--JDBC(Java Database Connectivity)--

jdbc is an Api (application programming interface) which is use to communicate from java application to the database and also helps in communicating back from database to java application.

--data--

it is a discrete value that conveys information describing qualities, quantity, facts, statistics and other basic units of meaning.

.eg.. stock market data, weather information,video,audio,images,locations.

--data storage--

1.the process of storing the data digitally is known as data storage.

2.data storage deals with how and where we store the data digitally.

To store the data digitally we have two systems

1.file system

2.database system

--adavatages of databse over file--

1.file->less secure

1.database->more secure

2.file->no languages are available to communicate with file

2.database->using query language we can communicate with database

3.file-> redundancy can be present

3.database->we can avoid redundancy

4.file->if data is distributed in different files, it is not easy to maintain

4.database-> due to centralized nature it is easy to manage the data

5.file->less data consistency

5.db->more consistency

6.file->support for complex scenarios or transaction is not available

6.db->support for complex scenarios or transaction is available

-----DataBase----

database is a place or media where we can store the data.

eg. relational database, no SQL database, graph database, cloud database, open-source database, data warehouse

---how jdbc works---

1.jdbc is an api which is used to communicate with a particular database

2.jdbc api alone cannot be able to communicate with particular database

3.jdbc Api uses driver software of particular database to communicate

4.jdbc api is independent of database where driver softwares are dependent on database

5.jdbc api provides in built mechanism to load and register respective java packages to the driver manager.

6.jdbc api has in built mechanism which helps us in creating connection to the database.

---steps to create jdbc connection---

1.load the driver

2.create connection

3.create statement

4.execute query

5.close the connection

->Step 1: Load the driver

1.we can load the driver of respective DB into the memory using public static forName method by passing fully qualified name of

driver(forname name present inside Class)

```
Class.forName("org.postgresql.Driver");
```

->Step 2: Create connection

we need to create a connection in order to execute any queries of the DB we can create connection by using getConnection method

present inside driverManager helper class while creating a connection we need to pass URL to specify or to locate the DB and

Username and password for validation

```
Connection connection =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/demodb","postgres","ro
ot");
```

->Step 3: Create Statement

to pass the query from java app to DB we need to create statement. we can create a statement using connection object.

```
Statement statement = connection.createStatement();
```

->step 4: Execute query

to execute the query we use execute method of statement.

```
String sql="";
statement.execute(sql);
```

->Step 5: Close the connection

if we don't close the connection data might not be getting stored into the table properly, we may not be able to create new connection,

data may get leaked because these reasons we need to close the connection

```
connection.close();
```

ResultSet

1.resultset is an interface present inside java.sql package

2.result set is used to store the data which is fetched form the table

3.we can get result set object by calling executeQuery method

4.resultset acts as pointer or cursor

5.initially result set will be pointing above the record

6.to move result set or pointer to the record we call next() method the return type of next method is Boolean it will return true if record exists else returns false.

7.resultset object contains getX methods (getter methods) to fetch the records from the particular column

8.getter method accepts column index or column label

(Note:we can create result set object in two ways

1.using executeQuery method

2.getResultset Method)

to create object of result set using getresultset method we need to call execute method first by select query then call getresultset method using statement object

```
eg.stm.execute(sql);
  stm.getResultSet();
```

-------Driver Manager--------

1.Driver manager is a helper class present in java.sql package

2.it acts like a mediator between database and java application

3.driver manager contains static methods which will helps us to do particular tasks like creating connection, registering the driver, etc.

---|||||||||||||||||||||||||||||||||||||||---

Step 1: load or register the driver

we can achieve first step in two ways

1.loading the driver- we can load the driver using public static forName method present in Class by passing qualified name of the driver

eg.Class.forName("org.postgresql.Driver");

2.registering the driver- we can register the driver to the driver manager and ask driver manager to load it into the memory.

driver manager contains register driver method which accepts driver object.

driver is a class present inside org.postgresql package.

```
eg.     try{Driver driver = new Driver();
DriverManager.registerDriver(driver);}
catch(Exception e){
 e.printStackTrace();
}
```

Step 2: Create connection

1.to create a connection we use getconnection method present inside DriverManager.

2.getconnection is a overloaded method.

