ALAGAPPA UNIVERSITY

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle and Graded as Category–I University by MHRD-UGC]

(A State University Established by the Government of Tamil Nadu)

KARAIKUDI – 630 003

Directorate of Distance Education

Master of Computer Applications

II - Semester

315 24

RDBMS LAB

Author

Dr. Kavita Saini, *Associate Professor, School of Computer Science & Engineering, Galgotias University, Greater Noida.*

Vikas® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: 7361, Ravindra Mansion, Ram Nagar, New Delhi 110 055
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

Work Order No. AU/DDE/DE1-291/Preparation and Printing of Course Materials/2018 Dated 19.11.2018 Copies - 500

RDBMS LAB

Syllabi

BLOCK 1: TABLE MANIPULATION

1. Table Creation, Renaming a Table, Copying another Table, Dropping a Table
2. Table Description: Describing Table Definitions, Modifying Tables, Joining Tables, Number and Date Functions.

BLOCK 2: SQL QUERIES AND SUB QUERIES

3. SQL Queries: Queries, Sub Queries, and aggregate functions
4. DDL: Experiments using database DDL SQL statements
5. DML: Experiment using database DML SQL statements
6. DCL: Experiment using database DCL SQL statements

BLOCK 3: INDEX AND VIEW

7. Index : Experiment using database index creation, Renaming a index, Copying another index, Dropping a index
8. Views: Create Views, Partition and locks

BLOCK 4: EXCEPTION HANDLING AND PL/SQL

9. Exception Handling: PL/SQL Procedure for application using exception handling
10. Cursor: PL/SQL Procedure for application using cursors
11. Trigger: PL/SQL Procedure for application using triggers
12. Package: PL/SQL Procedure for application using package
13. Reports: DBMS programs to prepare report using functions

BLOCK 5 : APPLICATION DEVELOPMENT

14. Design and Develop Application: Library information system, Students mark sheet processing, Telephone directory maintenance, Gas booking and delivering, Electricity bill processing, Bank Transaction, Pay roll processing. Personal information system, Question database and conducting Quiz and Personal diary

INTRODUCTION

NOTES

Rapid globalization coupled with the growth of the Internet and information technology has led to a complete transformation in the way organizations function today. Organizations require those information systems that would provide them a 'competitive strength' by handling online operations, controlling operational and transactional applications, and implementing the management control tools. All this demands the Relational Database Management System or RDBMS which can serve both the decision support and the transaction processing requirements. Technically, the present RDBMS handles the distributed heterogeneous data sources, software environments and hardware platforms. Precisely, RDBMS is a Database Management System or DBMS that is based on the relational model introduced by E. F. Codd.

The most widely used commercial and open source databases are based on the relational model. Characteristically, a RDBMS is a DBMS in which data is stored in tables and the relationships among the data are also stored in tables. This stored data can be accessed or reassembled in many different ways without having to change the table forms. RDBMS program lets you create, update and manage a relational database. In spite of repeated challenges by competing technologies, as well as the claim by some experts that no current RDBMS has fully implemented relational principles, the majority of new corporate databases are still being created and managed with an RDBMS. So, understanding RDBMS through lab manuals has become extremely important.

This Lab Manual is intended for the students of MCA in the subject of RDBMS. This manual typically contains practical/Lab Sessions related to RDBMS, covering various aspects related to the subject to enhanced understanding. Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the textbooks.

NOTES

A **Database Management System** (RDBMS) is a collection of database and stored procedures. A RDBMS enables you to store, extract and manage important information from a database. It is software that is used to maintain data security and data integrity in a structured database.

As mentioned earlier in section RDBMS helps in maintaining and retrieving data in different form. There are various tools available for RDBMS such as Oracle, INGRES, Sybase, Microsoft SQL Server, MS-Access, IBM-DB-II, and My SQL.

Application of DBMS in various fields

In day to day life, various applications are in use. Few of the application are given below where database is used:

- Banking: For account holder information, amount with draw and deposit, load and other transactions.
- Airlines: For reservations, cancelation, fare detail and airline schedules.
- Universities: For student registration, examination, fee detail, course detail and other information.
- Manufacturing: For inventory, production, sale and purchase orders
- Human Resources: Employee records, salaries, tax deductions, allowances
- Multimedia application
- Real Time Application
- Graphical Information System (GIS)

Introduction to Oracle

Oracle is a secure portable and powerful database management system of Oracle Corporation. Oracle Corporation is an American multinational computer technology corporation headquartered in Redwood Shores, California. Oracle Database is compatible and connectable with almost all operating systems and machine. It is based on relational data model and a non-procedural language called structure query language (SQL). It is a tool that supports storing managing and organization the data.

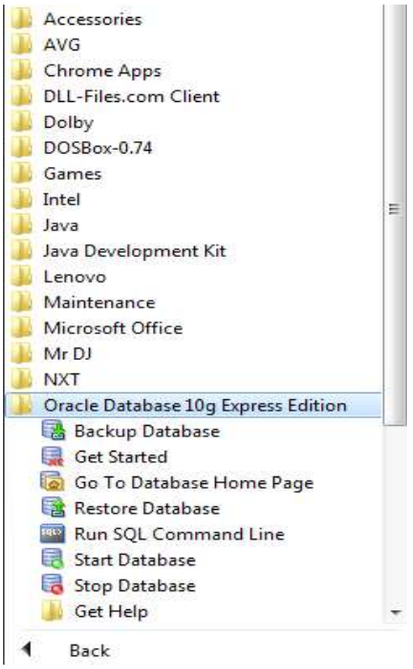
Getting Started with SQL:

To work with SQL *Plus Oracle should to be installed on computer system. The following steps are required to follow to invoke SQL plus:

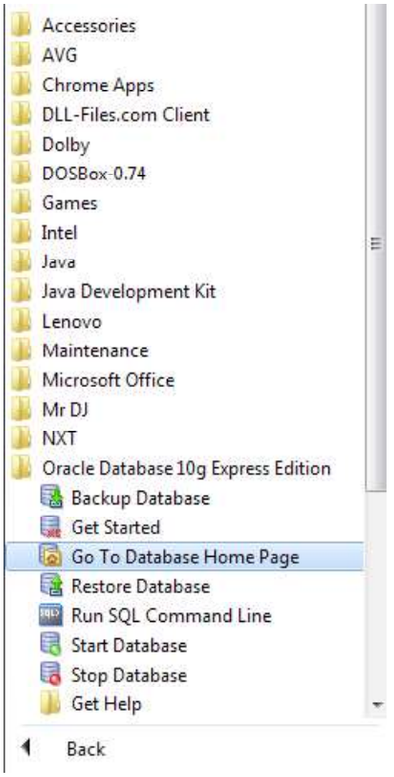
1. Click on Start button
2. Click on All Programs
3. Click on Oracle Database 10g Express Edition
4. Click on Go to Database Home Page

NOTES

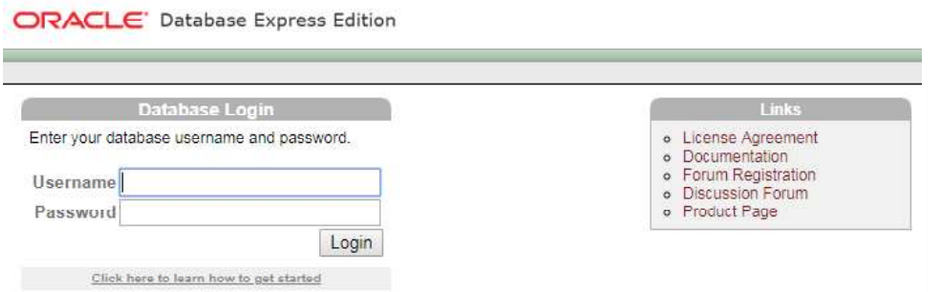
See the screenshots given below.



Click on **Go to Database Home Page**



The following Screen will appear:

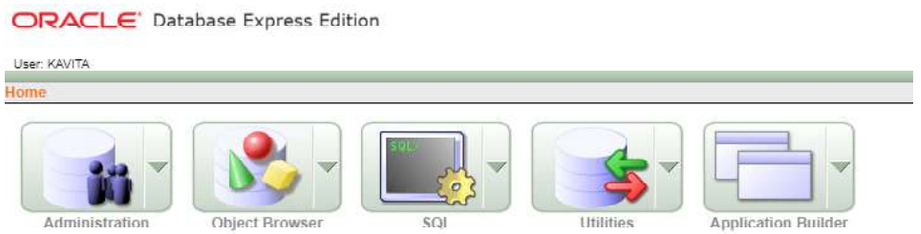


NOTES

Note:

1. **Enter the User Name and password** (Consult to your Lab Instructor for user name and password).
2. Click on "Login" button.

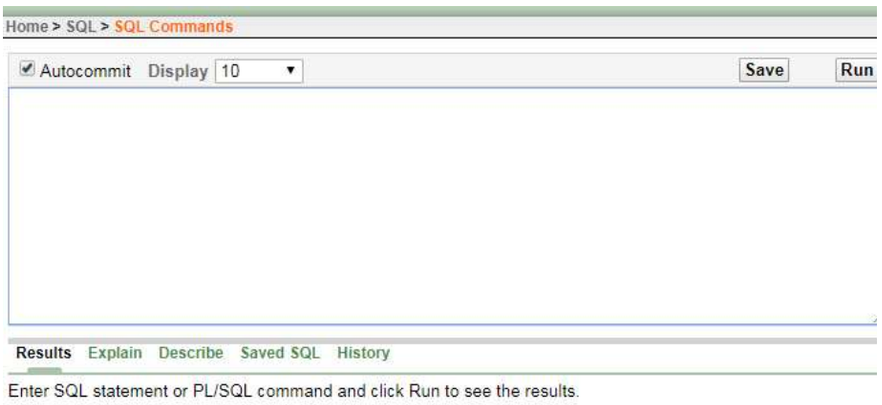
   The following screen will appear. Click on SQL.



After clicking on SQL, following screen will appear. Click on SQL Command.

NOTES

After clicking on **SQL Command,** following command screen will appear, where we can type and run all SQL commands:



Data Types in Oracle

When you define any table, it is required to specify the data type of fields. The main categories of data types are:

Data Type	Size
Char (size)	Maximum size of 2000 bytes
Varchar2 (size)	Maximum size of 4000 bytes
Long	Maximum size of 2GB
Raw (size)	Maximum size of 2000 bytes
long raw (size)	Maximum size of 2GB
Number(p,s)	Precision can range from 1 to 38. Scale can range from -84 to 127.
Date	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.

Operators in Oracle

Operators are the special characters that manipulate data items to produce some result. These data items are called *operands*. Operators are classified into two categories:

1. Unary Operators
2. Binary Operators

1. Unary Operators

A unary operator operates only on one operand. A unary operator is used as shown below:

Syntax:

Operator operand

2. Binary Operators

A binary operator operates on two operands. A binary operator is used as shown below:

Syntax:

Operand1 operator operand2

There are various types of operators to cater different purpose which includes:

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Set Operators
- Concatenates Operator

NOTES

Creating a Table

DDL (Data Definition Language) is the subset of SQL commands used to modify, create or remove ORACLE database objects, including tables.it is used to define the structure of a table. In a table structure you define various fields, their data types and constraints as per the requirement.

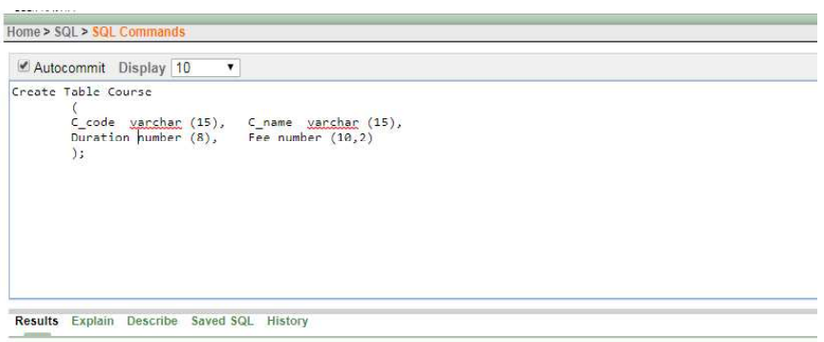
***Syntax*:**

```
Create table <table_name >
   (column_name data type(size), column_name data
type(size),.....);
```

Example 1: Create a table *Course* with the fields, data types and constraints as shown below.

Column Name	Data Type	Size
c_code	varchar2	15
c_name	varchar2	15
duration	number	8
free	number	10,2

The window below shows the query for creating the table as specified and Oracle will prompt a message.



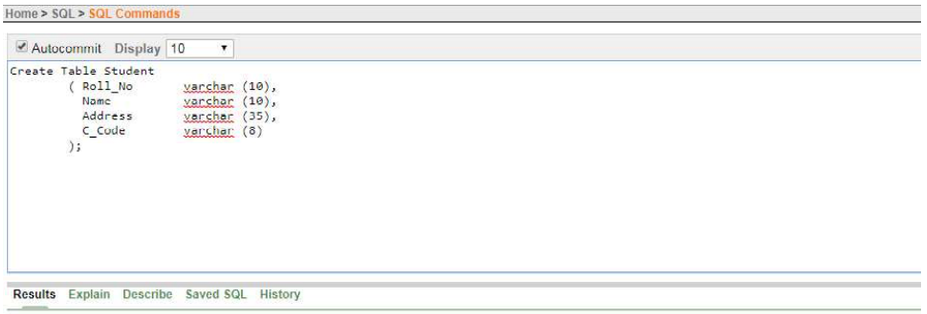

```
Table created.
```

NOTES

Example 2: Create a table *Student* with the fields, data types and constraints as shown below.

Column Name	Data Type	Size
Roll_No	Varchar	10
Name	Varchar	10
Address	Varchar	35
C_Code	Varchar	8

The window below shows the query for creating the table as specified and Oracle will prompt a message.

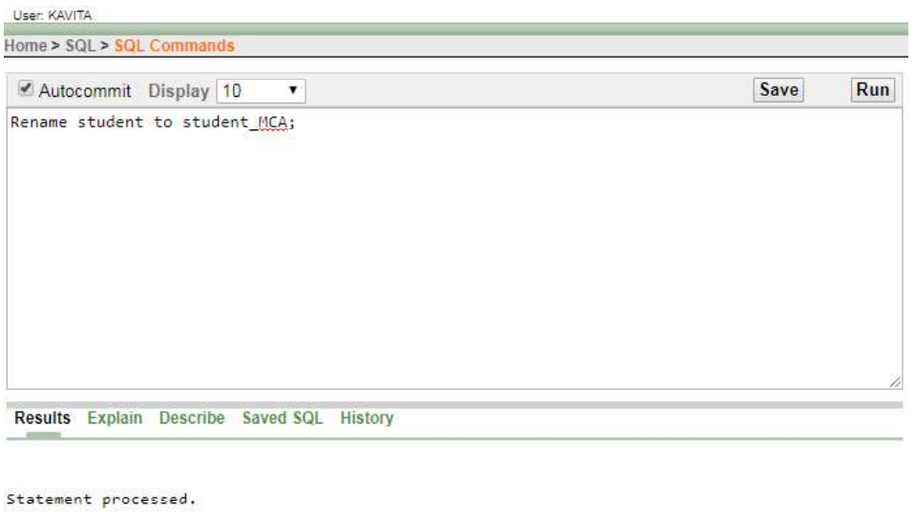


Rename Tables

The ***Syntax*** for renaming the table name is:

```
Rename old_table_name to new_table_name;
```

Example 3: Write a query to rename table *student* to *student_MCA*.



Dropping a Table

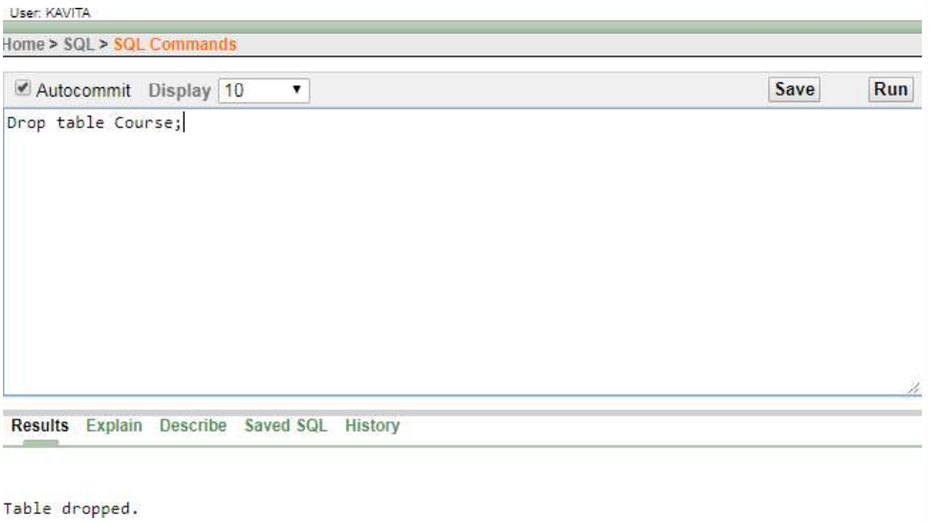
When a SQL table is no more required, you can delete it using DROP command. Drop command is used to drop any object such as table, index, view, package and function.

NOTES

Syntax:

```
Drop table <table_name >
```

Example 4: Write a query to drop table *course*.



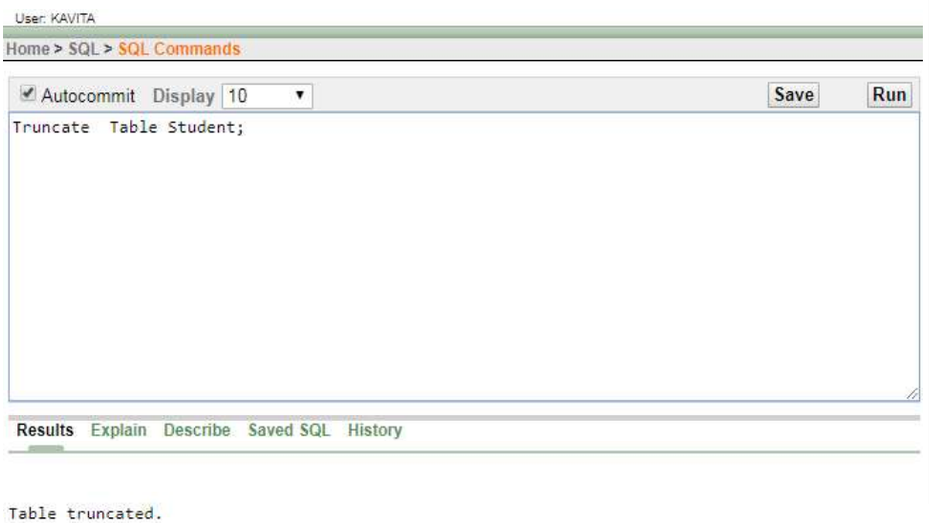
Truncate a Table

This command will remove all the records from a table. But structure will remain same.

Syntax:

```
Truncate Table <Table name>
```

Example 5: Write a query to truncate table *Student*.



Describe the Table

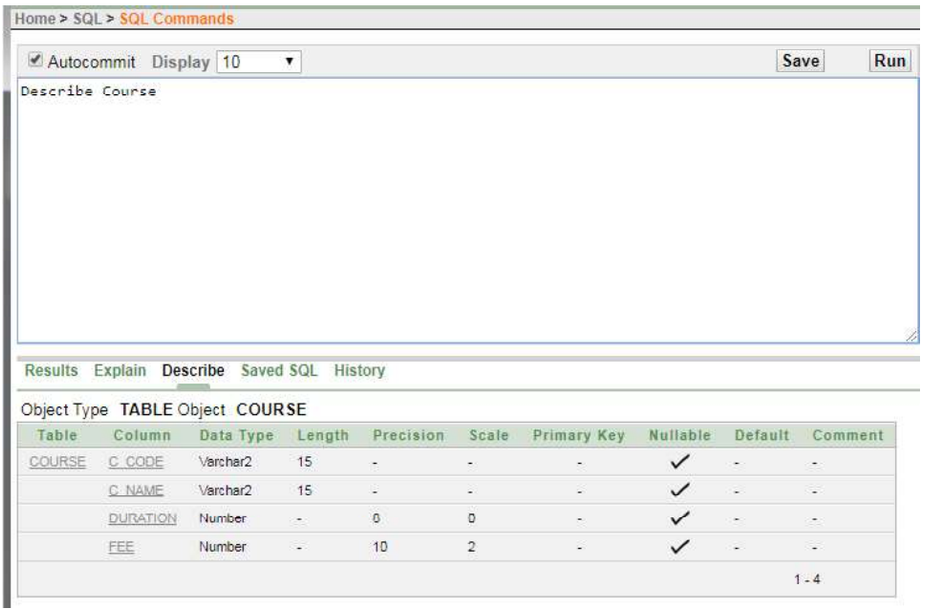
Describe command is used to describe the structure of a table created in the database.

NOTES

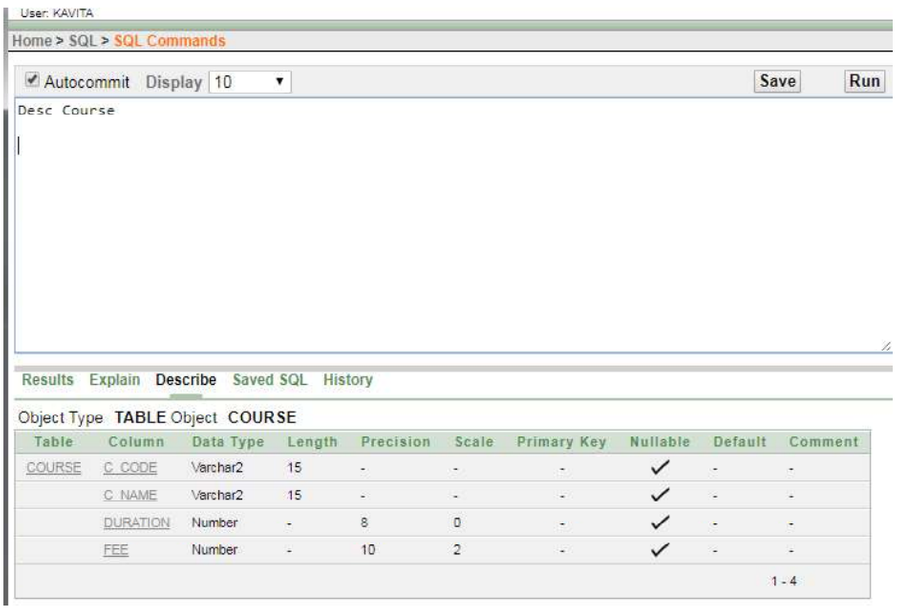
Syntax:

```
Describe <table_name>
      Or
Desc <table_name>
```

Example 6: Write a query to see the structure of *course* table.



Or



Modifying a Table

SQL provides an ALTER command to modify a table structure. It is a Data Definition Language (DDL) command. Following are the few examples to modify a table structure.

