

DBMS LAB PROJECT

ONLINE BANKING SYSTEM



# **SUBMITTED BY :**

### K.SAI KRISHNA-319126511031

### N.SHANMUKH TEJA-319126511040

### P.SAI VIVEK-319126511046

### S.PRASANTH-319126511049

### S.REHAMAN-319126511053

ACKNOWLEDGEMENTS

We express the sincere gratitude to our principal for his administration towards our academic growth. We express sincere gratitude to our Coordinator and HOD-IT for her leadership and constant motivation, provided in successful completion of our academic semester.

We record it as our privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality. We express our sincere thanks to our DBMS madam K.SWATHI LAKSHMI DURGA for her novel association of ideas, encouragement, appreciation and intellectual zeal which motivated us to venture this project successfully. We felt very happy for working under her guidance.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

INDEX

|  |  |  |
| --- | --- | --- |
| S.NO | TOPIC | PAGE NO |
| 1 | PROJECT NAME AND PURPOSE | 4 |
| 2 | PROJECT MODULE | 5 |
| 3 | FUTURE SCOPE OF PROJECT | 6 |
| 4 | JAVA SQL | 7 |
| 5 | ER DIAGRAM | 11 |
| 6 | CONVERTING ER DIAGRAM TO RELATIONS | 11 |
| 7 | INSERTING VALUES INTO TABLES | 12 |
| 8 | UPDATE AND DELETE COMMAND | 15 |
| 9 | IMPLEMENTATION OFTRIGGERS | 16 |
| 10 | IMPLEMENTATION OF CURSORS | 18 |
| 11 | JDBC DRIVERS | 20 |
| 12 | JDBC CONNECTIVITY STEPS | 22 |
| 13 | JDBC CONNECTIVITY FOR RELATIONS | 24 |
| 14 | NORMALIZATION | 29 |

Project title:

Online Banking System

Purpose:

The typical way to retain the data of the customer in a bank was to enter and Register the details. Any time a consumer has to make any transactions, he must go to the bank and take the required steps, which might not be possible all the time. It can also be a hard-hitting job for users and bankers. The project offers a real-life understanding of the Online Banking Environment and the tasks undertaken by the different positions in the supply chain. Here, we are offering automation for the banking system over the Internet. The Online Banking Framework project gathers tasks undertaken by various positions in real-life banking and offers improved strategies to keep the details needed up-to-date, resulting in productivity. The project offers a real-life understanding of the Online Banking Environment and the tasks undertaken by the different positions in the supply chain.

Scope of the Project:

This project explores the entry threshold for the provision of a new transaction processing channel using a real options strategy, where the entry threshold is determined using an Internet banking framework built for the use of regular users (individuals), Industrialists, Manufacturers, Educational Institutions (Financial Sections), Organizations and Academicians under transaction rate uncertainty.

• The customer must have a correct user Id and password to log in to your device.

• If the incorrect password is given three times in succession, the account will be locked and the user will not be able to access it. If an incorrect password is used, the user will be alerted that his account will be closed.

• When a legitimate user logs in, the number of accounts he has with the bank will be revealed.

• When choosing the desired account, it is taken to a page that displays the existing balance of that particular account number.

• Users may request details of the last 'n' number of transactions that they have carried out.

• The study can also be taken from this.

• Users may move funds to another account in the same bank. The use giyen a transaction password that is separate from the login password.

• The user can move funds from his account to any other bank account. If the transaction is successful, a note should show to the customer, if the transaction is unsuccessful, a valid message should be given to the customer as to why the transaction failed.

• Users can order the checkbook/address change/stop payment of the check.

• Users will display both their monthly and annual statements. He may even take the printout of the same item.

• Generate reports for each segment

• The administrator should take thebackup of the database periodically for any instance that occurs.

• Both customers are authenticated to access the services

• The FAQ portion is also included for the support of end-users. Definitions, Acronyms, and Abbreviations

Administrator: The super user who can add new customers to the banking system and assigns the required username, password, type of account, and other information. If any customer cancels their bank account, their account can be removed and transactions can be stopped immediately.

Customers: After logging in customer can request for balance inquiry in his account, Funds Transfer to another account in the same bank, Requests for checkbook/change of address/stop payment(Viewing Monthly and annual statements)

Project module:

The following modules are included in the Online Banking System :

1) BRANCH: This module helps us to know name of the branch, city in which branch is located and assets of the branch.

2) EMPLOYEE: This module tell us current working employee name, id, start date and phone number.

3) CUSTOMER: This module helps us to know about customer details like name, id ,street and city.

4)ACCOUNT: This module gives us the details like account id, account holder name, rate and type of account i.e.; either savings or current.

5) PAYMENT: This module gives us the payment details like payment no, date of payment and amount.

Future Scope of the Project:

The new framework is being developed as a software application. In the future, we would like to improve it for handheld devices such as mobile phones, wap, or GPRS links.

JAVA.SQL. \* :

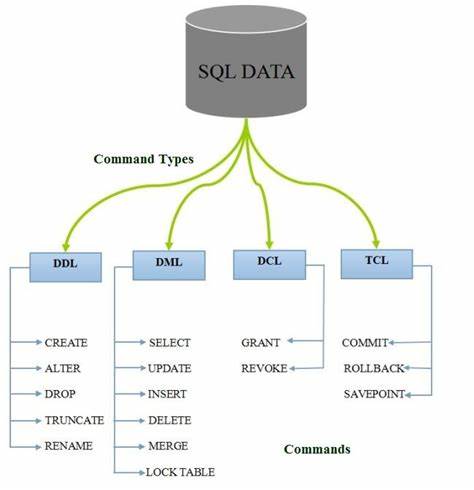
Structure Query Language(**SQL**) is a database query language used for storing and managing data in Relational **DBMS**. **SQL** was the first commercial language introduced for E.F Codd's Relational model of database. **SQL** is used to perform all types of data operations in **RDBMS**.

