a table is created, can you add a new column or drop an existing column?

4.4 What types of constraints can you define? Can you define the constraints using the create table statement? Can you add a constraint or drop a constraint after a table is created?

4.5 What are column-level constraints and table-level constraints?

4.6 What are the differences between a named constraint and an unnamed constraint?

4.7 Can you enable or disable a constraint?

4.8 How do you specify default values on attributes?

4.9 How do you display all user tables in Oracle?

4.10 How do you display table definition for a given table?

4.11 How do you display constraints for a given table?

4.12 What is the difference between the drop table command and the truncate table command?

4.13 How do you rename a table? Does renaming a table affect all the dependents of the table?

4.14 If two tables T1 and T2 reference each other in the foreign keys, how do you create these two tables?

4.15 How do you commit a DML statement? Is a DDL statement automatically committed?

4.16 What are the differences between aggregate functions and Oracle number functions?

4.17 How do you display distinct rows in the output?

4.18 How do you sort the output?

4.19 What is the group by clause for? What is the having clause for?

4.20 How do you format numbers and strings in the query output?

Exercises

4.1 Create a table named Card with the following columns:

cardNo: integer value, primary key
creditLimit: floating-point value with two decimal points, default to 10000
currentBalance: floating-point value with two decimal points, default to 0
since: date indicating when the credit was started

Create a table named Customer with the following columns:

ssn: fixed-length nine characters
cardNo: integer value, foreign key in the Card table
lastName: variant-length 25 characters maximum, not null
mi: single character
firstName: variant-length 25 characters maximum

(ssn and cardNO together form the primary key)

4.2 Alter the Card table as follows:

a. Add a new column named overDue whose type is floating-point value with two decimal digits.
b. Add a new constraint to specify that creditLimit is no more than 10000.

4.3 Insert the following records into the Card and Customer tables.

PD: Create two tables Card and Customer

Card Table

CARDNO	CREDITLIMIT	CURRENTBALANCE	SINCE	OVERDUE
1344	9000	2500	01-AUG-1990	300.5
2344	5000	1500	01-SEP-1999	400.56
3344	9000	7500	01-OCT-1979	500.47
4344	3000	3000	01-SEP-1999	600.59
5344	3000	3000	01-JAN-1990	700.59
6344	3000	3000		800.59
7344	9000	9000	01-JAN-1990	800.59
7345	9000	9000	01-JAN-1990	1800.59

Customer Table

SSN	Last Name	M	FIRSTNAME	CARDNO
444111111	Smith	H	John	1344
555111111	Jones	J	R	2344
666111111	Ostrow	R	John	3344
777111111	Jones	J	Beth	5344
888111111	Jones	J	Kevin	4344
888111111	Jones	J	Kevin	6344

4.4 Get the cardNo for all cards created since Jan 1, 1990 with current balance more than 5000.

4.5 Get the cardNo for all cards created between Jan 1, 1990 and March 1, 1990 and the current balance is not null.

4.6 Get the ssn for all customers whose last name begins with the letter S.

4.7 Assume the interest rate is 15% on over due. Display the interest charge on overdue amount for all credit cards.

4.8 List the full name of the customer who has at least one credit card.

4.9 List the full name of the customer, ordered primarily on the last name in descending order and secondarily on the first name in ascending order. (Hint: Use the order by clause.)

4.10 List the minimum current balance, maximum current balance, average current balance, and total current balance in the Card table.

4.11 Display the customers whose last name has a string 'sm' in lowercase or uppercase. (Hint: Use the lower or upper function.)

4.12 Display the cardNo and the date when the card was created. Display "1-JAN-1990" for null date values. (Hint: Use the nvl function.)

4.13 List the card number, creditLimit, and whether is over the limit. If the overdue is greater than the limit, display "over the limit," otherwise, display "under the limit." (Hint: Use the decode function.)

4.14 List customer name and the number of cards that belong to the same customer. (Hint: Use the group by clause.)

4.15 List customer with at least two cards. (Hint: Use the group by and having clauses.)

4.16 Format the overdue column to display in US currency. Format the lastName column to display only 10 characters.