3.it thows sql exception.

we can achieve second step i.e creating connection using following ways

I. using getConnection(String url, String username,String password);

eg.String url="jdbc:postgresql://localhost:5432/demodb";

  String user="postgres";

  String pwd="root";

try{

Class.forName("org.postgresql.Driver");

Connection conn= DiverManager.getConnection(url,user,pwd);

}catch(Exception e){

e.printstacktrace();

}

II.using getConnection(String url);

1.we need to pass username password and url as a String literal i.e. query parameter.

2.query parameters are the parameters which we pass at the end of the url in key value pair.

eg.Connection conn=
getConnection("jdbc:postgresql://localhost:5432/demodb?user=postgres&password=roo
t");

III.using getConnection(String url,Properties info)

1.Properties - it is a class present inside java.util .we can set the properties to the stream and we can get properities from stream.

we can create object of properties using no argumented constructor. properties is a subclass of HashTable and stores data in th form of key value pair

Map<k,v> <- HashTable<k,v> <- Properties<k,v>

we can get the data from properties using get property("key") method

to load data from inputstream object to the properties object we use load method

eg.

```
try{
 FileInputStream  fin = new FileInputStream("db.properties");
Properties  prop=new Properties();
prop.load(fin);
String driver=prop.getProperty("driver");
Class.forName(driver);
String url=prop.getProperty("url");
Connection conn= DriverManager.getConnection(url,prop);
}catch(Exception e){
e.printStackTrace();
}
```

Step 3: Create Statement

Statement:

1.statement is an interface present inside java.sql package

2.mainly we use statement to execute static queries

3.we can create an object of statement using createStatement() method present inside connection object

4.createStatement throws sql exception

eg. Statement stm = con.createStatement();

PrepareStatement:

1.it is a sub interface of statement interface

2.mainly we use preparestatement to execute dynamic query

--**Dynamic query**

1.the queries which accepts the values in runtime and created with placeholders known as dynamic query

2.we can pass values to the placeholders of dynamic query using setter methods of preparestatement

3. we can create object of preparestatement using prepareStatement(String sql)

eg.

```
String url="jdbc:postgresql://localhost:5432/demodb?user=postgres&password=root";

try{

Class.forName("org.postgresql.Driver");

Connection con=DriverManager.getConnection(url);

String sql="INSERT INTO myperson VALUES(?,?,?)";

PreparedStatement ps= con.prepareStatement(sql);

ps.setLong(1,62876833);

ps.setString(2,"Ritik");

ps.setLong(3,7237869);

ps.execute();

con.close();

}catch(Exception e){

e.printStackTrace();

}
```

Difference between statement and prepare statement

Statement	PreparedStatement
It is used when only one query to be executed	It is used when multiple queries to be executed
It used to execute normal queries	It used to execute dynamic queries
It is a base interface	It extends statement interface
You cannot pass parameters at runtime	You can pass parameters at runtime

Step 4: Execute the query

We can execute the queries using following methods

1. Boolean execute() method
   a. Execute method throws SQL exception
   b. Return type of return type is Boolean
   c. Query method return true if it is able to return data or return false
   d. We can execute select and non select queries
2. Resultset executeQuery() method
   a.it use to execute only select queries
   b.it will execute select query and return resultset object
   c.it throws exception if we execute non select queries

3. Int executeUpdate() Method

   a. we can execute only non-select queries (DML) using executeUpdate

b. return type of executeupdate() method is int

c.it returns number of rows affected while executing perticular quer

d.it throws SQL exception if we execute select query

Step 5: close connection

It is not best practice to close the connection in try block beacause if anythinng went wrong after creating the connection. connection will not be getting closed. To close the connection

A. We use Finally block
B. Try with resource

Using Finally block

Eg.

```
String url=”jdbc:postgresql://localhost:5432/demodb?user=postgres&password=root”;

Connection con=null;

Try{

Class.forName(“org.postgresql.Driver”);

con=DriverManager.getConnection(url);

Statement stm= con.createStatement();

String sql=”UPDATE myperson SET person_phone=123 WHERE person_name=’R’”;

Int update=stm.executeUpdate(up);

System.out.println(update);

}

Catch(Exception e){

e.printStackTrace();

}Finally{

Try{

if(con!=null){

Con.close()

}catch(Exception e){

e.printstackTrace();

}
```