Also Know, what are the subsets of SQL? **Subsets of SQL** There are three main **subsets** of the **SQL** language: . Data Control Language (DCL) . Data Definition Language (DDL) . Data Manipulation Language (DML) Each set of the **SQL** language has a special purpose: .

Keeping this in view, what are the commands of SQL?

**There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.**

* Data Definition Language (DDL) DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* Data Manipulation Language.
* Data Control Language.
* Transaction Control Language.
* Data Query Language.



### **1. Data Definition Language (DDL)**

* DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

**a. CREATE** It is used to create a new table in the database.

**Syntax:**

1. CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]);

**b. DROP:** It is used to delete both the structure and record stored in the table.

**Syntax**

1. DROP TABLE table\_name;

**c. ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**Syntax:**

To add a new column in the table

1. ALTER TABLE table\_name ADD column\_name COLUMN-

**d. TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

**Syntax:**

1. TRUNCATE TABLE table\_name;

### **2. Data Manipulation Language**

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

**a. INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:**

1. INSERT INTO TABLE\_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);

**b. UPDATE:** This command is used to update or modify the value of a column in the table.

**Syntax:**

1. UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

**c. DELETE:** It is used to remove one or more row from a table.

**Syntax:**

1. DELETE FROM table\_name [WHERE condition];

### **3. Data Control Language**

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.

**Example**

1. GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

**b. Revoke:** It is used to take back permissions from the user.

**Example**

1. REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

### **4. Transaction Control Language**

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.

**Syntax:**

COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:**

ROLLBACK;

**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

SAVEPOINT SAVEPOINT\_NAME;

### **5. Data Query Language**

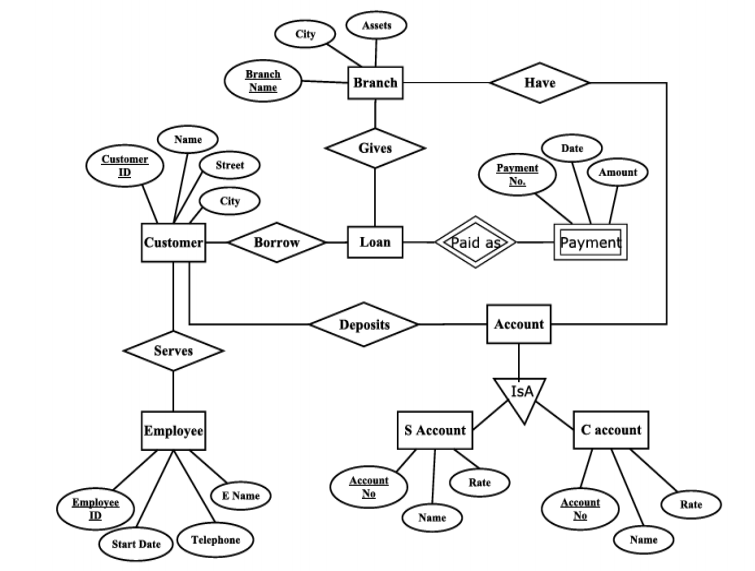
DQL is used to fetch the data from the database.It uses only one command:

* SELECT

**a. SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

**Syntax:**

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

ER DIAGRAM:

CONVERTING ER DIAGRAM TO RELATIONS:

create table branch(branch\_name varchar(20) **primary key**, city varchar(20),assets varchar(20));

create table employee(employee\_id int **primary key**,start\_date varchar(20),telephone int,e\_name varchar(20));

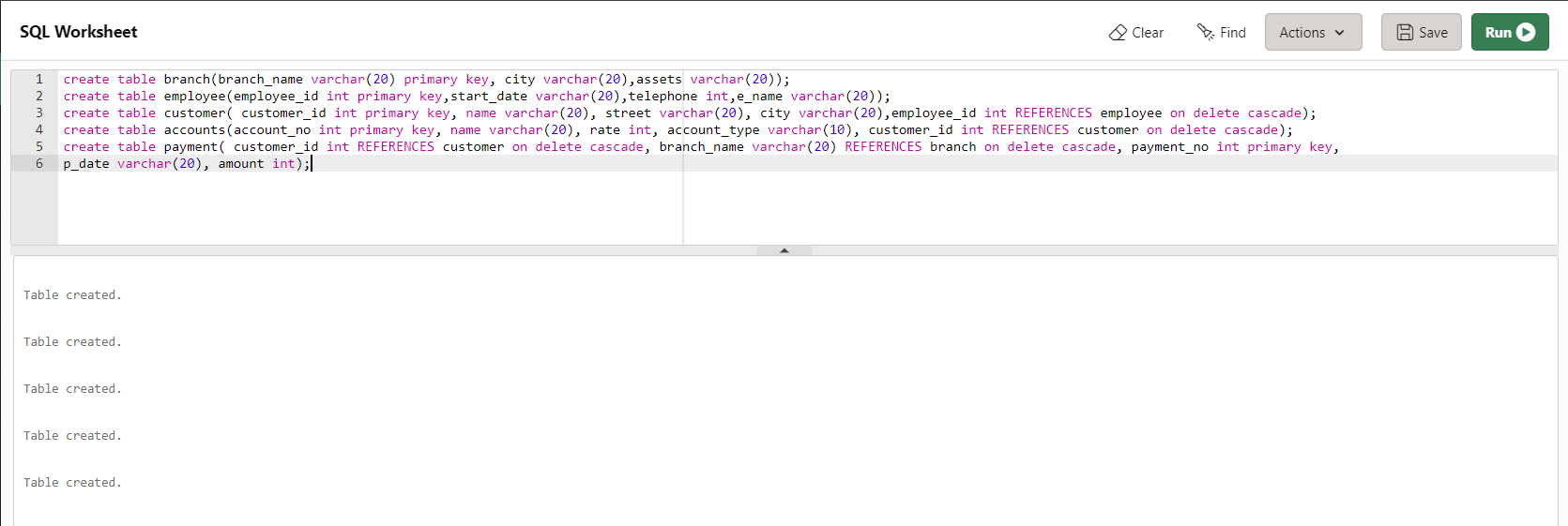
create table customer( customer\_id int **primary key**, name varchar(20), street varchar(20), city varchar(20),employee\_id int **REFERENCES** employee on **delete cascade**);

