UNIT-4

Structured Query Language: Introduction, History of SQL Standard, Commands in SQL, Data Types in SQL, Data Definition Language, Selection Operation, Projection Operation, Aggregate functions, Data Manipulation Language, Table Modification Commands, Table Truncation, Imposition of Constraints, Join Operation, Set Operation,View, Sub Query, Embedded SQL

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.

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.

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.

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." In the first phase of the System R project, researchers concentrated on developing a basic version of the RDBMS.

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 ranunder the MVS operating system.

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. 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

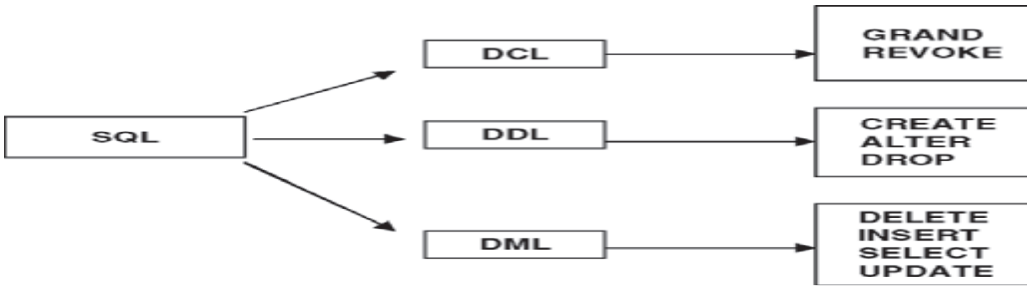
DDL commands are used to define a database, including creating, altering, and dropping tables and establishing constraints.

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 classification of commands in SQL is shown below.



4.4 Data Types in SQL

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.

NUMBER Datatype

The NUMBER datatype stores zero, positive, and negative fixed and floating point numbers. 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 4GB of binary data.

CLOB Datatype
The CLOB datatype can store up to 4GB 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 4GB of data.

4.5 Data Definition Language

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

```
SQL> create table peaks
  2  (serial_no number(2),
  3   mountain varchar(12),
  4   height number(5),
  5   place varchar(12),
  6   range varchar(12));

Table created.
```

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:

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 below shows the description of the table PEAKS.

```
SQL> desc peaks;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SERIAL_NO                                          NUMBER(2)
 MOUNTAIN                                           VARCHAR2(12)
 HEIGHT                                             NUMBER(5)
 PLACE                                              VARCHAR2(12)
 RANGE                                              VARCHAR2(12)
```

To **insert** values into the table

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

Now to insert the next set of values, use the slash as shown in Fig. 4.6.

```
SQL> /
Enter value for serial_no: 2
Enter value for mountain: godwinaustin
Enter value for height: 8611
Enter value for place: india
Enter value for range: karakoram
old   2: values('&serial_no','&mountain','&height','&place','&range')
new   2: values('2','godwinaustin','8611','india','karakoram')

1 row created.
```

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.

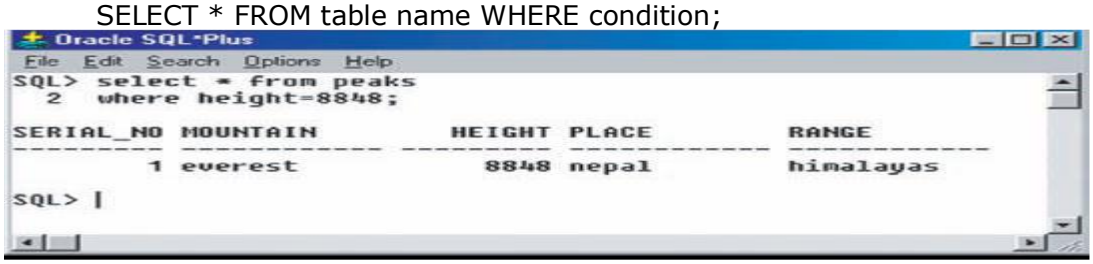
```
SQL> select * from peaks;

SERIAL_NO MOUNTAIN         HEIGHT PLACE         RANGE
--------- ------------ ---------- ------------ ------------
        1 everest            8848 nepal        himalayas
        2 godwinaustin       8611 india        karakoram
        3 kanchenjunga       8579 nepal        himalayas
```

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;



4.7 Projection Operation

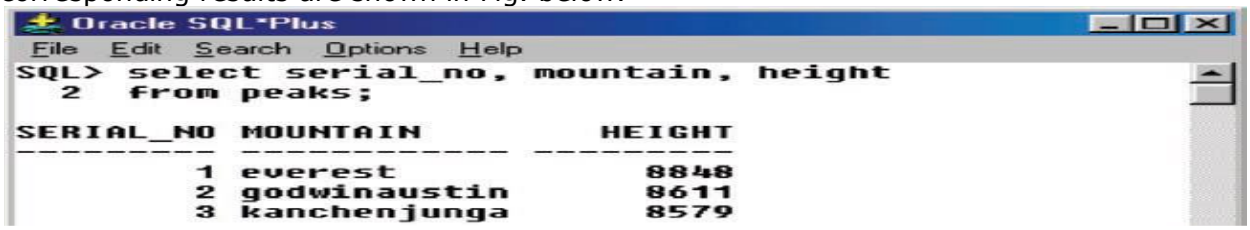
The projection operation performs column wise filtering. Specific columns are selected in projection operation.

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.