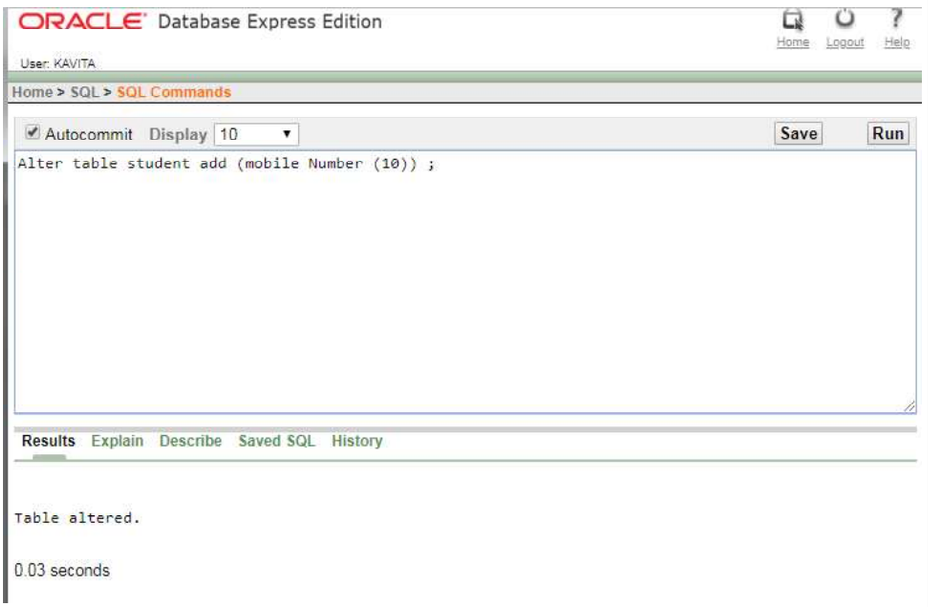
NOTES

Add a New Column

Syntax:

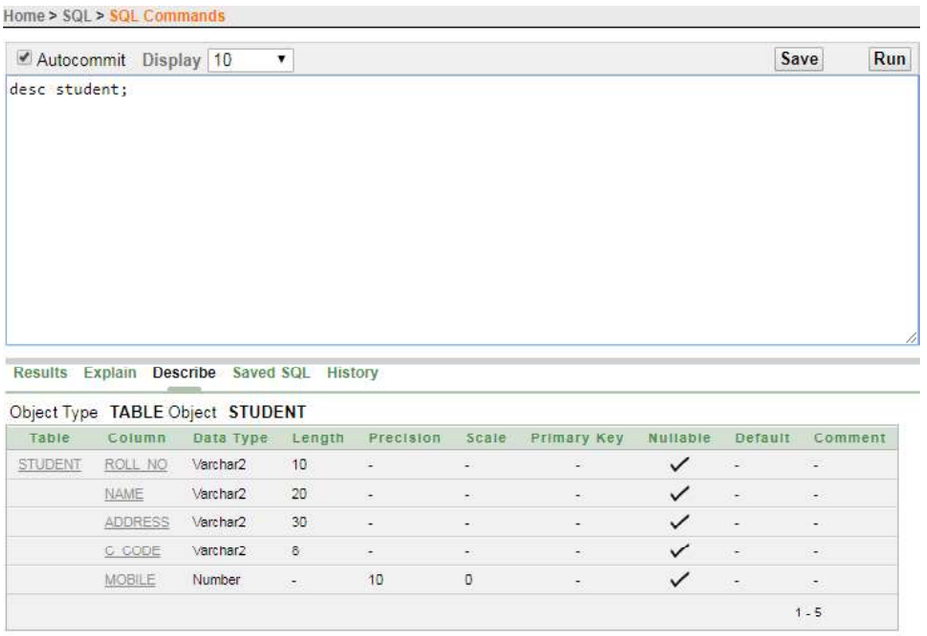
```
Alter table <table_name >
ADD (column_name data type(length), column_name data
type(length), …);
```

Example 7: Write a query to add new column (mobile Number (10)) in table *student*.



NOTES

You can see the new structure of *student* table as shown below.

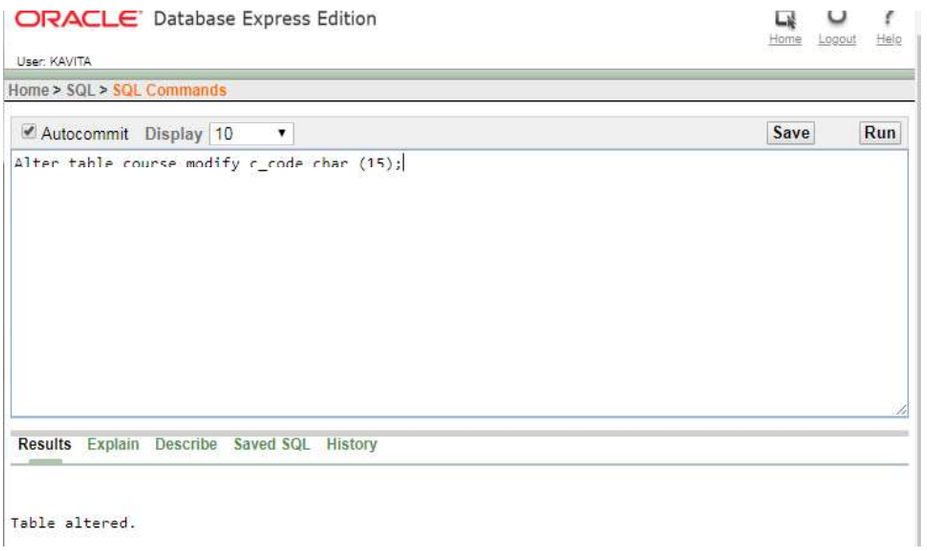


Change Data Type of an Existing Column

Syntax:

```
Alter table <table_name> modify (column data type (length),
column data type (length),…);
```

Example 8: Write a query to change the data type of column c_code from varchar to char (15).



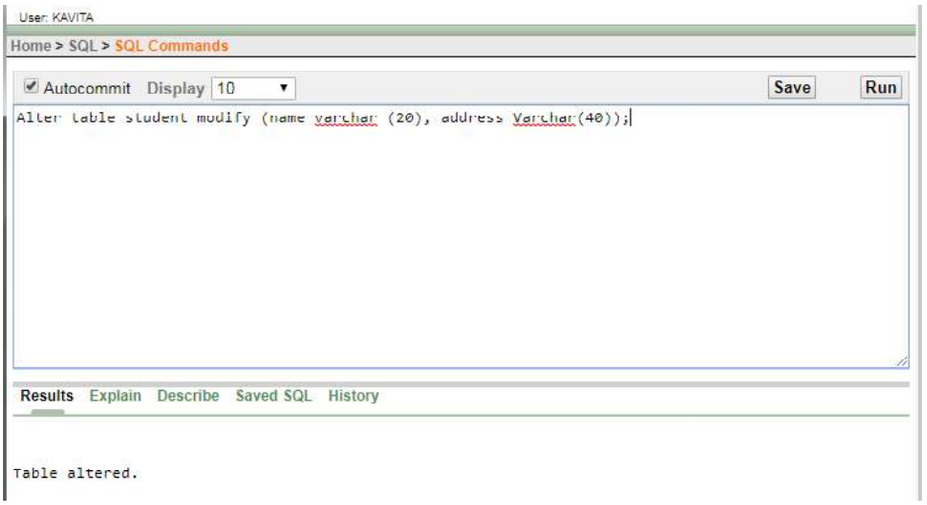
Modify the Length of an Existing Column

Syntax:

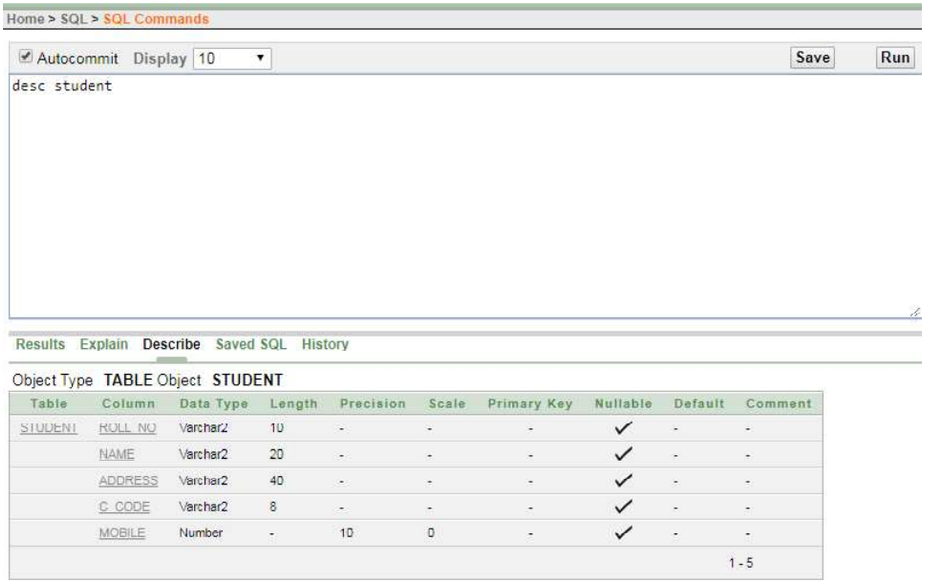
```
Alter table <table_name> modify (column data type (length),
column data type (length),…);
```

NOTES

Example 9: Write a query to change the length of columns **name varchar (20), address Varchar(40)** in table *student***.**



After altering student table structure will look like as shown below:



Important points to Remember

- If table column contains the values, then the length of column could be increase.

- To change the data type column should be empty.
- To decrease the size of data type column should be empty.

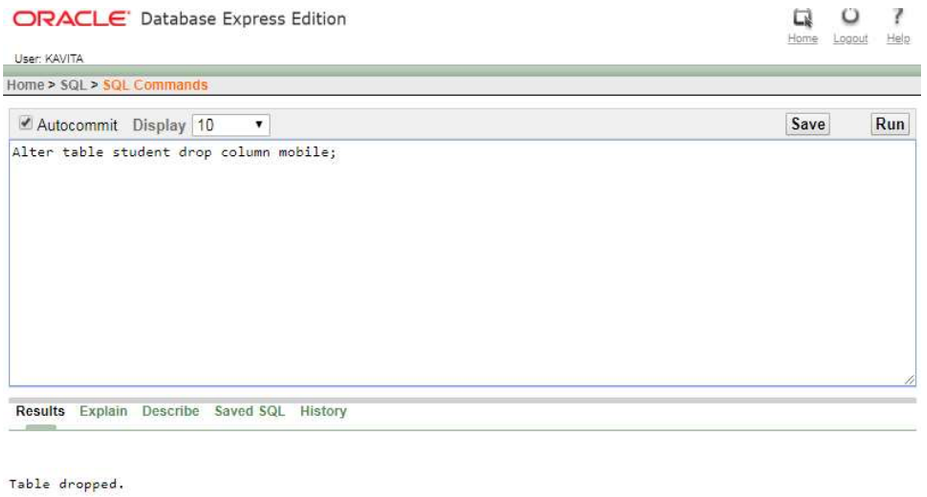
Delete any Column

NOTES

Syntax:

```
Alter table <table name> drop column column_name;
```

Example 10: Write a query to drop column mobile in table *student*.



Data Constraints

It is very important that whatever you store into your tables is as per the need of your organization. No false or incorrect data is stored by the user even intentionally or accidentally. Constraints are the restriction that you could put on your data to maintain data integrity. For example employer's salary should not be negative value, two students should not have the same enrollment number etc. The constraints helps in maintaining data integrity. Constraints could be specified when a table is created or even after the table is created with the ALTER TABLE command.

Oracle provides various types constraints as listed below:

- Primary Key
- Foreign Key or Reference Key
- Not Null
- Unique
- Check
- Default

Constraint could be defined at column level or at the table level. The only difference between these two is the syntax of these two.

Note: *Drop all table created previously in this manual.*

NOTES

Not Null constraint

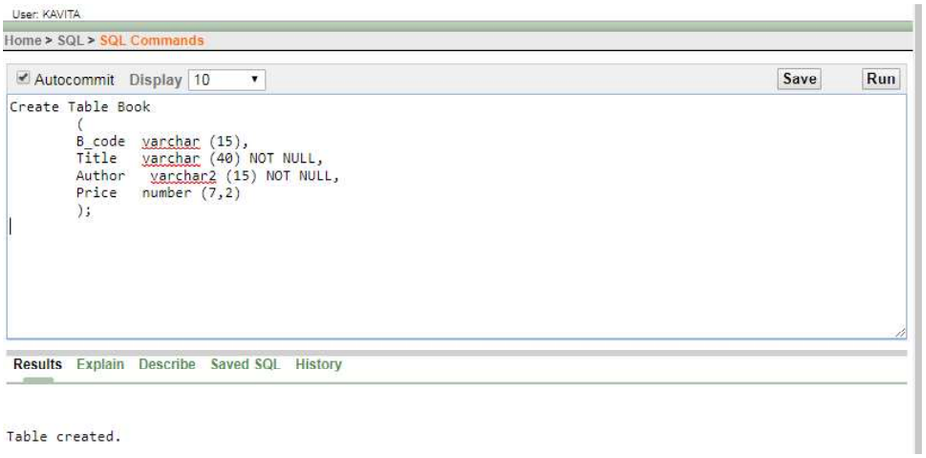
In database, NULL is a special value that is different from zero, space or blank. It represents an unknown value for the column. The NOT NULL constraint ensures that the value in column is not missing (NULL). This constraint enforce user to enter data into a specified column. A column with this constraint could have duplicate values but could not be NULL or empty.

You must have created your e-mail ID. When you create an e-mail ID, it is mandatory to fill certain entries (the field with *), these fields are the fields with the not null constraint.

Example 11: Create a table book with the NOT NULL constraint with the structure as shown below.

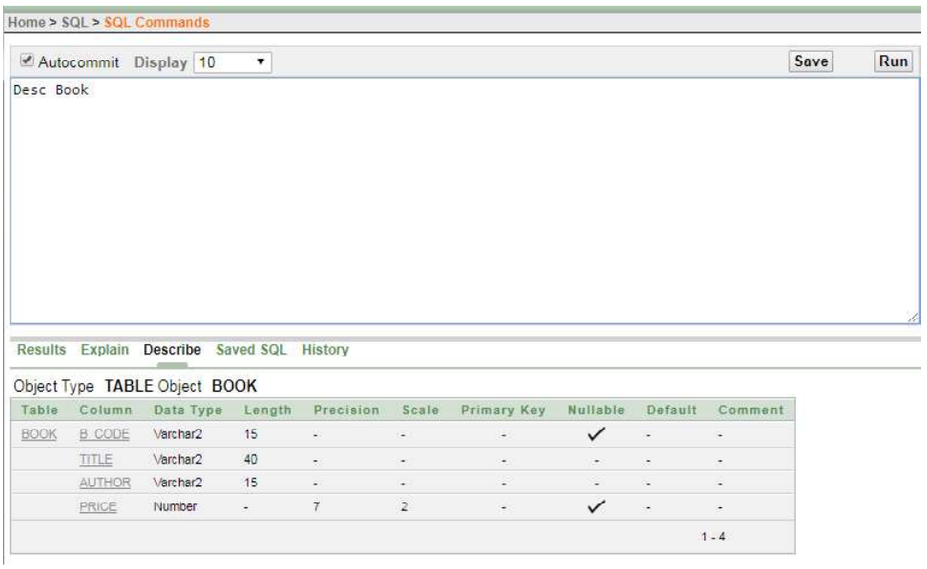
Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	
Title	varchar2	40	NOT NULL
Author	varchar2	15	NOT NULL
Price	number	7,2	

The SQL command to create table with NOT NULL constraint is given in window shown below.



NOTES

The structure of table *book* is given below.



The above SQL command will create a table book where Title and Author have NOT NULL constraints. These constraints would make it sure that both the columns have some values during inserting and updating of data to these columns.

Note: *NOT NULL constraints can be set at column level only.*

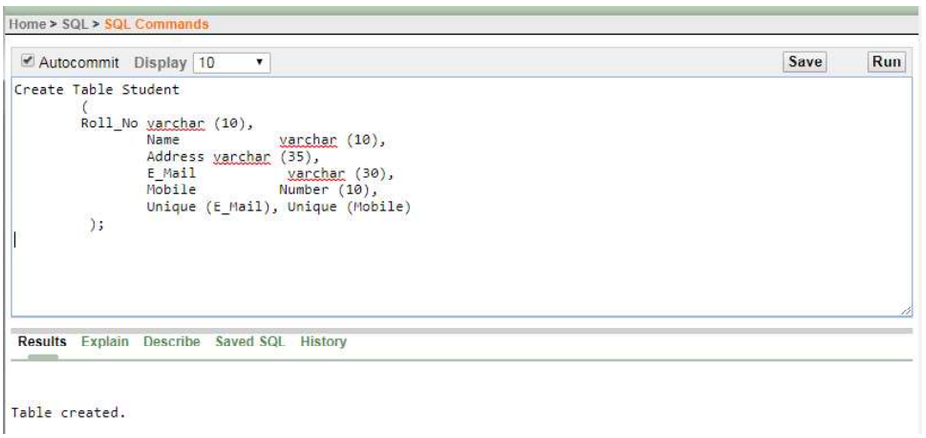
Unique Constraint

Sometimes, it is required that column must have unique values only. The unique constraint ensures that data to the specified column data is not duplicate but it could contain the NULL values. Let us take an example of contact number and e-mail ID; it is not necessary that every student has a contact number and an e-mail ID, if they have that will be unique only.

Example 12: Create a table *student* with the UNIQUE constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	
Name	Varchar	10	
Address	Varchar	35	
E_Mail	Varchar	20	Unique
Mobile	Number	10	Unique

The SQL command to create table with Unique constraint is given in window shown below.



NOTES

Primary Key Constraint

A primary key constraint is used to uniquely identify each and every record in a table. A primary key has properties of unique and not null constraints.

A primary key constraint has the following properties:

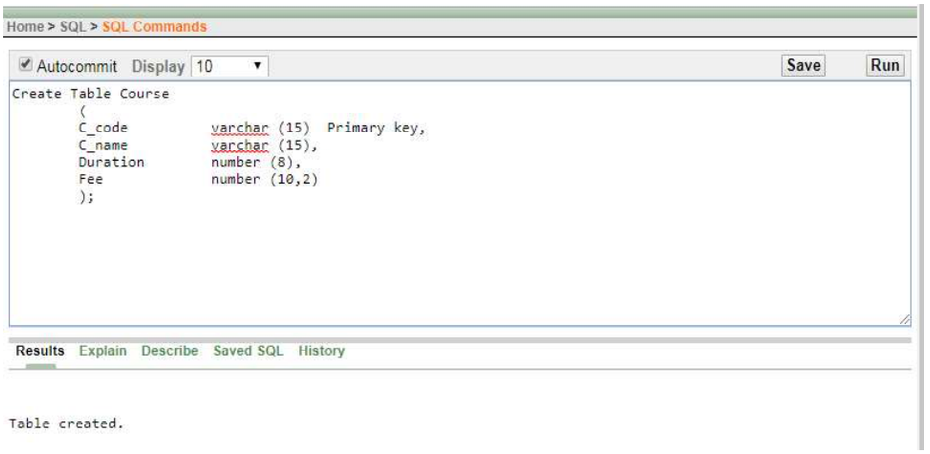
- A primary key column allows unique values only.
- It does not allow NULL value in column.
- A primary key column could be used for a reference in another table (child table).

Example 13: Create a table *course* having the Primary Key constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
c_code	varchar2	15	Primary Key
c_name	varchar2	15	
duration	number	8	
fee	number	10,2	

NOTES

The SQL command to create table with Primary Key constraint is given in window shown below.

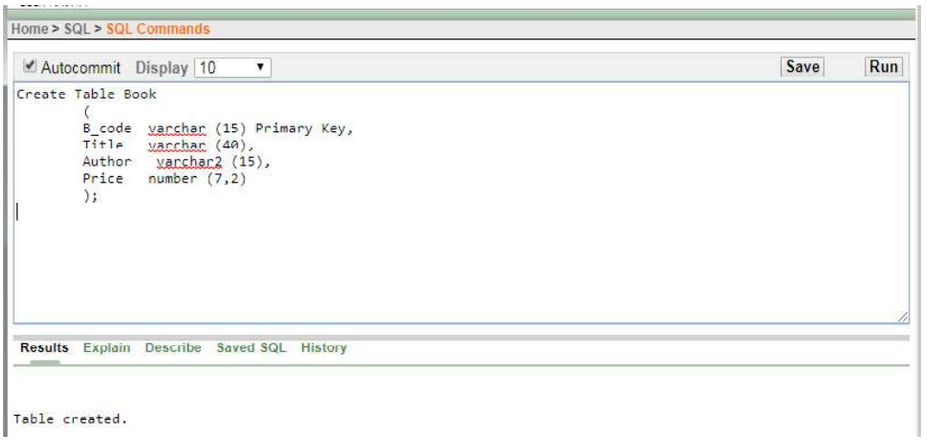


The query in window will create table course which contains a primary key field course code. Here primary key constraint will enforce the end user to enter unique and not null values only.

Example 14: Create a table book with the Primary Key constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	Primary Key
Title	varchar2	40	
Author	varchar2	15	
Price	number	7,2	

The SQL command to create table with Primary Key constraint is as follows:



Note: *A table can have only one primary key.*

Foreign Key Constraint or Reference Key Constraint

NOTES

A foreign key column in a table derived values from a primary key of another table that helps in establishing relationship between tables.

A table having primary key column is called a Master Table or a parent table and a table with the reference key is known as a Transaction Table or a child table.

C_code and B_code are the primary key of the tables *course* and *book* respectively. These columns can be used to as a reference key in another table.

Important Points to Remember

- Reference key column in a table must have the same data type be as specified in primary key column in another table.
- Size of data type must be the same or more as defined in a primary key column.
- Name of reference key column could be same or different as defined in primary key column.
- A table may contain more than one reference keys.
- Reference keys column values could be duplicate or not NULL.
- Reference keys column can have the same values as stored in primary key column.

Suppose that students can enrolled in the course which are offered by the university. Course table contains the detail of all the courses offered by the university, so C_code column in *student* table must have reference of C_code column of *course* table.

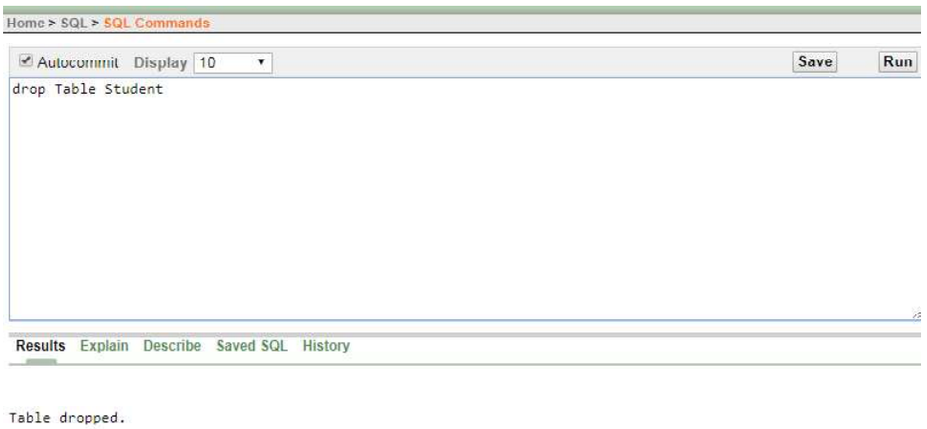
Example 15: Create a table student with the Reference Key constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	
Name	Varchar	10	
Address	Varchar	35	
C_code	Varchar	15	Reference Key

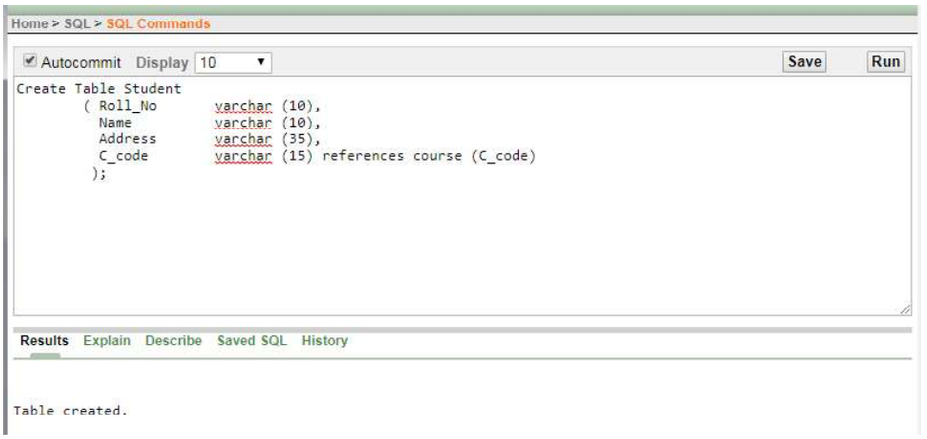
The SQL command to create table with REFERENCE KEY constraint is as follows:

NOTES

Note: drop student table



Now create Student table again with reference key as shown below:



The above command will create table *student* which contains a reference key column course code. This column will create reference of course code of *course* table, when record in *student* table will be inserted or updated by the user.

Note: *A table can have more than one reference keys.*

Check Constraint

A check constraint enforce user to enter data as specified condition. For example marks in any subject should be between the ranges 0 to 100, fee should not be negative, book code must start with 'B', and book price should be between the ranges 1 to 15000 and employee HRA could not be more than 40% of basic salary and so on.

