4

Structured Query Language

Learning Objectives. This chapter focuses on how to access the data within a DBMS. An introduction to SQL, an international standard language for manipulating relational database is given in this chapter. After completing this chapter the reader should be familiar with the following concepts in SQL.

- Evolution and benefits of SQL
- Datatypes in SQL
- SQL commands to create a table, inserting records into the table, and extracting information from the table
- Aggregate functions, GROUP BY clause
- Implementation of constraints in SQL using CHECK, PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE commands
- Concepts of sub query, view, and trigger

4.1 Introduction

SQL stands for "Structured Query Language." The Structured Query Language is a relational database language. By itself, SQL does not make a DBMS. SQL is a medium which is used to communicate to the DBMS. SQL commands consist of English-like statements which are used to query, insert, update, and delete data. English-like statements mean that SQL commands resemble English language sentences in their construction and use and therefore are easy to learn and understand.

SQL is referred to as nonprocedural database language. Here nonprocedural means that, when we want to retrieve data from the database it is enough to tell SQL what data to be retrieved, rather than how to retrieve it. The DBMS will take care of locating the information in the database.

Commercial database management systems allow SQL to be used in two distinct ways. First, SQL commands can be typed at the command line directly. The DBMS interprets and processes the SQL commands immediately, and the results are displayed. This method of SQL processing is called interactive SQL. The second method is called programmatic SQL. Here, SQL

S. Sumathi: *Structured Query Language*, Studies in Computational Intelligence (SCI) **47**, 111–212 (2007)
www.springerlink.com

statements are embedded in a host language such as COBOL, FORTRAN, C, etc. SQL needs a host language because SQL is not a really complete computer programming language as such because it has no statements or constructs that allow branch or loop. The host language provides the necessary looping and branching structures and the interface with the user, while SQL provides the statements to communicate with the DBMS.

Some of the features of SQL are:

- SQL is a language used to interact with the database.
- SQL is a data access language.
- SQL is based on relational tuple calculus.
- SQL is a standard relational database management language.
- The first commercial DBMS that supported SQL was Oracle in 1979.
- SQL is a "nonprocedural" or "declarative" language.

4.2 History of SQL Standard

The origin of the SQL language date back to a research project conducted by IBM at their research laboratories in San Jose, California in the early 1970s. The aim of the project was to develop an experimental RDBMS which would eventually lead to a marketable product. At that time, there was a lot of interest in the relational model for databases at the academic level, in conferences and seminars. IBM, which already had a large share of the commercial database market with hierarchical and network model DBMSs, realized that the relational model would dominate the future database products. The project at IBM's San Jose labs was started in 1974 and was named System R. A language called SEQUEL (Structured English QUEry Language) was chosen as the relational database language for System R. A version of SEQUEL was developed at the IBM San Jose research facilities and tested with college students.

In November 1976, specifications for SEQUEL2 were published. In 1980 minor revisions were made to SEQUEL, and it was renamed "SQL." SEQUEL was renamed to SQL because the name SEQUEL had already been used for hardware product. In order to avoid confusion and legal problems SEQUEL was renamed to SQL. In the first phase of the System R project, researchers concentrated on developing a basic version of the RDBMS. The main aim at this stage was to verify that the theories of the relational model could be translated into a working, commercially viable product. This first phase was successfully completed by the end of 1975, and resulted in a single-user DBMS based on the relational model. The System R project was completed in 1979. The theoretical work of the System R project resulted in the development and release of IBM's first commercial relational database management system in 1981. The product was called SQL/DS (Structured Query Language/Data Store) and ran under the DOS/VSE operating system environment. Two years later, IBM announced a version of SQL/DS for VM/CMS operating system.

In 1983, IBM released a second SQL-based RDBMS called DB2, which ran under the MVS operating system. DB2 quickly gained widespread popularity and even today, versions of DB2 form the basis of many database systems found in large corporate data-centers. During the development of System R and SQL/DS, other companies were also at work creating their own relational database management systems. Some of them, Oracle being an example, even implemented SQL as the relational database language for their DBMSs concurrently with IBM. Later on, SQL language was standardized by ANSI and ISO. The ANSI SQL standards were first published in 1986 and updated in 1989, 1992, and 1999.

4.2.1 Benefits of Standardized Relational Language

The main advantages of standardized language are given below.

1. Reduced training cost
2. Enhanced productivity
3. Application portability
   Application portability means applications can be moved from machine to machine when each machine uses SQL.
4. Application longevity
   A standard language tends to remain so for a long time, hence there will be little pressure to rewrite old applications.
5. Reduced dependence on a single vendor

SQL language development is given in a nutshell below:

1. In 1970 E.F. Codd of IBM released a paper "A relational model of data for large shared data banks." IBM started the project System R to demonstrate the feasibility of implementing the relational model in a database management system. The language used in system R project was SEQUEL. SEQUEL was renamed SQL during the project, which took place from 1974 to 1979.
2. The first commercial RDBMS from IBM was SQL/DS. It was available in 1981.
3. Oracle from relational software (now Oracle corporation) was on the market before SQL/DS, i.e., 1979.
4. Other products included INGRES from relational Technology Sybase from Sybase, Inc. (1986), DG/SQL from Data General Corporation (1984).

4.3 Commands in SQL

SQL commands can be classified in to three types:

1. Data Definition Language commands (DDL)
2. Data Manipulation Language commands (DML)
3. Data Control Language commands (DCL)

DDL

DDL commands are used to define a database, including creating, altering, and dropping tables and establishing constraints.

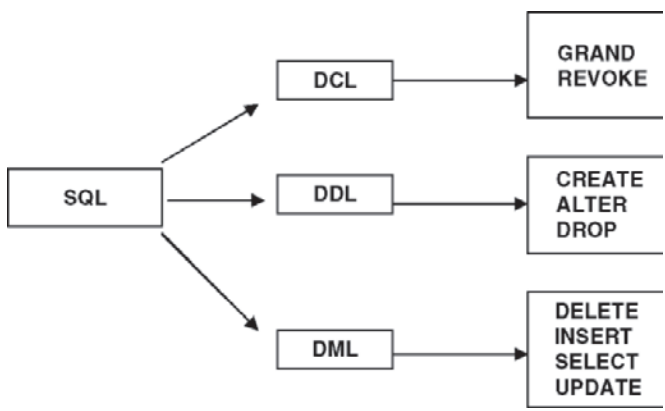
DML

DML commands are used to maintain and query a database, including updating, inserting, modifying, and querying data.

DCL

DCL commands are used to control a database including administering privileges and saving of data. DCL commands are used to determine whether a user is allowed to carry out a particular operation or not. The ANSI standard groups these commands as being part of the DDL.

The classification of commands in SQL is shown below.



4.4 Datatypes in SQL

In relational model the data are stored in the form of tables. A table is composed of rows and columns. When we create a table we must specify a datatype for each of its columns. These datatypes define the domain of values that each column can take. Oracle provides a number of built-in datatypes as well as several categories for user-defined types that can be used as datatypes. Some of the built-in datatypes are string datatype to store characters, number datatype to store numerical value, and date and time datatype to store when the event happened (history, date of birth, etc.).

STRING

In string we have CHAR and VARCHAR datatypes. Character datatype store data which are words and free-form text, in the database character set.

CHAR Datatype

The CHAR datatype specifies a fixed-length character string. The syntax of CHAR datatype declaration is:

CHAR (n) – Fixed length character data, "n" characters long.

Here "n" specifies the character length. If we insert a value that is shorter than the column length, then Oracle blank-pads the value to column length. If we try to insert a value that is too long for the column then Oracle returns error message.

VARCHAR2 Datatype

The VARCHAR2 datatype specifies a variable-length character string. The syntax of VARCHAR2 datatype declaration is:

VARCHAR2 (n) – Variable length character of "n" length.

Here "n" specifies the character length.

VARCHAR vs. VARCHAR2

The VARCHAR datatype behaves like VARCHAR2 datatype in the current version of Oracle.

In order to justify the above statement, let us create a table CHAMPION, which refers to Wimbledon Champions. The attributes of the table CHAMPION are *Name*, *Nation*, *Year* (the year in which the sportsman has won the title). For our example, let us use the datatype VARCHAR for the attribute Name and VARCHAR2 for the datatype Nation. The SQL command to create CHAMPION is shown in Fig. 4.1.

Now let us try to see the description of the table. The description of the table is shown in Fig. 4.2.

From Fig. 4.2, it is clear that both name and nation are stored as VARCHAR2(12). This means that VARCHAR datatype in the Oracle 8i version behaves the same as VARCHAR2.

NUMBER Datatype

The NUMBER datatype stores zero, positive, and negative fixed and floating point numbers.

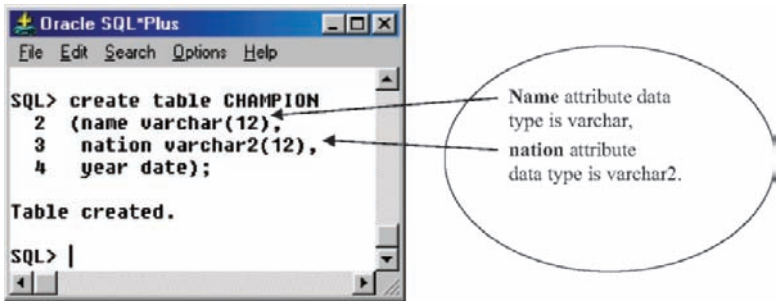


Fig. 4.1. CHAR and VARCHAR2 datatype

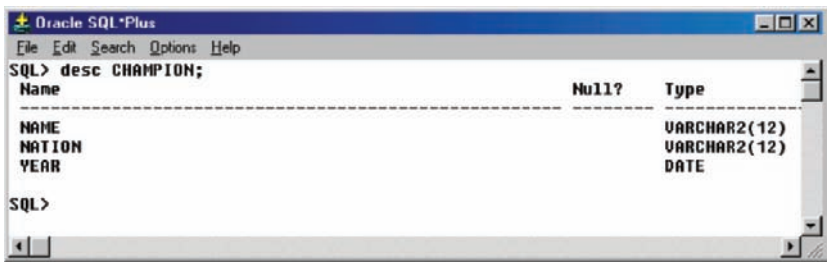


Fig. 4.2. Table description

The syntax to store fixed-point number is NUMBER (p, q) where "p" is the total number of digits and "q" is the number of digits to the right of decimal point.

The syntax to specify an integer is NUMBER (p).

DATE Datatype

The DATE datatype is used to store the date and time information. For each DATE value, Oracle stores the century, year, month, date, hour, minute, and second information. The ANSI date literal contains no time portion, and must be specified in YYYY-MM-DD format where Y stands for Year, M for month, and D for date.

TIME STAMP Datatype

The TIME STAMP datatype is used to store both date and time. It stores the year, month, and day of the DATE datatype, and also hour, minute, and second values.

LOB Datatype

Multimedia data like sound, picture, and video need more storage space. The LOB datatypes such as BLOB, CLOB, and BFILE allows us to store large block of data.

BLOB Datatype

The BLOB datatype stores unstructured binary data in the database. BLOBs can store up to 4 GB of binary data.

CLOB Datatype

The CLOB datatype can store up to 4 GB of character data in the database.

BFILE Datatype

The BFILE datatype stores unstructured binary data in operating system files outside the database. A BFILE can store up to 4 GB of data.

4.5 Data Definition Language (DDL)

The Data Definition Language is

- Used to define schemas, relations, and other database structures
- Also used to update these structures as the database evolves

Examples of Structure Created by DDL

The different structures that are created by DDL are Tables, Views, Sequences, Triggers, Indexes, etc.

1. Tables
   The main features of table are:
   - It is a relation that is used to store records of related data. It is a logical structure maintained by the database manager.
   - It is made up of columns and rows.
   - At the intersection of every column and row there is a specific data item called a value.
   - A base table is created with the CREATE TABLE statement and is used to hold persistent user data.
2. Views
   The basic concepts of VIEW are:
   - It is a stored SQL query used as a "Virtual table."
   - It provides an alternative way of looking at the data in one or more tables.
   - It is a named specification of a result table. The specification is a SELECT statement that is executed whenever the view is referenced in an SQL statement. Consider a view to have columns and rows just like a base table. For retrieval, all views can be used just like base tables.

   - When the column of a view is directly derived from the column of a base table, that column inherits any constraints that apply to the column of the base table. For example, if a view includes a foreign key of its base table, INSERT and UPDATE operations using that view are subject to the same referential constraints as the base table. Also, if the base table of a view is a parent table, DELETE and UPDATE operations using that view are subject to the same rule as DELETE and UPDATE operations on the base table.

3. Sequences
   - A sequence is an integer that varies by a given constant value. Typically used for unique ID assignment
4. Triggers
   - Trigger automatically executes certain commands when given conditions are met.
5. Indexes
   - Indexes are basically used for performance tuning. Indexes play a crucial role in fast data retrieval.

Create Table Command

- The CREATE TABLE command is used to implement the schemas of individual relations.

Steps in Table Creation

1. Identify datatypes for attributes
2. Identify columns that can and cannot be null
3. Identify columns that must be unique
4. Identify primary key–foreign key mates
5. Determine default values
6. Identify constraints on columns (domain specifications)
7. Create the table

Syntax

CREATE TABLE *table name*
(column-name1 data-type-1 [constraint],
column-name2 data-type-2 [constraint],
column-nameN data-type-N [constraint]
);

Example Table

See Table 4.1.

Table 4.1. Peaks of the world

Serial number	Peak	Mountain range	Place	Height
1	Everest	Himalayas	Nepal	8,848
2	Godwin Austin	Karakoram	India	8,611
3	Kanchenjunga	Himalayas	Nepal	8,579

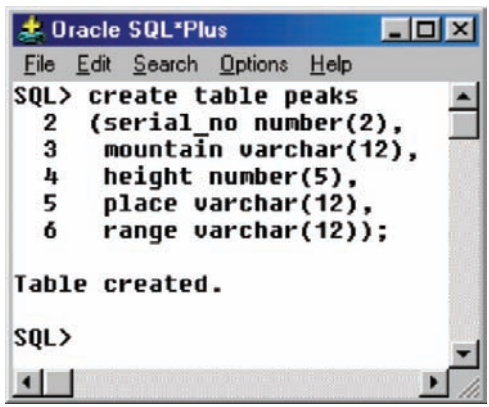


Fig. 4.3. Table creation example

Syntax to Create the Table

The general syntax to create the table is given below. Here the key words are shown in bold and capital letters.

CREATE TABLE table name
(column name1 data type (size),
column name2 data type (size),
column name N data type (size));

Example

The SQL command to define Table 4.1 is shown in Fig. 4.3. In this example the name of the table is *peaks*. The table has five columns which are serial number, name of the mountain (peak), height, place where the mountain is situated, range of the mountain.

To see the description of the table

To see the description of the table we have created we have the command **DESC**. Here **DESC** stands for description of the table. The syntax of **DESC** command is:

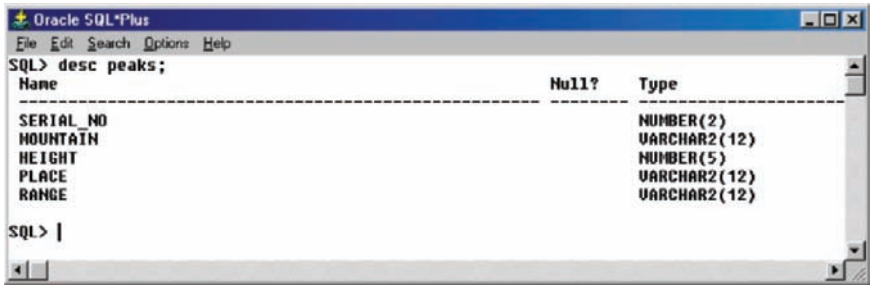


Fig. 4.4. Table description

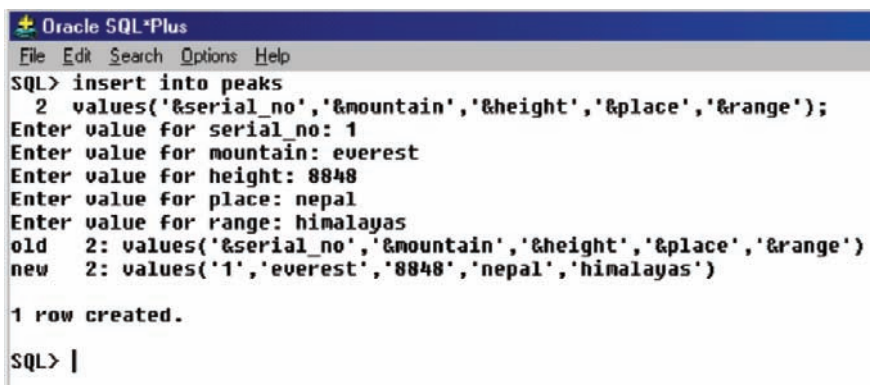


Fig. 4.5. Inserting values into the table

Syntax: **DESC** table name;

The DESC command returns the attributes (columns) of the table, the datatype associated with the column, and also any constraint (if any) imposed on the column. Figure 4.4 shows the description of the table PEAKS.

To insert values into the table

Syntax: **Insert into <tablename> values ('&columnname1', '&columnname2', &col3,. . .);**

(e.g.) The SQL syntax and the corresponding output are shown in Fig. 4.5. Now to insert the next set of values, use the slash as shown in Fig. 4.6.

To view the entire table

The SQL syntax to see all the columns of the table is:

SELECT * FROM table name;

Here the asterisk symbol indicates the selection of all the columns of the table.

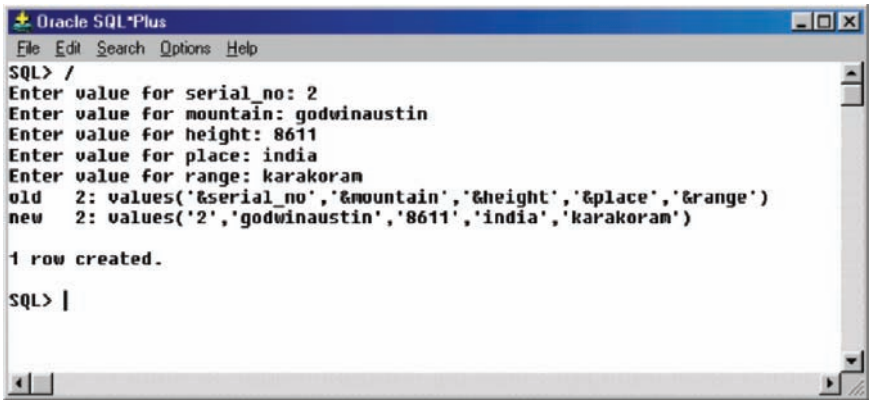


Fig. 4.6. Inserting successive values into the table

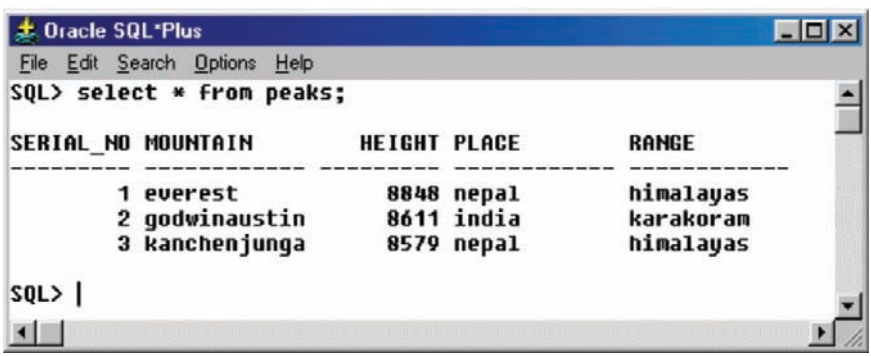


Fig. 4.7. SELECTION of all columns of the table

Example

The SQL command to see all the columns of the table PEAKS and the corresponding output are shown in Fig. 4.7.

SQL> select * from peaks;

4.6 Selection Operation

Selection operation can be considered as row wise filtering. We can select specific row(s) using condition.

Syntax of SELECTION Operation

SELECT * FROM table name
WHERE condition;

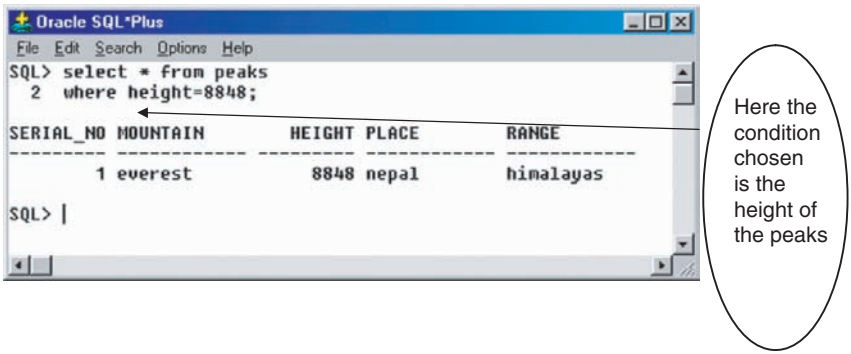


Fig. 4.8. SELECTION operation

Example of SELECTION operation

In the example Table 4.1, there are three rows. Let us filter two rows so that only one row will appear in the result. Here the condition used to filter the rows is the "height" of the PEAKS. The SQL command to implement SELECTION operation and the corresponding output are shown in Fig. 4.8.

From Fig. 4.8 it is clear that even though there are three rows in the Table 4.1, it is reduced to one using the condition the height of the peaks. This operation which filters the rows of the relation is called SELECTION.

4.7 Projection Operation

The projection operation performs column wise filtering. Specific columns are selected in projection operation.

Syntax of PROJECTION Operation

SELECT column name1, column name2, Column name N **FROM** table name;

If all the columns of the table are selected, then it cannot be considered as PROJECTION.

The SQL command to perform PROJECTION operation on the relation PEAKS and the corresponding results are shown in Fig. 4.9.

From Fig. 4.9, it is clear that only three columns are selected in the result, even though there are five columns in the Table 4.1.

SELECTION and PROJECTION Operation

We can perform both selection and projection operation in a relation. If we combine selection and projection operation means naturally we are restricting the number of rows and the columns of the relation.

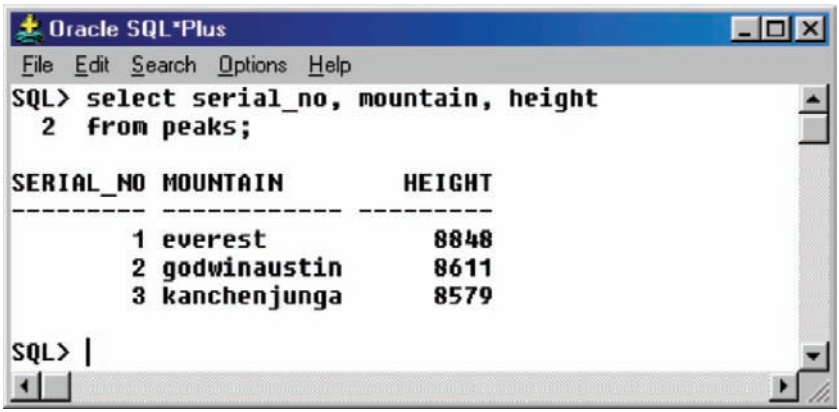


Fig. 4.9. PROJECTION operation

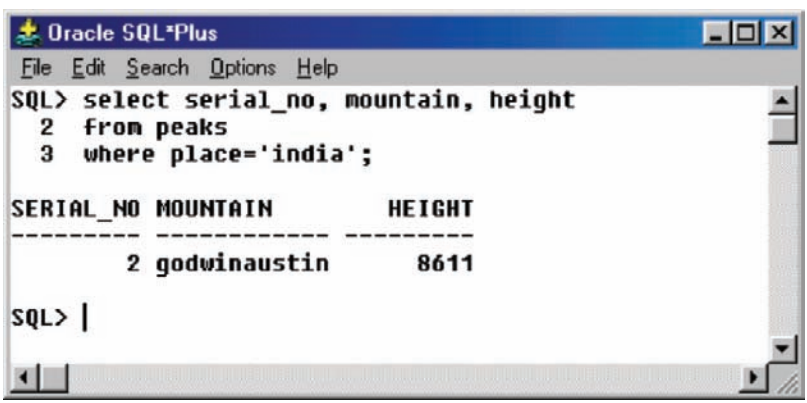


Fig. 4.10. SELECTION and PROJECTION operation

Syntax for Selection and Projection

SELECT column name1, column name 2. . . . . column name N
FROM table name
WHERE condition;

The selection and projection operation applied to the peaks relation is shown in Fig. 4.10.

From Fig. 4.10, we can observe that the selection operation is based on the "place" of the peaks. As a result only one row is obtained as the result. Because of projection operation only three columns are obtained in the result as shown in Fig. 4.10.

4.8 Aggregate Functions

SQL provides seven built-in functions to facilitate query processing. The seven built-in functions are COUNT, MAX, MIN, SUM, AVG, STDDEV, and VARIANCE. The uses of the built-in functions are shown in Table 4.2.

4.8.1 COUNT Function

The built-in function returns the number of rows of the table. There are variations of COUNT function. First let us consider COUNT (*) function. In order to understand the COUNT (*) function consider the relation PERSON_SKILL as shown in Table 4.3, the relation PERSON has only two columns, name of the person and skills associated with the person. It is to be noted that some persons may have more than one skill and some persons may not have any skills.

From Table 4.3, we can observe that the table PERSON_SKILL has six rows and two columns and the person Ashok has more than one skill and Sam has no skill hence a NULL is inserted against Sam.

(A) COUNT (*) Function

The syntax of Count (*) function is:

SELECT COUNT (*)
FROM table name;

Table 4.2. Built-in functions

Serial number	Built-in function	Use
1	COUNT	to count the number of rows of the relation
2	MAX	to find the maximum value of the attribute (column)
3	MIN	to find the minimum value of the attribute
4	SUM	to find the sum of values of the attribute provided the datatype of the attribute is number
5	AVG	to find the average of n values, ignoring null values
6	STDDEV	standard deviation of n values ignoring null values
7	VARIANCE	variance of n values ignoring null values

Table 4.3. PERSON_SKILL

Name	Skill
Ashok	fitter
Ashok	welder
Kumar	piping
Rajan	electrician
Ravi	turner
Sam	NULL

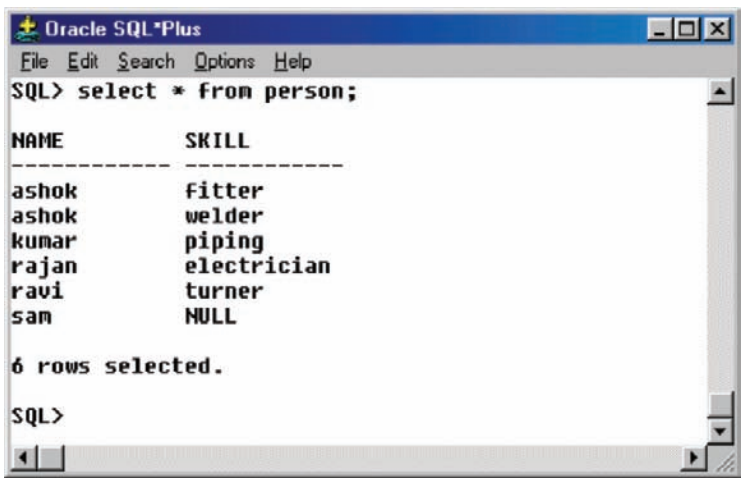


Fig. 4.11. PERSON table

Now let us try to view the table PERSON, and the contents of the table PERSON as shown in Fig. 4.11. From this figure, it is clear that the number of rows of the table is six.

Now let us use the COUNT (*) function to view the number of rows of the relation PERSON. The SQL command and the corresponding output are shown in Fig. 4.12.

From Fig. 4.12, we can observe that the number of rows returned is six, which means that the COUNT(*) function takes into account the NULL values.

(B) COUNT (attribute name) Function

A slight variation of COUNT (*) function is COUNT (attribute name) function. The syntax of this function is given by:

SELECT COUNT (attribute name)
FROM table name;

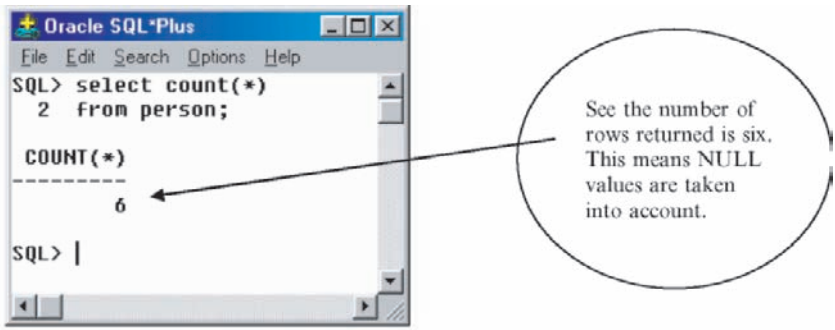


Fig. 4.12. COUNT (*) Function

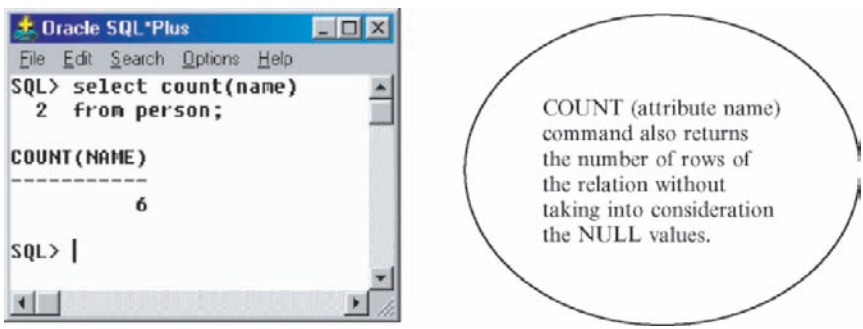


Fig. 4.13. SELECT (attribute name) command

The application of COUNT (attribute name) to the PERSON table and the corresponding output are shown in Fig. 4.13.

From Fig. 4.13, it is clear that count (attribute name) command will take NULL values into account as a result the number of rows selected is six.