**Now we combine both loan and payment together and create a single table as payment and we use two foreign keys i.e customer\_id and branch\_name.**

create table payment(customer\_id int **REFERENCES** customer on **delete cascade**, branch\_name varchar(20) **REFERENCES** branch on **delete cascade**, payment\_no int **primary key**, p\_date varchar(20), amount int);

**We create a single table for both the type of accounts by adding a single attribute account\_type for differentiating whether it is a current or savings.**

create table accounts(account\_no int **primary key**, name varchar(20), rate int, account\_type varchar(10), customer\_id int REFERENCES customer on delete cascade);

CREATING TABLES IN SQL: 

INSERTING VALUES:

->INTO BRANCH

insert into branch values('tagarapuvalasa','tagarapuvalasa','5-computers,2-ac');

insert into branch values('kovvur','kovvur','5-computers, 1-ac');

insert into branch values('kphb','kukatpally','7-computers, 3-ac');



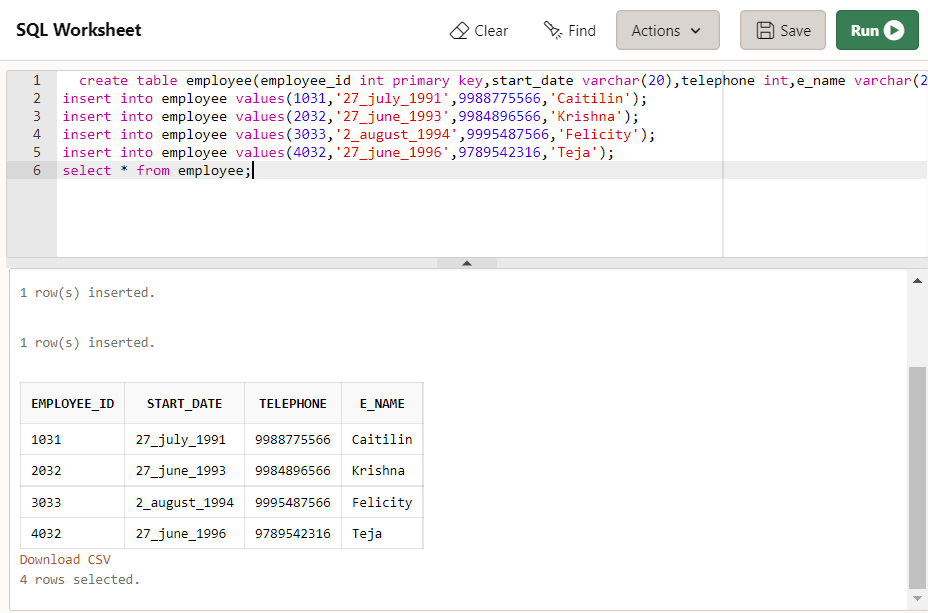
-> INTO EMPLOYEE

insert into employee values(1031,'27\_july\_1991',9988775566,'Caitilin');

insert into employee values(2032,'27\_june\_1993',9984896566,'Krishna');

insert into employee values(3033,'2\_august\_1994',9995487566,'Felicity');

insert into employee values(4032,'27\_june\_1996',9789542316,'Teja');

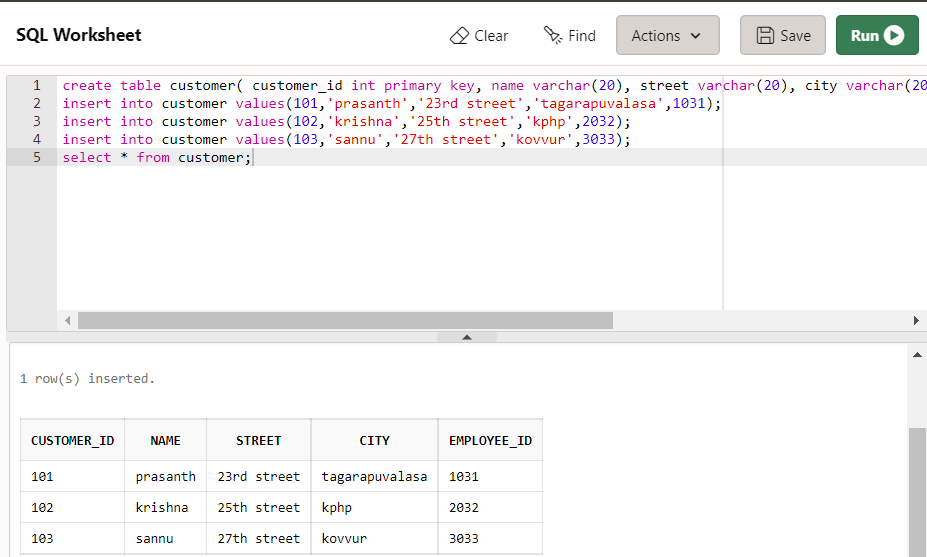


->INTO CUSTOMER

insert into customer values(101,'prasanth','23rd street','tagarapuvalasa',1031);

insert into customer values(102,'krishna','25th street','kphp',2032);

insert into customer values(103,'sannu','27th street','kovvur',3033);