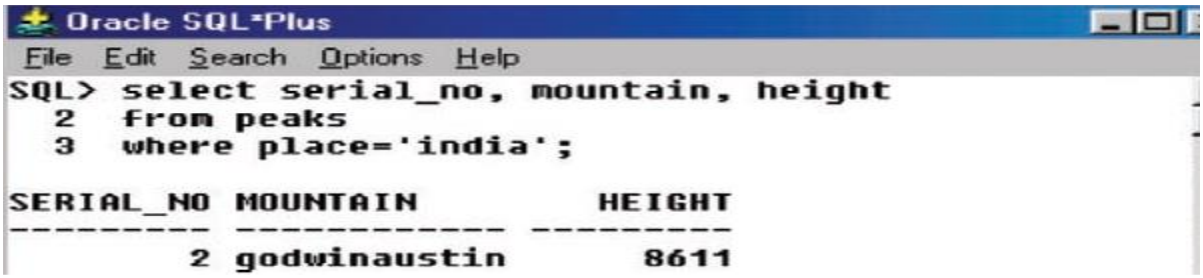
The SQL command to perform PROJECTION operation on the relation PEAKS and the corresponding results are shown in Fig. below.



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

Syntax for Selection and Projection

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



SELECTION and PROJECTION operation

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.

S.No	Built-in function	Use
1.	COUNT	to count the number of rows of the relation `SELECT COUNT(column_name) FROM table_name WHERE CONDITION;`
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

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.

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.

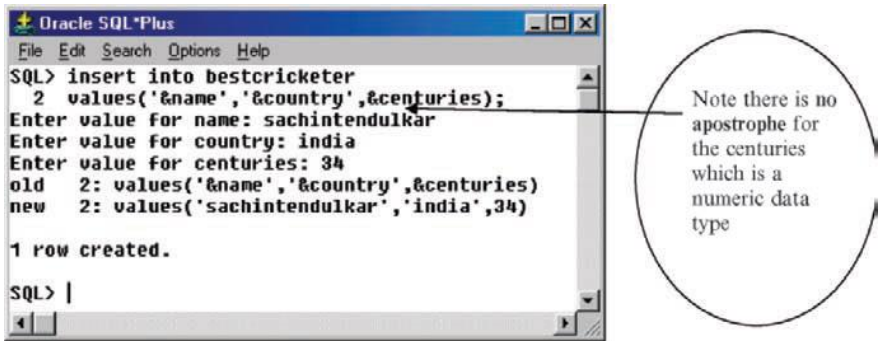


Fig. 4.29. Inserting a new row to the table

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.

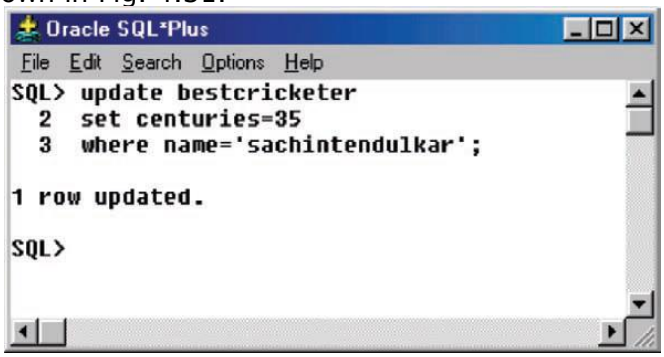


Fig. 4.31. Table updation using UPDATE command

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.

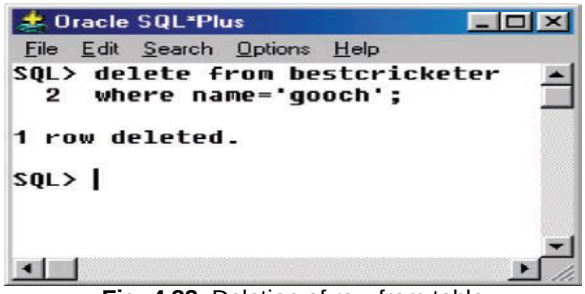


Fig. 4.33. Deletion of row from table

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.

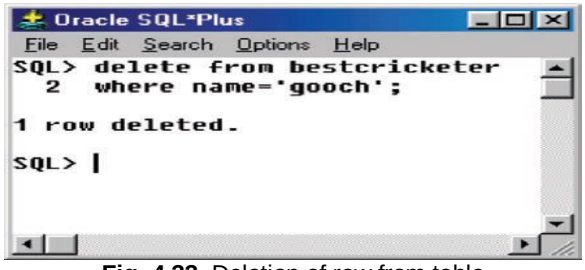


Fig. 4.33. Deletion of row from table

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;

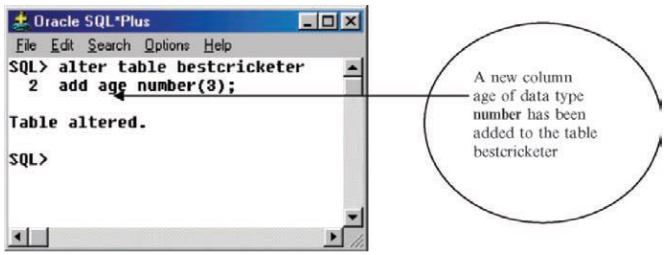


Fig. 4.35. Adding a column to the table

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.

