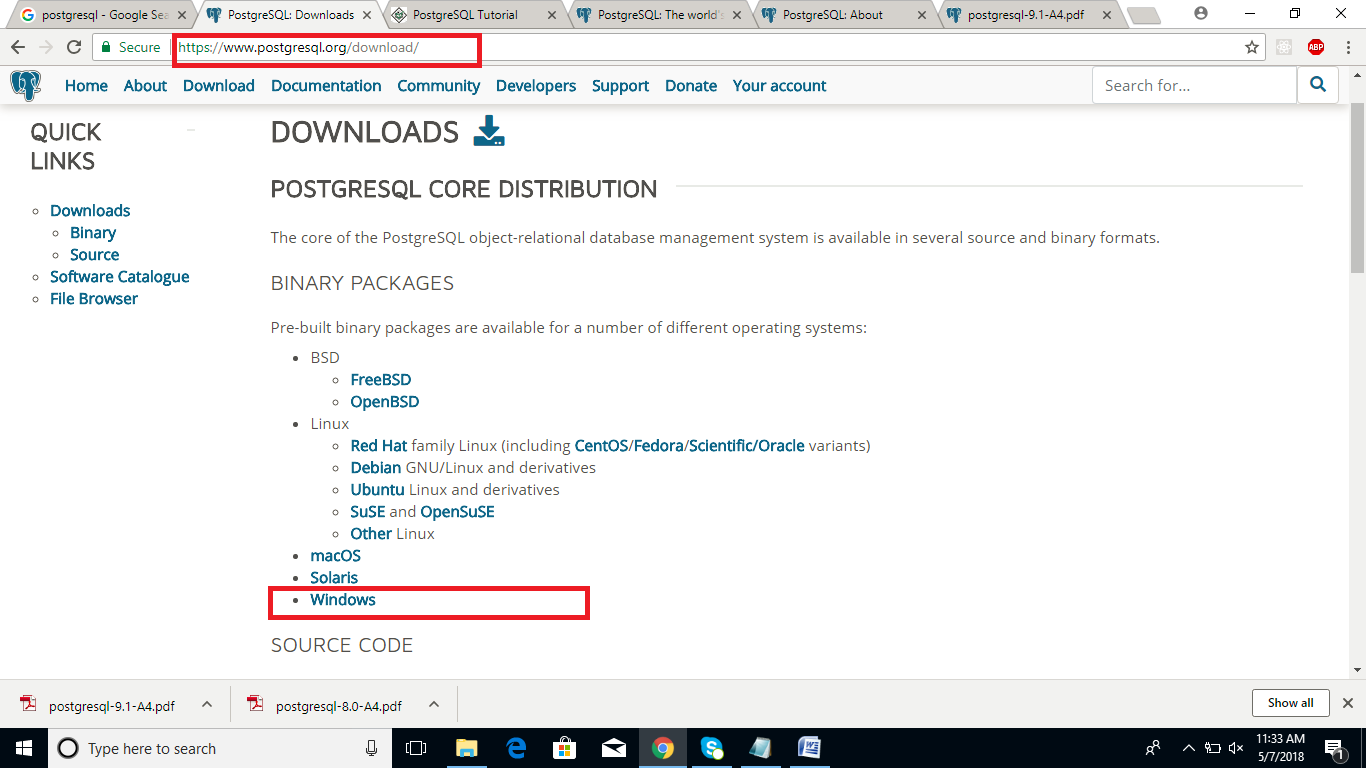
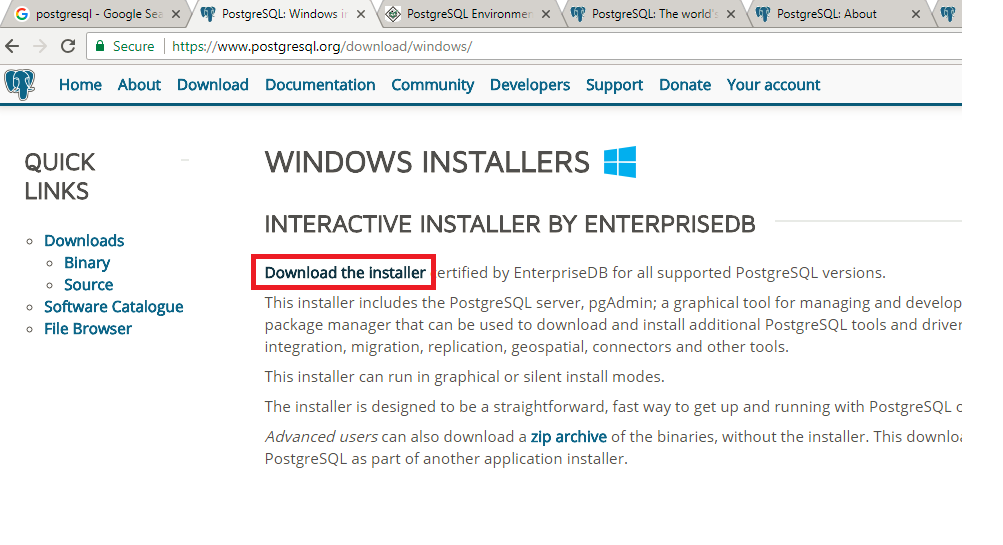
**POSTGRESQL**