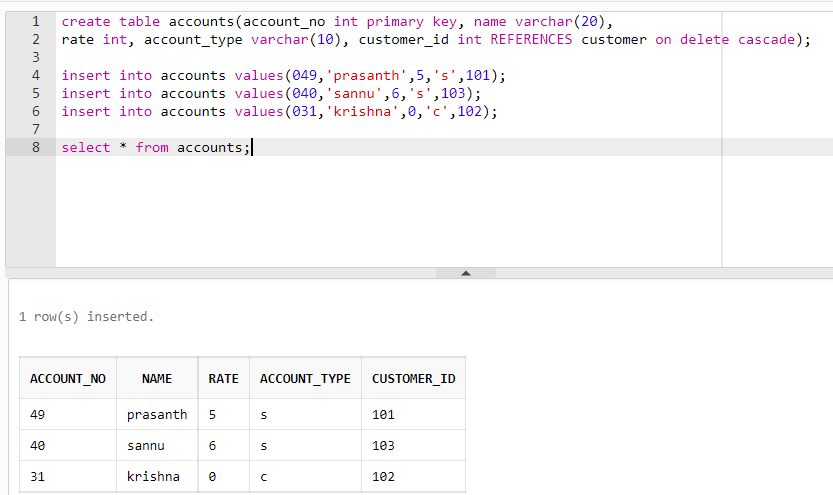


->INTO ACCOUNT

insert into accounts values(049,'prasanth',5,'s',101);

insert into accounts values(040,'sannu',6,'s',103);

insert into accounts values(031,'krishna',0,'c',102);



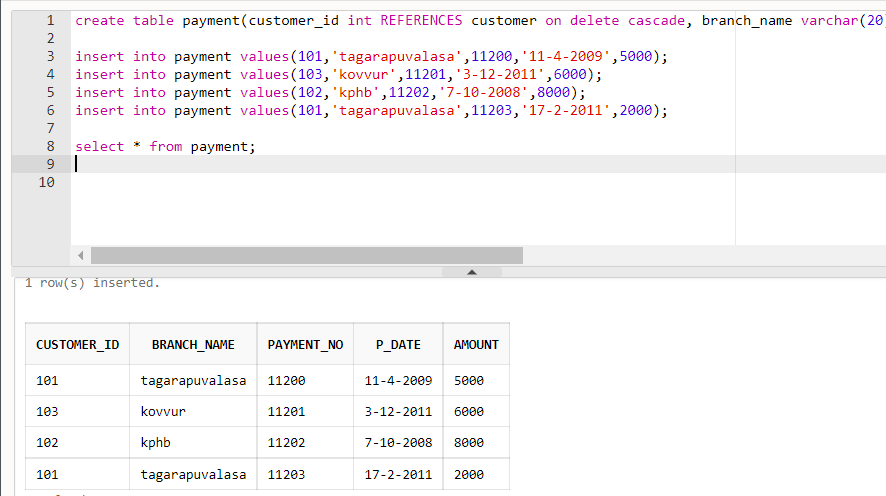
->INTO PAYMENT

insert into payment values(101,'tagarapuvalasa',11200,'11-4-2009',5000);

insert into payment values(103,'kovvur',11201,'3-12-2011',6000);

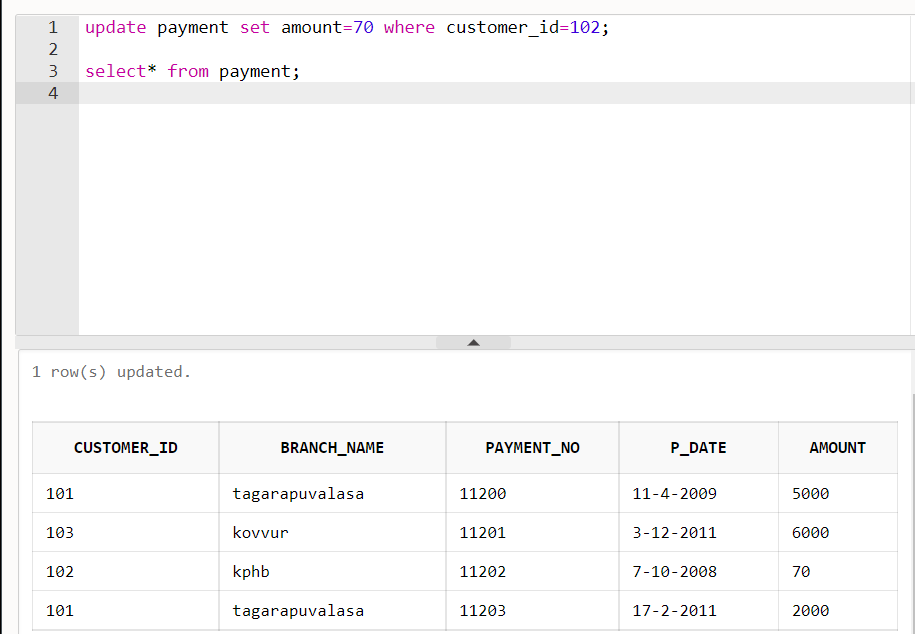
insert into payment values(102,'kphb',11202,'7-10-2008',8000);

insert into payment values(101,'tagarapuvalasa',11203,'17-2-2011',2000);



