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

B.Sc. [Computer Science]

V - Semester

130 54

LAB: RELATIONAL DATABASE MANAGEMENT SYSTEMS

Author:

Dr. Kavita Saini, Associate Professor, Galgotias University, Greater Noida

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

Work Order No.AU/DDE/DE12-27/Preparation and Printing of Course Materials/2020 Dated 12.08.2020 Copies.......

LAB: RELATIONAL DATABASE MANAGEMENT SYSTEMS

SYLLABI

BLOCK 1: TABLE MANIPULATION
Table: Creation, Renaming a Table, Copying Another Table, Dropping a Table
Table Description: Describing Table Definitions, Modifying Tables

BLOCK 2: SQL QUERIES AND SUB QUERIES
SQL Queries: Queries, Sub Queries, and Aggregate Functions
DDL: Experiments Using Database DDL SQL Statements
DML: Experiment Using Database DML SQL Statements
DCL: Experiment Using Database DCL SQL Statements

BLOCK 3: INDEX AND VIEW
Index: Experiment Using Database Index Creation, Renaming an Index, Copying Another Index, Dropping an Index
Views: Create Views, Partition and Locks

BLOCK 4: EXCEPTION HANDLING AND PL/SQL
Exception Handling: PL/SQL Procedure for Application Using Exception Handling
Cursor: PL/SQL Procedure for Application Using Cursors
Trigger: PL/SQL Procedure for Application Using Triggers
Package: PL/SQL Procedure for Application Using Package
Reports: DBMS Programs to Prepare Report Using Functions

BLOCK 5: APPLICATION DEVELOPMENT
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

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 Edgar F. Codd, who was an English computer scientist. Edgar F. Codd, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases and relational database management systems. He made other valuable contributions to computer science, but the relational model, a very influential general theory of data management, remains his most mentioned, analysed and celebrated achievement.

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 'Laboratory Manuals' has become extremely important.

This laboratory manual, *RELATIONAL DATABASE MANAGEMENT SYSTEMS*, is intended for the students of undergraduate courses in the subject of RDBMS. This laboratory manual typically contains 'Practical/ Laboratory Sessions' related to RDBMS, covering various significant topics on the subject to enhance the understanding. This laboratory manual will help the students to understand the concepts, such as data normalization, link between tables by means of foreign keys and other relevant database concepts, menu-driven query processing and reports, the SQL commands, the cursor, triggers, and packages. In addition, the students will be able to write SQL queries, PL/SQL statements and the database applications using SQL. Students are advised to thoroughly go through this laboratory manual rather than only topics mentioned in the syllabus as practical aspects are the key to understand the conceptual visualization of theoretical aspects covered in the textbooks.

RELATIONAL DATABASE MANAGEMENT SYSTEMS

NOTES

A **Relational 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 a software that is used to maintain data security and data integrity in a structured database.

RDBMS helps in maintaining and retrieving data in different forms. There are various tools available for RDBMS, such as Oracle, Ingres, Sybase, Microsoft SQL Server, MS-Access, IBM-Db-2, MySQL.

Introduction to Oracle

Oracle is a secure portable and powerful database management system of Oracle Corporation. Oracle database is also termed as **Oracle Database**. It is compatible and connectable with almost all operating systems and machines.

Oracle database is based on relational data model and a non-procedural language called Structure Query Language (SQL). Oracle database is a tool that supports storing managing and organization the data.

Structured Query Language (SQL)

SQL is query language used for all database relation management systems. It is a standard language for all RDBMS's. It could be classified into various types where every sub-variety plays its own role and different purpose SQL commands which can further be classified as:-

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

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.

NOTES

Data Type	Size
Date	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.

BLOCK 1: TABLE MANIPULATION

This block will cover the following topics:

1. *Table Creation, Renaming a Table, Copying Another Table, Dropping a Table.*
2. *Table Description: Describing Table Definitions, Modifying Tables.*

Table

A table is represented in two dimensional structure containing rows and columns. It contains interrelated data, for example, an employee table contains data about employees only, i.e., Emp_ID, name, designation, etc. A table is also termed as a relation. A table is depicted in following table:

Emp_ID	Name	Designation	Salary
5001	Tom	Sr. Programmer	38,000
5002	Merlisa	Proj. Leader	60,000
5003	George	Programmer	26,000

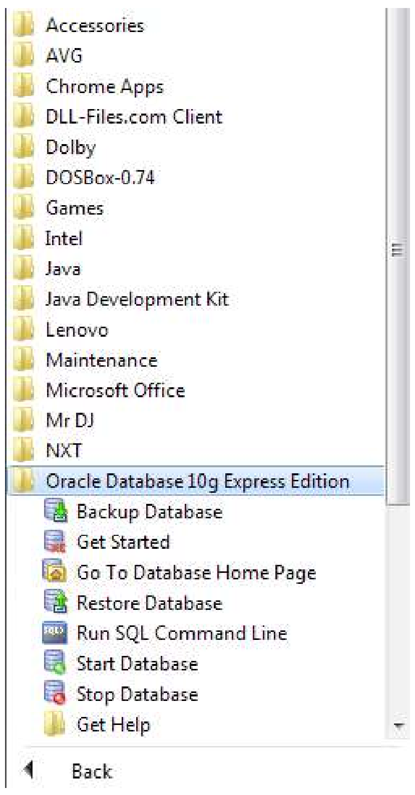
Table - Employee

Getting Started with SQL:

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

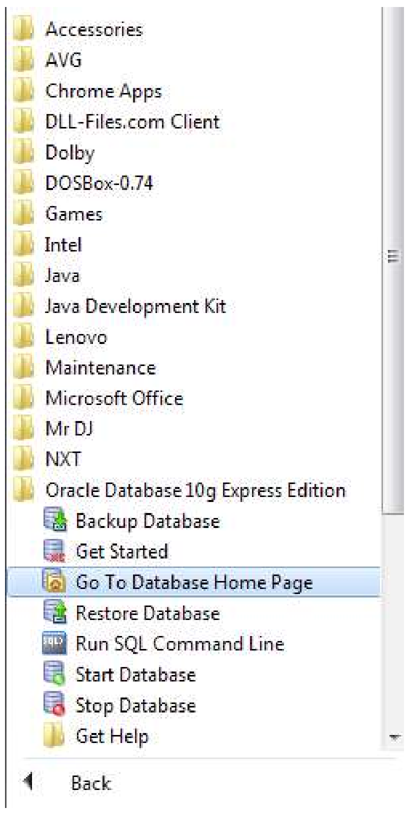
1. Click on Start Button.
2. Point on All Programs.
3. Point on Oracle Database 10g Express Edition.
4. Click on Go to Database Home Page.

NOTES



Click on **Go to Database Home Page**

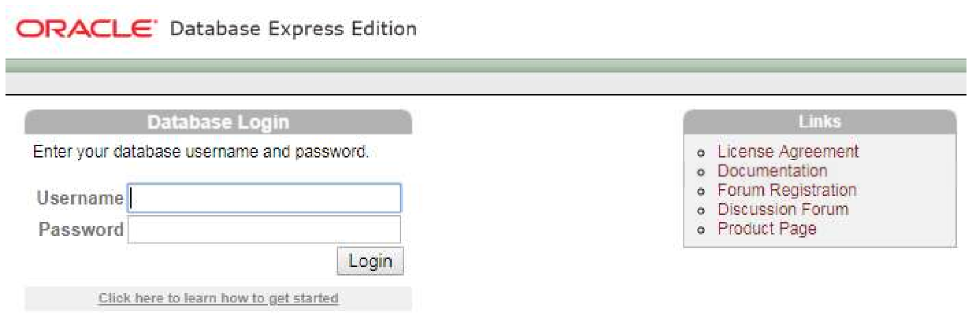
The following Screen will appear:



Note:

Oracle user name and password may be different and need to be verified in lab. In this manual, User name is Demo and Password is Demo.

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


ORACLE Database Express Edition

Database Login

Enter your database username and password.

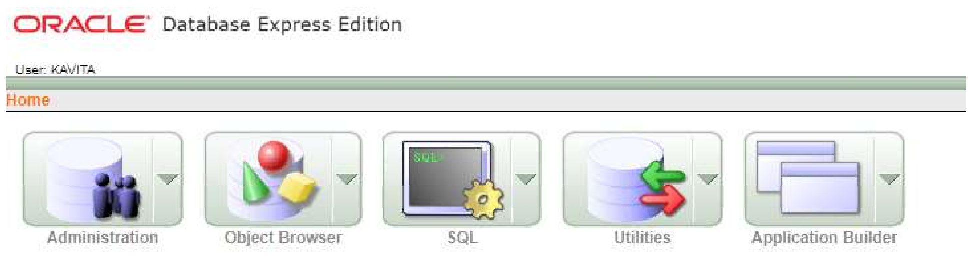
Username

Password

Login

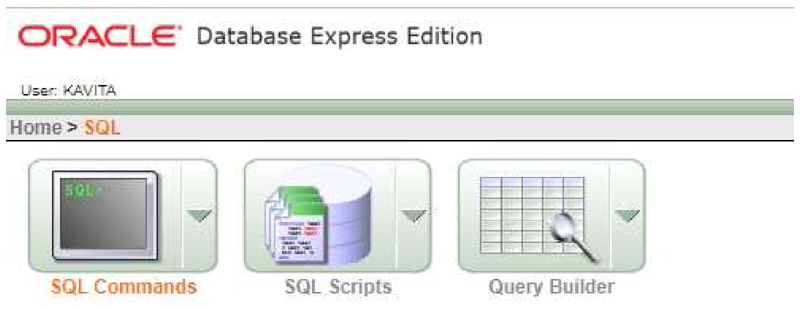
Click here to learn how to get started

Enter the User name and Password as created during installation. The following screen will appear. In this screen click on **SQL**.


ORACLE Database Express Edition

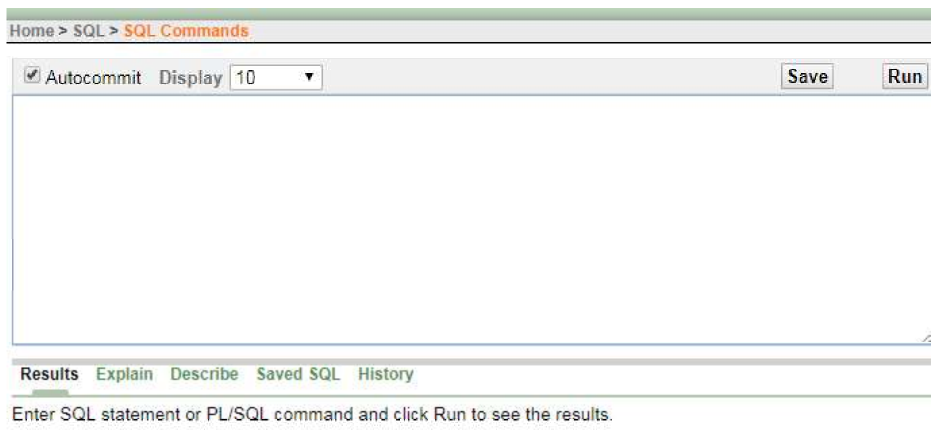
User: KAVITA

After clicking on SQL following screen will appear. Click on **SQL Command** to go to SQL command window.


ORACLE Database Express Edition

User: KAVITA

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


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

Enter SQL statement or PL/SQL command and click Run to see the results.

Creating a Table

This is a Data Definition Language (DDL) command that is used to define the structure of any table. In a table structure you can define various fields, their data types and constraints as per the requirements.

Syntax:

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

Example:

1. Create a table Course.

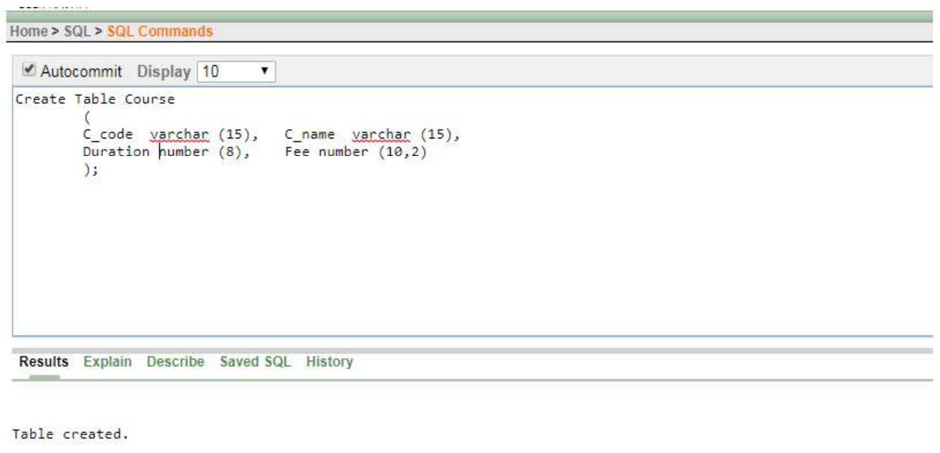
Column Name	Data Type	Size
C_Code	varchar2	15
C_Name	varchar2	15
Duration	number	8
Fee	number	10, 2

The SQL command to create table is as follows:

```
Create Table Course
(
C_Code varchar (15),  C_Name varchar (15),
   Duration number (8),    Fee number (10, 2)
   );
```

NOTES

The above command will create table Course and Oracle will prompt a message as shown below:


Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History

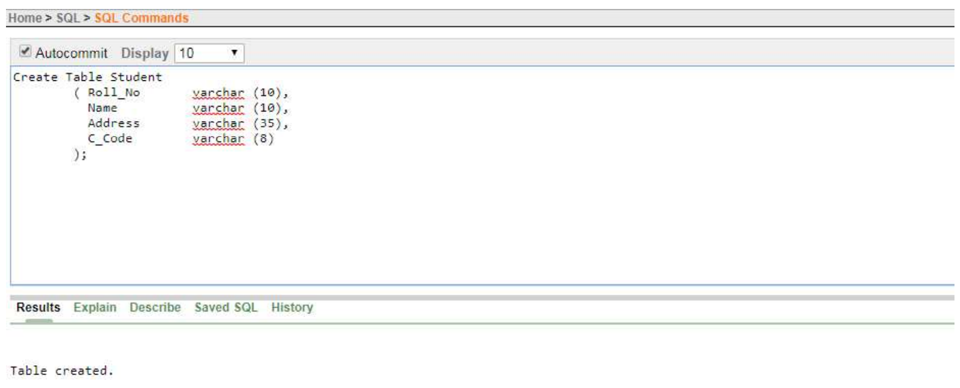
```
Create Table Course
        (
        C_code  varchar (15),   C_name  varchar (15),
        Duration number (8),    Fee number (10,2)
        );
```

```
Table created.
```

Example-2

Create a table Student

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


Home > SQL > SQL Commands
Autocommit Display 10
Results Explain Describe Saved SQL History

```
Create Table Student
        ( Roll_No       varchar (10),
          Name          varchar (10),
          Address       varchar (35),
          C_Code        varchar (8)
        );
```

```
Table created.
```

The above command will create a table structure to store student's information. Where Roll_No, Name, Address and C_Code are the field names and varchar is a data type.

Rename Tables

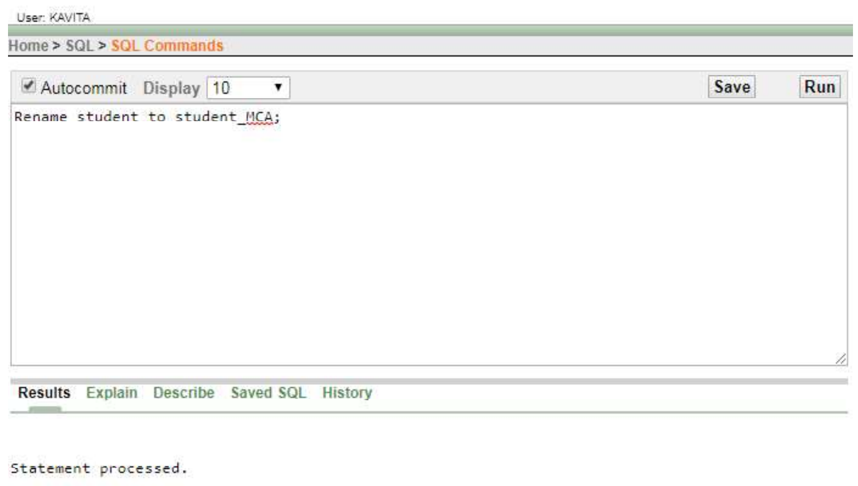
To rename table, you can use Rename command.

The **Syntax** for **Alter** command:

```
Rename old_table_name to new_table_name;
```

NOTES

Example:


User: KAVITA
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Rename student to student_MCA;
```

```
Statement processed.
```

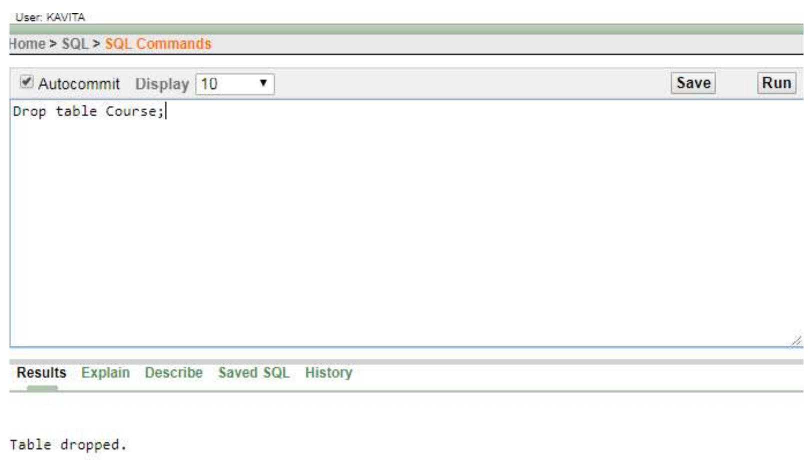
Dropping a Table

When a SQL table is no more required, you can get rid of the table by using DROP command. Drop table is a Data Definition Language (DDL). Drop command is used to drop any object, such as table, index, view, package and function.

The **Syntax** to drop a table:

```
Drop table <table_name >
```

Example:


User: KAVITA
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Drop table Course;
```

```
Table dropped.
```

The above command will remove the course tables.

TRUNCATE:

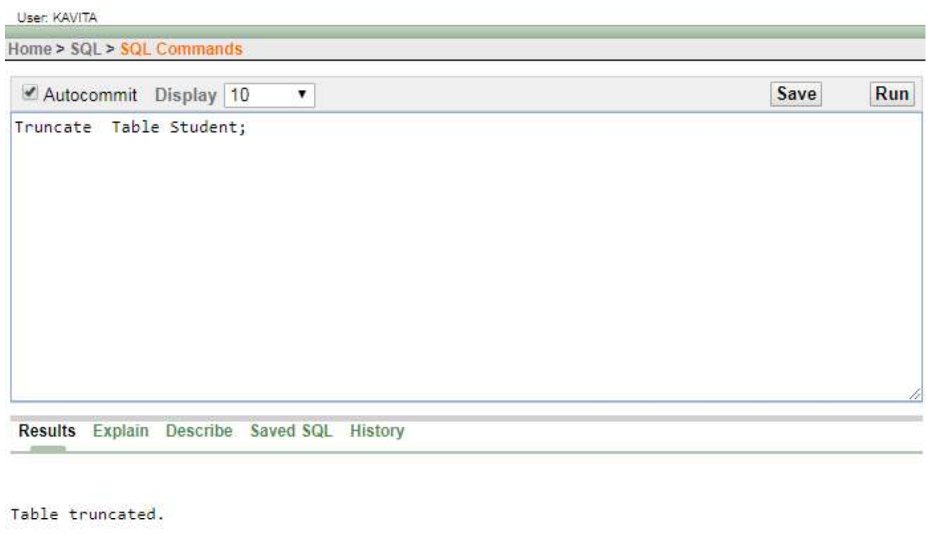
This command will remove the data permanently but the structure will not be removed.

Syntax: `Truncate Table <Table name>;`

Example: `Truncate Table Student;`

Difference between Truncate and Delete:-

- By using Truncate command, data will be removed permanently and will not get back whereas by using Delete command, data will be removed temporarily and get back by using Roll Back command.
- By using Delete command, data will be removed based on the condition **'where as'** or by using Truncate command if there is no condition.
- Truncate is a DDL command and Delete is a DML command.



Display: 10

```
Truncate  Table Student;
```

```
Table truncated.
```

Try yourself

1. Write SQL Queries to create following tables:

Table: Student:

SID	Name	Login	Age	gpa

Table: Enrolled:

SID	CID	grade

2. Write SQL Queries to add address column in student tables.

3. Write SQL Queries to drop address column in student tables.

4. Write SQL Queries to delete student and enrolled tables.

5. Delete the student table.

Describing a Table:

To see the table structure Oracle provides command Describe (or Desc).

The **Syntax** to describe a table:

```
Describe  <table_name>
```

Or

```
Desc  <table_name>
```

Example

```
Describe  Course
Or  Desc  Course
```

Command to Modifying Table:

Table modification may be required in future if requirements gets change after creating table. Modification includes:

- *Add a New Column*
- *Change Data Type of an Existing Column*
- *Modify the Length of on Existing Column*
- *Delete any Column*

There are certain points to remember while modifying table:

- 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.

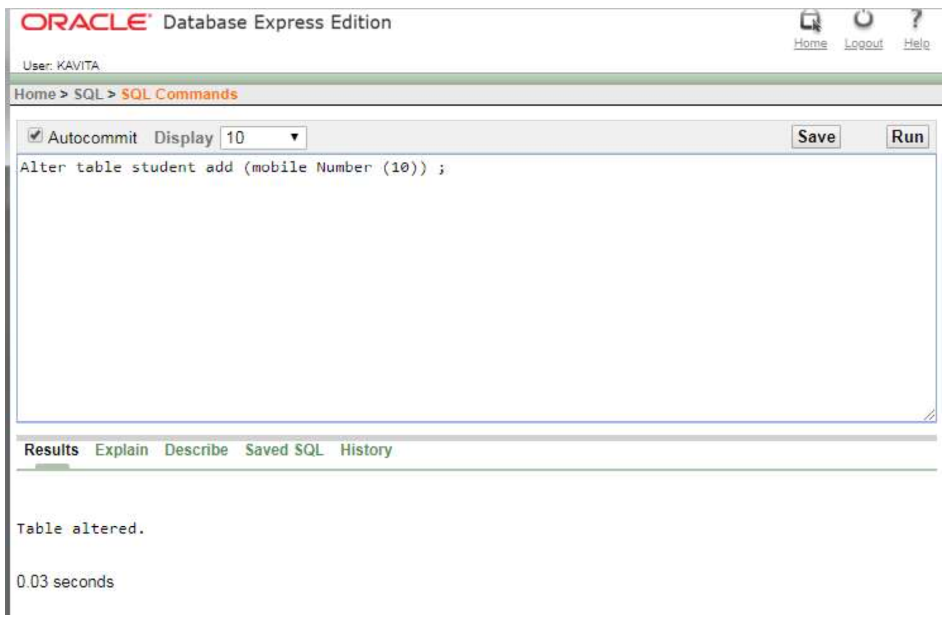
Add a New Column:

The **Syntax** for alter command:

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

Example:

```
Alter table STUDENT add (MOBILE Number (10));
```



```
Alter table student add (mobile Number (10)) ;
```

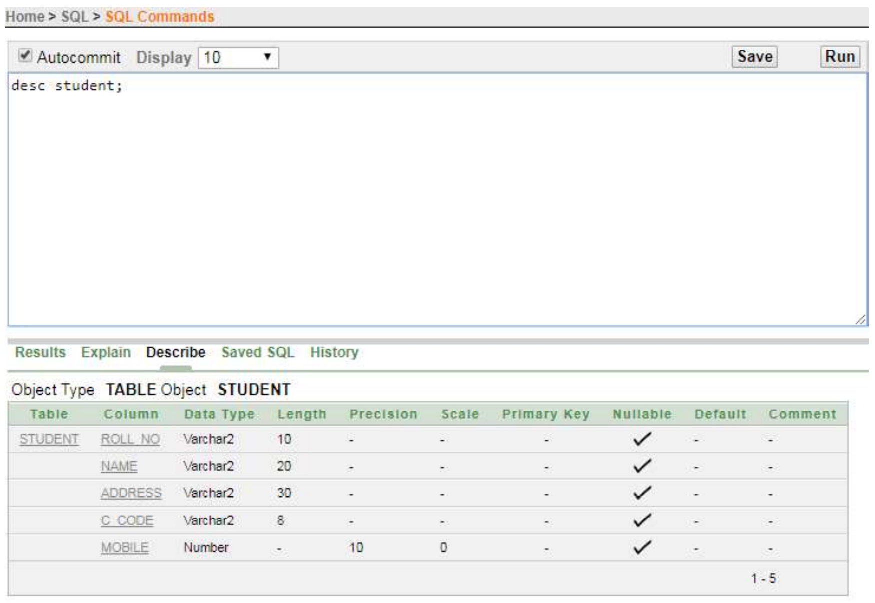
```
Table altered.

0.03 seconds
```

NOTES

NOTES

The above command will add a new column, MOBILE in STUDENT table. You could see the new structure of STUDENT table:

```
desc STUDENT;
```


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run

```
desc student;
```

Results Explain Describe Saved SQL History

Object Type TABLE Object STUDENT

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENT	ROLL_NO	Varchar2	10	-	-	-	✓	-	-
	NAME	Varchar2	20	-	-	-	✓	-	-
	ADDRESS	Varchar2	30	-	-	-	✓	-	-
	C_CODE	Varchar2	8	-	-	-	✓	-	-
	MOBILE	Number	-	10	0	-	✓	-	-

1 - 5

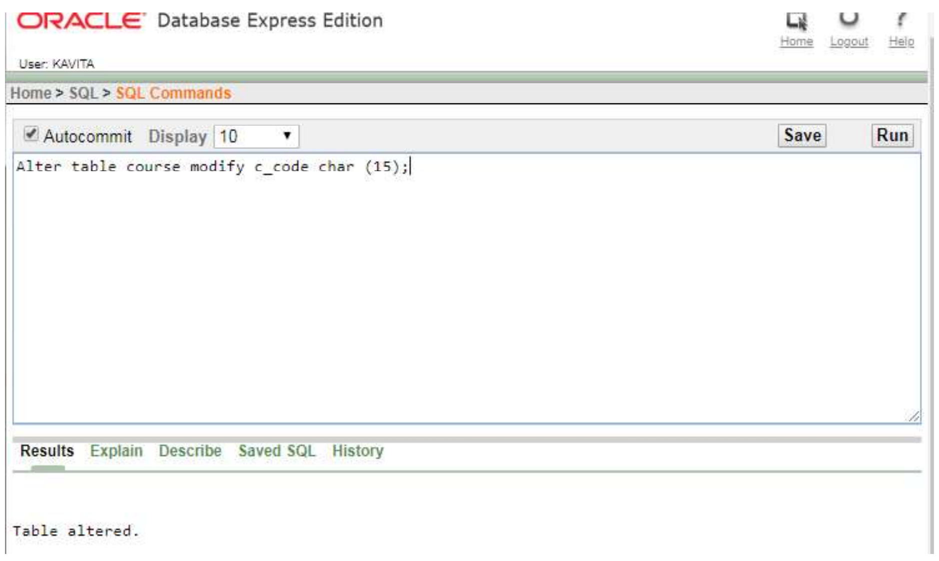
To change data type of an existing column

The **Syntax** for alter command:

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

Example:

```
Alter table course modify c_code char (15);
```


ORACLE Database Express Edition
Home
Logout
Help
User: KAVITA
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run

```
Alter table course modify c_code char (15);
```

Results Explain Describe Saved SQL History

```
Table altered.
```

The above command will change the data type of c_code field from varchar to char.