Using try with resource

1.after java 8 try with resource was introduced in try with resource compiler itself will close the resource

Eg.

```
String url=”jdbc:postgresql://localhost:5432/demodb?user=postgres&password=root”;

Try{

Class.forName(“org.postgresql.Driver”);

}

Catch(Exception e){

e.printStackTrace();

}

Try(Connection con =DriverManager.getConnection(url)){

Statement stm= con.createStatement();

String sql=”UPDATE myperson SET person_phone=123 WHERE person_name=’R’”;

stm.execute(sql);

}catch(Exception e){

e.printstackTrace();

}
```

Batch Execution/Batch Processing

1.Whenever we want to execute multiple queries in single shot we make use of batch execution

2.Batch execution helps us to improve performance of application and also in data consistency

3. to add queries to the batch to use addBatch(String sql);

4. to execute the batch we use executeBatch() method present inside statement object

5. the return type of execute batch is integer array it contains the number of rows which got affected by each and every query present inside batch

Eg.

```
String url=”jdbc:postgresql://localhost:5432/demodb?user=postgres&password=root”;
```

```
Try{
Class.forName(“org.postgresql.Driver”);
}
Catch(Exception e){
e.printStackTrace();
}
Try(Connection con =DriverManager.getConnection(url)){
Statement stm= con.createStatement();
String sql1=”INSERT INTO myperson VALUES(16573568,’Tunga’,6586993);
String sql2=”UPDATE myperson SET person_phone=482482 WHERE
person_name=’Rahul’”;
String sql3=”DELETE FROM myperson WHERE person_adhar_id=6345”;
Stm.addBatch(sql1);
Stm.addBatch(sql2);
Stm.addBatch(sql3);
int[] result=Stm.executeBatch();
System.out.println(Arrays.toString(result));
}catch(Exception e){
e.printStackTrace();
}
```

Stored Procedure

Whenever we are dealing with complex scenarios and multiple tables, we make use of stored procedure

2.It is a block of code which will be compiled and executed on the database server

3.they accepts input parameters, output parameters and both input & output.

4.we can call the stored procedure using following command

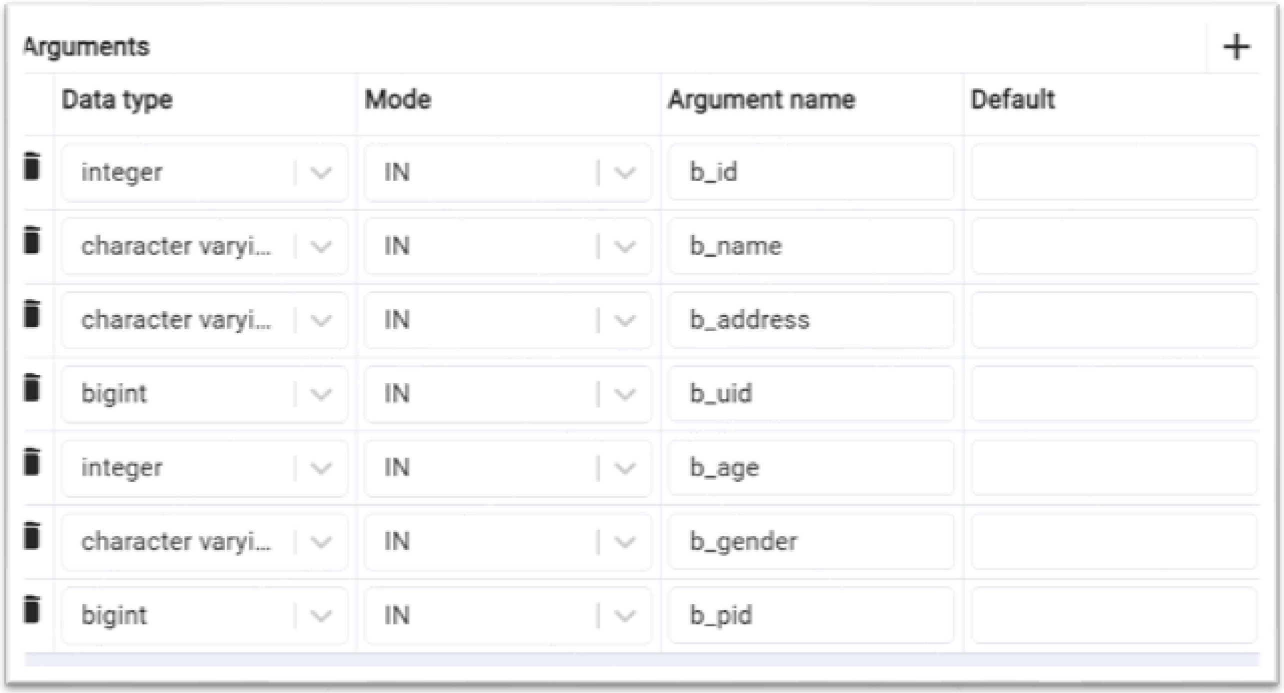
Call procedure_name(actual_values/actual_data);