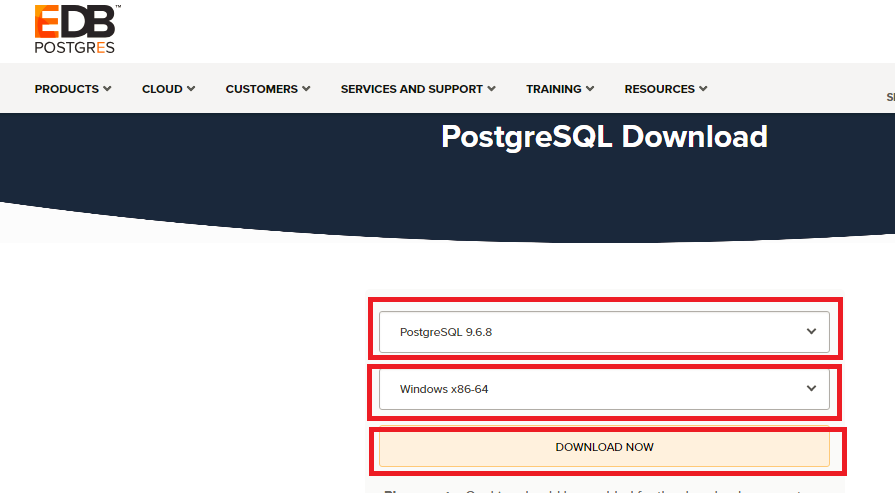
Introduction to PostgreSql

**Installation :**

For Downloading postGReSql use the link **https://postgresql.org/download**



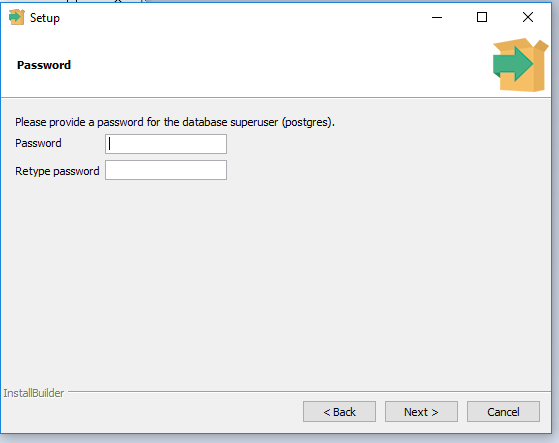




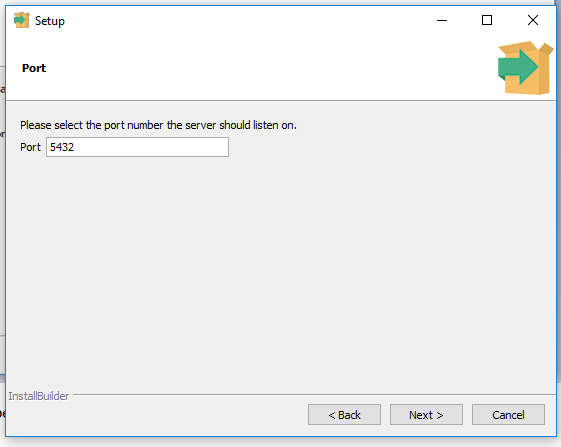
After the Exe downloaded click to start Installation



Click Next >> next >> to install in Program Files Folder



Provide the super user password >> Next



By Default It will take port number >> Click next >> Select the locale as English United States >> Next >> Finish

After Installing we can check the installed in program file s

**Datatypes in postGreSql**

While creating table, for each column, you specify a data type, i.e., what kind of data you want to store in the table fields.