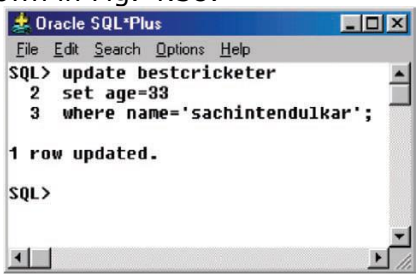


Fig. 4.38. Insertion of data to the new column age

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;

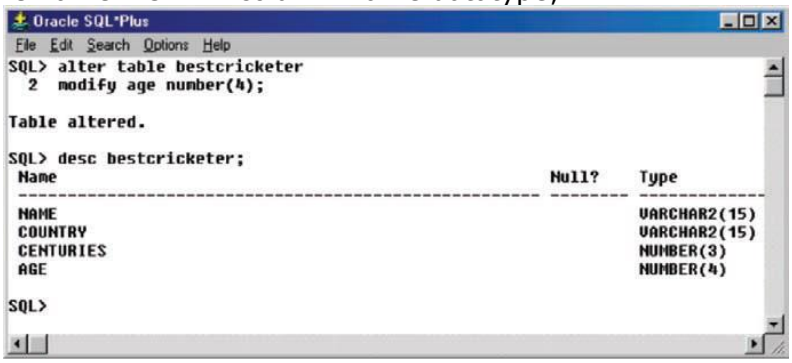


Fig. 4.40. Modified width of the datatype

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;

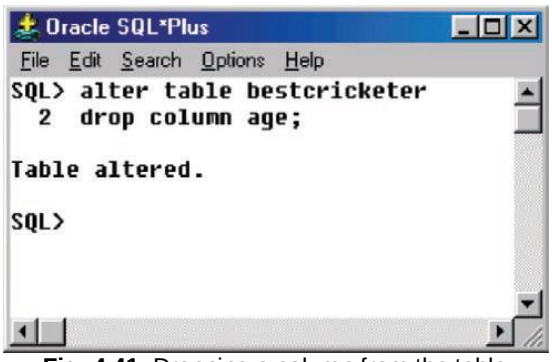


Fig. 4.41. Dropping a column from the table

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. 4.44. Table truncation

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;

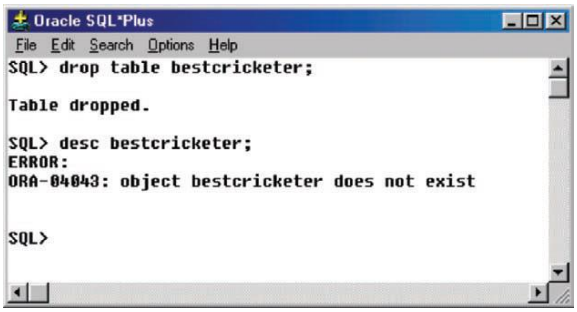


Fig. 4.48. Table descriptions after dropping the table

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

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.

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.

Difference Between NOT NULL and UNIQUE Constraint

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

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."

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

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.

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

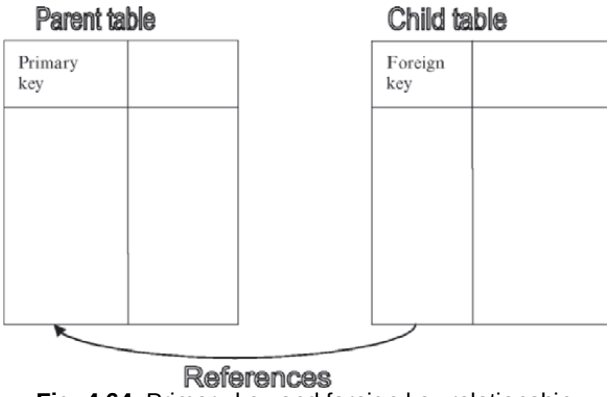
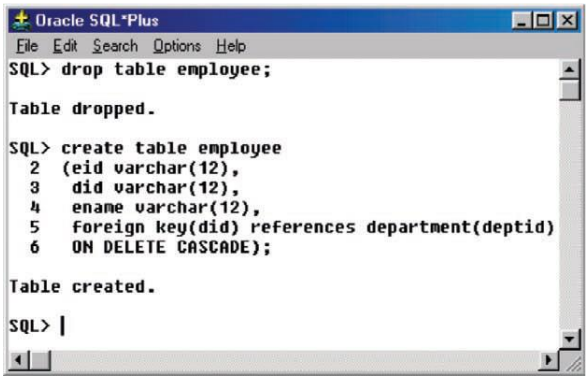


Fig. 4.64. Primary key and foreign key relationship

4.12.6 ON DELETE CASCADE

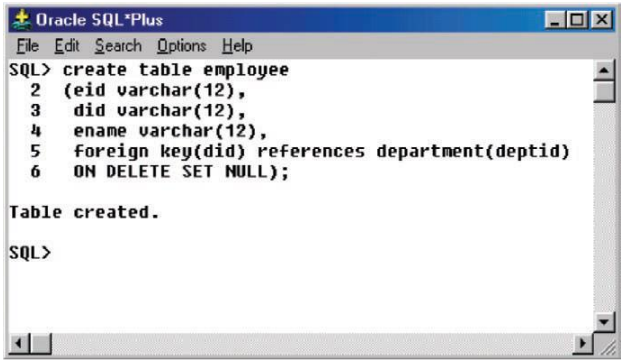
When the clause ON DELETE CASCADE is included in the child table, and if a row is deleted from the parent table then the corresponding referenced value in the child table will also be deleted.



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.