(C) COUNT (DISTINCT attribute name)

The COUNT (DISTINCT attribute name) command returns the number of rows of the relation, by eliminating duplicate values. The syntax of COUNT (DISTINCT attribute name) is:

SELECT COUNT (DISTINCT attribute name)
FROM table name;

The usage of COUNT (DISTINCT attribute name) in the table PERSON and the corresponding output is shown in Fig. 4.14.

It is worthwhile to note that the DISTINCT command will not take into consideration the NULL value. In order to prove this, let us select the attribute be skill rather than the attribute name. The result of choosing the attribute as skill is show in Fig. 4.15.

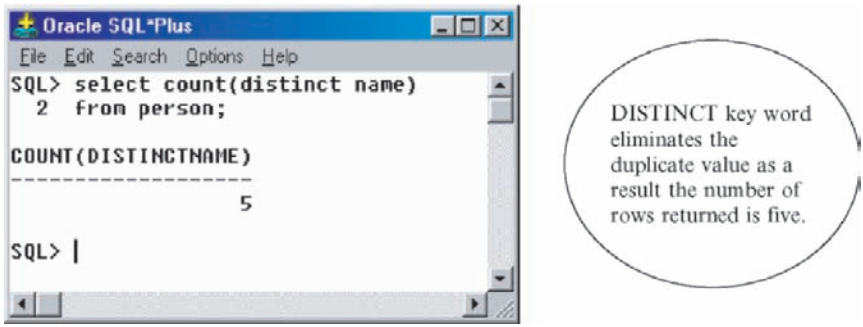


Fig. 4.14. COUNT (DISTINCT attribute name)

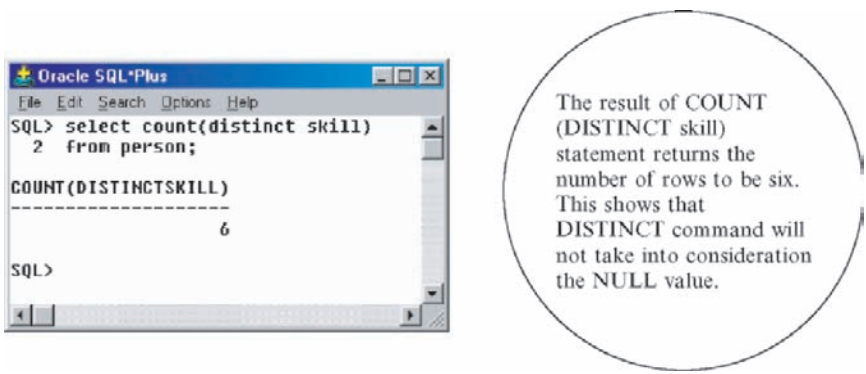


Fig. 4.15. COUNT command

4.8.2 MAX, MIN, and AVG Aggregate Function

In order to understand MAX, MIN, and AVG aggregate function consider the relation CONSUMER PRODUCTS. The relation CONSUMER PRODUCTS has two attributes, the name of the product and the price associated with the product as shown in Table 4.4.

(A) MAX Command

The MAX command stands for maximum value. The MAX command returns the maximum value of an attribute. The syntax of MAX command is:

SELECT MAX (attribute name)
FROM table name;

Let us apply the MAX command to Table 4.4 to get the maximum price of the product, the SQL command and the corresponding output are shown in Fig. 4.16.

Let us try to find the name of the product which has maximum price by using PROJECTION operation and the IN operator as shown in Fig. 4.17.

Table 4.4. Consumer product

Name	Price (in Rs.)
TV	15,000
refrigerator	10,000
washing machine	17,000
mixie	3,500

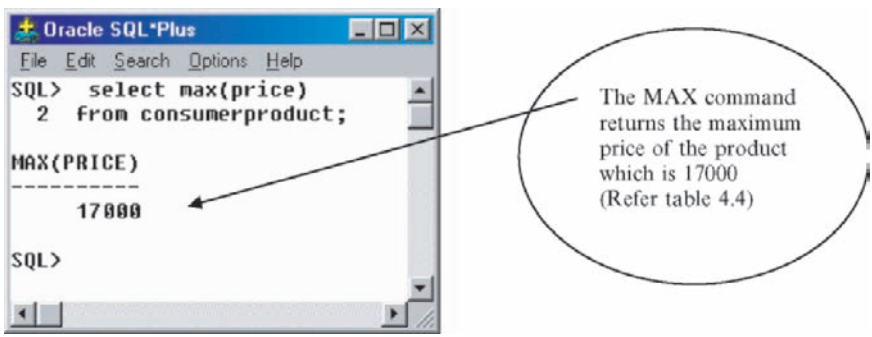


Fig. 4.16. MAX command

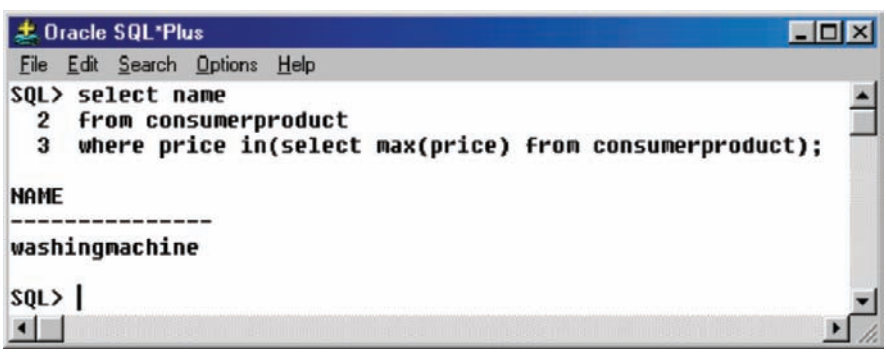


Fig. 4.17. Maximum price product name

(B) MIN Command

The MIN command is used to return the minimum value of an attribute. The syntax of MIN command is same as MAX command.

Syntax of MIN Command is

SELECT MIN (attribute name)
FROM table name;

The use of MIN command and the corresponding result are shown in Fig. 4.18.

From Table 4.4 the minimum price of the product is 3,500 which are returned as the result.

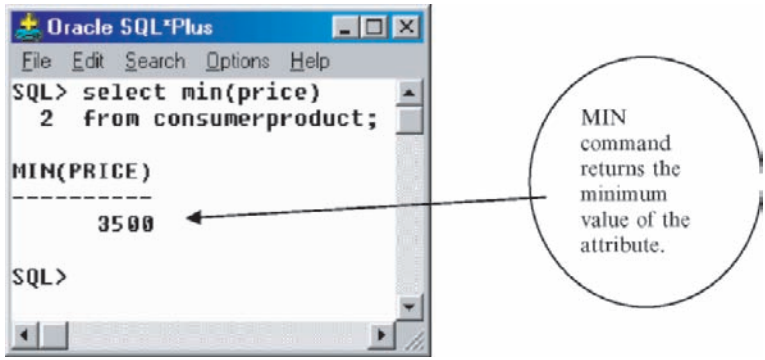


Fig. 4.18. MIN command applied to Table 4.4

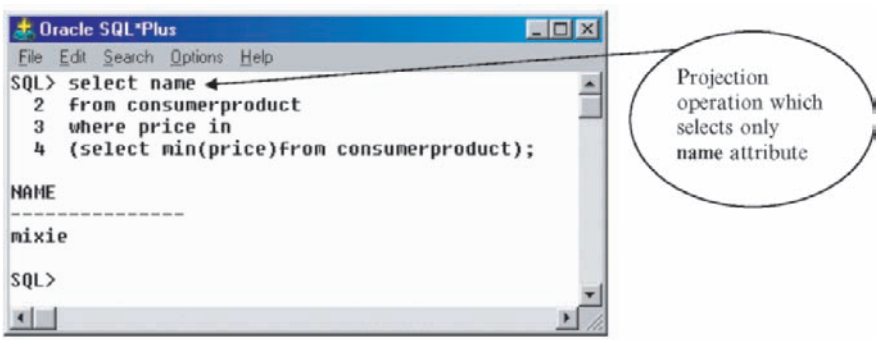


Fig. 4.19. Minimum price product name

To know the name of the product which has minimum price, we can use IN operator as shown in Fig. 4.19.

From Fig. 4.19, it is clear that we can use IN operator along with PROJECTION operation to get the name of the product with minimum price.

(C) AVG Command

The AVG command is used to get the average value of an attribute. The syntax of AVG command is:

SELECT AVG (attribute name)
FROM table name;

Let us apply AVG command to the Table 4.4, to get the average price of the product. The result of applying AVG command is shown in Fig. 4.20. The average price of the product is $(15,000 + 10,000 + 17,000 + 3,500)/4$ which is 11,375 as shown in Fig. 4.20.

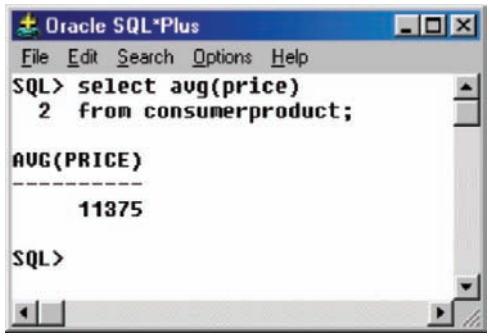


Fig. 4.20. AVG command

(D) STDDEV Function

The STDDEV function is used to compute the standard deviation of the attribute values. The syntax of the standard deviation function is:

SELECT STDDEV (attribute name)
FROM table name;

The STDDEV function applied to the relation CONSUMERPRODUCT (Table 4.4) is shown in Fig. 4.21.

(E) VARIANCE Function

The variance function is used to get the variance of the attribute values. The syntax of VARIANCE function is:

VARIANCE (attribute name)
FROM table name;

Let us apply the VARIANCE to the consumer product table; the result is shown in Fig. 4.22. We know that the variance is the square of the standard deviation. We have obtained the standard deviation from Fig. 4.21 as 6019.0669; the square of this value is approximately 36229167 which is obtained in Fig. 4.22.

(F) GROUP BY Function

The GROUP BY clause is used to group rows to compute group-statistics. It is to be noted that when the GROUP BY clause is present, then the SELECT clause may include only the columns that appear in the GROUP BY clause and aggregate functions.

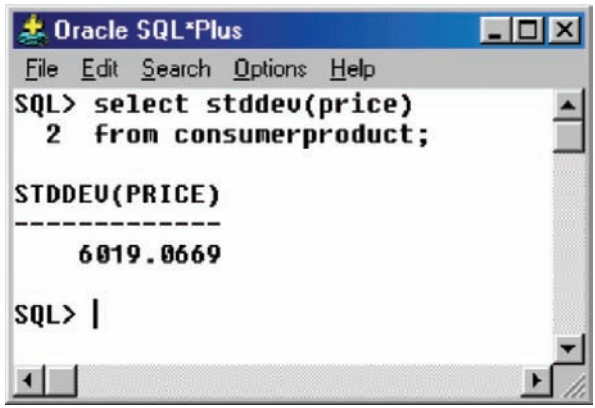


Fig. 4.21. STDDEV function

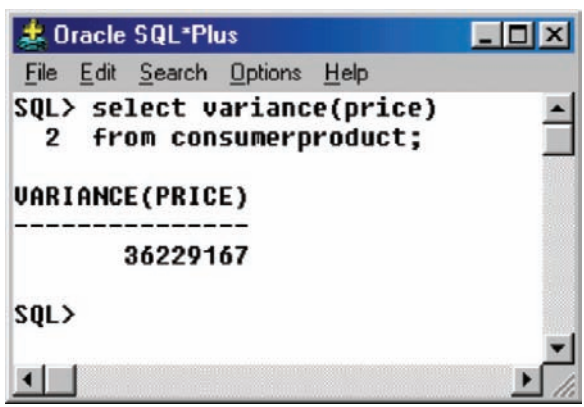


Fig. 4.22. Variance function

In order to understand the GROUP BY Function let us consider the table PLACEMENT as shown in Table 4.5 which refers to the number students placed in different companies. The table PLACEMENT consists of three attributes (columns) which are company name, department name which refers to the curriculum stream and strength which refers to the number of students placed.

Now we want to know the total number of students placed in each branch. For this we can use the GROUP BY command. The syntax of GROUP BY command is:

SELECT attribute name, aggregate function
FROM table name
GROUP BY attribute name;

Table 4.5. Placement

Company name	Department	Strength
TCS	CSE	54
TCS	ECE	40
TCS	EEE	32
GE	CSE	5
GE	ECE	8
GE	EEE	20
L&T	CSE	12
L&T	ECE	20
L&T	EEE	18
IBM	CSE	24
IBM	ECE	20
IBM	EEE	12

It is to be noted that the attribute name after SELECT command should match with the attribute name after GROUP BY command. The GROUP BY command which is used to find the total number of students placed in each branch is shown in Fig. 4.23.

(G) HAVING Command

The HAVING command is used to select the group. In other words HAVING restricts the groups according to a specified condition. The syntax of HAVING command is:

SELECT attribute name, aggregate function
FROM table name
GROUP BY attribute name
HAVING condition;

Let us use the HAVING command as shown in Fig. 4.24 to find the details of the department in which more than 90 students got placement.

From Fig. 4.24, we are able to get the details of the department where more than 90 students were placed.

(H) SORTING of Results

The SQL command ORDER BY is used to sort the result in ascending or descending order.

The table used to understand ORDER BY command is BESTCRICKETER. The table BESTCRICKETER as shown in Table 4.6 gives the details of best batsman of the world. The attributes of the BESTCRICKETER are the name of the batsman, the country they belong to, and the number of centuries they scored.

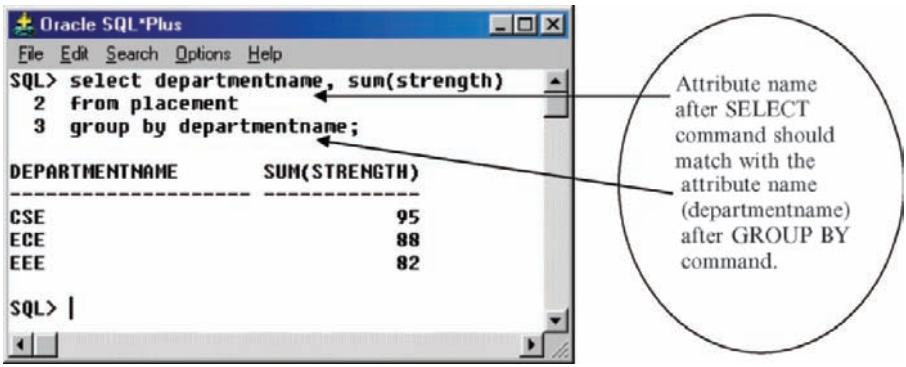


Fig. 4.23. GROUP BY command

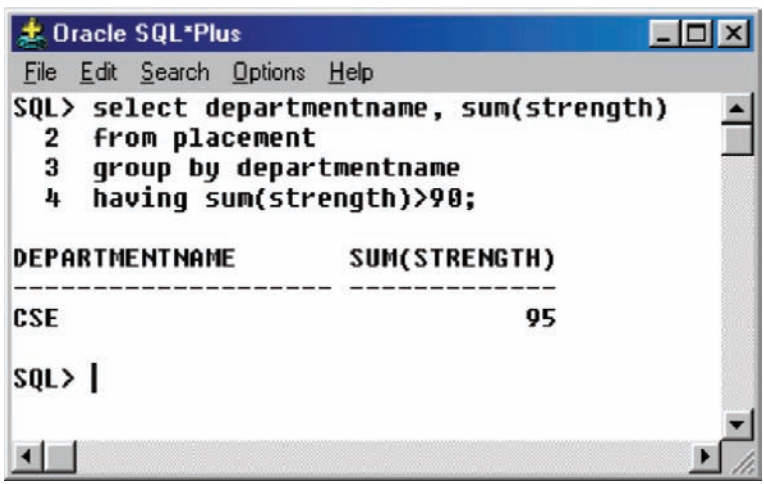


Fig. 4.24. GROUP BY and HAVING command

Table 4.6. BESTCRICKETER

Name	Country	Centuries
Gavaskar	India	34
Sobers	Westindies	26
Chappel	Australia	24
Bradman	Australia	29
Border	Australia	27
Gooch	England	20

Case 1: The syntax of ORDER BY command to arrange the result in ascending order is:

SELECT *
FROM table name
ORDER BY attribute name **ASC**;

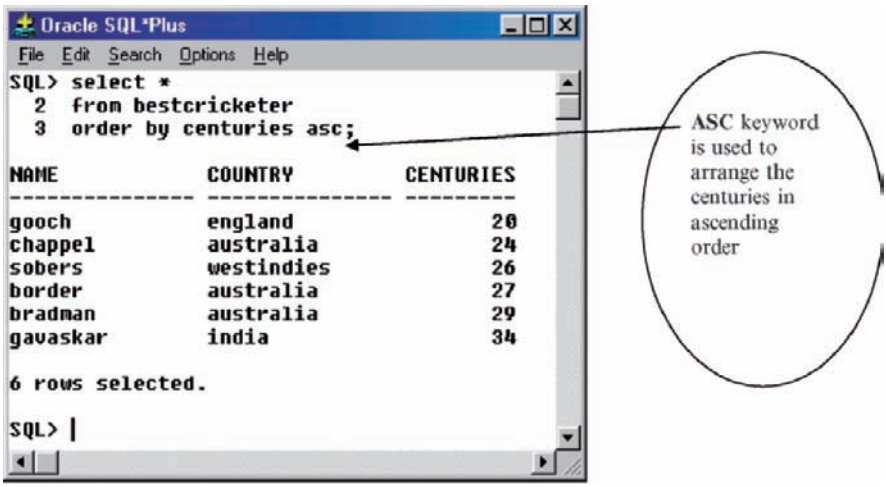


Fig. 4.25. Sorting in ascending order

Here ASC stands for ascending order.

Let us apply the command to the Table 4.6, the result of using ORDER BY command and the corresponding results are shown in Fig. 4.25.

Case 2: The syntax to arrange the result in descending order is:

SELECT *
FROM table name
ORDER BY attribute name **DESC**.

Here DESC stands for descending order.

Let us apply this DESC keyword to arrange the centuries in descending order. The SQL command and the corresponding output are shown in Fig. 4.26.

Case 3: If we do not specify as ASC or DESC after ORDER BY key word, by default, the results will be arranged in ascending order.

From Fig. 4.27, it is evident that if nothing is specified as ASC or DESC then by default, the results will be displayed in ascending order.

(I) Range Queries Using Between

The SQL has built-in command BETWEEN which is used to perform range queries.

Let us try to find the details of the batsman who has scored centuries greater than 20 and less than 30. The SQL command to accomplish this task and the corresponding output are shown in Fig. 4.28.

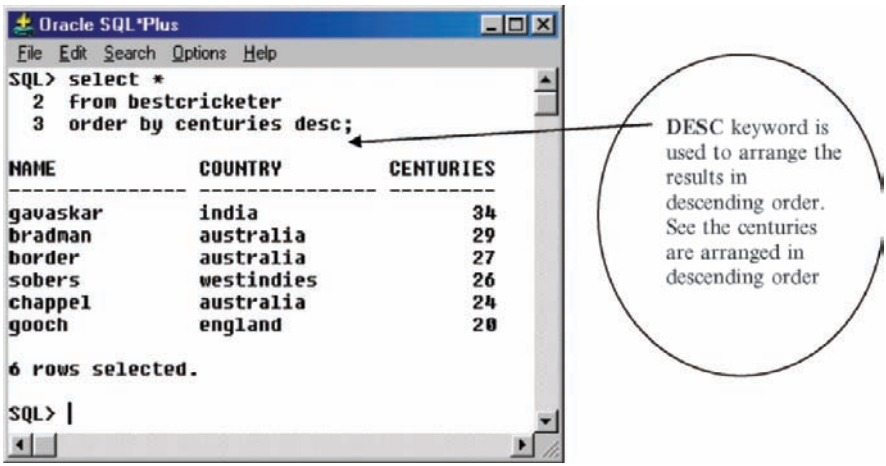


Fig. 4.26. Sorting in descending order

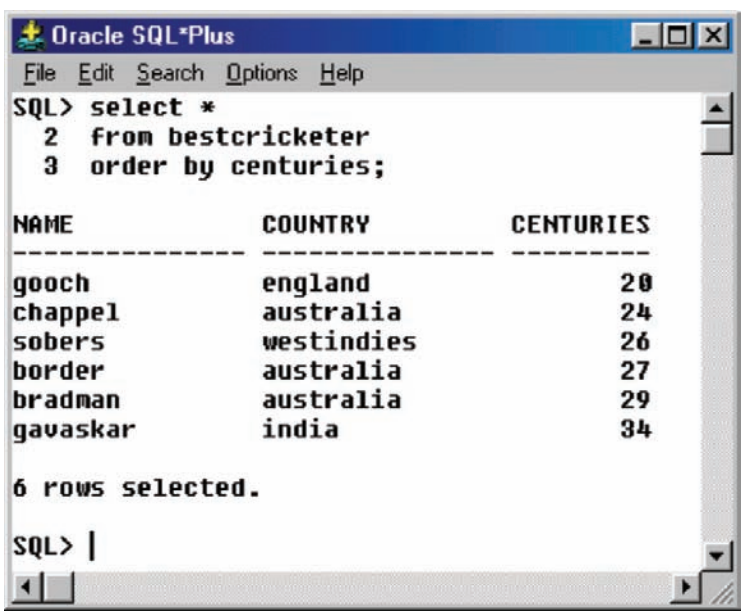


Fig. 4.27. Ascending order

4.9 Data Manipulation Language

The data manipulation language is used to add, update, and delete data in the database. The SQL command INSERT is used to add data into the database, the SQL command UPDATE is used to modify the data in the database, and the SQL command DELETE is used to delete data in the database. Here the term database refers to the table.

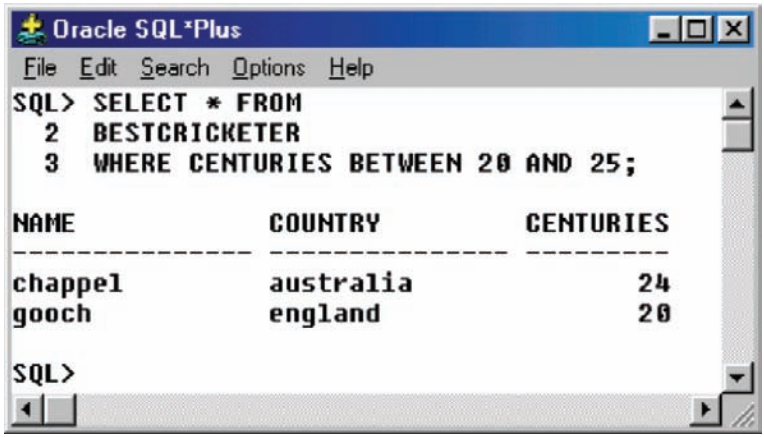


Fig. 4.28. Range query using BETWEEN command

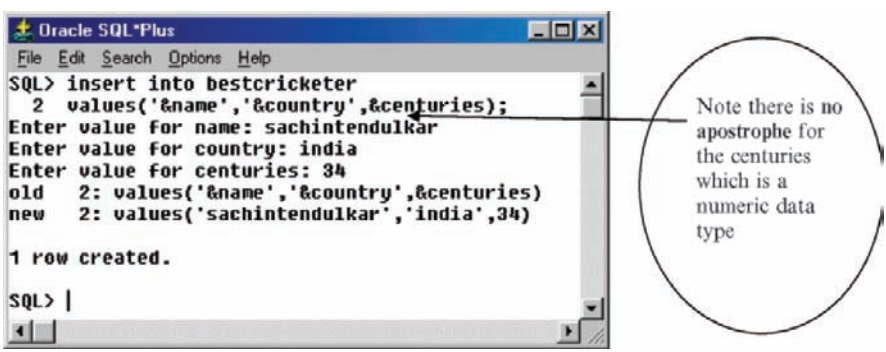


Fig. 4.29. Inserting a new row to the table

4.9.1 Adding a New Row to the Table

The INSERT command is to add new row to the table. The syntax of INSERT command is:

INSERT INTO table name
VALUES ('&column1-name', '&column2-name'... &columnN-name);

It is to be noted that apostrophe is not required for numeric datatype.

Let us try to insert a new row to the Table 4.6 (which has already six rows) to include the little master Sachin Tendulkar. The SQL command and the corresponding output are shown in Fig. 4.29.

To verify whether the new row has been added to the Table 4.6 which had six rows before inserting the new row, let us issue SELECT command as shown in Fig. 4.30.

From Fig. 4.30, it is clear that little master Sachin Tendulkar record being added to the best cricketer table so that the total number of rows is seven.

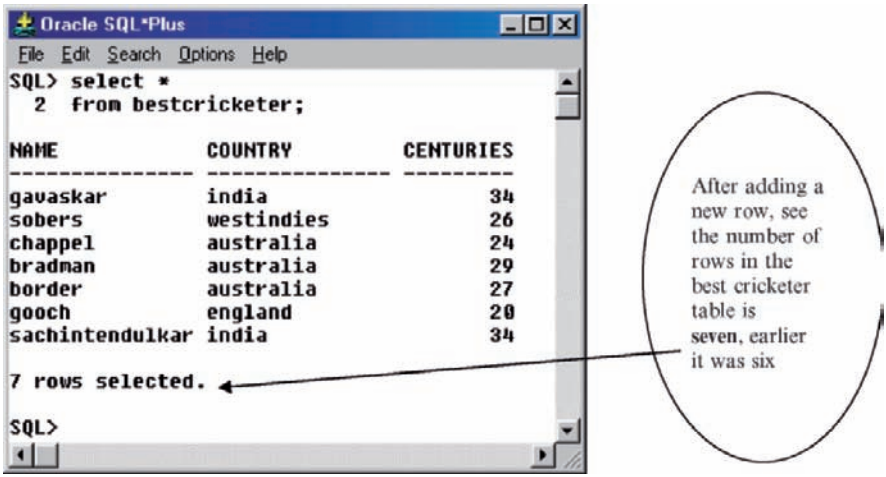


Fig. 4.30. Modified table

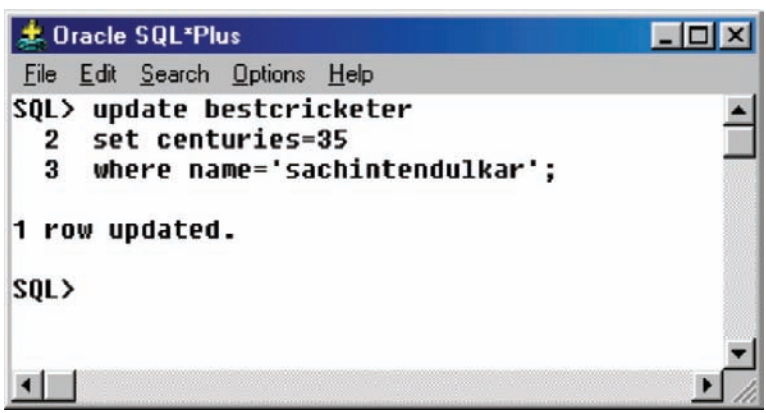


Fig. 4.31. Table updation using UPDATE command

4.9.2 Updating the Data in the Table

The data in the table can be updated by using UPDATE command. The syntax of the UPDATE command is:

UPDATE table name
SET attribute value=new value
WHERE condition;

Let us apply this UPDATE command to the table BESTCRICKETER. The motive is to modify the number of centuries hit by Sachin Tendulkar to 35. The corresponding SQL command and the output are shown in Fig. 4.31.

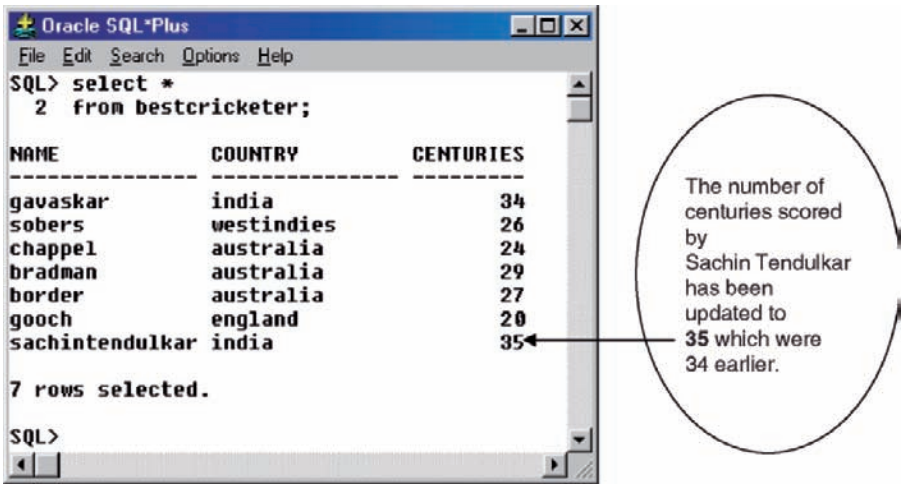


Fig. 4.32. Updated table BESTCRICKETER

To see whether the table has been updated or not use SELECT statement to view the content of the table BESTCRICKETER. The updated table is shown in Fig. 4.32.

4.9.3 Deleting Row from the Table

The DELETE command in SQL is used to delete row(s) from the table. The syntax of DELETE command is

DELETE FROM table name
WHERE condition;

Let us delete the record of a particular player (say Gooch) from the table BESTCRICKETER. The SQL command to delete a particular row and the corresponding output are shown in Fig. 4.33.

To verify whether the player Gooch record has been deleted, let us use SELECT command to view the content of the table as shown in Fig. 4.34. From this figure it is evident that the player Gooch record has been successfully deleted.

4.10 Table Modification Commands

We can use ALTER command to alter the structure of the table, that is we can add a new column to the table. It is also possible to delete the column from the table using DROP COLUMN command.