To create Stored procedures in database:

Open schemas ->right click on procedure ->go to create a procedure

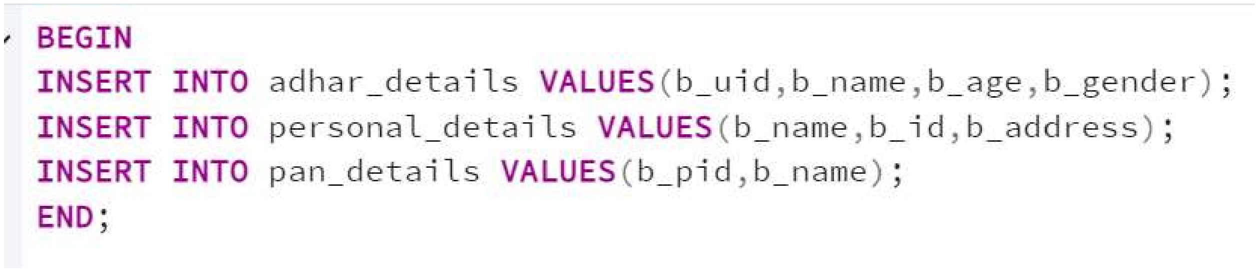
- Note: general

1. Name

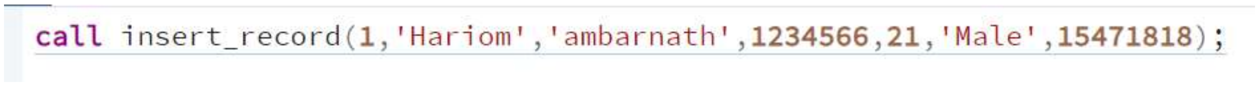


Arguments

Data type	Mode	Argument name	Default
integer	IN	b_id	
character varyi...	IN	b_name	
character varyi...	IN	b_address	
bigint	IN	b_uid	
integer	IN	b_age	
character varyi...	IN	b_gender	
bigint	IN	b_pid	



```
BEGIN
INSERT INTO adhar_details VALUES(b_uid,b_name,b_age,b_gender);
INSERT INTO personal_details VALUES(b_name,b_id,b_address);
INSERT INTO pan_details VALUES(b_pid,b_name);
END;
```



```
call insert_record(1,'Hariom','ambarnath',1234566,21,'Male',15471818);
```

Callable Statement

1. It is an interface present inside java.sql package
2. It used to call stored procedure and functions
3. We can get callable statement object by using preparecall() method present inside connection object
4. Callable statement has setter methods to set values to the placeholders
5. Callable statement is sub interface of preparedstetement interface

   Eg.