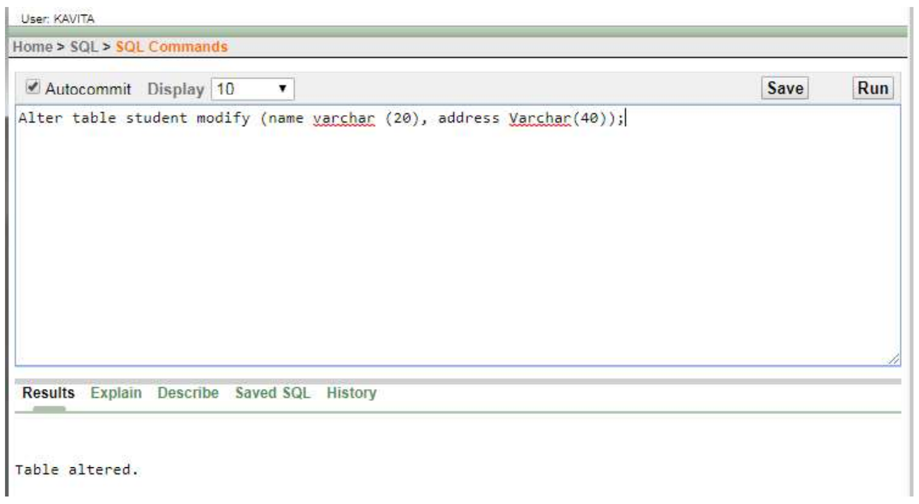
To modify the length of an existing column:

The **Syntax** for alter command:

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

Example:

```
Alter table student modify (name varchar (20), address
varchar (40));
```


User: KAVITA
Home > SQL > SQL Commands
Autocommit Display 10 Save Run

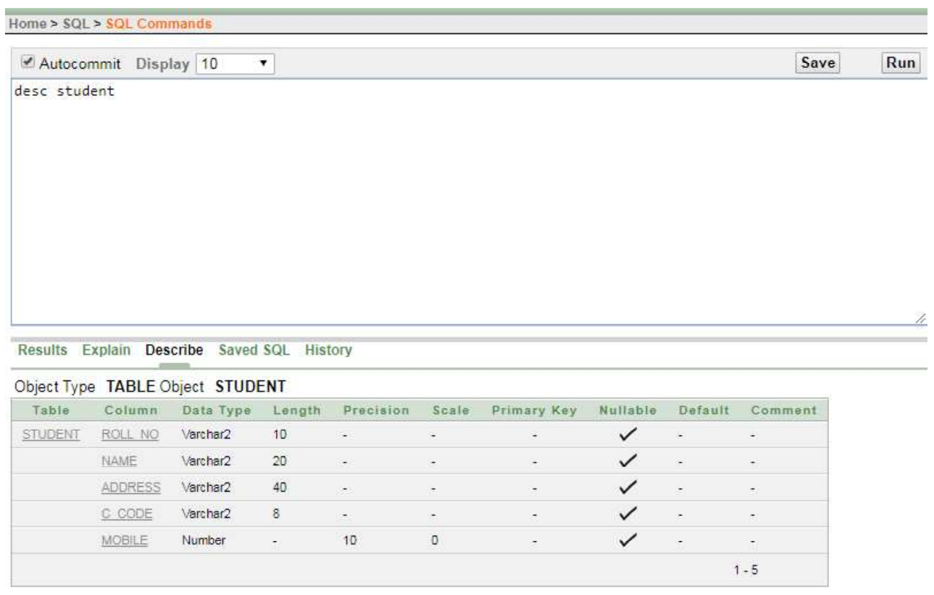
```
Alter table student modify (name varchar (20), address Varchar(40));
```

Results Explain Describe Saved SQL History

```
Table altered.
```

The above command will change the length of name column from 15 to 20 and address from 35 to 40.

After altering student table structure will look like:


Home > SQL > SQL Commands
Autocommit Display 10 Save Run

```
desc student
```

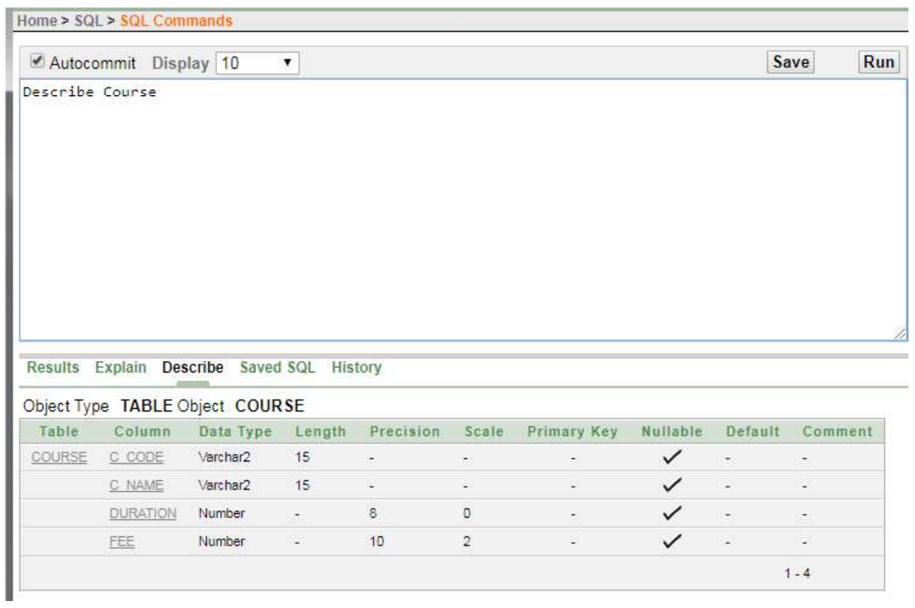
Results Explain Describe Saved SQL History

Object Type TABLE Object STUDENT

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENT	ROLL_NO	Varchar2	10	-	-	-	✓	-	-
	NAME	Varchar2	20	-	-	-	✓	-	-
	ADDRESS	Varchar2	40	-	-	-	✓	-	-
	C_CODE	Varchar2	8	-	-	-	✓	-	-
	MOBILE	Number	-	10	0	-	✓	-	-

1 - 5

NOTES


Home > SQL > SQL Commands

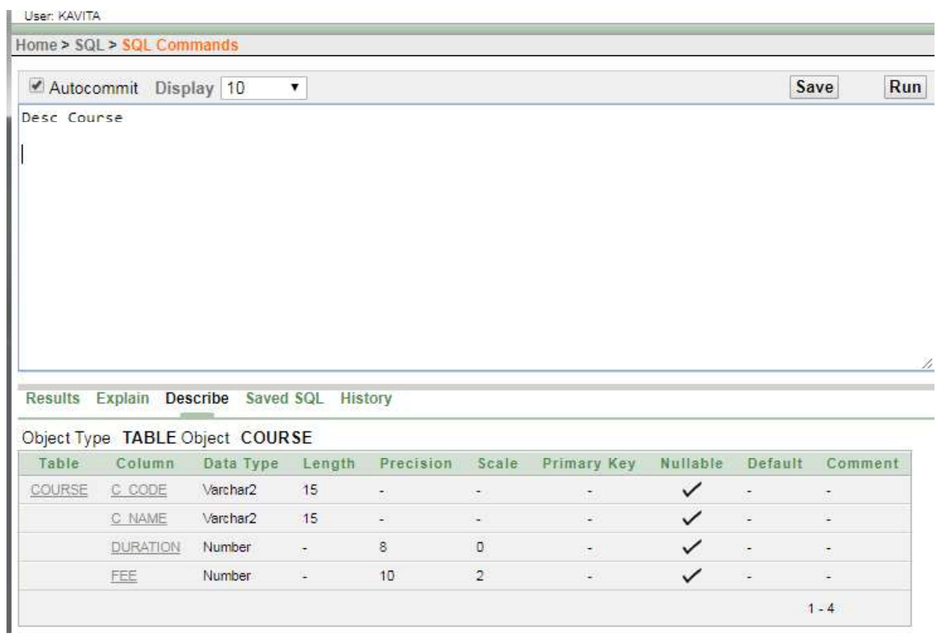
```
Describe Course
```

Object Type TABLE Object COURSE

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
COURSE	C_CODE	Varchar2	15	-	-	-	✓	-	-
	C_NAME	Varchar2	15	-	-	-	✓	-	-
	DURATION	Number	-	8	0	-	✓	-	-
	FEE	Number	-	10	2	-	✓	-	-

1 - 4

The above command will describe the structure of COURSE table as shown below:


User: KAVITA

Home > SQL > SQL Commands

```
Desc Course
```

Object Type TABLE Object COURSE

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
COURSE	C_CODE	Varchar2	15	-	-	-	✓	-	-
	C_NAME	Varchar2	15	-	-	-	✓	-	-
	DURATION	Number	-	8	0	-	✓	-	-
	FEE	Number	-	10	2	-	✓	-	-

1 - 4

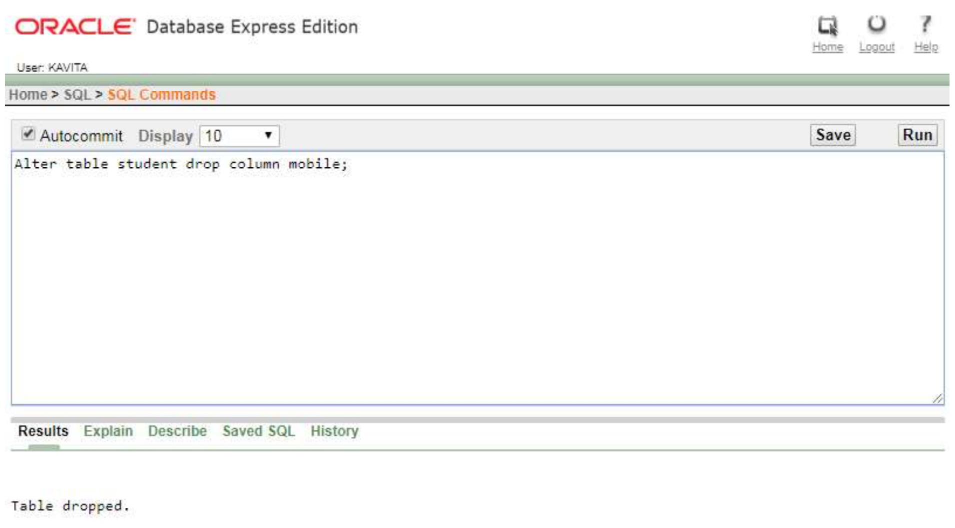
To Delete any Column

The **Syntax** for alter command:

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

Example

```
Alter table student drop column mobile;
```



```
Alter table student drop column mobile;
```

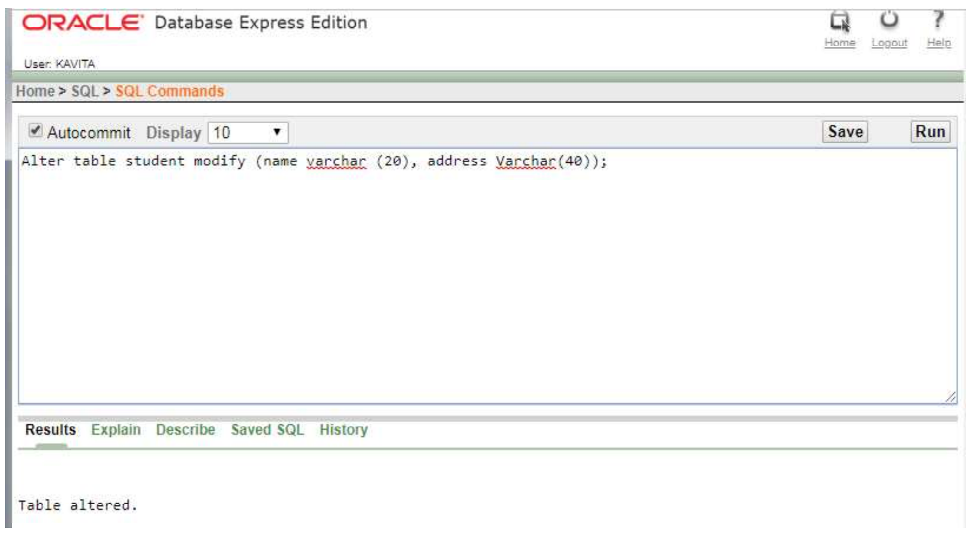
```
Table dropped.
```

The above command will delete the column mobile form students table.

The **Syntax** for alter command:

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

Example:



```
Alter table student modify (name varchar (20), address Varchar(40));
```

```
Table altered.
```

BLOCK 2: SQL QUERIES AND SUB QUERIES

NOTES

This block will discuss about the following topics:

- SQL Queries: Queries, Sub Queries, and aggregate functions
- DDL: Experiments using database DDL SQL statements
- DML: Experiment using database DML SQL statements
- DCL: Experiment using database DCL SQL statements

SQL Queries:

DDL: EXPERIMENTS USING DDL SQL STATEMENTS

Data Definition Language (DDL)

Data definition language commands are used for creating, modifying and removing database objects wherein the object could be a table cursor, view trigger or a sequence.

Data definition language commands are:

- CREATE
- ALTER
- DROP

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 either 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 help in maintaining data integrity which is one of the rules defined by E.F. Codd.

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 of 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.

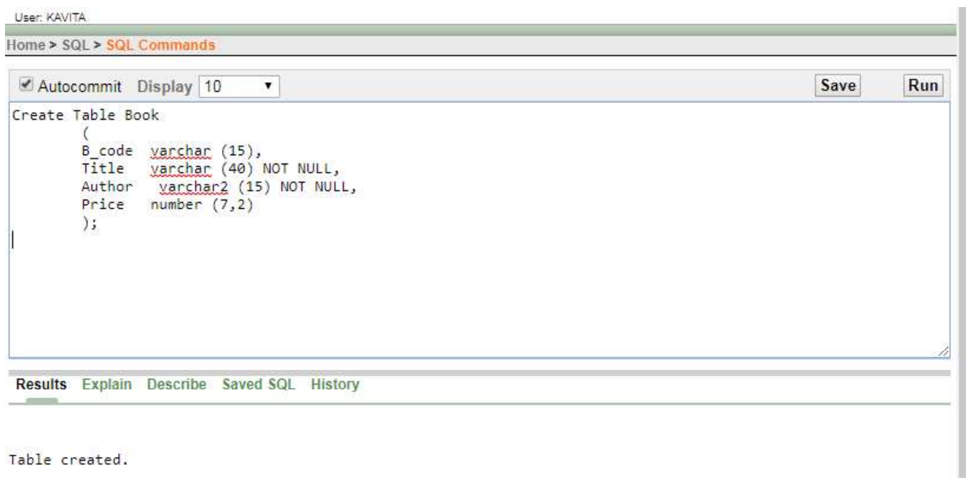
The NOT NULL constraint ensures that the value in column is not missing (NULL). This constraint enforce user to enter data into a specified column. A column with this constraint could have duplicate values but could not be null or empty.

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 *), those fields are the fields with the NOT NULL constraint.

The following example creates 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 as follows:

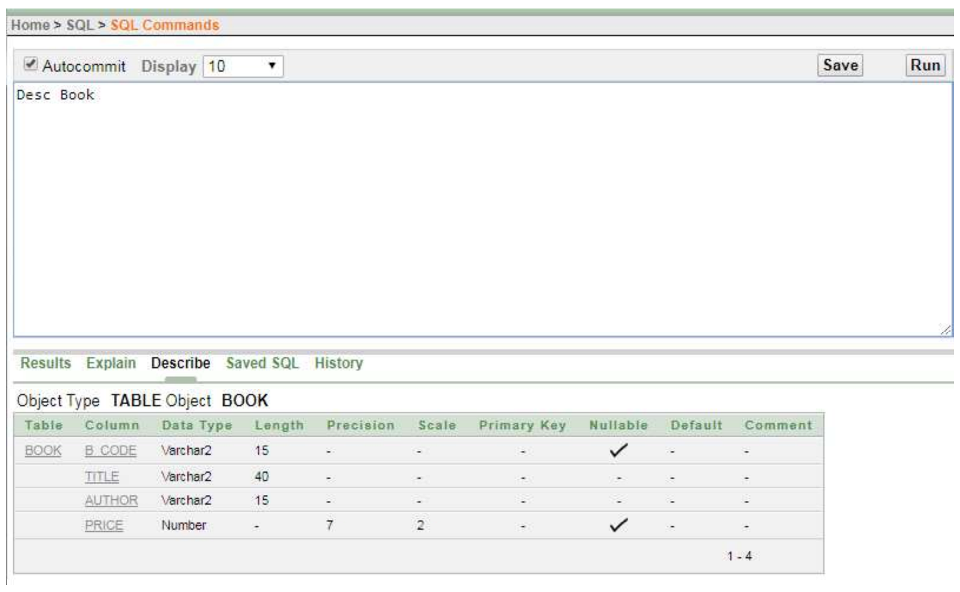

User: KAVITA
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create Table Book
       (
       B_code  varchar (15),
       Title   varchar (40) NOT NULL,
       Author   varchar2 (15) NOT NULL,
       Price   number (7,2)
       );
```

```
Table created.
```

NOTES

Book Table Structure;



Home > SQL > SQL Commands

Autocommit: ✓ Display: 10

```
Desc Book
```

Object Type TABLE Object BOOK

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
BOOK	B_CODE	Varchar2	15	-	-	-	✓	-	-
	TITLE	Varchar2	40	-	-	-	-	-	-
	AUTHOR	Varchar2	15	-	-	-	-	-	-
	PRICE	Number	-	7	2	-	✓	-	-

1 - 4

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. NOT NULL constraints could be set at column level only.

UNIQUE Constraint:

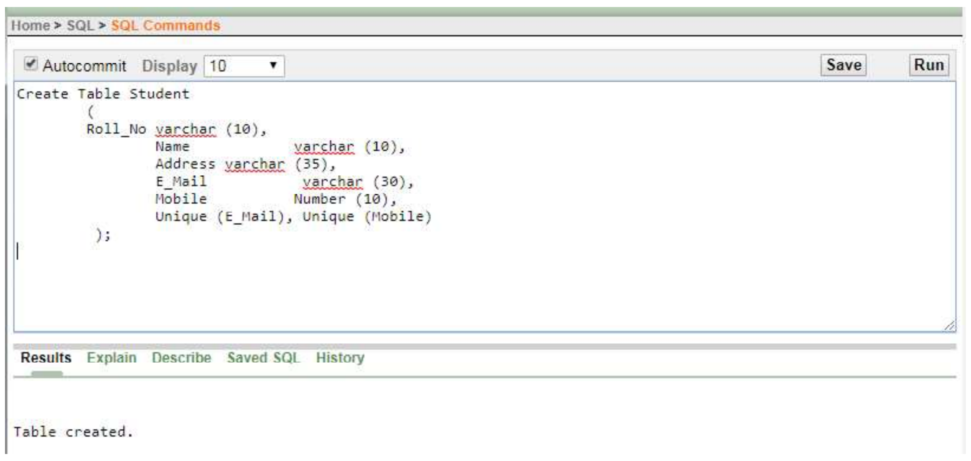
Sometimes, it is required that column must have unique values only. The unique constraint ensures that data to the specified column data is not duplicate but it could contain the NULL values. Let us take an example of contact number and E-Mail ID; it is not necessary that every student has a contact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

The following example creates 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

NOTES

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


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create Table Student
        (
        Roll_No varchar (10),
                Name            varchar (10),
                Address varchar (35),
                E_Mail           varchar (30),
                Mobile          Number (10),
                Unique (E_Mail), Unique (Mobile)
         );
```

```
Table created.
```

In the above example UNIQUE constraints are set at table level.

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).

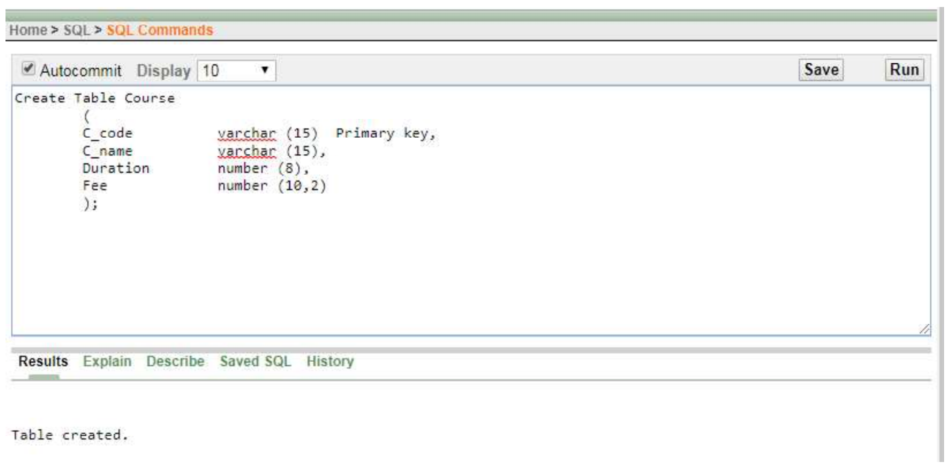
Example-1

The following example creates a table Course with 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 as follows:


Home > SQL > SQL Commands
Autocommit Display 10
Save Run
Results Explain Describe Saved SQL History

```
Create Table Course
        (
        C_code          varchar (15)  Primary key,
        C_name          varchar (15),
        Duration        number (8),
        Fee             number (10,2)
        );
```

```
Table created.
```

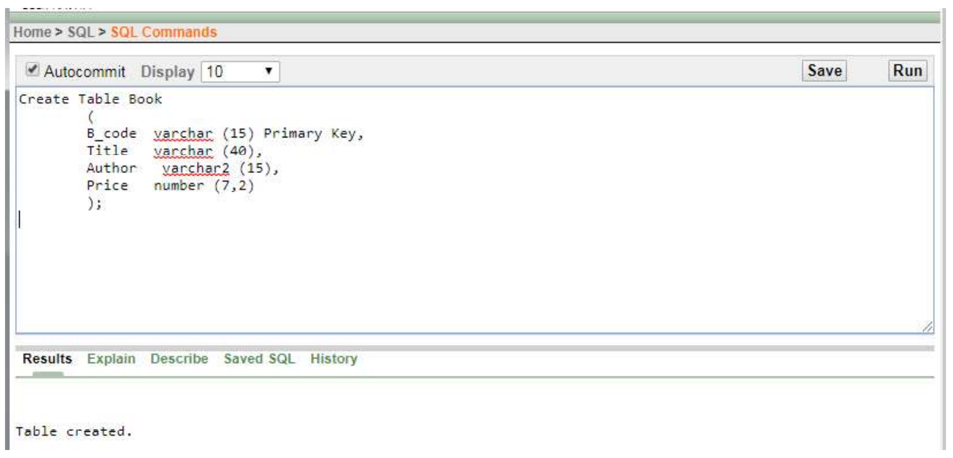
The above command will create table course which contains a Primary Key field course code. Here, PRIMARY KEY constraint will enforce the end user to enter Unique and Not Null values only.

Example-2

The following example creates 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:


Home > SQL > SQL Commands
Autocommit Display 10
Save Run
Results Explain Describe Saved SQL History

```
Create Table Book
        (
        B_code  varchar (15) Primary Key,
        Title   varchar (40),
        Author   varchar2 (15),
        Price   number (7,2)
        );
```

```
Table created.
```

The above command will create table Book which contains a Primary Key field Book Code. This constraint is required to have unique and not null book code in a library. * A table can have only and only one Primary Key.

FOREIGN KEY Constraint or REFERENCE KEY Constraint:

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.

A Course and Book tables created in the PRIMARY KEY constraint section have the Primary Key columns C_code and B_code respectively. These columns could 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 Key's column values could be duplicate or not null.
- Reference Key's column can have the same values as stored in Primary Key column.

Let us suppose that students in any University could be enrolled in the course which are offered by that 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.

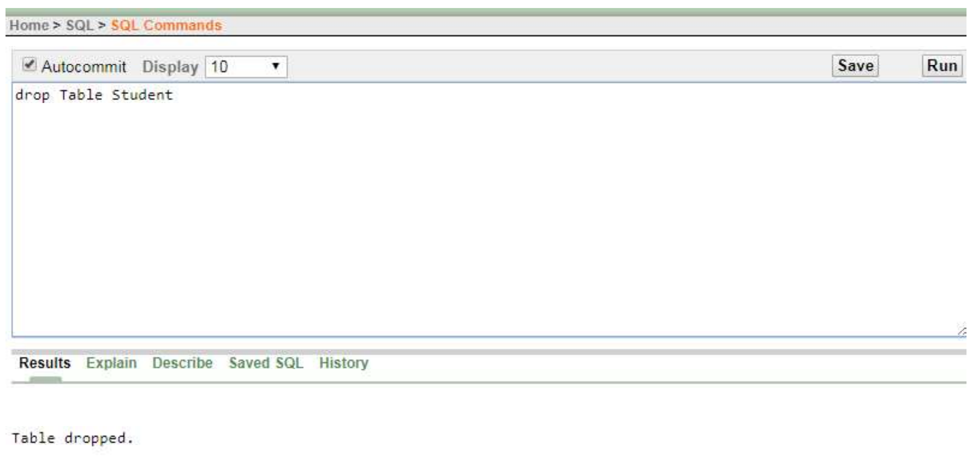
The following example creates a table student with the REFERENCE KEY constraint with the structure as shown below:

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

NOTES

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

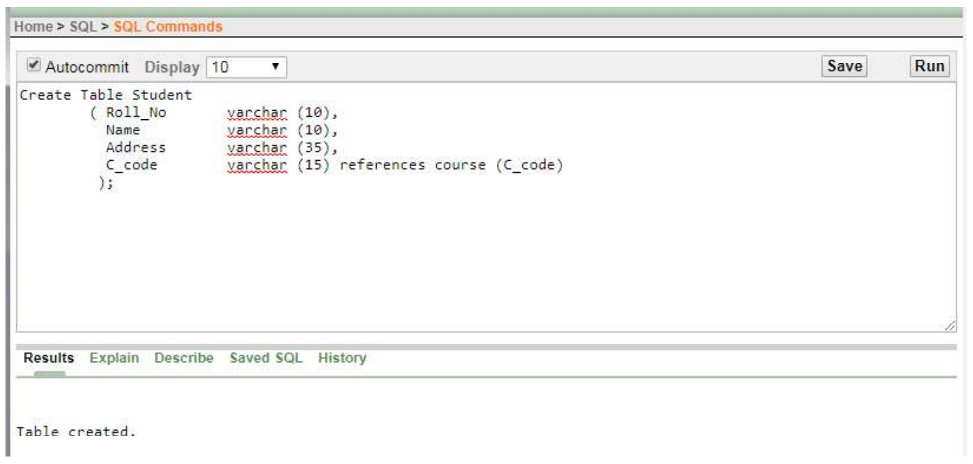
Note: `drop student table`


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
drop Table Student
```

```
Table dropped.
```

Now create Student table again with Reference Key as shown below:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create Table Student
        ( Roll_No       varchar (10),
          Name          varchar (10),
          Address       varchar (35),
          C_code        varchar (15) references course (C_code)
         );
```

```
Table created.
```

The above command will create Table Student which contains a Reference Key column Course Code. This column will table reference of Course Code of Course table when record in Student table will be inserted or updated by the user. * 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.

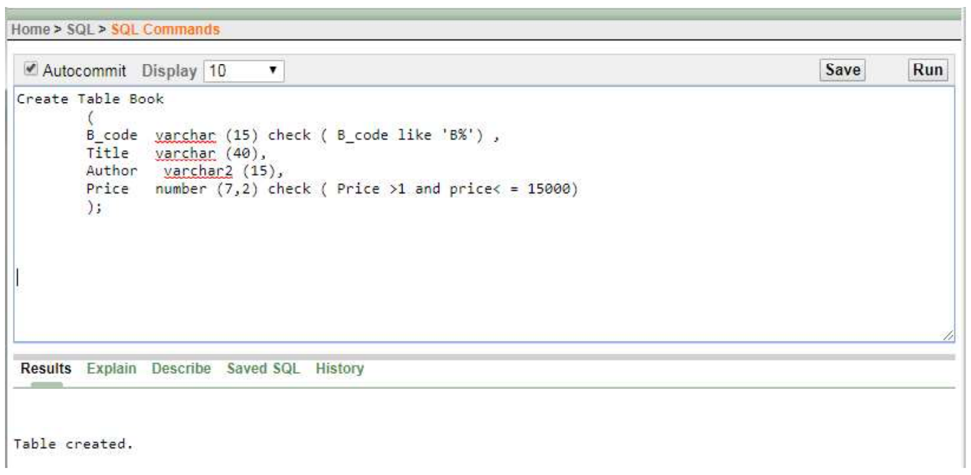
NOTES

The following example creates 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 as follows:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create Table Book
        (
        B_code  varchar (15) check ( B_code like 'B%') ,
        Title   varchar (40),
        Author   varchar2 (15),
        Price   number (7,2) check ( Price >1 and price< = 15000)
        );
```

```
Table created.
```

The above command will create table Book which contains a check constraints with the field Book Code and Price.

DEFAULT Constraint:

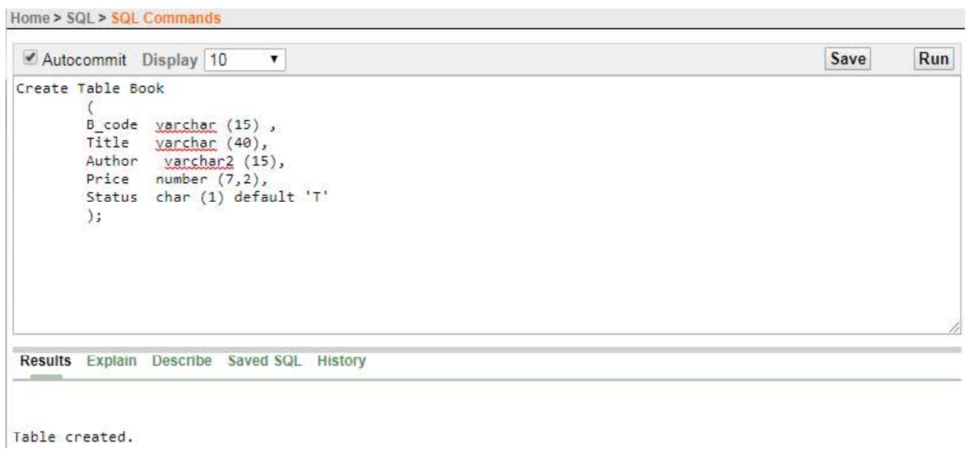
Sometimes, the value of any column for every new record is same. To maintain the status of book in a library is either available to issue or not you must keep the status of book as 'T' (Available) or 'F' (Issued). Every new book purchased for library the status of book is required to be 'T'. Default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

The following example creates 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 as follows:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create Table Book
        (
        B_code  varchar (15) ,
        Title   varchar (40),
        Author   varchar2 (15),
        Price   number (7,2),
        Status  char (1) default 'T'
        );
```

```
Table created.
```

The above command will create table Book which contains a default constraints with the field status.