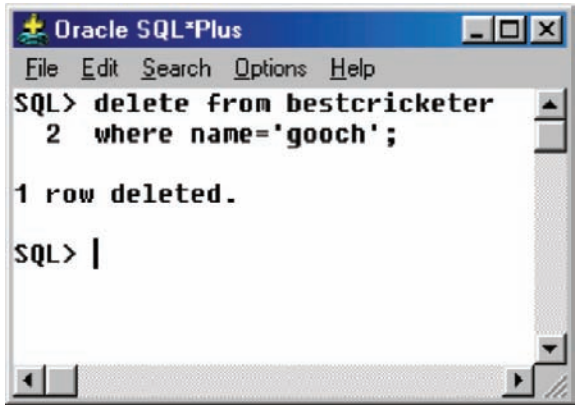


Fig. 4.33. Deletion of row from table

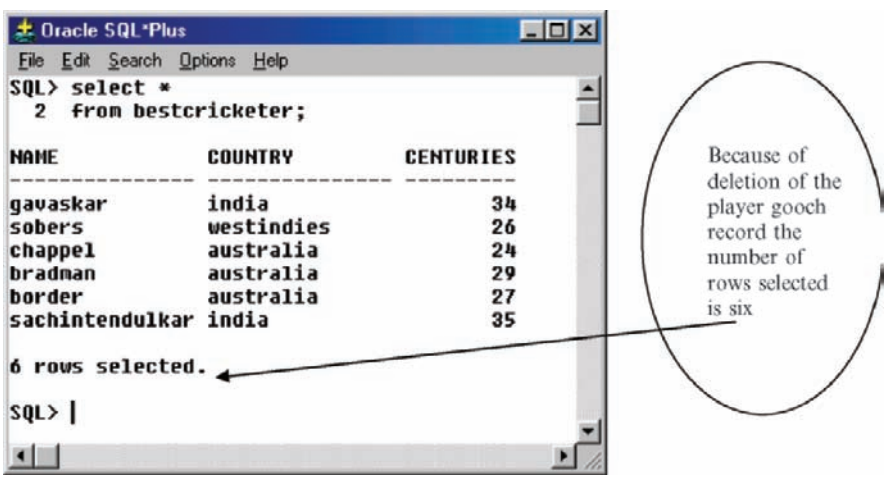


Fig. 4.34. Modified table

4.10.1 Adding a Column to the Table

We can add a column to the table by using ADD command. The syntax to add a new column to the table is:

ALTER TABLE table name
ADD column name datatype;

Example to Add a New Column

Let us consider the Table 4.6 BESTCRICKETER, which has three columns which are **name** of the player, **country** the player belong to, and the **centuries** which refer to the number of centuries scored by the player. Now try

to add one more column to the table BESTCRICKETER. The new column to be added is **age** which refers to player age. The SQL command to add the new column age and the corresponding output are shown in Fig. 4.35.

To see the description of the table after adding the new column **age** to the table bestcricketer, let us use DESC command as shown in Fig. 4.36.

From Fig. 4.36 we can observe that a new column **age** of datatype number has been added to the table bestcricketer.

After successfully inserting the column **age**, we will be interested to know the content of the table to see any value is assigned to the column age. Figure 4.37 shows the content of the table after adding a new column.

From Fig. 4.37, it is clear that the table already contains rows when the column **age** is added, then the new column **age** is initially null for all the rows.

To Insert Values into the New Column

Data can be inserted to the newly added column (in our example it is *age*) by using UPDATE command.

For example, we want to insert the age of Sachin Tendulkar to be 33. This is done using UPDATE command as shown in Fig. 4.38.

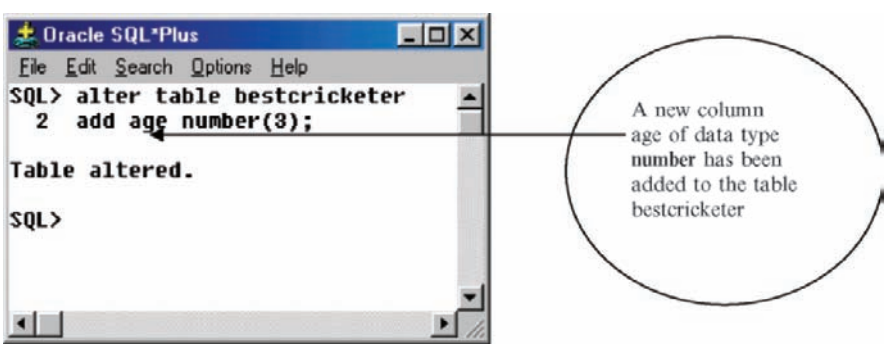


Fig. 4.35. Adding a column to the table

Fig. 4.36. Table descriptions after the addition of new column

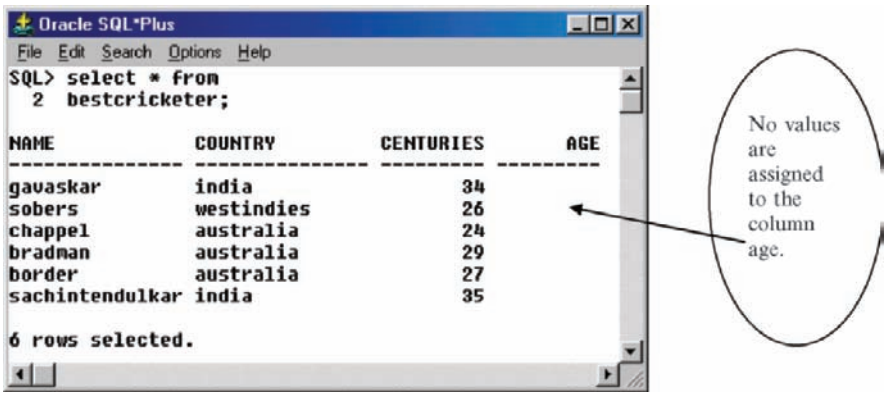


Fig. 4.37. Content of the table after the insertion of new column

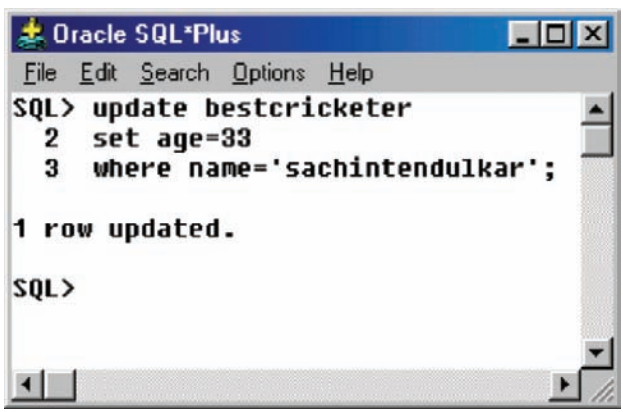


Fig. 4.38. Insertion of data to the new column age

To verify whether the age of sachintendulkar has been added as 33, see Fig. 4.39.

4.10.2 Modifying the Column of the Table

We can modify the width of the datatype of the column by using ALTER and MODIFY command. The syntax to change the datatype of the column is:

ALTER table name
MODIFY column-name datatype;

Example to Modify the Width of the Datatype of the Column

For example, we want to modify the width of the datatype **age** which is three as shown in Fig. 4.36 to four. The SQL command and the corresponding output are shown in Fig. 4.40.

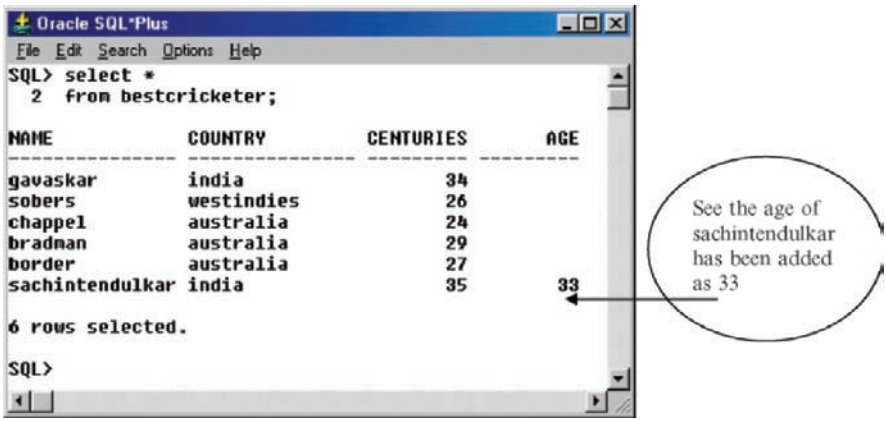


Fig. 4.39. Modified table

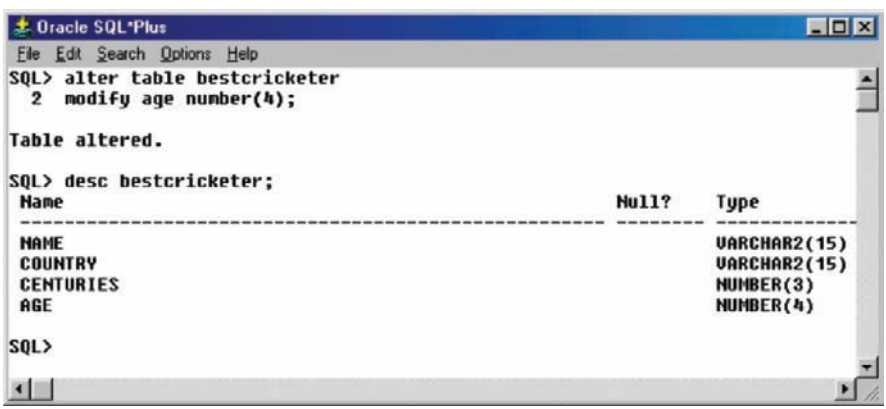


Fig. 4.40. Modified width of the datatype

From Fig. 4.40 we can observe that the width of the datatype **age** modified as four which was three earlier as shown in Fig. 4.36.

4.10.3 Deleting the Column of the Table

The DROP COLUMN command can be used along with the ALTER table command to delete the column of the table. The syntax to delete the column from the table is:

ALTER table name
DROP COLUMN column name;

Example

Let us try to delete the column **age** from the BESTCRICKETER by using DROP COLUMN command. The syntax to drop the column and the corresponding output are shown in Fig. 4.41.

After dropping the column **age**, the description of the table will be as shown in Fig. 4.42.

From Fig. 4.42, it is evident that the column **age** is not included in the table description.

The content of the table after dropping the column age is shown in Fig. 4.43.

4.11 Table Truncation

The TRUNCATE TABLE command removes all the rows from the table. The truncate table also releases the storage space used by the table. The syntax of TRUNCATE command is:

TRUNCATE TABLE table name;

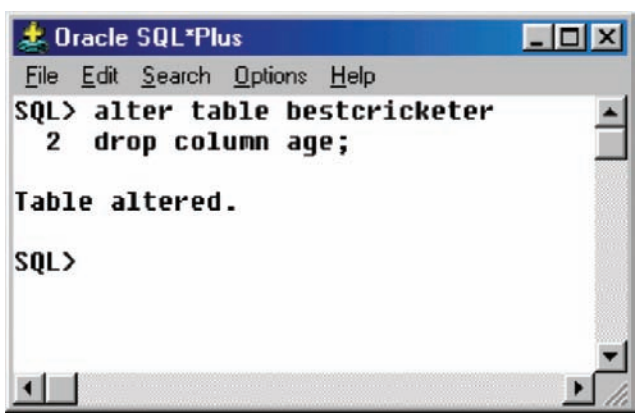


Fig. 4.41. Dropping a column from the table

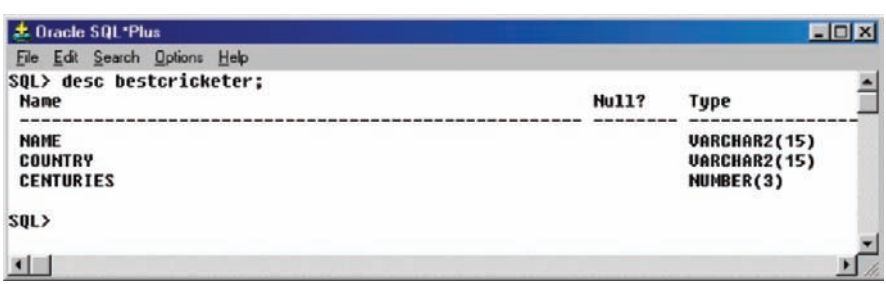


Fig. 4.42. Table descriptions after dropping the column age

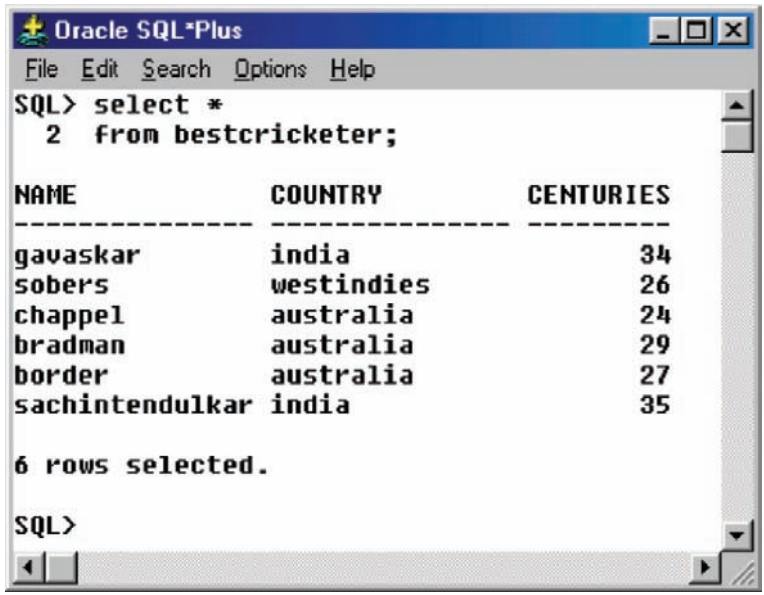


Fig. 4.43. Table content after dropping the column age

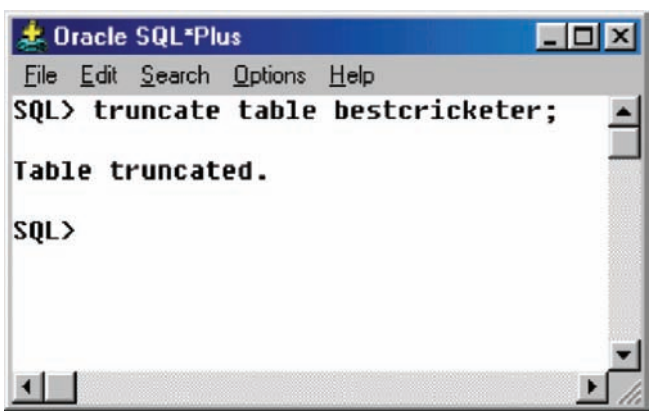


Fig. 4.44. Table truncation

Example

Let us try to delete all the rows of the table bestcricketer by issuing TRUNCATE TABLE command. The SQL command and the corresponding output are shown in Fig. 4.44.

After table truncation, if we try to select the rows, what will be the output? To answer this question, let us try to see the content of the table by using SELECT command as shown in Fig. 4.45.

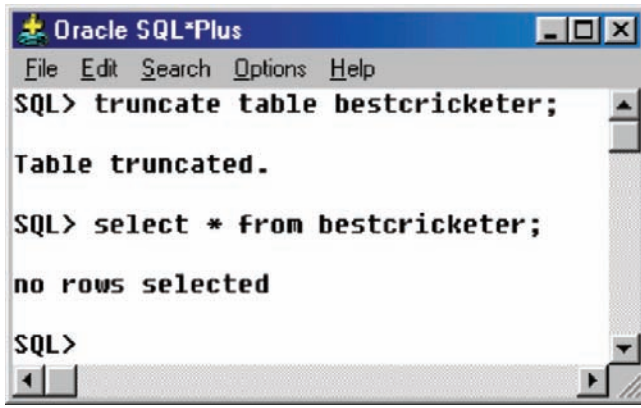


Fig. 4.45. Table content after truncation

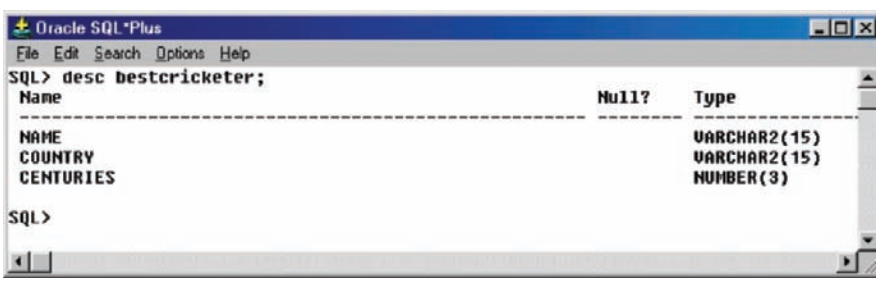


Fig. 4.46. Table descriptions after table truncation

From Fig. 4.45, it is clear that all the rows are deleted by issuing TRUNCATE TABLE command. After the TRUNCATE TABLE command if we try to see the description of the table by issuing DESC command as shown in Fig. 4.46.

From Fig. 4.46, it is clear that the TRUNCATE TABLE command deletes the content (all rows) of the table but not the table definition.

Note Another way to delete all the rows of the table is to use DELETE command. The syntax is:

DELETE FROM table name;

4.11.1 Dropping a Table

The definition of the table as well as the contents of the table is deleted by issuing DROP TABLE command. The syntax of DROP TABLE command is:

DROP TABLE table name;

Example

Let us issue the DROP TABLE command to the table BESTCRICKETER as shown in Fig. 4.47.

After issuing the DROP TABLE command if we try to see the description of the table, we will get the result as shown in Fig. 4.48.

From Fig. 4.48 it is clear that DROP TABLE command deletes both the content and the descriptions of the table.

4.12 Imposition of Constraints

Constraints are basically used to impose rules on the table, whenever a row is inserted, updated, or deleted from the table. Constraints prevent the deletion of a table if there are dependencies. The different types of constraints that

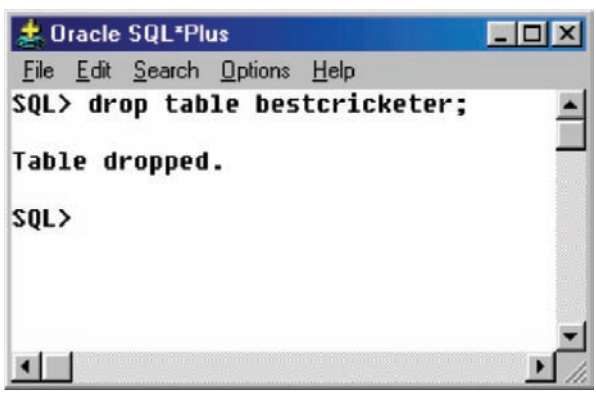


Fig. 4.47. Dropping a table

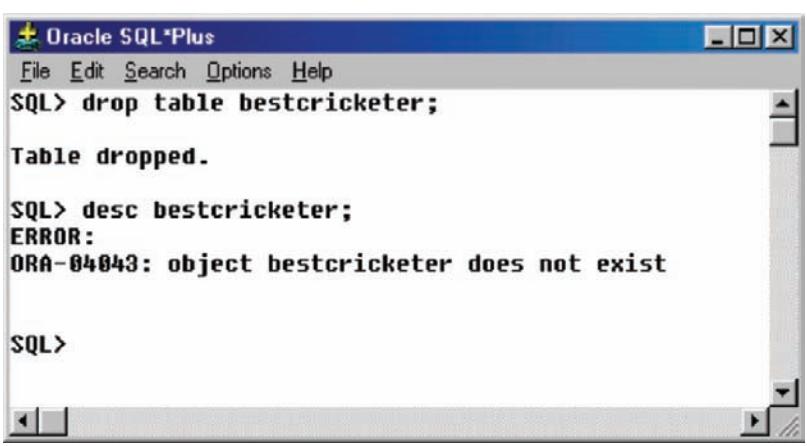


Fig. 4.48. Table descriptions after dropping the table

can be imposed on the table are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK.

Whenever an attribute is declared as NOT NULL then it specifies that the attribute cannot contain a NULL value.

The UNIQUE constraint specifies that whenever an attribute or set of attributes are specified as UNIQUE, then the values of the attribute should be unique for all the rows of the table. For example, consider the Roll number of the student in the class, every student should have UNIQUE roll number.

PRIMARY KEY constraint is used to identify each row of the table uniquely.

FOREIGN KEY constraint specifies that the value of an attribute in one table depends on the value of the same attribute in another table.

CHECK constraint defines a condition that each row must satisfy. Also there is no limit to the number of CHECK constraints that can be imposed on a column.

4.12.1 NOT NULL Constraint

If one is very much particular that the column is not supposed to take NULL value then we can impose NOT NULL constraint on that column. The syntax of NOT NULL constraint is:

CREATE TABLE table name
(column name1, data-type of the column1, **NOT NULL**
column name2, data-type of the column2,
column nameN, data-type of the columnN);

The above syntax indicates that column1 is declared as NOT NULL.

Example

Consider the relation PERSON, which has the attributes **name** of the person, **salary** of the person, **phone number** of the person. Let us try to declare the column **name** as **NOT NULL**. This implies that every person should have a name. The syntax to declare the column **name** as NOT NULL is shown in Fig. 4.49.

From Fig. 4.49, it is clear that the attribute **name** is declared as NOT NULL. Now let us try to insert NOT NULL values and NULL value to the attribute **name**.

Case 1: Inserting a NOT NULL value to the attribute **name**.

From Fig. 4.50, it is clear that when we try to insert a NOT NULL name into the name attribute, the name is included in the relation **PERSON1**.

Case 2: A NULL value to the attribute **name**.

From Fig. 4.51, it is clear that when we try to insert a NULL value into the **PERSON1** relation, we get the error message as shown in Fig. 4.51 since the attribute **name** is declared as **NOT NULL**.

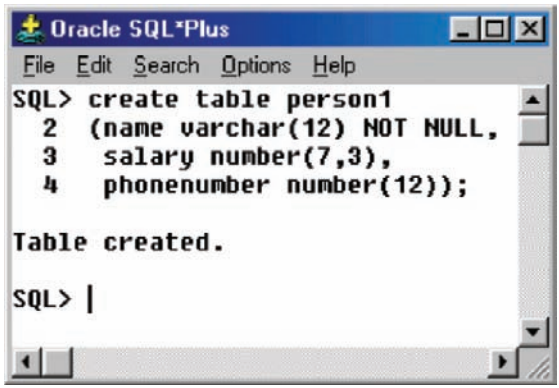


Fig. 4.49. NOT NULL constraint

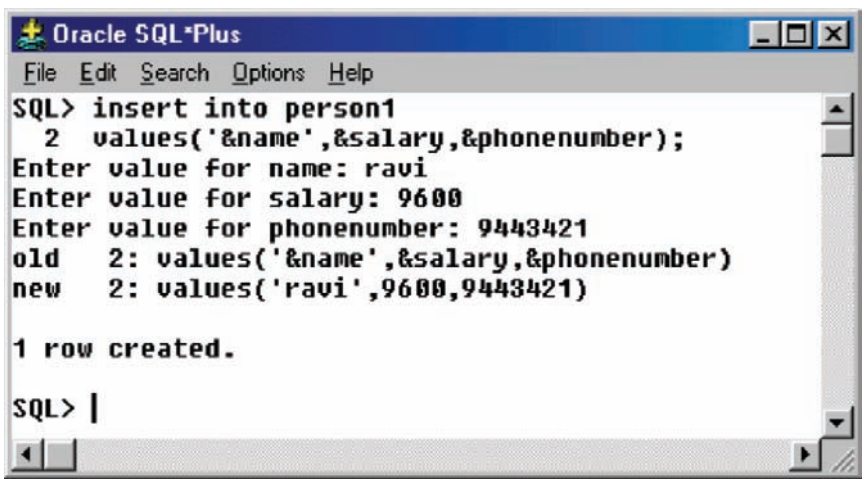


Fig. 4.50. A NOT NULL value to the attribute name

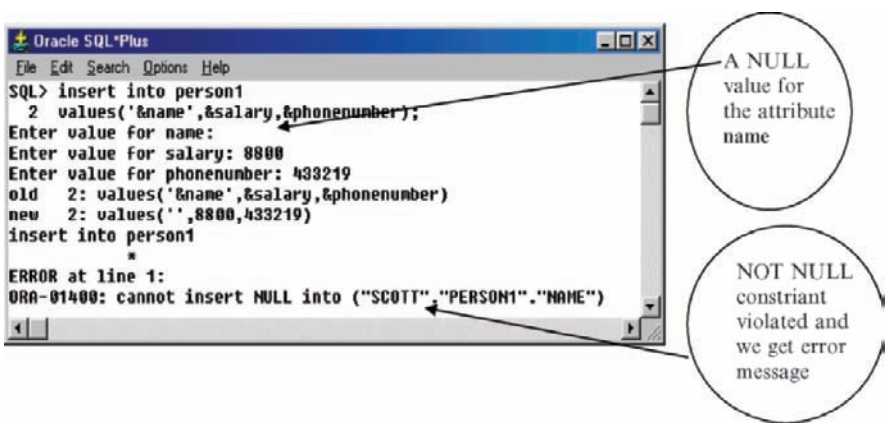


Fig. 4.51. NOT NULL constraint violated

4.12.2 UNIQUE Constraint

The UNIQUE constraint imposes that every value in a column or set of columns be unique. It means that no two rows of a table can have duplicate values in a specified column or set of columns.

Example

In order to understand unique constraint, let us create the table CELLPHONE, which has three attributes. The three attributes are **model** of the cellphone, **make** which refers to manufacturer, and the **price**.

The relation CELLPHONE is created as shown in Fig. 4.52 with unique constraint on **model**. When a **unique** constraint is imposed on the attribute **model**, then no two models should have same number.

The values are inserted into the table CELLPHONE. The resulting tables after inserting the values are shown in Fig. 4.53.

From Fig. 4.53, we can observe that the table CELLPHONE has three rows.

Case 1: Now let us try to insert a row in the relation CELLPHONE by violating the UNIQUE constraint, i.e., we are trying to insert a row with model number 1100 which already exists. The insertion and the corresponding result are shown in Fig. 4.54. From this figure, we can observe that there is an error message "unique constraint (SCOTT.SYS_C00820) violated." The reason for getting this error message is we tried to enter the model (1100) which exists already in the CELLPHONE relation as shown in Fig. 4.53.

Case 2: Insertion of NULL Value to the Model Attribute. Let us try to insert a null value to the attribute model. The SQL command to insert a null value to the attribute model and the corresponding result are shown in Fig. 4.55.

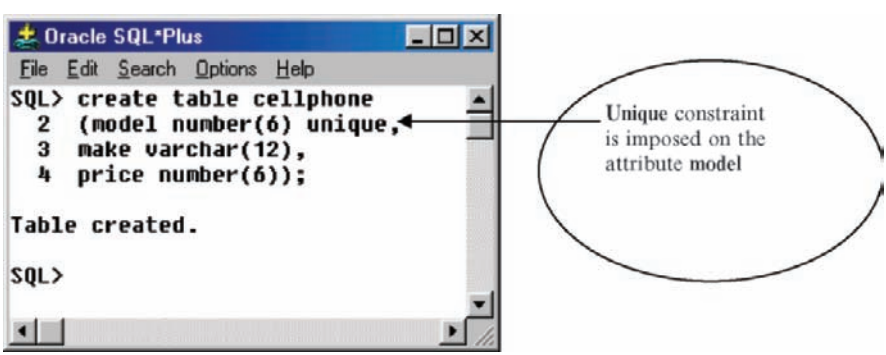


Fig. 4.52. Unique constraint on a column

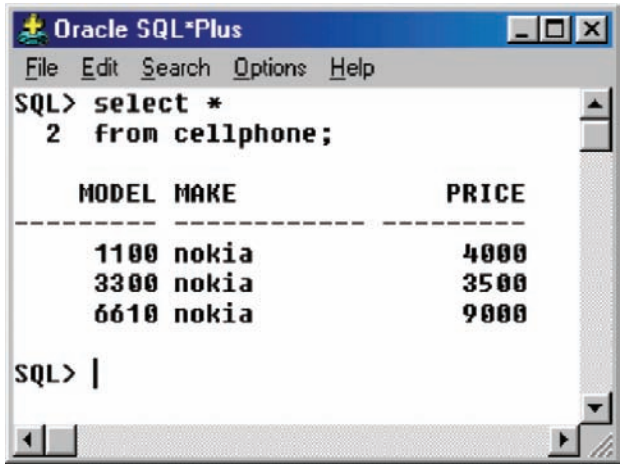


Fig. 4.53. Values inserted into the table CELLPHONE

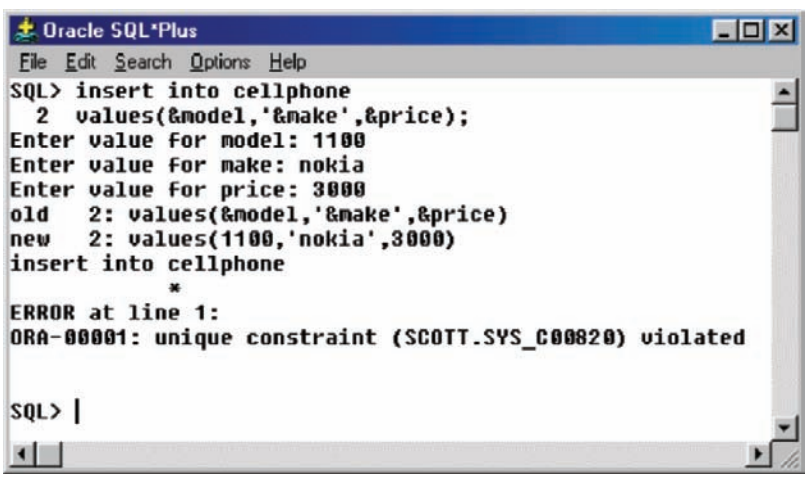


Fig. 4.54. Violation of UNIQUE constraint

Difference Between NOT NULL and UNIQUE Constraint

The unique constraint accepts NULL value as shown in Fig. 4.55, whereas the NOT NULL constraint will not accept NULL values.

Note NOT NULL constraint accepts duplicate values, whereas UNIQUE constraint will not accept null values. Moreover when a UNIQUE constraint is imposed on an attribute means that attribute can accept NULL values. Whereas NOT NULL constraint will not accept NULL values.