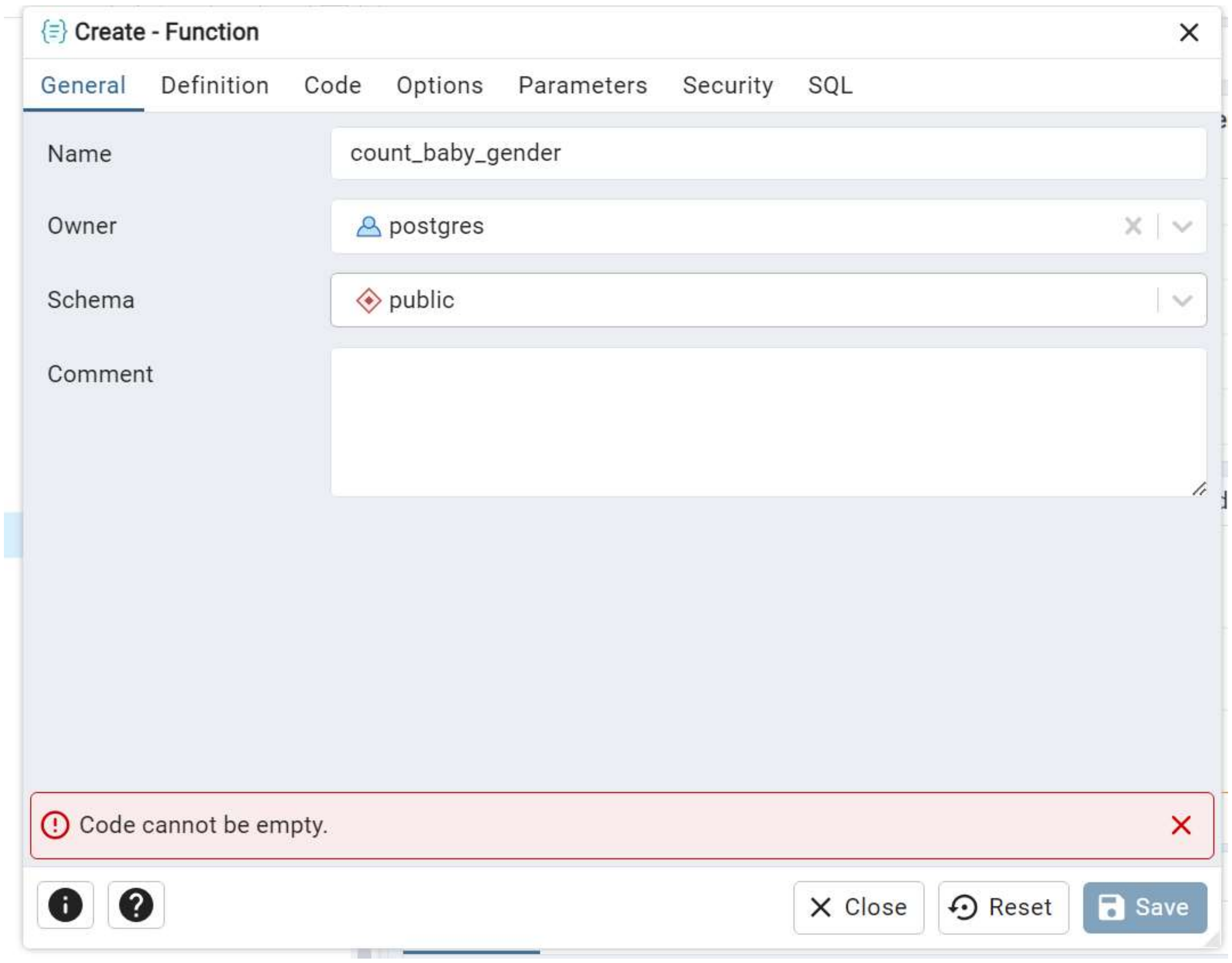
   Program to call stored procedure from java application

```
String
url=”jdbc:postgresql://localhost:5432/demodb?user=postgres&password=root”;
Try{
Class.forName(“org.postgresql.Driver”);
}catch(Exception e){
e.printStackTrace();
}
Try(Connection con = DriverManager.getConnection(url)){
CallableStatement cstm= con.prepareCall(“call baby_record_insert(?,?,?,?,?,?,?)”);
Cstm.setInt(1,201);
Cstm.setString(2,”ritik”);
Cstm.setString(3,”male”);
Cstm.setInt(4,4);
Cstm.setLong(5,672987398);
Cstm.setString(6,”kalyan”);
Cstm.setString(6,”AFFG5675”);
Cstm.execute();
}catch(Exception e){
e.printStackTrace();
}
```

Functions

1.Whenever we are going to have calculations in complex scenarios we make use of functions

2. Function should have return type
3. Function is block of code which will be present and compiled at database side
4. To create function in database