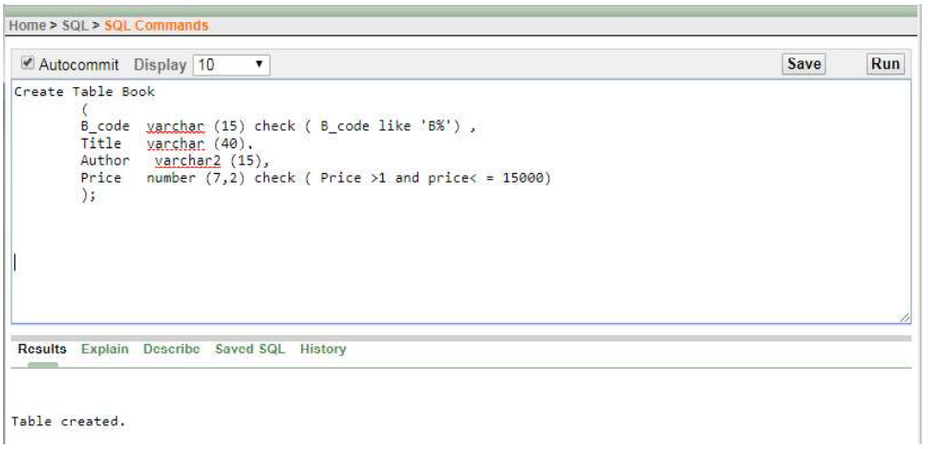
NOTES

Example 16: Create a table book with the Check constraint with the structure as shown below:

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	Check
Title	varchar2	40	
Author	varchar2	15	
Price	number	7,2	Check

Note: *drop table book created earlier.*

The SQL command to create table with Check constraint is given in window shown below.



Default Constraint

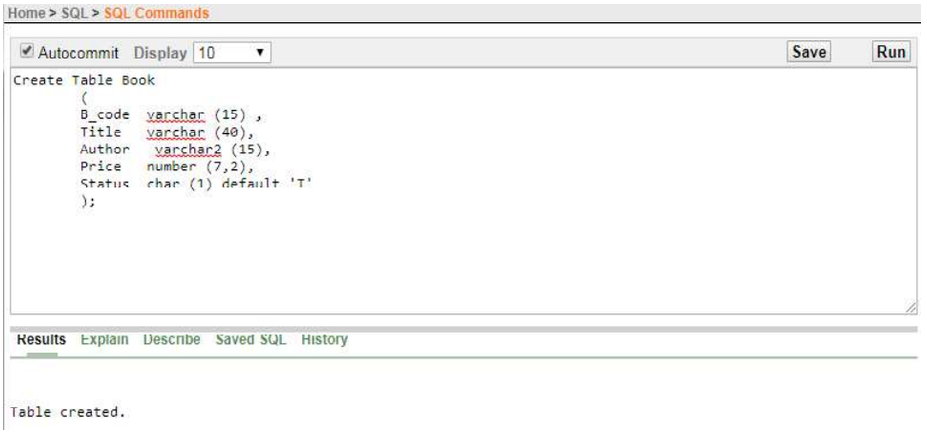
Sometimes, the value of any column for every new record is same. To maintain the status of book in a library either it is available to issue or not, you must keep the status of book as 'T' (available) or 'F' (Issued). Every new book purchased for library, the status of book is required to be 'T'. Default value concept is suitable for these types of situations.

Example 17: Create a table book with the Default constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	
Title	varchar2	40	
Author	varchar2	15	
Price	number	7,2	
Status	Char	1	Default

NOTES

The SQL command to create table with Default constraint is given in window shown below.

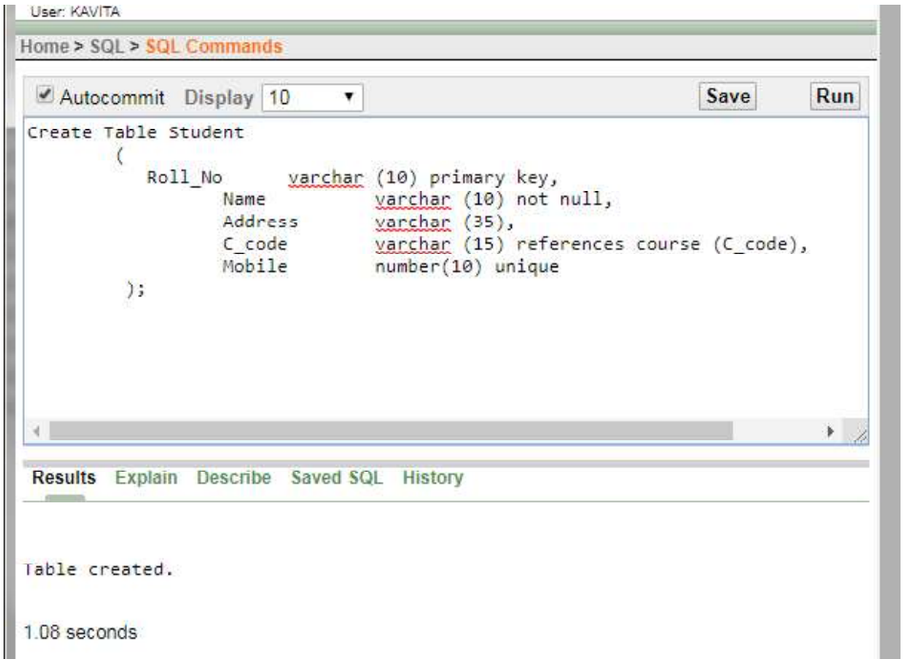


Example 18: Create a table student with multiple constraints having the structure as shown below:

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	Primary Key
Name	Varchar	10	Not Null
Address	Varchar	35	
C_code	Varchar	15	Reference Key
Mobile	Number	10	Unique

Note: *drop student table then create student table again*

The SQL command to create table as specified above is shown below:



Data Manipulation Language (DML)

Data Manipulation Language (DML) commands are used to insert, manipulate and access data. The data manipulation language statements are Insert, Delete, and Update.

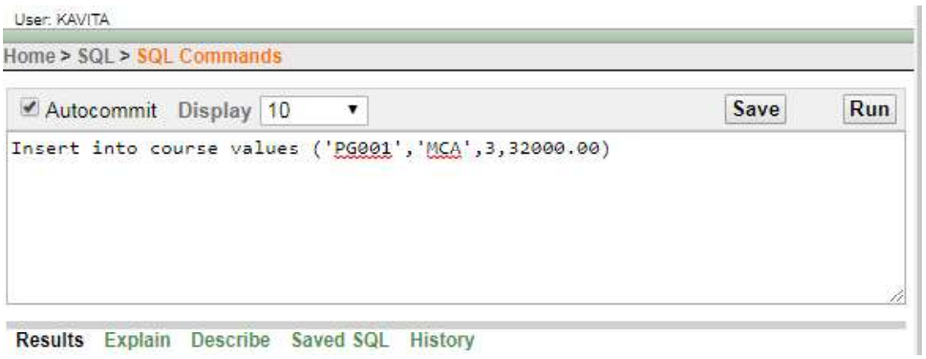
NOTES

Insert Records in a Table

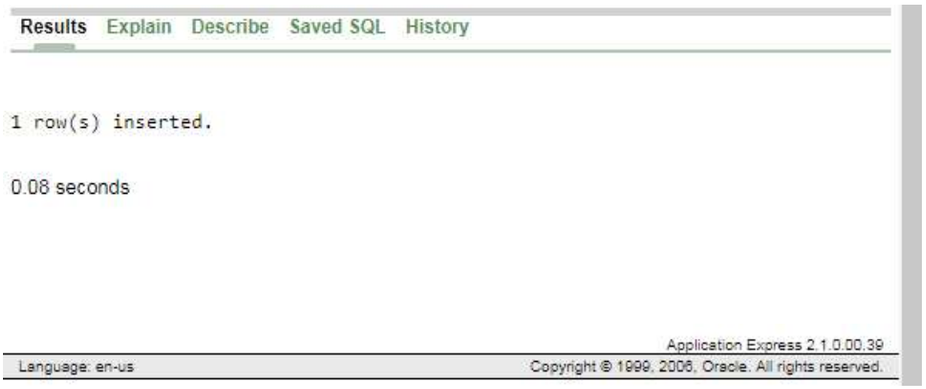
Syntax:

```
Insert into <table name> values (value1, value2, …);
```

Example 19: Insert (course code – PG001, course name- MCA, duration- 3, fee-32000) in the *course* table.



Output:



After executing the above command system will prompt a message **1 row inserted**.

Note: *All char, varchar and date values should be enclosed in single quotes (') for example 'MCA', '07-Sept-09', 'A-08-02', ...*

NOTES

Try yourself:

1. Insert into course values ('PG003','M Sc-IT',3,32000.00)
2. Insert into course values ('PG002','MBA',2,40000.00)
3. Insert into course values ('UG002','B SC-IT',3,25000.00)
4. Add five records in *course* table.
5. Create a new table *Book* with the following fields and data types.

Field Name	Data Type	Size
B_Code	varchar	15
Title	varchar	30
Author	varchar	15
Price	Number	6,2

6. View the structure of *Book* table.
7. Add five records in *Book* table.

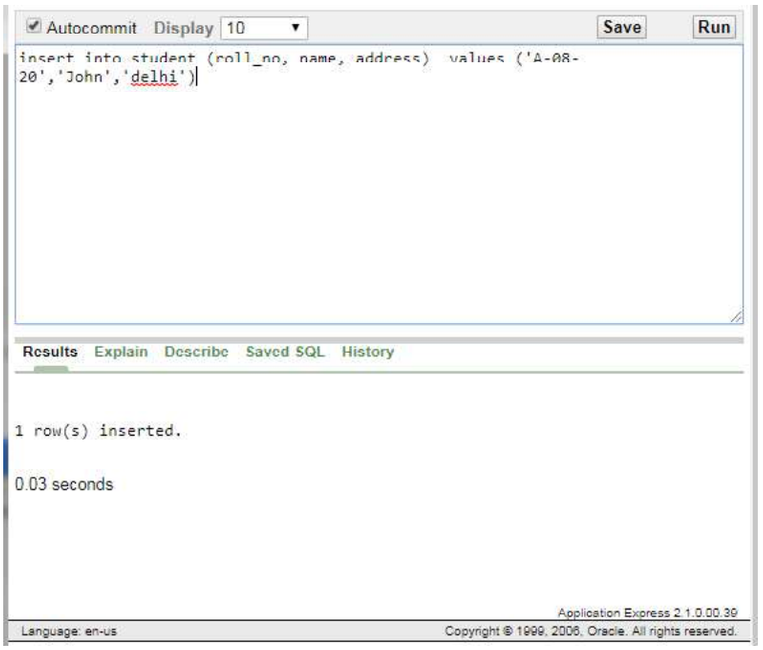
Insert Data into Specific Fields

In the insert command shown above, it is necessary to insert data in all the fields in the same sequence as defined in the table. But sometimes, few fields are required to update later on. For example, student's subjects marks are inserted in the table and total, percentage or grade is required to calculate later on.

Syntax: (to insert data into selected fields only)

```
Insert into <table name> (column1, column2, …)
values (value1, value2, …);
```

Example 20: Write a query to insert (roll_no= 'A-08-20', name='John', address= 'delhi') in the *student* table.

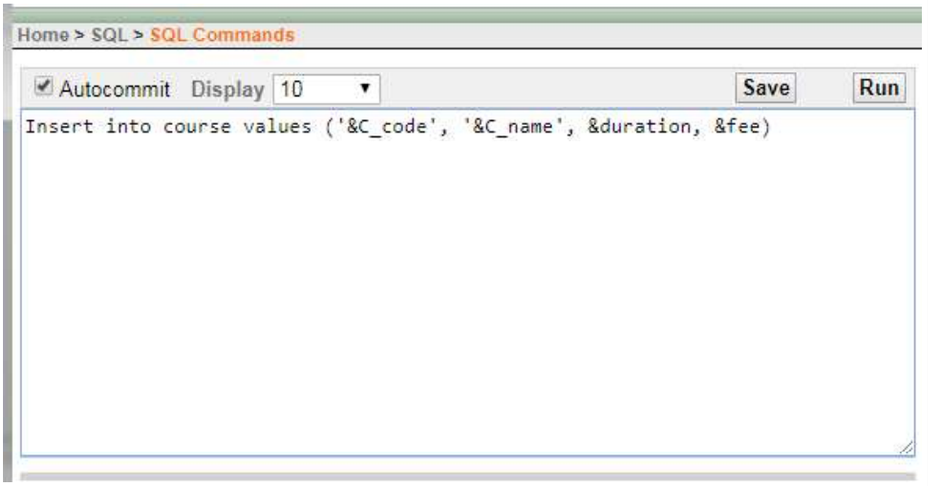


Insert Data with User Interaction

NOTES

If hundreds or thousands of records are to be inserted in a table, it will be very tedious job to do it with the constant values. The other ways to insert records into table is take input from the user and repeat the command.

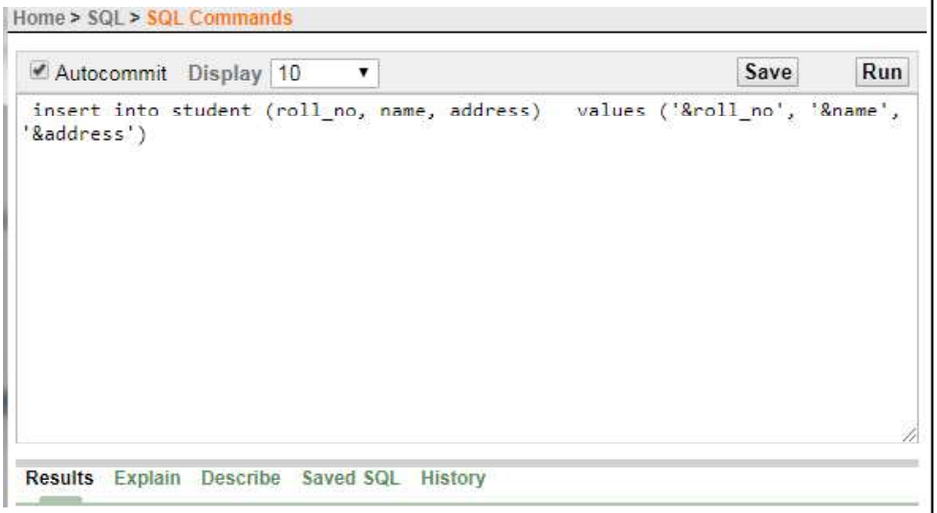
Example 21: Insert into course values ('&C_code', '&C_name', &duration, &fee);



The same command cab be repeated to insert more records by putting / and pressing enter key at SQL prompt.

You can also insert records interactively into specific fields as shown below.

Example 22: Insert into student (roll_no, name, address) values ('&roll_no', '&name', '&address');



Note: *The & symbol would prompt user to input data to the various variable. The variable name that is written after & is not required to the same as field names.*

NOTES

Try yourself:

1. Add the following data into C_code, C_name and duration fields of Course table.

C_code	C_name	Duration
UG001	BCA	3
UG002	B Sc-IT	3
PG003	M Sc-IT	2

2. Add 10 records into student table with the user interaction.
3. Add data into b_code, title, and author fields of book table with the user interaction.

Display Table Records

Select command is used to display the records in the table. All the fields and records could be displayed or only selective records and fields could be retrieved.

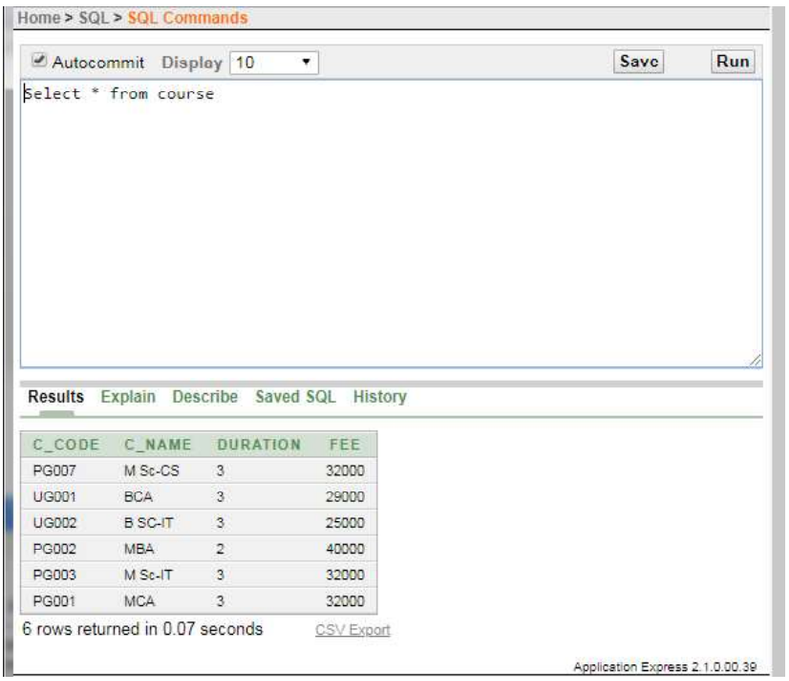
To view all the Records

To retrieve all the recoeds use "*" as shown below:

Syntax:

```
Select * from <table name>;
```

Example 23: Write a query to display all the records in the *course* table.



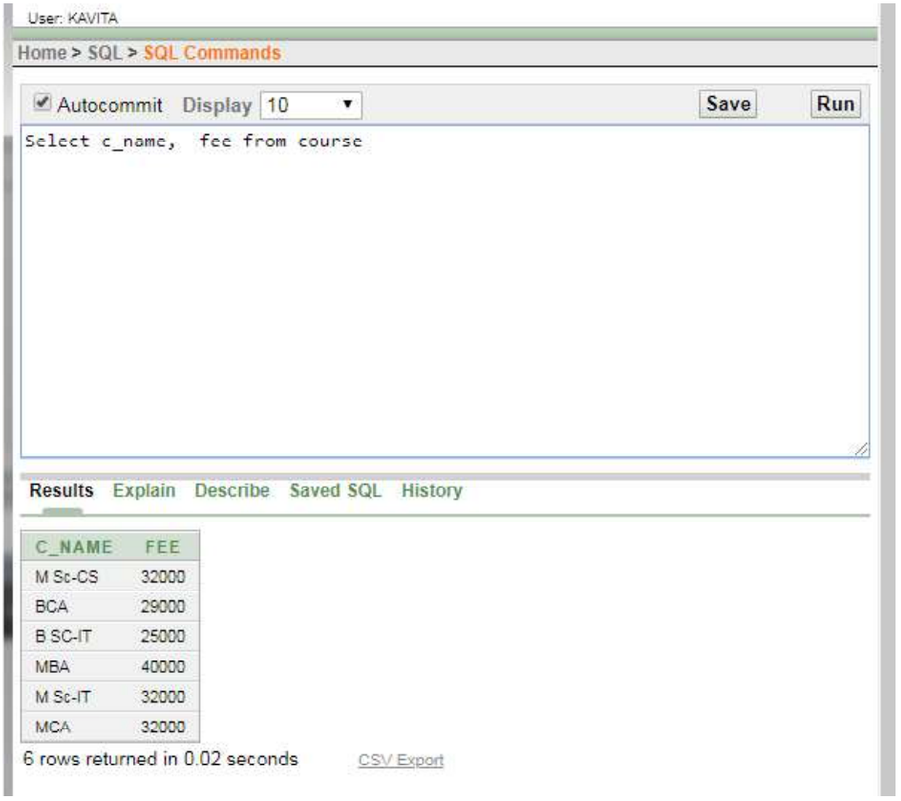
To View Selected Columns

To view only selective columns, enter column names separated by comma (,) as shown below:

Syntax:

```
Select field1, field2, ….from <table_names>;
```

Example 24: Write a query to display the column c_*name* and *fee* in the *course* table.



Update Table Records

Update command is used to change or update the records in a table. For example, the contact no. or address of any person has been changed or course fee is changed by the university.

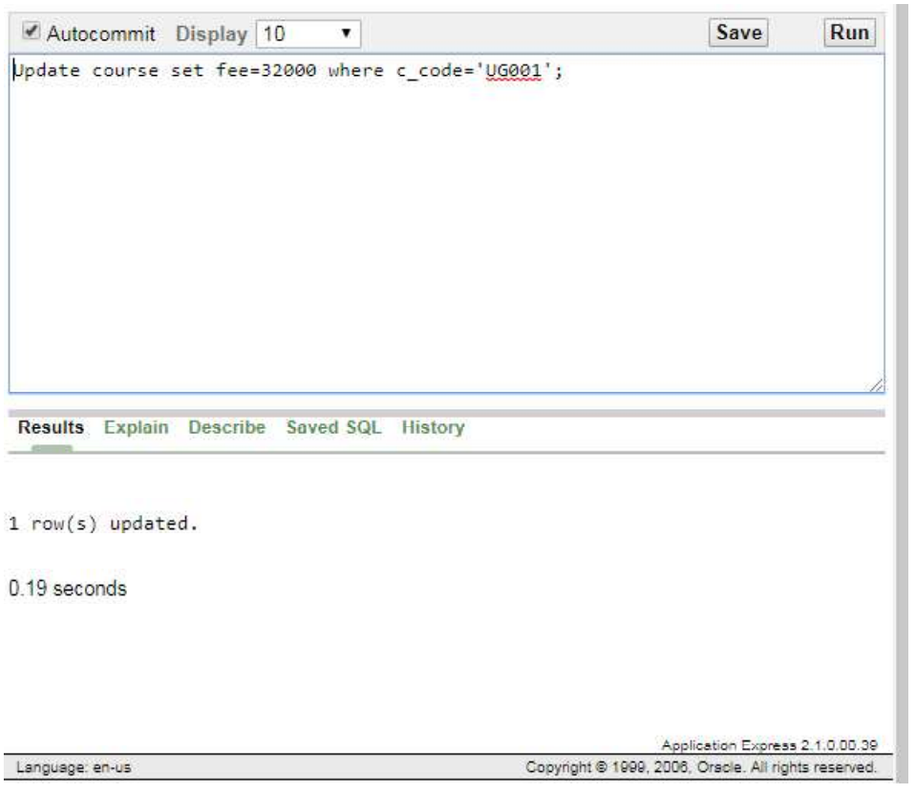
Syntax:

```
Update <table name>
Set <column_name1 = <new value>,
     <column_name2=<new value,
     …
    [where <condition>];
```

NOTES

NOTES

Example 25: Write a query to update fee=32000 having course code 'UG001' in the *course* table.

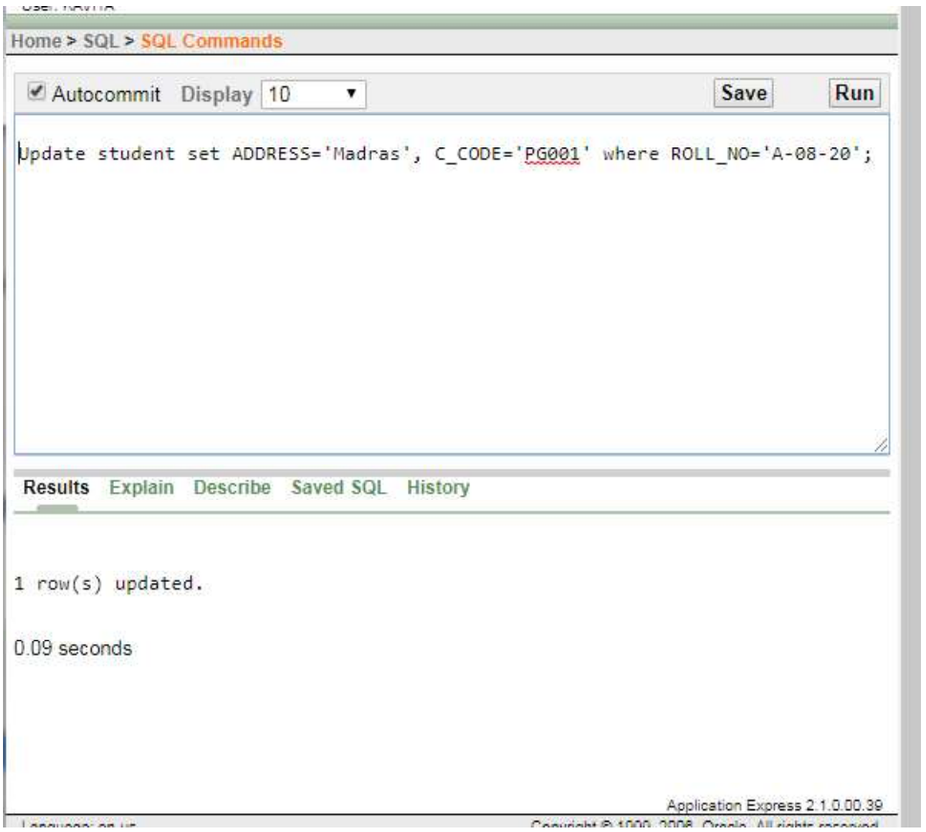


Where clause is used to specify the condition for which this fee should be changed. Without any condition all the records will be updated with the new fee Rs. 32,000.