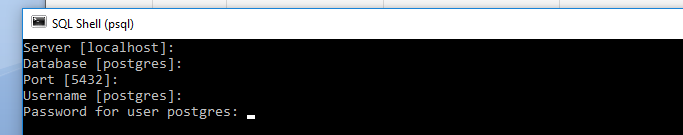
This enables several benefits −

* **Consistency** − Operations against columns of same data type give consistent results and are usually the fastest.
* **Validation** − Proper use of data types implies format validation of data and rejection of data outside the scope of data type.
* **Compactness** − As a column can store a single type of value, it is stored in a compact way.
* **Performance** − Proper use of data types gives the most efficient storage of data. The values stored can be processed quickly, which enhances the performance.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Storage Size** | **Description** | **Range** |
| smallint | 2 bytes | small-range integer | -32768 to +32767 |
| integer | 4 bytes | typical choice for integer | -2147483648 to +2147483647 |
| bigint | 8 bytes | large-range integer | -9223372036854775808 to 9223372036854775807 |
| decimal | variable | user-specified precision,exact | up to 131072 digits before the decimal point; up to 16383 digits after the decimal point |
| numeric | variable | user-specified precision,exact | up to 131072 digits before the decimal point; up to 16383 digits after the decimal point |
| real | 4 bytes | variable-precision,inexact | 6 decimal digits precision |
| serial | 4 bytes | autoincrementing integer | 1 to 2147483647 |
| money | 8 bytes | currency amount | -92233720368547758.08 to +92233720368547758.07 |
| Char |  |  | Limited chars |
| Varchar |  |  | Limited text with alphanumeric |
| Text |  |  | Unlimited text |

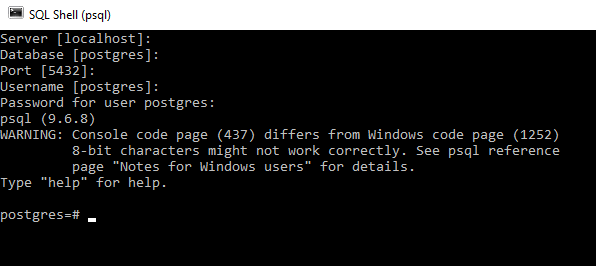
**Working With POSTGRESQL**

To open Command Shell >> Start >> Search for SQL Shell and click it >>



Press enter for server , database, port , username , and enter password for the super user . press enter

The we will get postgres prompt



We can use commands at postgreprompt

\L To List all the Databases

\c to connect to a database

\c databaseName

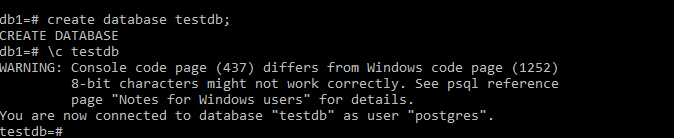
\d to list the tables in the current database

**To Create a Database**

Syn: Create Database Databasename ;

Ex: Create Database testdb; >> press enter

Aftre creating the database connect to the database using \c command



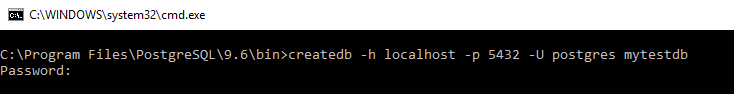
We can create database using createdb command also

**To use createdb command**

Open Command prompt >> change to the directory where postgreSQL installed and change to the bin directory >> the use the command