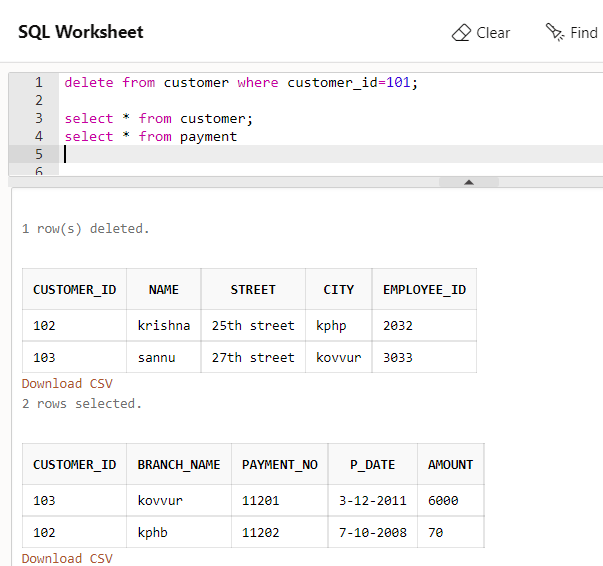
UPDATE COMMAND:

update payment set amount=70 where customer\_id=102;



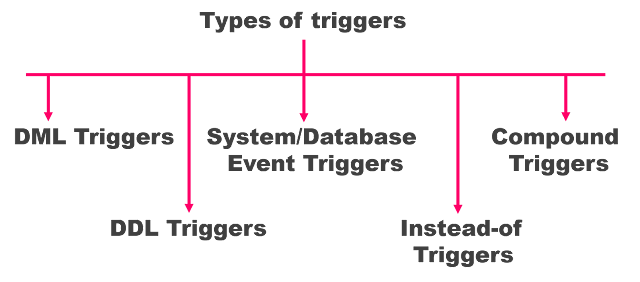
DELETE COMMAND:

delete from customer where customer\_id=101;



IMPLEMENTING TRIGGERS:

Triggers are named PL/SQL blocks which are stored in the database. We can also say that they are specialized stored programs which execute implicitly when a triggering event occurs. This means we cannot call and execute them directly instead they only get triggered by events in the database.



**Syntax for creating a trigger:**

|  |
| --- |
| CREATE [OR REPLACE ] TRIGGER trigger\_n  {BEFORE | AFTER | INSTEAD OF }  {INSERT [OR] | UPDATE [OR] | DELETE}  [OF column\_n]  ON table\_n  [REFERENCING OLD AS o NEW AS n]  [FOR EACH ROW]  WHEN (condition)  DECLARE      << Declaration statement >>  BEGIN     << Block of executable code>>  EXCEPTION     << Exception handling if any >>    END; |

**USE OF TRIGGERS:** A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database.Because a trigger resides in the database and anyone who has the required privilege can use it, a trigger lets you write a set of SQL statements that multiple applications can use.

**TRIGGER FOR EMPLOYEE TABLE:**

CREATE OR REPLACE TRIGGER emp\_trigger

AFTER update

ON employee

for each row

BEGIN

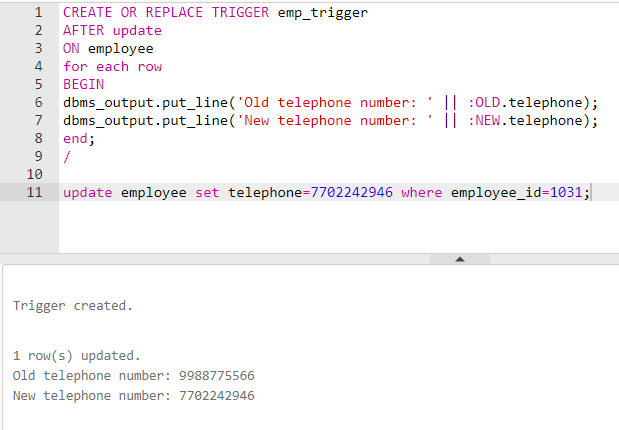
dbms\_output.put\_line('Old telephone number: ' || :OLD.telephone);

dbms\_output.put\_line('New telephone number: ' || :NEW.telephone);

end;

/

**EXECUTION:**



IMPLEMENTING CURSORS:

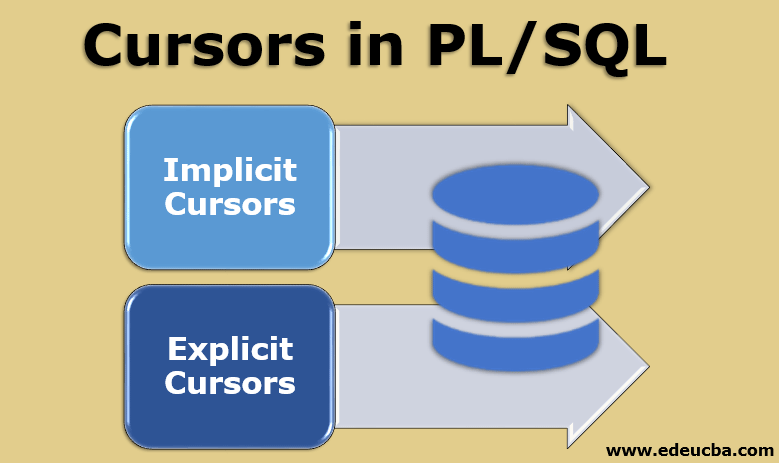
To execute SQL statements, a work area is used by the Oracle engine for its internal processing and storing the information. This work area is private to SQL’s operations. The ‘Cursor’ is the PL/SQL construct that allows the user to name the work area and access the stored information in it.