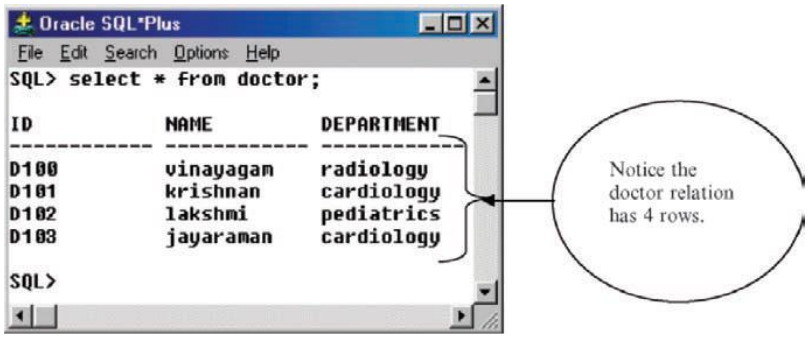
4.13 Join Operation

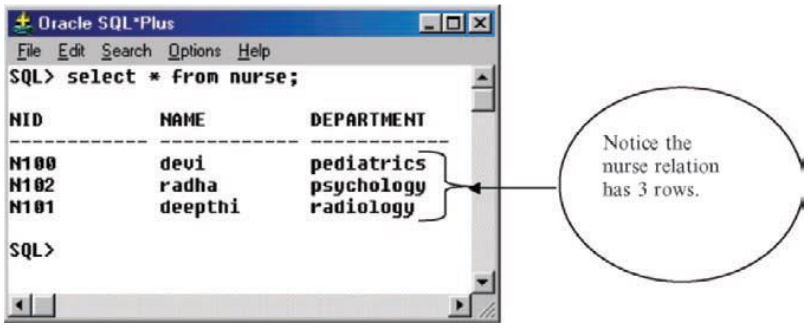
Join operation is used to retrieve data from more than one table. Before proceeding to JOIN operation let us discuss first the Cartesian product. Cartesian product with suitable selection and projection operation forms different types of join.

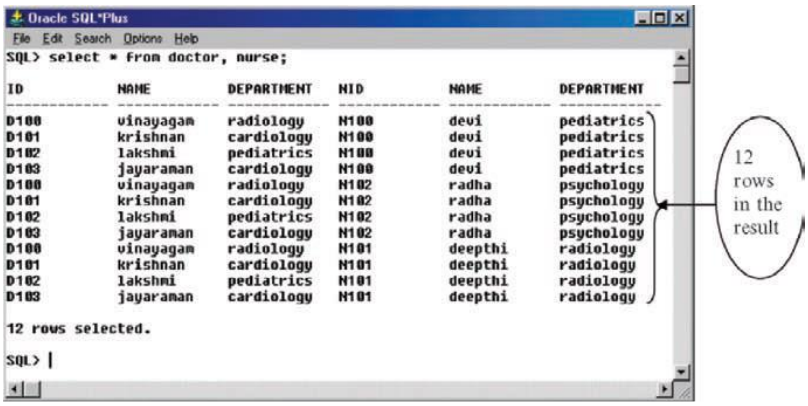
Cartesian Product

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:







The Cartesian product should return 4×3 = 12 rows. The SQL command to perform Cartesian product between the two relations doctor and nurse and the corresponding output are shown in Fig. above . 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.

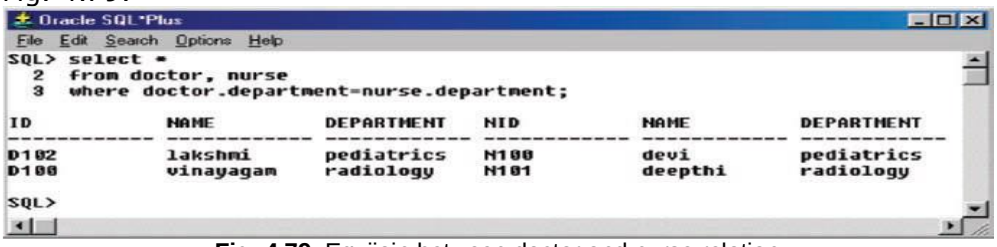


Fig. 4.79. Equijoin between doctor and nurse relation

4.14 Set Operation

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

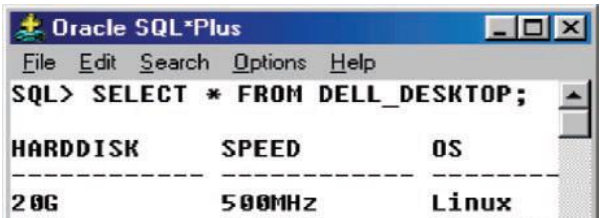
If we have two relations R and S then the set UNION operation contains tuples that either occurs in R or S or both.

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.



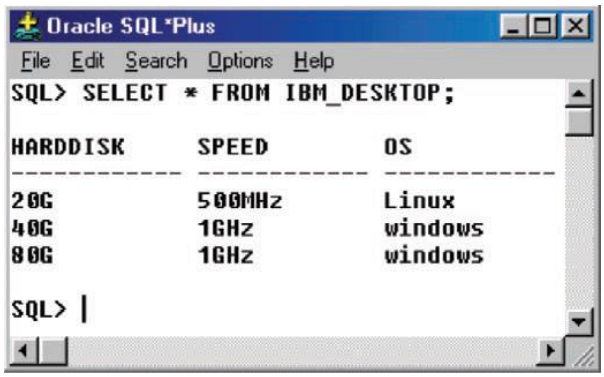


Fig.4.80

Fig.4.81

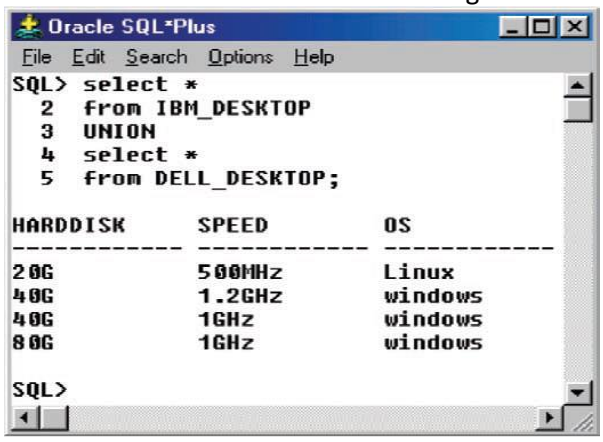


Fig 4.82

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.