Createdb –h localhost –p 5432 –U postgres Databasename

And enter the password



**To Drop a DataBase**



**Using pgAdmin**

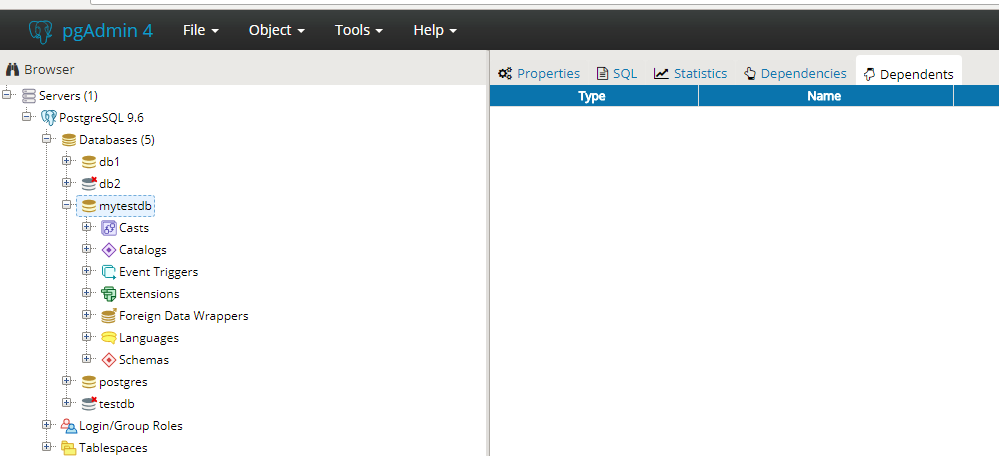
Pgadmin is a Tool to work with postgresql in GUI Ide.

Has different version , latest is 4.0 . it will be installed automatically when we install postgreSql

There are some Other tools also which will help to work with postgreSql , refer check in the official website

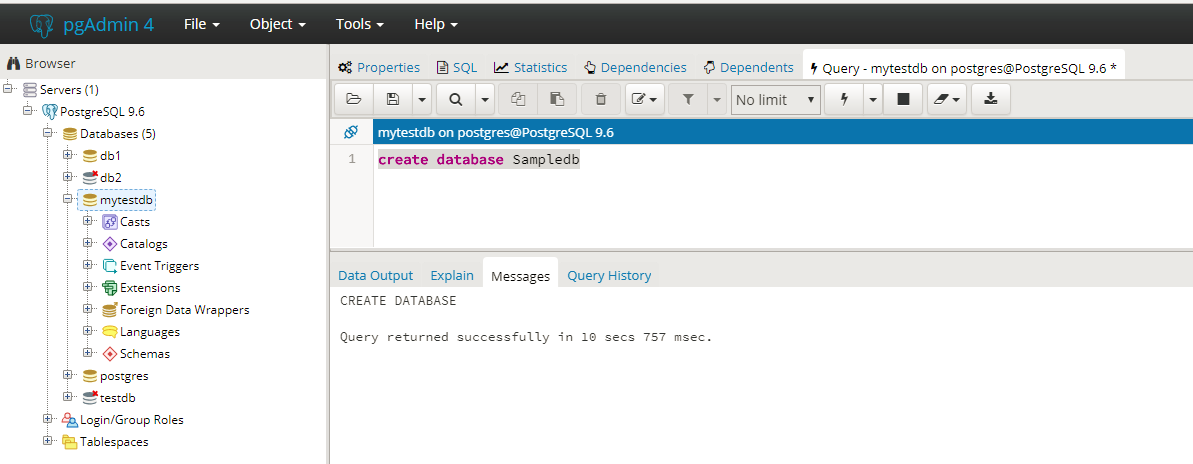
**To Open pgadmin**

Start >> pgadmin4



In the browser window select the database to connect to the database .

**To Write Queries use the Query Tool :** Tools Menu >> Query Tools



Create Table

Syn:

CREATE TABLE table\_name(

column1 datatype, column2 datatype, ..... columnN datatype, PRIMARY KEY( one or more columns )

);

Ex:

Create table Employee(

Eno int ,

Ename varchar(30),

city varchar(30),

DeptId int ,

salary money

)

After executing the command table will be created under schema public . we can the table in the browser window >> postgresql9.6 >> databases >> testdb >> shemas >> public >> tables >> employee

**To Drop the tables**

drop table employee

**Insert the data into the table**

Insert into tableName(col1, col2, col3,….) values (val1, val2, val3,….)