## **Why Use a Cursor in SQL Server?**

## Although using an INSERT, UPDATE or DELETE statement to modify all of the applicable data in one transaction is generally the best way to work with data in SQL Server, a cursor may be needed for:

* Iterating over data one row at a time
* Completing a process in a serial manner such as SQL Server database backups
* Updating data across numerous tables for a specific account
* Correcting data with a predefined set of data as the input to the cursor

**TYPES OF CURSORS:**



Cursors are classified depending on the circumstances in which they are opened:-

● Implicit Cursor: If the Oracle engine opened a cursor for its internal processing it is known as an Implicit Cursor. It is created “automatically” for the user by Oracle when a query is executed and is simpler to code.

● Explicit Cursor: A Cursor can also be opened for processing data through a PL/SQL block, on demand. Such a user-defined cursor is known as an Explicit Cursor.

**CURSOR FOR CUSTOMER DETAILS:**

DECLARE

c\_id customer.customer\_id%type;

c\_name customer.name%type;

c\_street customer.street%type;

c\_city customer.city%type;

CURSOR customer\_cursor is

SELECT customer\_id, name, street, city FROM customer;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor into c\_id, c\_name, c\_street,c\_city;

EXIT WHEN customer\_cursor%notfound;

dbms\_output.put\_line(c\_id||' '||c\_name||' '||c\_street||' '||c\_city);

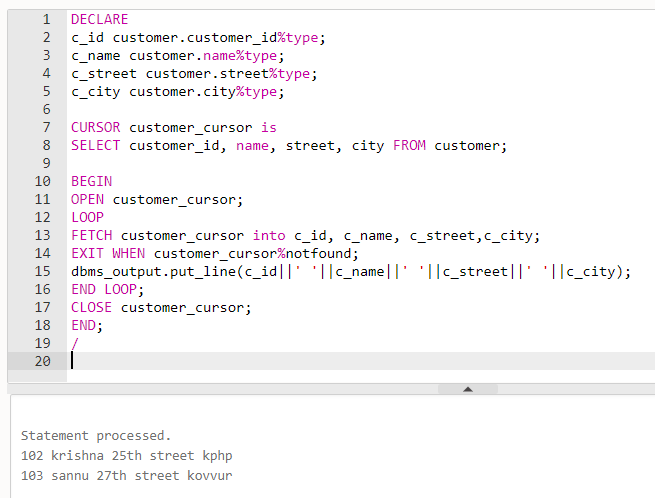
END LOOP;

CLOSE customer\_cursor;

END;

/

**EXECUTION:**

****

## **What is JDBC Driver?**

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

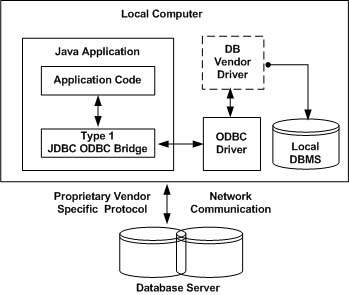
The *Java.sql* package that ships with JDK, contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers. Third party vendors implements the *java.sql.Driver* interface in their database driver.

## **JDBC Drivers Types**

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below −

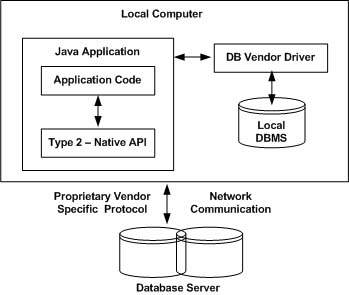
## **Type 1 − JDBC-ODBC Bridge Driver**

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.



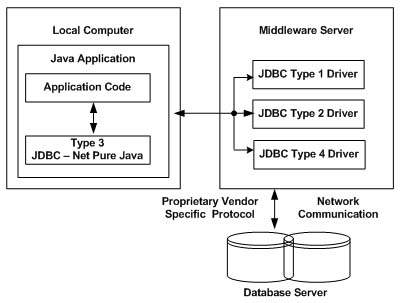
## **Type 2 − JDBC-Native API**

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.



## **Type 3 − JDBC-Net pure Java**

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

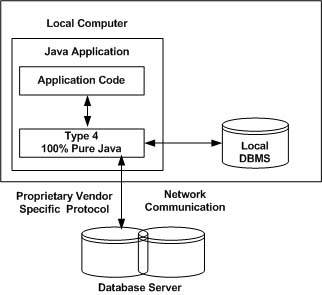


You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

## **Type 4 − 100% Pure Java**

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.



|  |
| --- |
| There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:   * Register the Driver class * Create connection * Create statement * Execute queries * Close connection |