More than one columns can also be updated by specifying multiple columns and there new values after set keyword.

NOTES

Example 26: Write a query to change address to madras and course code to 'PG001' having roll_no= 'A-08-20' in the *student* table.



Try yourself:

1. Display name and c_code of students.
2. Change the address from Madras to Delhi of student whose roll number is A-08-20.
3. Change the fee from Rs. 32000 to Rs. 38000 of course where c_code is PG001.

Delete Records

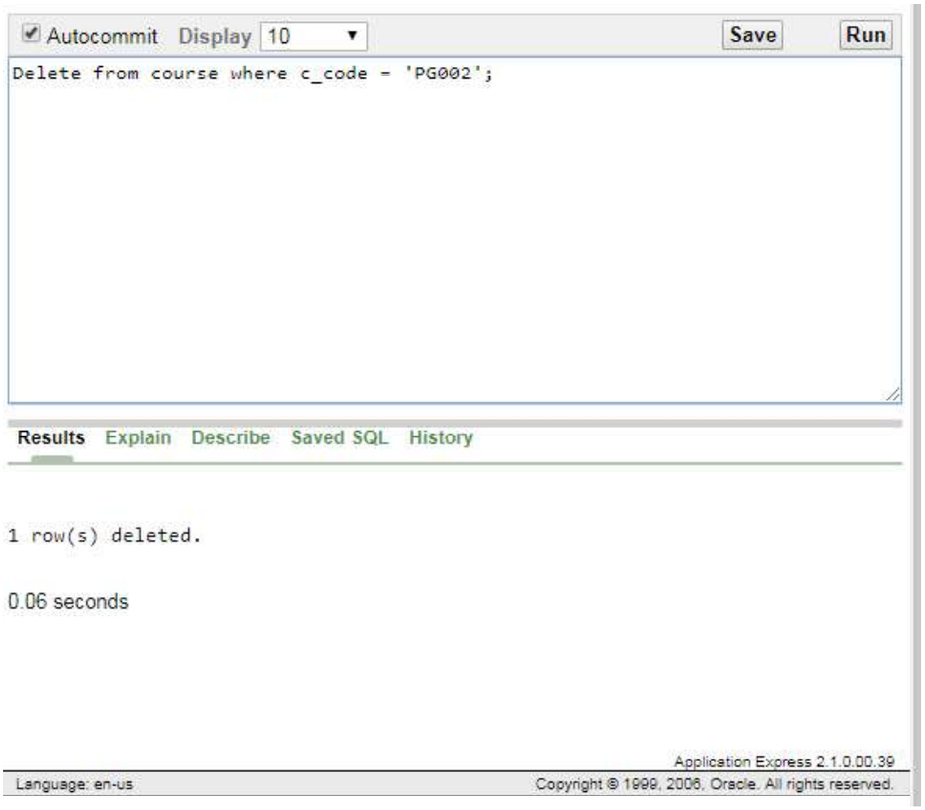
Delete command is used to delete records from the table. One or more or all the records can be deleted from the table depending upon the where condition.

Syntax:

```
Delete <table_name> [where <condition>];
      Or
Delete from <table_name> where <condition>];
```

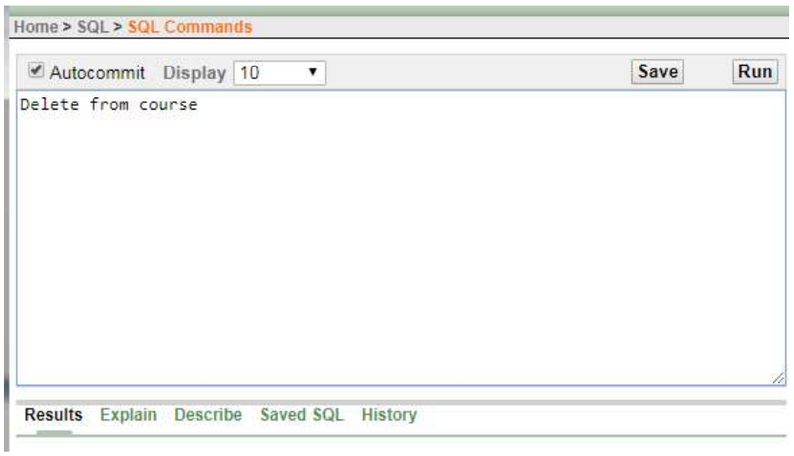
NOTES

Example 27: Write a query to delete a record from the *course* table where course code is 'PG002'.



To delete all the records from a table, you can write the delete command without where clause as given below:

```
Delete from course;
    Or
Delete course;
```



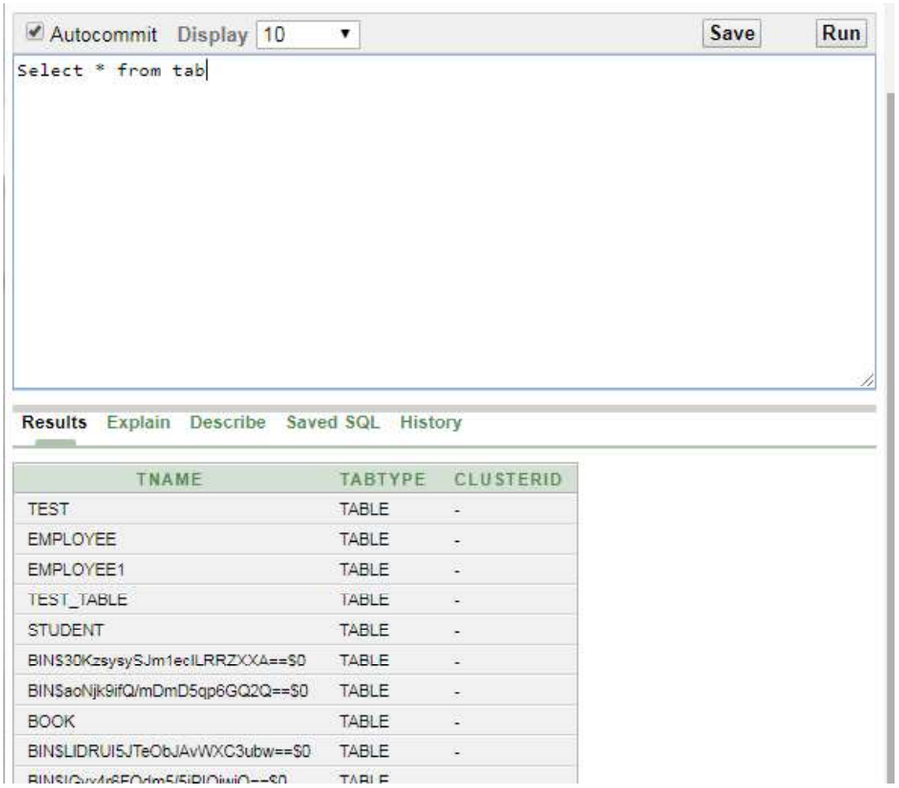
The above command will delete all the records from the course table.

View the Existing Tables

To view all the existing tables in database, you can use **Tab**. Tab is a view which displays the name and type of object such as table, view, or synonym.

Example 28: Write a query to display all the tables in the database.

NOTES



TNAME is a column which displays the object name as table, view, index, or synonym.

TABTYPE is a column which displays the type of object. The type of object may be any table, view, index, or synonym.

Filtering Records using Where Conditions

A university can have thousands of records but all these records are not required to view every time. Many users might need to view different records from the same table at different time. To filter various records of table, **where** clause can be used with conditional, logical and other operators.

Syntax:

```
Select * from <table name>    [where <condition>];
```

NOTES

The following is the *course* table contains 8 records. Let us filter records from this table with different conditions.

C_CODE	C_NAME	DURATION	FEE
PG001	MCA	3	55000
PG007	M Sc-CS	2	50000
UG001	BCA	3	32000
UG002	B Sc-IT	3	25000
PG003	M Sc-IT	2	48000
PG002	B Tech-CS	4	60000
PG004	B Tech-EC	4	64000
PG005	B Tech-IT	4	58000

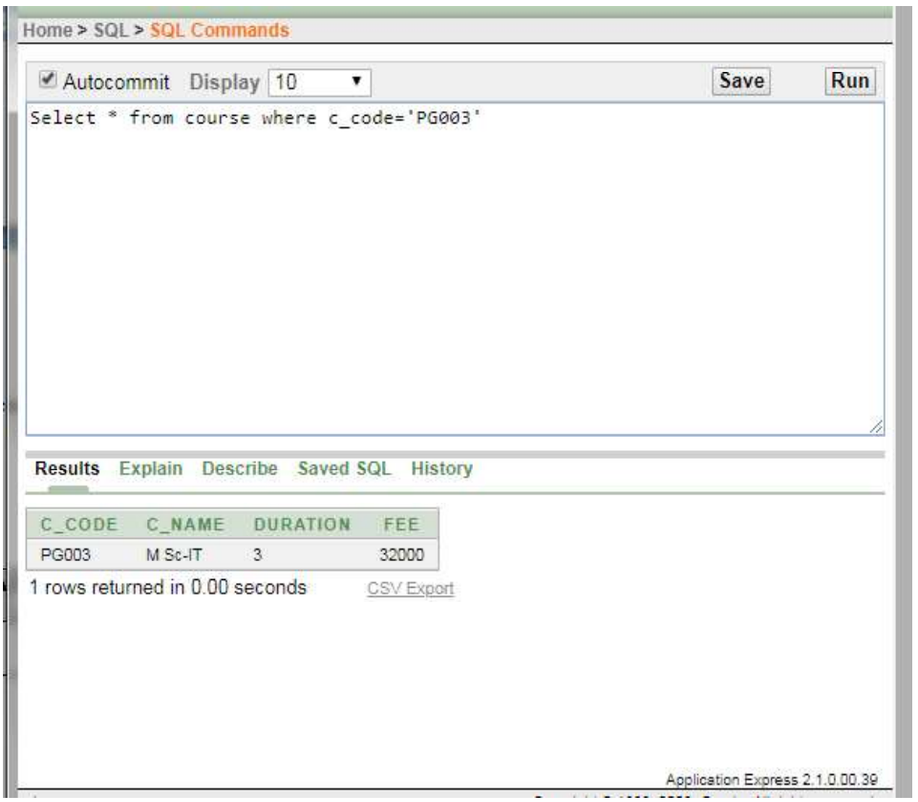
Conditional Operators in SQL

Equal to (=)

To see the detail of course where course code equal to PG003 then the query will be:

```
Select * from course where c_code='PG003';
```

Output of the above query is shown below:

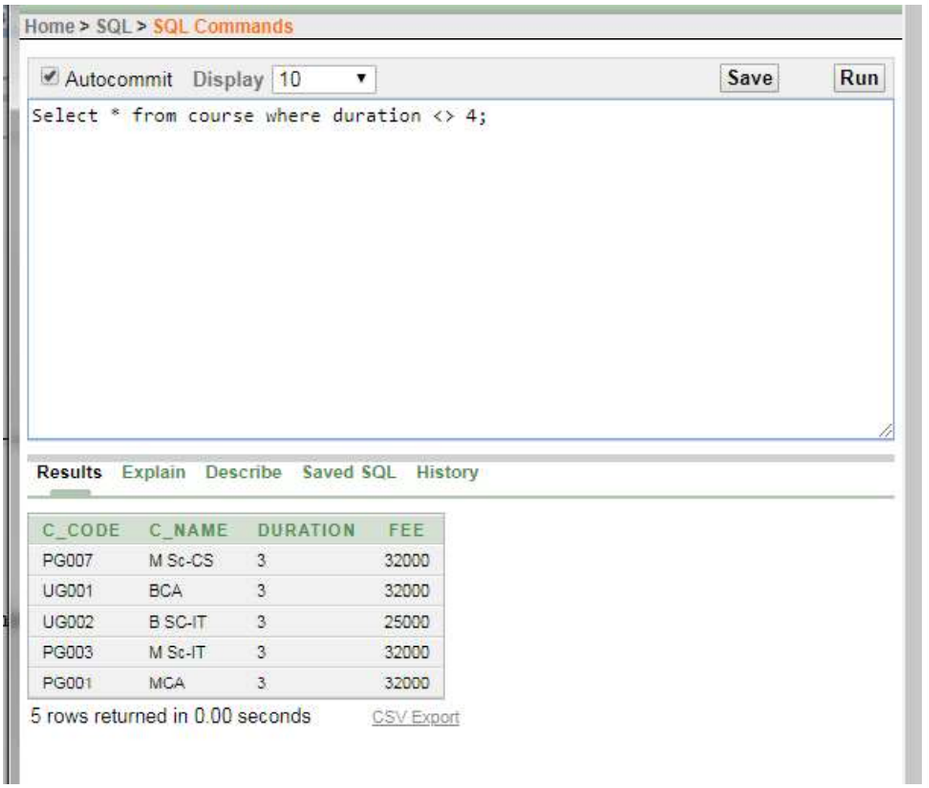


Not Equal to (<>, !=)

To see the detail of course where course duration is not 4 years then the query will be:

```
Select * from course where duration <> 4;
```

Output of the above query is shown below:



Greater Than (>)

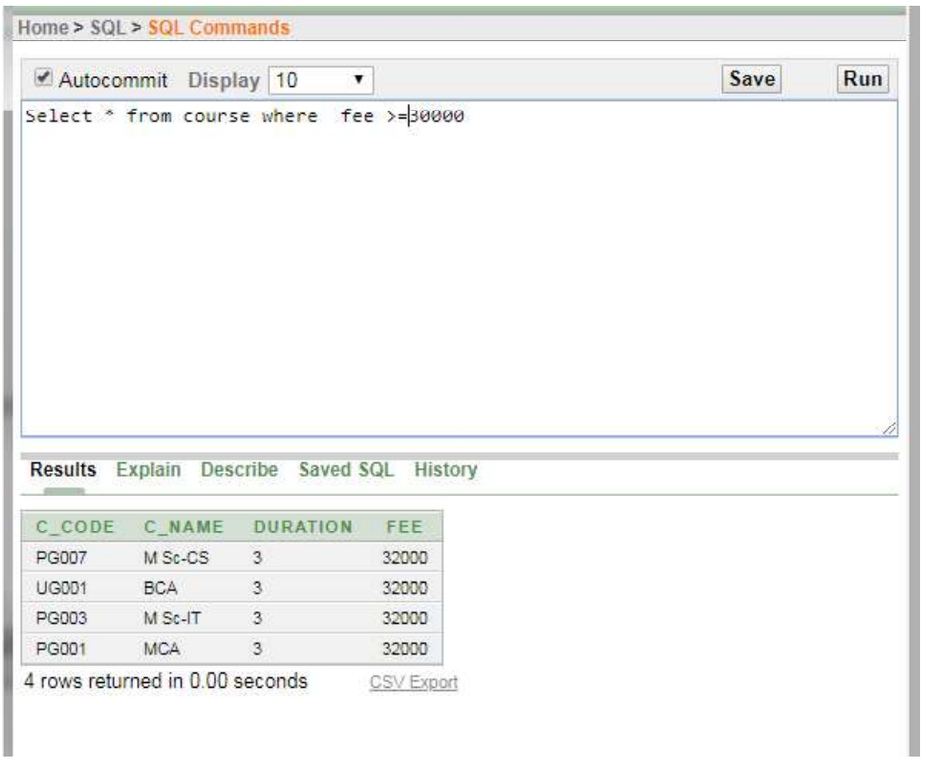
To see the detail of course where course fee is greater than Rs. 50000 then the query will be:

```
Select * from course  where fee >50000;
```

NOTES

NOTES

Output of the above query is shown below:



Similar to operators, equal to, not equal to and greater than operators are used to filer records. Other operators like less than, less than equal to, greater than equal to can be used.

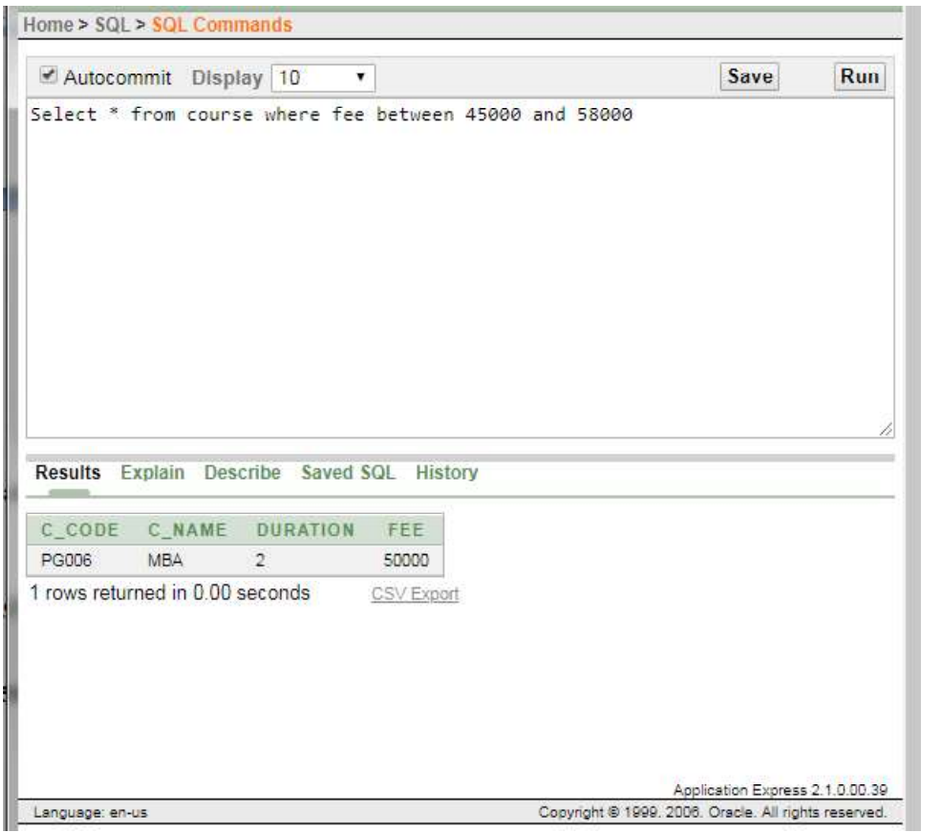
Other Operators in SQL

BETWEEN

The BETWEEN operator filters the records between a given range. Suppose you want to filter the courses where fee is in between Rs. 45000 to Rs. 58000. The query to retrieve such records is given below:

```
Select * from course where fee between 45000 and
58000
```

Output of the above query is shown below:



The between operators can filter the numbers, text, or date values.

NOT BETWEEN

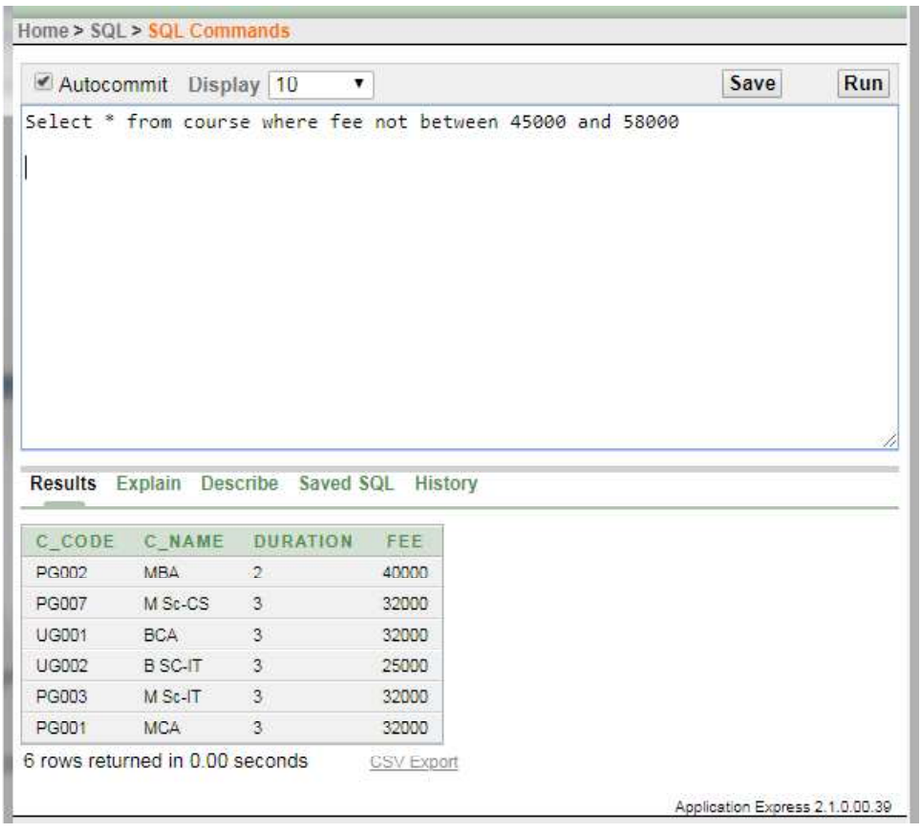
The NOT BETWEEN operator filters the records where the data is not in between a given range.

```
Select * from course where fee not between 45000 and
58000
```

NOTES

NOTES

Output of the above query is shown below:



Oracle Functions

Oracle provides various built in functions for different purpose such as calculation, comparison and conversion of data. Functions may or may not have the arguments (input) and have the capability to return a value.

Basically there are two types of function:

- Aggregate Functions
- Scalar functions

Aggregate Functions

Aggregate functions work on a group of values (a column values) and returns a single value.

Few aggregate functions are listed below:

- SUM()
- MAX()
- MIN()
- AVG()
- COUNT()

Scalar functions

SQL scalar functions return a single value, based on the input value.