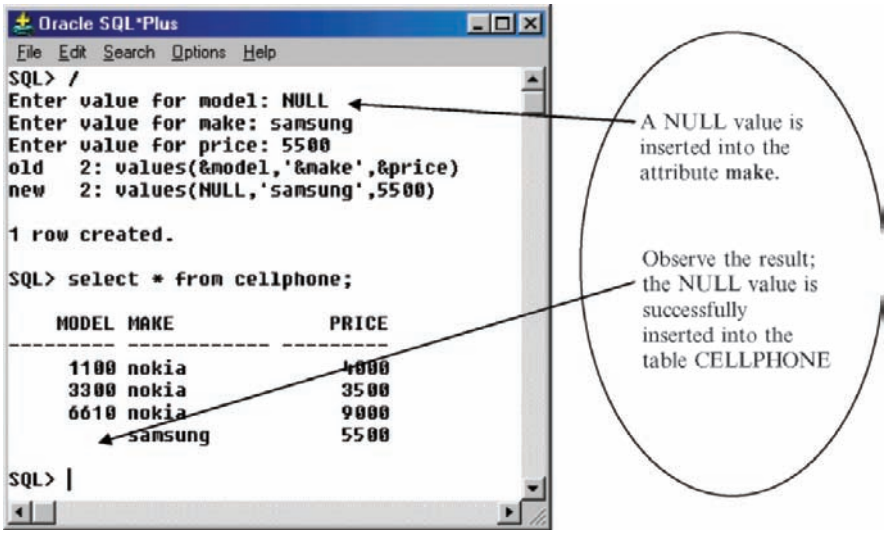


Fig. 4.55. Insertion of NULL value into CELLPHONE

4.12.3 Primary Key Constraint

When an attribute or set of attributes is declared as the primary key, then the attribute will not accept NULL value moreover it will not accept duplicate values. It is to be noted that "only one primary key can be defined for each table."

Example

Consider the relation EMPLOYEE with the attributes **ID** which refers to Employee identity, **NAME** of the employee, and **SALARY** of the employee. Each employee will have unique ID hence ID is declared as the primary key as shown in Fig. 4.56.

From Fig. 4.56, it is clear that the attribute employee ID is declared as the primary key.

Case 1: Insertion of NULL Value to the Primary Key Attribute.

It is to be noted that the primary key will not take any NULL value. This is called entity integrity. Now let us try to insert a NULL value to the employee ID in the SQL syntax, and the corresponding output is shown in Fig. 4.57. From Fig. 4.57, it is evident that an attribute or set of attributes declared as primary key will not accept NULL values.

Case 2: Insertion of Duplicate Values into an Attribute Declared as Primary Key.

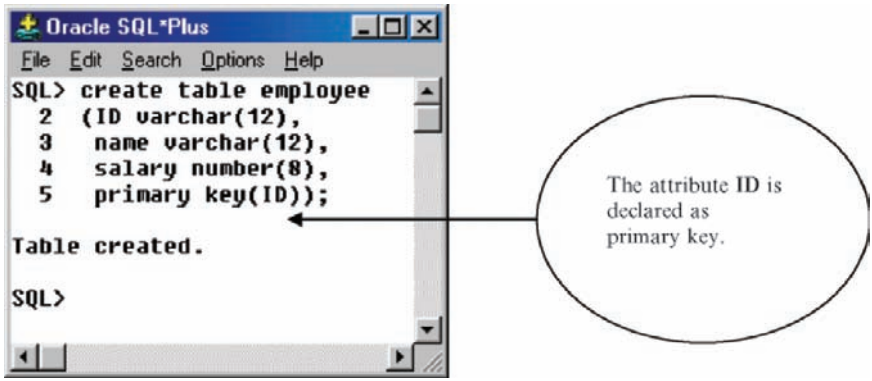


Fig. 4.56. Attribute declared as primary key

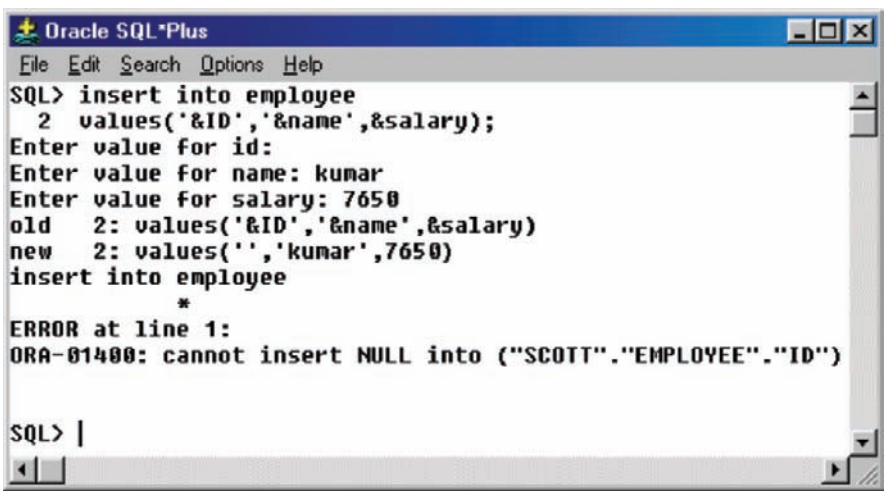


Fig. 4.57. Inserting NULL value into primary key attribute

When an attribute is declared as primary key, all the values of the attribute should be UNIQUE. The primary key attribute will not accept duplicate values.

Let us try to insert duplicate values to the attribute employee ID which is declared as primary key. The SQL command and the corresponding output are shown in Fig. 4.58.

We got an error message in Fig. 4.54, because we have tried to insert the employee **ID** *e*101 twice. From this we can understand that when an attribute is declared as primary key, the values of the attribute should be UNIQUE.

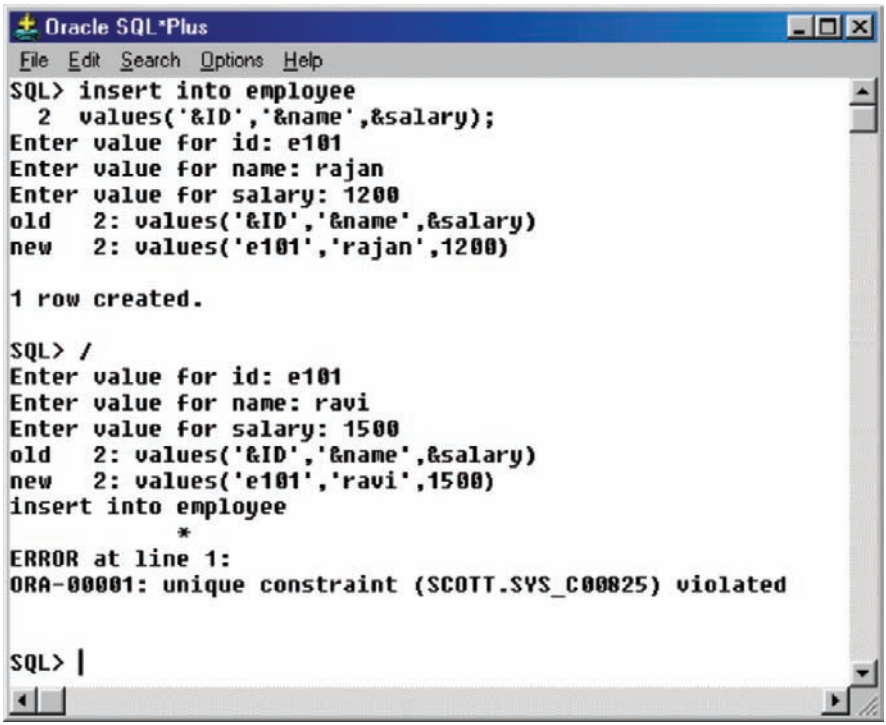


Fig. 4.58. Insertion of duplicate values to an attribute declared as primary key

Difference Between UNIQUE and NOTNULL Constraint

The difference between UNIQUE and NOTNULL constraint is given in the tabular form as

NOTNULL constraint	UNIQUE constraint
an attribute declared as NOTNULL will not accept NULL values	an attribute declared as UNIQUE can accept NULL values
an attribute declared as NOTNULL will accept duplicate values	an attribute declared as UNIQUE will not accept duplicate values

Difference Between UNIQUE and PRIMARY KEY Constraint

The difference between UNIQUE and PRIMARY KEY is given in tabular form as

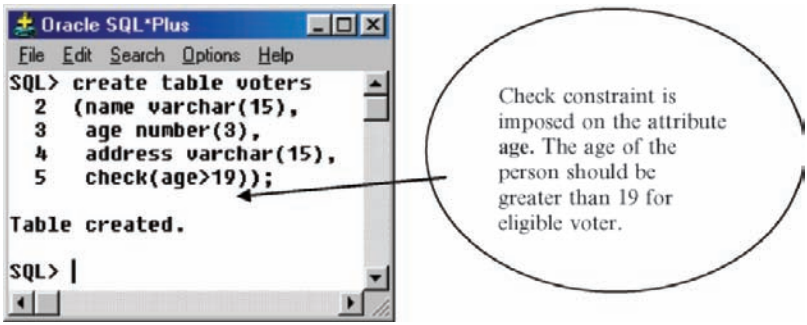


Fig. 4.59. Check constraint on an attribute

PRIMARY KEY constraint	**UNIQUE constraint**
an attribute declared as primary key will not accept NULL values	an attribute declared as UNIQUE will accept NULL values
only one PRIMARY KEY can be defined for each table	more than one UNIQUE constraint can be defined for each table

4.12.4 CHECK Constraint

CHECK constraint is added to the declaration of the attribute. The CHECK constraint may use the name of the attribute or any other relation or attribute name may in a subquery. Attribute value check is checked only when the value of the attribute is inserted or updated.

Syntax of CHECK Constraint

In order to understand check constraint, consider the relation **VOTERS**. In India, only those who have completed the age of 19 are eligible to vote. Let us impose this constraint on age in our relation **VOTERS**. The **VOTERS** relation has the attributes **name,** which refers to the name of the voter, **age** of the voter, **address** of the voter.

The creation of the table **VOTERS** with **CHECK** constraint imposed on **age** is shown in Fig. 4.59.

From Fig. 4.59, we can observe that CHECK constraint is imposed on the attribute **age**.

Case 1: Insertion of Data Without Violating the Constraint.

Let us try to insert the values into the table VOTERS without violating the constraint, that is the age of the voter is greater than 19. The SQL syntax and the corresponding output are shown in Fig. 4.60. From this figure, it is evident that the data are successfully inserted into the table VOTERS because the age of the voter is greater than 19.

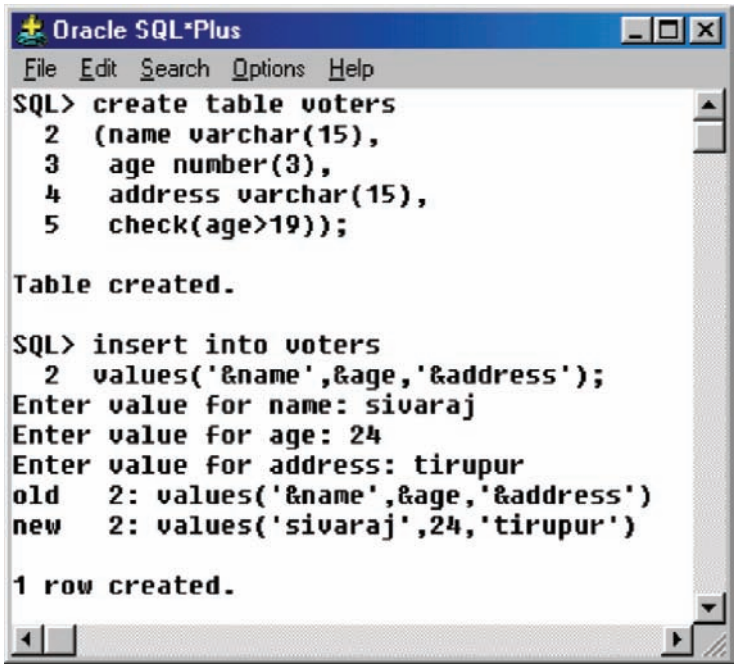


Fig. 4.60. Data insertion without violating the constraint

Case 2: Insertion of Data into the Table VOTERS by Violating the CHECK Constraint.

Now let us try to insert data into the table VOTERS by violating the CHECK constraint, that is inserting the record of the voter with age less than 19. The SQL command to insert the data and the corresponding output are shown in Fig. 4.61.

From Fig. 4.61, we can observe that we try to insert a value which violates the CHECK constraint, we get error message.

Case 3: CHECK Constraint During Updation of Record.

The content of the VOTER table is given in Fig. 4.62.

For simplicity, there is only one record in the VOTERS table. Now let us try to update the record by changing the age of the voter to less than 19, as shown in Fig. 4.63.

From Fig. 4.63, we can observe that it is not possible to update the record by violating the CHECK constraint.

4.12.5 Referential Integrity Constraint

According to referential integrity constraint, when a foreign key in one relation **references** primary key in another relation, the foreign key value must

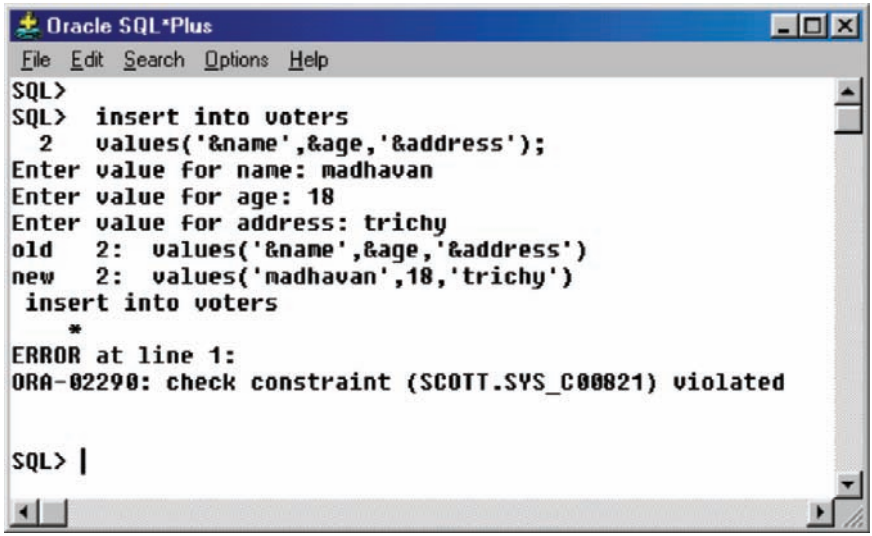


Fig. 4.61. Data insertion by violating the CHECK constraint

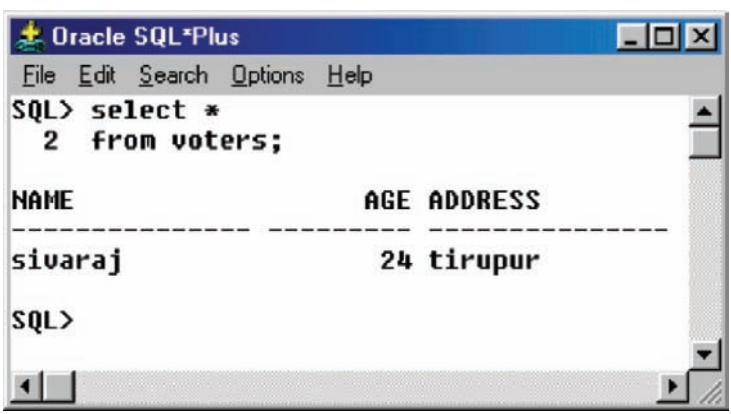


Fig. 4.62. The content of VOTERS table

match with the primary key value. In other words, the referential integrity says "pointed to" information must exist.

Example

In order to understand referential constraint, consider two relation DEPARTMENT and EMPLOYEE. Here the DEPARTMENT relation forms the parent table. The meaning is the DEPARTMENT table contains the primary key. The relation EMPLOYEE forms the child table. The meaning is the relation EMPLOYEE has foreign key which references to primary key in DEPARTMENT table. Figure 4.64 shows parent–child relationship.

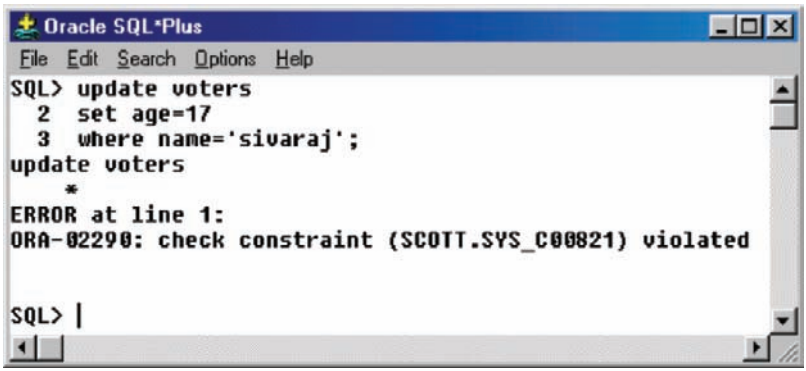


Fig. 4.63. Updation of record voters by violating CHECK constraint

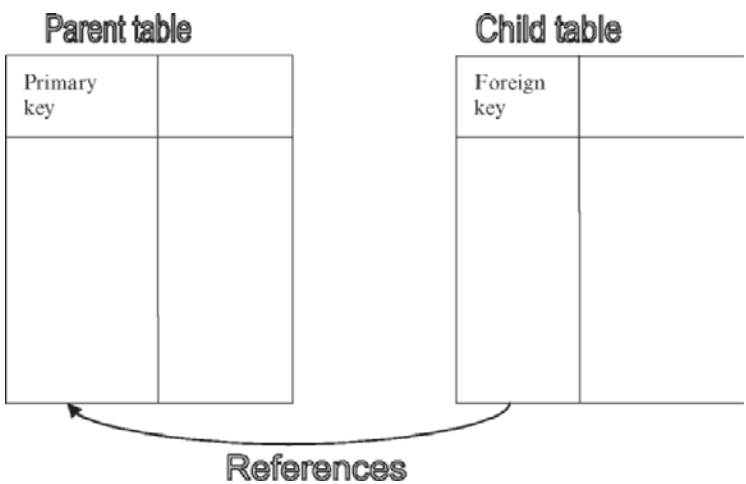


Fig. 4.64. Primary key and foreign key relationship

In our example, the relation DEPARTMENT is the parent table which holds the parent table, and the relation EMPLOYEE forms the child table which has foreign key which references primary key in DEPARTMENT table. It is to be noted that the parent table should be created first, then the child table.

DEPARTMENT

DeptID	Dname	Location
D100	electrical	B
D101	civil	A
D102	computer	C

EMPLOYEE

EID	DID	Ename
E201	D100	Raman
E202	D101	Ravi
E203	D101	Krishnan

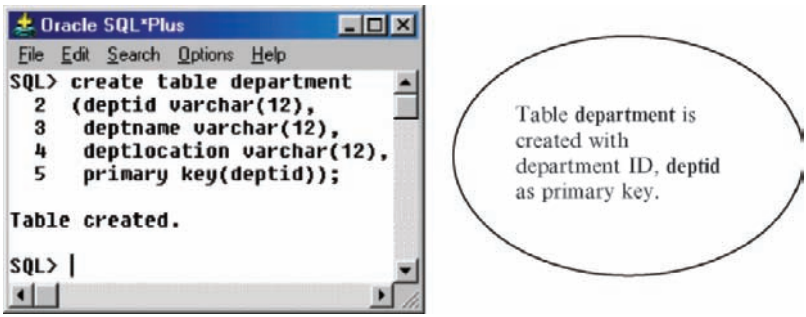


Fig. 4.65. DEPARTMENT table

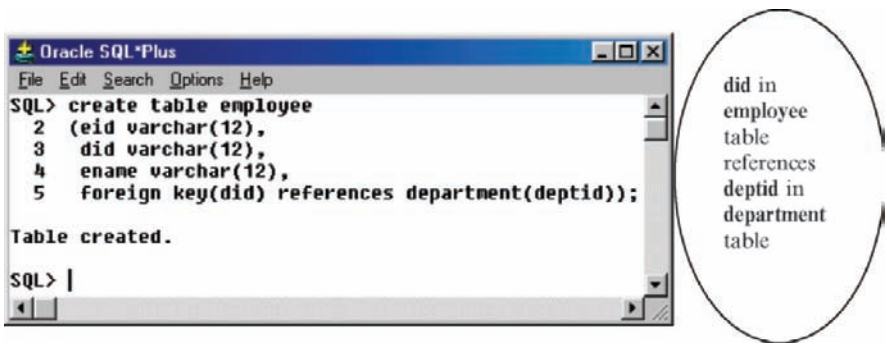


Fig. 4.66. EMPLOYEE table

The SQL syntax to create the two relations DEPARTMENT and EMPLOYEE with primary key and foreign key constraints is shown in Fig. 4.65 and Fig. 4.66, respectively.

Case 1: Now let us try to insert a value into DepartmentID of the **employee** table which is not in **department** table. The department relation has only three department IDs D100, D101, D102. Now we are trying to insert D103 in the **DID** (which stands for department ID) of employee table. The SQL command and the corresponding output are shown in Fig. 4.67.

From Fig. 4.67, it is evident that the values are not able to insert into the **employee** table. The reason for not able to insert value into the **employee** table is: we have tried to insert the **DID** (department id) into the **employee** table (child table) which is not matching with **DeptID** (department id) of the **department** table (parent table). In other words the foreign key value in the child table does not match with the primary key value in the parent relation.

The referential integrity rule says that the foreign key value should match with the primary key value.

Case 2: NULL Value into Foreign Key Attribute.

Now let us try to insert a null value into the foreign key attribute. The SQL command and the corresponding output are shown in Fig. 4.68.

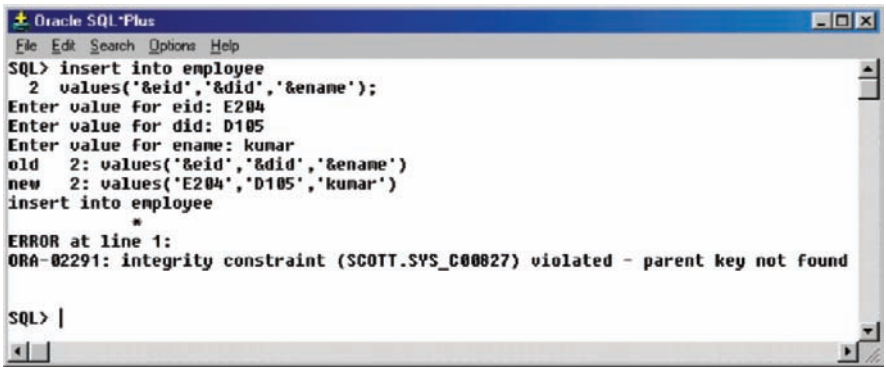


Fig. 4.67. Violation of referential integrity

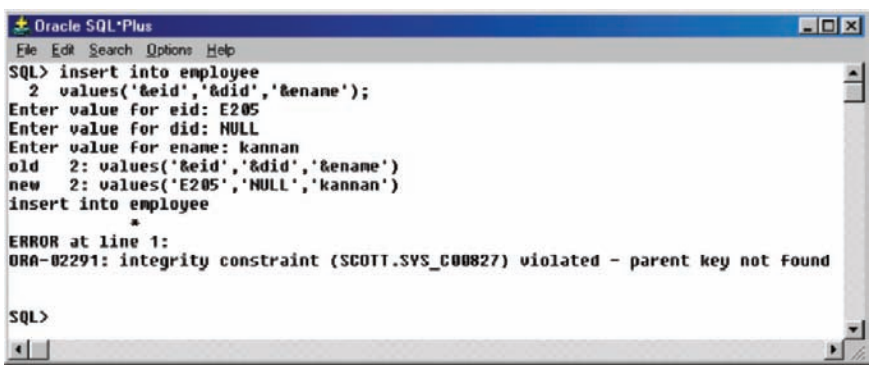


Fig. 4.68. NULL value to the foreign key attribute

From Fig. 4.68, it is evident that NULL value cannot be inserted into foreign key attribute unless it matches with the primary key attribute.

4.12.6 ON DELETE CASCADE

When the clause ON DELETE CASCADE is included in the child table, and if a row is deleted from the parent table then the corresponding referenced value in the child table will also be deleted.

Example

Let us consider the DEPARTMENT (parent table) and EMPLOYEE (child table) relation. The employee relation is modified as shown in Fig. 4.69. From this figure, it is clear that we have included the clause ON DELETE CASCADE in the child table.

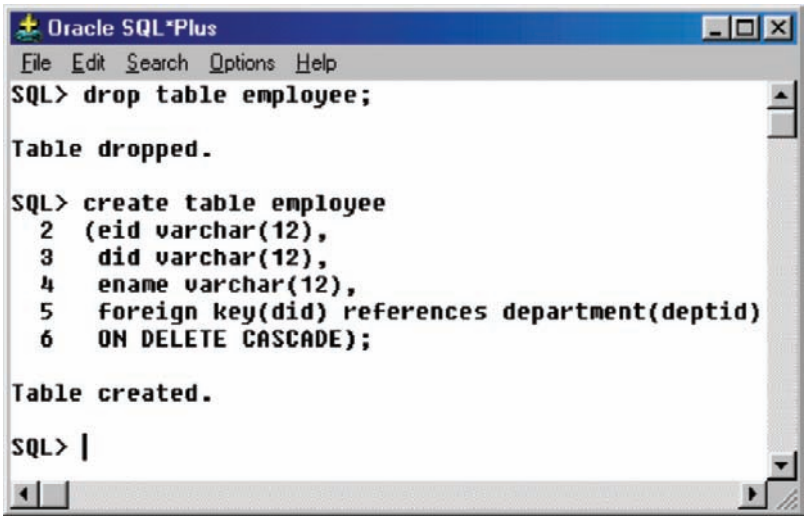


Fig. 4.69. Modified EMPLOYEE relation

The content of the table DEPARTMENT and EMPLOYEE are shown below.

DEPARTMENT

DeptID	Dname	Location
D100	electrical	B
D101	civil	A
D102	computer	C

EMPLOYEE

EID	DID	Ename
E201	D100	Raman
E202	D101	Ravi
E203	D101	Krishnan

Now let us try to delete the department "Civil" in the DEPARTMENT table. If we delete the row "civil" in the DEPARTMENT table, what will be the impact in the EMPLOYEE table?

First the content of employee table is shown in Fig. 4.70. The number of tuples in the EMPLOYEE relation is three.

Now we are going to delete the department "civil" in the table DEPARTMENT. The SQL command and the corresponding output are shown in Fig. 4.71.

Now let us see the impact of deleting the record "civil" in the child table which is EMPLOYEE in our case. The modified table EMPLOYEE is shown in Fig. 4.72.

By carefully analyzing the Figs. 4.71 and 4.72, we can observe that the record "civil" in the child table (employee) being deleted.

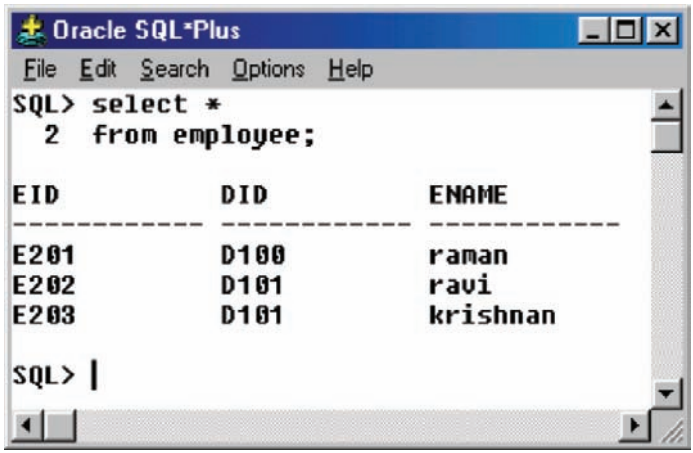


Fig. 4.70. EMPLOYEE table (child table) before deletion of record in parent table

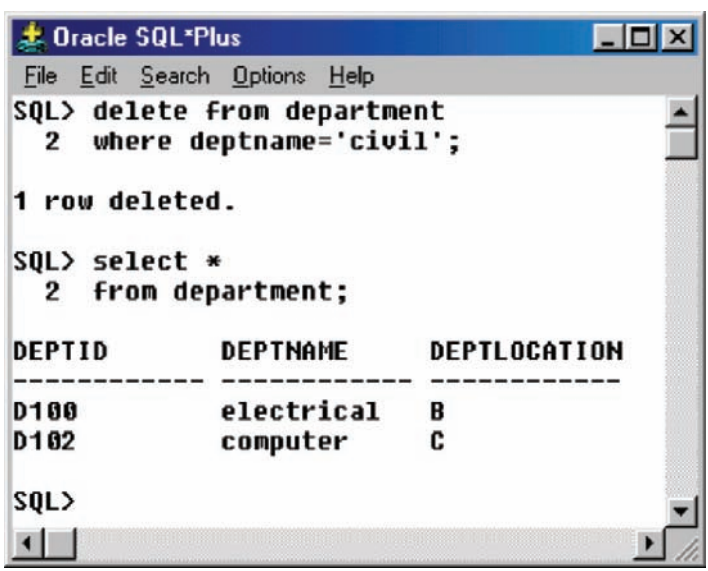


Fig. 4.71. DEPARTMENT table without "civil" department

If ON DELETE CASCADE clause is included in the child table means whatever record deleted in the parent table will be deleted in the child table.

4.12.7 ON DELETE SET NULL

If ON DELETE SET NULL clause is include in the child table means, whenever a row in the parent table is deleted, then the corresponding referenced value in the child table will be set null.