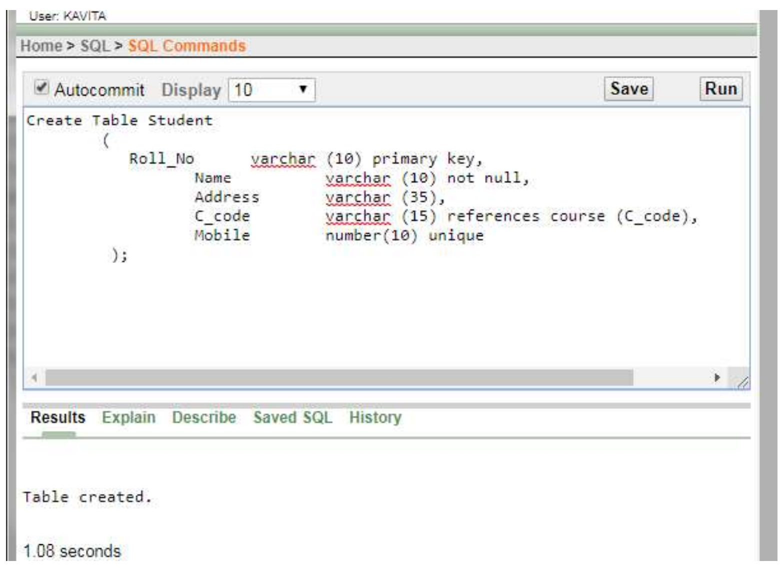
Multiple constraints could be set together in a table as shown below,

The following example creates a table student with the multiple constraints with 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 the above mentioned table as follows:


User: KAVITA
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create Table Student
        (
          Roll_No       varchar (10) primary key,
                Name          varchar (10) not null,
                Address       varchar (35),
                C_code        varchar (15) references course (C_code),
                Mobile        number(10) unique
        );
```

```
Table created.

1.08 seconds
```

DML: EXPERIMENTS USING DML SQL STATEMENTS

NOTES

Data Manipulation Language (DML)

Once a table or other object is created using data definition language, Data Manipulation Language (DML) commands are used to insert, manipulate and access data. DML commands helps in inserting, updating, deleting and searching of data.

The data manipulation language statements are INSERT, DELETE, and UPDATE.

Insert Records in Table

Once structure of a table is created the next action is to insert records on table. Insert is a Data Manipulation Language (DML) command.

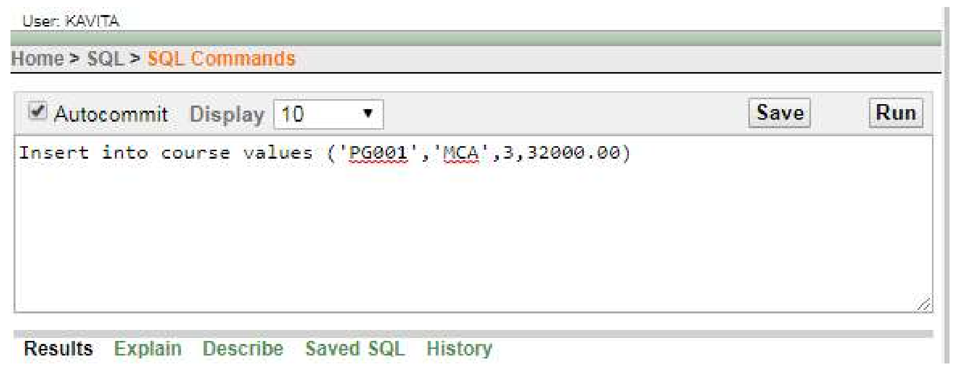
The **Syntax** for insert command,

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

Few **Examples** to insert records in table

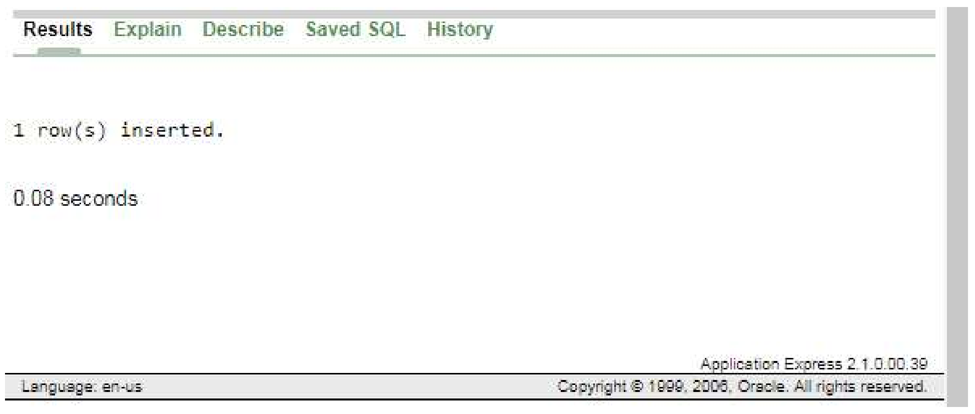
Example-1

To insert records data into course table the command is as follows:



```
Insert into course values ('PG001','MCA',3,32000.00)
```

Output:


Results
Explain
Describe
Saved SQL
History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

```
1 row(s) inserted.

0.08 seconds
```

NOTES

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

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

Try Yourself:

1. Add five records in course table
2. 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

3. View the structure of Book table
4. Add five records in Book table

INSERT FEW MORE RECORDS:

- Insert into course values (‘PG003’, ‘M Sc-IT’, 3, 32000.00)
- Insert into course values (‘PG002’, ‘MBA’, 2, 40000.00)
- Insert into course values (‘UG002’, ‘B SC-IT’, 3, 25000.00)

Insert Data into Specific Fields

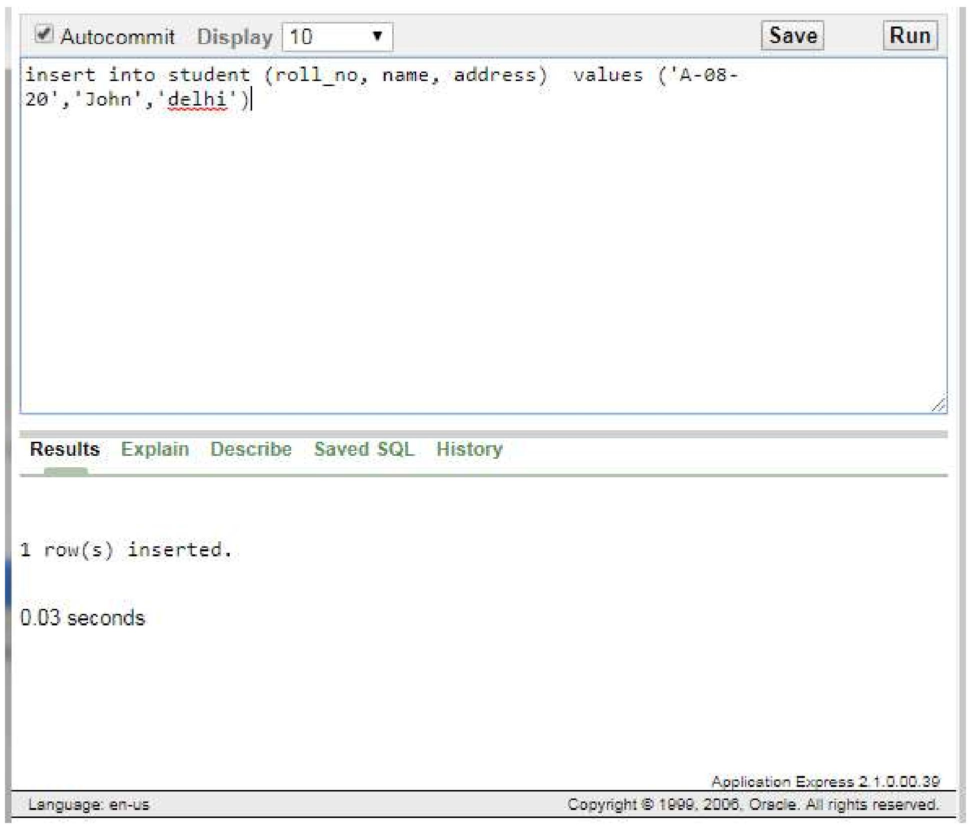
With the above syntax of insert command it is necessary to insert data in all the fields in the same sequence as defined in the table. But sometimes few fields are required to update later on for example student’s subjects marks are inserted in the table and total, percentage or grade is required to calculate later on. To deal with such a situation you could use the following syntax:

The **Syntax** to insert data into selected fields only

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

NOTES

Examples


Autocommit
Display 10
Save
Run
Results
Explain
Describe
Saved SQL
History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

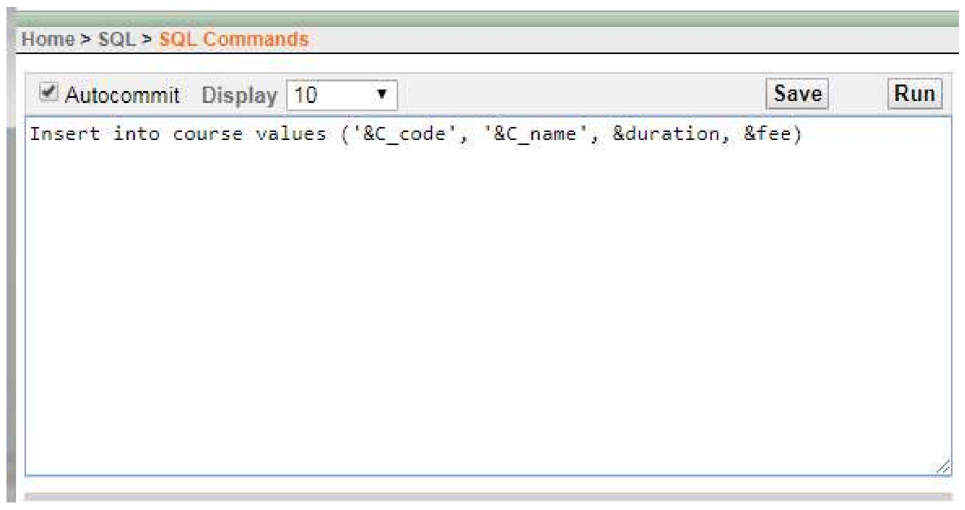
```
insert into student (roll_no, name, address)  values ('A-08-
20','John','delhi')
```

```
1 row(s) inserted.

0.03 seconds
```

Insert Data with User Interaction

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:



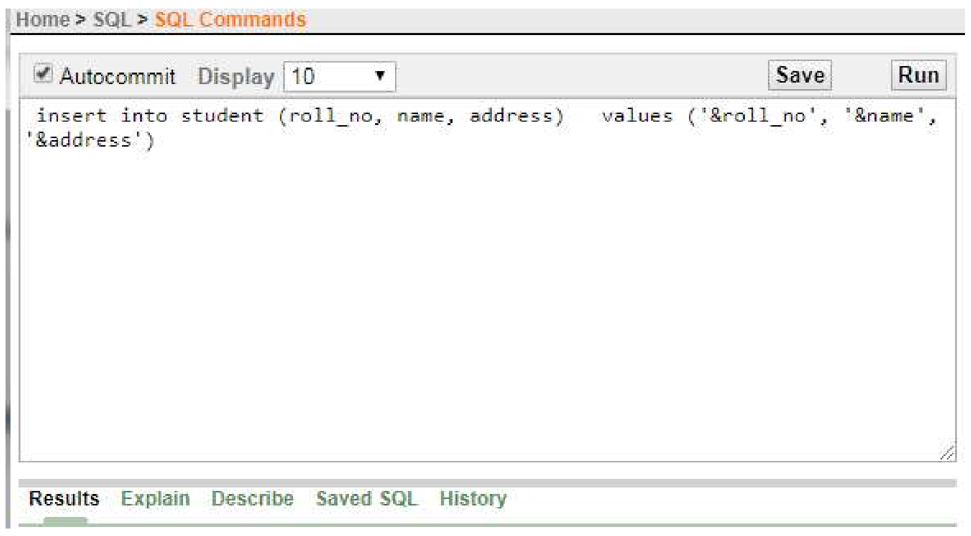
```
Insert into course values ('&C_code', '&C_name', &duration, &fee)
```

To insert more record the same command could be repeated by putting / and pressing enter key at SQL prompt.

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

NOTES

Example:



```
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.

Check Your Progress

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 three 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

After inserting records into table, data could be displayed with the command select. All the fields and records could be displayed or only selective records and fields could be retrieved.

NOTES

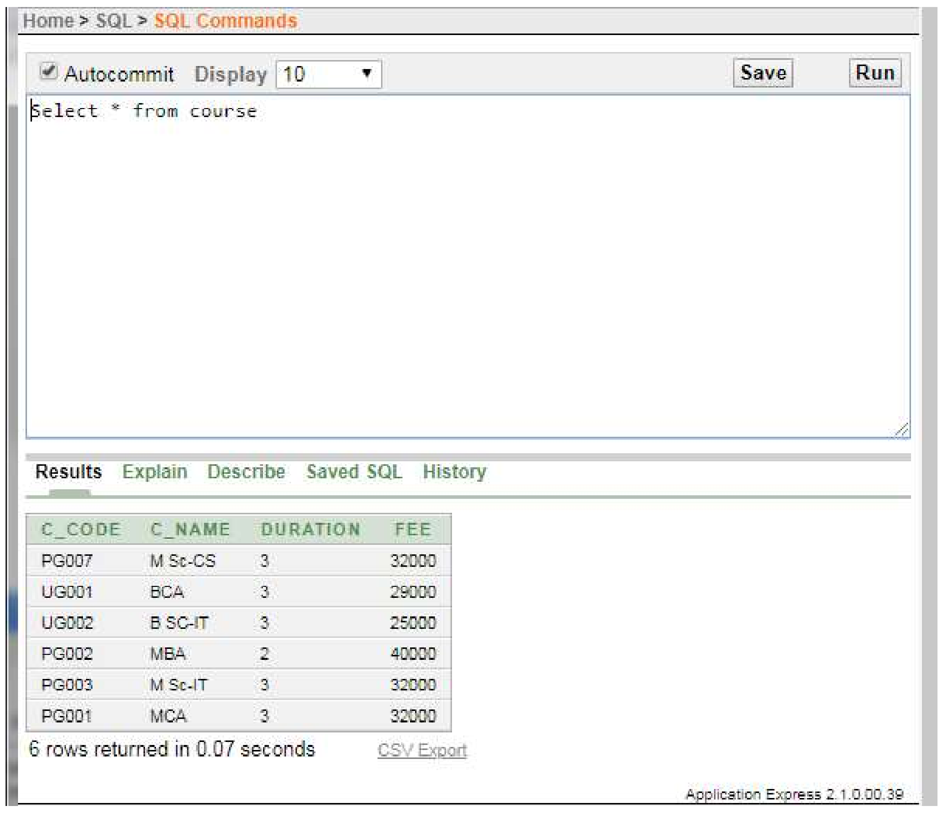
To View All the Columns

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

The **Syntax** for select command:

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

The **Example** for select command:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39

```
Select * from course
```

```
C_CODE  C_NAME   DURATION  FEE
PG007   M Sc-CS  3         32000
UG001   BCA      3         29000
UG002   B SC-IT  3         25000
PG002   MBA      2         40000
PG003   M Sc-IT  3         32000
PG001   MCA      3         32000
6 rows returned in 0.07 seconds
```

To View Selective Columns

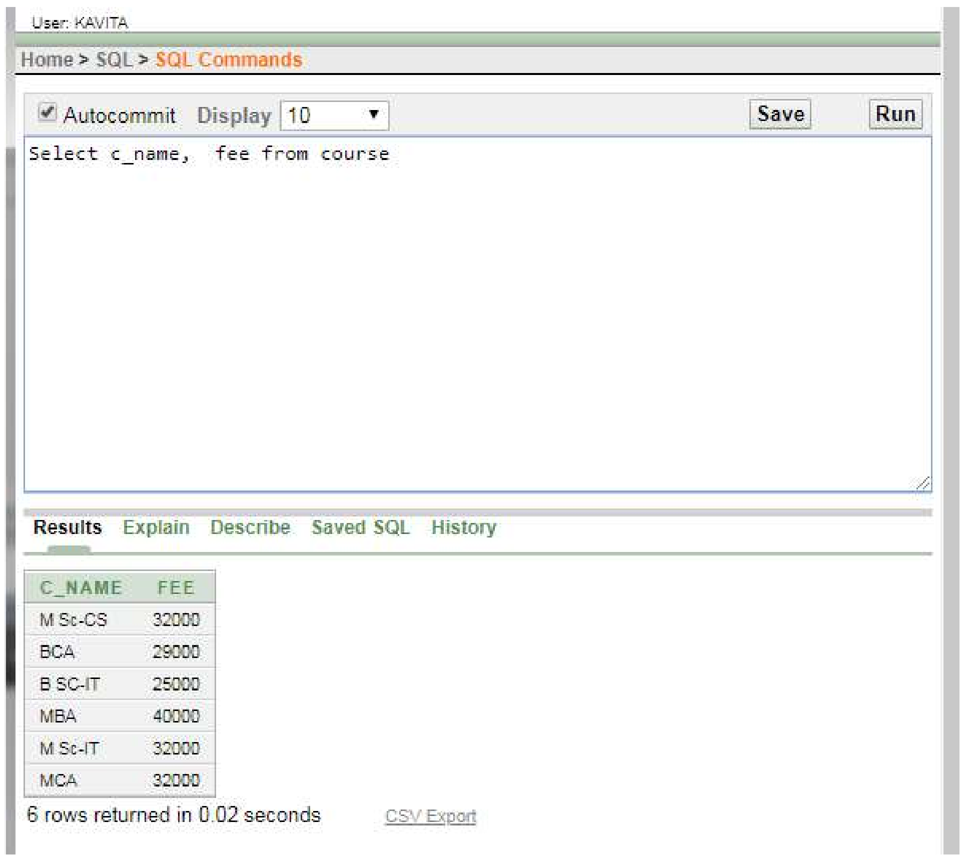
To view only selective fields enter column names separated by comma (,) as shown below:

The **Syntax** to select required fields:

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

NOTES

The **Example** for select:



User: KAVITA

Home > SQL > SQL Commands

☑ Autocommit Display 10 Save Run

```
Select c_name,  fee from course
```

Results Explain Describe Saved SQL History

```
C_NAME    FEE
M Sc-CS   32000
BCA       29000
B SC-IT   25000
MBA       40000
M Sc-IT   32000
MCA       32000
6 rows returned in 0.02 seconds     CSV Export
```

Update Table Records

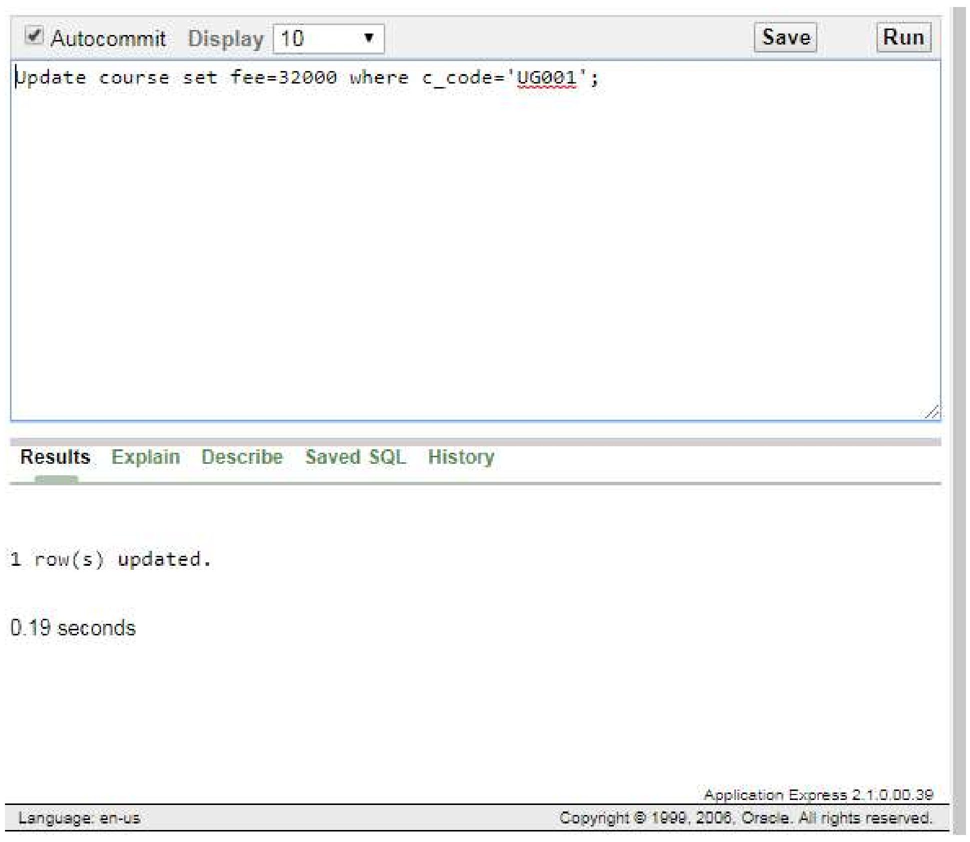
You may sometimes need to update the records that you have in your table. The Contact No. or an Address of any person has been changed or Course Fee is changed by the university. In such cases, the Data manipulation language update command is used.

The **Syntax** to Update:

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

NOTES

The **Example** for Update command:


Autocommit
Display 10
Save
Run
Results
Explain
Describe
Saved SQL
History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

```
Update course set fee=32000 where c_code='UG001';
```

```
1 row(s) updated.

0.19 seconds
```

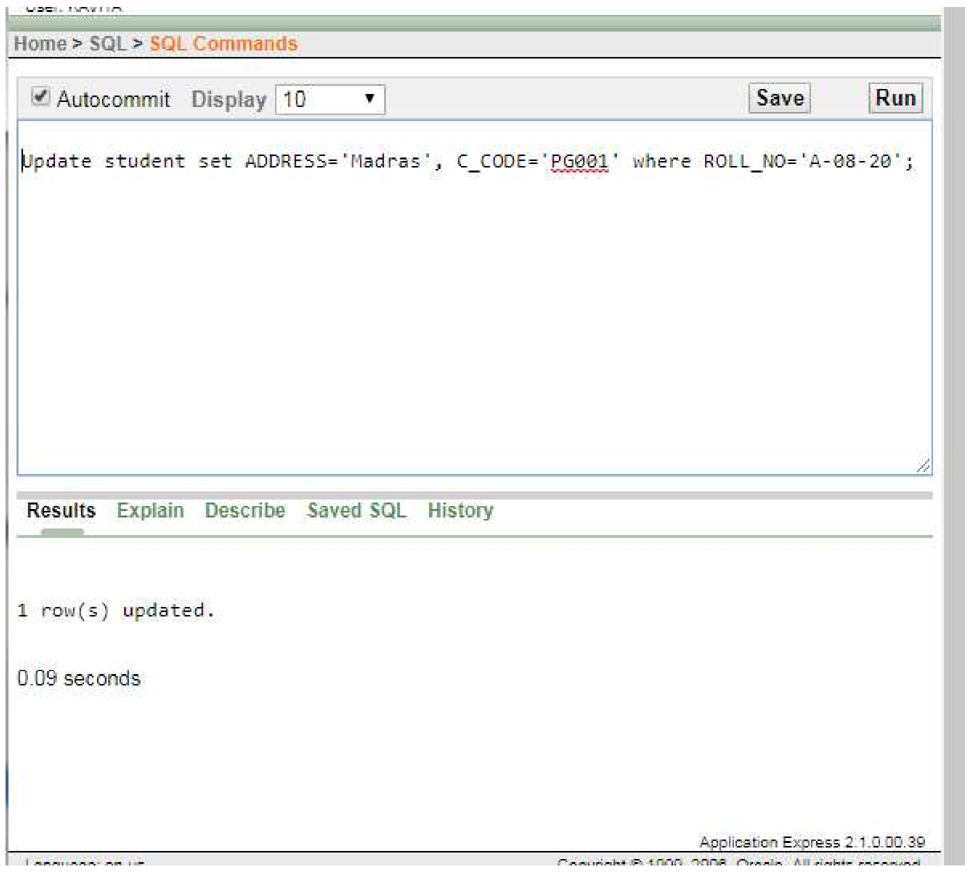
The above command will update the fee of course UG001 from ₹ 29,000 to ₹ 32,000.

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 ₹ 32,000.

More than one columns could also be updated by specifying multiple columns and there new values after set key words.

NOTES

The **Example** to update multiple columns:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39

```
Update student set ADDRESS='Madras', C_CODE='PG001' where ROLL_NO='A-08-20';
```

```
1 row(s) updated.

0.09 seconds
```

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 ₹ 32000 to ₹ 38000 of course where C_code is PG001

Delete Records

If records are no more needed you could delete records form the table. For this purpose you may use Data Manipulation Language (DML) command, i.e., **delete**. One, more than one or all the records could be deleted from the table depending upon the 'Where' condition.

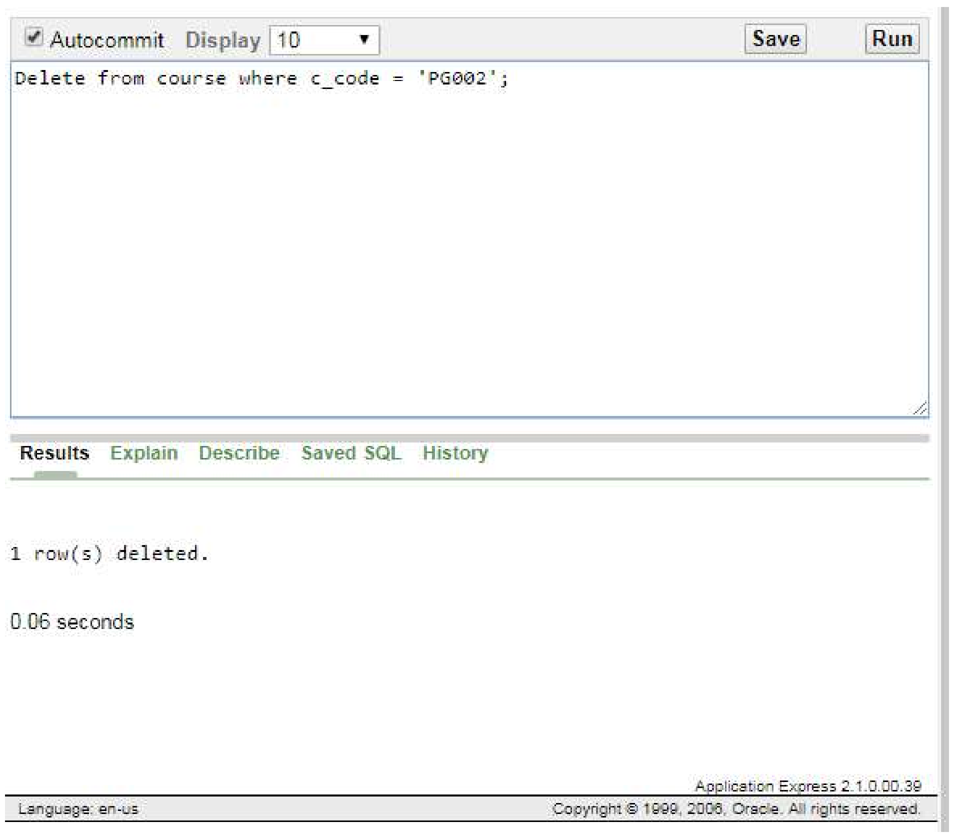
The **Syntax** for delete command:

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

Or

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

The **Example** for delete command:


Autocommit
Display 10
Save
Run
Results
Explain
Describe
Saved SQL
History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

```
Delete from course where c_code = 'PG002';
```

```
1 row(s) deleted.

0.06 seconds
```

The above command will delete one record from Course table where Course Code is PG002.