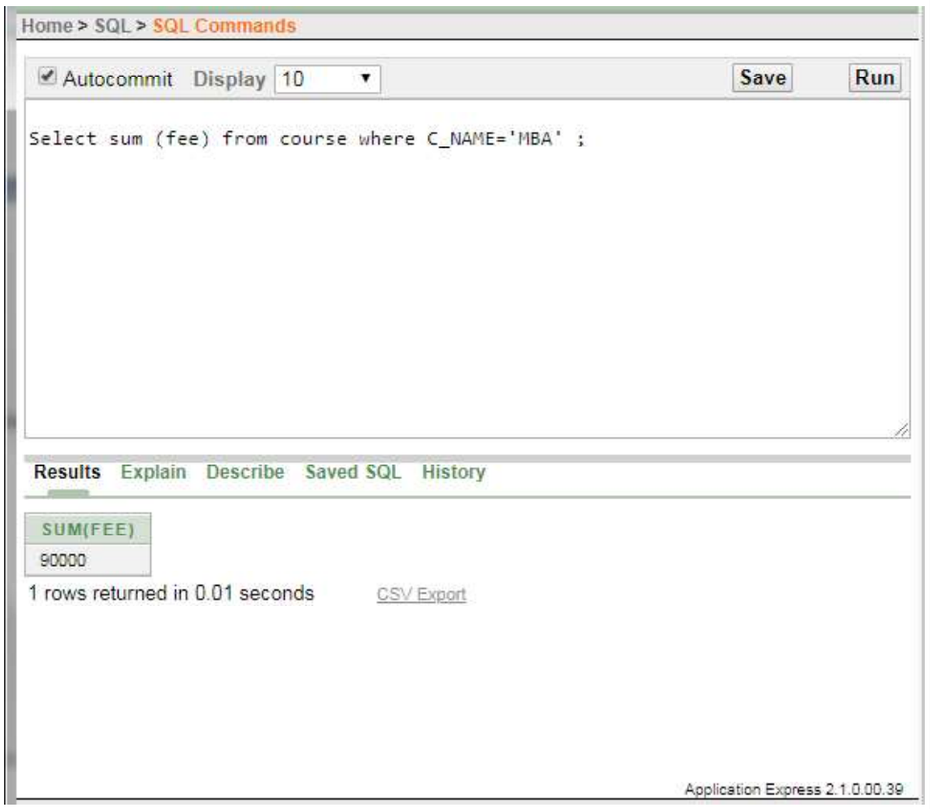
Few scalar functions are listed below:

- MID()
- LEN()
- Upper()
- Lower()

Consider a table *course* with the following records:

C_CODE	C_NAME	DURATION	FEE
PG002	MBA	2	40000
PG006	MBA	2	50000
PG007	M Sc-CS	3	32000
UG001	BCA	3	32000
UG002	B SC-IT	3	25000
PG003	M Sc-IT	3	32000
PG001	MCA	3	32000

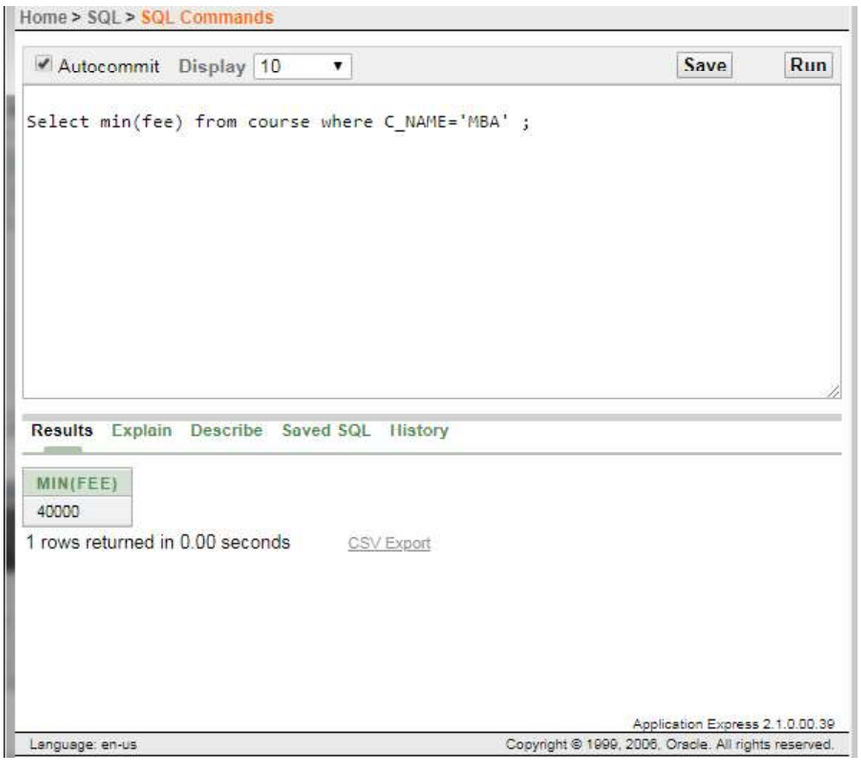
Example 29: Write a query to find the total fee received in MBA course.



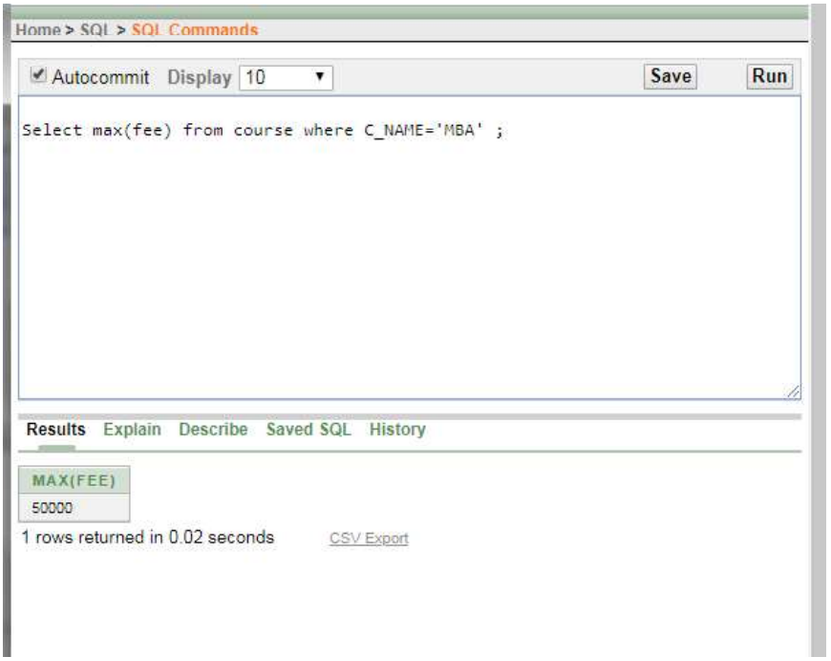
NOTES

NOTES

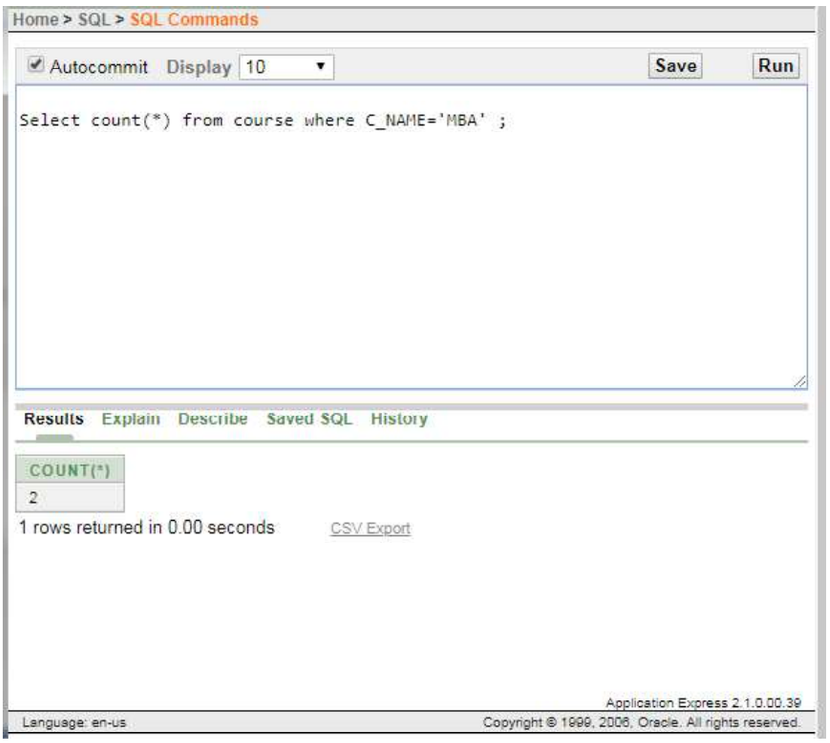
Example 30: Write a query to find the minimum fee received in MBA course from the *course* table.



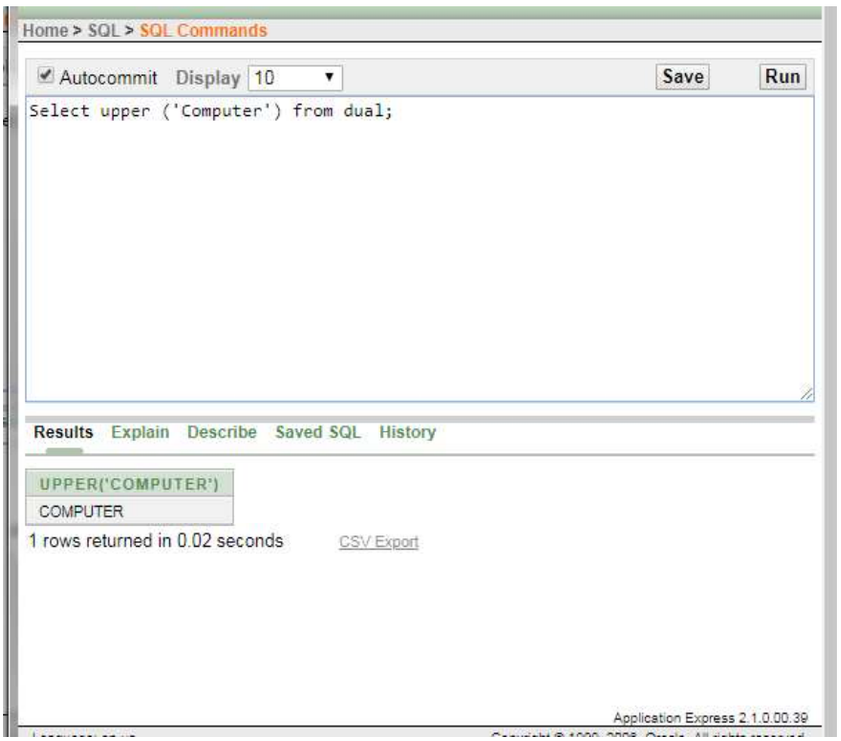
Example 31: Write a query to find the maximum fee received in MBA course from the *course* table.



Example 32: Write a query to count the number of records in *course* table where c_name= 'MBA'.



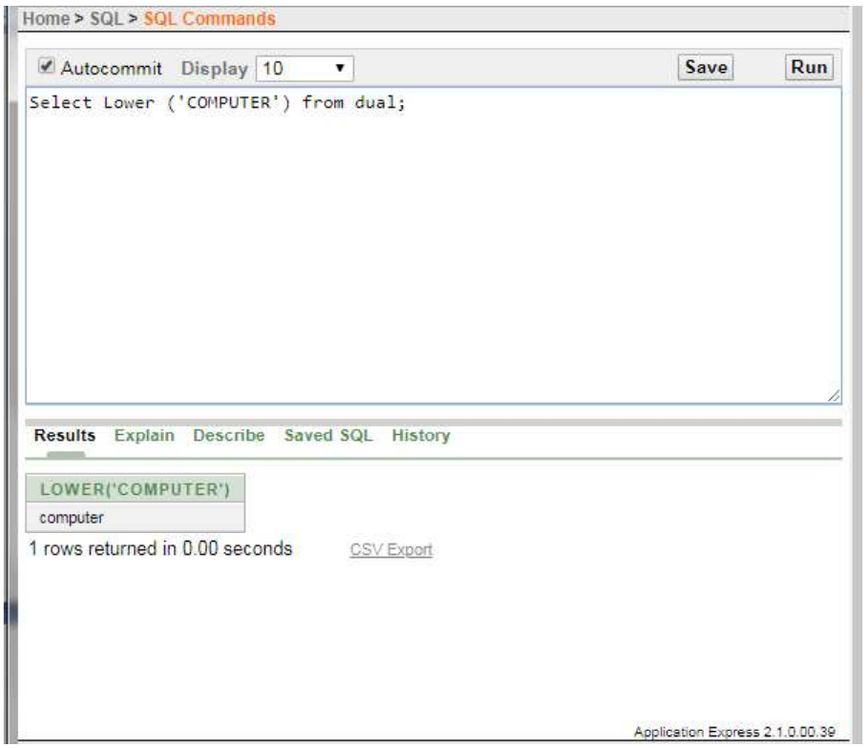
Example 33: Write a query to converts the text (i.e. Computer) to uppercase.



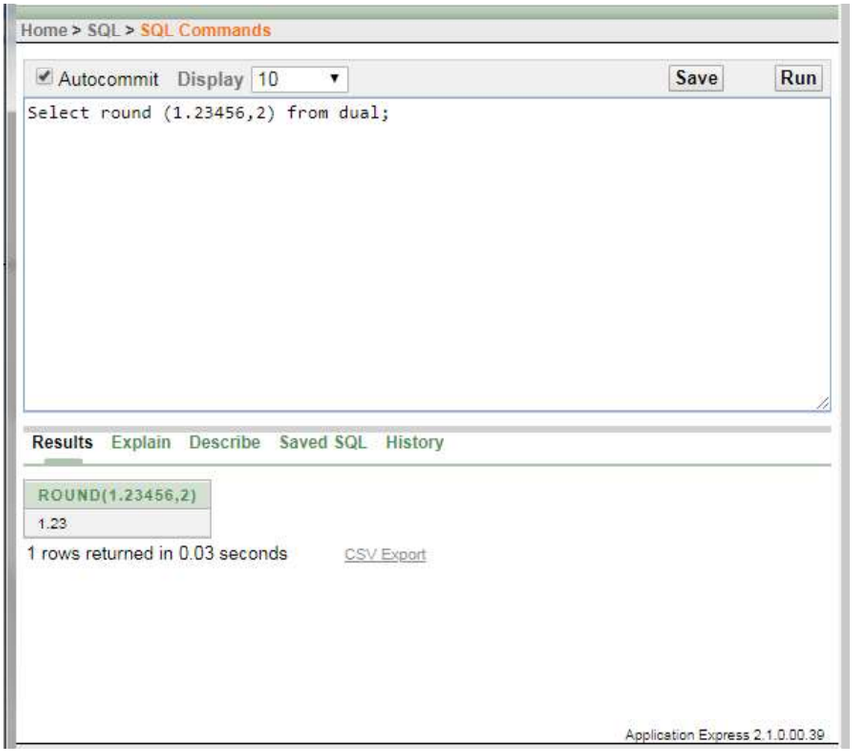
NOTES

NOTES

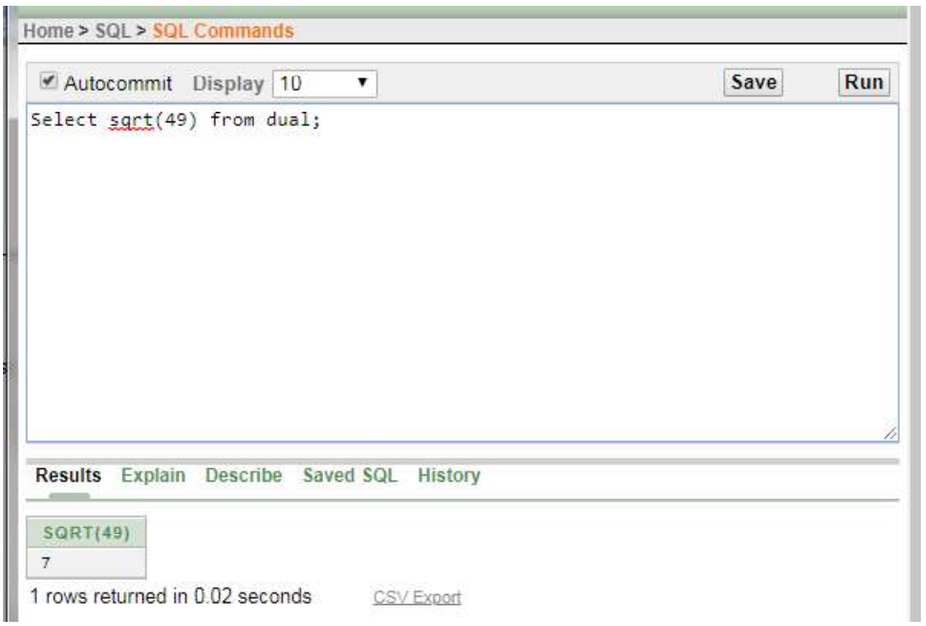
Example 34: Write a query to converts the text (i.e. COMPUTER) to lowercase.



Example 35: Write a query to round the figure (i.e. 1.23456).



Example 36: Write a query to find the square root of 49.



NOTES

Join Commands

Table 1:

```
create table student1(rno number(10),name char(30),course
char(30),fee number(10));
insert into student1 values(101,'NAMAN','B.tech',59000);
insert into student1 values(102,'AMAN','B.tech',59000);
insert into student1 values(102,'SITA','BCA',49000);
insert into student1 values(105,'GITA','MCA',59000);
select * from student1
```

RNO	NAME	COURSE	FEE
101	NAMAN	B.tech	59000
102	AMAN	B.tech	59000
103	SITA	BCA	49000
105	GITA	MCA	59000

Table 2:

```
create table marks1(rno number(10),sub1 number(10),sub2
number(10),sub3 number(10),total number(10));
insert into marks1 values(101,50,40,40,130);
insert into marks1 values(103,60,40,40,140);
insert into marks1 values(105,50,40,50,140);
select * from marks1
```

RNO	SUB1	SUB2	SUB3	TOTAL
101	50	40	40	130
103	60	40	40	140
105	50	40	50	140

NOTES

EQUI JOIN

Example 37: Write a query to display roll no., name, sub1, sub2, sub3 and total form the table student1 and marks1 where student1.rno=marks1.rno.

```
Select student1.rno, name, sub1, sub2, sub3, total from
student1,marks1 where student1.rno=marks1.rno;
```

Output:

RNO	NAME	SUB1	SUB2	SUB3	TOTAL
101	NAMAN	50	40	40	130
103	SITA	60	40	40	140
105	GITA	50	40	50	140

Left Outer Join

```
select  student1.rno,name,sub1,sub2,sub3,total  from
student1 left outer join marks1 on student1.rno=marks1.rno;
OR
select  student1.rno,name,sub1,sub2,sub3,total  from
student1,marks1 where student1.rno=marks1.rno(+);
```

RNO	NAME	SUB1	SUB2	SUB3	TOTAL
101	NAMAN	50	40	40	130
103	SITA	60	40	40	140
105	GITA	50	40	50	140
102	AMAN	-	-	-	-

Table Project:

```
insert into project values(102,'Railway','Manager');
insert into project values(106,'AI','Coder');
select * from project
```

RNO	PNAME	ROLE
102	Railway	Manager
106	AI	Coder

Right Outer Join

```
Select student1.rno, project.rno, name,pname from student1
right outer join project on student1.rno=project.rno;
OR
Select student1.rno, name, sub1, sub2, sub3, total from
student1, marks1 where student1.rno(+)=marks1.rno;
```

RNO	RNO	NAME	PNAME
102	102	AMAN	Railway
-	106	-	AI

Full Outer Join

```
Select student1.rno, project.rno, name, pname from student1
full outer join project on student1.rno= project.rno;
```

RNO	RNO	NAME	PNAME
102	102	AMAN	Railway
103	-	SITA	-
105	-	GITA	-
101	-	NAMAN	-
-	106	-	AI

NOTES

Data Control Language (DCL)

Data Control Language are the commands that allow authorized database users to share the data with other users. The shared data can be accessed or manipulated by other users as per the permission granted.

The data manipulation language statements are GRANT and REVOKE

- **GRANT**-provides user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by the GRANT command.

Oracle Transactions

All the changes made through DML commands are known as transaction. A transaction is a logical group of work. Transactions that you do on a database temporarily stored on the client machine that can be make permanent or canceled by the user. Oracle provides few commands to control the transactions as given below:

- Commit
- Savepoint
- Rollback

Commit

The commit command is used to make the transaction permanent to the database. The commit command ends the current transactions.

```
SQL > Commit;
```

Rollback

The rollback command is used to terminate the current transaction. All the changes made to the rollback database can be undone by rollback. It is generally used when a session disconnects from the database without completing the current transaction.

```
SQL > rollback;
```

When rollback command is executed, Oracle prompts a message as shown below:

```
Rollback complete.
```

** Rollback undone the whole transaction made after the last committed transaction.*

NOTES

Index

An index is a performance-tuning method of allowing faster retrieval of records. An index creates an entry for each value that appears in the indexed columns. By default, Oracle creates B-tree indexes.

Syntax:

The syntax for creating an index in Oracle/PLSQL is:

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column1, column2, ... column_n)
[ COMPUTE STATISTICS ];
```

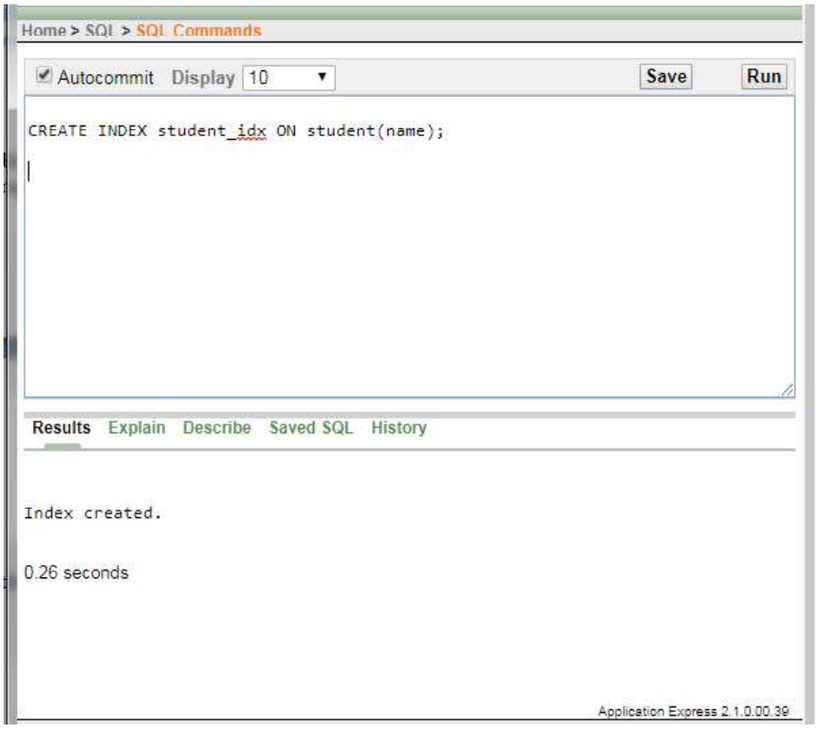
UNIQUE refers to the combination of values in the indexed columns must be unique, Compute Statistics tells Oracle to collect statistics during the creation of the index. The statistics are then used by the optimizer to choose a "plan of execution", when SQL statements are executed.

Example 38: An example to create an index in Oracle/PLSQL.