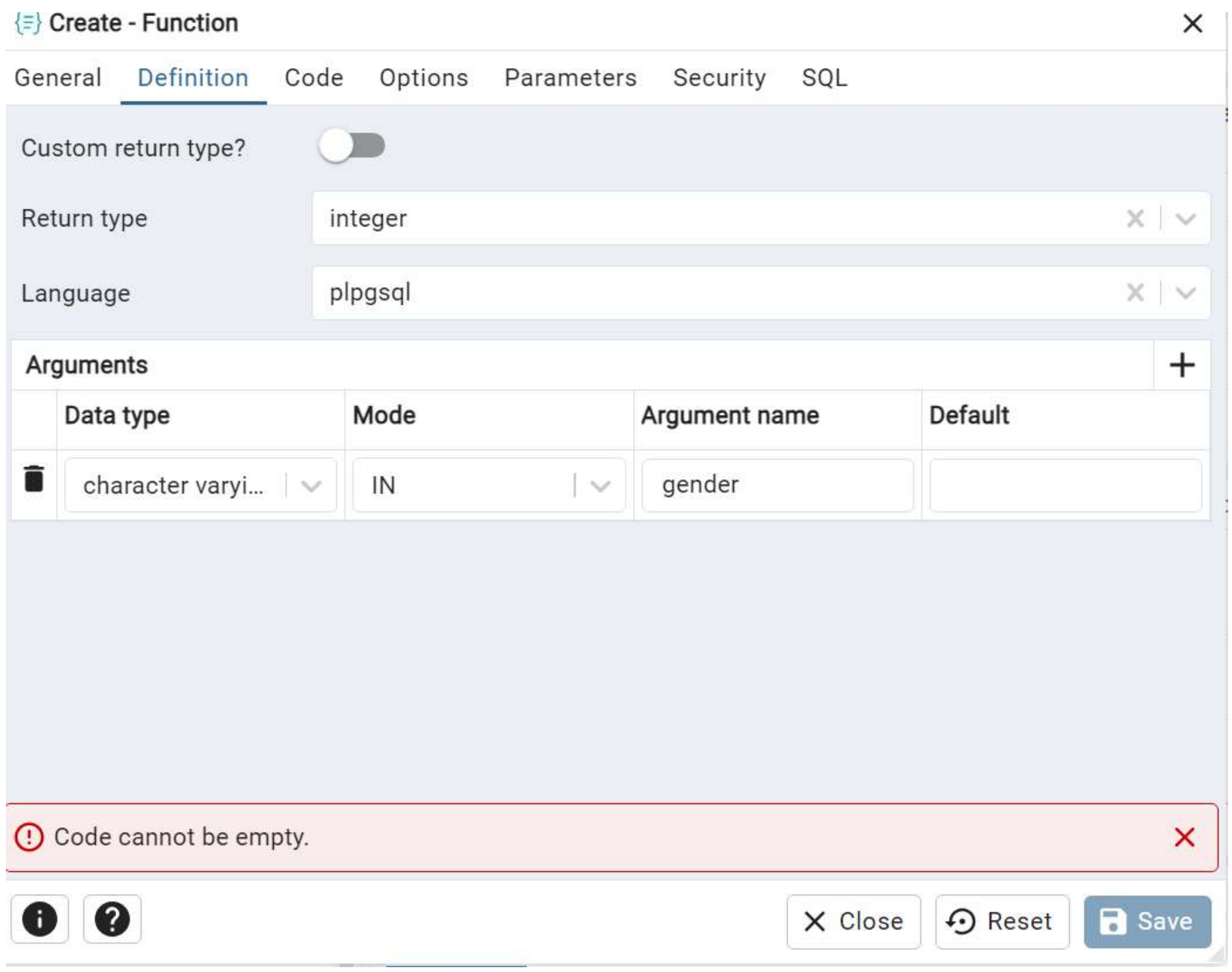
Name: count_baby_gender

Owner: postgres

Schema: public

Comment:

Code cannot be empty.