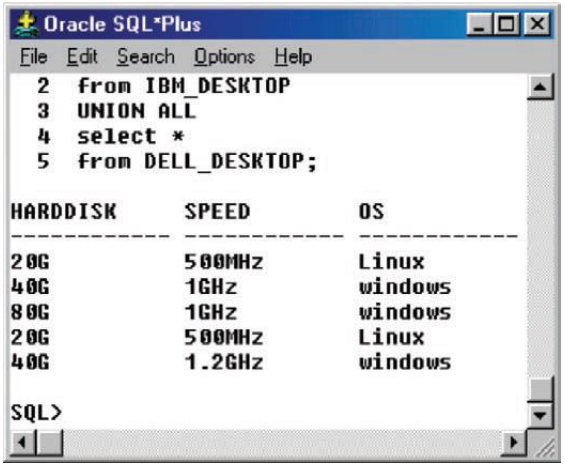


Fig.4.83

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.

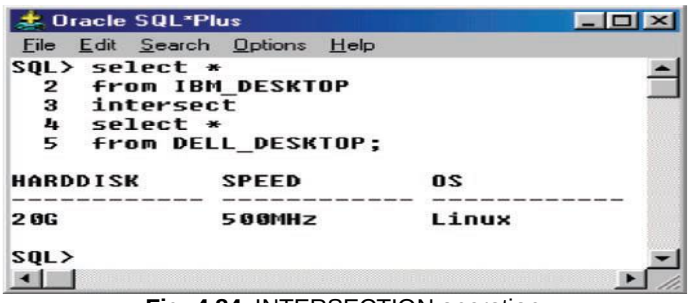


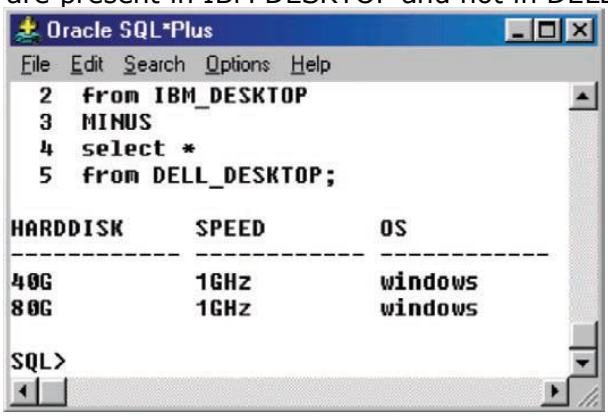
Fig. 4.84. INTERSECTION operation

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.

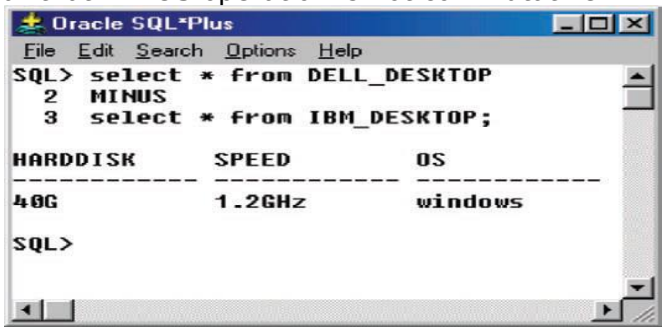
Case 1: IBM DESKTOP–DELL DESKTOP.

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

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

In this article we will learn about creating, deleting and updating Views.

Sample Tables:

StudentDetails

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

StudentMarks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

CREATING VIEWS

We can create View using **CREATE VIEW** statement. A View can be created from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHERE condition;
```

view_name: Name for the View
table_name: Name of the table
condition: Condition to select rows

Examples:

- **Creating View from a single table:**
  - In this example we will create a View named DetailsView from the table StudentDetails.
    Query:

```
CREATE VIEW DetailsView AS SELECT NAME, ADDRESS FROM StudentDetails
WHERE S_ID < 5;
```

To see the data in the View, we can query the view in the same manner as we query a table.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

- In this example, we will create a view named StudentNames from the table StudentDetails.
  Query:
- CREATE VIEW StudentNames AS SELECT S_ID, NAME FROM StudentDetails ORDER BY NAME;
  If we now query the view as,

```
SELECT * FROM StudentNames;
```

Output:

S_ID	NAMES
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

- **Creating View from multiple tables**: In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement. Query:
- CREATE VIEW MarksView AS SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME = StudentMarks.NAME;
  To display data of View MarksView:

```
SELECT * FROM MarksView;
```

Output:

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

DELETING VIEWS

We have learned about creating a View, but what if a created View is not needed any more? Obviously we will want to delete it. SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

Syntax:

```
DROP VIEW view_name;
```

view_name: Name of the View which we want to delete.

For example, if we want to delete the View **MarksView**, we can do this as:

```
DROP VIEW MarksView;
```

UPDATING VIEWS

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

- We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.

  Syntax:

- CREATE OR REPLACE VIEW view_name AS SELECT column1,coulmn2,.. FROM table_name WHERE condition;

  For example, if we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks** Table, we can do this as:

```
CREATE OR REPLACE VIEW MarksView AS SELECT StudentDetails.NAME,
StudentDetails.ADDRESS, StudentMarks.MARKS, StudentMarks.AGE
FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME =
StudentMarks.NAME;
```

  If we fetch all the data from MarksView now as:

```
SELECT * FROM MarksView;
```

Output:

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

- **Inserting a row in a view**:

  We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.**Syntax**:

- INSERT INTO view_name(column1, column2 , column3,..) VALUES(value1, value2, value3..);
-
- **view_name**: Name of the View

  Example:

  In the below example we will insert a new row in the View DetailsView which we have created above in the example of "creating views from a single table".