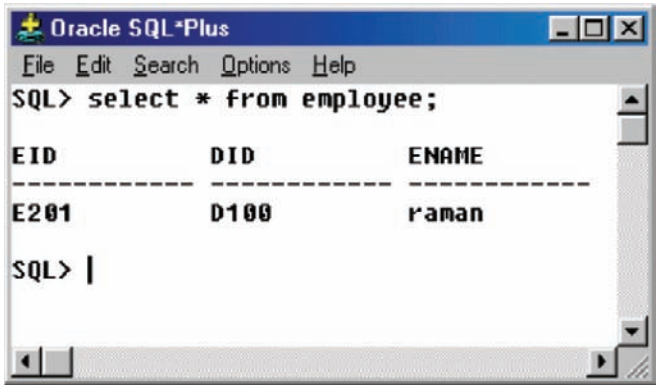


Fig. 4.72. Modified EMPLOYEE table

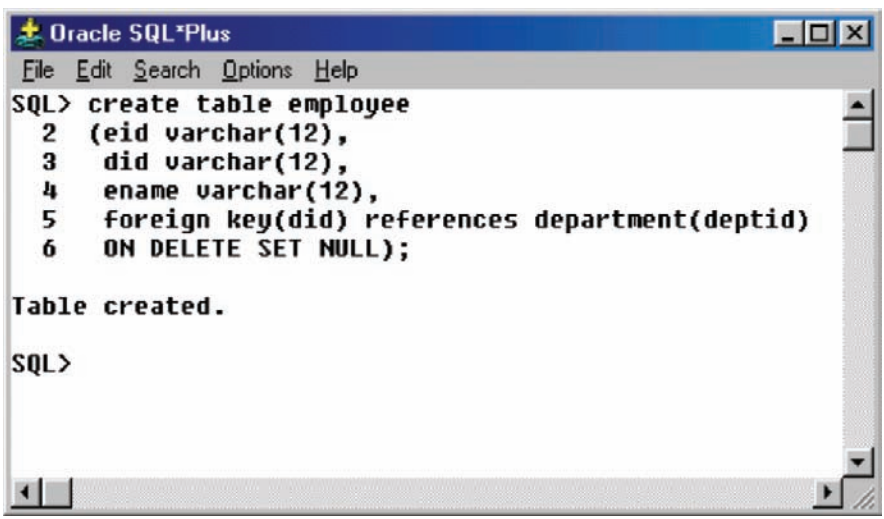


Fig. 4.73. Modified employee table definition

Example

Let us consider the parent table as DEPARTMENT and the child table as EMPLOYEE as before. The child table is created with ON DELETE SET NULL as shown in Fig. 4.73.

The EMPLOYEE table before modification is shown below.

EID	DID	Ename
E201	D100	Raman
E202	D101	Ravi
E203	D101	Krishnan

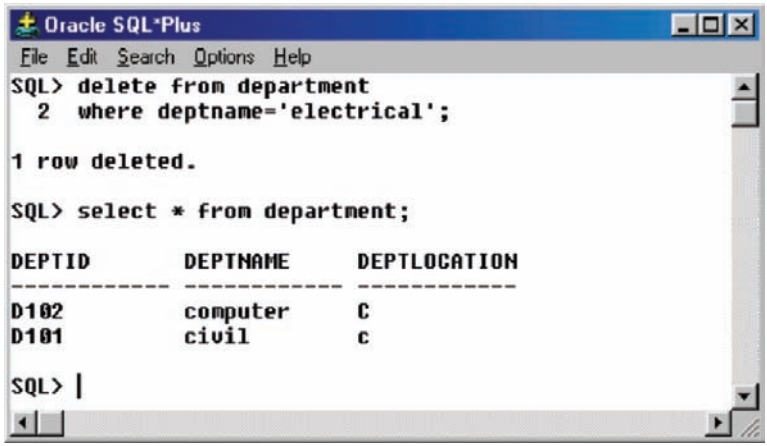


Fig. 4.74. Modified table DEPARTMENT

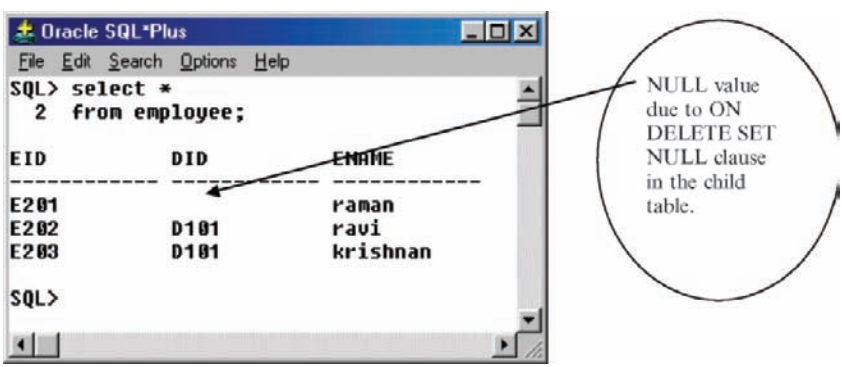


Fig. 4.75. Modified child table (EMPLOYEE)

Now modify the table DEPARTMENT by deleting the "electrical" department record. The SQL command to delete the record "electrical" and the corresponding output are shown in Fig. 4.74.

The impact of deleting the record "electrical" in parent table DEPARTMENT on the child table EMPLOYEE is shown in Fig. 4.75.

From Fig. 4.75, we can observe that a NULL value is there corresponding to the ID of the "electrical" department. This is due to inclusion of the clause ON DELETE NULL in the child table (EMPLOYEE).

4.13 Join Operation

Join operation is used to retrieve data from more than one table. Before proceeding to JOIN operation let us discuss first the Cartesian product. Cartesian product with suitable selection and projection operation forms different types of join.

Cartesian Product

If we have two tables A and B, then Cartesian product combines all rows in the table A with all rows in the table B. If n1 is the number of rows in the table A and n2 is the number of rows in the table B. Then the Cartesian product between A and B will have n1 × n2 rows.

Example

In order to understand Cartesian product, let us consider two relations **doctor** and **nurse**. The relation **doctor** has the attribute ID which refers to identity of the doctor, **name** and **department**. Similarly, the relation **nurse** has three attributes **NID**, which refers to nurse identity, **name** and **department**. The doctor relation is shown in Fig. 4.76.

Similarly the nurse relation is shown in Fig. 4.77.

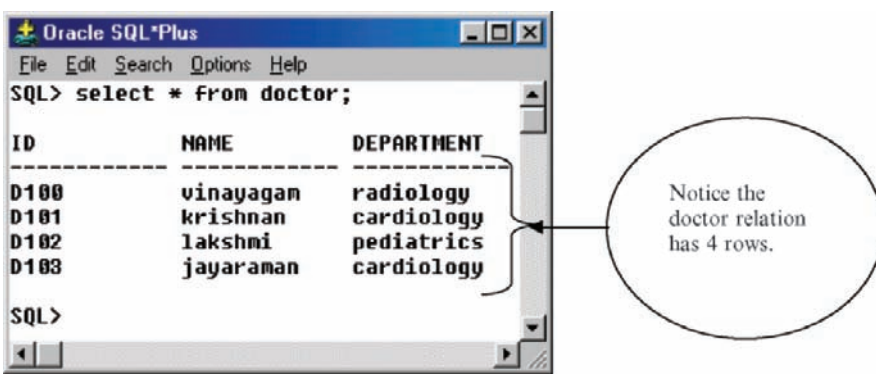


Fig. 4.76. DOCTOR relation

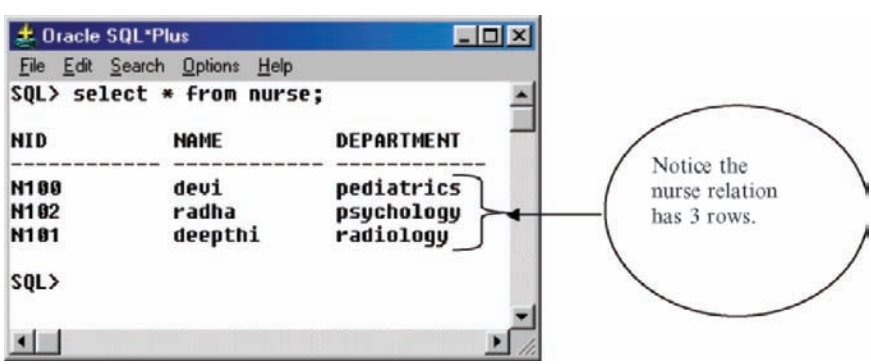


Fig. 4.77. NURSE relation

From Figs. 4.76 and 4.77 we can observe that the number of rows in doctor and nurse relation is 4. Now let us try to find the Cartesian product between the two relations doctor and nurse. The Cartesian product should return $4 \times 3 = 12$ rows. The SQL command to perform Cartesian product between the two relations doctor and nurse and the corresponding output are shown in Fig. 4.78. From this figure, it is evident that the Cartesian product between two relations has 12 tuples (rows).

4.13.1 Equijoin

In equijoin, the join condition is based on equality between values in the common columns. Moreover the common columns appear redundantly in the result. Equijoins are also called as simple joins or inner joins. The equijoin between the two relations doctor and nurse (The relations doctor and nurse are shown in Figs. 4.76 and 4.77, respectively) is shown in Fig. 4.79.

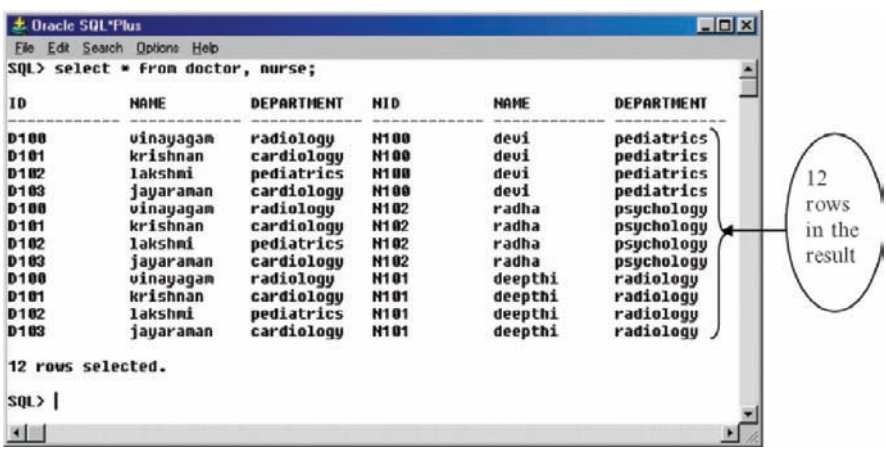


Fig. 4.78. Cartesian product between the relations doctor and nurse

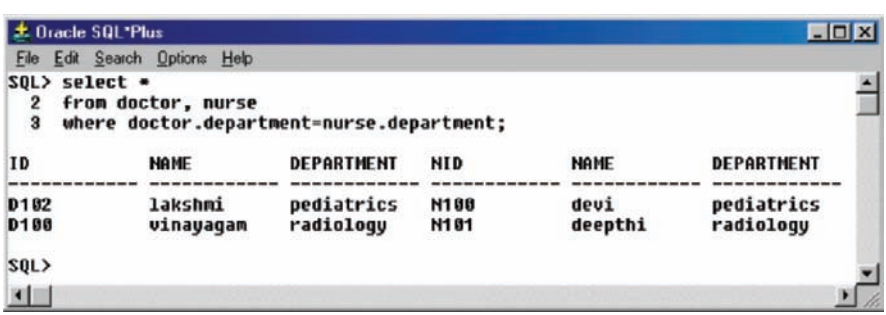


Fig. 4.79. Equijoin between doctor and nurse relation

From Fig. 4.79, it is evident that the join condition is equality condition on the attribute **department**. We can also observe that the common columns appear redundantly in the result.

4.14 Set Operations

The UNION, INTERSECTION, and the MINUS (Difference) operations are considered as SET operations. Out of these three set operations, UNION, INTERSECTION operations are commutative, whereas MINUS (Difference) operation is not commutative. All the three operations are binary operations. The relations that we are going to consider for UNION, DIFFERENCE, and MINUS operations are IBM_DESKTOP and DELL_DESKTOP as shown in Figs. 4.80 and 4.81, respectively.

4.14.1 UNION Operation

If we have two relations R and S then the set UNION operation contains tuples that either occurs in R or S or both.

Case 1: UNION command.

The union of two relations IBM_DESKTOP, DELL_DESKTOP is given in Fig. 4.80. From Fig. 4.81, it is clear that the UNION command eliminates duplicate values.

Case 2: UNION ALL command.

The UNION command removes duplicate values. In order to get the duplicate values, we can use UNION ALL command. The use of UNION ALL command and the corresponding results are shown in Fig. 4.83.

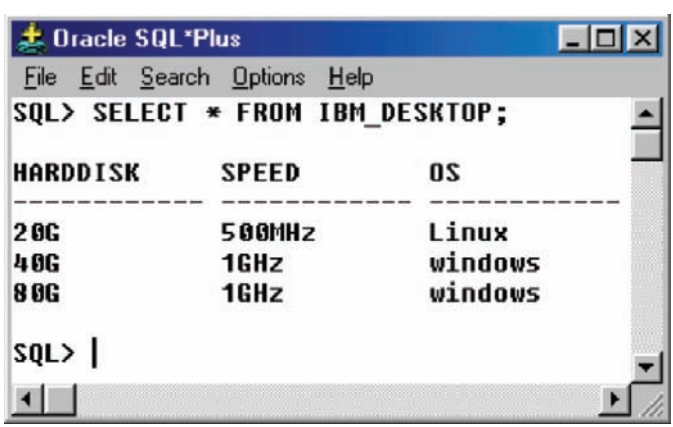


Fig. 4.80. IBM_DESKTOP

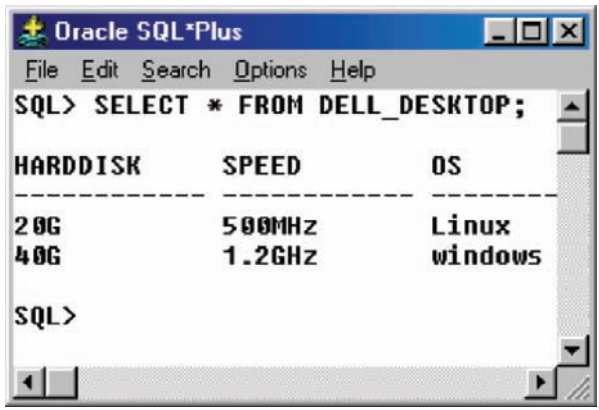


Fig. 4.81. DELL_DESKTOP

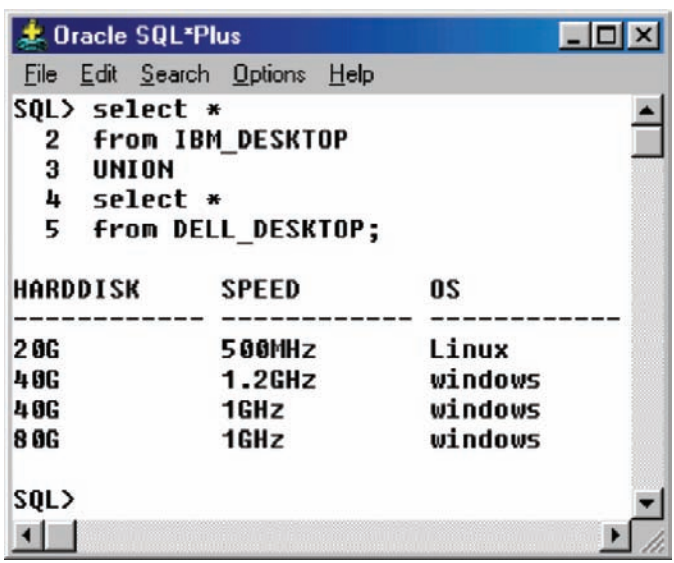


Fig. 4.82. UNION command

By carefully looking into the Figs. 4.82 and 4.83, the number of tuples in the Fig. 4.82 is four; whereas the number of tuples in Fig. 4.83 is five. The difference in two results is due to the fact that UNION command rejects duplicate values, whereas UNION ALL command includes duplicate values.

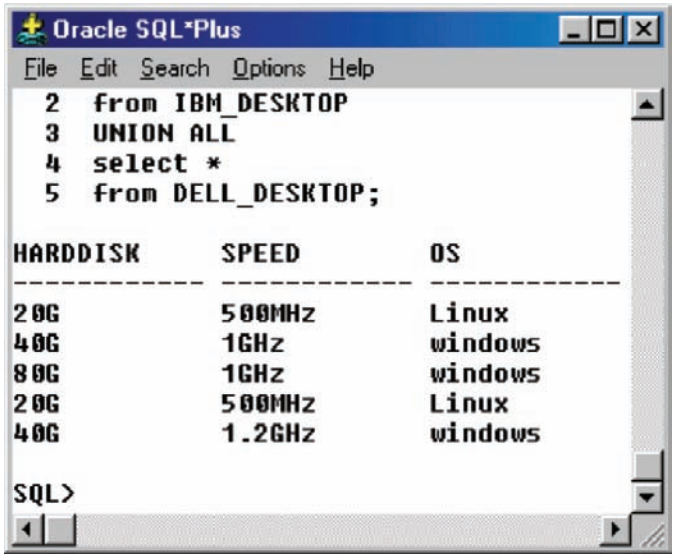


Fig. 4.83. UNION ALL command

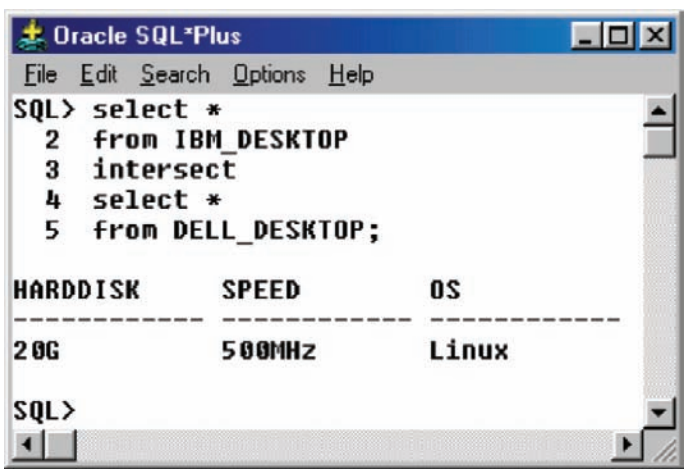


Fig. 4.84. INTERSECTION operation

4.14.2 INTERSECTION Operation

The intersection operation returns the tuples that are common to the two relations. The intersection of the two relations IBM_DESKTOP and DELL_DESKTOP is shown in Fig. 4.84.

4.14.3 MINUS Operation

If R and S are two union compatible relations then R–S returns the tuples that are present in R but not in S. S–R returns the tuples that are present in S but not in R. It is to be noted that MINUS operation is not commutative. That is R–S # S–R.

Case 1: IBM_DESKTOP–DELL_DESKTOP.

Let us first determine IBM_DESKTOP–DELL_DESKTOP. The SQL command and the corresponding output are shown in Fig. 4.85.

From Fig. 4.85, we can observe that the result contains the tuples that are present in IBM_DESKTOP and not in DELL_DESKTOP.

Case 2: DELL_DESKTOP–IBM_DESKTOP.

Let us try to compute DELL_DESKTOP–IBM_DESKTOP. The SQL command and the corresponding output are shown in Fig. 4.86. From Fig. 4.86, it is clear that the result contains tuple that are present in DELL_DESKTOP but not in IBM_DESKTOP.

Note From Figs. 4.85 and 4.86 it is clear that MINUS operation is not commutative.

4.15 View

View is a pseudotable or virtual table. View is called as "pseudotable" because view does not actually store data. View just displays the data. The data are derived from one or more base tables. View table can be used like any other table for querying. View can be considered as a window to the database. The view can also be considered as customized presentation of data from one or more tables. It is to be noted that all views are not updatable.

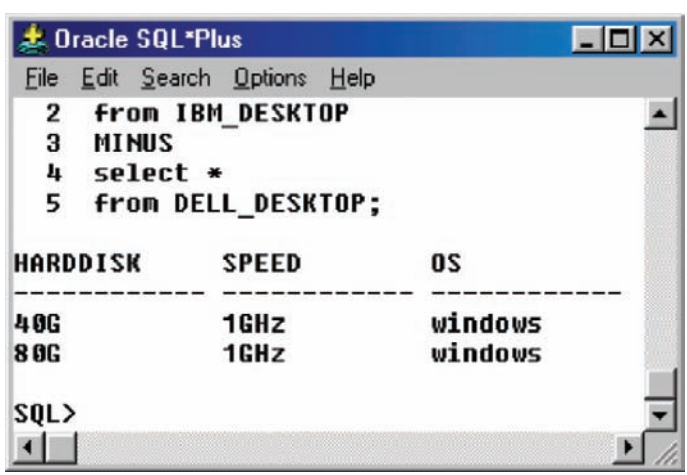


Fig. 4.85. IBM_DESKTOP–DELL_DESKTOP

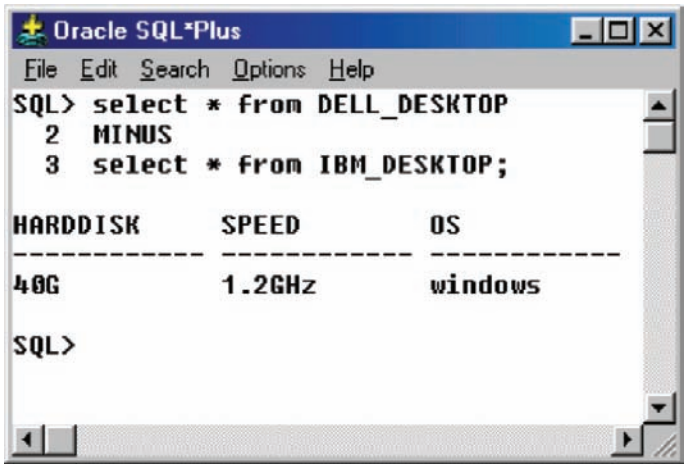


Fig. 4.86. DELL_DESKTOP–IBM_DESKTOP

The Syntax of VIEW is given as

CREATE VIEW view name
AS SELECT attribute list
FROM table(s)
WHERE condition(s)

Case 1: VIEW from a Single Table.

Consider the base table RECORD which gives the record of the student such as his/her Roll Number, Age, GPA (Grade Point Average), and institution which refers to the institution where he/she has got the degree (Fig. 4.87). The base table RECORD is shown below.

RECORD

S.I. No	Name	Age	GPA	Institution
1	Anbalagan	22	9.2	PSG
2	Balu	22	9.4	PSG
3	Dinesh	22	8.4	CIT
4	Karthik	21	8.5	REC
5	Kumar	22	8.7	MIT
6	Kishore	22	8.8	MIT
7	Rajan	22	9.1	PSG
8	Lavanya	21	9.1	CIT

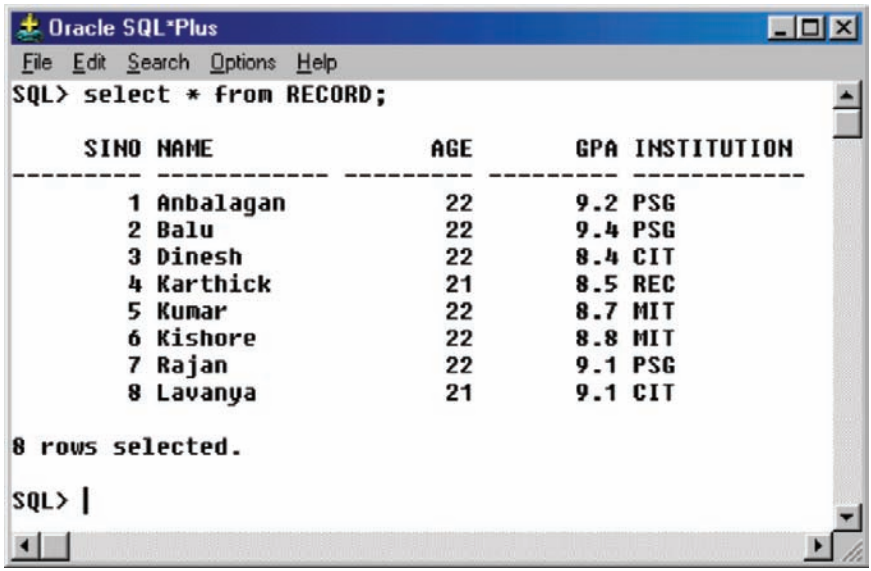


Fig. 4.87. Base table RECORD

Now we want to create a view by name PLACED, which gives the list of students placed in a particular organization (say IBM). The attribute associated with the view PLACED are Name, Age, and Institution. The view PLACED is shown below.

PLACED

Name	Age	Institution
Anbalagan	22	PSG
Balu	22	PSG
Rajan	22	PSG
Lavanya	21	CIT

From the table PLACED, it is obvious that only those students with GPA greater than nine are placed. The SQL command to create the view PLACED from the base table RECORD and the output are shown in Fig. 4.88. From Fig. 4.88, it is clear that the view PLACED has only three columns Name, Age, and Institution.

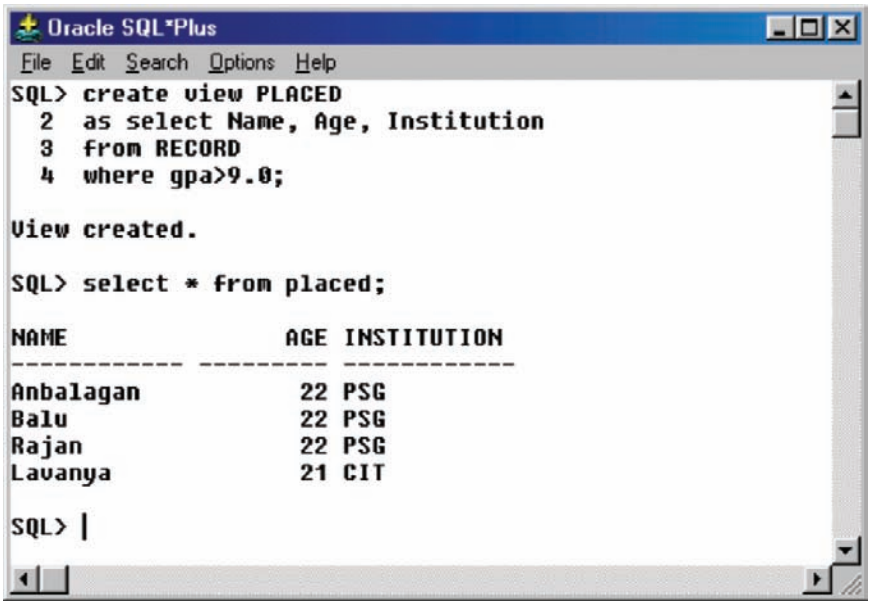


Fig. 4.88. View PLACED from base table RECORD

4.15.1 Nonupdatable View

Case 1: A view created using DISTINCT clause is usually nonupdatable.

Example

To prove that the view created using DISTINCT clause is nonupdatable, consider the base relation SAMPLE, which has two attributes Name and Age. Let us create a view UPSAMPLE from the base relation SAMPLE using DISTINCT clause. The base relation SAMPLE and the view UPSAMPLE is shown below:

SAMPLE

Roll No	Name	Age
1	Anand	20
2	Anandi	19
3	Banu	20
4	Chandran	20
5	Ravi	21
6	Chandran	21
7	Anand	20

UPSAMPLE

Name	Age
Anand	20
Anandi	19
Banu	20
Chandran	20
Chandran	21
Ravi	21

The SQL command to create the view UPSAMPLE from the base relation SAMPLE using DISTINCT clause is shown in Fig. 4.89.

The created view UPSAMPLE is shown in Fig. 4.90. Now let us try to update the view UPSAMPLE, the SQL command to update the view and the corresponding output are shown in Fig. 4.91.

From Fig. 4.91, it is clear that the view defined by DISTINCT clause is nonupdatable.

Case 2: It is not possible to update the view if it contains group function or if it contains group by clause.

Example

In order to prove that the view is nonupdatable if it contains group function or group by clause, let us consider the base relation BOOKS. The attributes of the relation BOOKS are author, title, price. The content of the base relation BOOKS is shown in Fig. 4.92. Now let us define the view COUNTS, which gives the number of books written by the author. The SQL syntax to create the view is shown in Fig. 4.93. The contents of the view COUNTS are shown in Fig. 4.94.