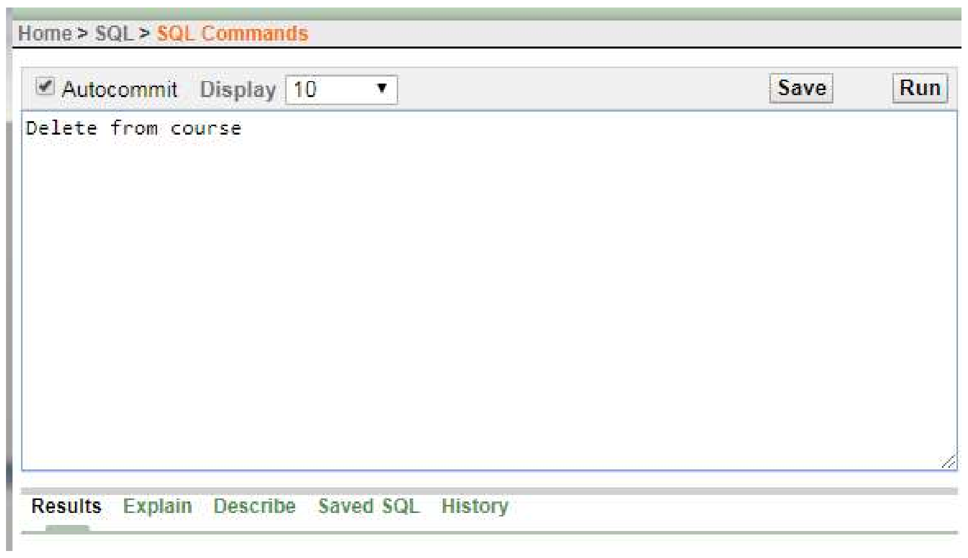
To delete all the records from a table you could write the delete command without 'Where' clause as given below:

```
Delete from course;
```

Or

```
Delete course;
```

NOTES



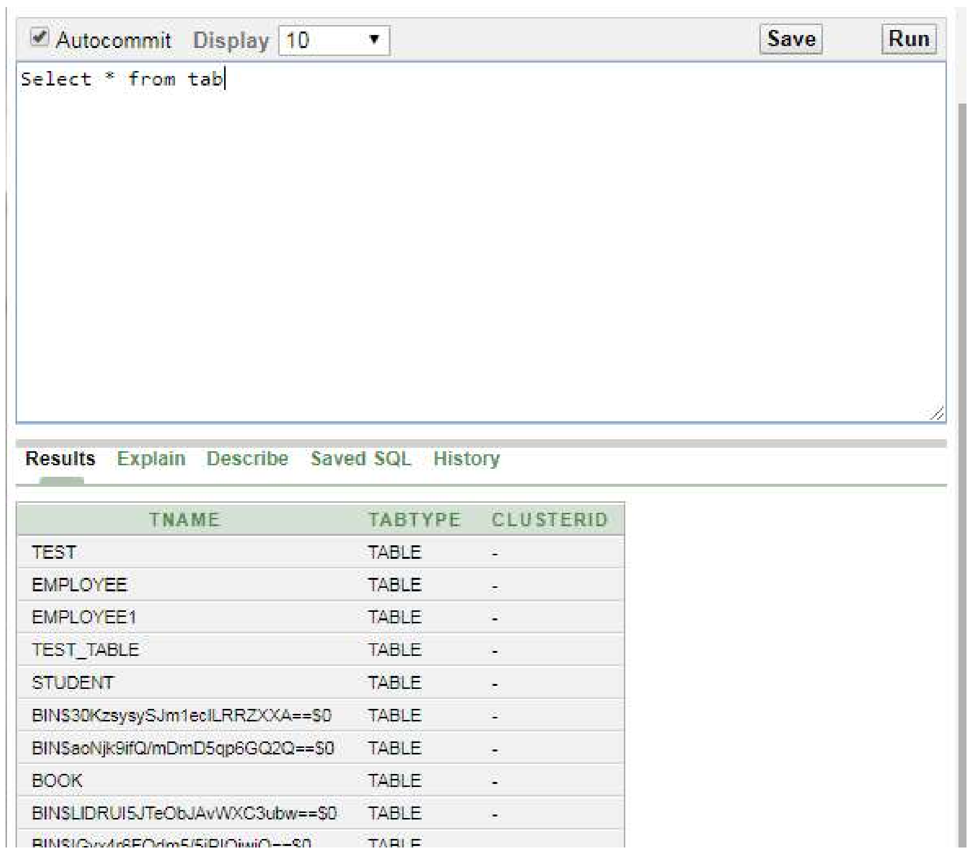
```
Delete from course
```

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

View the Existing Tables

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

The **Example** to view all tables:



```
Select * from tab
```

TNAME	TABTYPE	CLUSTERID
TEST	TABLE	-
EMPLOYEE	TABLE	-
EMPLOYEE1	TABLE	-
TEST_TABLE	TABLE	-
STUDENT	TABLE	-
BIN$30KzsysySJm1ecILRRZXXA==$0	TABLE	-
BIN$aoNjk9ifQ/mDmD5qp6GQ2Q==$0	TABLE	-
BOOK	TABLE	-
BIN$LIDRUI5JTeObJAvWXC3ubw==$0	TABLE	-
BIN$lGvx4r6EOdm5/5iPlOiwiQ==$0	TABLE	

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.

Before discussing about the 'Where' condition in SQL commands, it is important to know about the operators in Oracle.

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

Syntax:

```
Operator operand
```

2. Binary Operators

A binary operator operates 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
- Concatenation Operator

Arithmetic Operators: Arithmetic operators manipulate two operands and produce one result. These operators are Addition, Subtraction, Division, Multiplication and Modulus. These operators work on numeric data type for any calculation, such as addition, subtraction, and division operators also works on date data type.

The examples of arithmetic operators are as given below:

Operator	Description	Example	Result
/	Division	Select 345 / 4 from dual;	86.25
*	Multiplication	Select 345 * 4 from dual;	1380
+	Addition	Select 345 + 4 from dual;	349
-	Subtraction	Select 345 - 4 from dual;	341
Mod	Modulus **(**returns the reminder of *m* divided by *n***)**	Select 345 mod 4 from dual;	1

Dual Table

Dual is a dummy table in Oracle that could be used to perform temporary calculations and to check the result of any Oracle function on data which is not stored in any table. A dual table is consisting of only one row and a column.

Comparison Operators: Comparison operators are used to compare one expression with another. The result of a comparison could be TRUE, FALSE. It is mainly used with WHERE clause of select, update and delete commands.

Operator	Description	Example
=	Equal to	Select roll_no, name from student where c_code = 'PG001';
!= or <>	Not equal to	Select roll_no, name from student where c_code <> 'UG003';
<	Less than	Select c_name, duration from course where fee < 50000
>	Greater than	Select c_name, duration from course where fee >50000;
<=	Less than or equal to	Select c_name, duration from course where fee <=45000;
>=	Greater than or equal to	Select c_name, duration from course where fee >=56000;
In / Not In	Compare if a value lies within a specified list of values	Select * from student where name IN ('Smith', 'John'); Select * from student where name IN NOT ('Smith', 'John');
Between/ Not Between	Compare if a value lies within a specified range of values	Select c_name, duration from course where fee between 45000 and 56000;
Like / Not Like	Pattern Matching	Select * from student where name LIKE 'd%';
Is Null/ Is Not Null	Compare if a value is null	Select * from student where contact_no IS NULL;

Note: In a pattern matching operator LIKE, the underscore character (_) represents any one character and the percent character (%) represents a group of characters.

NOTES

Logical Operators: Logical operators test for the truth of some condition. Logical operators, like comparison operators, return a **Boolean** data type with a value of TRUE, FALSE, or UNKNOWN.

Operator	Description	Example
AND	Returns TRUE if both conditions are TRUE. Returns FALSE if either is FALSE	Select c_name, duration from course where fee >=45000 AND fee<=56000;
OR	Returns TRUE if either condition is TRUE. Returns FALSE if both are FALSE.	Select roll_no, name from student where c_code = ‘PG001’ OR c_code = ‘PG002’
NOT	Returns TRUE if the condition is FALSE. Returns FALSE if it is TRUE.	Select * from course where fee not >78000;

Set Operators: Set operators combine the results of two queries into a single result.

Operator	Description	Example
UNION	Returns all distinct rows selected by either query.	Select roll_no,b_code from issue UNION Select roll_no,b_code from return;
INTERSECT	Returns all distinct rows selected by both queries.	Select roll_no,b_code from issue INTERSECT Select roll_no,b_code from return;
MINUS	Returns all distinct rows selected by the first query but not the second.	Select roll_no,b_code from issue MINUS Select roll_no,b_code from return;

Concatenation Operator: Concatenation operator is used to concatenate two strings.

Operator	Description	Example
||	Concatenates character strings	Select ‘Student Name' || name from student;

Filtering Records Using Where Conditions

A university could have thousand of records but all those 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 could be used with conditional, logical and other operators could be used. Following are the examples of various operators in ‘Where’ clause of select query:

The **Syntax** for select command with ‘Where’ clause:

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

NOTES

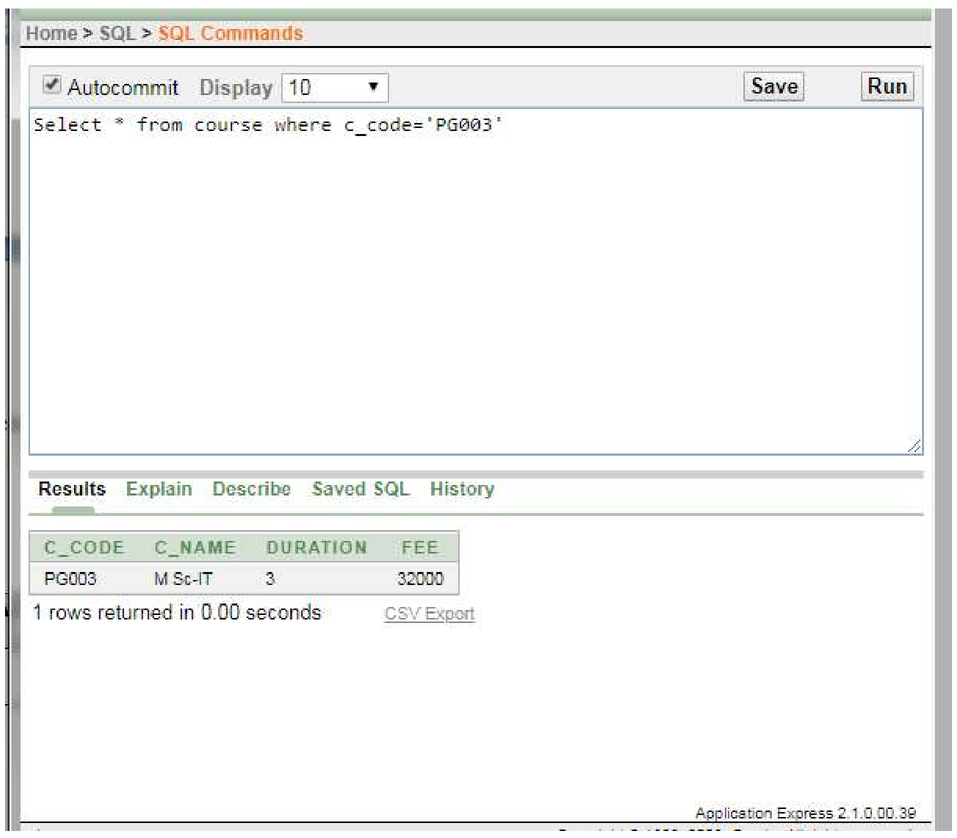
The following is the Course table contains 8 records. Let us filter records from this table with different conditions.

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:

a) **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:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39

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

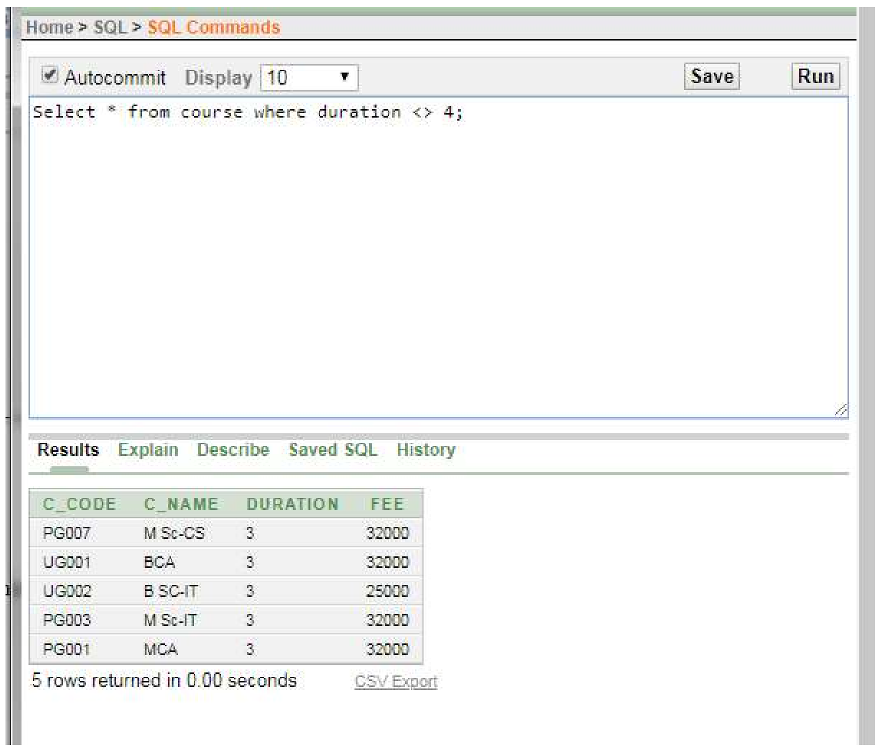
```
C_CODE  C_NAME   DURATION  FEE
PG003   M Sc-IT  3         32000
1 rows returned in 0.00 seconds
```

b) **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;
```

NOTES

Output of the above query is shown below:



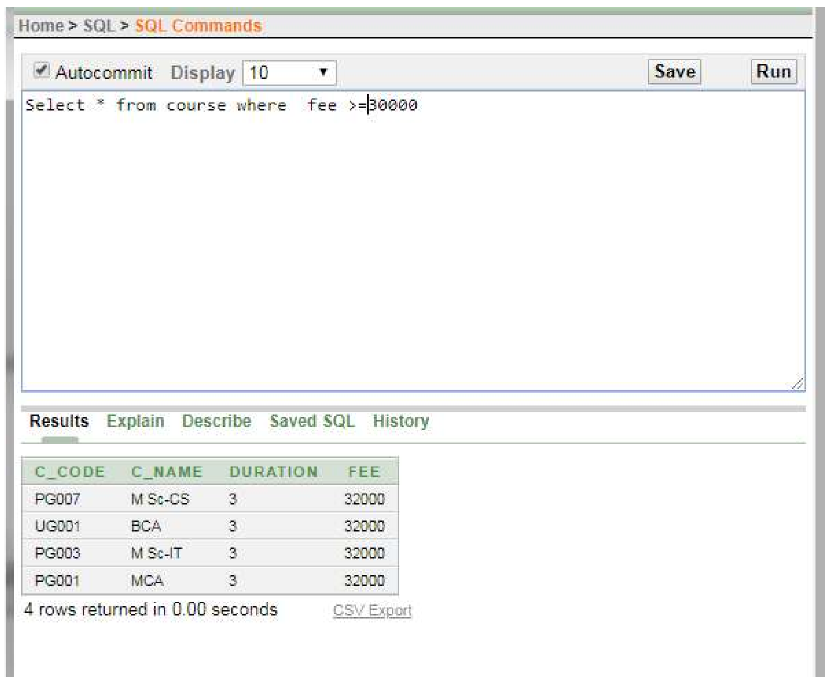
```
Select * from course where duration <> 4;
```

```
C_CODE  C_NAME   DURATION  FEE
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
5 rows returned in 0.00 seconds
```

c) Greater Than (>): To see the detail of course where course fee is Greater than ₹ 50000 then the query will be

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

Output of the above query is shown below:



```
Select * from course where  fee >=30000
```

```
C_CODE  C_NAME   DURATION  FEE
PG007   M Sc-CS  3         32000
UG001   BCA      3         32000
PG003   M Sc-IT  3         32000
PG001   MCA      3         32000
4 rows returned in 0.00 seconds
```

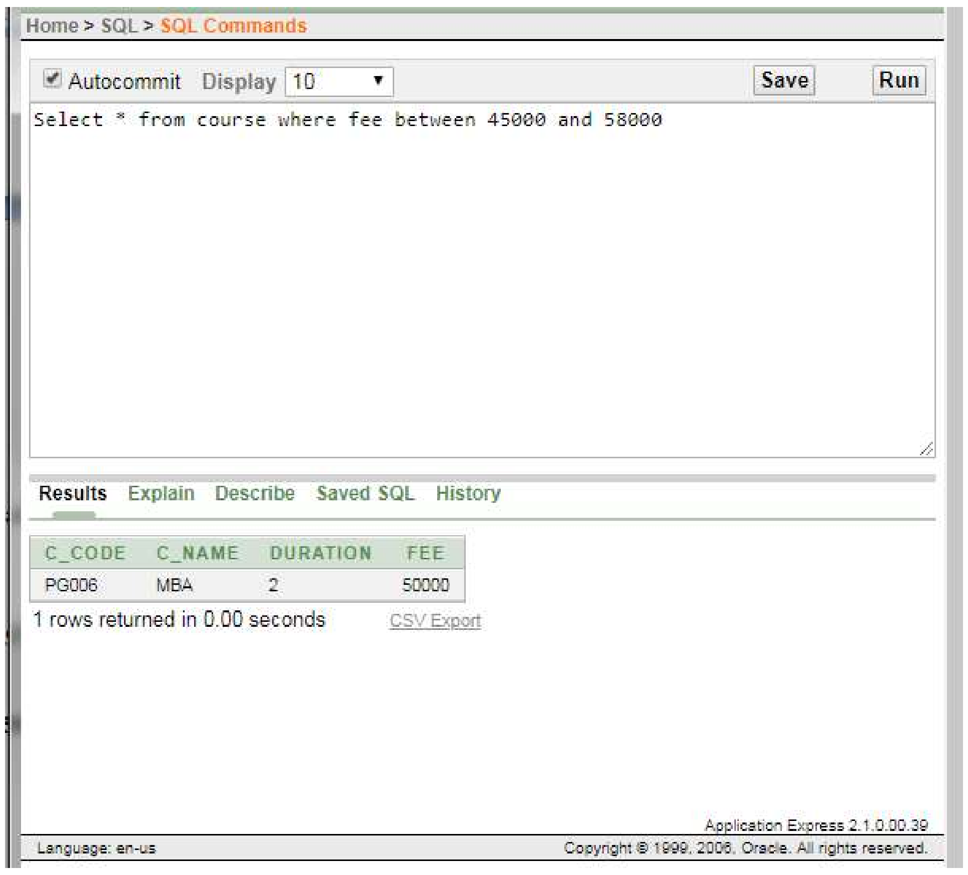
As equal to, not equal to and greater than operators are used to filer records other operators as less than, less than equal to, greater than equal to could be used.

Other Operators in SQL:

(a) BETWEEN: The BETWEEN operator filters the records between a given range. Suppose you want to filter the courses where fee in between ₹ 45000 to ₹ 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:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

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

```
C_CODE  C_NAME  DURATION  FEE
PG006   MBA     2         50000
1 rows returned in 0.00 seconds
```

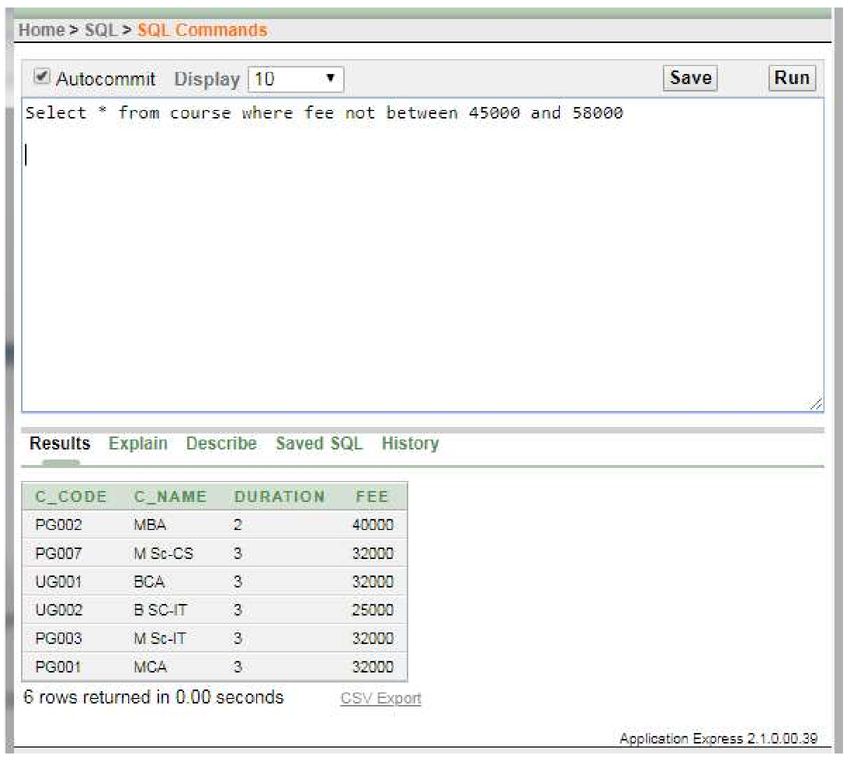
The between operators can filter the numbers, text, or date values.

(b) NOT BETWEEN: The NOT BETWEEN operator filters the records where the data in not between a given range.

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

NOTES

Output of the above query is shown below:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39

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

```
C_CODE   C_NAME    DURATION   FEE
PG002    MBA       2          40000
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
6 rows returned in 0.00 seconds
```

Oracle Functions

Oracle provided various built-in functions for different purposes, 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.

NOTES

Few scalar functions are listed below:

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

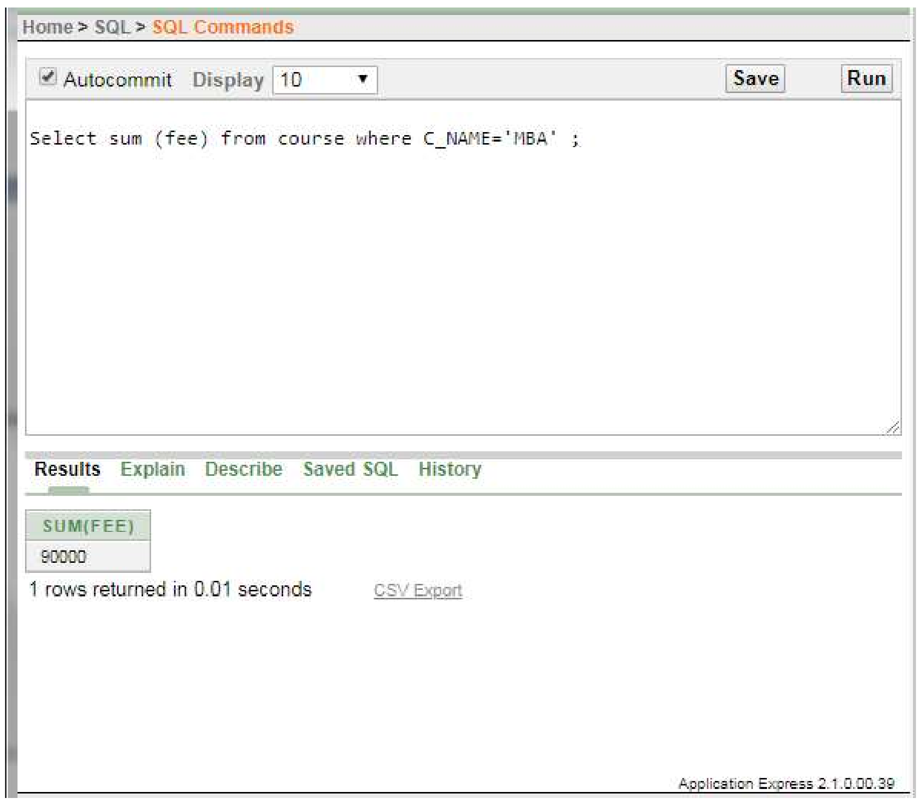
Let suppose we have a table course with the following records:-

```
Select * from course;
```

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

Table - Book

(i) `Sum ()`: To see the sum of price of the item_ID I003 SQL query is as follows:

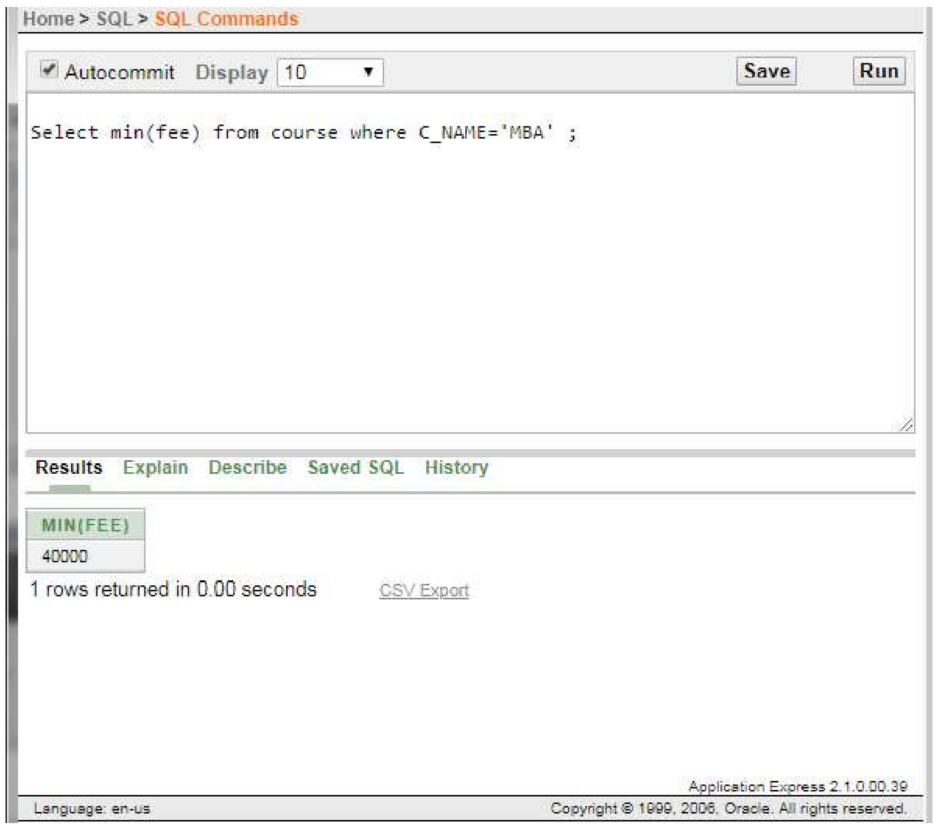

Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39

```
Select sum (fee) from course where C_NAME='MBA' ;
```

```
SUM(FEE)
90000
1 rows returned in 0.01 seconds
```

NOTES

(ii) `Min ()`: To see the order detail where item price is minimum SQL query is as follows:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