```
INSERT INTO DetailsView(NAME, ADDRESS) VALUES("Suresh","Gurgaon");
```

  If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

- **Deleting a row from a View**:
- Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.**Syntax**:

```
- DELETE FROM view_name WHERE condition;
-
```

- **view_name**:Name of view from where we want to delete rows
- **condition**: Condition to select rows

Example:
In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

```
DELETE FROM DetailsView WHERE NAME="Suresh";
```

If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

WITH CHECK OPTION
The WITH CHECK OPTION clause in SQL is a very useful clause for views. It is applicable to a updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.

- The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.
- If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

Example:
In the below example we are creating a View SampleView from StudentDetails Table with WITH CHECK OPTION clause.

```
CREATE VIEW SampleView AS SELECT S_ID, NAME FROM  StudentDetails WHERE NAME IS NOT NULL WITH CHECK OPTION;
```

In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL. For example,though the View is updatable but then also the below query for this View is not valid:

```
INSERT INTO SampleView(S_ID) VALUES(6);
```

NOTE: The default value of NAME column is *null*.

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.

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.

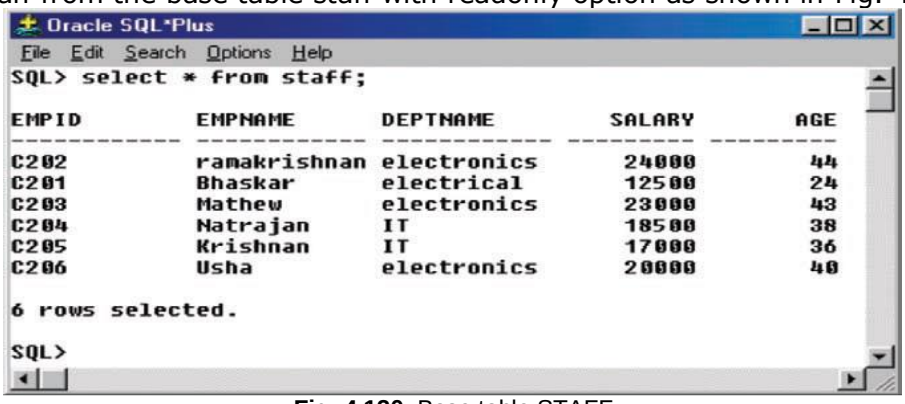


Fig. 4.120. Base table STAFF

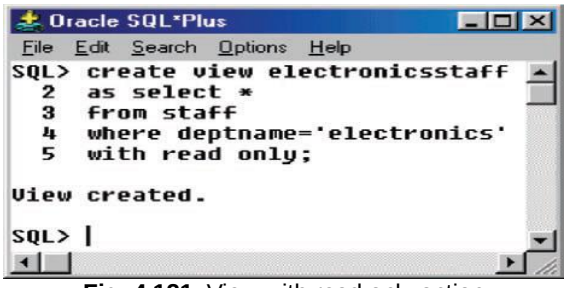


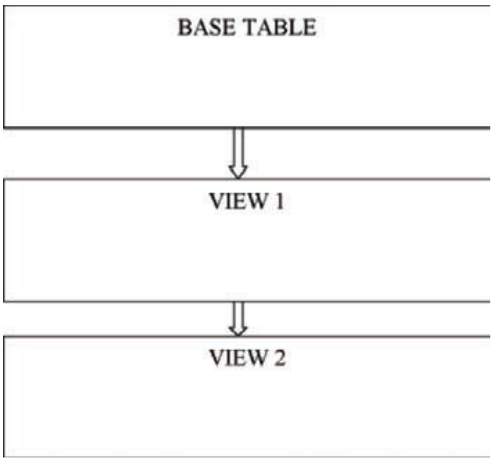
Fig. 4.121. View with read only option

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.

View From View