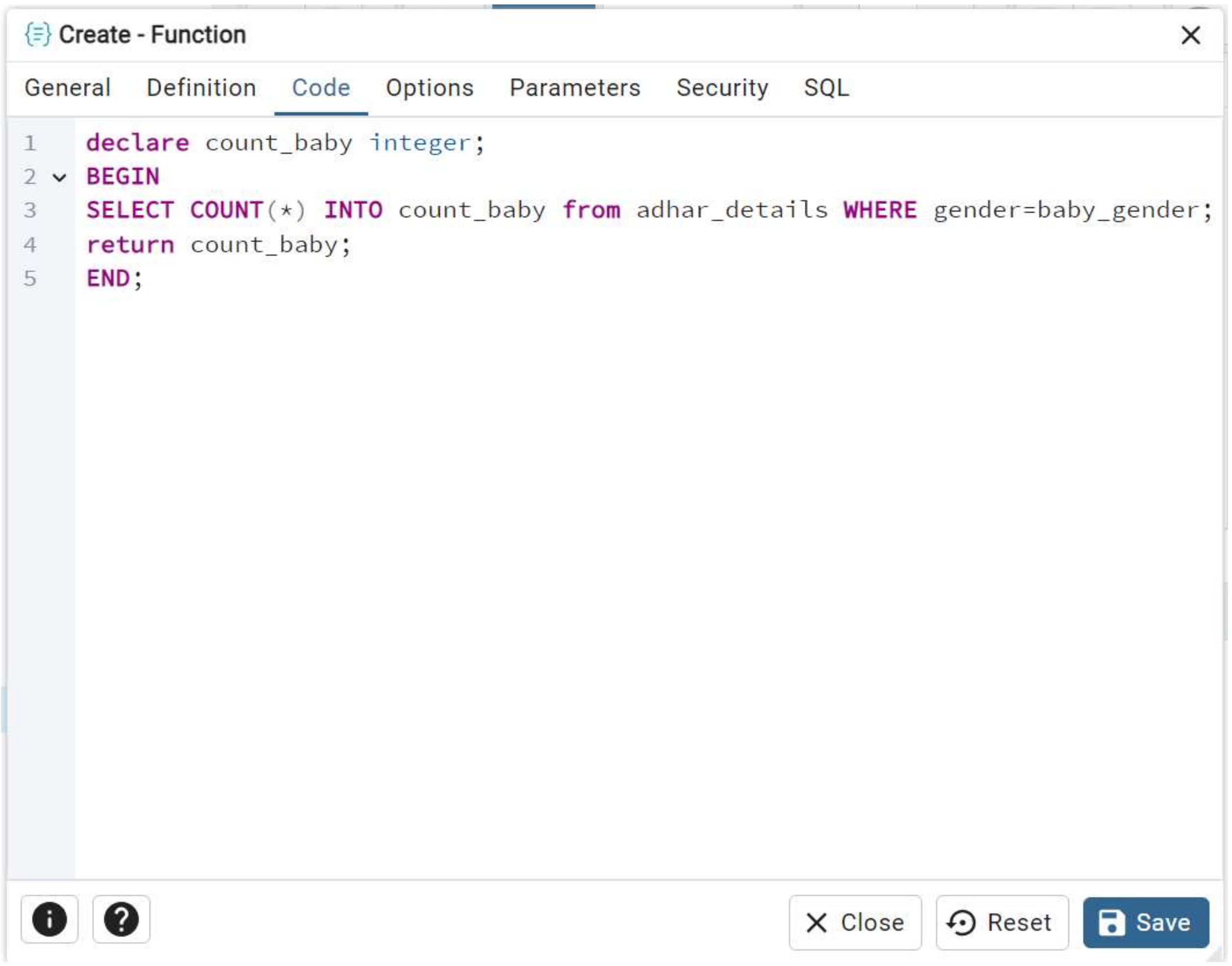
Custom return type? No

Return type: integer

Language: plpgsql

	Data type	Mode	Argument name	Default
	character varyi...	IN	gender	

Code cannot be empty.



```
declare count_baby integer;
BEGIN
SELECT COUNT(*) INTO count_baby from adhar_details WHERE gender=baby_gender;
return count_baby;
END;
```

Command to call function

Select function_name();

Example to call function from java application

Try(Connection con = DriverManager.getConnection(url)){
CallableStatement cstm = con.PrepareCall("select count_baby_gender(?)");
Cstm.setString(1,"male");
ResultSet rs = cstm.executeQuery();
If(rs.next()){
Int count=rs.nextInt(1);
Sout(count);
}else{
Sout("no record");
}
}catch(Exception e){
e.printstacktrace();}