```
Create index employee_idx
ON employee (name);
```

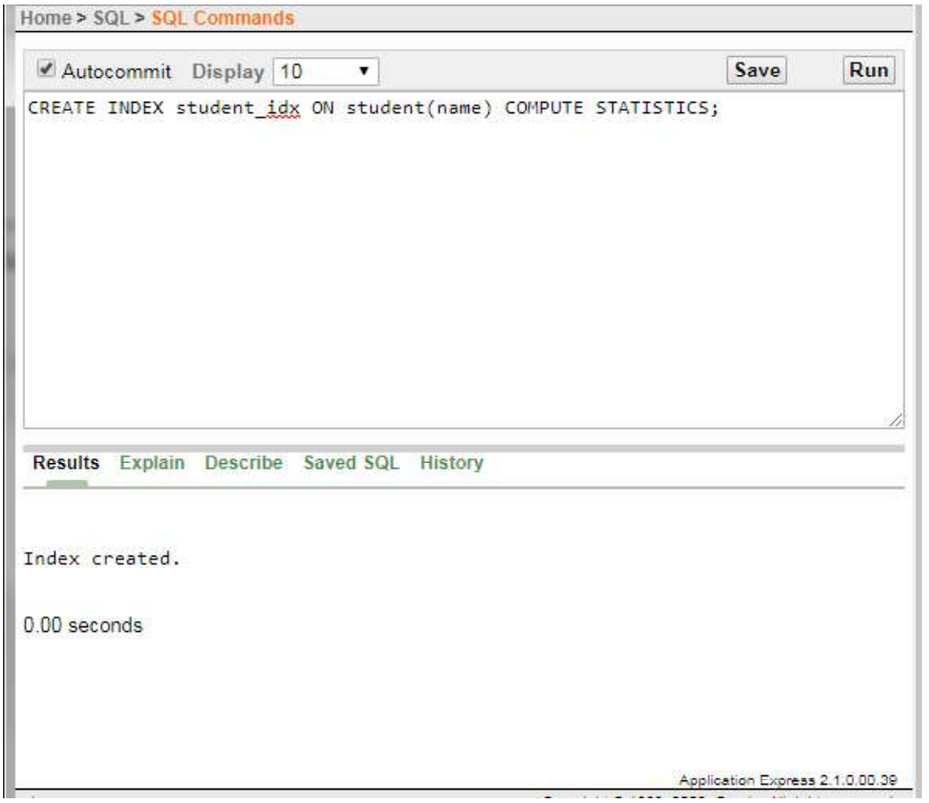
In this example, we've created an index on the employee table called employee_idx. We can also create an index with more than one field as in the example below:

```
CREATE INDEX student_idx ON student(name);
```



We can also choose to collect statistics upon creation of the index as follows:

```
CREATE INDEX student_idx ON student(name) COMPUTE
STATISTICS;
```



NOTES

Rename an Index

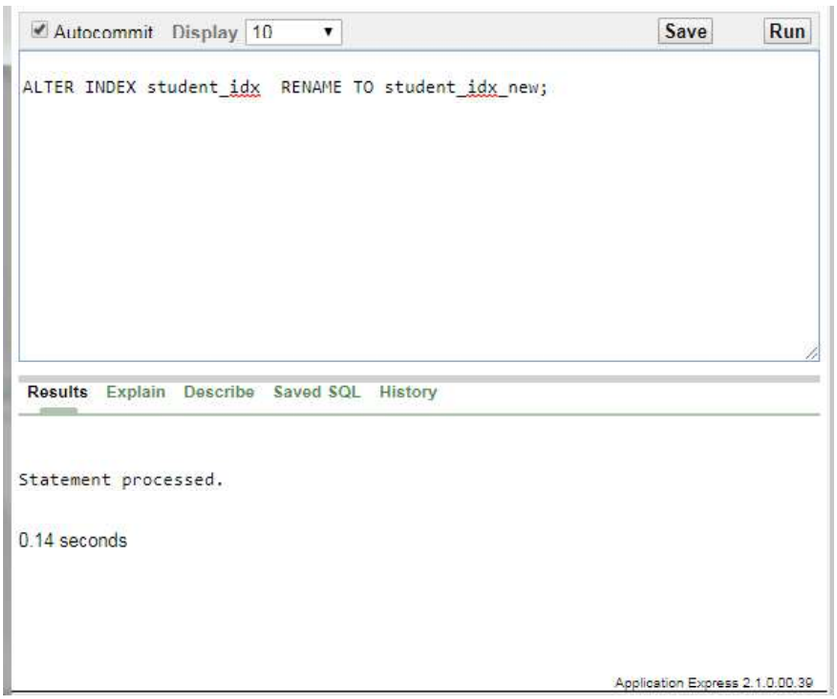
Syntax:

The syntax for renaming an index in Oracle/PLSQL is:

```
ALTER INDEX index_name
 RENAME TO new_index_name;
```

NOTES

Example 39: An example of how to rename an index in Oracle/PLSQL.

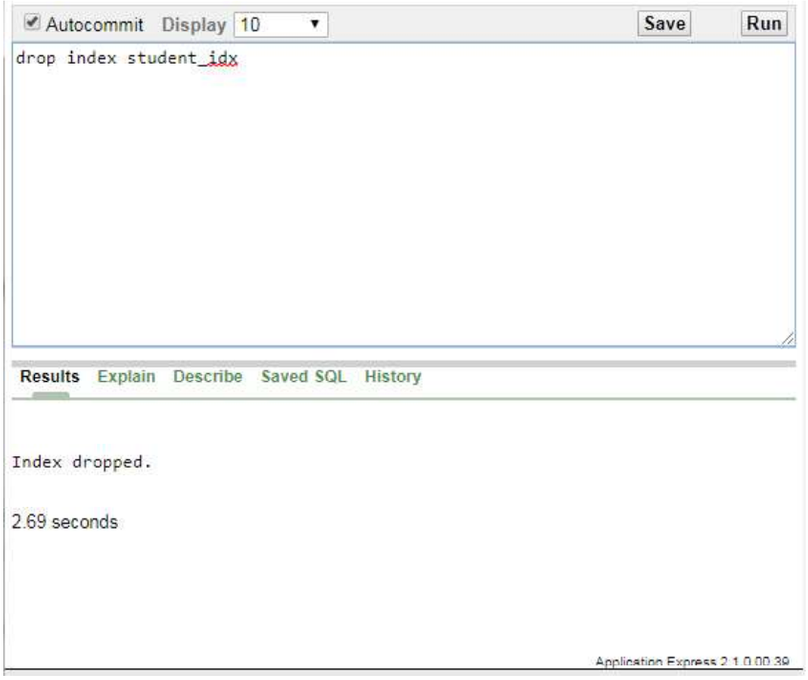


Drop an Index

Syntax: The syntax for dropping an index in Oracle/PLSQL is:

DROP INDEX index_name;

Example 40: An example of how to drop an index in Oracle/PLSQL.



NOTES

View

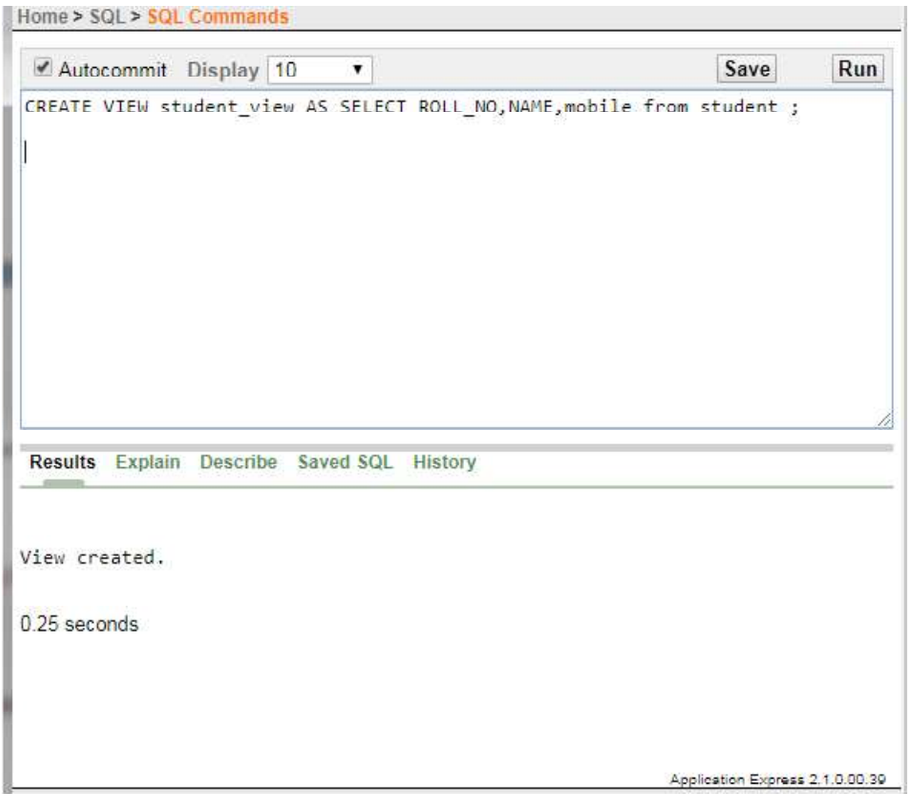
A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it doesn't not store in the database. View is a query stored as an object.

Syntax:

```
CREATE VIEW view_name AS SELECT set of fields FROM
relation_name WHERE (Condition)
```

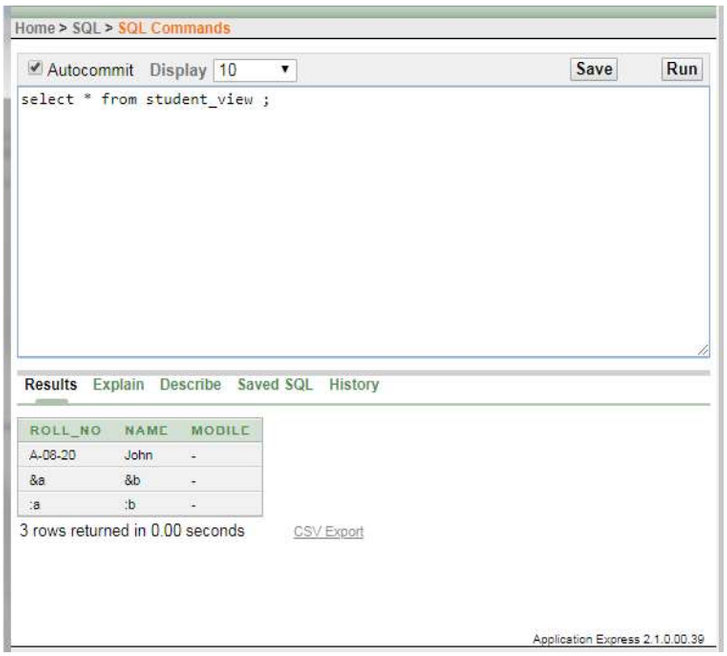
Example 41: Write a query to create a view Student_view having fields roll number, name, mobile using table *student*.



NOTES

Display Records from View

Example 42: To display the records from view.

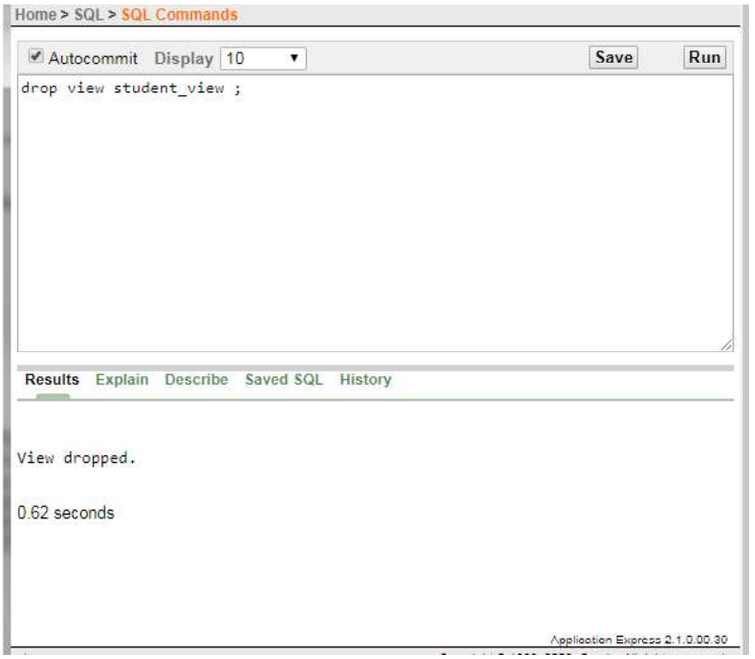


Drop View

Syntax:

```
Drop View View_name;
```

Example 43: Write a query to drop student_view.



PL/ SQL

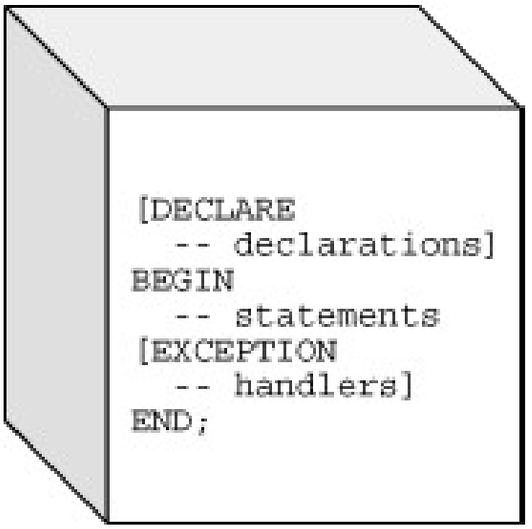
PL/ SQL is also known as an embedded SQL and is a superset of SQL. PL/ SQL is an acronym of Procedural Language/Structure Query Language. It supports procedural features and SQL commands.

NOTES

Structure of PL/ SQL Program

PL/ SQL program block is divided in three sections.

1. Declaration section
2. Execution section
3. Exception handling section



Declaration Section

In declaration section, variables, constants, user defined exceptions, cursor and other objects are declared. This is an optional section. This section begins with the keyword `DECLARE`.

Execution Section

All the executable statements such as SQL statements, control statements, loops are written under this section. This is a mandatory section. This section begins with the keyword `BEGIN` and ends with the keyword `END`.

The Exception Handling Section

During program execution many abnormal situations may occur. To handle these situations, statements are written in this block. These situations are known as errors which occur due to the logical error, syntax error or system error. This is an optional section.

NOTES

Syntax:

```
DECLARE
 declaration_statements
 …
BEGIN
 executable_statements
 …
EXCEPTION
 exception_handling_statements
 …
END ;
..
```

PL/ SQL Engine

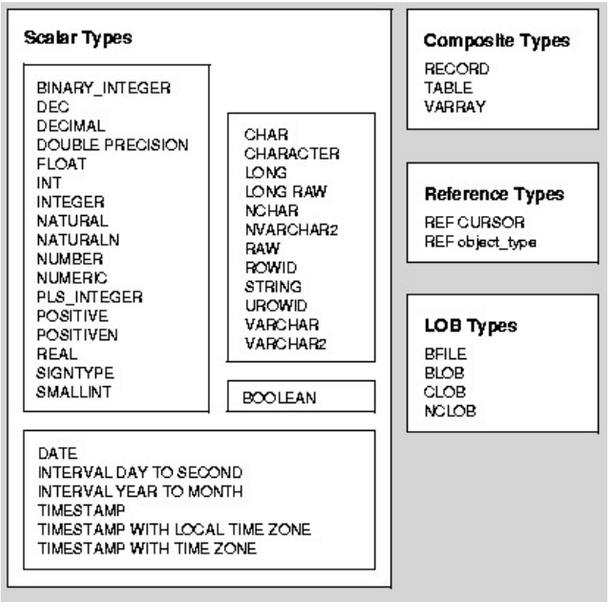
Oracle uses a PL/ SQL engine to processes the PL/ SQL statements. Either the PL/ SQL program is stored on the client side or on the server side. PL/ SQL engine is used by Oracle to execute the program statements.

Data Types in PL/ SQL

A program has many inputs and outputs in the form of variable and constant. These variable and constant specifies the storage format, type of value and a range of the values that can be stored. PL/ SQL provides various data types which are system defined and also gives the flexibility to the programmer to create their own data types.

Classification of Data Types

- Scalar Data Types
- Composite Data Types



Comments in PL/ SQL

In Oracle, comments may be introduced either for single line or for multiple lines.

1. /*...*/ is used for multiple line comments.
2. - - is used for single line comments.

The example for single line comment is given below :

- - This is a PL/ SQL program to calculate employee salary

Variables in PL/ SQL

Variables are the identifiers of data type. These variables could be the identifiers of either system defined (scalar) data types or the identifiers of user defined (composite) data type i.e. record, table or Varray.

Variable declaration can be of any data type. For example**:**

Name char (30) ;

Salary Number (8, 2) ;

Date_of_join Date ;

Constants can be of any data type. For example**:**

Pi constant number (3, 2) := 3.5 ;

Status Booleans := TRUE ;

Pi and Status are assigned with a value during declaration, makes them constant.

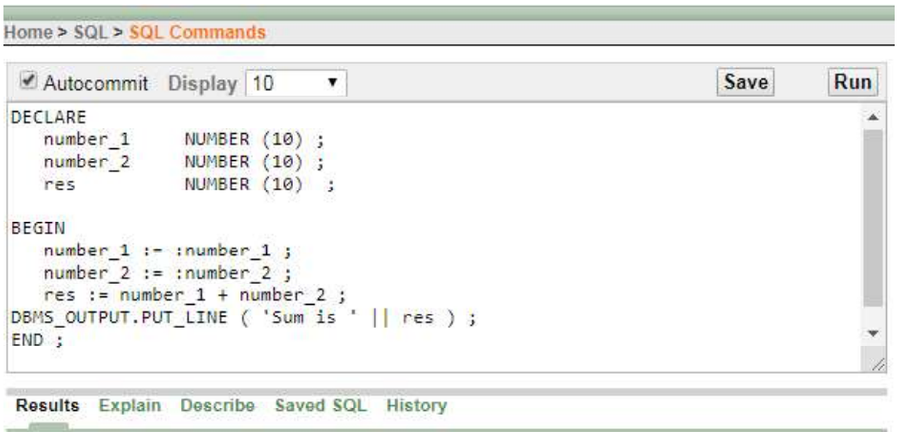
Example 44: Write a PL/SQL program to display ‘’First PL/SQL Program’.

Click on Run button to run program.

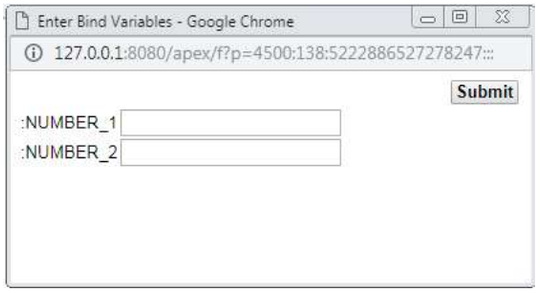
Output:

NOTES

Example 45: Write a PL/SQL program to display sum of two numbers given at run time.



After running this program it will show input screen as shown below:



Enter values in text boxes and click on **Submit** button.

Output:

Example 46: Write a PL/SQL Program to print Prime Number.

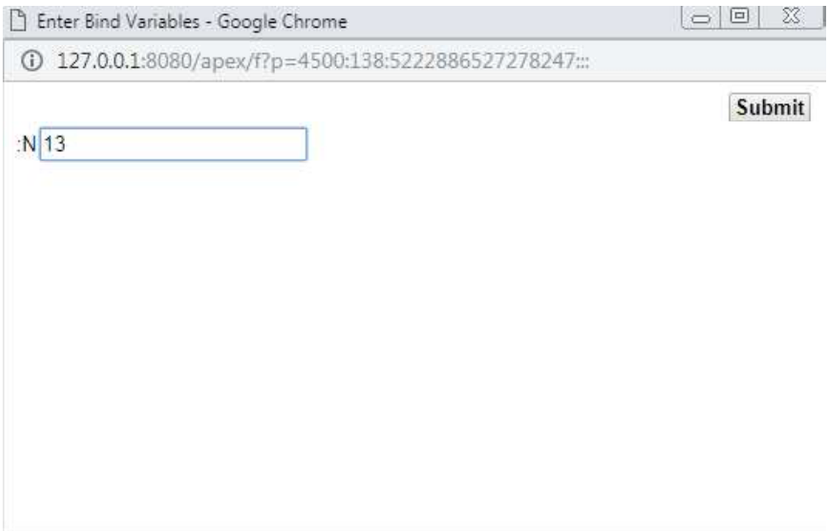
NOTES

Home > SQL > SQL Commands

Autocommit Display 50

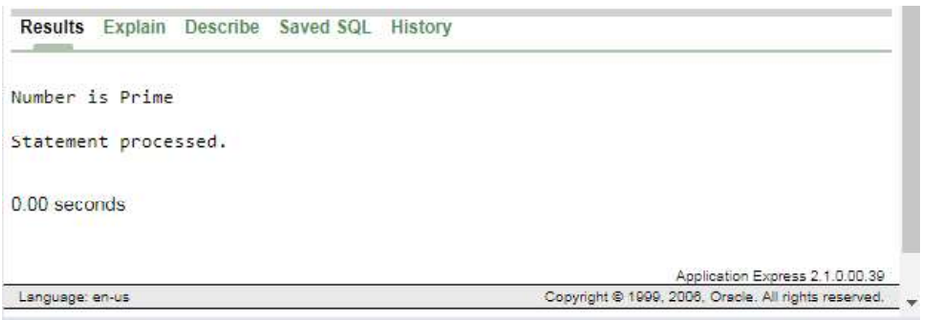
```
declare
n number;
i number;
        flag number;
 begin
        i:=2;
        flag:=1;
        n:=:n;

        for i in 2..n/2
        loop
                if mod(n,i)=0
                then
                        flag:=0;
                        exit;
                end if;
        end loop;

        if flag=1
        then
                dbms_output.put_line('Number is Prime');
        else
                dbms_output.put_line('Number is not Prime');
        end if;
end;
```

Input:

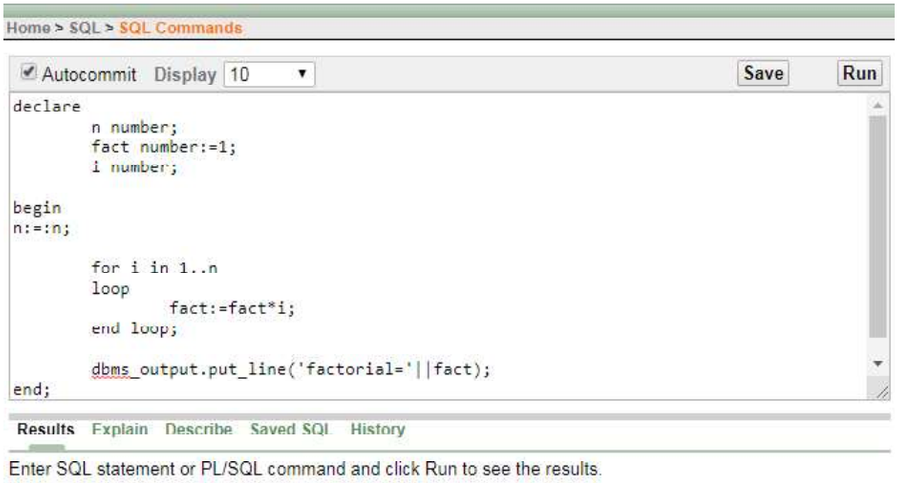


NOTES

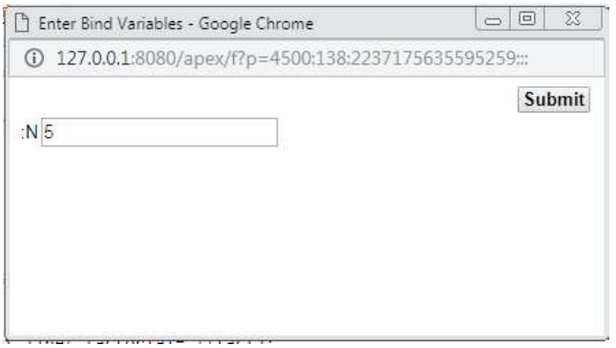
Output:



Example 47: Write a PL/SQL Program to find factorial of a number given number.



Input:



Output:

NOTES

Try Yourself:

1. Write PL/SQL program to display demonstrate all sections of PL/SQL program.
2. Write PL/SQL program to display HELLO.

Exception Handling

In PL/ SQL, error is called as exception. Error may occur due to various reasons such as semantic error, hardware failure, system resources problems and many other reasons. Due to these errors program terminates abnormally.

Types of Exception

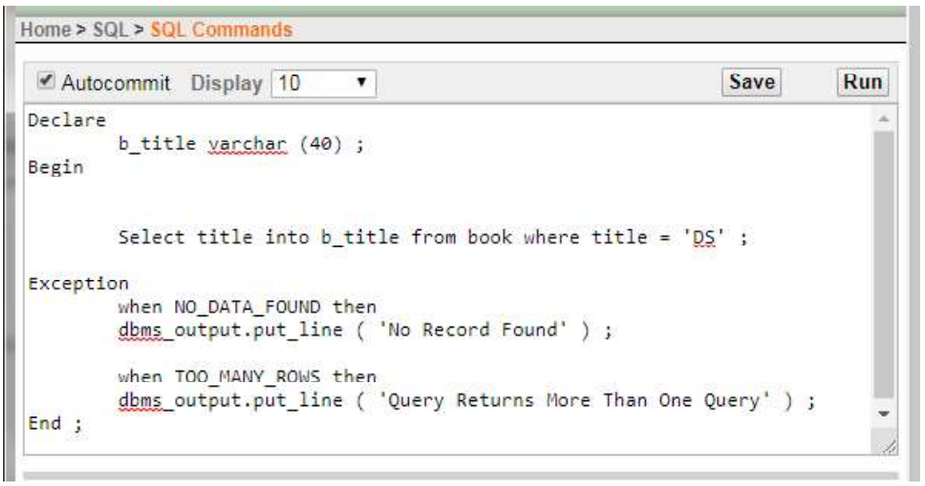
1. Internal exception
2. User-defined exceptions

Table: *Internal Exceptions*

Exceptions	Explanation
ZERO_DIVIDE	This exception raised when PL/SQL program attempts to divide a number by zero.
NO_DATA_FOUND	This exception raised when SELECT INTO statement returns no rows while expected to return.
CURSOR_ALREADY_OPEN	This exception raised when you try to open a cursor which is already.
INVALID_NUMBER	This exception raised when, the conversion of a string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	This exception raised when PL/SQL program attempts to log on to Oracle with an invalid username and/or password.
NOT_LOGGED_ON	This exception raised when PL/SQL program issues a database call without being connected to Oracle.
STORAGE_ERROR	This exception raised when PL/SQL runs out of memory.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row while expected only one.
VALUE_ERROR	This exception raised when data type or data size is invalid.
PROGRAM_ERROR	This exception raised when PL/SQL has an internal problem.
OTHERS	This exception raised when error is unknown or not explicitly defined.

NOTES

Example 48: Write a program to demonstrate exception handling.



Query returns more than one records then TOO_MANY_ROWS exception:

In the above program, select query is used to select book title into variable B_title. Two internal exceptions are handled named NO_DATA_FOUND and TOO_MANY_ROWS. If query returns more than one records then TOO_MANY_ROWS exception would be raised by the system, if no record matches then NO_DATA_FOUND exception would be raised.

User Defined Exceptions

You can assign a name to unnamed system exceptions using a **Pragma** called **Exception_Init as shown below:**