It is possible to create view from another view. This is diagrammatically shown in Fig. below. From Fig. below, 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.



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 YOUNGITSTAFF 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.

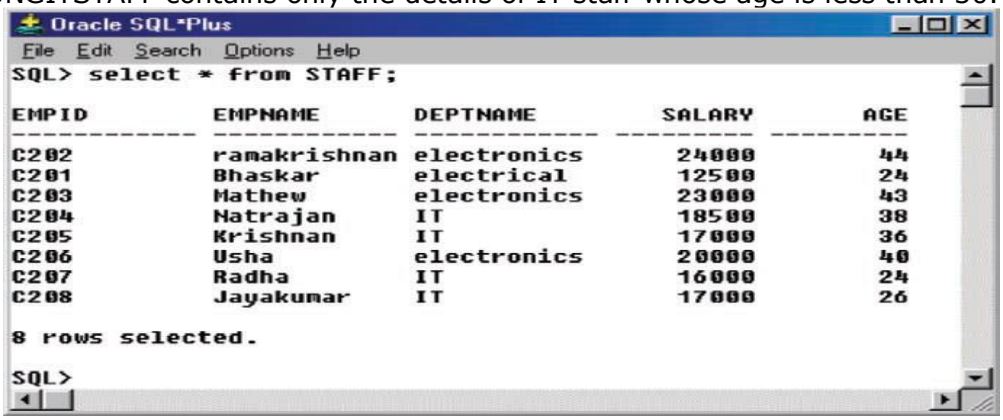


Fig. 4.103. Base table STAFF

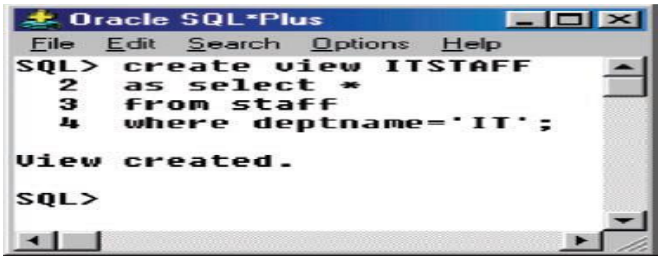


Fig. 4.104. View ITSTAFF from base table STAFF

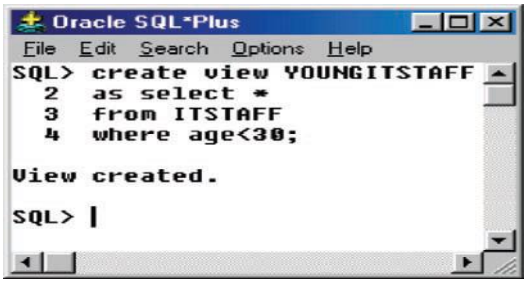


Fig. 4.105. View YOUNGITSTAFF from the view ITSTAFF

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

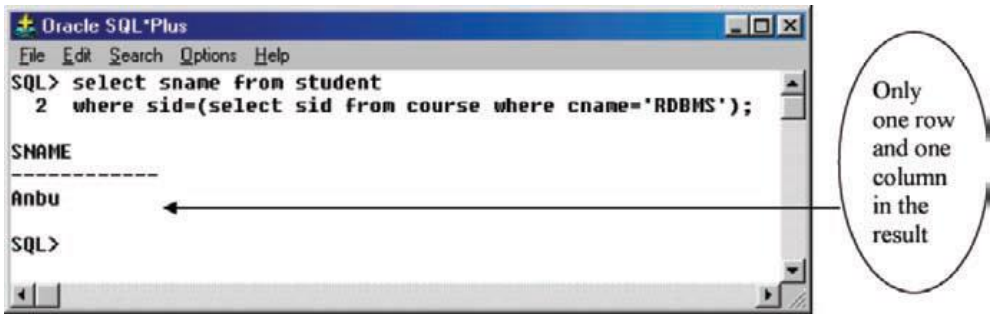
Answer : Yes. The view YOUNGITSTAFF, which is created from another view ITSTAFF can be queried like the base table.
Doubt 2: If it is possible to make any change in the view ITSTAFF which was created from the base table STAFF, will it reflect in the base table STAFF.
Answer : Yes, if it is possible to make any change in the view which was derived from the base table then the change will be reflected in the base table.
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.

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.

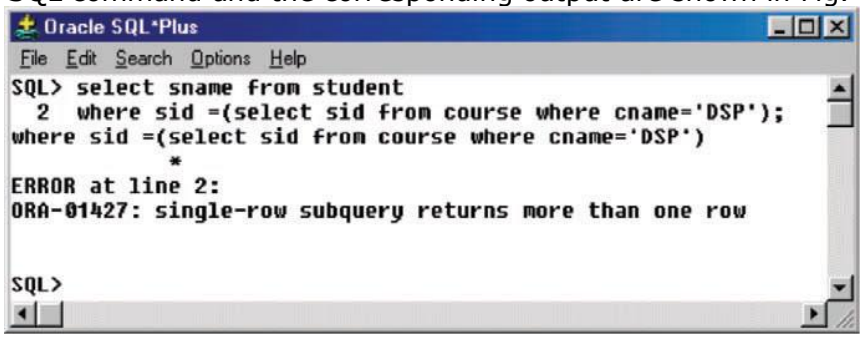
4.16 Sub Query

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.

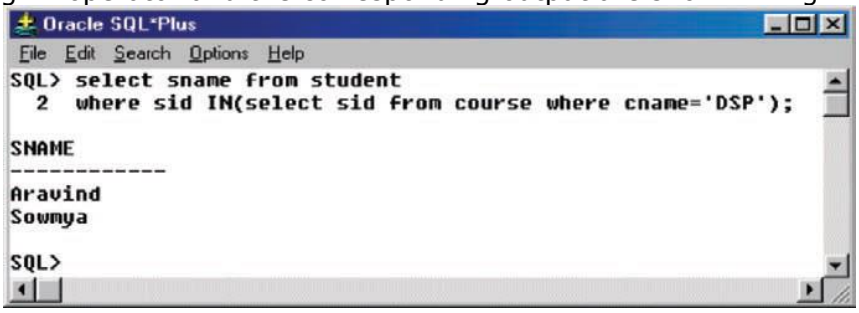
Example of Scalar Subquery
Scalar subquery returns single row single column result. To understand scalar subquery, consider two relations STUDENT and COURSE. The attributes of the STUDENT relation are SID, SNAME, AGE, and GPA. The attributes of COURSE relation are CID (Course ID), CNAME (Course ID),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



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

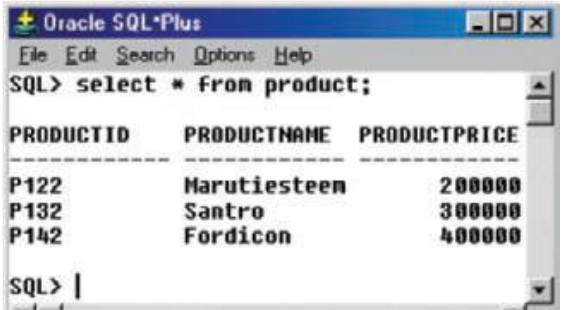
In the case of correlated subquery, the processing of subquery requires data from the outer query.