```
Select min(fee) from course where C_NAME='MBA' ;
```

```
MIN(FEE)
40000
1 rows returned in 0.00 seconds
```

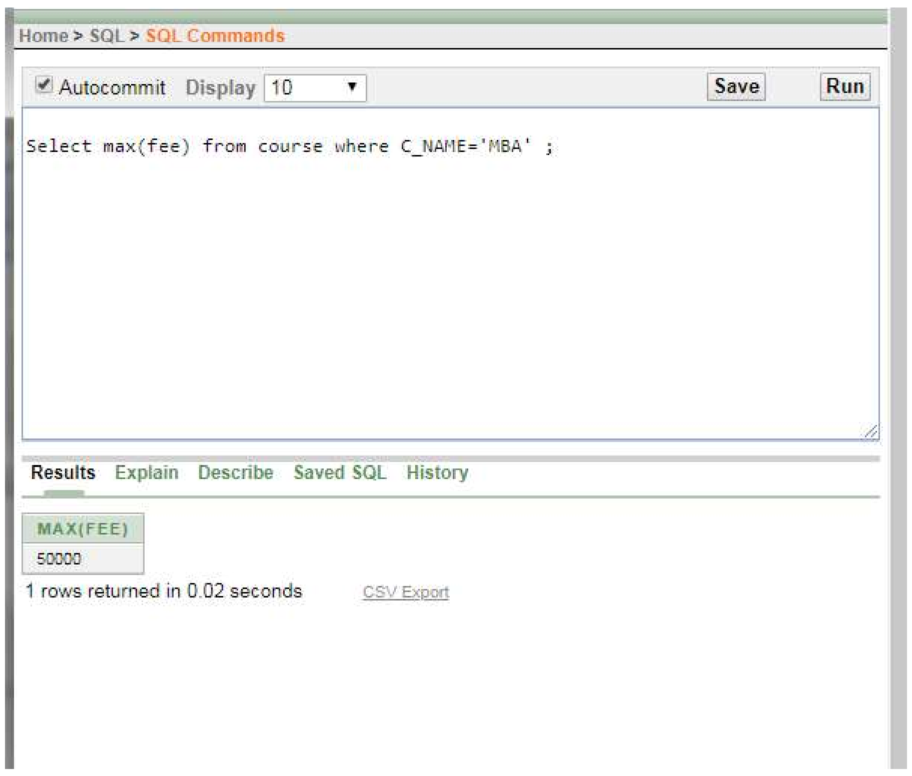
(iii) `Max ()`: To see the order detail where item price is maximum SQL query is as follows:

```
Select max (fee) from course where C_NAME= 'MBA';
```

(iv) `Count ()`: To see the number of orders for item_ID I001 SQL query is as follows:

```
Select count (item_id) from order where item_ID = 'I001';
```

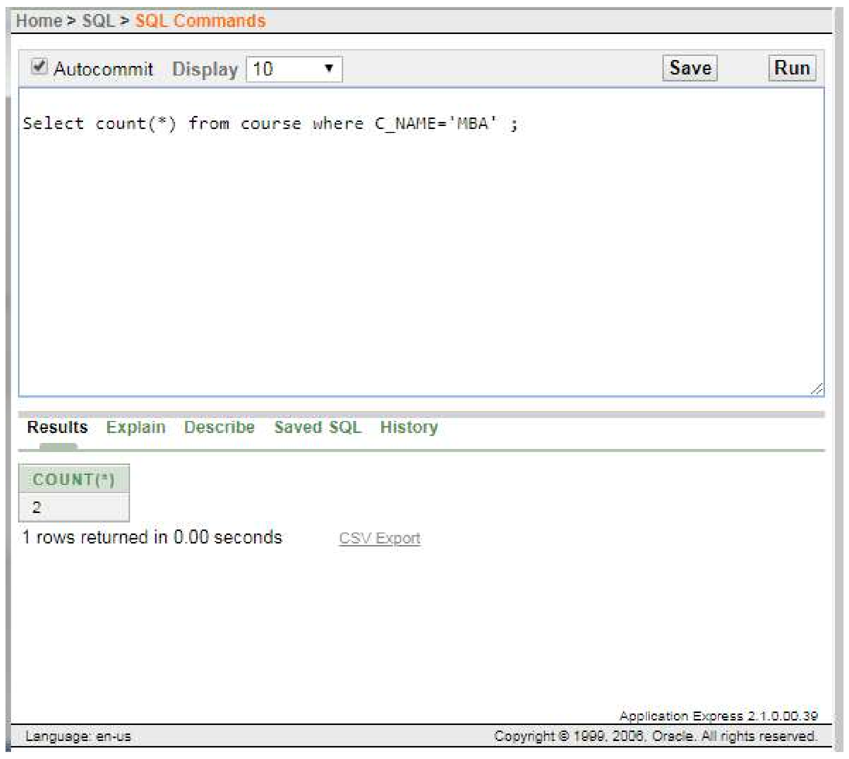
NOTES


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export

```
Select max(fee) from course where C_NAME='MBA' ;
```

```
MAX(FEE)
50000
1 rows returned in 0.02 seconds
```

(v) **`Count (*)`:** To see the number of records in a table SQL query is as follows:

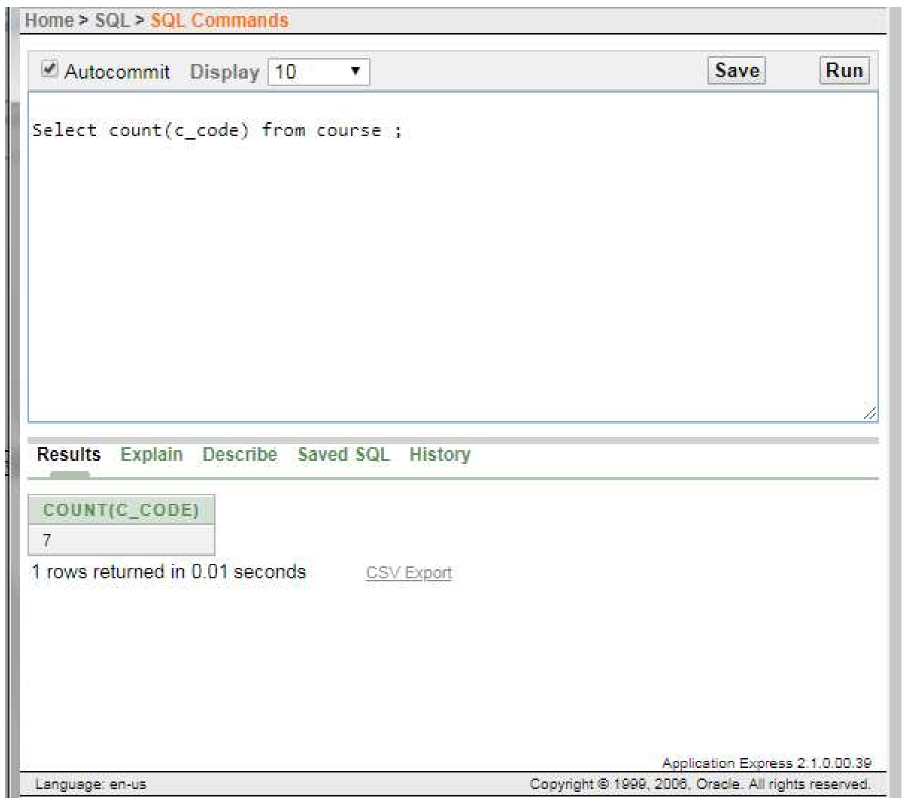

Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

```
Select count(*) from course where C_NAME='MBA' ;
```

```
COUNT(*)
2
1 rows returned in 0.00 seconds
```

NOTES

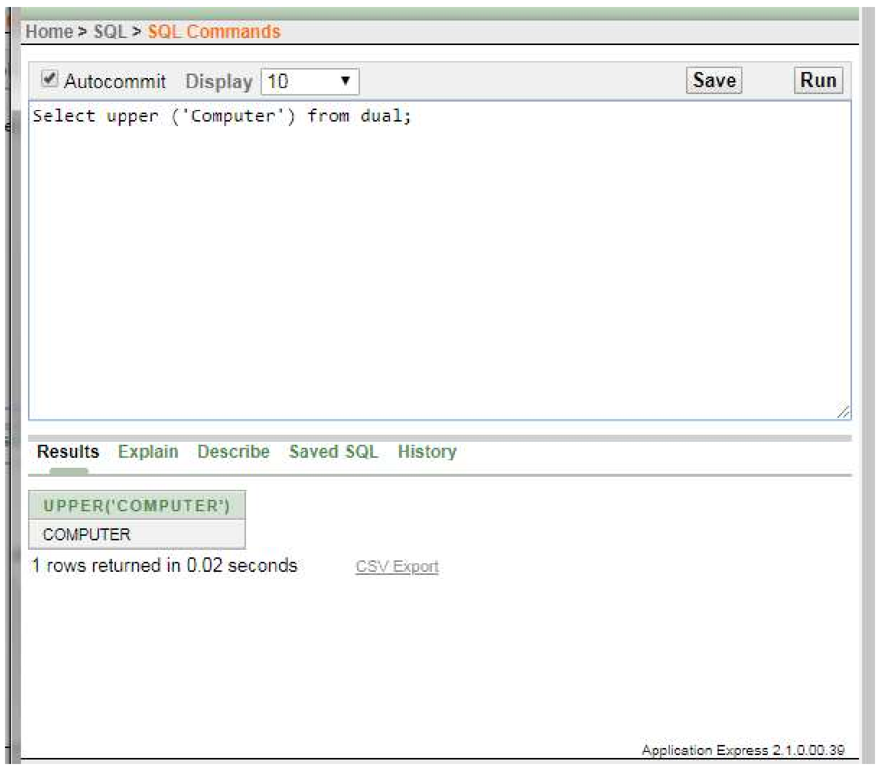
```
Select count (c_code) from course;
```


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

```
Select count(c_code) from course ;
```

```
COUNT(C_CODE)
7
1 rows returned in 0.01 seconds
```

(vi) `Upper ()`: To converts the text to upper case SQL query is as follows:

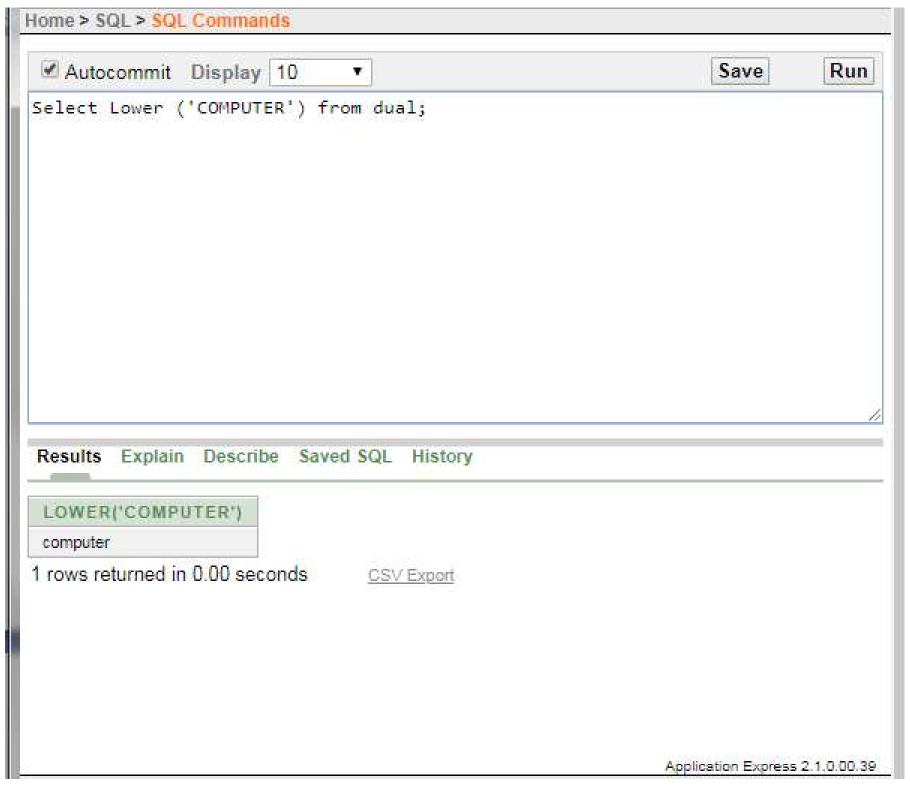

Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39

```
Select upper ('Computer') from dual;
```

```
UPPER('COMPUTER')
COMPUTER
1 rows returned in 0.02 seconds
```

NOTES

(vii) `Lower ()`: To converts the text to upper case SQL query is as follows:


Home > SQL > SQL Commands

Autocommit Display 10 Save Run

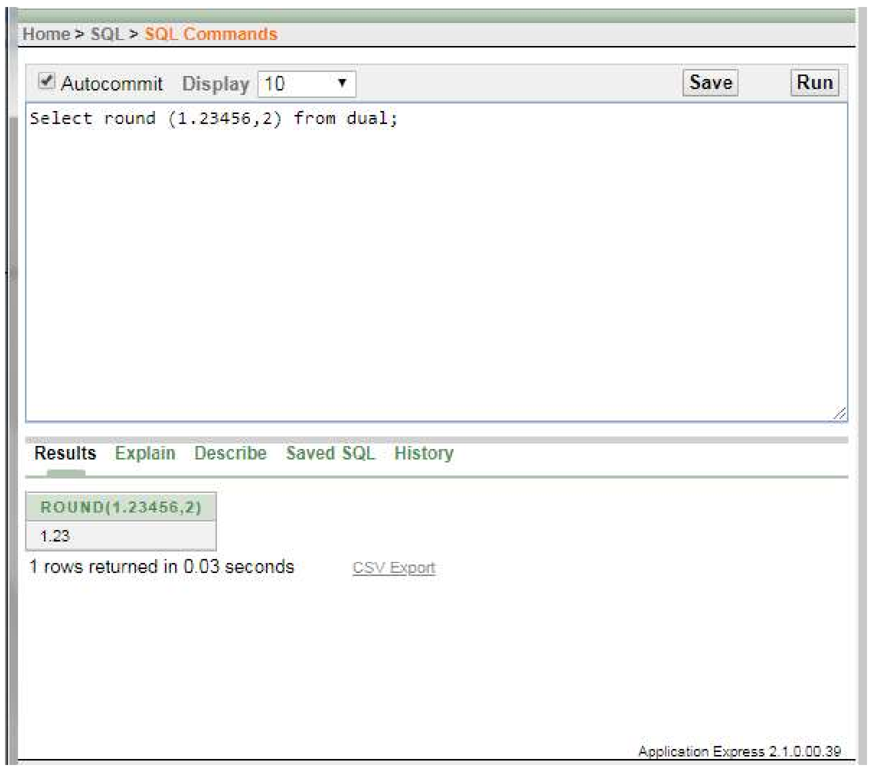
```
Select Lower ('COMPUTER') from dual;
```

Results Explain Describe Saved SQL History

```
LOWER('COMPUTER')
computer
1 rows returned in 0.00 seconds     CSV Export
```

Application Express 2.1.0.00.39

(viii) `Round (n)`: To round of any number SQL query is as follows:


Home > SQL > SQL Commands

Autocommit Display 10 Save Run

```
Select round (1.23456,2) from dual;
```

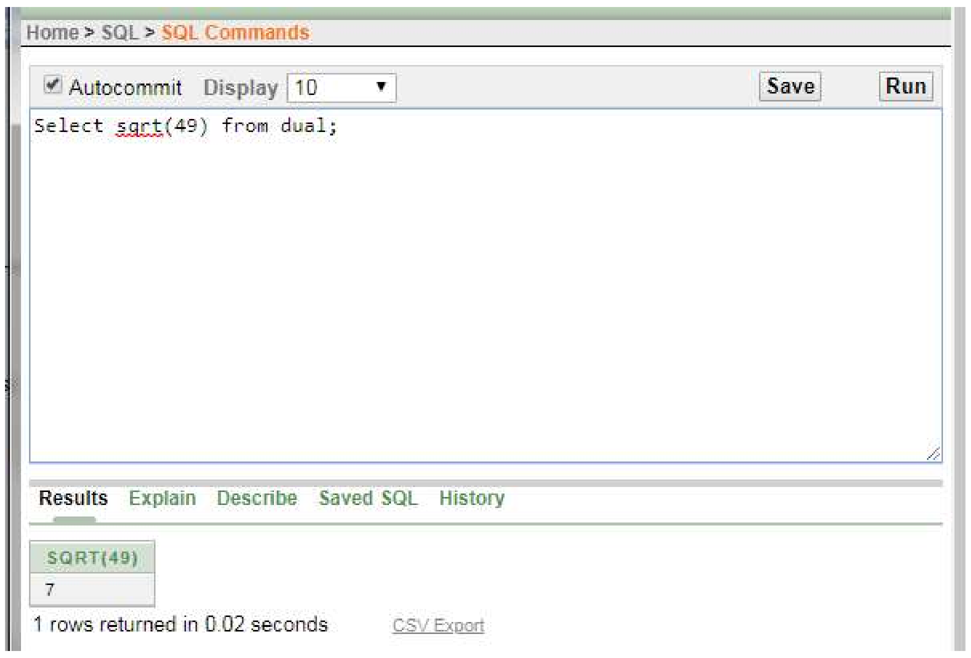
Results Explain Describe Saved SQL History

```
ROUND(1.23456,2)
1.23
1 rows returned in 0.03 seconds     CSV Export
```

Application Express 2.1.0.00.39

NOTES

(ix) `Sqrt (n)`: It calculates square root value of number SQL query is as follows:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export

```
Select sqrt(49) from dual;
```

```
SQRT(49)
7
1 rows returned in 0.02 seconds
```

Join Commands

To collect or see data from two or more tables Oracle provides Join command. The Join command in SQL helps in fetching rows and columns from multiple tables and also we can apply some condition while fetching records from multiple tables.

Oracle vs. ANSI Syntax

There are two syntaxes for join commands Oracle and ANSI. To join table in Oracle syntax where clause is used. Where in ANSI format there is a separate join clause which makes the query more clear and easy to read and understand.

There are mainly four types of joins that you need to understand. They are:

- CROSS JOINS
- INNER JOIN
- FULL JOIN
- LEFT JOIN
- RIGHT JOIN

(a) Cross Joins (Cartesian Product)

A cross join returns every row from the first table matched to every row in the second. This will always return the Cartesian product of the two table's rows i.e., the number of rows in the first table times the number in the second. Student and project both store three rows. So cross joining those returns 3 * 3 = 9 rows.

To cross join tables using Oracle syntax, simply list the tables in the 'from' clause:

```
select *
from student, project;
```

Using ANSI style, type cross join between the tables you want to combine:

```
select *
from student
cross join project;
```

(b) INNER JOINS (Join)

An inner join is used to compares values in one or more columns from each. It only returns rows which match the join conditions in both tables. The simplest join checks if the values in a column from one table equal the values in a column from the other. This join is also known as **Equi join**.

For example:

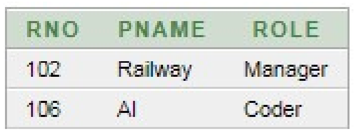
Create following tables Student, Marks and Project:

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

Table: Student

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

Table: Marks



RNO	PNAME	ROLE
102	Railway	Manager
106	AI	Coder

Table: Project

Oracle Format

```
select student.rno, name, sub1, sub2, sub3, total
from student, marks
where student.rno=marks.rno;
```

ANSI Format

```
select student.rno, name, sub1, sub2, sub3, total
from student
inner join marks
on student.rno= marks.rno;
```

(c) Outer Join:

An outer join returns all the rows from one table along with the matching rows from the other. Rows without a matching entry in the outer table return null for the outer table's columns.

An outer join can either be left or right. The direction defines which side of the join the database preserves the rows for.

There are three types of outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

(i) Left Outer Join: The LEFT JOIN or the LEFT OUTER JOIN returns all the records from the left table and also those records which satisfy a condition from the right table. Also, for the records having no matching values in the right table, the output or the result-set will contain the NULL values.

Oracle Format

```
select student.rno, name, sub1, sub2, sub3, total
from student,marks
where student.rno=marks.rno(+);
```

ANSI Format

```
select student.rno,name,sub1,sub2,sub3,total from student
left outer join marks on student.rno=marks.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
102	AMAN	-	-	-	-

(ii) Right Outer Join: The RIGHT JOIN or the RIGHT OUTER JOIN returns all the records from the right table and also those records which satisfy a condition from the left table. Also, for the records having no matching values in the left table, the output or the result-set will contain the NULL values.

Oracle Format

```
select student.rno, project.rno, name, pname
from student , project
where student.rno(+)=project.rno;
```

ANSI Format

```
select student.rno, project.rno, name, pname from student
right outer join project on student.rno=project.rno;
```

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

(iii) Full Outer Join: Sometimes you may want to join two tables to find the matching rows. But also include any unmatched rows from both tables i.e., a "double outer" join. This is known as a full (outer) join.

```
select student.rno, project.rno, name, pname
from student full outer join project on
student.rno=project.rno;
```

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

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 could be accessed or manipulated by other users as per the permission granted to those users.

The data manipulation language statements are GRANT and REVOKE

- **GRANT**- Gives user's access privileges to database.
- **REVOKE**- Withdraw user's access privileges given by using the GRANT command.

TCL: EXPERIMENTS USING TCL SQL STATEMENTS

Transaction Control Language (TCL)

Various transactions are being done by different users these transactions then could be saved permanently or could be can called by the user. TCL commands manage changes made by DML statements. The transaction control statements are COMMIT, SAVEPOINT and ROLLBACK.

Examples of TCL Commands:

- **COMMIT**– Commits a Transaction.
- **ROLLBACK**– Rollbacks a transaction in case of any error occurs.
- **SAVEPOINT**– Sets a save-point within a transaction.
- **SET TRANSACTION**– Specify characteristics for the transaction.

Oracle Transactions:

All the changes that you make through DML command are known as transaction. A transaction is a logical group of work. Transactions that you do on a database temporarily stores data on the client machine that is either it could be made permanent or could be cancelled by the user.

Oracle provides few commands to control the transactions as given below:

- Commit
- Save-Point
- Rollback

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

```
SQL > Commit to work;
```

The keyword work is optional which is used to increased readability only, you could also write

```
SQL > Commit;
```

Rollback: The rollback command is used to terminate the current transaction all the change 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.

Example:

```
SQL > Rollback work;
```

The keyword work is optional which is used to increased readability only, you could also write

```
SQL > rollback;
```

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

```
Rollback complete.
```

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

BLOCK 3: INDEX AND VIEW

This block will discuss about the following topics:

- *Index : Experiment Using Database Index Creation, Renaming an Index, Copying Another Index, Dropping an Index*
- *Views: Create Views, Partition and Locks*

Index and View:

(a) 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.

(i) Creating an Index

The syntax for creating an index in Oracle/PL/SQL is:

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

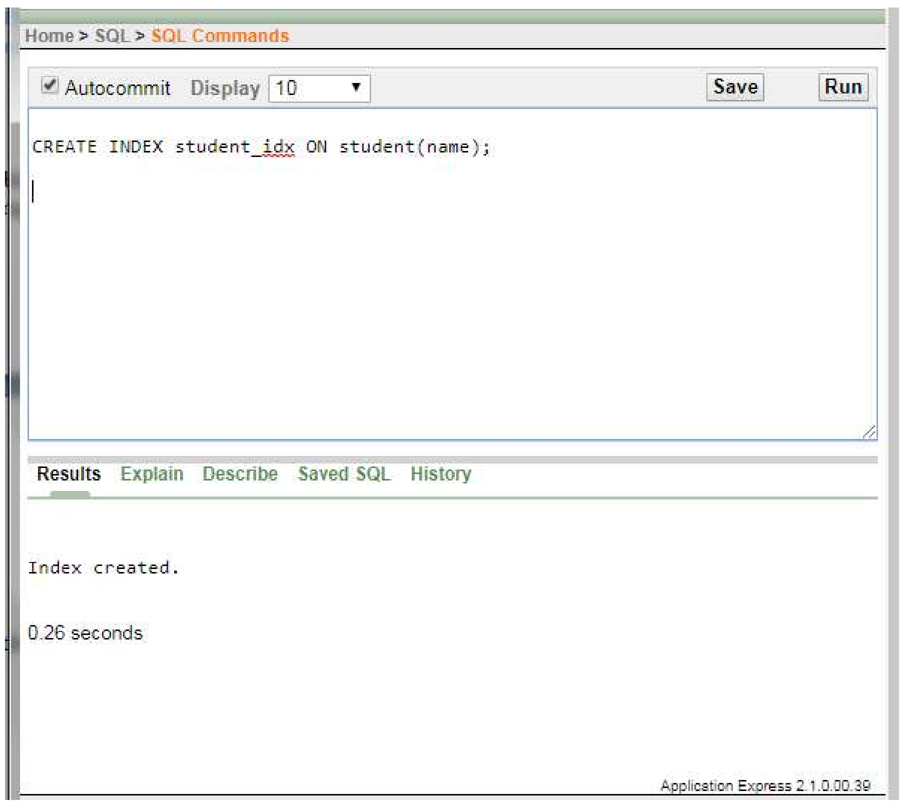
In above syntax, **UNIQUE** refers that the combination of values in the indexed columns must be unique, **index_name** is the name of index, **table_name** is the name of table on which we are creating the index, **column1, column2, ... column_*n*** refers to the columns to use in the index and **COMPUTE 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: Let us look at an example of how to create an index in Oracle/PL/SQL. For example:

```
CREATE INDEX employee_idx
ON employee (name);
```

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

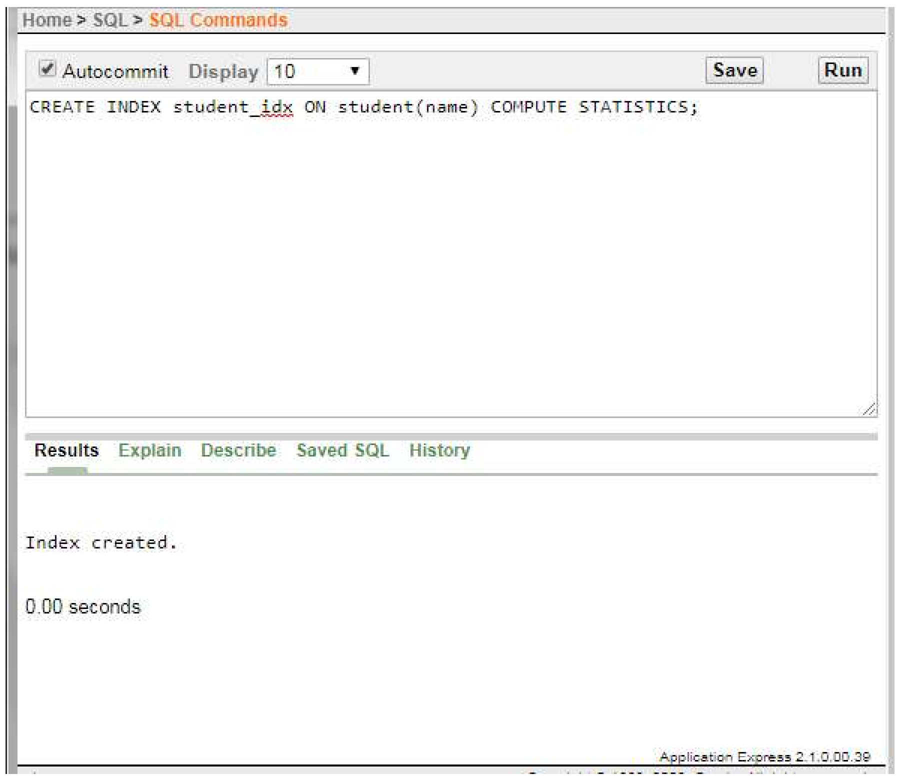
NOTES


Home > SQL > SQL Commands
Autocommit Display 10 Save Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39

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

```
Index created.

0.26 seconds
```

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


Home > SQL > SQL Commands
Autocommit Display 10 Save Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39

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

```
Index created.

0.00 seconds
```

NOTES

(ii) Rename an Index

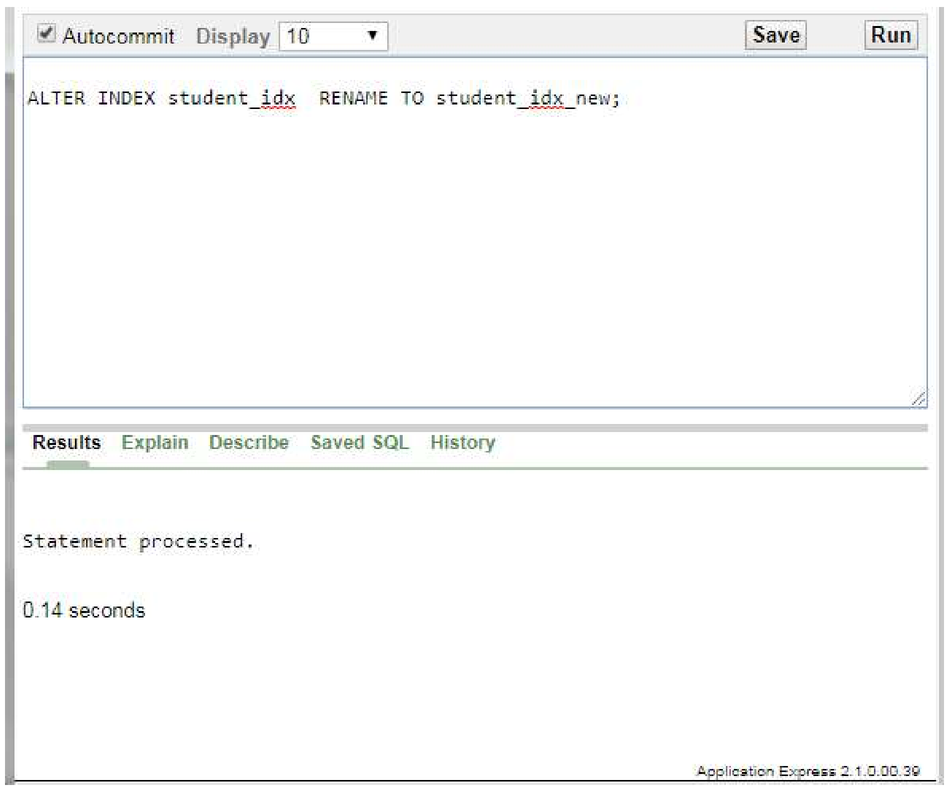
The syntax for renaming an index in Oracle/PL/SQL is:

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

Here, **`index_name`** is the name of the index that you wish to rename and **`new_index_name`** is the new name to assign to the index.