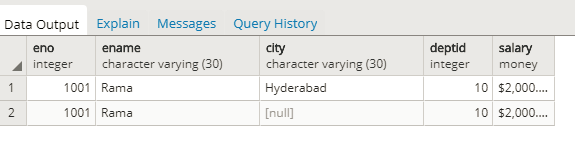
Ex:

1. insert into Employee(Eno, Ename, city , DeptId, salary)

values (1001, 'Rama','Hyderabad', 10, 2000)

1. insert into Employee(Eno,Ename ,DeptId,salary) values (1001, 'Rama', 10, 2000)

City column will be null as we have not provided the value to the city column



Ex 3 : insert into Employee values (1004, 'Rahesh','Hyderabad', 20, 2400)

Ex 4: insert into Employee values (1005, 'Singu','Hyderabad', 20, 2400),(1006, 'Suresh','Hyderabad', 30, 2500)

**Retrieving the data from Database**

**SELECT** statement is used to fetch the data from a database table, which returns data in the form of result table. These result tables are called result-sets.

Select \* from tablename - will displays all columns and all rows

Select colname1, colname2, colname3, .. .from tablename

– will display only the specified columns and all rows

Ex:

1. select \* from Employee
2. select Eno, Ename ,Salary from Employee

**Operators In PostgreSql**

An operator is a reserved word or a character used primarily in a PostgreSQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in a PostgreSQL statement and to serve as conjunctions for multiple conditions in a statement.

* Arithmetic operators - +, - , \* ,%, ^, |/ Squareroot, ||/ cube root, ! factorial
* Comparison operators - <,>,<=,>=,=,!=
* Logical operators - AND , OR , NOT
* Bitwise operators - &, |, ~

Ex:

select 2+2

select 2+2 as Addition – Renames the column

select 2^5

select 2=5

select 2<5

**Where - this clause is used to retrieve the specific rows which satisfies the Condition**

Ex:

select \* from Employee where eno = 1001

select \* from Employee where city = 'Hyderabad'

select \* from Employee where city != 'Hyderabad'

select \* from Employee where city = 'Hyderabad' AND salary ='2500'

select \* from Employee where city = 'Hyderabad' Or city = 'Banglore'

**UPDATE** Query

is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update the selected rows. Otherwise, all the rows would be updated.

Syn : Update tablename set colname1 = val1, colname2= val2,… where condition

Ex:

update employee set City = 'Delhi' where eno = 1004

update employee set city ='Delhi', salary =3000 where eno =1006

**DELETE** Query

is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete the selected rows. Otherwise, all the records would be deleted.

Ex: delete from employee where eno = 1001

Delete form employee – will removes all the records form the table

**TRUNCATE TABLE** command is used to delete complete data from an existing table. You can also use DROP TABLE command to delete complete table but it would remove complete table structure from the database and you would need to re-create this table once again if you wish to store some data.

It has the same effect as DELETE on each table, but since it does not actually scan the tables, it is faster.

Ex:

truncate table employee

**LIKE** operator

is used to match text values against a pattern using wildcards. If the search expression can be matched to the pattern expression, the LIKE operator will return true, which is **1**.

There are two wildcards used in conjunction with the LIKE operator −

* The percent sign (%)
* The underscore (\_)

The percent sign represents zero, one, or multiple numbers or characters. The underscore represents a single number or character. These symbols can be used in combinations.

Ex:

select \* from employee where ename like 'S%'

select \* from employee where ename like '\_a%'

select \* from employee where ename like '%a%'

select \* from employee where ename like '\_i%a'

select \* from employee where ename like 'Siva'

**LIMIT** clause is used to limit the data amount returned by the SELECT statement.

Ex:

select \* from employee limit 3

select \* from employee limit 3 offset 2

**ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. By Default it will consider asc

Ex:

select \* from employee order by ename asc

select \* from employee order by ename

select \* from employee order by ename desc

select \* from employee order by ename, salary asc

**GROUP BY** clause is used in collaboration with the SELECT statement to group together those rows in a table that have identical data. This is used compute aggregates that apply to these groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