Connection Pool

- Collection of connection objects is known as connection pool.
- In order to reduce the load which is going to put on driver manager and also to improve performance of the application we make use of connection pool.
- Connection pool can be created using array or collection framework
  Eg.

```
Public class ConnectionPool{
Private static String
url=”jdbc:postgresql://localhost:5432/baby?user=postgres&password=root”;
Private static String driver=”org.postgresql.Driver”;
Private static final int POOL_SIZE=4;
Private static List<Connection> connectionPool= new ArrayList<>();

Static{
        For(int i=0;i<POOL_SIZE;i++){
                ConnectionPool.add(createConnection());
        }
}

Private static Connection createConnection(){
        Connection connection =null;
        Try{
                Class.forName(driver);
                Connection = DriverManager.getConnection(url);
        }catch(Exception e){
e.printStackTrace();
        }
        Return connection;
}

Public static Connection giveConnection(){
If(!connectionPool.isEmpty()){
        Return connectionPool.remove(0);
        }else{
        Return createConnection();
        }
}
Public static void submitConnection(Connection connecction){
If(connectionPool.size()<POOL_SIZE){
connectionPool.add(connection);
        }else{
        Try{
                Connection.close();
                }catch(Exception e){
                e.printStackTrace();
                }
        }
}
}
```

Example using pool:

```
Public class Test{
Psvm(String[] args){
        Connection connection = ConnectionPool.giveConnection();
        Try{
                Statement stm = connection.creatStatement();
                String sql="INSERT INTO personal_details
        VALUES('Rahul',2,'male',420)";
                        Int update=stm.executeUpdate(sql);
                System.outPrintln(update);
                ConnectionPool.submitConnection(connection);
                System.out.println("data is inserted and connection submitted");

                }catch(Exception e){
                e.printStackTrace();
                }
        }
}
```

Transaction

1. Transaction is nothing but a group of tasks which succeed together or fail together is known as transaction

2. Transaction management is a process of managing the transaction

Eg.

```
Connection connection =ConnectionPool.giveConnection();
Try{
Connection.setAutoCommit(false);

Statement stm1=connection.createStatement();
String sql1=”INSERT INTO product VALUES(101,’pizza’,250,’cheeze burst’);
Stm1.execute(sql1);

Statement stm2=connection.createStatement();
String sql2=”INSERT INTO product VALUES(102,’burger’,300,’extra slice cheeze’);
Stm2.execute(sql2);

Statement stm3=connection.createStatement();
String sql3=”INSERT INTO product VALUES(111,550,25,25);
Stm3.execute(sql3);

If(paymentGateway.pay()){
Connection.commit();
System.out.println(“payment is successful and order is placed”);
}else{
Connection.rollback();
System.out.println(“payment is failed and order is canceled”);
}
ConnectionPool.submitConnection(connection);

}catch(Exception e){

e.printStackTrace();

}
```

Transaction management with savepoint

- ❖ Savepoint
  - ➢ savepoint is an interface present inside java.sql package we can get savepoint object by calling set savepoint method present inside connection object
  - ➢ To rollback till the savepoint we use rollback(savepoint)

Eg.

```
Connection.setAutoCommit(false);


Statement stm1=connection.createStatement();

String sql1=”INSERT INTO product VALUES(103,’pizza’,250,’cheeze burst’);

Stm1.execute(sql1);


Statement stm2=connection.createStatement();

String sql2=”INSERT INTO product VALUES(104,’burger’,300,’extra slice cheeze’);

Stm2.execute(sql2);


//savepoint

Savepoint savepoint = connection.setsavepoint();
```

```
Statement stm3=connection.createStatement();

String sql3=”INSERT INTO product VALUES(222,550,25,25);

Stm3.execute(sql3);


If(paymentGateway.pay()){

Connection.commit();

System.out.println(“payment is successful and order is placed”);

}else{

Connection.rollback(savepoint);

Connection.commit();

System.out.println(“payment is failed and order is canceled”);

}

ConnectionPool.submitConnection(connection);

}catch(Exception e){

e.printStackTrace();

}
```