Example:

Let us look at an example of how to rename an index in Oracle/PL/SQL. For example:



```
ALTER INDEX student_idx  RENAME TO student_idx_new;
```

```
Statement processed.

0.14 seconds
```

(iii) Drop an Index

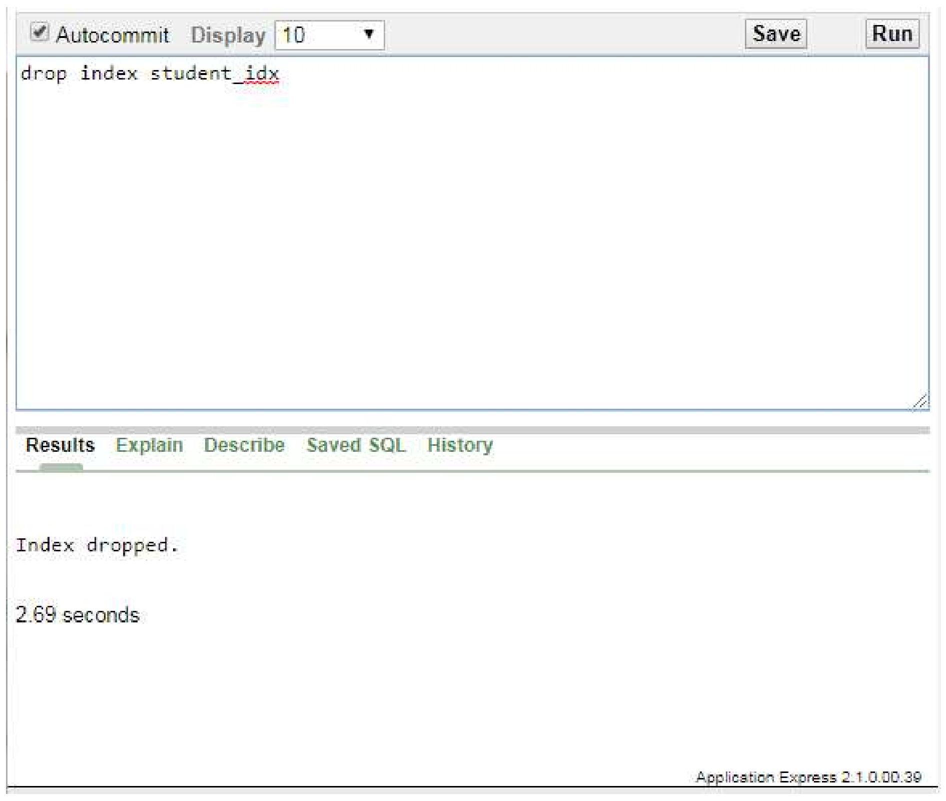
The syntax for dropping an index in Oracle/PL/SQL is:

```
DROP INDEX index_name;
```

Where **`index_name`** is the name of the index to drop.

Example:

Let us look at an example of how to drop an index in Oracle/PL/SQL. For example:



```
drop index student_idx
```

```
Index dropped.

2.69 seconds
```

NOTES

View

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 does not store in the database. View is a query stored as an object. In SQL, a view is a virtual table based on the result-set of an SQL statement. A “View Table” or just “View” makes the Select a long-lived virtual table: no data storage in its own right, just window on data it selects from.

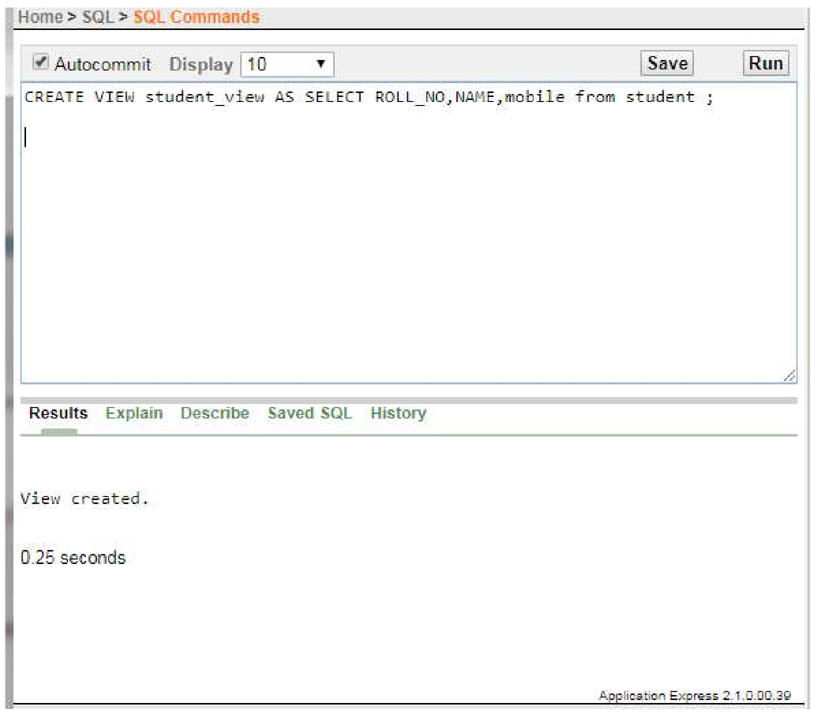
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Syntax:

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

NOTES

```
CREATE VIEW student_view AS SELECT ROLL_NO, NAME, mobile
from student;
```

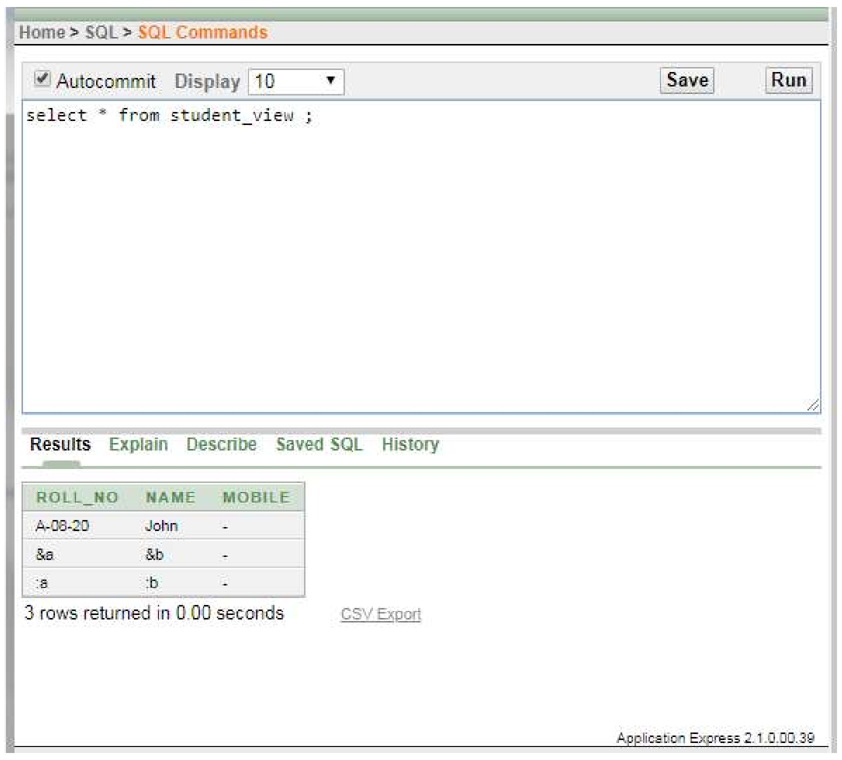

Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39

```
CREATE VIEW student_view AS SELECT ROLL_NO,NAME,mobile from student ;
```

```
View created.

0.25 seconds
```

Display Records from View:

```
select * from student_view;
```


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export
Application Express 2.1.0.00.39

```
select * from student_view ;
```

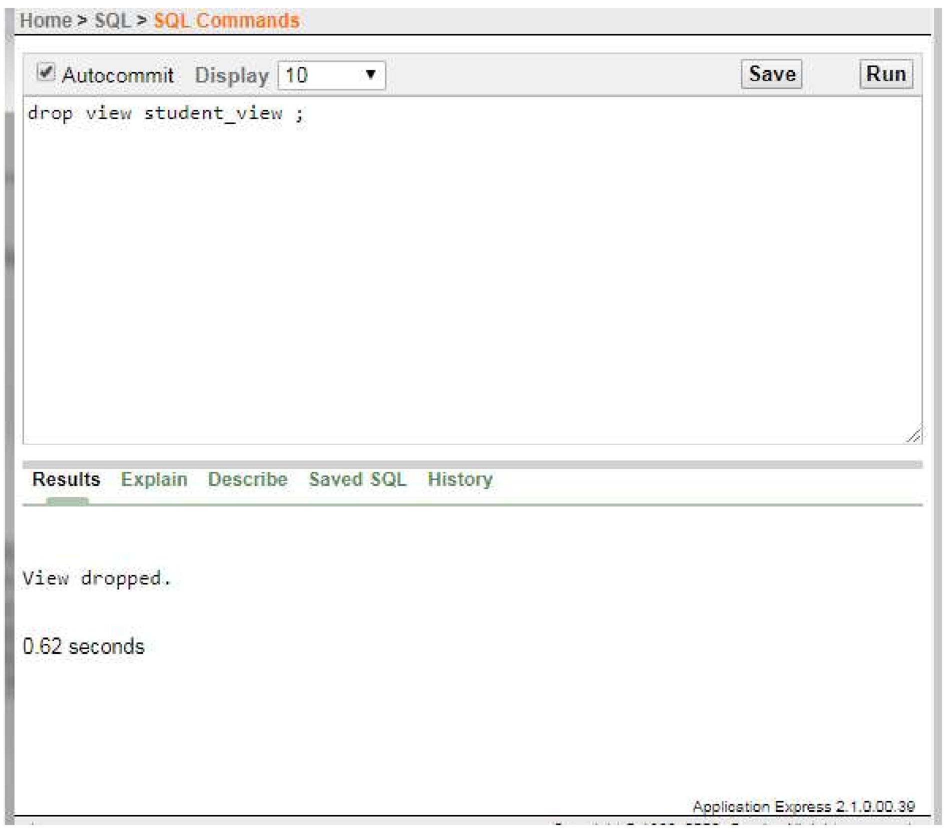
ROLL_NO	NAME	MOBILE
A-08-20	John	-
&a	&b	-
:a	:b	-

3 rows returned in 0.00 seconds

DROP VIEW: This query is used to delete a view, which has been already created.

Syntax: `DROP VIEW View_name;`

```
Drop view student_view;
```


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39

```
drop view student_view ;
```

```
View dropped.

0.62 seconds
```

NOTES

A view can a simple view or a complex view.

Differences between Simple VIEW and Complex VIEW:

Simple VIEW	Complex VIEW
It contains only one table.	It contains one or more number of tables
It does not contain aggregate function like Sum (), Max () or any other aggregate function.	It can contain aggregate function.
Insert and update commands can be performed through a Simple view.	Insert and update commands cannot be performed through a Simple view.
INSERT, DELETE and UPDATE are directly possible on simple view.	We cannot apply INSERT, DELETE and UPDATE on complex view directly.
It does not include NOT NULL columns from base table.	NOT NULL columns that are not selected by simple view can be included in complex view.

NOTES

Complex View

```
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

```
Select * from marks1
```

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

Example-1

```
Create view students_marks1 as
Select student1.rno, name, sub1, sub2, sub3, total from
student1, marks1
Where student1.rno=marks1.rno;
```

Display Records from View:

```
Select * from students_marks1;
```

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

Example-2

Create view students_marks2 as

```
Select student1.rno, name, sub1, sub2, sub3, total from
student1 left outer join marks1 on
student1.rno=marks1.rno;
```

Display Records from View:

```
Select * from students_marks2;
```

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	-	-	-	-

Example-3

Create view students_view as

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

Display Records from View:

```
Select * from students_view;
ORDER BY Clause (Sorting Records)
```

The ORDER BY clause is used to arrange data in ascending or descending order. One or more fields can be used to arrange records.

Example-4

```
Select Rno, name
From student
ORDER BY name;
```

The above command will arrange name in ascending order. Asc and Desc can be used along with order by clause to arrange records n ascending order, which is default or Desc to arrange records in descending order.

```
Select Rno, name, Fee
From student
ORDER BY name Asc, fee Desc
```

BLOCK 4: EXCEPTION HANDLING AND PL/SQL

This block will cover the following topics:

- Exception Handling: PL/SQL Procedure for Application Using Exception Handling
- Cursor: PL/SQL Procedure for Application Using Cursors
- Trigger: PL/SQL Procedure for Application Using Triggers
- Package: PL/SQL Procedure for Application Using Package
- Reports: DBMS Programs to Prepare Report Using Functions

PL/ SQL

Introduction:

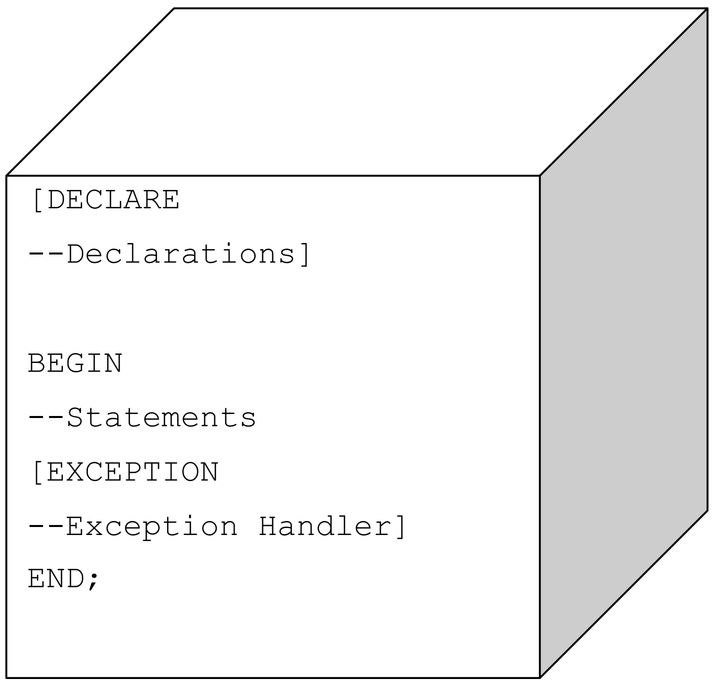
PL/ SQL is also known as an embedded SQL and is a superset of SQL. PL/ SQL is an acronym of Procedural Language/Structure Query Language. It supports procedural features and SQL commands.

NOTES

Structure of PL/ SQL Program:

PL/ SQL program block is divided in three sections, such as

1. Declaration Section
2. Execution Section
3. Exception Handling Section



```
[DECLARE
--Declarations]

BEGIN
--Statements
[EXCEPTION
--Exception Handler]
END;
```

Description of the Blocks

(a) 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 key word **DECLARE.**

(b) 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 key word **BEGIN** and ends with the key word **END.**

(c) The Exception Handling Section: During program execution many abnormal situations may occur. To handle those 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.

The PL/ SQL **Syntax** is as follows:

```
DECLARE
Declaration_Statements
…
BEGIN
```

NOTES

```
Executable_Statements
…
EXCEPTION
Exception_Handling_Statements
…
END ;
..
```

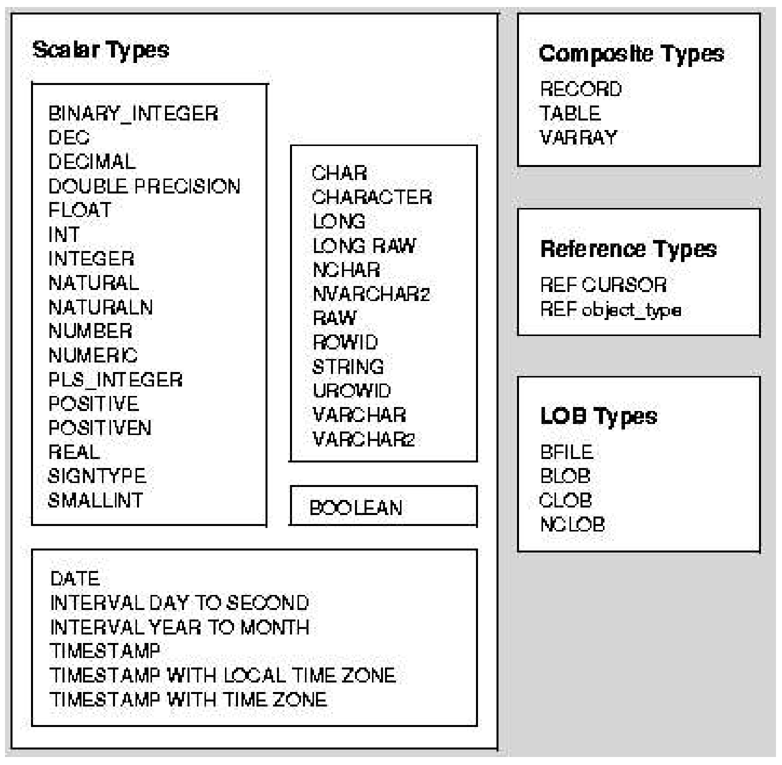
The PL/ SQL Engine:

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 could be stored. PL/ SQL provides a various data types which are system defined and also gives the flexibility to the programmer to create their own data types which fit to the business needs. Classification of data types:

- Scalar Data Types
- Composite Data Types



NOTES

Comments in PL/ SQL:

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

Types of Comments:

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
   Declare
   …
```

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:

Example:

```
   Name char (30);
Salary Number (8, 2);
   Date_of_join Date;
```

Constants can be of any data type:

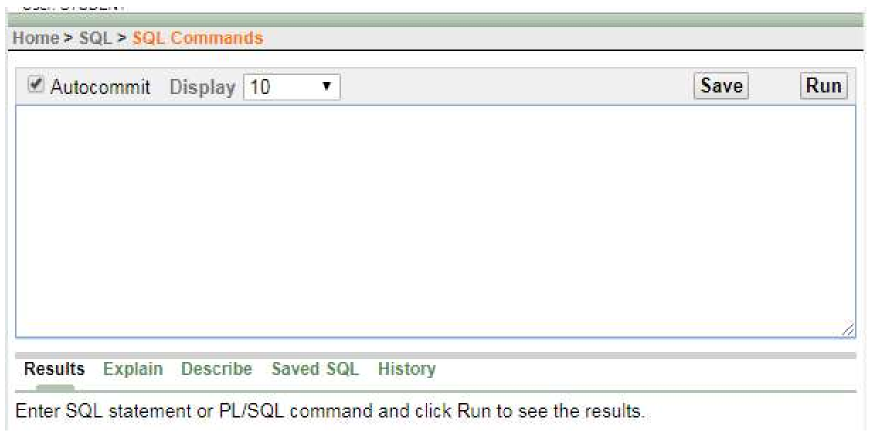
Example:

```
Pi constant number (3, 2) : = 3.5;
Status Booleans: = TRUE;
```

Pi and Status are assigned valued during declaration makes them constant.

Example of PL/ SQL Program:

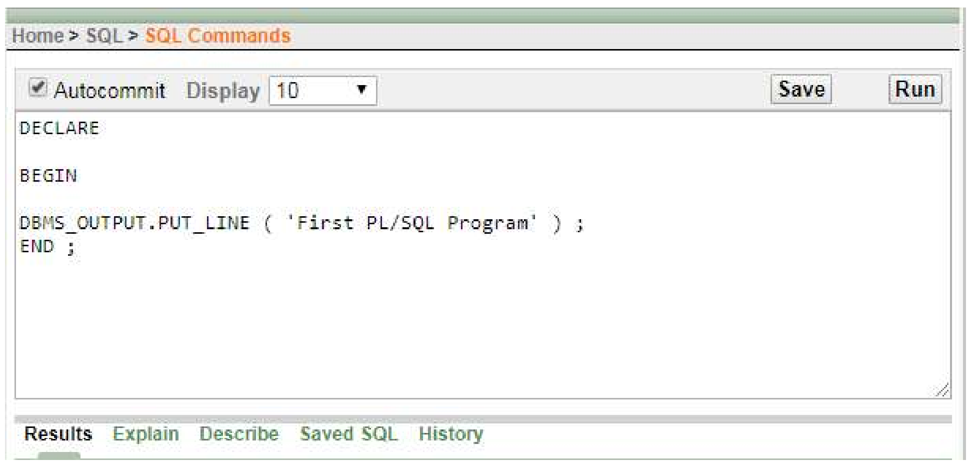
Step-1: Write PL/SQL Program in SQL Commands as shown below:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

Enter SQL statement or PL/SQL command and click Run to see the results.

Example: Write a PL/SQL program to display 'First PL/SQL Program'.

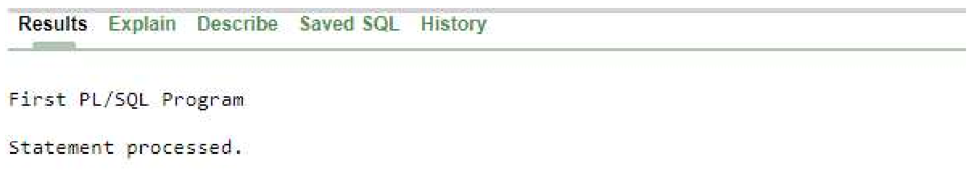
```
DECLARE
 BEGIN
 DBMS_OUTPUT.PUT_LINE ('First PL/SQL Program');
END;
```



```
DECLARE

BEGIN

DBMS_OUTPUT.PUT_LINE ( 'First PL/SQL Program' ) ;
END ;
```

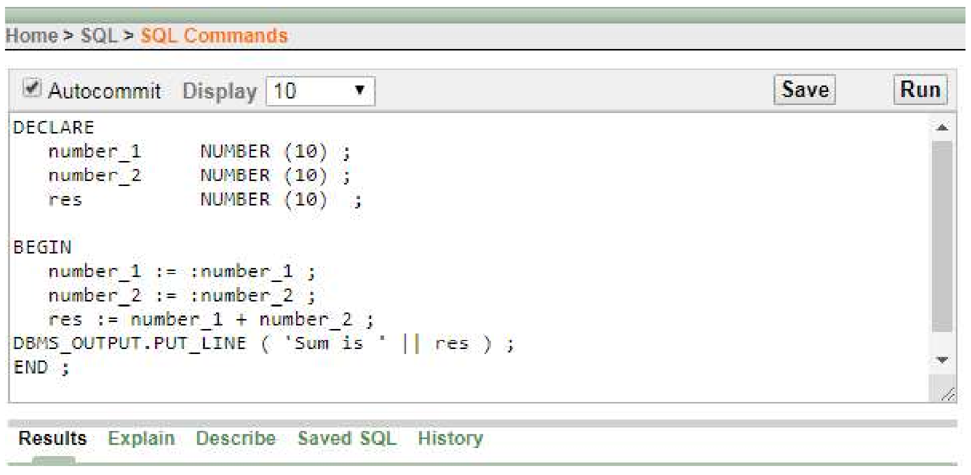
Click on Run button to run program.

Output:



```
First PL/SQL Program

Statement processed.
```

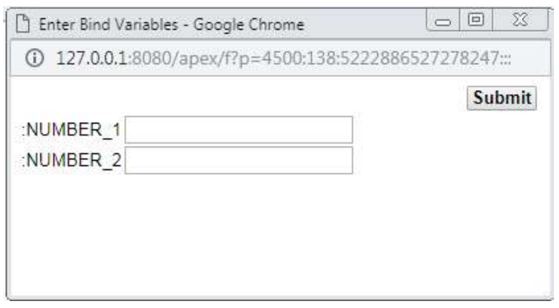
Example: Write a PL/SQL program to display sum of two numbers given at run time.



```
DECLARE
   number_1     NUMBER (10) ;
   number_2     NUMBER (10) ;
   res          NUMBER (10)  ;

BEGIN
   number_1 := :number_1 ;
   number_2 := :number_2 ;
   res := number_1 + number_2 ;
DBMS_OUTPUT.PUT_LINE ( 'Sum is ' || res ) ;
END ;
```

NOTES

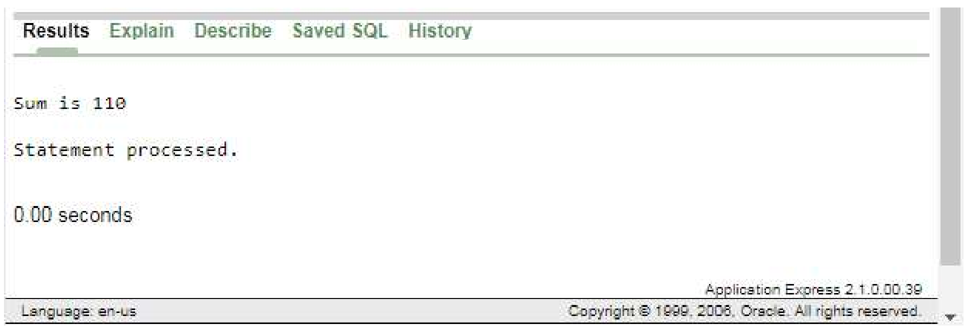
After running this program it will show input screen as shown below:


Enter Bind Variables - Google Chrome
127.0.0.1:8080/apex/f?p=4500:138:5222886527278247:::
Submit

```
:NUMBER_1
:NUMBER_2
```

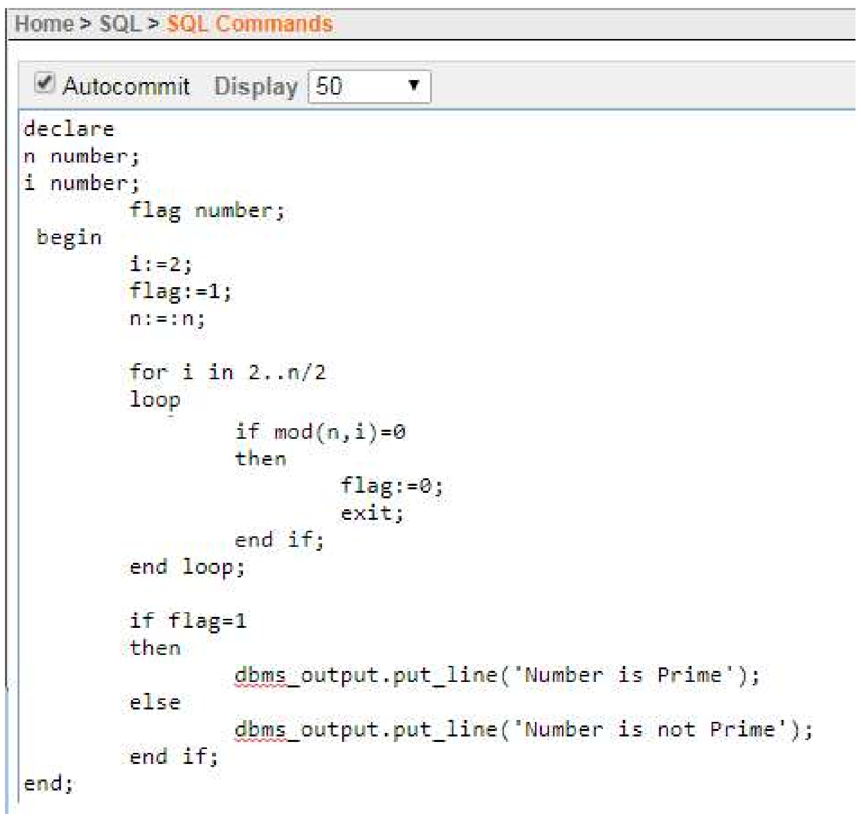
Enter values in text boxes and click on **Submit** button.