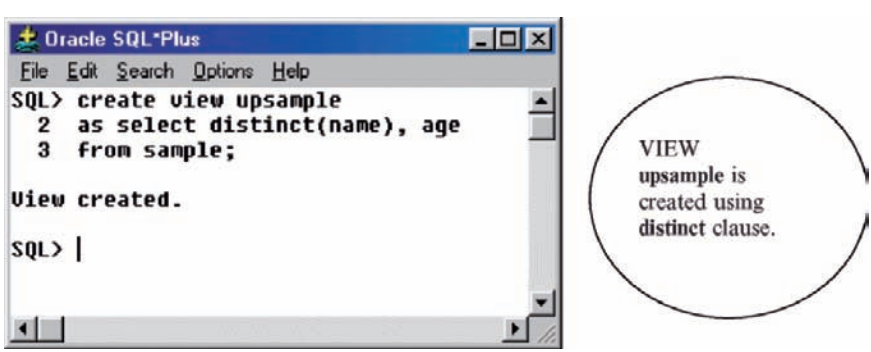


Fig. 4.89. VIEW creation using DISTINCT

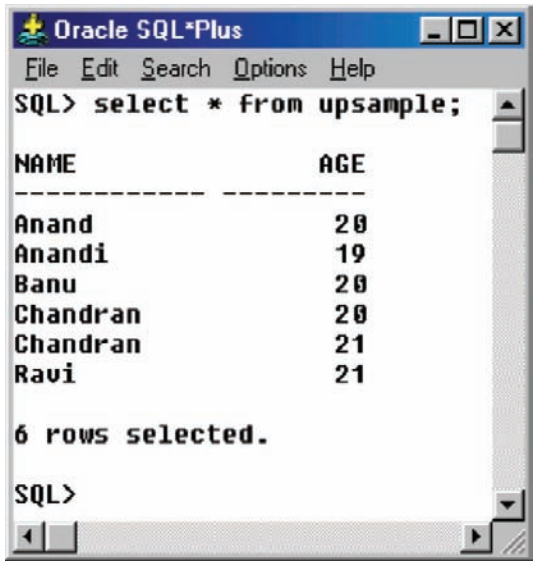


Fig. 4.90. Contents of the view UPSAMPLE

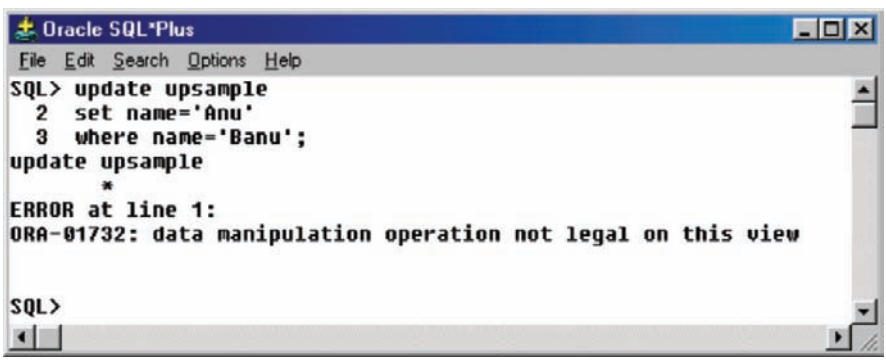


Fig. 4.91. Result of update operation in nonupdatable view

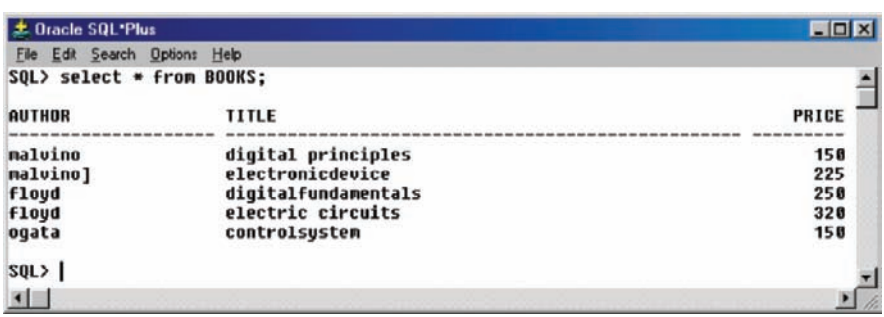


Fig. 4.92. The base relation BOOKS

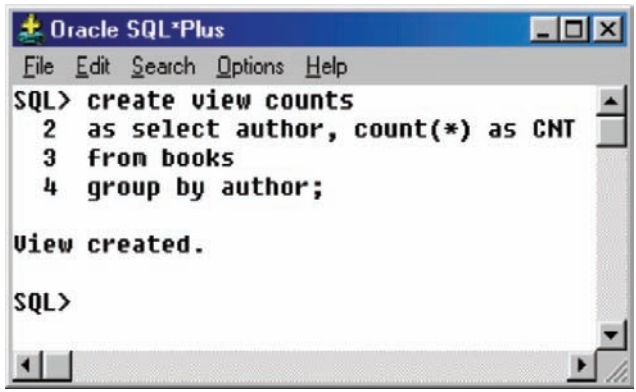


Fig. 4.93. View COUNTS from BOOKS

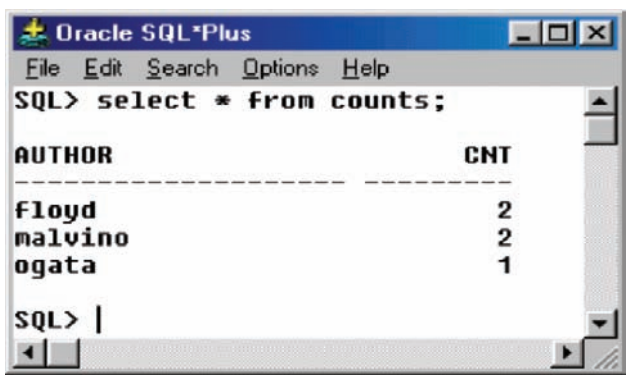


Fig. 4.94. Contents of COUNTS

Try1

First let us try to delete a row from the view COUNTS. The SQL command to delete a row from the view COUNTS and the corresponding output are shown in Fig. 4.95. From Fig. 4.95, it is clear that it is not possible to delete a row from the view if it is created using group function or group by clause.

Try2

Now let us try to update the view COUNTS by modifying the name malvino to malvinoleech. The SQL command to modify the name in the view COUNTS and the corresponding output are shown in Fig. 4.96.

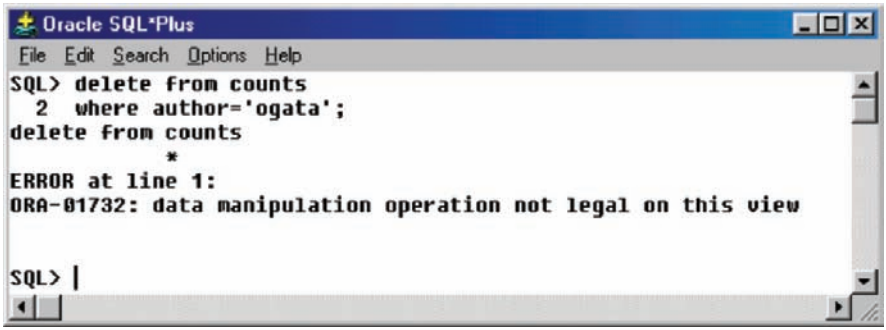
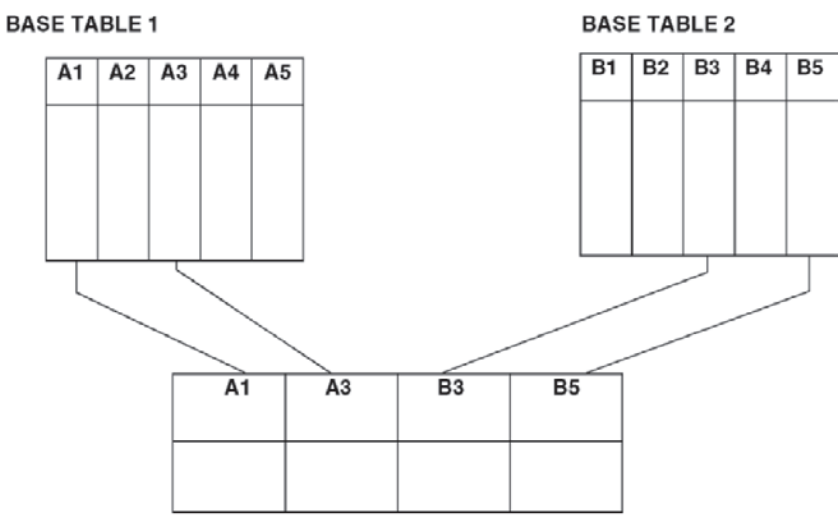


Fig. 4.95. Deletion of row in the view COUNTS

From Fig. 4.96, it is clear it is not possible to update the view if it contains group function or group by clause.

4.15.2 Views from Multiple Tables

Views from multiple tables are termed as complex views, whereas views from single table are termed as simple views. View from multiple tables is illustrated as follows:



View from two base tables

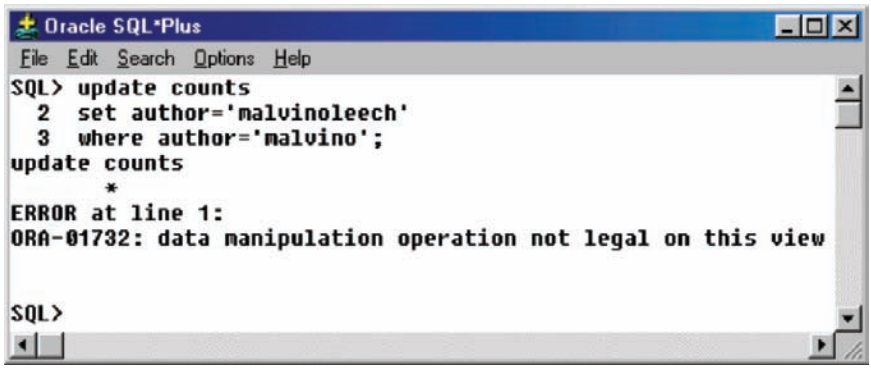


Fig. 4.96. View updation

Example

Let us try to create view from two tables. Here one table is COURSE and the other table is STAFF. The attribute of the COURSE table are courseID, course name, LectID (which refers to Lecturer Identity number). The attributes of STAFF table are name, LectID, and position.

STAFF

Name	LectID	Position
Rajan	E121	lecturer
Sridevi	E122	lecturer
Jayaraman	E123	professor
Navaneethan	E124	professor

COURSE

CourseID	Course Name	LectID
C200	RDBMS	E121
C201	GraphTheory	E122
C202	DSP	E123
C203	OS(Operating System)	E124

The view COURSE_STAFF is created by selecting course name from course and Name from staff as shown in Fig. 4.97.

The SQL command to create the view COURSE_STAFF from COURSE and STAFF is shown in Fig. 4.98. From Fig. 4.98 it is evident that the view COURSE_STAFF is created from two tables COURSE and STAFF.

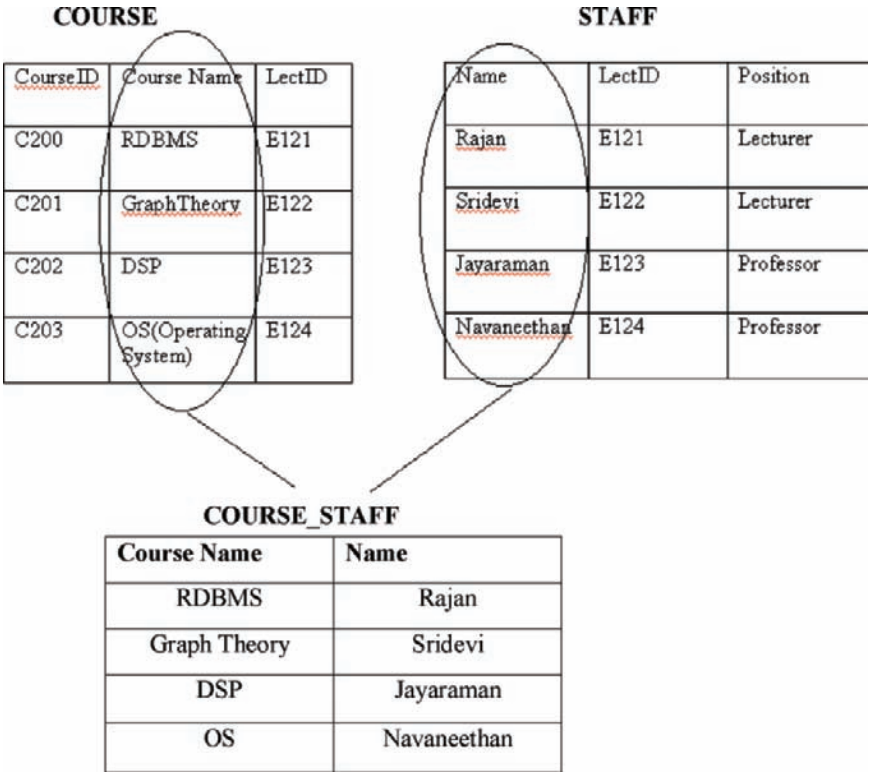


Fig. 4.97. View COURSE_STAFF from COURSE and STAFF

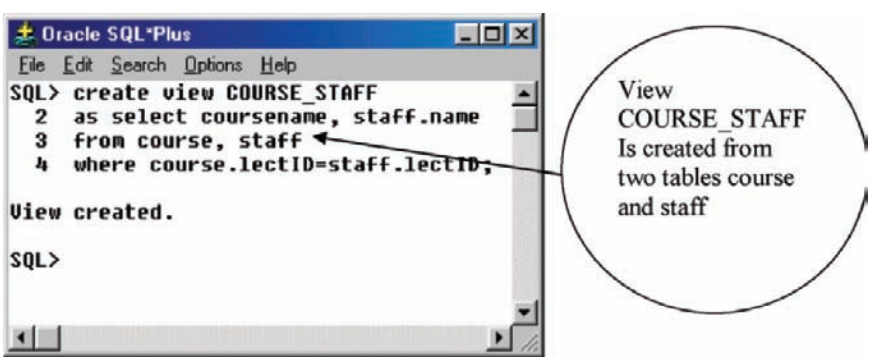


Fig. 4.98. VIEW from two tables

Let us try to see the contents of the view COURSE_STAFF by using SELECT command as shown in Fig. 4.99.

Note The view COURSE_STAFF is created from two tables, hence it can be considered as complex views. Complex views are in general not updatable. Let

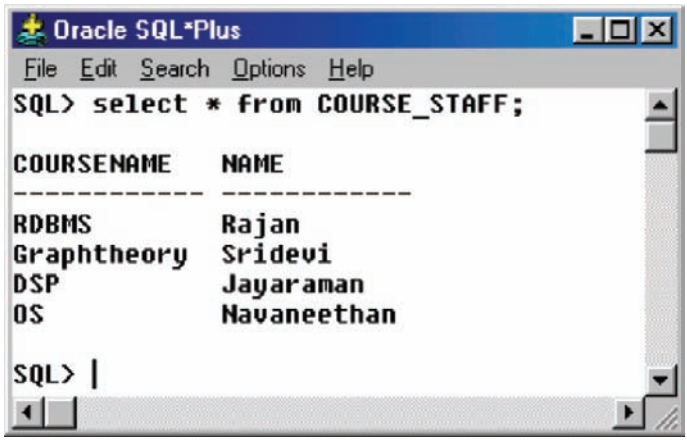


Fig. 4.99. Contents of the view COURSE_STAFF

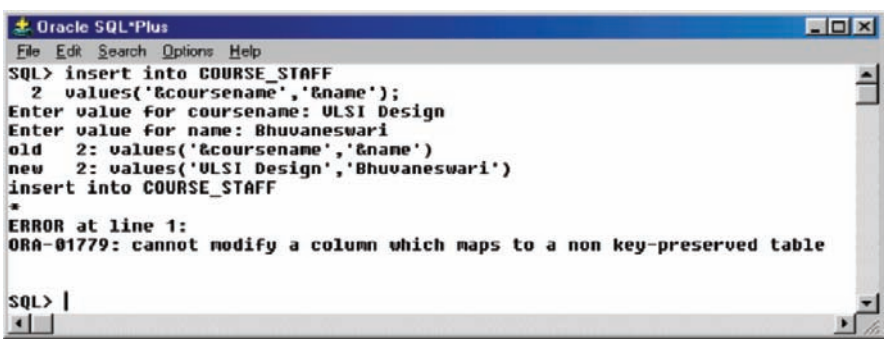


Fig. 4.100. View nonupdatable

us check whether the view COURSE_STAFF is updatable or not by trying to insert tuples into the view COURSE_STAFF as shown in Fig. 4.100.

From Fig. 4.100, it is clear that it is not possible to insert tuples into complex view (COURSE_STAFF). Now let us try to update the view COURSE_STAFF by modifying the name Rajan as Siva as shown in Fig. 4.101.

From Fig. 4.101, it is clear that the complex view (view created from more than one table) is usually nonupdatable.

4.15.3 View From View

It is possible to create view from another view. This is diagrammatically shown in Fig. 4.102. From Fig. 4.102, it is clear that the view2 is created from view1 and not from the base table. View1, View2 can be queried similar to the base table.

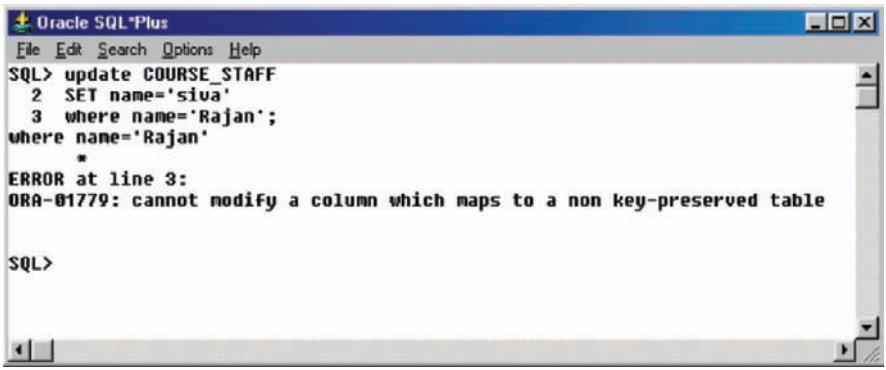


Fig. 4.101. Nonupdatable view COURSE_STAFF

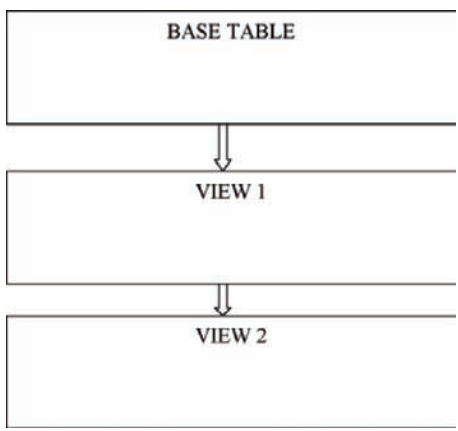


Fig. 4.102. View from a view

Example

Let us consider base table STAFF as shown in Fig. 4.103, the view ITSTAFF is created from the base table STAFF (Fig. 4.104). Then the view YOUNGIT-STAFF is created from the view ITSTAFF (Fig. 4.105). The view ITSTAFF is shown in Fig. 4.106 and the view YOUNGITSTAFF is shown in Fig. 4.107.

Figure 4.104 shows the SQL command to create the view ITSTAFF from the base table STAFF. The view ITSTAFF contains only the details of the staff who belong to the IT department as shown in Fig. 4.104.

The contents of the view YOUNGITSTAFF is shown in Fig. 4.107. We can observe that the view YOUNGITSTAFF contains only the details of IT staff whose age is less than 30.

Doubt 1: Whether the view YOUNGSTAFF which is created from another view ITSTAFF can be queried like the base table?

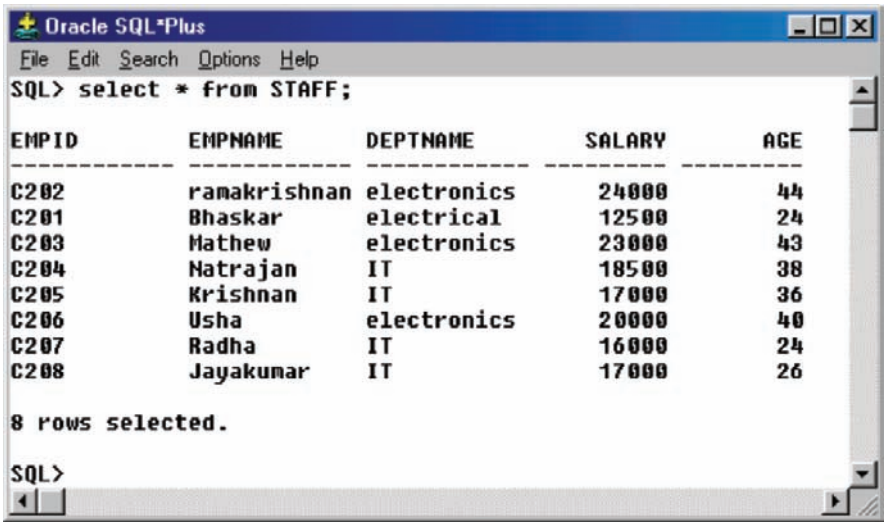


Fig. 4.103. Base table STAFF

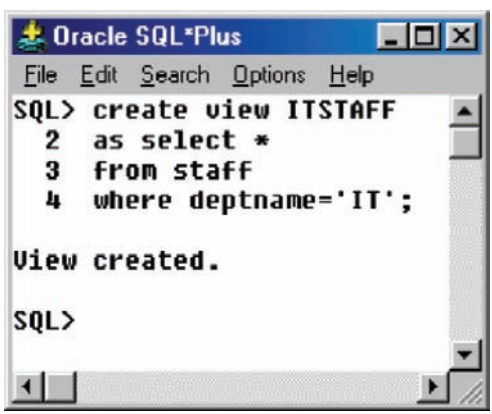


Fig. 4.104. View ITSTAFF from base table STAFF

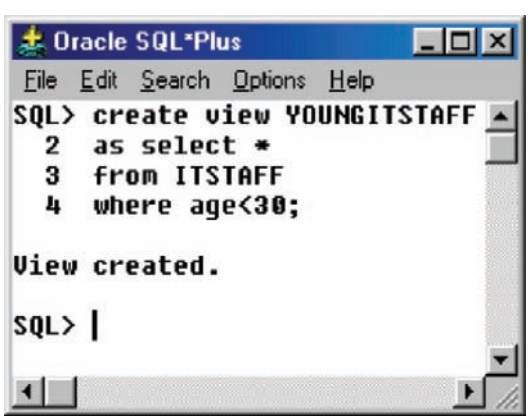


Fig. 4.105. View YOUNGITSTAFF from the view ITSTAFF

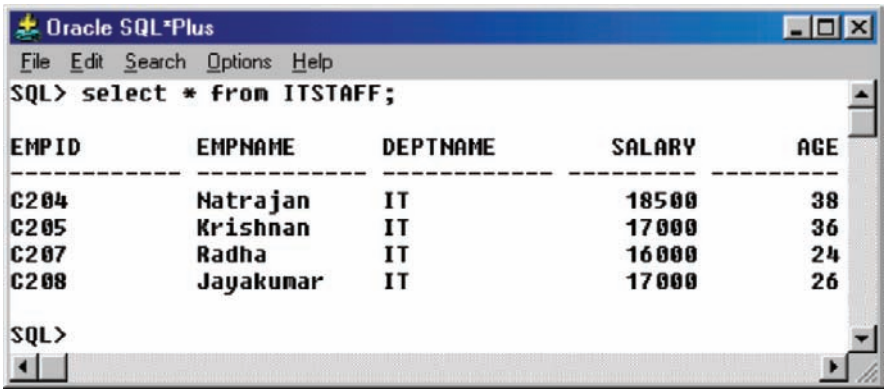


Fig. 4.106. Contents of the view ITSTAFF

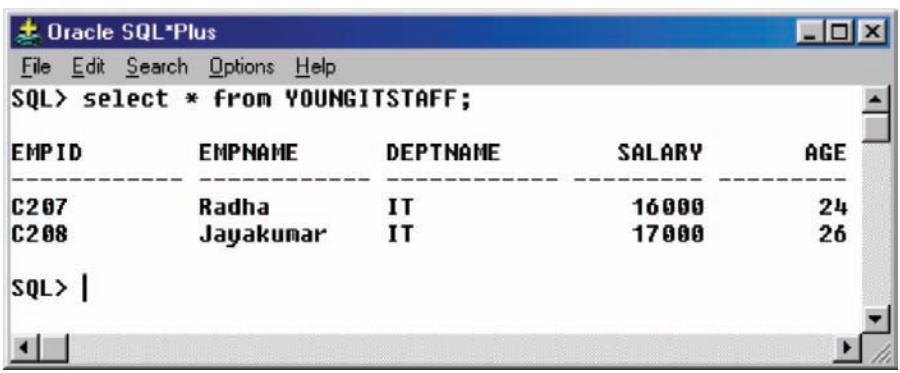


Fig. 4.107. Contents of the view YOUNGITSTAFF

Answer : Yes. The view YOUNGITSTAFF, which is created from another view ITSTAFF can be queried like the base table.

Example

Let us consider the query: What is the pay offered to the YOUNGITSTAFF Radha? The SQL command to answer the query is shown in Fig. 4.108.

From Fig. 4.108, it is clear that the view YOUNGITSTAFF which is created from another view ITSTAFF can be queried similar to the base table STAFF

Doubt 2: If it is possible to make any change in the view ITSTAFF which was created from the base table STAFF, will it reflect in the base table STAFF.

Answer : Yes, if it is possible to make any change in the view which was derived from the base table then the change will be reflected in the base table.

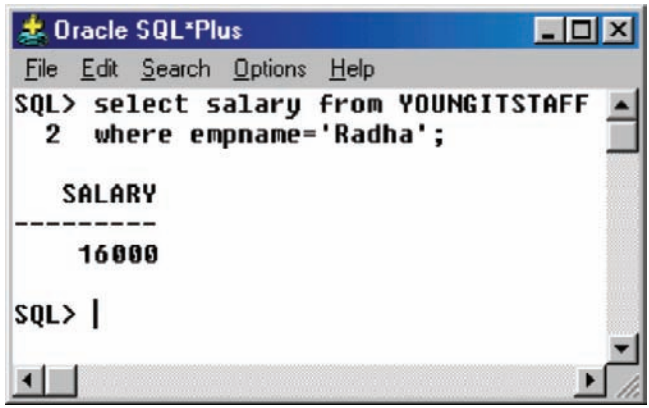


Fig. 4.108. Query on YOUNGITSTAFF

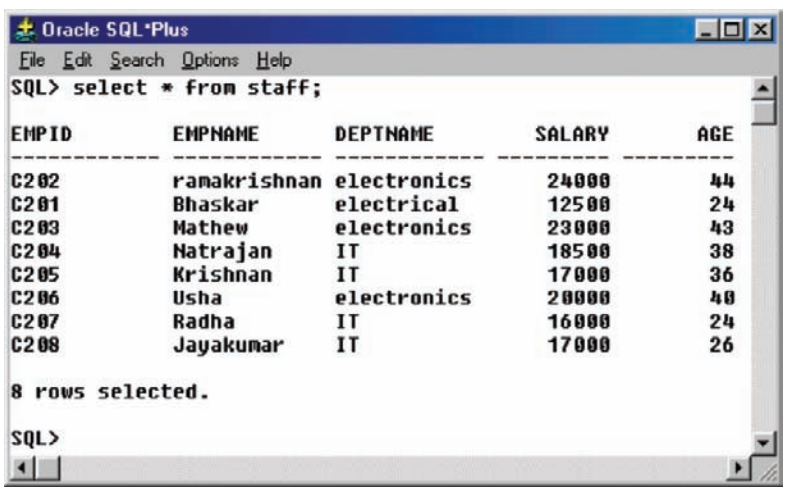


Fig. 4.109. Contents of the base table before any updation in the view ITSTAFF

Example

Let us modify the view ITSTAFF by including one row. Before modification the contents of the base table STAFF is shown in Fig. 4.109.

From Fig. 4.109, we can observe that there are eight rows in the base table STAFF.

Now let us update the view ITSTAFF by including one row in the view ITSTAFF. The SQL command to insert the row in the view ITSTAFF is shown in Fig. 4.110.

Contents of the ITSTAFF after inserting a row are shown in Fig. 4.111. From Fig. 4.111, we can observe that the new row being included in the IT-STAFF view.

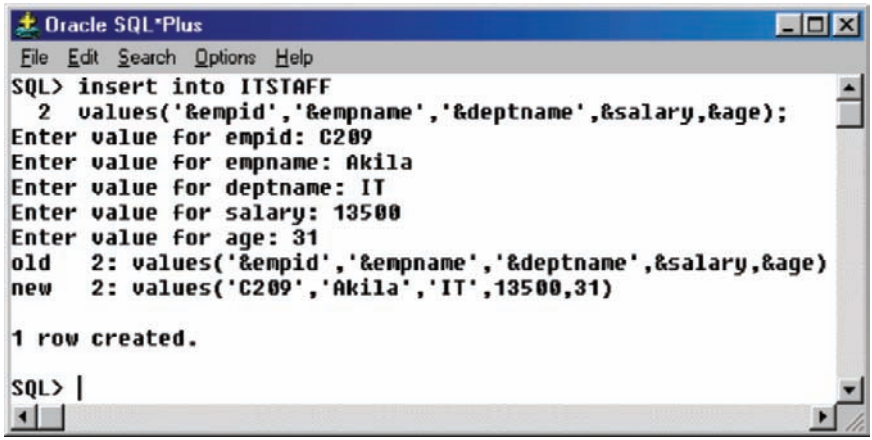


Fig. 4.110. Insertion of a row into the view ITSTAFF

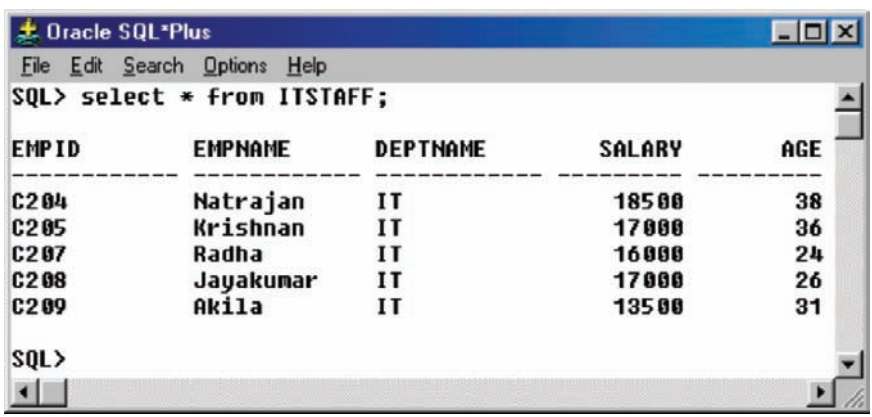


Fig. 4.111. Content of the view ITSTAFF after inserting a row

Now let us see the content of the base table STAFF to find whether the change made in the view ITSTAFF is reflected in the base table STAFF. The content of the base table STAFF is shown in Fig. 4.112.

Comparing Fig. 4.109 with Fig. 4.112 it is clear that one new row being included in the base table STAFF. This means that the change in the view will be reflected in the base table.

Doubt 3: If the view ITSTAFF is dropped, then is it possible to get the content of the view YOUNGITSTAFF which is derived from ITSTAFF?

Answer : For the view YOUNGITSTAFF, the contents are from another view ITSTAFF. Hence if ITSTAFF is dropped means it is not possible to get the contents of the view YOUNGITSTAFF.