```
Pragma Exception_Init (exception name, Oracle error
number);
```

In the above example, exception name is the user defined name of the exception that will be associated with Oracle error number.

Syntax:

```
DECLARE
   exception_name EXCEPTION ;
   PRAGMA EXCEPTION_INIT (exception_name, Err_code);
BEGIN
   Executable statement;
   . . .
```

```
EXCEPTION
    WHEN exception_name THEN
    Handle the exception
END;
```

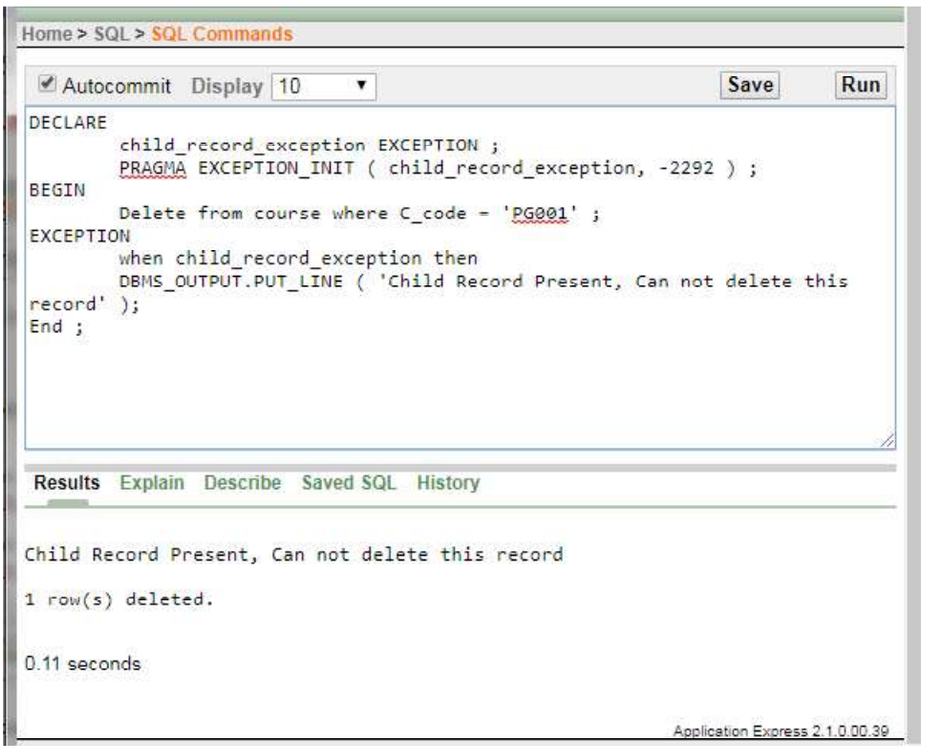
NOTES

Example 49: Write PL/SQL program to the given scenario given below:

Let's consider the student table and course tables.

The c_code is a primary key in course table and c_code is a foreign key in student table.

If you try to delete a c_code from course table and it has a corresponding child records in student table an exception will be thrown with oracle code number -2292.



child_record_exception is a user defined name of exception in the above example.

RAISE_APPLICATION_ERROR ( )

A user can assign an error message by using

Raise_application_error ( ) to make the error message more descriptive for the end-user. It is a build-in procedure.

Example 50: Write a PL/SQL program to demonstrate User-defined Exceptions.

Other than the pre-defined exceptions, you can define your own exception to validate data against business requirements. For example, if user wants to update

NOTES

total marks of student but subject marks are NULL, an error must be raised by the system to alert the user.

A user defined exceptions must be declared within declaration section by the keyword EXCEPTION and must be raised explicitly by RAISE statement within the executable section.

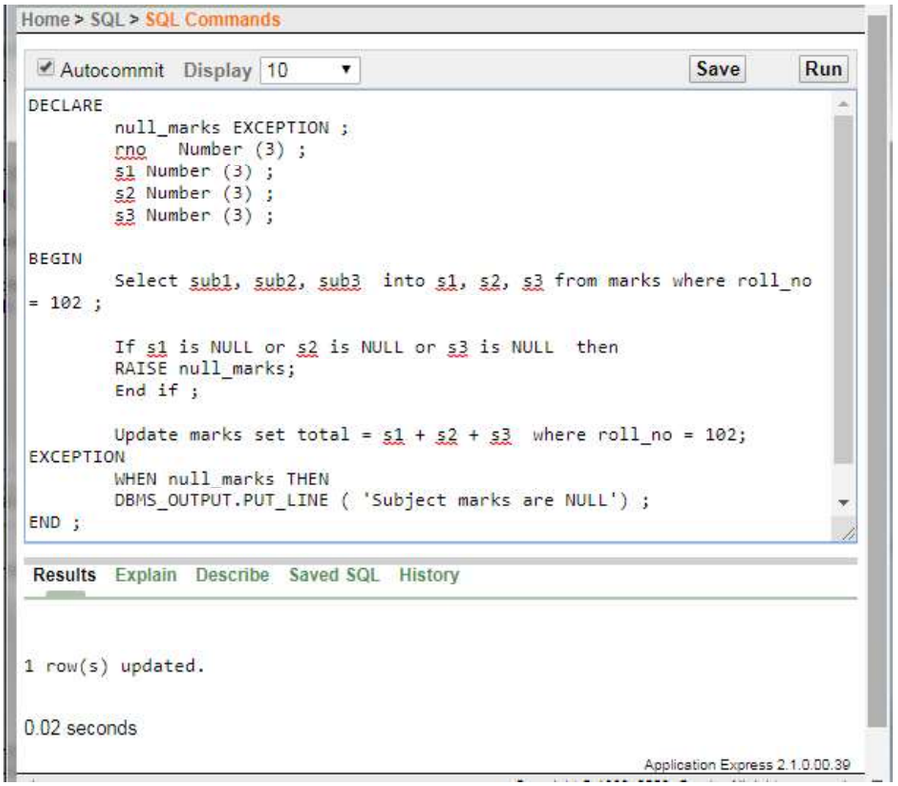
Create Table Marks:

```
Create table marks ( roll_no number(3), sub1 number(3),
sub2 number(3), sub3 number(3), total number(3) )
```

Insert values in roll_no, sub1, sub2, sub3 fields only:

```
insert into marks (roll_no, sub1 ,sub2, sub3) values
(101,34,54,43)
insert into marks (roll_no, sub1 ,sub2, sub3) values
(102,54,54,50)
insert into marks (roll_no, sub1 ,sub2, sub3) values
(104,65,44,40)

Select * from marks;
```

ROLL_NO	SUB1	SUB2	SUB3	TOTAL
101	34	54	43	-
102	54	54	50	-
104	65	44	40	-



In the above example, **null_marks** is a user defined exception which must be raised explicitly using RAISE statement. This exception would be raised, when marks in any subject would be NULL.

Check student's marks, after executing the above program:

```
select *from marks;
```

ROLL_NO	SUB1	SUB2	SUB3	TOTAL
101	34	54	43	-
102	54	54	50	158
104	65	44	40	-

Try Yourself:

1. Write a PL/SQL code block that will accept an account number from the user and debit an amount of Rs. 2000 from the account, if the account has a minimum balance of 500 after the amount is debited.
2. Write a PL/SQL code block to calculate the area of the circle for a value of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in a table Areas.

   Areas – radius, area.
3. Write a PL/SQL block of code for inverting a number 5639 or 9365.
4. Write a PL/SQL block of code to achieve the following: if the price of Product 'p00001' is less than 4000, then change the price to 4000. The Price changes to be recorded in the old_price_table along with Product_no and the date on which the price was last changed. Tables involved: product_master- product_no, sell_price.

   Old_price_table- product_no,date_change, Old_price

Cursor

Oracle allocates a memory known as the context area for the processing of the SQL statements. A cursor is a pointer or handle to the context area. Through the cursor, a PL/SQL program can control the context area and what happens to it as the statement is processed.

The three types of the cursors are:

1. Static cursors
2. Dynamic cursors
3. REF cursors

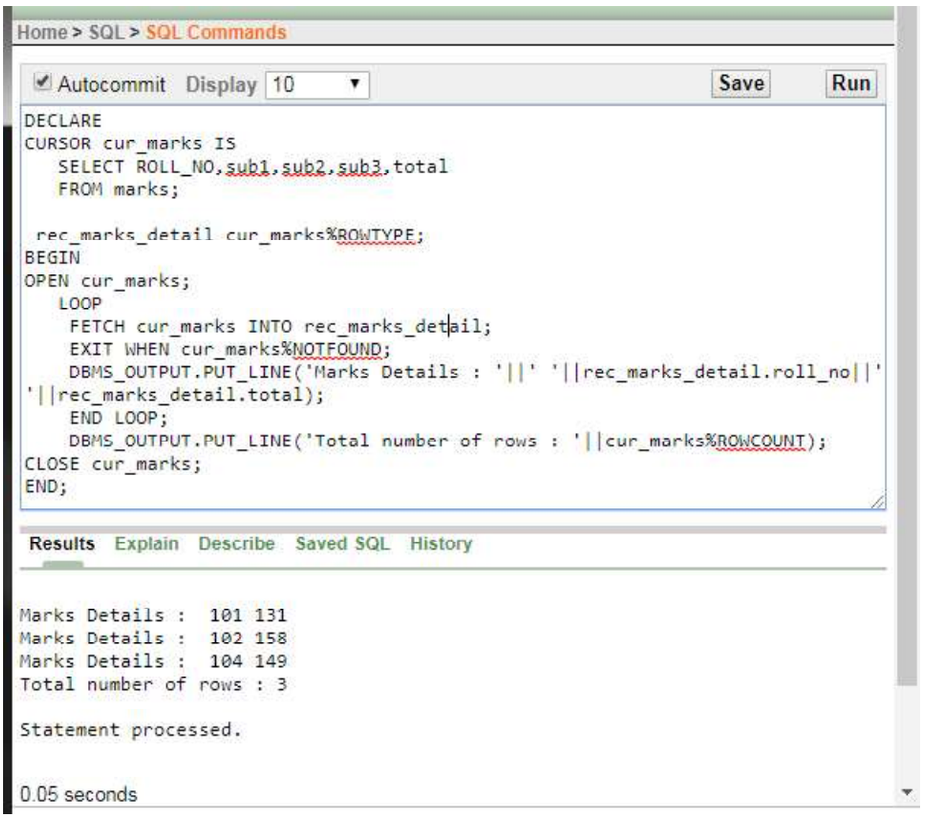
Static cursors are the ones whose select statements are known at the compile time. These are further classified into:

- Explicit cursors
- Implicit cursors

NOTES

NOTES

Example 51: Create a cursor to show roll number and total marks of students from *marks* table using cursor.



Trigger

A trigger is a PL/ SQL code block that runs automatically when an event occurs. An event in PL/ SQL is the data definition language such as INSERT, UPDATE or DELETE.

Uses of a Trigger

A database trigger helps in maintaining the organization's database in such a manner that without executing the PL/ SQL code explicitly, it update and validate the data. Triggers have the capabilities to provide a customized management system of your database.

Database trigger can be used to cater the following purposes:

- To enforce integrity constraints (e.g. check the referenced data to maintain referential integrity) across the clients in a distributed database
- To prevent generate invalid transactions in database.
- To update data automatically to one or more tables or views without user interaction

NOTES

- Automatically generate derived column values
- To customize complex security authorizations.
- To permit insert, update or delete operations to a associated table only during predetermined a date and time.
- Provide auditing
- Provide transparent event logging
- Helps in prompting information about various events taken on database, events of users, and SQL statements to subscribe applications.
- Helps in maintaining replication of synchronous table
- Helps in gathering statistics on various table accesses.

Structure of PL/SQL Trigger

Syntax:

```
CREATE [OR REPLACE ]
    TRIGGER <trigger_name>
    BEFORE (or AFTER)
    INSERT OR UPDATE [OF COLUMNS] OR DELETE
    ON table_name
    [FOR EACH ROW [WHEN (condition)]]

DECLARE
Declaration statements
…
BEGIN
Executable statements
...
EXCEPTION
Exception handling statement
…
END;
```

A database trigger can also have declarative and exception handling parts.

How to Apply a Trigger

A database trigger has three sections namely a trigger statement, a trigger body and a trigger restriction.

Three of Parts of Trigger are:

1. A Trigger Statement
2. A Trigger Body Action
3. A Trigger Restriction

NOTES

Example 52: To Create a Trigger.

A company XYZ has the employee detail in employee table. Company wants to have the history of all the employees who have left the organization. To store the employee history, a new table emp_history is created with the same structure as employee table.

The structure of employee table is shown below:

Column Name	Data Type	Size
EMP_CODE	NUMBER	10
E_NAME	Varchar2	15
DESIGNATION	Varchar2	35
SALARY	NUMBER	10,2
DEPTNO	NUMBER	2

The employee table contains the following records:

EMP_CODE	E_NAME	DESIGNATION	SALARY	DEPTNO
7369	SMITH	CLERK	15000	20
7499	ALLEN	SALESMAN	35000	30
7521	WARD	SALESMAN	32000	30
7566	JONES	MANAGER	55000	20
7654	MARTIN	SALESMAN	30000	30
7698	BLAKE	MANAGER	60000	30
7782	CLARK	MANAGER	64000	10
7788	SCOTT	ANALYST	58000	20
7839	KING	PRESIDENT	70040	10
7844	TURNER	SALESMAN	30430	30
7876	ADAMS	CLERK	23000	20

Create a Duplicate Table of *Employee*

To maintain the employee history, a table emp_history can be created with the SQL command given below:

```
Create table emp_history as select * from employee where
emp_code is null;
```

You can see the structure of new table emp_history by giving command as written below:

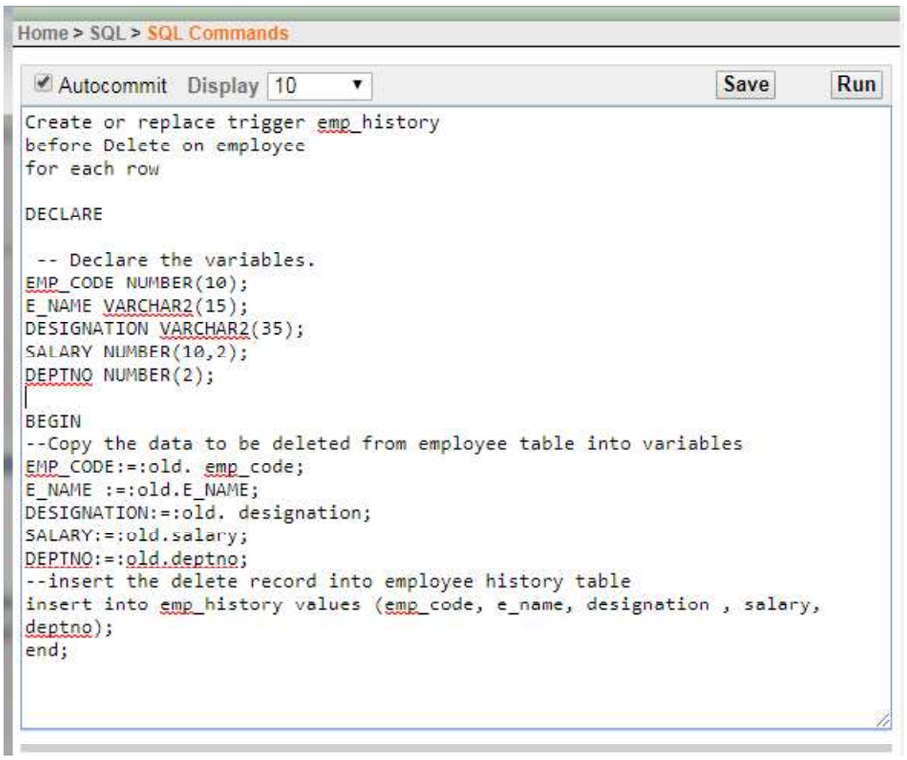
```
Desc emp_history;
```

NOTES

Column Name	Data Type	Size
EMP_CODE	NUMBER	10
E_NAME	Varchar2	15
DESIGNATION	Varchar2	35
SALARY	NUMBER	10,2
DEPTNO	NUMBER	2

Whenever any employee leaves the organization his or her detail will be deleted from the employee table and the same record should be inserted into emp_history table. A trigger can be associated on table employee on the event delete.

The code for trigger is given below:



In the above example, **emp_history** is a trigger which is associated with the employee table. This is a trigger which should be fired with delete command on *employee* table and will store the deleted record in *emp_history* table.

Application:

To test whether the trigger is fired and insert the deleted record in emp_history table delete few records from employee table as shown below:

```
SQL> delete from employee where emp_code = 7782;
SQL> delete from employee where emp_code = 7876;
SQL> delete from employee where emp_code = 7844;
```

NOTES

After executing the above queries, display all the records from the *emp_history* table.

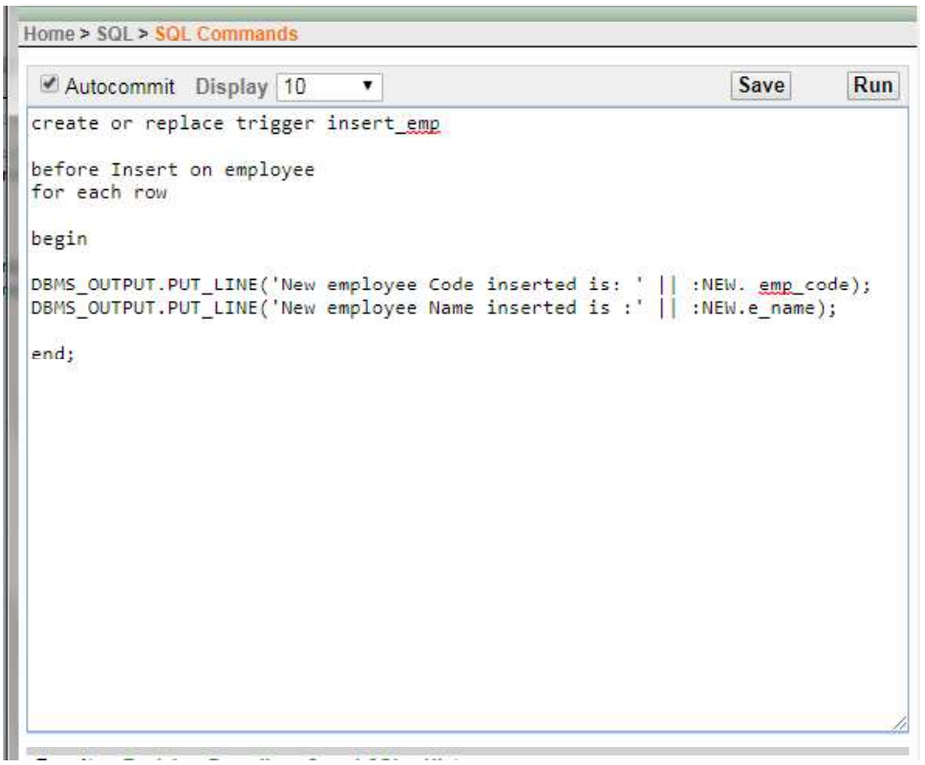
```
Select *from emp_history;
```

The above command would prompt the record as shown below:

EMP_CODE	E_NAME	DESIGNATION	SALARY	DEPTNO
7782	CLARK	MANAGER	64000	10
7876	ADAMS	CLERK	23000	20
7844	TURNER	SALESMAN	30430	30

Example 53: Before Insert Trigger

In the below example, a trigger is associated with the employee table. This trigger would fire before inserting a new record in the table.



In the above example, **insert_emp** is a trigger which is associated with the employee table. This is a trigger would fire on insert command on *employee* table and would prompt new employee code and employee name before inserting it in to employee table.

Application:

To test whether the trigger is fired and display message on screen, insert new record into *employee* table as shown below:

```
SQL> Insert into employee (emp_code, e_name) values
(321,'Scott');
```

When new record is inserted into *employee* table, system prompts the message as shown below:

New employee Code inserted is :321

New employee Name inserted is :Scott

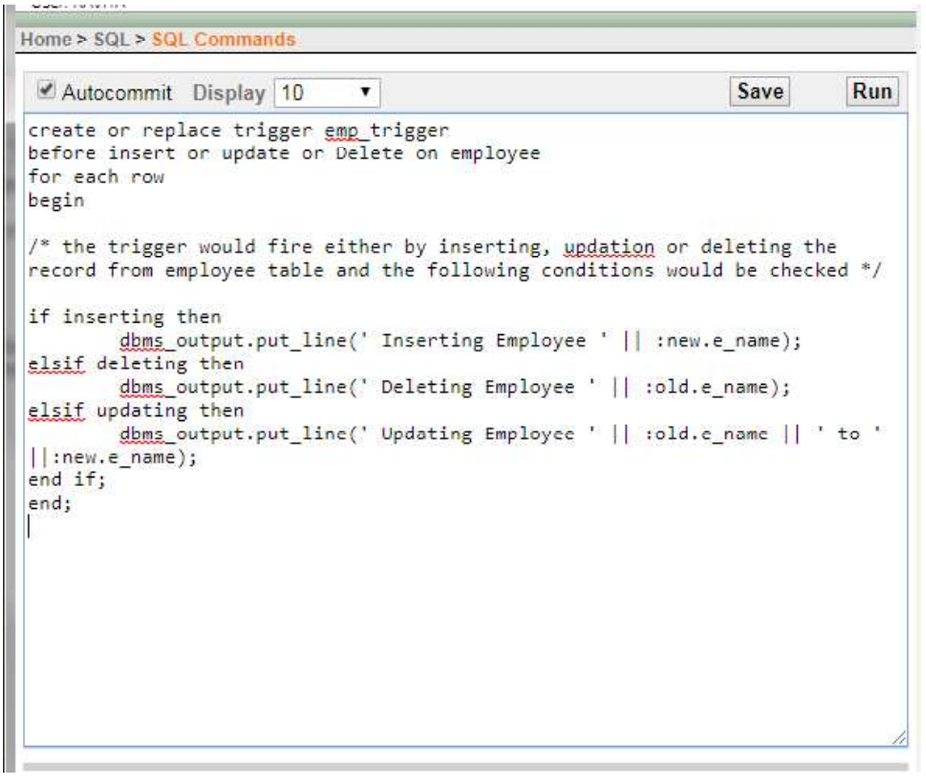
Note: The trigger would execute even if you insert data in all the fields of employee table.

Example 54: To create IF Statement in Trigger.

A database trigger also use if statement. If statements in database triggers is used to determine what statement caused the execution of the trigger, such as inserting, updating or deleting a data from the associated table.

The general form of if statements in trigger are:

- If Inserting Then
- If Deleting Then
- If Updating Then



NOTES

NOTES

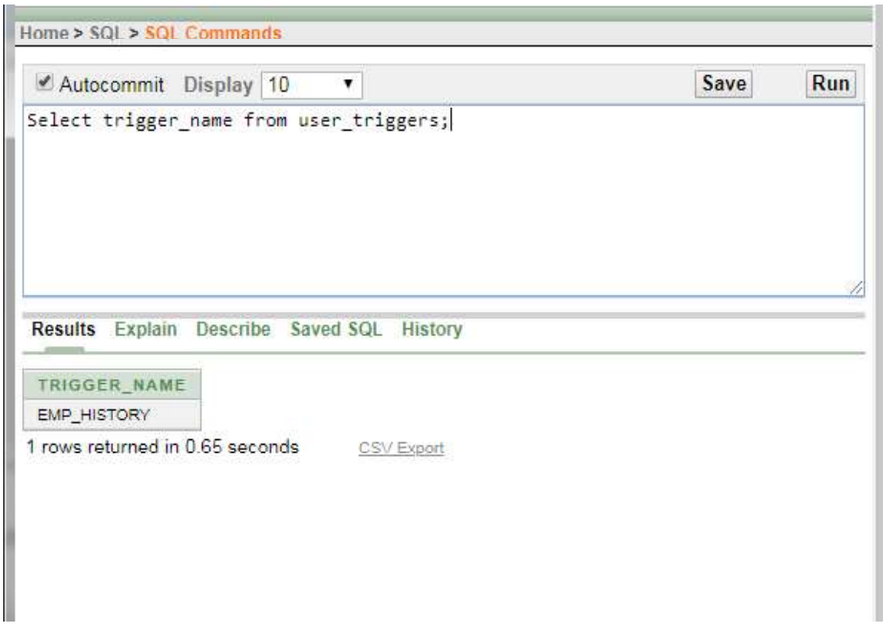
In the above example, **emp_trigger** is a database trigger which is associated with the employee table. This is a trigger having three if conditions to determine what statement invoked it, and prompts an appropriate message in various cases.

Viewing Triggers

To view all the triggers created by the user, a data dictionary named USER_TRIGGERS can be used.

To see all the triggers use select statement on USER_TRIGGERS as shown below:

```
Select trigger_name from user_triggers;
```



For more description, you can also write the following command:

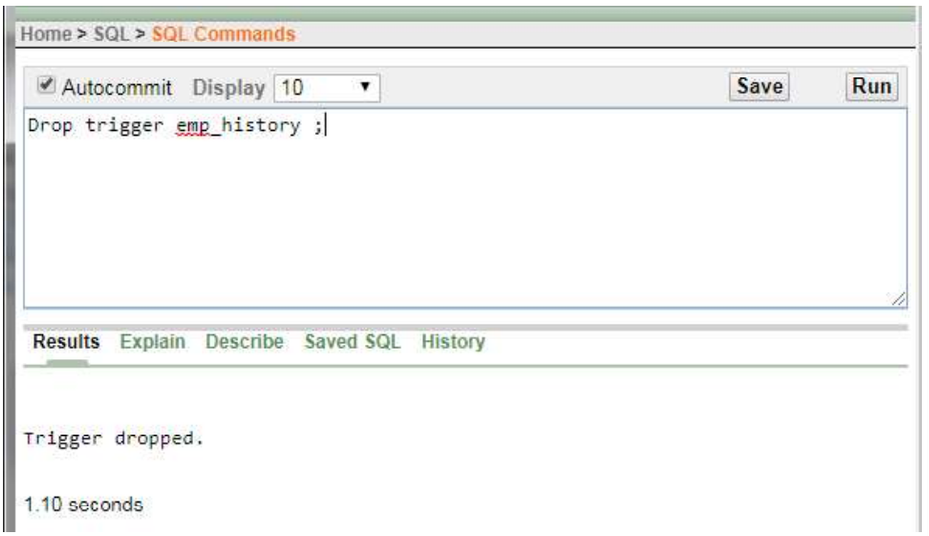
```
SQL> Select * from user_triggers;
```

Deleting a Trigger

Syntax:

```
SQL > Drop trigger < trigger name >
```

Example 55: Write a query to delete a trigger from emp_history.



NOTES

PL/ SQL Package

A package is a database object. It is a collection of various database objects as procedures, functions, cursors, variables and constants.

There are two types of packages:

1. Built-in Packages
2. User defined Packages

Built-in Packages

Built-in Packages such as DBMS_OUTPUT, DBMS_SQL, DBMS_DDL, DBMS_TRANSACTION etc. caters pre-defined functionality.

User defined Packages

User defined package serve the user as per the changed business needs.

A package consists of two parts:

- Package Specification
- Package Body

NOTES

Package Specification

In package specification one can declare variables, constants, exceptions, cursors, sub-procedures and other database objects.

Syntax:

```
CREATE [or Replace] Package < package_name > IS <
declarations >
Begin
       (Executable statements)
END <package_name >;
```

The sub-procedures declared in package specification must be declared in package body.

Package Body

The actual implementation of declared sub-procedures and cursors is done in package body. The sub-procedures declared in package specification must be declared in package body.

Syntax: The CREATE BODY statement is as follows:

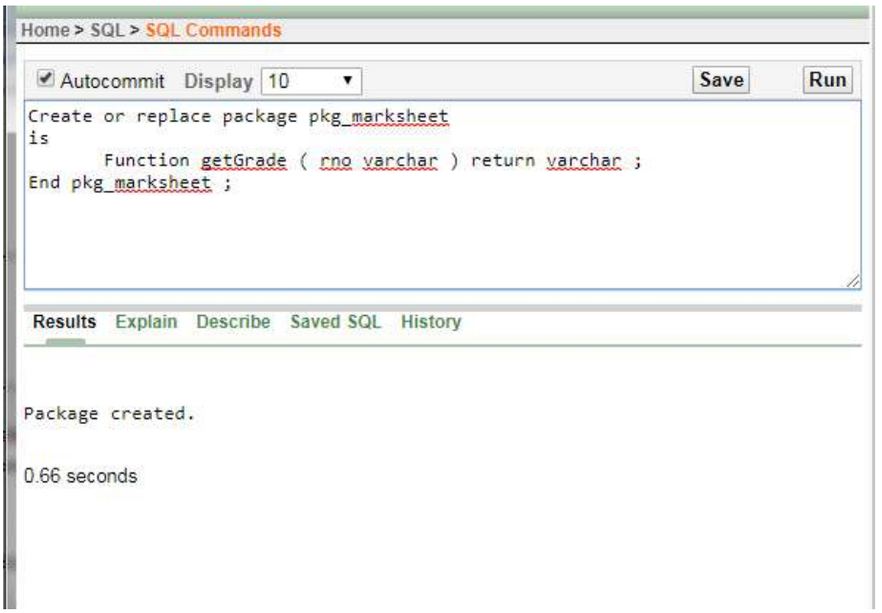
```
CREATE [or Replace] package < package_name > IS <
declarations >
Procedure < procedure_name > (variable data type);
Function < function_name > (variable data type) return
data type;
END < body_name >;
```

A Package Function

The example given below declares a function getGrade which would accept an argument of varchar data type and would return a value of varchar data type.

Example 56: To create or replace a package.

Step 1:



NOTES

The above code will create a package with the name pkg_marksheet. This package contains a function named getGrade. This function will accept an argument of varchar type and will return a value of varchar type.

Package created.

Step 2:

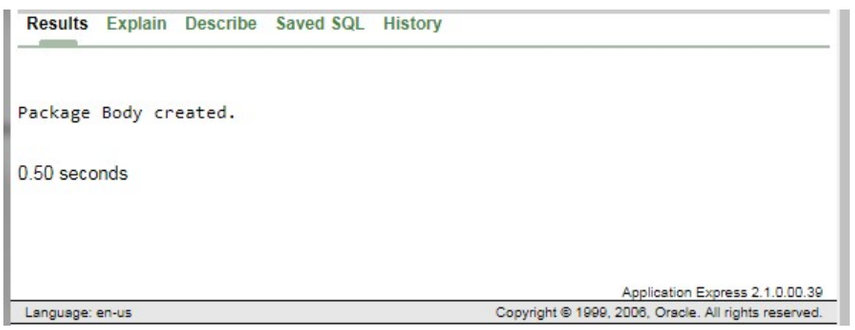
The function pkg_marksheet is declared in package body as shown below:

```
create or replace package body pkg_marksheet as
function getgrade (rno varchar ) return varchar IS
    s1 number (3) ;
    s2 number (3) ;
    s3 number (3) ;
    s4 number (3) ;
    total number (3) ;
    per number (3) ;
```

NOTES

```
begin
    select sub1, sub2, sub3, sub4 into s1, s2, s3 , s4
from marks where roll_no = rno ;
    total := s1 + s2 + s3 + s4 ;
    per := total / 4 ;
if per >= 90 then
    return 'A+' ;
elsif per >= 80 then
    return 'A' ;
elsif per >= 70 then
    return 'A-' ;
elsif per >= 60 then
    return 'B+' ;
elsif per >= 50 then
    return 'B' ;
elsif per >= 40 then
    return 'B-' ;
elsif per >= 30 then
    return 'C' ;
else
    return 'F' ;
end if ;
end getgrade ;
end pkg_marksheet ;
/
```

The output of the above PL/ SQL code, when compiled is given below:



NOTES

Calling Package Function

To call the function declared in package specification, the reference of package name need to give as given below:

An example to call a package function is as follows:

```
pkg_marksheet.getGrade ('A-08-12');
```

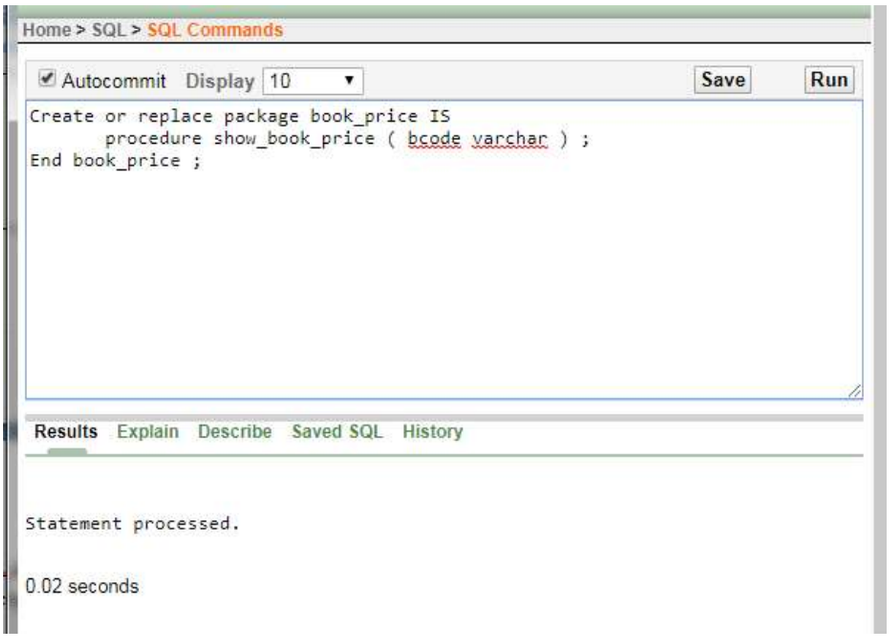
Where, pkg_marksheet is a package name in which a function getGrade is declared which takes a varchar argument A-08-12.

A Package Procedure

The example given below declares a procedure show_book_price which would accept an argument of varchar data type.

Example 57: To create a package procedure.

Step 1:



The above code will create a package with the name book_price. This package contains a procedure named show_book_price. This procedure will accept an argument of varchar type.

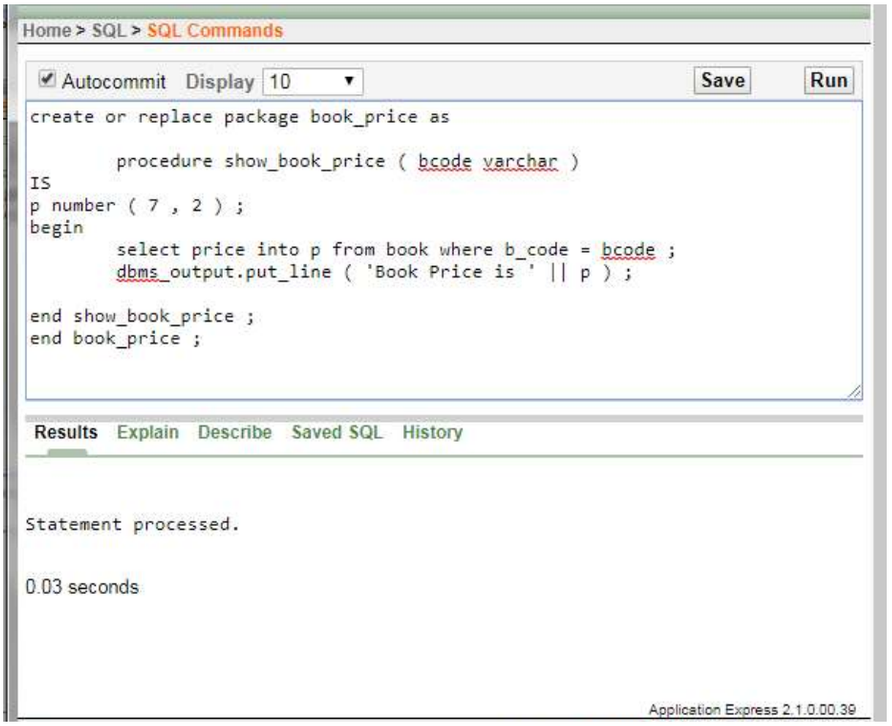
Note: Procedure cannot return any value.

NOTES

The output of the above PL/ SQL code when compiled is given below:

Package created.

Step-2



Save the above program with the any name (let us suppose show_price) and then run it.

The output of the above PL/ SQL code when compiled is given below:

Package body created.

Calling Package Procedure

To call the procedure declared in package specification, the reference of package name need to give as shown below:

The ***Syntax*** to call a package procedure is as follows:

```
Package_name.procedure_name;
```

The example to call a package procedure is as follows :

```
book_price.  show_book_price  ('B003');
```

Where, book_price is a package name in which a procedure show_book_price is declared which takes a varchar argument B003.

NOTES

Reports using functions

A stored function always returns a result and can be called inside an SQL statement just like ordinary SQL function. A function parameter is the equivalent of the in procedure parameter, as functions use the RETURN keyword to determine what is passed back. User-defined functions or stored functions are the stored procedures which have the features of all procedures. They can accept parameters, perform calculations based on data retrieved and return the result to the calling SQL statement, procedure, function or PL/SQL program.

Create a Function

The syntax to create a function is as follows:

```
CREATE OR REPLACE FUNCTION function_name (function_params)
 RETURN return_type IS
Declaration statements
 BEGIN
 Executable statements
RETURN something_of_return_type ;
EXCEPTION
Exception section
 END;
```

Description of the Syntax

CREATE Function:

This is used to create a function, if no other function with the given name exists.

OR REPLACE Function:

OR REPLACE is used to re-create the function if the given function name already exists. If no function exists with the given name, it creates the new function. You can also use OR REPLACE clause to change the definition of an existing function without dropping, re-creating and regranting privileges previously granted on the function to other users. If you redefine a function, then Oracle Database recompiles it.

IS:

It is similar to DECLARE in PL/SQL Blocks. Variables could be declared between IS and BEGIN.

RETURN

Clause Function returns a value. The RETURN clause is used to specify the data type of the return value of the function. Since every function must return a value,

this clause is mandatory to use. The return value can have any data type supported by PL/SQL.

Example 58: Consider table given below, which contains the detailed of accounts of account holders of bank.

NOTES

Table: *Account_holder*

ACC_NO	NAME	TYPE_OF_AC	CONTACT_NO	AC_BALANCE
120040	Tom	Saving	98978800	15620
120040	Merlisa	Saving	98981600	26500
120041	George	Saving	8787700	16560
120041	Smith	Saving	6050234	25500
120042	Loise	Current	6050234	26660
120043	marry	Current	38042342	70080

A stored function is given to return the balance of an account holder. The account number is passed as a parameter in this function.

```
Function: get_balance ()
 /* This is a stored function which returns the total balance of all saving accounts*/
CREATE or replace FUNCTION get_balance ( no IN NUMBER)
 RETURN NUMBER
 IS acc_bal NUMBER ( 11 , 2 ) ;
BEGIN
 SELECT sum ( ac_balance ) INTO acc_bal from account_holder WHERE acc_no = no ;
RETURN ( acc_bal ) ;
END;
/
```

The given function, get_balance () has a parameter of number type to accept the account holder's account number. The acc_bal is a variable in which the balance of the given account holder is stored and returned to the caller program.

Save file

Save the above file with the name account_balance.SQL

Compile Function

NOTES

To execute any stored procedure it is necessary to compile it. To compile a procedure the following command is used:

The syntax is as follows:

```
SQL> @ function_name ;
```

For example,

SQL> @ account_balance ;

Example 59: Based on library information system.

List of tables:

Book_Details

Binding_Details

Category_Details

Borrower_Details

Student_Details

Staff_Details

Student_Details

Shelf_Details

Library Management System (SQL Commands)

Creating table "Book_Details":

```
1. CREATE TABLE Book_Details
2. (
3. ISBN_Codeint PRIMARY KEY,
4. Book_Titlevarchar(100),
5. Language varchar(10),
6. Binding_Idint,
7. No_Copies_Actualint,
8. No_Copies_Currentint,
```

```
 9. Category_idint,
10. Publication_yearint
11. )
```

NOTES

Inserting Some Data in "Book_Details":

```
1. INSERT INTO Book_details
2. VALUES('0006','Programming Concept',
   'English',2,20,15,2,2006);
```

Creating table "Binding_Details":

```
1. CREATE TABLE Binding_details
2. (
3.  Binding_idint PRIMARY KEY,
4.  Binding_Namevarchar(50)
5. )
```

Describe Binding table:

```
Describe binding_details;
```

Inserting Some data in Binding Table:

```
1. INSERT INTO Binding_DetailsVALUES
   (1,'McGraw Hill);
2. INSERT INTO Binding_DetailsVALUES
   (2,'BPB Publication');
```

All Data of Binding Table:

```
1. select *from binding_Details
```

BINDING_ID	BINDING_NAME
1	McGraw Hill
2	BPB Publication

Creating Relationship between Book and Binding Table:

```
1. ALTER TABLE Book_details
2. ADD CONSTRAINT Binding_ID_FK FOREIGN KEY(
   Binding_Id) REFERENCES Binding_Details(Binding_Id);
```

Checking Relationship:

```
1. selectb.Book_Title, e.binding_name
2. fromBook_Detailsb, Binding_Details e
3. whereb.binding_id = e.binding_id;
```

BOOK_TITLE	BINDING_NAME
Introduction to database	McGraw Hill
Programming Concept	BPB Publication

Creating Category Table:

```
1. CREATE TABLE Category_Details
2. (
3.  Category_Idint PRIMARY KEY,
4.  Category_Namevarchar(50)
5. )
```

Inserting some data in Category Table:

```
1. INSERT INTO CATEGORY_DETAILS VALUES
   (1,'Database');
2. INSERT INTO CATEGORY_DETAILS VALUES(
   2,'Programming Language');
```

Building Relationship between Book & Category Table:

```
1. ALTER TABLE Book_details
2. ADD CONSTRAINT Category_Id_FK FOREIGN KEY
   (Category_Id) REFERENCES Category_Details(Category_Id);
```

Checking Relationship:

```
1. Select  b.Book_Title,e.Category_Name
2. From  Book_Detailsb,Category_Details e
3. whereb.binding_id = e.Category_id;
```

BOOK_TITLE	CATEGORY_NAME
Introduction to database	Database
Programming Concept	Programming Language

NOTES

NOTES

Creating Borrower Table:

```
1. CREATE TABLE Borrower_Details
2. (
3.  Borrower_Idint PRIMARY KEY,
4.  Book_Idint,
5.  Borrowed_From date,
6.  Borrowed_TO date,
7.  Actual_Return_Date date,
8.  Issued_byint
9. )
```

Inserting some data in Category Table:

```
1. Insert into BORROWER_DETAILS VALUES(1,0004,'
   01-Aug-2014','7-Aug-2014','7-Aug-2014',1)
2. Insert into BORROWER_DETAILS VALUES(2,6,'02-
   Aug-2014','8-Aug-2014',NULL,1)
```

Building Relation between Book & Borrower Table:

```
1. ALTER TABLE Borrower_details ADD
   CONSTRAINT Book_Id_FK FOREIGN KEY(Book_Id)
   REFERENCES Book_Details(ISBN_Code);
```

Checking Relationship:

```
1. Select  Borrower_Details.Borrower_id,
   Book_Details.Book_title
2. From  Borrower_Details,Book_Details
3. Where  Borrower_Details.book_id=Book_Details.
   ISBN_Code
```

BORROWER_ID	BOOK_TITLE
1	Introduction to database
2	Programming Concept

```
1. ALTER TABLE Borrower_Details
2. ADD CONSTRAINT Issued_by_FK FOREIGN KEY(
   Issued_by) REFERENCES Staff_Details(Staff_Id);
```

Creating Staff Table:

```
1. CREATE TABLE Staff_Details
2. (
3.  Staff_Idint PRIMARY KEY,
4.  Staff_Namevarchar(50),
5.  Password varchar(16),
6.  Is_Adminbinary_float,
7.  Designation varchar(20)
8. )
```

Inserting some data in Staff Table:

```
1. Insert into STAFF_DETAILS values
   (1,'Tarek Hossain','1234asd',0,'Lib_mgr');
2. Insert into STAFF_DETAILS values
   (2,'Md.Kishor Morol','iloveyou',0,'Lib_clr');
```

All Data of Staff table:

```
1. select * from staff_details
```

STAFF_ID	STAFF_NAME	PASSWORD	IS_ADMIN	DESIGNATION
1	Tarek Hossain	1234asd	1.0E+000	Lib_mgr
2	Md.Kishor Morol	iloveyou	0	Lib_clr

Creating Student Table:

```
1. Create TABLE Student_Details
2. (
3. Student_Idvarchar(10) PRIMARY KEY,
4. Student_Namevarchar(50),
5. Sex Varchar(20),
```

NOTES

NOTES

```
 6. Date_Of_Birth date,
 7. Borrower_Idint,
 8. Department varchar(10),
 9. contact_Numbervarchar(11)
10. )
```

Inserting Some Data in Student Table:

```
1. Insert into STUDENT_DETAILS values ('13-23059-
   1','Ahmed,Ali','Male','05-Oct-
   1995',1,'CSSE','01681849871');
2. Insert into STUDENT_DETAILS values ('13-23301-
   1','MOrol MD.Kishor','Male','03-Jan-
   1994',2,'CSE','01723476554');
```

All Data of Student Table:

```
1. select *from student_details;
```

STUDENT_ID	STUDENT_NAME	SEX	DATE_OF_BIRTH	BORROWER_ID	DEPARTMENT	CONTACT_NUMBER
13-23059-1	Ahmed,Ali	Male	05-OCT-95	1	CSSE	01681849871
13-23301-1	MOrol MD.Kishor	Male	03-JAN-94	2	CSE	01723476554

Building Relationship between student and Borrower table:

```
1. ALTER TABLE student_details
2. ADD CONSTRAINT borrower_id_FK FOREIGN KEY(
   Borrower_Id) REFERENCES Borrower_Details(Borrower_Id);
```

Checking Full Relationship:

```
1. select student.student_id, student.student
   _name, book.Book_Title, staff.staff_name, b.Borrowed_To
2. fromstudent_Detailsstudent, Staff_
   Detailsstaff, Borrower_Detailsb, book_details book
3. wherestudent.Borrower_id = b.Borrower_i
   d and book.ISBN_Code = b.book_id and b.Issued_by = staff.Staff_id;
```

STUDENT_ID	STUDENT_NAME	BOOK_TITLE	STAFF_NAME	BORROWED_TO
13-23059-1	Ahmed,Ali	Introduction to database	Tarek Hossain	07-AUG-14
13-23301-1	MOrol MD.Kishor	Programming Concept	Tarek Hossain	08-AUG-14

Adding Shelf Table:

```
1. Create Table Shelf_Details
2. (
3.  Shelf_idint PRIMARY KEY,
4.  Shelf_Noint,
5.  Floor_Noint
6. );
```

Inserting Some Data from Shelf Table:

```
1. Insert into Shelf_DetailsValues(1, 1, 1);
2. Insert into Shelf_DetailsValues (2, 2, 10001);
3. Insert into Shelf_DetailsValues (3, 1, 10002);
```

All Data in Shelf Table:

```
1. select*from Shelf_Details;
```

SHELF_ID	SHELF_NO	FLOOR_NO
1	1	1
2	2	10001
3	1	10002

Adding Relationship between Shelf and Book Table:

```
1. ALTER TABLE Book_Details
2. ADD(Shelf_Idint);
3. UPDATE Book_Details set Shelf_Id = 1
4. where ISBN_CODE = 4;
5. UPDATE Book_Details set Shelf_Id = 2
6. where ISBN_CODE = 6;
7. ALTER TABLE Book_Details
8. ADD CONSTRAINT Shelf_Id_FK FOREIGN KEY
   (Shelf_Id) REFERENCES Shelf_Details(Shelf_Id);
```

NOTES

NOTES

Combine all Relationship:

```
1. select student.student_id, student.student
   _name, book.Book_Title, staff.staff_name, b.Borrowed_To,
   shelf.shelf_No
2. fromstudent_Detailsstudent,
   Staff_Detailsstaff,  Borrower_Detailsb,
   book_detailsbook, Shelf_Details shelf
3. wherestudent.Borrower_id = b.Borrower_
   id and book.ISBN_Code = b.book_id and b.Issued_by
   = staff.Staff_id and book.Shelf_Id = shelf.Shelf_Id;
```