Output:


Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

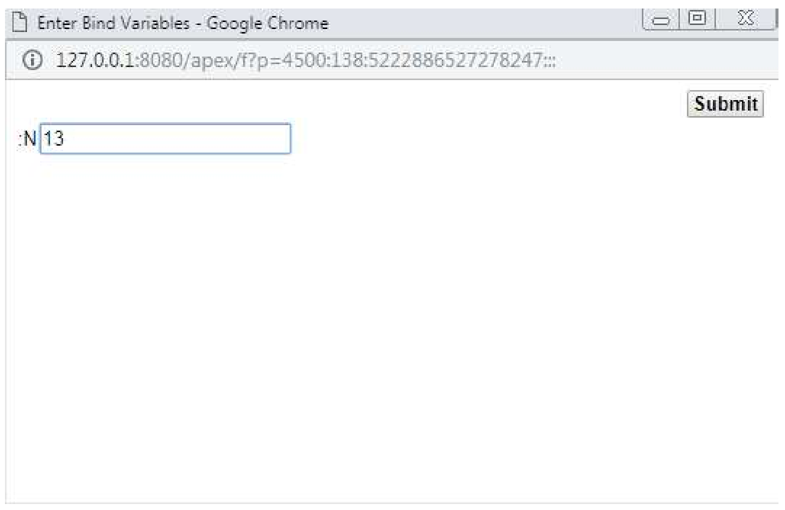
```
Sum is 110

Statement processed.

0.00 seconds
```

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



```
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;
```

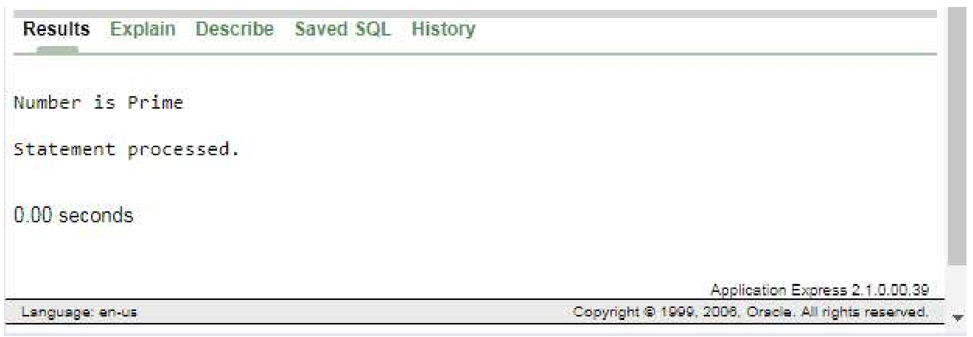
Input:


Enter Bind Variables - Google Chrome
127.0.0.1:8080/apex/f?p=4500:138:5222886527278247:::
Submit

:N 13

NOTES

Output:

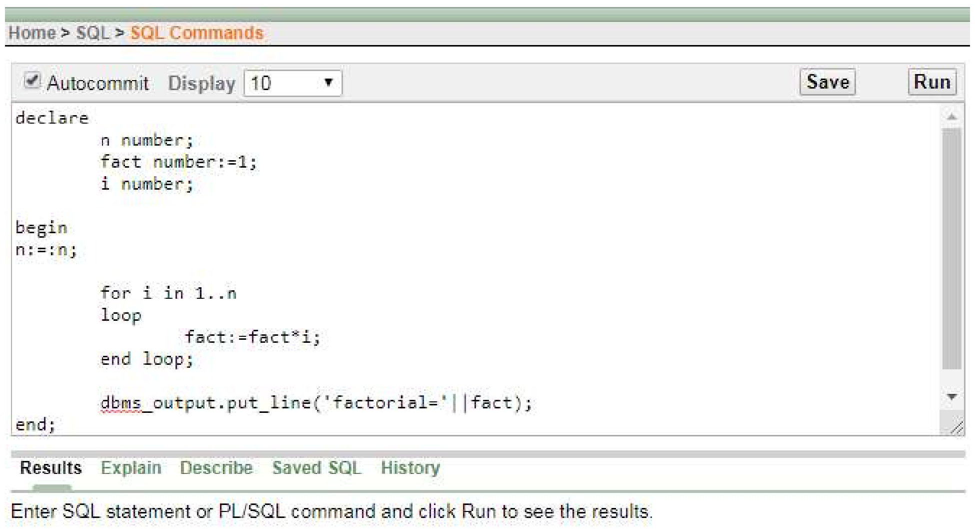

Results Explain Describe Saved SQL History

```
Number is Prime

Statement processed.
```

0.00 seconds

Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

Example: Write a PL/SQL Program to Find Factorial of a Number given number.


Home > SQL > SQL Commands
Autocommit Display 10 Save Run

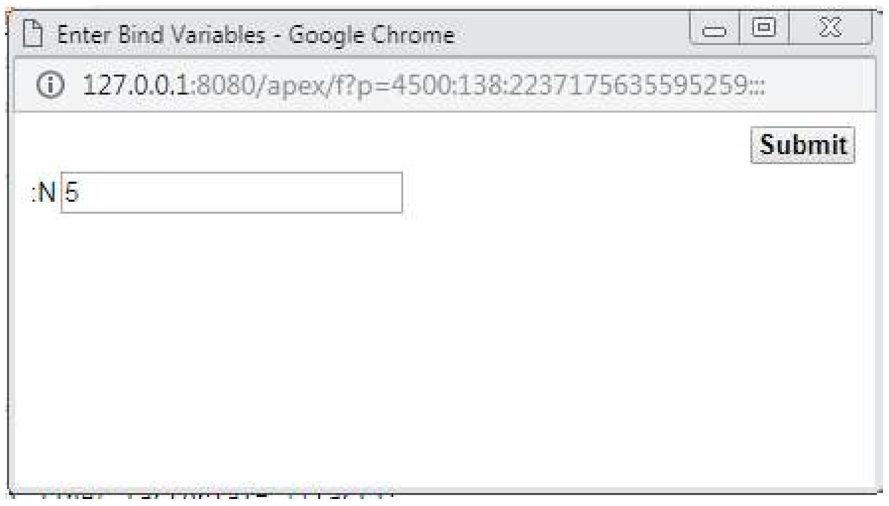
```
declare
        n number;
        fact number:=1;
        i number;

begin
n:=:n;

        for i in 1..n
        loop
                fact:=fact*i;
        end loop;

        dbms_output.put_line('factorial='||fact);
end;
```

Results Explain Describe Saved SQL History

Enter SQL statement or PL/SQL command and click Run to see the results.

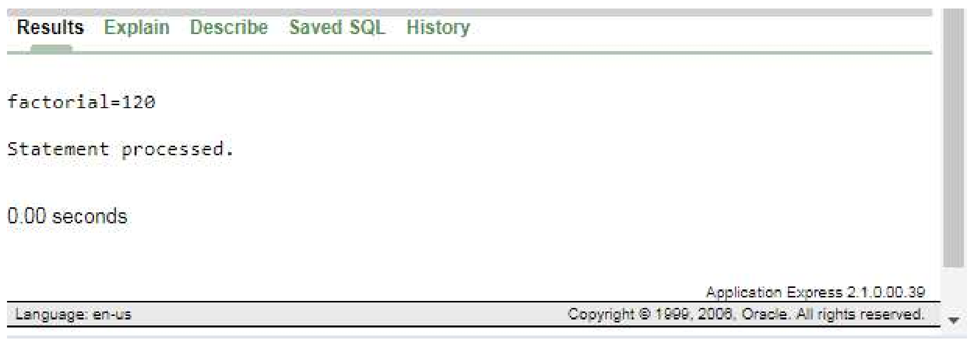
NOTES

Input:


Enter Bind Variables - Google Chrome
127.0.0.1:8080/apex/f?p=4500:138:2237175635595259:::
Submit

```
:N 5
```

Output:


Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

```
factorial=120

Statement processed.

0.00 seconds
```

Description of above program is given below:

In **Declaration Section** three variables are declared named number_1, number_2 and res of number data type.

In **Executable Section** value in *number_1* and *number_2* variables are taken by the user interactively. Here **&** symbol prompts the user to enter the value and**:=** (assignment operator) is used to assign value to variables.

Value for *res* variable is calculated to produce the sum of number_1 and number_2.

DBMS_OUTPUT.PUT_LINE is used to display output of a program.

Compile Procedure:

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

Syntax: `SQL> @ procedure_name ;`

Example: `SQL> @ search_book ;`

If the procedure does not contain any error then the system would prompt a message as follows: Procedure created.

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 exception. Error may occur due to various reasons, such as coding mistakes, hardware failure, system resources problems and many other reasons. Due to these errors program terminates abnormally.

Type of Exception:

1. Internal Exception
2. User-Defined Exceptions

The following is the list of internal exceptions:

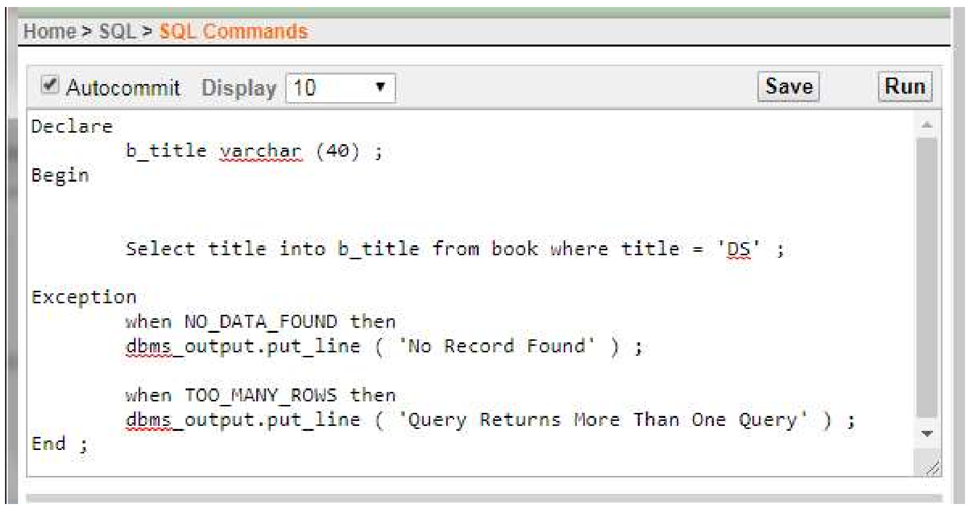
Exception	Explanation
ZERO_DIVIDE	This exception raised when PL/SQL program attempts to divide a number by zero.
NO_DATA_FOUND	This exception raised when SELECT INTO statement returns no rows while expected to return.
CURSOR_ALREADY_OPEN	This exception raised when you try to open a cursor which is already.
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.

Exception handling program is based on book table as shown below:

B_CODE	TITLE	AUTHOR	PRICE	STATUS
B004	DBMS	Korth	440	T
B006	DCN	forouzan	500	T
B001	DBMS	NAVATHE	400	T
B003	programming in c	Kavita	340	T

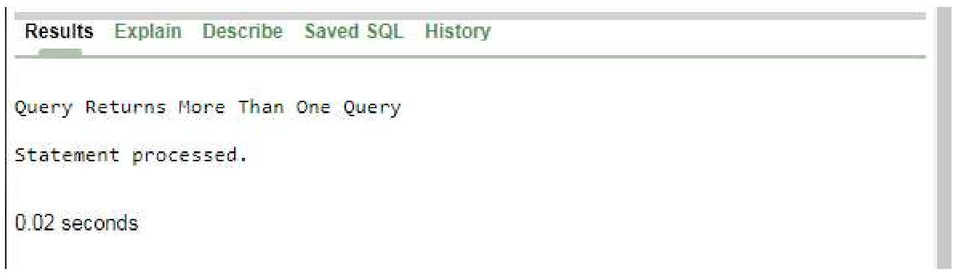
NOTES

Example: Write a program to demonstrate exception handling.



```
Declare
        b_title varchar (40) ;
Begin


        Select title into b_title from book where title = 'DS' ;

Exception
        when NO_DATA_FOUND then
        dbms_output.put_line ( 'No Record Found' ) ;

        when TOO_MANY_ROWS then
        dbms_output.put_line ( 'Query Returns More Than One Query' ) ;
End ;
```

Query returns more than one records then TOO_MANY_ROWS exception:



```
Query Returns More Than One Query

Statement processed.

0.02 seconds
```

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.

Example:

```
Write a program to demonstrate User-named Exception
handlers.
```

You could 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;
```

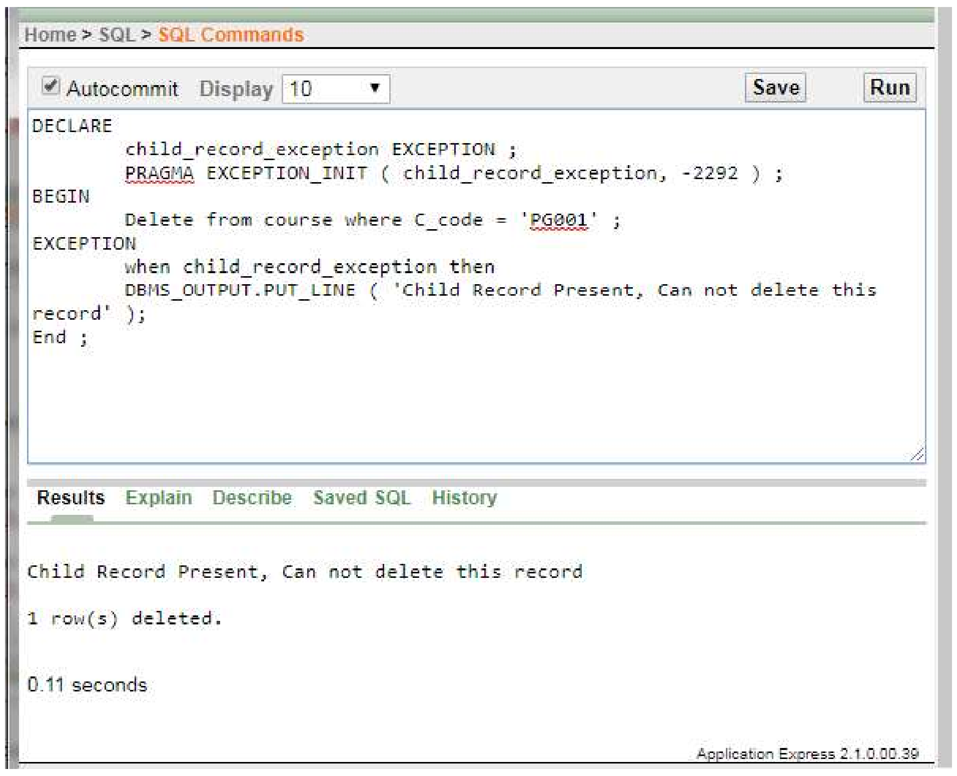
Example: Let us consider the student table and course tables.

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. You could assign a user defined name to this exception that could be handled in the exception block as given below:

NOTES

Example:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39

```
DECLARE
       child_record_exception EXCEPTION ;
       PRAGMA EXCEPTION_INIT ( child_record_exception, -2292 ) ;
BEGIN
       Delete from course where C_code = 'PG001' ;
EXCEPTION
       when child_record_exception then
       DBMS_OUTPUT.PUT_LINE ( 'Child Record Present, Can not delete this
record' );
End ;
```

```
Child Record Present, Can not delete this record

1 row(s) deleted.

0.11 seconds
```

As user could assign name to the Oracle exceptions in the above example `child_record_exception` is a user defined name of exception.

`RAISE_APPLICATION_ERROR ( )`

A user could assign an error message by using `Raise_application_error ( )` to make the error message more descriptive for the end-user. Raise_application_error ( ) is a build-in procedure.

NOTES

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

Other than the pre-defined exceptions you could define your own exception to validate data against business requirements. For example, if user wants to update total marks of student but subject marks are NULL an error must be raised 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:

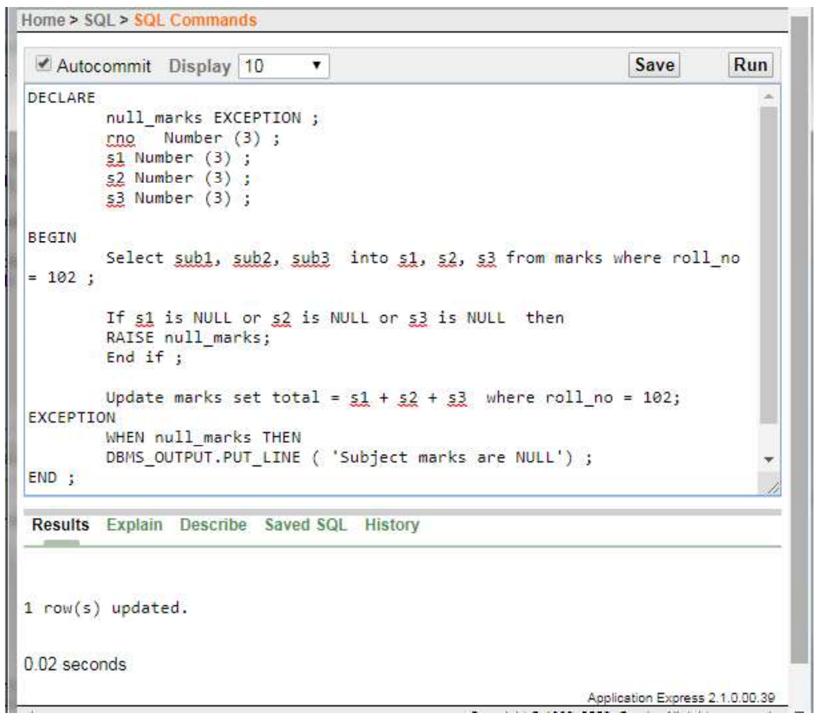
```
Create table marks (roll_no number(3), sub1 number(3),
sub2 number(3), sub3 number(3), total number(3) )
```

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	-

Example:



Autocommit ✓ Display 10

```
DECLARE
        null_marks EXCEPTION ;
        rno   Number (3) ;
        s1 Number (3) ;
        s2 Number (3) ;
        s3 Number (3) ;

BEGIN
        Select sub1, sub2, sub3  into s1, s2, s3 from marks where roll_no
= 102 ;

        If s1 is NULL or s2 is NULL or s3 is NULL  then
        RAISE null_marks;
        End if ;

        Update marks set total = s1 + s2 + s3  where roll_no = 102;
EXCEPTION
        WHEN null_marks THEN
        DBMS_OUTPUT.PUT_LINE ( 'Subject marks are NULL') ;
END ;
```

```
1 row(s) updated.

0.02 seconds
```

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.

After executing above program check students marks:

```
Select *from marks;
```

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

Try Yourself

- Write a PL/SQL code block that will accept an account number from the user and debit an amount of ₹ 2000 from the account if the account has a minimum balance of 500 after the amount is debited. The Process is too fired on the Accounts table.
- 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. **(Usage While loop)**

  Areas – radius, area.
- Write a PL/SQL block of code for inverting a number 5639 or 9365. **(Usage For loop)**
- **Usage of `for` and `goto` Statement:** 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 change s 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:

- **Static Cursors**
- **Dynamic Cursors**
- **REF Cursors**

NOTES

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

- Explicit Cursors
- Implicit Cursors

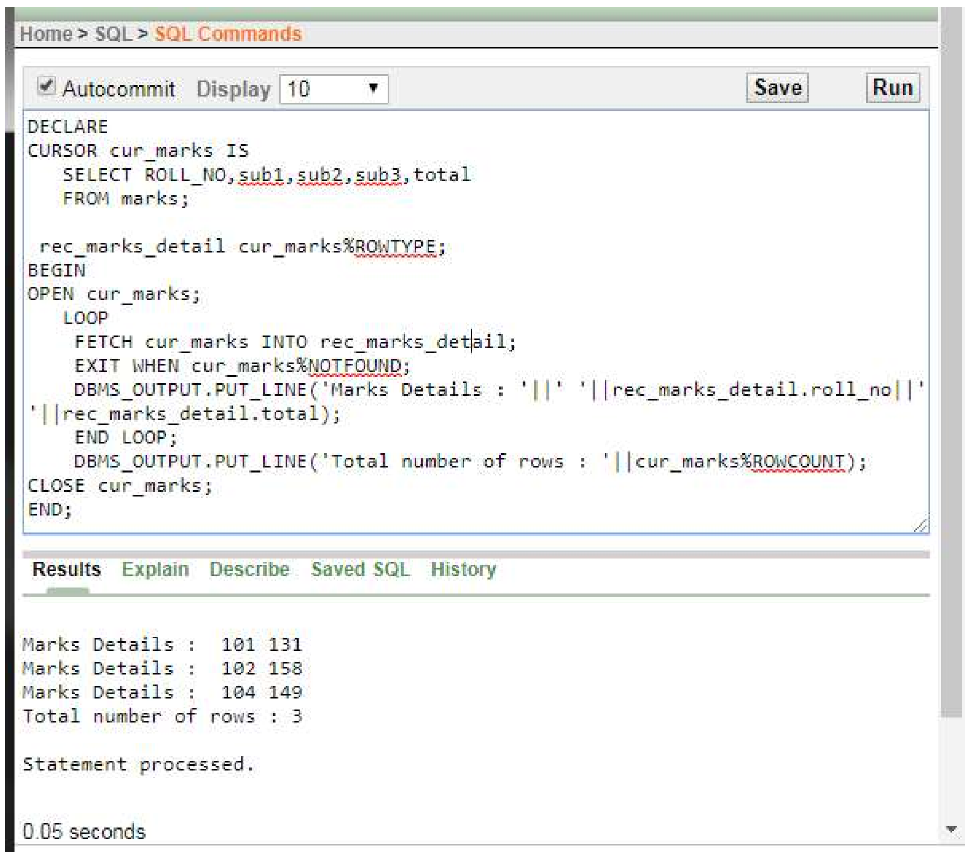
Use marks table to practice cursor:

```
Select * from marks;
```

ROLL_NO	SUB1	SUB2	SUB3	TOTAL
101	34	54	43	131
102	54	54	50	158
104	65	44	40	149

Cursors:

Q. Create a cursor to show roll number and total marks of students from marks table using cursor.



```
DECLARE
CURSOR cur_marks IS
   SELECT ROLL_NO,sub1,sub2,sub3,total
   FROM marks;

 rec_marks_detail cur_marks%ROWTYPE;
BEGIN
OPEN cur_marks;
   LOOP
    FETCH cur_marks INTO rec_marks_detail;
    EXIT WHEN cur_marks%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Marks Details : '||' '||rec_marks_detail.roll_no||'
'||rec_marks_detail.total);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Total number of rows : '||cur_marks%ROWCOUNT);
CLOSE cur_marks;
END;
```

```
Marks Details :  101 131
Marks Details :  102 158
Marks Details :  104 149
Total number of rows : 3

Statement processed.

0.05 seconds
```

Trigger:

A trigger is a PL/ SQL code block that triggered (runs) automatically an event. An event in PL/ SQL is the Data Definition Language such as INSERT, UPDATE or DELETE done on a table.

Use 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 (for example, 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
- Automatically generate derived column values
- To customize complex security authorizations.
- To permit insert, update or delete operations to an 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

Like a PL/SQL code block procedure and function also divided into different sections.

The **Syntax** for creating a trigger

```
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 could also have declarative and exception handling parts.

How to Apply Trigger:

A database trigger has three sections namely a trigger statement, a trigger body and a trigger restriction. Three of Parts of Trigger:

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

The above mentioned parts are described below:

Create a Trigger

A company XYZ has the employee detail in employee table. Company wants to have the history of all the employees how have left the organization. To store the employee history a new table emp_history is create 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

Table- Employee

Create a Duplicate Table:

To maintain the employee history a table emp_history could be created with the SQL command as written below:

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

The above command would create a new table emp_history which would contaion all the fields of employee table (as * represents all the fields of a table).

The where condition "emp_code is null" is used to create the duplicate table empty.

* Without where clause duplicate table would contain all the records of employee table.

After writing the above command system would prompt a message **Table created.**

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

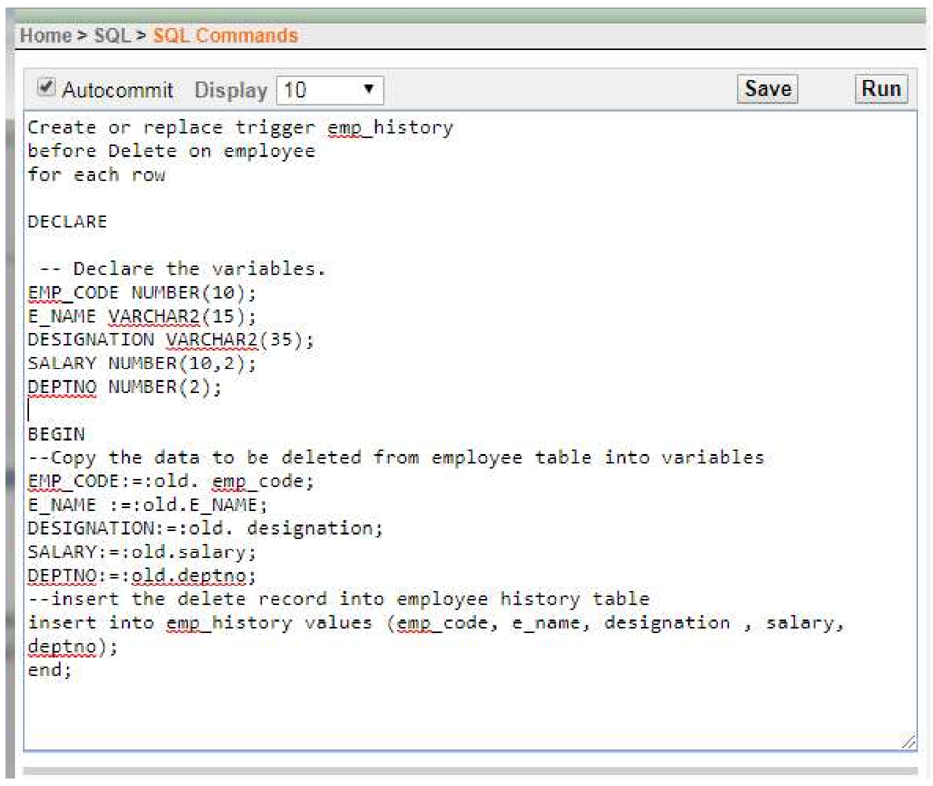
```
SQL> Desc emp_history;
```

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

Table- Emp_History

When any employee leaves the organization his or her detail would be deleted from the employee table and the same record should be inserted into emp_history table. A trigger could be associated on table employee on the event delete. The code for trigger is given below:

Example-1: Before Delete Trigger:



```
Create or replace trigger emp_history
before Delete on employee
for each row

DECLARE

 -- Declare the variables.
EMP_CODE NUMBER(10);
E_NAME VARCHAR2(15);
DESIGNATION VARCHAR2(35);
SALARY NUMBER(10,2);
DEPTNO NUMBER(2);

BEGIN
--Copy the data to be deleted from employee table into variables
EMP_CODE:=:old. emp_code;
E_NAME :=:old.E_NAME;
DESIGNATION:=:old. designation;
SALARY:=:old.salary;
DEPTNO:=:old.deptno;
--insert the delete record into employee history table
insert into emp_history values (emp_code, e_name, designation , salary,
deptno);
end;
```

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

NOTES

Now to test it:

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;
```

The above command would delete a record from employee table where emo_code is 7782. Now check whether this record has been inserted in to emp_history table or not write the following command on SQL prompt:

```
Select *from emp_history;
```

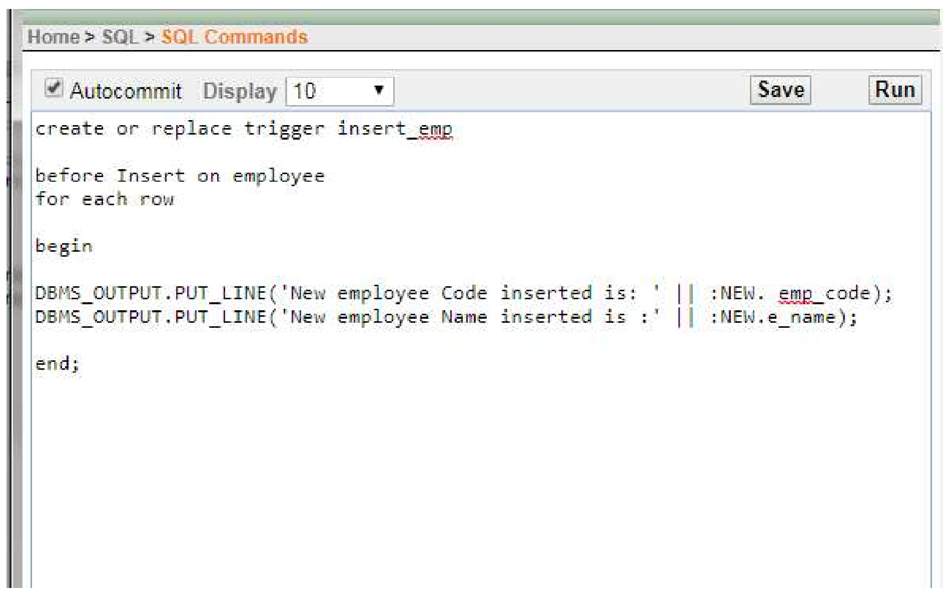
The above command would prompt the record as shown below:

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

Table- Emp_History

Example-2: 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.