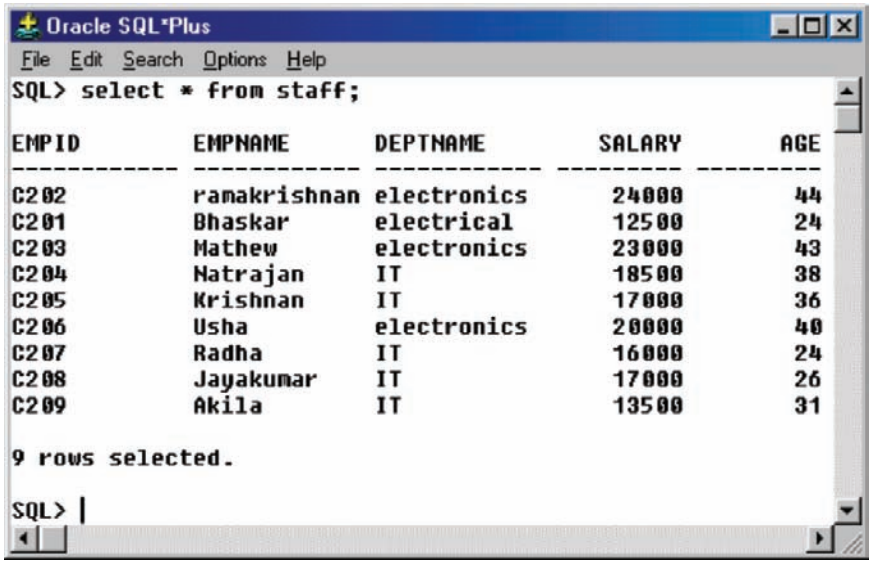


Fig. 4.112. Content of the base table STAFF after modification in the view ITSTAFF

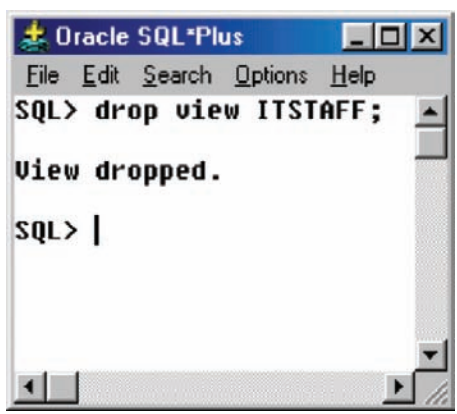


Fig. 4.113. Dropping the view

Example

Let us drop the view ITSTAFF as shown in Fig. 4.113. Figure 4.114 ensures that the view ITSTAFF is successfully dropped.

Now let us try to see the content of the view YOUNGITSTAFF which is derived from the view ITSTAFF. The SQL command to retrieve the contents of the view YOUNGITSTAFF is shown in Fig. 4.115.

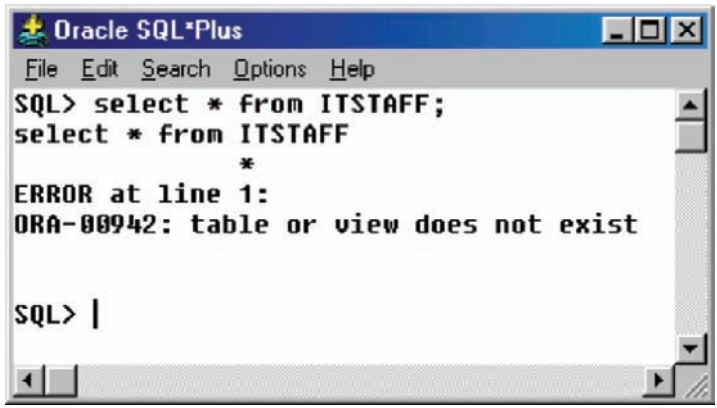


Fig. 4.114. Contents after dropping the view

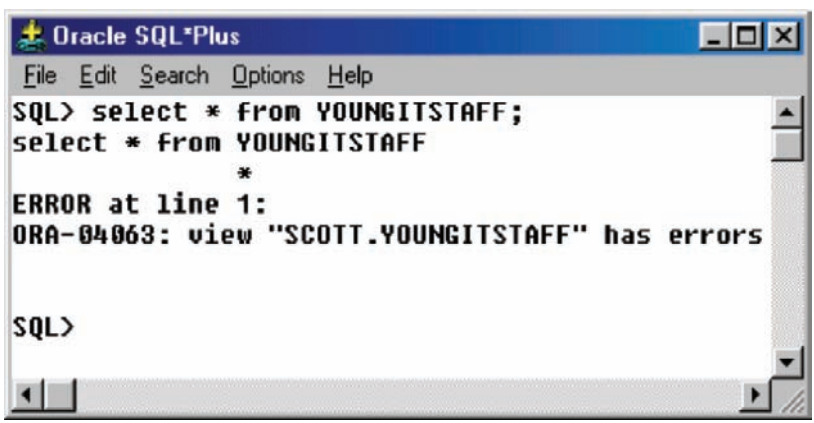


Fig. 4.115. Contents of YOUNGITSTAFF after dropping the view ITSTAFF

From Fig. 4.115, it is clear that once the view ITSTAFF is dropped then it is not possible to retrieve the contents of the view YOUNGITSTAFF which is derived from the view ITSTAFF.

4.15.4 VIEW with CHECK Constraint

It is possible to create view with CHECK constraint. If we create a view with CHECK constraint, then it is not possible to update the view if the CHECK constraint is violated.

Example of View with CHECK Constraint

Let us consider the base relation CITIZEN which has the attributes name, age, and address. Now let us create the view VOTERS from the base relation

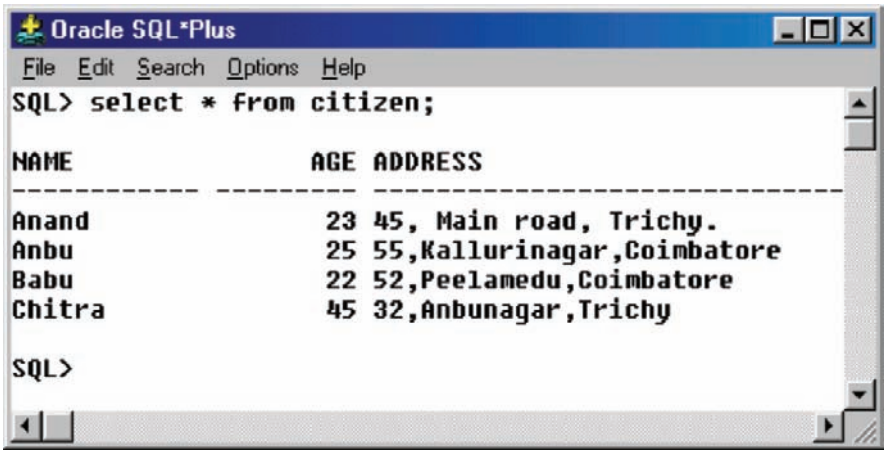


Fig. 4.116. Contents of base table CITIZEN

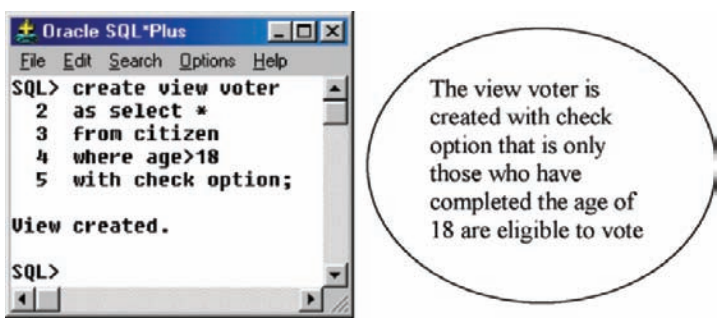


Fig. 4.117. View with check option

CITIZEN. We know that, the citizen of India becomes eligible voter if he/she attains the age of 18. The base relation CITIZEN is shown in Fig. 4.116. The view VOTER from base relation CITIZEN is shown in Fig. 4.117.

Case 1: Low let us try to insert value into the view voter who is eligible to vote, that is the age of the voter is greater than 18. The SQL command and the corresponding output are shown in Fig. 4.118. From Fig. 4.118, it is clear that the value is successfully inserted into the view VOTER.

Case 2: Let us try to insert a row into the view VOTER by violating the check constraint (age of the voter is less than 18). The SQL command and the corresponding output are shown in Fig. 4.119.

4.15.5 Views with Read-only Option

A view can be created with read only option. Such views cannot be modified using INSERT, DELETE, and UPDATE commands.

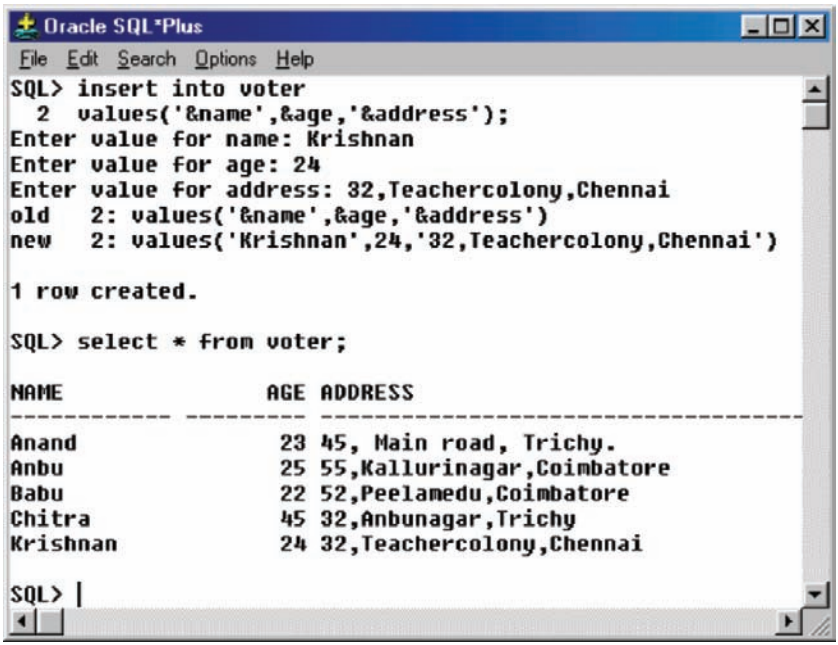


Fig. 4.118. Inserting valuable record into the view VOTER

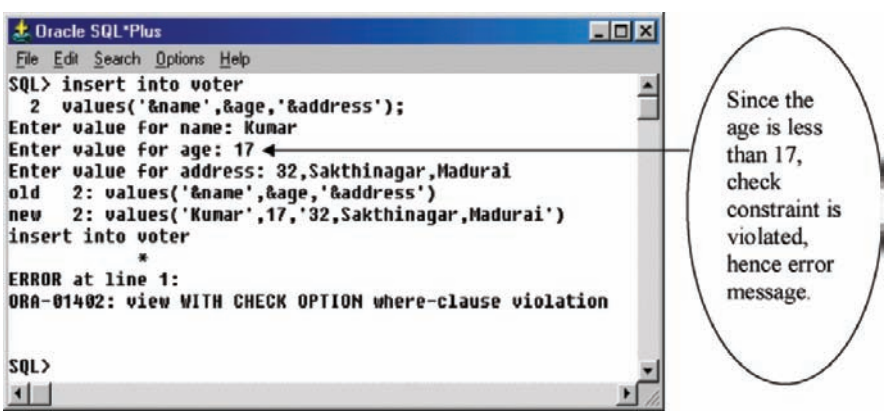


Fig. 4.119. Violation of check constraint

Example

Consider the base table STAFF as shown in Fig. 4.120. Let us create the view electronicsstaff from the base table staff with readonly option as shown in Fig. 4.121.

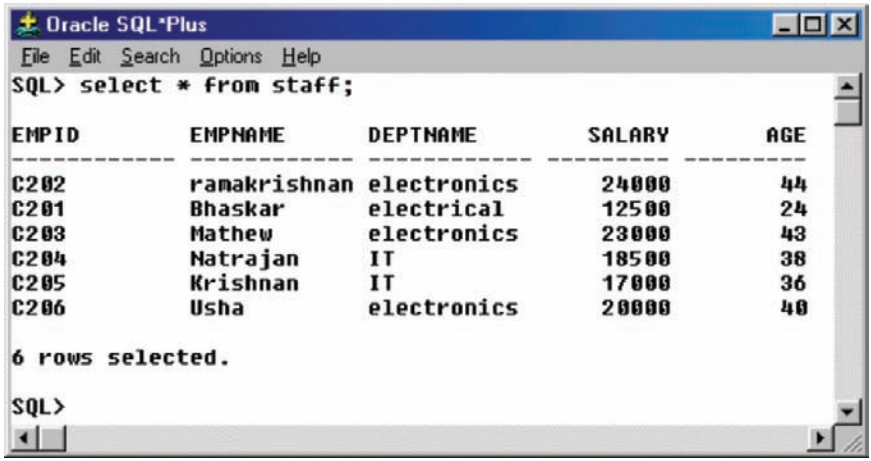


Fig. 4.120. Base table STAFF

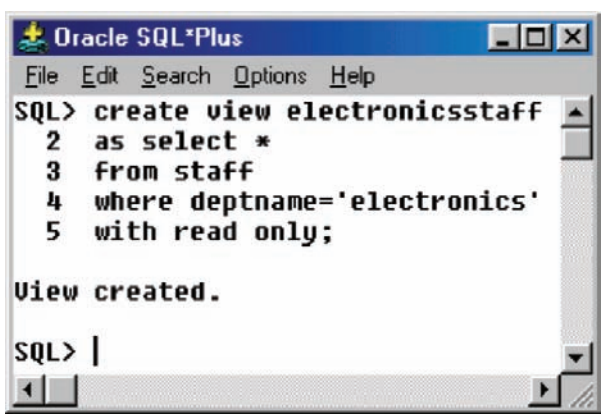


Fig. 4.121. View with read only option

From Fig. 4.121 it is clear that the view electronicsstaff is created with read only option. Now we have to check whether the view electronicsstaff is updatable, that is whether is it possible to INSERT, DELETE and UPDATE values in the view electronicsstaff. The content of the view electronicsstaff is shown in Fig. 4.122.

Case 1: INSERTING Values into the Read-Only View.

Let us try to insert values into the view "electronicsstaff." The SQL command and the corresponding output are shown in Fig. 4.123.

From Fig. 4.123, it is clear that it is not possible to insert values into a read-only view.

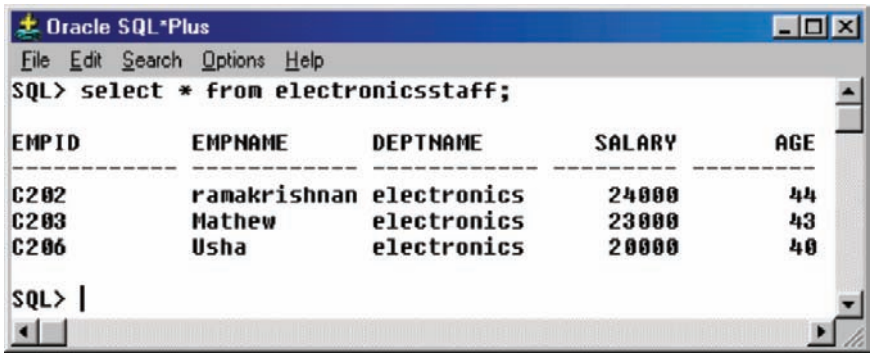


Fig. 4.122. Contents of the read only view electronicsstaff

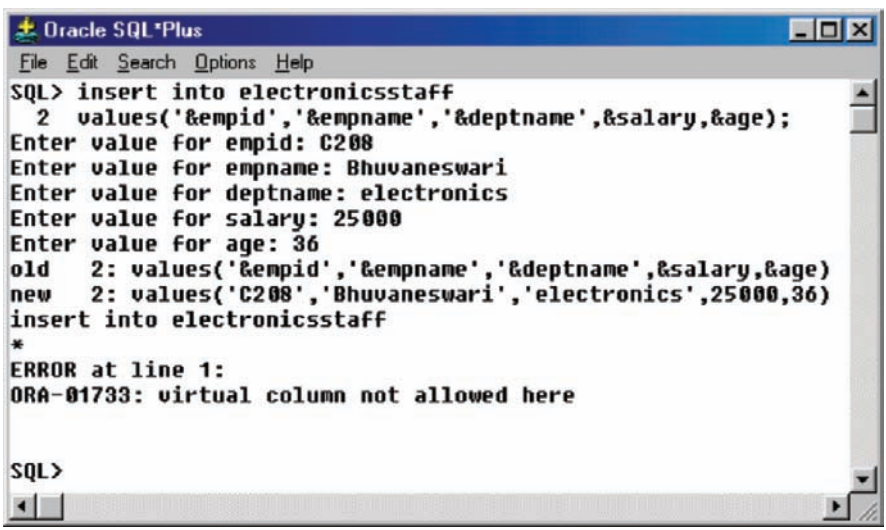


Fig. 4.123. Insertion of values into read-only view

Case 2: Deleting value from a read-only view.

Let us try to delete a value (in our case deleting the record of the electronicsstaff "usha") from the read-only view "electronicsstaff," the SQL command and the corresponding output are shown in Fig. 4.124.

From Fig. 4.124, it is evident that it is not possible to delete value from the read-only view.

Case 3: Updating the record of read-only view.

Let us try to update the record of the read-only view "electronicsstaff" by modifying the age of "usha" to 30. The SQL command to modify the age of the staff "usha" and the corresponding output are shown in Fig. 4.125.

From Fig. 4.125, it is clear that it is not possible to update the view since it is read-only.

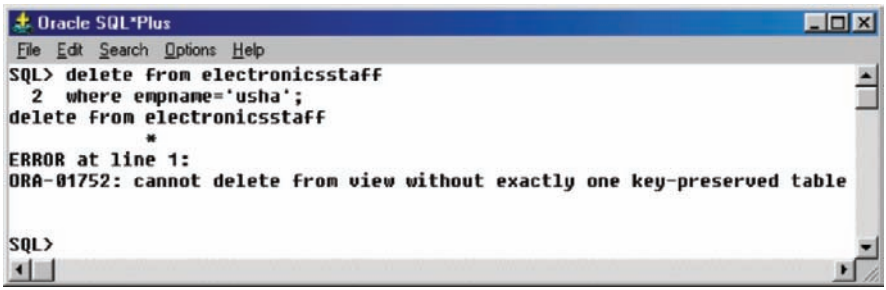


Fig. 4.124. Deleting a tuple from read-only view

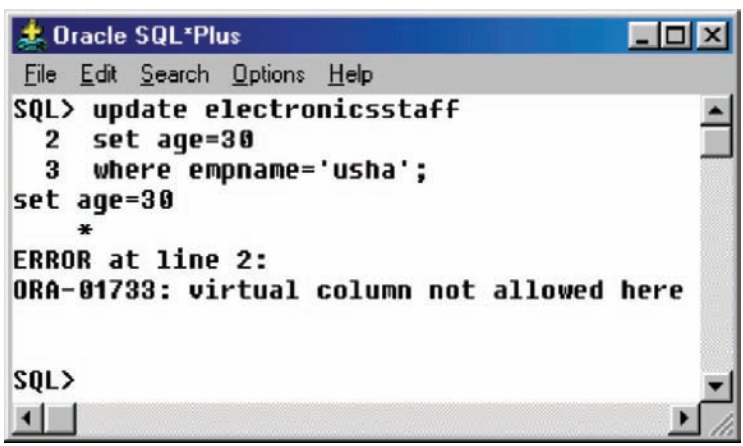


Fig. 4.125. Updating the record in read-only view

4.15.6 Materialized Views

A materialized view is a physical copy of the base table with the results moved to another schema object. Materialized views are also called snapshots, because they are a kind of photograph of the base table.

Advantage of VIEW

The main advantages of view are improved security, less complexity, better convenience, and customization.

1. *Improved security.* We can restrict the user to access on the data that are appropriate for the user. Hence views provide improved security.
2. *Less complexity.* A view can simplify queries, by getting data from several tables into a single table thus transforming multitable queries into a single table queries.

3. *Convenience.* A database may contain much information. All the information will not be useful to the users. The users are provided with only the part of the database that is relevant to them rather than the entire database; hence views provide great convenience to the users.
4. *Customization.* Views provide a method to customize the appearance of the database so that the users need not see full complexity of database. View creates the illusion of a simpler database customized to the needs of a particular category of users.

Drawback of VIEW

1. If the base table is modified by adding one or more columns then the columns added will not be available in the view unless it is recreated.
2. When a view is created from the base table, it is to be noted that all the views are not updatable. Views created from multiple tables are in general not updatable when there is a group function, a GROUP BY clause, or restriction operators.

4.16 Subquery

Subquery is query within a query. A SELECT statement can be nested inside another query to form a subquery. The query which contains the subquery is called outer query.

Scalar subquery

A scalar subquery returns single row, single column result.

Example of Scalar Subquery

Scalar subquery returns single row single column result. To understand scalar subquery, consider two relations STUDENT and COURSE. The attributes of the STUDENT relation are SID, SNAME, AGE, and GPA. The attributes of COURSE relation are CID (Course ID), CNAME (Course ID), SID (Student ID), and INSTRUCTOR (Name of the Instructor). The two relations are shown below.

STUDENT

SID	SNAME	AGE	GPA
E100	Anbu	21	9.6
E101	Aravind	21	9.2
E102	Balu	21	9.4
E103	Chitra	22	8.8
E104	Sowmya	21	9.8

COURSE

CID	CNAME	SID	INSTRUCTOR
C100	RDBMS	E100	Rajan
C101	OS	E102	Sumathi
C102	DSP	E101	Jayaraman
C103	DSP	E104	Jayaraman

Query 1: Find the name of the student who has opted for the course RDBMS?

Solution. From the STUDENT and COURSE table, it is clear that only one student has opted for RDBMS (just for example). We can get the name of the student using scalar subquery. The SQL command and the corresponding output are shown in Fig. 4.126.

Query 2: Find the Names of the Student who have Opted for DSP Course

Solution. From the STUDENT and COURSE table, we can observe that more than one student has opted for DSP course. Here we cannot use scalar subquery because scalar subquery gives single row and single column result. But our result has more than one row. First let us try to get by scalar subquery. The SQL command and the corresponding output are shown in Fig. 4.127.

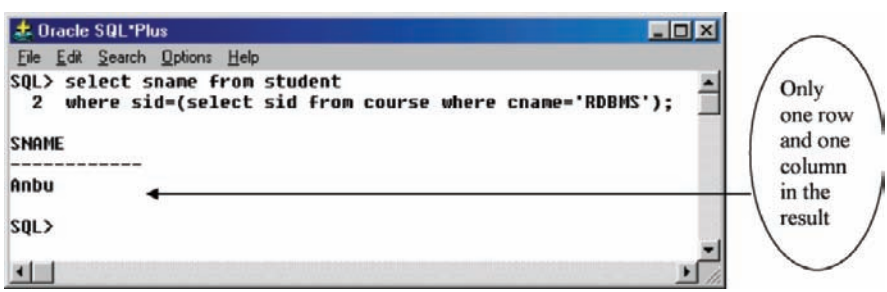


Fig. 4.126. Scalar subquery

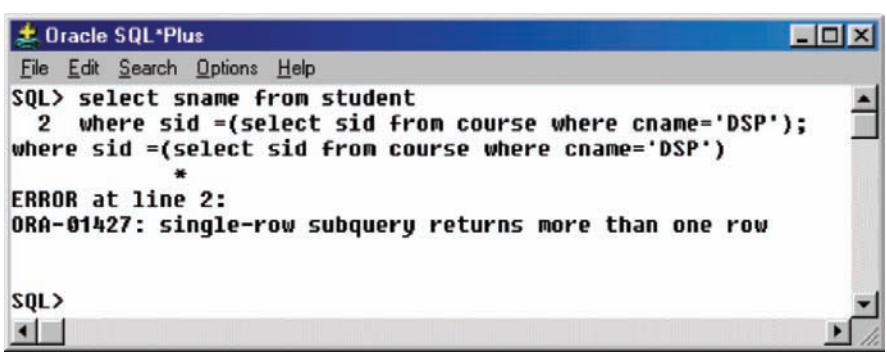


Fig. 4.127. Wrong use of scalar subquery

From Fig. 4.127, it is clear that scalar subquery cannot be used to retrieve multiple rows or multiple column result.

The solution to get the name of the student who has opted for DSP course is to use IN operator. The IN operator is true if value exists in the result of subquery. The SQL command using IN operator and the corresponding output are shown in Fig. 4.128.

4.16.1 Correlated Subquery

In the case of correlated subquery, the processing of subquery requires data from the outer query.

EXISTS Operator in Correlated Subquery

The **EXISTS** operator is used in correlated subquery to find whether a value retrieved by the outer query exists in the results set of the values retrieved by the inner query or subquery.

Example of EXISTS Command

Let us consider two tables ORDER1 and PRODUCT. The attributes (columns) of the table ORDER1 are orderID, quantity, productID. The attributes of the table PRODUCT are productID, productname, and price. The contents of the two table ORDER1 and PRODUCT are shown in Figs. 4.129 and 4.130.

The orderID which gives the order for the car "Maruti Esteem" can be found using the SQL command EXISTS. The SQL command and the corresponding output are shown in Fig. 4.131.

From Fig. 4.131, we can observe that the data for the inner query require the data from the outer query.

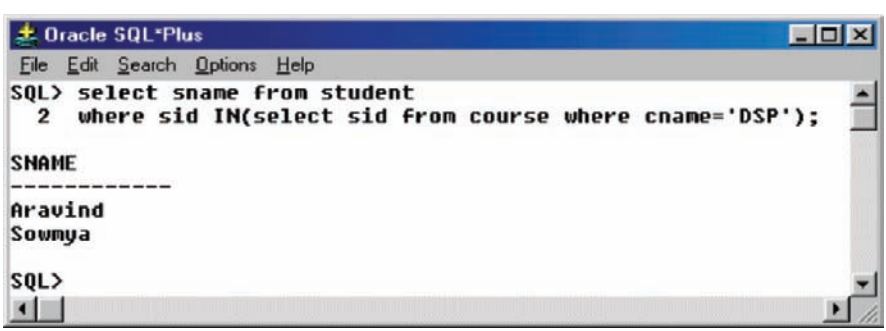


Fig. 4.128. Subquery to return multiple row result

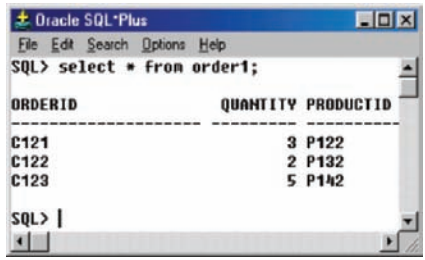


Fig. 4.129. Table order1

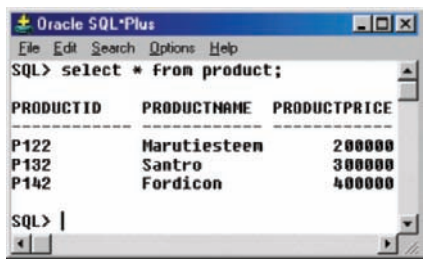


Fig. 4.130. Table product

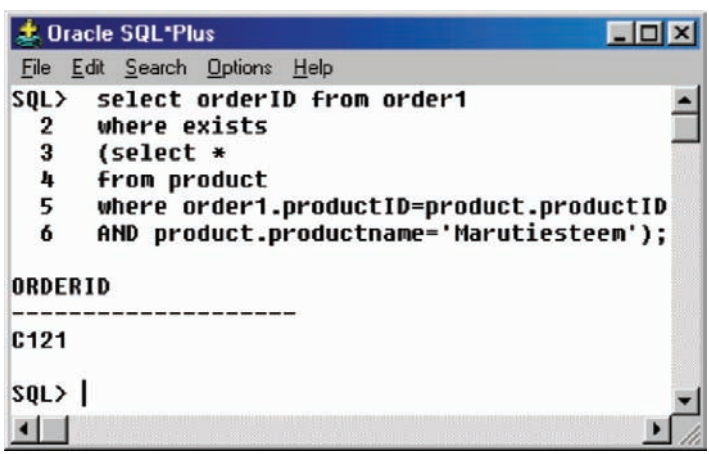


Fig. 4.131. Data retrieval using EXISTS command

Example of NOT EXISTS Operator

In order to understand NOT EXISTS clause, let us consider two relations EMPLOYEE and DEPENDENT. Here DEPENDENT refers to those who are dependent on EMPLOYEE. The attributes of EMPLOYEE relation are eid (employee ID), ename (employee name). The attributes of the DEPENDENT relation are name (which refers to dependent name) and eid (employee ID).

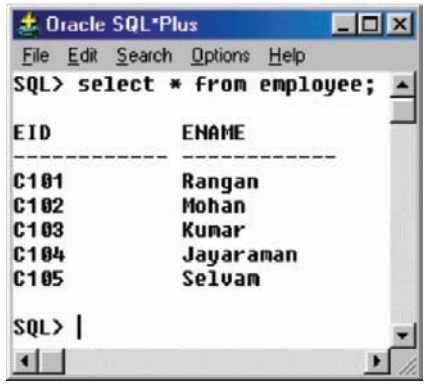


Fig. 4.132. EMPLOYEE table

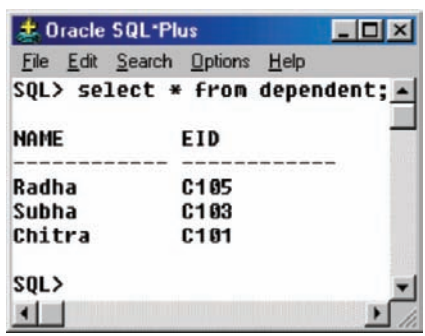


Fig. 4.133. DEPENDENT table

The contents of the table EMPLOYEE and DEPENDENT are shown in Figs. 4.132 and 4.133.

Query: Find the name of the employee who is not having any dependent?

Solution. The SQL command to get the name of the employee who is not having any dependent and the corresponding output are shown in Fig. 4.134.

The NOT EXISTS clause is used to retrieve the name of the employee who is not having dependent.

Comparison Operator ALL

The comparison operators that are used in multiple row subqueries are IN, ANY, ALL. In this section let us discuss the use of ALL comparison operator. The ALL comparison operator compare value to every value returned by the subquery.

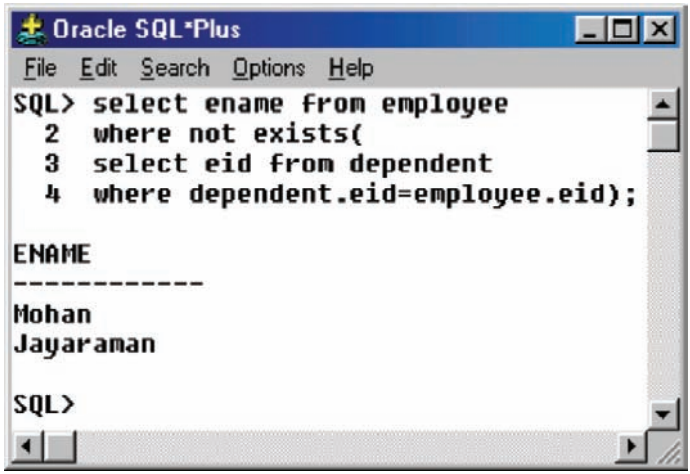


Fig. 4.134. NOT EXISTS command

Example

In order to understand the ALL comparison operator, let us consider the relation STAFF. The attributes of the staff relation are shown in table STAFF.