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.



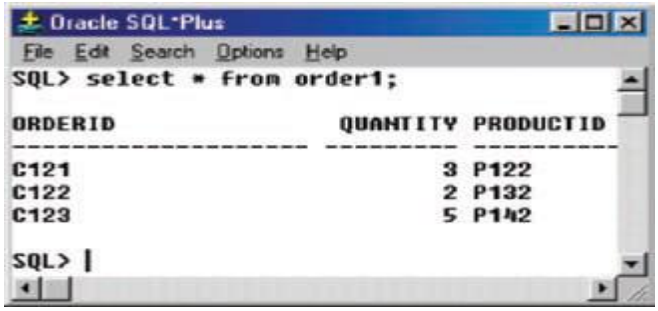


Fig. 4.129. Table order1

Fig. 4.130. Table product

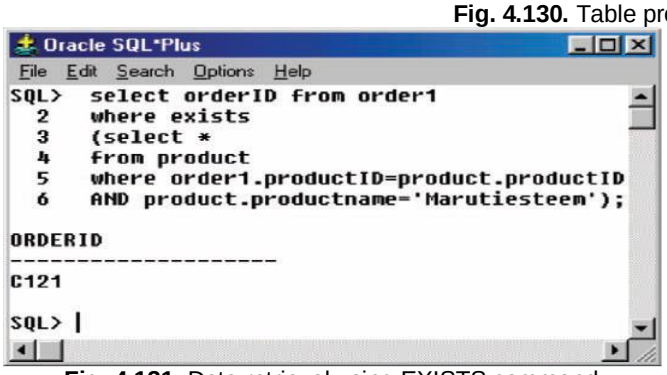


Fig. 4.131. Data retrieval using EXISTS command

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.

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

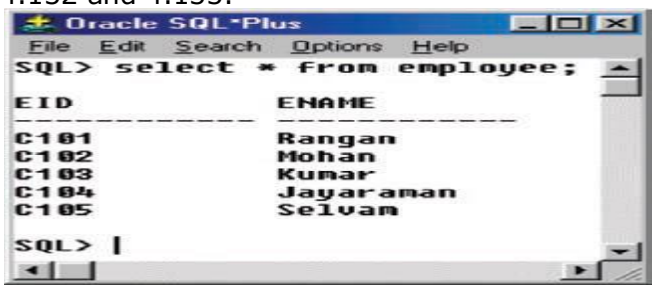


Fig. 4.132. EMPLOYEE table

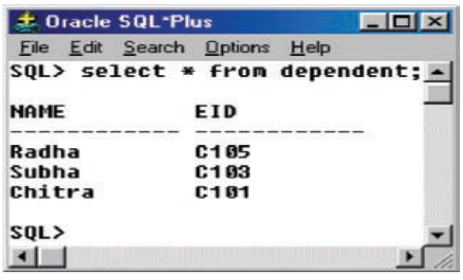


Fig. 4.133. DEPENDENT table

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.

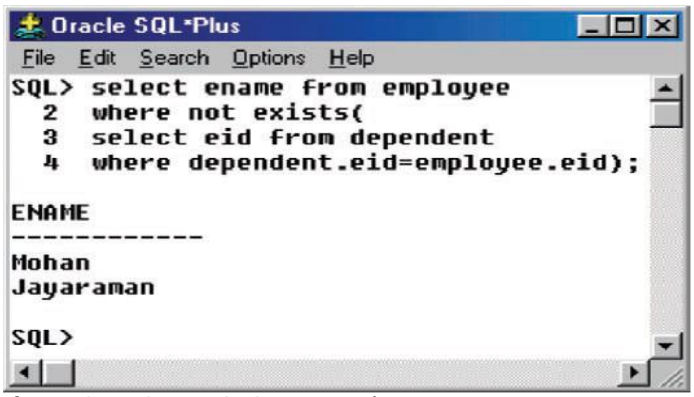


Fig. 4.134. NOT EXISTS command

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.

Comparison Operator ANY

The ANY operator compares a value to each value returned by a subquery.

Here

<ANY means less than maximum

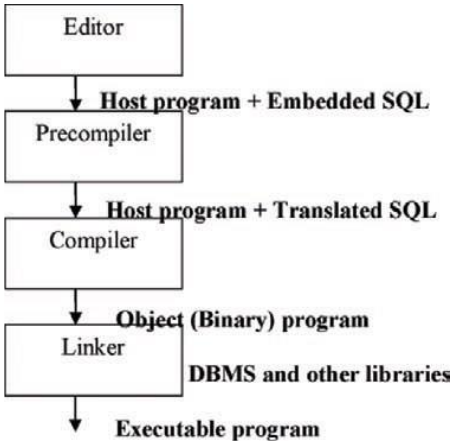
>ANY means more than the minimum

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.

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

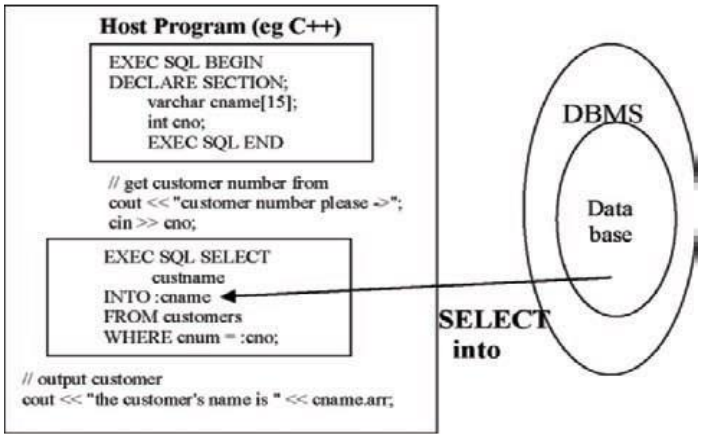
Before operations can be performed on the database, a valid connection has to be established. A model is shown below:
EXEC SQL connect :userid identified by :password;
– In all SQL statements, variables with the ":" prefix refer to shared host variables, as opposed to database variables.
– This assumes 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.

Questions:

1. Explain about various data types in sql;
2. Write about DDL commands
3. Discuss about DML commands with examples.
4. Explain about selection and projection operations.
5. List out various aggregate functions.
6. Write about various constraints.
7. Describe join and set operations.
8. Write short notes on views.
9. Explain about sub query.
10. Write about embedded sql.