```
create or replace trigger insert_emp

before Insert on employee
for each row

begin

DBMS_OUTPUT.PUT_LINE('New employee Code inserted is: ' || :NEW. emp_code);
DBMS_OUTPUT.PUT_LINE('New employee Name inserted is :' || :NEW.e_name);

end;
```

NOTES

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.

Now to test it:

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

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

When new record is inserted in to employee table system prompts the message as shown below:

```
New employee Code inserted is: 321
New employee Name inserted is: Scott
```

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

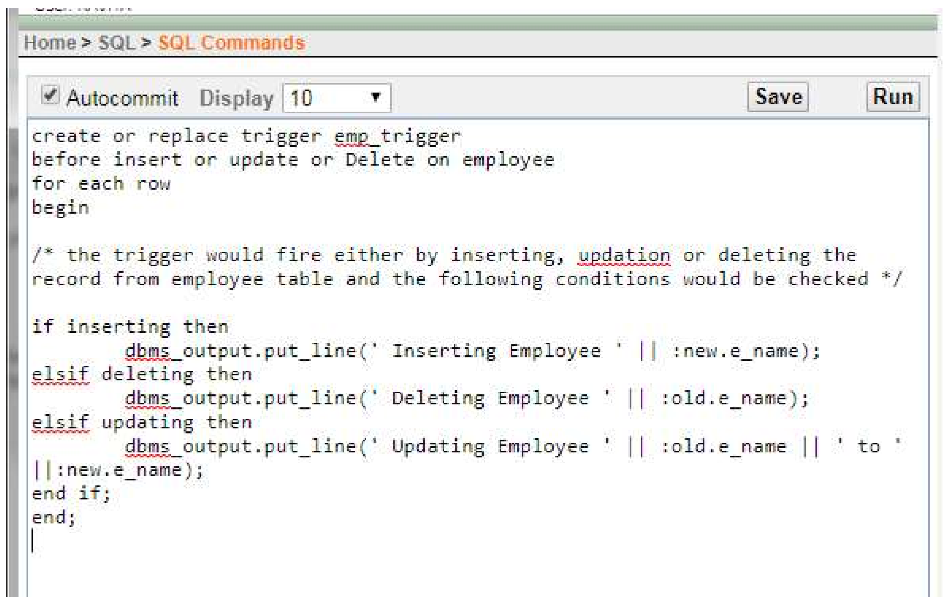
IF Statement in Trigger

As to control the PL/ SQL code execution if statement is used, 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

An **Example** of is statement in trigger is given below:



```
create or replace trigger emp_trigger
before insert or update or Delete on employee
for each row
begin

/* the trigger would fire either by inserting, updation or deleting the
record from employee table and the following conditions would be checked */

if inserting then
        dbms_output.put_line(' Inserting Employee ' || :new.e_name);
elsif deleting then
        dbms_output.put_line(' Deleting Employee ' || :old.e_name);
elsif updating then
        dbms_output.put_line(' Updating Employee ' || :old.e_name || ' to '
||:new.e_name);
end if;
end;
```

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

Different Conditions of Trigger Execution

1. Insert record in to employee table:

Syntax: `SQL> insert into employee (emp_code, e_name, designation) values (1001,'xyz', 'manager');`

When inserting a record in to employee table the first condition is true and the system would prompt a message as shown below:

```
Inserting Employee xyz
New employee Number inserted is: 1001
New employee Name inserted is: xyz
1 row created.
Deleting Employee KING
```

2. Delete record from employee table:

Syntax: `SQL> delete from employee where emp_code=7839;`

When deleting a record from employee table, the second condition is true and the system would prompt a message as shown below:

```
Deleting Employee KING
1 row deleted.
```

3. Update record from employee table:

Syntax: `SQL> update employee set e_name= 'Spark' where emp_code=7934;`

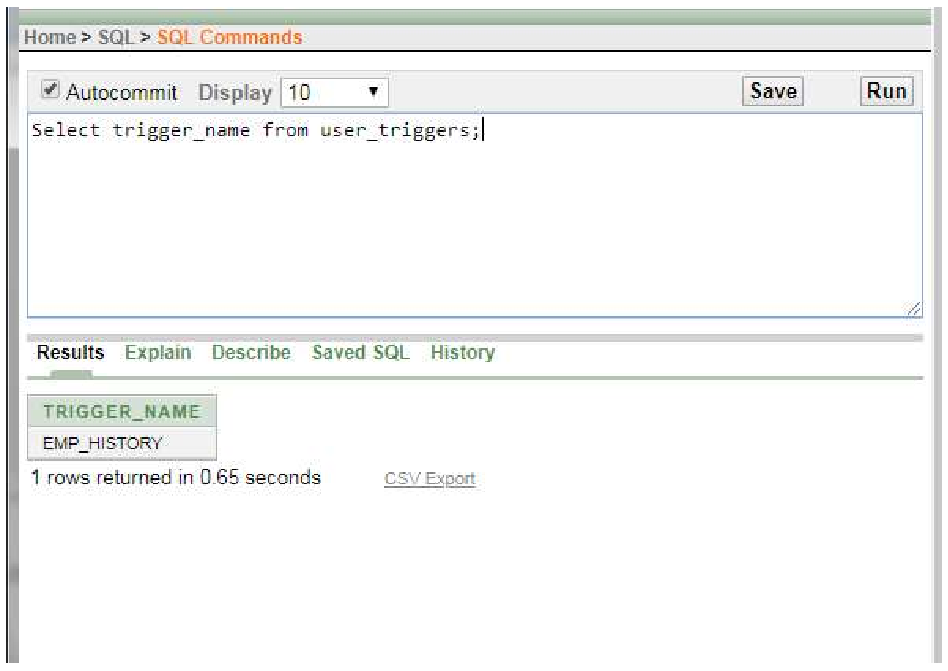
When updating a record from employee table, the third condition is true and the system would prompt a message as shown below:

```
Updating Employee MILLER to Spark
1 row updated.
```

Viewing Triggers

To view all the triggers created by the user, a data dictionary named **USER_TRIGGERS** could be used. To see all the triggers, use select statement on USER_TRIGGERS as shown below:

```
Select trigger_name from user_triggers;
```


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
CSV Export

```
Select trigger_name from user_triggers;
```

```
TRIGGER_NAME
EMP_HISTORY
1 rows returned in 0.65 seconds
```

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

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

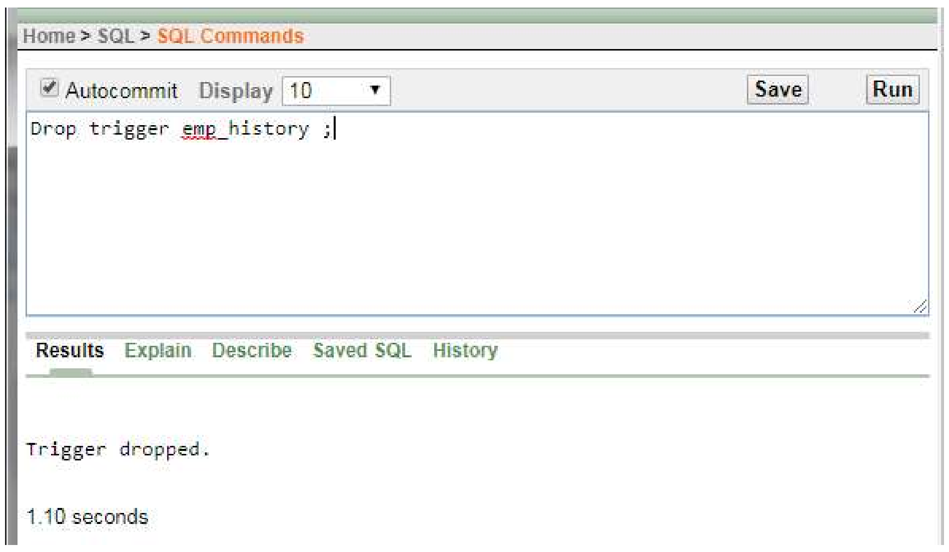
Deleting a Trigger

If trigger is more required, you can get side of it by using a Data Definition Language command **Drop.**

Syntax:

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

Example:


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Drop trigger emp_history ;
```

```
Trigger dropped.

1.10 seconds
```

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

(a) Package Specification: In package specification one could declare variables, constants, exceptions, cursors, sub-procedures and other database objects as mentioned earlier. To package specification could be created using the **CREATE PACKAGE** statement. The **Syntax** to create package specification is as follows:

```
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.

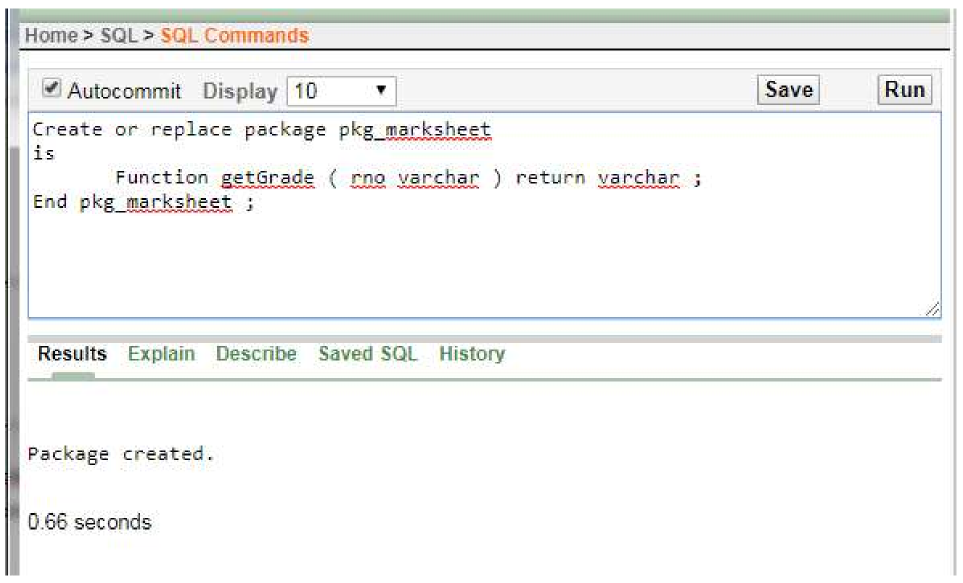
(b) Package Body: The actual implementation of declared sub-procedures and cursors is done in package body . The sub-procedures declared in package specification must be declared in package body. The **Syntax** for the CREATE BODY statement is as follows :

```
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:

Step-1


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create or replace package pkg_marksheet
is
       Function getGrade ( rno varchar ) return varchar ;
End pkg_marksheet ;
```

```
Package created.

0.66 seconds
```

The above code would 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) ;
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 ;
```

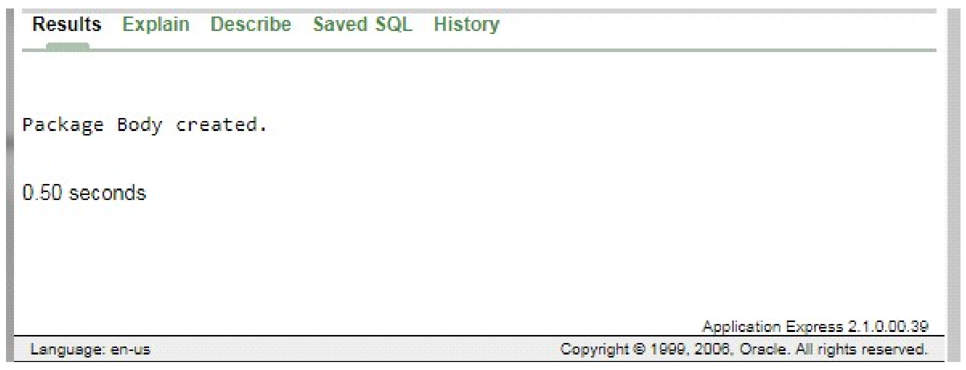
NOTES

```
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:

```
Package body created.
```

Calling Package Function: To call the function declared in package specification the reference of package name need to give as given below. The **Syntax** to call a package function is as follows :


Results
Explain
Describe
Saved SQL
History

```
Package Body created.

0.50 seconds
```

Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

The ***Example*** to call a package function is as follows:

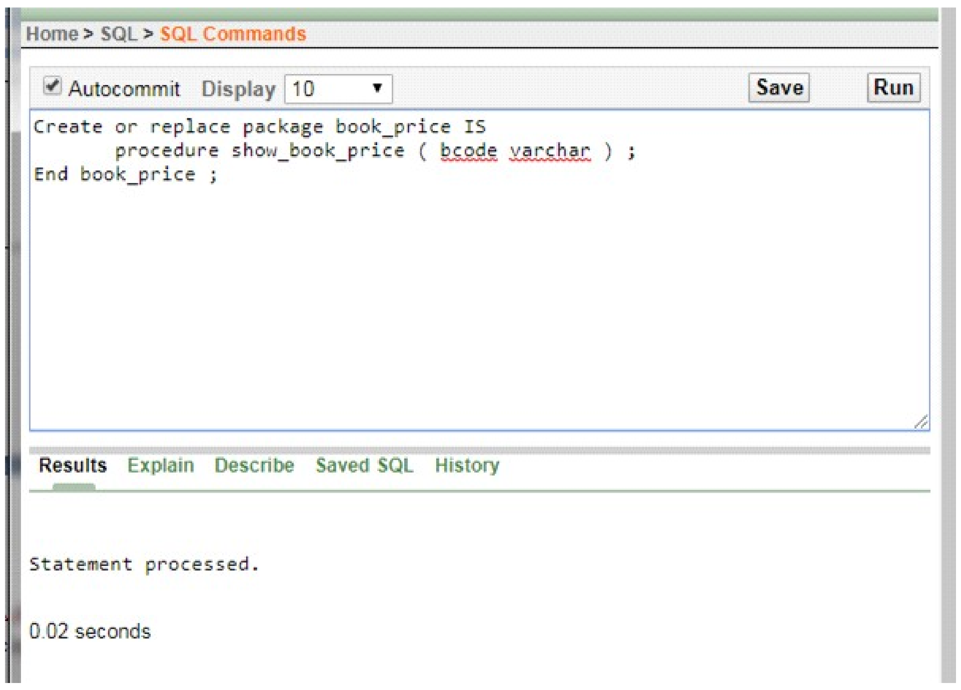
```
pkg_marksheet.getGrade ('A-08-12');
```

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 :

Step-1


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
Create or replace package book_price IS
       procedure show_book_price ( bcode varchar ) ;
End book_price ;
```

```
Statement processed.

0.02 seconds
```

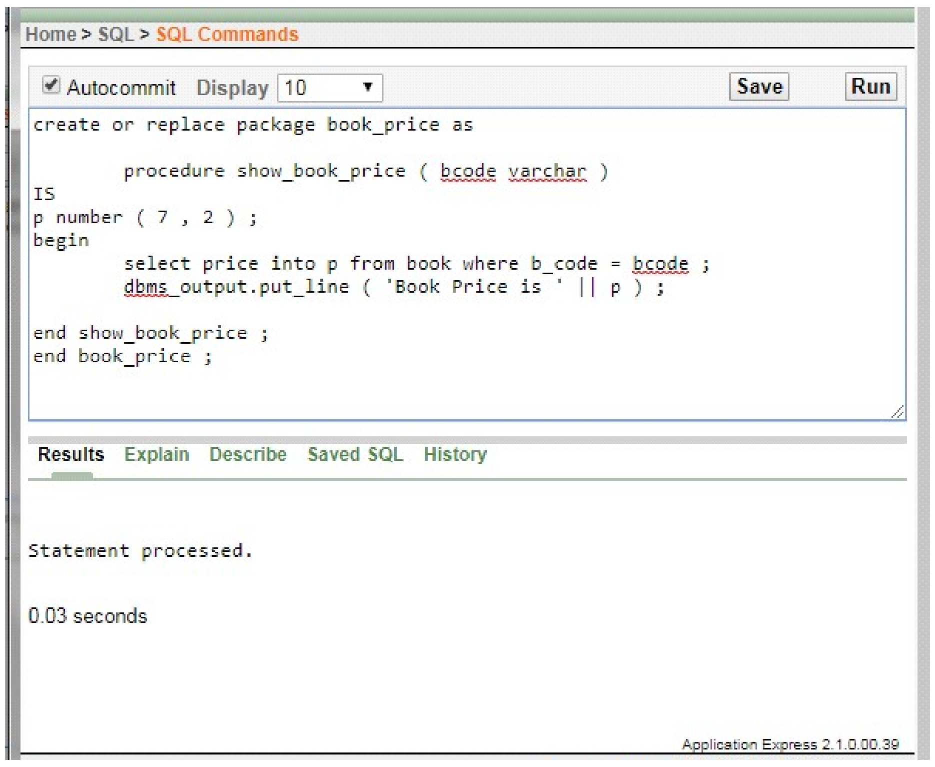
The above code would create a package with the name **book_price**. This package contains a procedure named **show_book_price**. This procedure will accept an argument of varchar type.

* Procedure cannot return any value.

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

NOTES

```
Package created.
Step-2
```


Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39

```
create or replace package book_price as

        procedure show_book_price ( bcode varchar )
IS
p number ( 7 , 2 ) ;
begin
        select price into p from book where b_code = bcode ;
        dbms_output.put_line ( 'Book Price is ' || p ) ;

end show_book_price ;
end book_price ;
```

```
Statement processed.

0.03 seconds
```

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

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 given 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.

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 functionals 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, and function or PL/SQL program. A function returns a value.

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: Functions can be very useful in many situations. For example, functions can be useful when you need to calculate the total monthly sale in different areas and of different items. Or you want to calculate the expenses of an organization. In such instances functions are useful. Consider Table, 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: 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 ;
```

BLOCK 5: APPLICATION DEVELOPMENT

This block will deal with Design and Development of various Applications including, 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.

Library Information System:

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":

```
CREATE TABLE Book_Details
(
 ISBN_Codeint PRIMARY KEY,
Book_Titlevarchar(100),
 Language varchar(10),
Binding_Idint,
No_Copies_Actualint,
No_Copies_Currentint,
Category_idint,
Publication_yearint
)
```

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,
```

NOTES

```
4. Binding_Namevarchar(50)
5. )
```

Describe Binding Table:

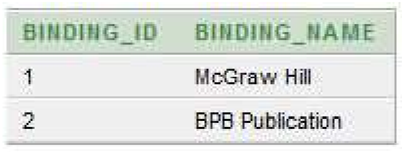
Describe binding_details

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



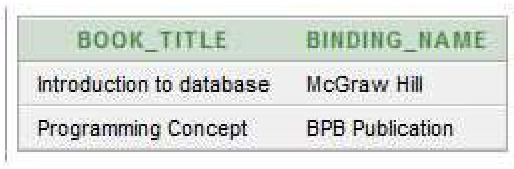
BINDING_ID	BINDING_NAME
1	McGraw Hill
2	BPB Publication

Creating Relationship between Book and Binding Table:

```
1. ALTER TABLE Book_details
2. ADD CONSTRAINT Binding_ID_FK FOREIGN KEY
   (Binding_Id)REFERENCESBinding_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:

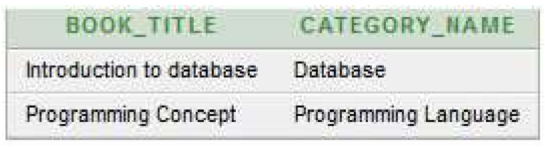
```
1. INSERT INTO CATEGORY_DETAILS VALUES
   (1,'Database');
2. INSERT INTO CATEGORY_DETAILS VALUES
   (2,'Programming Language');
```

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. selectb.Book_Title,e.Category_Name
2. fromBook_Detailsb,Category_Details e
3. whereb.binding_id = e.Category_id;
```



BOOK_TITLE	CATEGORY_NAME
Introduction to database	Database
Programming Concept	Programming Language

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)
```

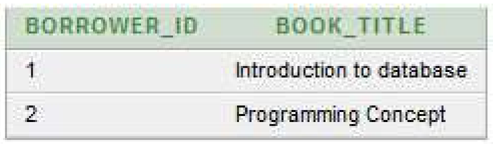
Building Relation between Book & Borrower Table:

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

NOTES

Checking Relationship:

```
selectBorrower_Details.Borrower_id,
Book_Details.Book_title
fromBorrower_Details,Book_Details
whereBorrower_Details.book_id=
Book_Details.ISBN_Code
```



BORROWER_ID	BOOK_TITLE
1	Introduction to database
2	Programming Concept

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

Creating Staff Table:

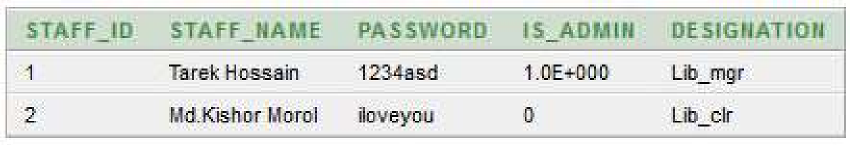
```
CREATE TABLE Staff_Details
(
 Staff_Idint PRIMARY KEY,
 Staff_Namevarchar(50),
 Password varchar(16),
 Is_Adminbinary_float,
 Designation varchar(20)
)
```

Inserting Some Data in Staff Table:

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

All Data of Staff Table:

```
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:

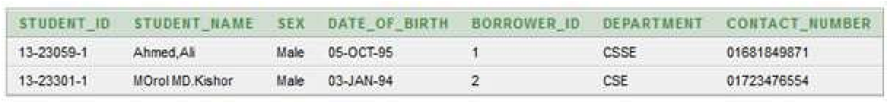
```
Create TABLE Student_Details
(
 Student_Idvarchar(10) PRIMARY KEY,
Student_Namevarchar(50),
Sex Varchar(20),
Date_Of_Birth date,
Borrower_Idint,
Department varchar(10),
contact_Numbervarchar(11)
)
```

Inserting Some Data in Student Table:

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

All Data of Student Table:

```
select *from student_details
```



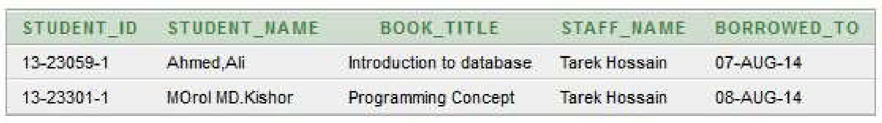
STUDENT_ID	STUDENT_NAME	SEX	DATE_OF_BIRTH	BORROWER_ID	DEPARTMENT	CONTACT_NUMBER
13-23059-1	Ahmed,Ali	Male	05-OCT-95	1	CSSE	01681849871
13-23301-1	MOrol MD.Kishor	Male	03-JAN-94	2	CSE	01723476554

Building Relationship between Student and Borrower Table:

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

Checking Full Relationship:

```
select student.student_id, student.
student_name,book.Book_ Title, staff.
staff_name, b.Borrowed_To
fromstudent_Detailsstudent,Staff_
Detailsstaff, Borrower_Detailsb,book_
details book
wherestudent.Borrower_id = b.Borrower
_id 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:

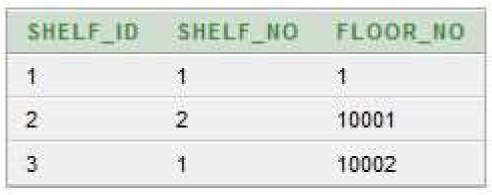
```
Create Table Shelf_Details
(
 Shelf_idint PRIMARY KEY,
 Shelf_Noint,
 Floor_Noint
);
```

Inserting some Data from Shelf Table:

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

All Data in Shelf Table:

```
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:

```
ALTER TABLE Book_Details
ADD(Shelf_Idint);

UPDATE Book_Details set Shelf_Id = 1
where ISBN_CODE = 4;

UPDATE Book_Details set Shelf_Id = 2
where ISBN_CODE = 6;

ALTER TABLE Book_Details
ADD CONSTRAINT Shelf_Id_FK FOREIGN KEY
(Shelf_Id)   REFERENCES Shelf_Details
(Shelf_Id);
```

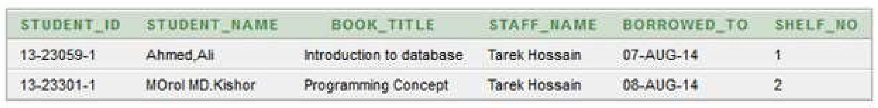
Combined All Relationship:

```
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;
```

NOTES



STUDENT_ID	STUDENT_NAME	BOOK_TITLE	STAFF_NAME	BORROWED_TO	SHELF_NO
13-23059-1	Ahmed,Ali	Introduction to database	Tarek Hossain	07-AUG-14	1
13-23301-1	MOrol MD.Kishor	Programming Concept	Tarek Hossain	08-AUG-14	2

Student marks sheet processing

Telephone directory

Gas booking and delivering

Electricity bill processing

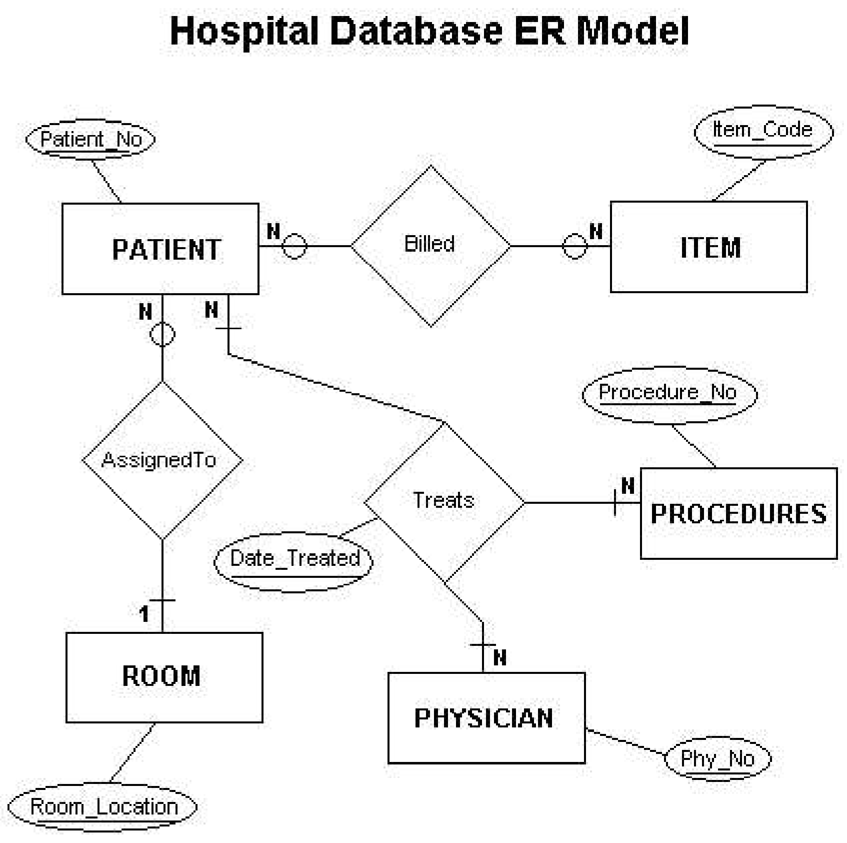
Bank transaction

Payroll processing

Personal information system

Question database and conducting quiz and personal diary

Implement the Hospital Database and execute the given queries:



NOTES

Relations (Include All the Necessary Integrity Constraints):

BILLED	
BILL NO	NUMBER(5) - PRI KEY
PATIENT_NO	NUMBER(9)
ITEM_CODE	NUMBER(5)
CHARGE	NUMBER(7,2)

TREATS	
PHY_ID	NUMBER(4) - PRI KEY
PATIENT_NO	NUMBER(4) - PRI KEY
PROCEDURE_NO	NUMBER(4) - PRI KEY
DATE_TREATED	DATE - PRI KEY
TREAT_RESULT	VARCHAR2(50)

ITEM	
ITEM_CODE	NUMBER(4) - PRI KEY
DESCRIPTION	VARCHAR2(50)
NORMAL_CHARGE	NUMBER(7,2)

PHYSICIANS	
PHY_ID	NUMBER(4) - PRI KEY
PHY_PHONE	CHAR(8)
PHY_NAME	VARCHAR2(50)

PATIENT	
PATIENT_NO	NUMBER(4) - PRI KEY
DATE_ADMITTED	DATE
DATE_DISCHARAGED	DATE
PAT_NAME	VARCHAR2(50)
ROOM_LOCATION	CHAR(4)

ROOM	
ROOM_LOCATION	CHAR(4) - PRI KEY
ROOM_ACCOMODATION	CHAR(2)
ROOM_EXTENSION	NUMBER(4)

PROCEDURES	
PROCEDURE_NO	NUMBER(4) - PRI KEY
PROC_DESCRIPTION	VARCHAR2(50)