STAFF

EMPID	EMPNAME	DEPTNAME	AGE	SALARY
C201	Bhaskar	Electrical	24	12,500
C202	Ramakrishnan	Electronics	44	24,000
C203	Mathew	Electronics	43	23,000
C204	Natrajan	IT	38	18,500
C205	Krishnan	IT	36	17,000
C206	Usha	Electronics	40	20,000

Query: Find the name of the employee in Electronics Department who is getting the maximum salary?

Solution: The SQL command ALL can be used to find the name of the STAFF in the Electronics who is getting maximum salary. The SQL command and the corresponding output are shown in Fig. 4.135.

Here the ALL comparison operator is used to retrieve the name of the staff from a particular department who is getting maximum salary.

Comparison Operator ANY

The ANY operator compares a value to each value returned by a subquery. Here <ANY means less than maximum

>ANY means more than the minimum

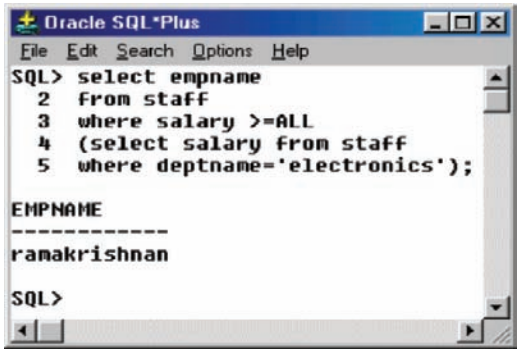


Fig. 4.135. Use of ALL comparison operator

Case 1: <ANY. Let us use the operator <ANY to retrieve the names of the staff who are getting salary less than the staff who is getting the maximum salary (in our case it is "ramakrishnan").

The SQL command to retrieve the names of the staff who are getting salary less than the staff who is getting the maximum salary is shown in Fig. 4.136.

The SQL command <ANY is used to retrieve the name of the staff who are getting the salary less than the staff who is getting the maximum salary. In our case staff "ramakrishnan" of electronics is getting the maximum salary (refer STAFF table). Our query should return the name of the staff who are getting salary less than the staff "ramakrishnan." From Fig. 4.136, it is evident the name returned by the query are the staff who are getting salary less than the staff "ramakrishnan."

Case 2: >ANY Clause. The operator >ANY returns values that are greater than the minimum value.

Example

Query: Retrieve the name of the staff who are getting salary greater than the staff who is getting the least salary?

Solution: The SQL operator >ANY can be used to get answer for the query mentioned above. The SQL command and the corresponding output are shown in Fig. 4.137.

From the table STAFF it is clear that the staff who is getting the least salary is "Bhaskar." We have to get the names of the staff who are getting salary greater than "Bhaskar."

From Fig. 4.137, it is clear that the operator >ANY has returned the names of the staff who are getting salary greater than "Bhaskar."

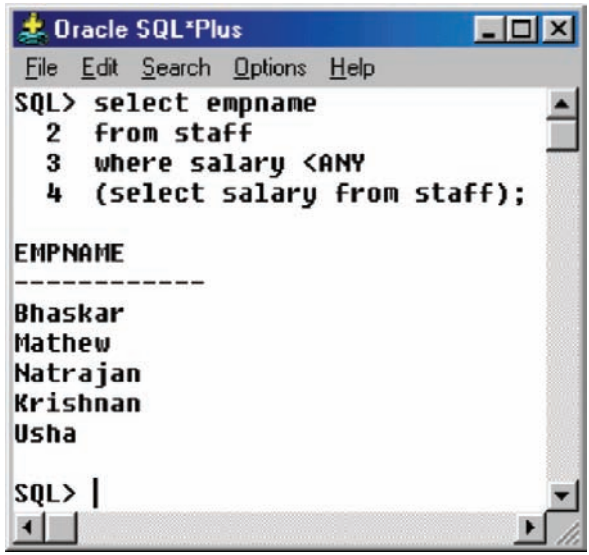


Fig. 4.136. Use of <ANY clause

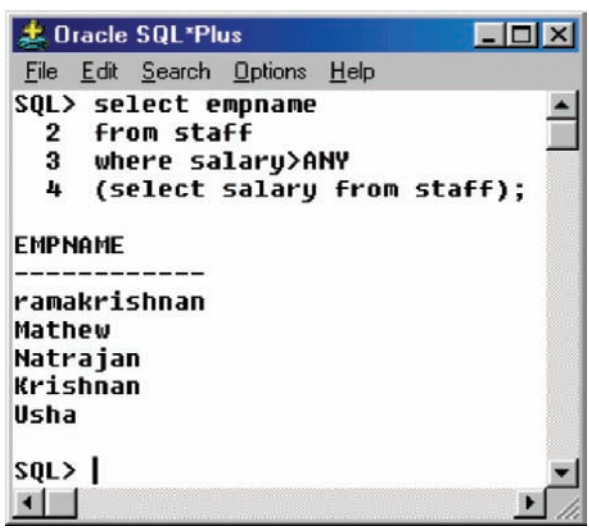


Fig. 4.137. Use of >ANY clause

Dual Table

The dual table contains one row and one column. The datatype associated with the dual table is varchar2(1). In order to know about dual table, we can issue DESC command as shown in Fig. 4.138.

Fig. 4.138. Description of dual table

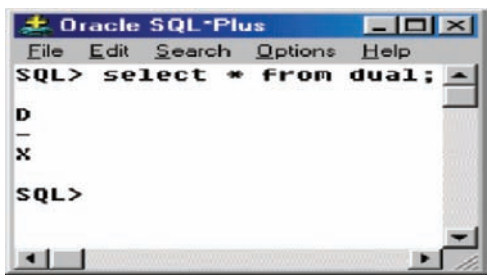


Fig. 4.139. Selection from Dual

From Fig. 4.138, it is clear that the name of the column is DUMMY. If we want to know how many rows that a DUAL table can return, we can issue SELECT command as shown in Fig. 4.139. From Fig. 4.139, it is clear that the dual table can return a row. Dual table can be used to compute a constant expression.

Determining System Date from Dual

It is possible to determine system date from the dual table. The SQL command and the corresponding output are shown in Fig. 4.140.

We have evaluated the system date from the dual table. It is also possible to evaluate constant expression using the dual table.

Evaluation of Constant Expression Using DUAL

It is possible to evaluate constant expressions using DUAL table. Some of the examples of evaluation of constant expressions are shown in Fig. 4.141.

From Fig. 4.141 it is clear that DUAL table can be used to evaluate constant expressions which will give single row output. For our example we have taken simple mathematical operations like addition, multiplication, division, and subtraction.

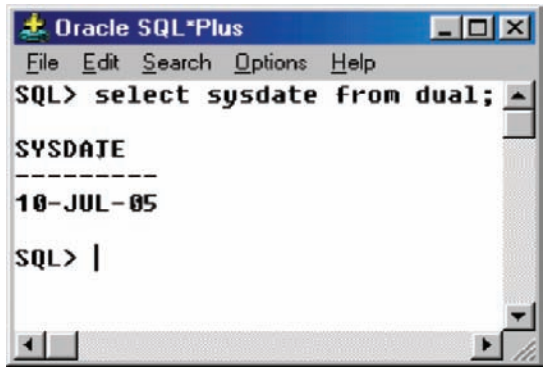


Fig. 4.140. System date from dual

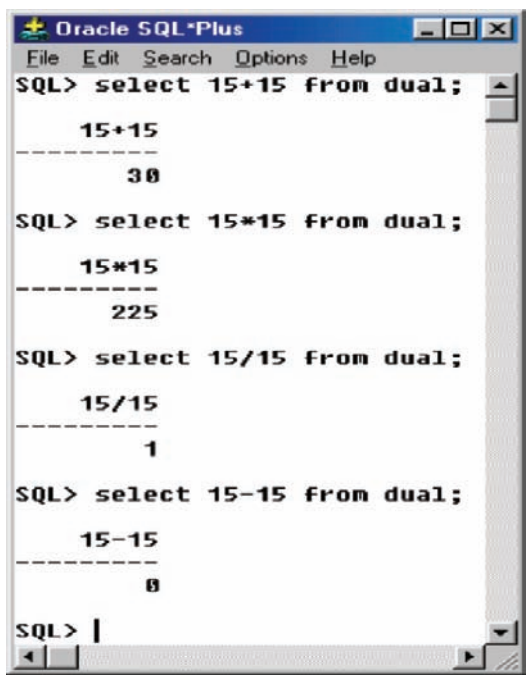


Fig. 4.141. Evaluation of constant expressions

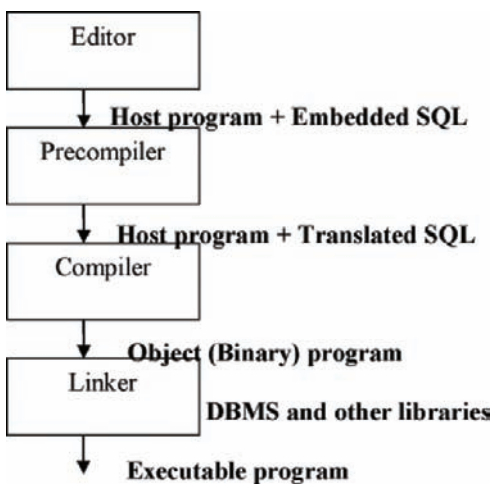
4.17 Embedded SQL

SQL can be used in conjunction with a general purpose programming language such as PASCAL, C, C++, etc. The programming language is called the host language. Embedded SQL statements are SQL statements written within application programming languages such as C and Java. The embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor

can extract the SQL statements. These statements are preprocessed by an SQL precompiler before the application program is compiled. There are two types of embedded SQL, Static SQL, and Dynamic SQL. Embedded SQL provides the 3GL (Third Generation Language) with a way to manipulate a database. Embedded SQL supports highly customized applications. It also supports background applications running without user intervention.

SQL Precompiler

A precompiler is used to translate SQL statements embedded in a host language into DBMS library calls, which can be implemented in the host language. The function of the precompiler is shown below:



Sharing Variables

Variables to be shared between the embedded SQL code and the host language have to be specified in the program.

EXEC SQL begin declare section;
Varchar userid [10], password [10], cname [15];
Int cno;
EXEC SQL end declare section;
We also should declare a link to the DBMS so that database status information can be accessed.
EXEC SQL include sqlca;
This allows access to a structure sqlca, of which the most common element sqlca.sqlcode has the value 0 (operation OK), >0 (no data found), and <0 (an error).

Connecting to the DBMS

Before operations can be performed on the database, a valid connection has to be established. A model is shown below:
EXEC SQL connect :userid identified by :password;

- In all SQL statements, variables with the ":" prefix refer to shared host variables, as opposed to database variables.
- This assumes that userid and password have been properly declared and initialized.

When the program is finished using the DBMS, it should disconnect using:
EXEC SQL commit release;

Queries Producing a Single Row

A single piece of data (or row) can be queried from the database so that the result is accessible from the host program.

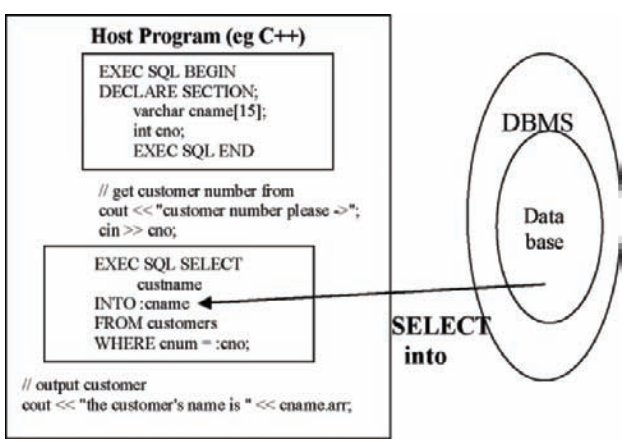
EXEC SQL SELECT custname
INTO :cname
FROM customers
WHERE cno = :cno;

Thus the custname with the unique identifier :cno is stored in :cname.

However, a selection query may generate many rows, and a way is needed for the host program to access results one row at a time.

SELECT with a Single Result

The syntax to select with a single result is shown below:



Static SQL

The source form of a static SQL statement is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

A source program containing static SQL statements must be processed by an SQL precompiler before it is compiled. The precompiler turns the SQL statements into host language comments, and generates host language statements to invoke the database manager. The syntax of the SQL statements is checked during the precompile process.

The preparation of an SQL application program includes precompilation, the binding of its static SQL statements to the target database, and compilation of the modified source program.

Dynamic SQL

Programs containing embedded dynamic SQL statements must be precompiled like those containing static SQL, but unlike static SQL, the dynamic SQL statements are constructed and prepared at run time. The SQL statement text is prepared and executed using either the PREPARE and EXECUTE statements, or the EXECUTE IMMEDIATE statement. The statement can also be executed with cursor operations if it is a SELECT statement.

Summary

This chapter has introduced the most popular relational database language SQL (Structured Query Language). SQL has become the de facto standard language for interacting with all major database programs. The three main divisions in SQL are DDL, DML, and DCL. The data definition language (DDL) commands of SQL are used to define a database which includes creation of tables, indexes, and views. The data manipulation commands (DML) are used to load, update, and query the database through the use of the SELECT command. Data control language (DCL) is used to establish user access to the database.

This chapter has focused on how to create the table, how to insert data into the table. Examples are shown to understand the table creation and manipulation process. The subset of SELECT command described in this chapter allows the reader to formulate problems involving the project, restrict, join, union, intersection, and difference operators of relational algebra.

Review Questions

4.1. Prove the statement "*When the column of a view is directly derived from a column of a base table, that column inherits any constraints that apply to the column of the base table*" by using suitable example.

To prove this statement, let us create a base table by name t1. The base table t1 has two columns name and age. Now a constraint is imposed on the age, that is age should be greater than 18. The syntax to create the base table t1 with the constraint on the age is shown below:

Step 1: Base table creation with the name t1 and constraint on age (Fig. 4.142).

```
SQL> create table t1
2 (name varchar(12),
3 age number(3),
4 check(age>18));
Table created.
```

Step 2: Create a view by name t2 from the base table t1. The SQL command to create the view t2 is shown in Fig. 4.143.

Step 3: Now try to insert values into view t2 by not violating the constraint and then by violating the constraint (Fig. 4.144). Then try to insert values into the view t2 by violating the check constraint.

Note: Since the age is greater than 18 the values are inserted into view t2. Now insert value into t2 by violating the constraint (by inserting the age less than or equal to 18).

If we are violating the constraints on the column of the base table we are getting an error message.

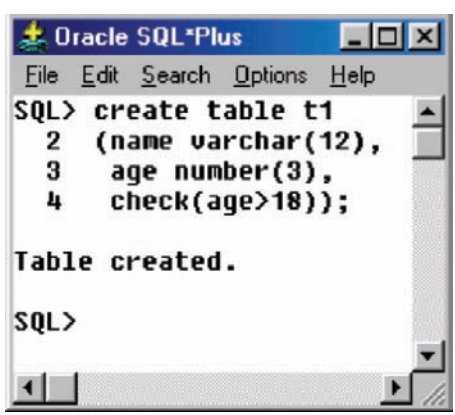


Fig. 4.142. Creation of table t1

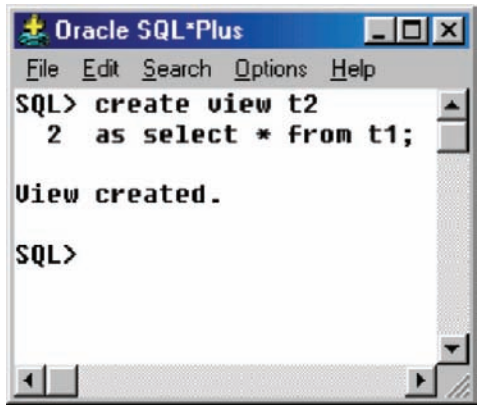


Fig. 4.143. Creation of view t2

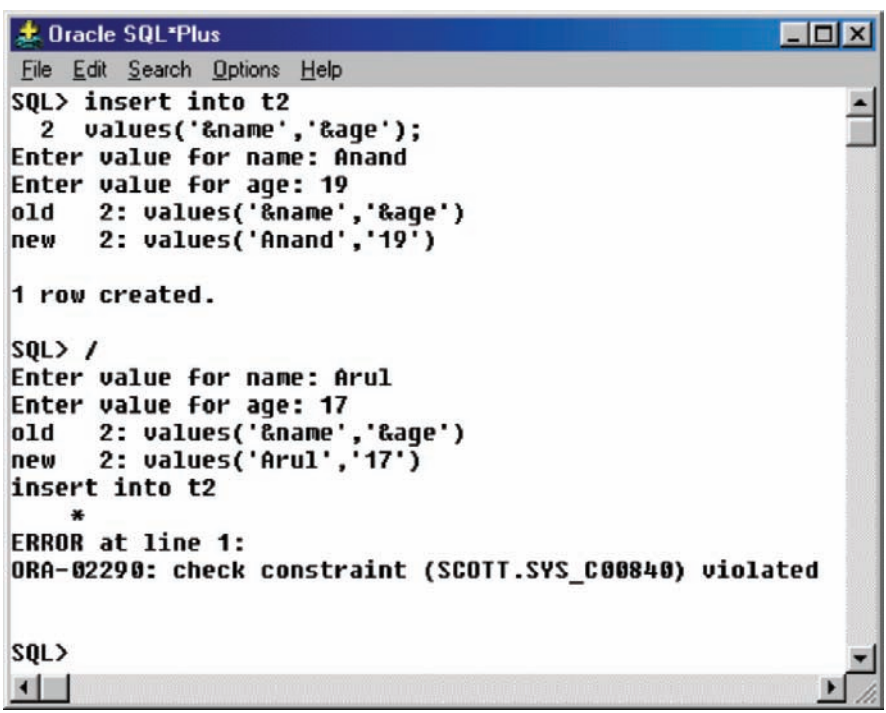


Fig. 4.144. Insertion of values into t2 without violating and violating constraint

4.2. What is the difference between the two SQL commands DROP TABLE and TRUNCATE TABLE?

Drop table command deletes the definition as well as the contents of the table, whereas truncate table command deletes only the contents of the table but not the definition of the table.

Example

We have a table by name t1. The contents of the table are seen by issuing the select command as shown in Fig. 4.145.

Step 1: Now issue the truncate table command. The syntax is:
TRUNCATE TABLE table name; as shown in Fig. 4.146.

Step 2: After issuing the truncate table command try to see the contents of the table. You will get the message as no rows selected as shown in Fig. 4.147.

Step 3: Now we have the table t2. See the contents of the table by issuing select command as shown in Fig. 4.148.

Step 4: Now use the drop command, to drop the table t2 as shown in Fig. 4.149.

Step 5: Now see the effect of the drop command by using the select command as shown in Fig. 4.150.

Note: If we issue the drop command, the definition as well as the contents of the table is deleted and we get the error message as shown in Fig. 4.150.

4.3. Is it possible to create a table from another table. If so give an example

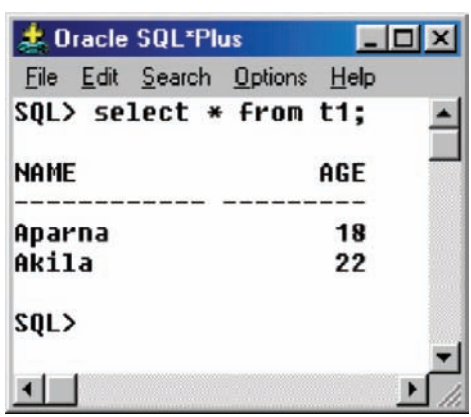


Fig. 4.145. Content of the table t1

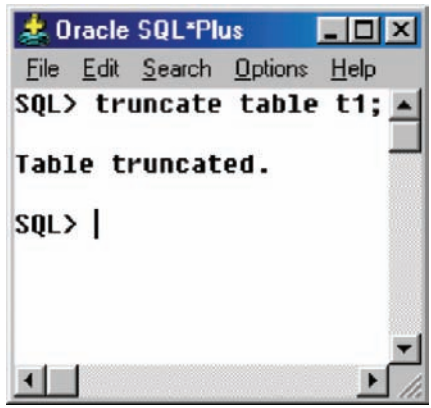


Fig. 4.146. Truncation of table t1

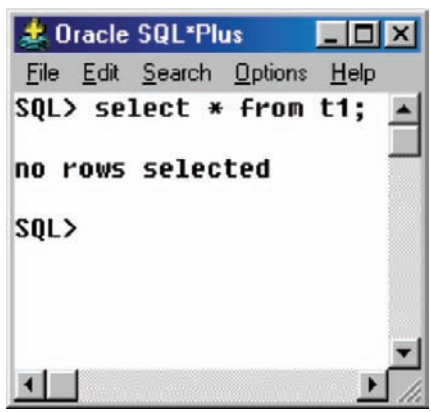


Fig. 4.147. Selection after truncation

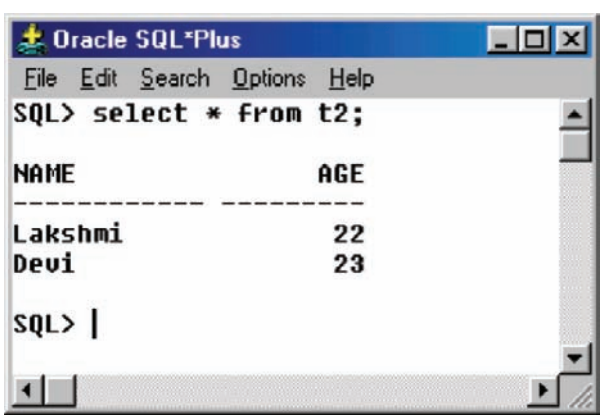


Fig. 4.148. Contents of the table t2

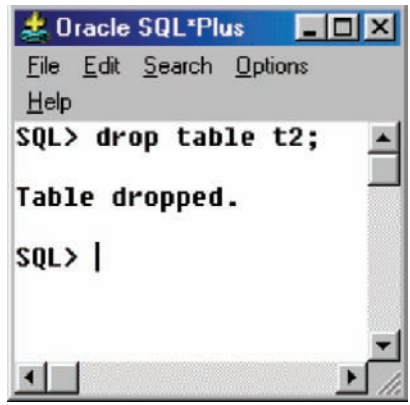


Fig. 4.149. Dropping the table t2

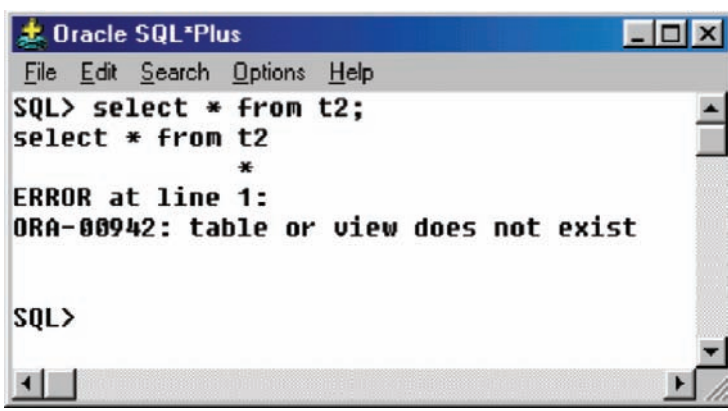


Fig. 4.150. Selection after dropping the table

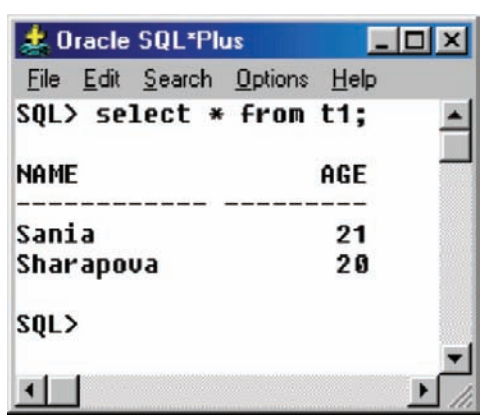


Fig. 4.151. Contents of table t1

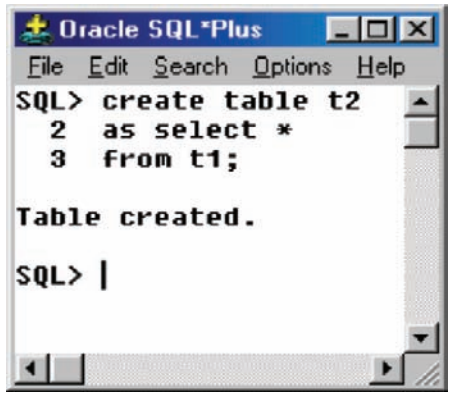


Fig. 4.152. Table t2 from table t1

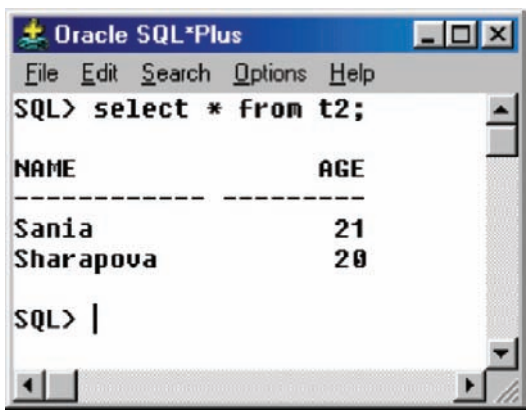


Fig. 4.153. Contents of table t2

Yes, it is possible to create table from another table using SQL. Consider table t1 as shown in Fig. 4.151. We can create another table t2 from the table t1. The SQL command to create the table t2 from the table t1 is shown in the Fig. 4.152.

Now let us try to view the content of the table t2. The content of the table t2 is shown in Fig. 4.153.

From Fig. 4.153, it is clear that the contents of the table t2 matches with the table t1 (refer Fig. 4.151). Hence it is possible to create table from another table.

4.4. What is the difference between COUNT, COUNT DISTINCT, and COUNT (*) in SQL?

The command COUNT counts the number of rows in a table by ignoring all null values. The command COUNT (*) counts the number of rows in a

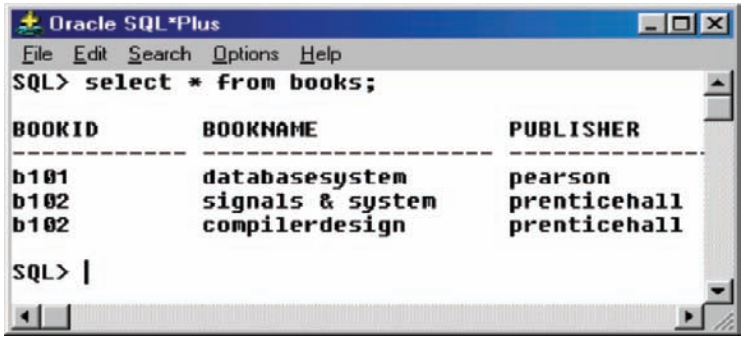


Fig. 4.154. Contents of the table BOOKS

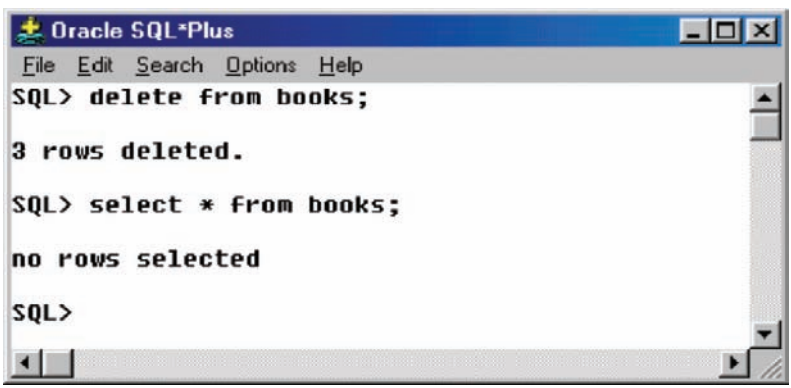


Fig. 4.155. Contents of the table BOOKS deleted using DELETE command

table by including the rows that contains null values. COUNT DISTINCT counts the number of rows in the table by ignoring duplicate values.

4.5. If we want to delete all the rows in the table, it can be done in two ways (1) Issue the command DELETE FROM table name (2) TRUNCATE TABLE table name. What is the difference between these two commands?

We have a table by name BOOKS. The content of the table BOOKS are shown in the Fig. 4.154.

Step 1: The contents of the table BOOKS are deleted by using DELETE command as shown in Fig. 4.155.

Step 2: The table BOOKS is again populated with the data and the command TRUNCATE is used to delete the contents of the table which is shown in Fig. 4.156.

The advantage offered by the TRUNCATE command is the speed. When Oracle executes this command, it does not evaluate the existing records within a table; it basically chops them off. In addition to speed, the TRUNCATE

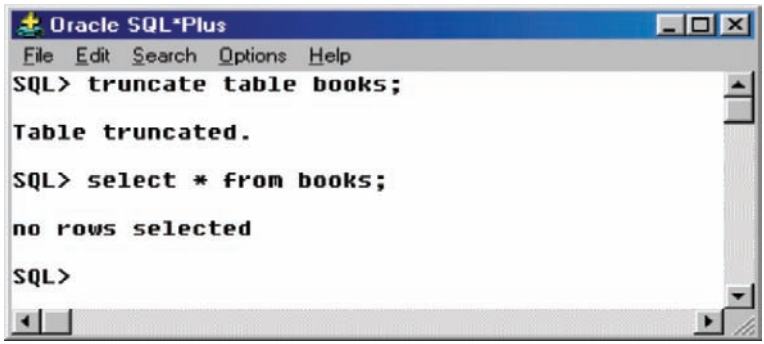


Fig. 4.156. Contents of the table BOOKS deleted using TRUNCATE command

command provides the added benefit of automatically freeing up the table space that the truncated records previously occupied.

When the table contents are deleted by using DELETE command, it forces Oracle to read every row before deleting it. This can be extremely time consuming.

4.6. What are subqueries? How will you classify them?

Subquery is query within a query. A SELECT statement can be nested inside another query to form a subquery. The query which contains the subquery is called outer query. It can be classified as (a) scalar subquery and (b) correlated subquery, and (c) uncorrelated subquery.