SELECT –FROM-WHERE-GROUP BY-ORDER BY

Ex:

select deptid, count (eno) from employee group by deptid

select deptid, count (eno) as TotalEmployees from employee where deptid>10 group by deptid

**HAVING** Clause allows us to pick out particular rows where the function's result meets some condition.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause

SELECT –FROM-WHERE-GROUP BY-HAVING-ORDER BY

Ex:

select deptid, count (eno) as TotalEmployees from employee where deptid>10 group by deptid having count(eno)>2

**DISTINCT :** used to elimnate duplicate values from a column

Ex:

select ename from employee

select distinct ename from employee

select distinct ename,salary from employee

**Working with NULL values**

NULL means we left the field empty while creating the record . NULL is not equal to 0

We use **IS** to query on NULL values

Ex:

select \* from employee where city is NULL

select \* from employee where city is NOT NULL

**Working with Constraints**

Constraints are the rules enforced on data columns on table. These are used to prevent invalid data from being entered into the database. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column whereas table level constraints are applied to the whole table.

Defining a data type for a column is a constraint in itself. For example, a column of type DATE constrains the column to valid dates.

The following are commonly used constraints available in PostgreSQL.

* **NOT NULL Constraint** − Ensures that a column cannot have NULL value.
* **UNIQUE Constraint** − Ensures that all values in a column are different.
* **PRIMARY Key** − Uniquely identifies each row/record in a database table.
* **FOREIGN Key** − Constrains data based on columns in other tables.
* **CHECK Constraint** − The CHECK constraint ensures that all values in a column satisfy certain conditions.
* **EXCLUSION Constraint** − The EXCLUDE constraint ensures that if any two rows are compared on the specified column(s) or expression(s) using the specified operator(s), not all of these comparisons will return TRUE.

**NOT NULL Constraint**

will not allows NULL values into a column

A NULL is not the same as no data; rather, it represents unknown data.

Create table Employee(

Eno int NOT NULL,

Ename varchar(30) NOT NULL,

city varchar(30),

DeptId int ,

salary money

)

Now if we try insert data with out Eno and Ename column values it will throws error.

* NOT NULL will allows duplicate values

**UNIQUE Constraint**

The UNIQUE Constraint prevents two records from having identical values in a particular column.

Create table Employee(

Eno int UNIQUE,

Ename varchar(30) NOT NULL,

city varchar(30),

)

UNIQUE constraint will allow only one NULL value into a column

**PRIMARY KEY Constraint**

A primary key is a field in a table, which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a **composite key**.

Ex 1 :

Create table Employee(

Eno int PRIMARY KEY ,

Ename varchar(30) NOT NULL,

city varchar(30),

DeptId int ,

salary money

)

Ex 2 :

Create table Employee(

Eno int constraint pk primary key ,

Ename varchar(30) NOT NULL,

city varchar(30),

DeptId int ,

salary money

)

**FOREIGN KEY Constraint**

A foreign key constraint specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table. We say this maintains the referential integrity between two related tables. They are called foreign keys because the constraints are foreign; that is, outside the table. Foreign keys are sometimes called a referencing key.

create table department (

deptid int constraint pkd primary key ,

deptname varchar(30)

)

Create table Employee(

Eno int constraint pk primary key ,

Ename varchar(30) NOT NULL,

city varchar(30),

DeptId int ,

salary money, foreign key (DeptID) references department(deptid)

)

Create table Employee(

Eno int constraint pk primary key ,

Ename varchar(30) NOT NULL,

city varchar(30),

DeptId int constraint fke references department(deptid),

salary money

)

**CHECK Constraint**

enables a condition to check the value being entered into a record. The record will inserted into the table only when the condition is evaluating true

Ex:

Create table Employee(

Eno int constraint pk primary key ,

Ename varchar(30) NOT NULL,

city varchar(30),

DeptId int constraint fke references department(deptid),

salary money check (salary >0)

)

**Dropping Constraints**

To remove a constraint you need to know its name. If the name is known, it is easy to drop. Else, you need to find out the system-generated name. The psql command \d table name can be helpful here. The general syntax is −

ALTER TABLE table\_name DROP CONSTRAINT constraintName;