**Java Database Connectivity Steps:**



**1) Register the driver class**

|  |
| --- |
| The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. |

### **Syntax of forName() method**

**public** **static** **void** forName(String className)**throws** ClassNotFoundException.

### **2) Create the connection object**

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with the database. |

### **Syntax of getConnection() method**

**public** **static** Connection getConnection(String url)**throws** SQLException

### **3) Create the Statement object**

|  |
| --- |
| The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. |

### **Syntax of createStatement() method**

**public** Statement createStatement()**throws** SQLException

### **4) Execute the query**

|  |
| --- |
| The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table. |

### **Syntax of executeQuery() method**

**public** ResultSet executeQuery(String sql)**throws** SQLException.

### **5) Close the connection object**

|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

### **Syntax of close() method**

### **public** **void** close()**throws** SQLException

JDBC CONNECTIVITY:

**BRANCH :**

**import** java.sql.\*;

**class** branch{

**public** **static** **void** main(String args[]){

**try**{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe","system","sys");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from branch");

**while**(rs.next())

System.out.println(rs.getString(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

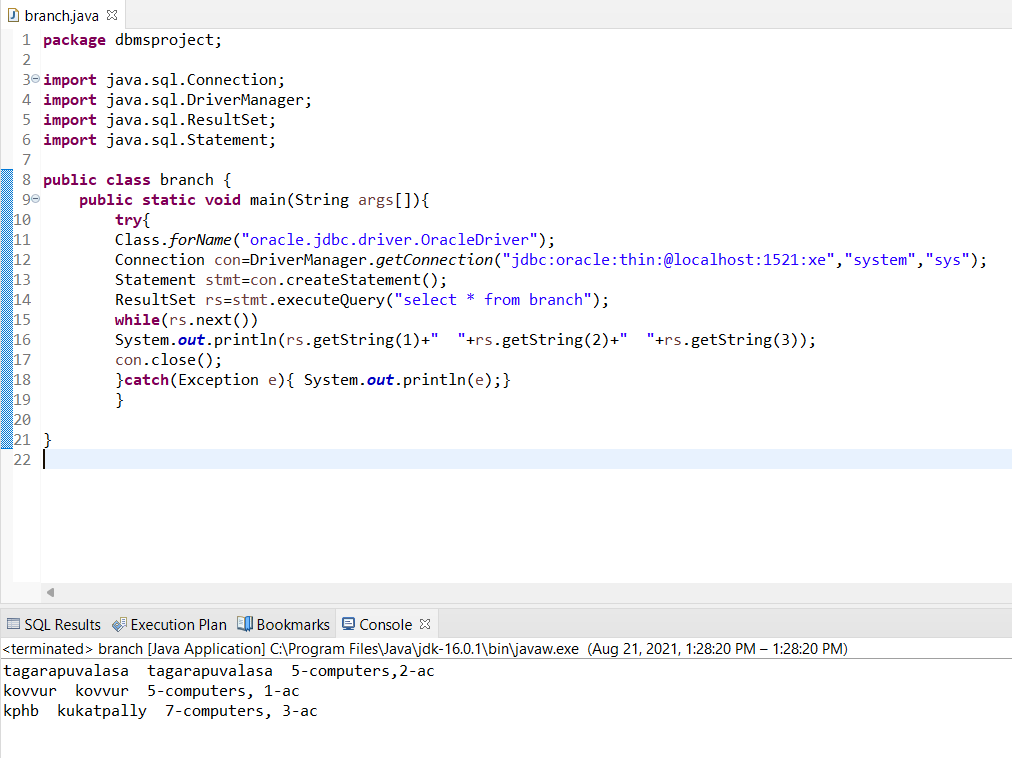
con.close();

}

**catch**(Exception e){ System.out.println(e);}

}  }

**EXECUTION:**



**EMPLOYEE:**

**import** java.sql.\*;

**class** employee{

**public** **static** **void** main(String args[]){

**try**{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from employee");

**while**(rs.next())

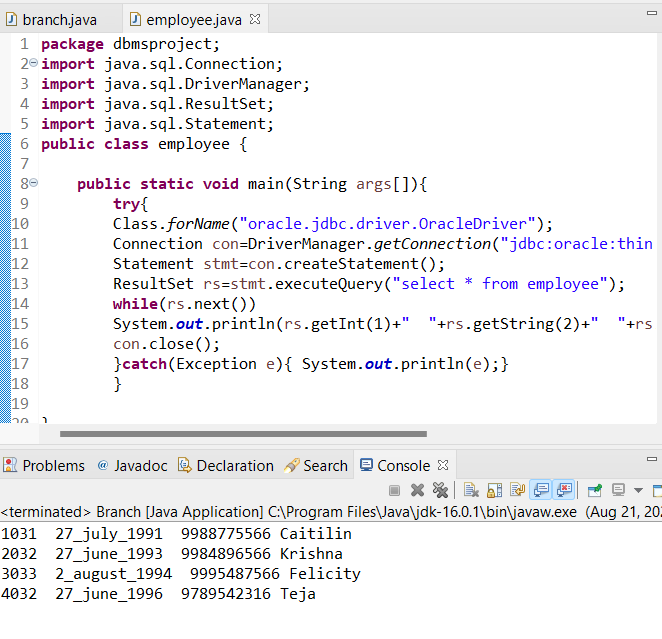
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3)+ "  "+rs.getSrting(4));

con.close();

}**catch**(Exception e){ System.out.println(e);}

}  }

**EXECUTION:**

****

**CUSTOMER:**

**import** java.sql.\*;

**public** **class** customer {

**public** **static** **void** main(String args[]){

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.*getConnection*(

"jdbc:oracle:thin:@localhost:1521:xe","system","sys");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from customer");

**while**(rs.next())

System.***out***.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)+" "+rs.getString(4)+" "+rs.getInt(5));

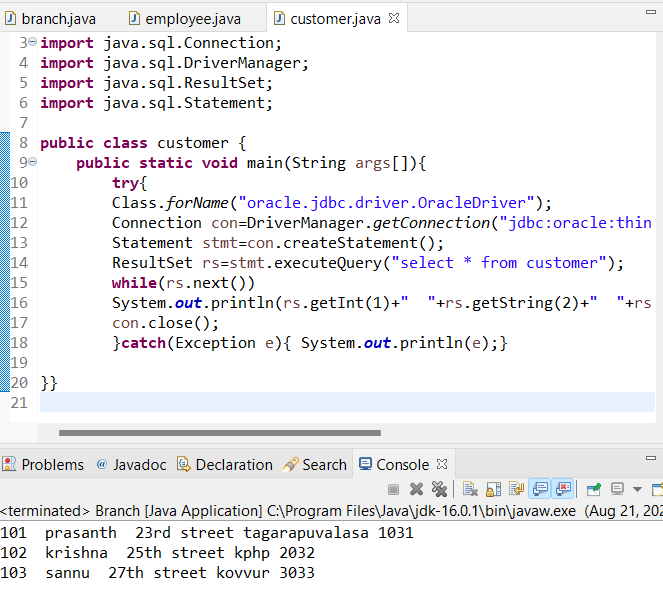
con.close();

}

**catch**(Exception e){ System.***out***.println(e);}

} }

**EXECUTION:**

****

**ACCOUNT:**

**import** java.sql.\*;

**public** **class** accounts {

**public** **static** **void** main(String args[]){

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.*getConnection*(

"jdbc:oracle:thin:@localhost:1521:xe","system","sys");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from accounts");

**while**(rs.next())

System.***out***.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3)+" "+rs.getString(4)+" "+rs.getInt(5));

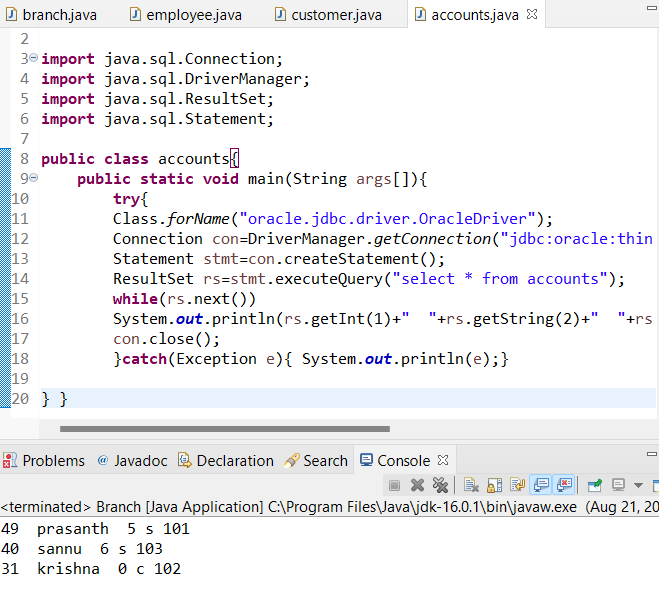
con.close();

}

**catch**(Exception e){ System.***out***.println(e);}

} }

**EXECUTION:**

****

**PAYMENT:**

**import** java.sql.\*;

**public** **class** payment {

**public** **static** **void** main(String args[]){

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.*getConnection*(

"jdbc:oracle:thin:@localhost:1521:xe","system","sys");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from payment");

**while**(rs.next())

System.***out***.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)+" "+rs.getString(4)+" "+rs.getInt(5));

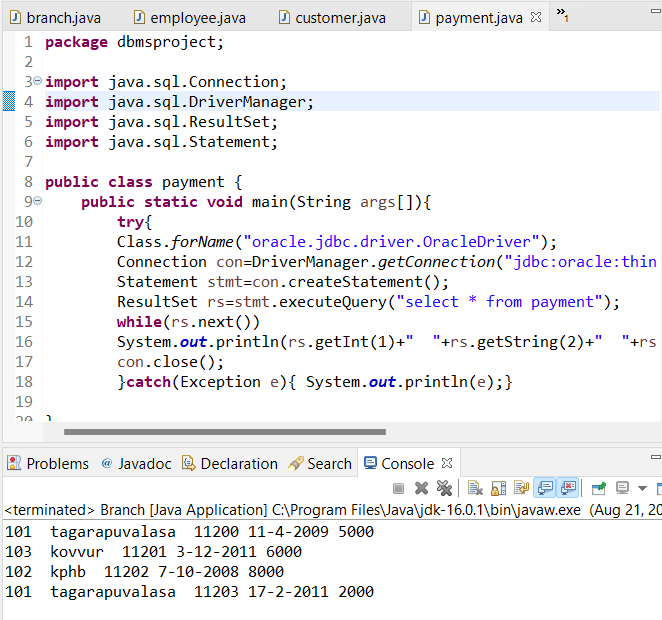
con.close();

}

**catch**(Exception e){ System.***out***.println(e);}

} }

**EXECUTION:**

****

NORMALIZATION:

Database normalization is a database schema design technique, by which an existing schema is modified to minimize redundancy and dependency of data.

Normalization split a large table into smaller tables and define relationships between them to increases the clarity in organizing data.

**FIRST NORMAL FORM:-**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPLOYEE\_ID** | **START\_DATE** | **TELEPHONE** | **E\_NAME** |
| 1031 | 27\_july\_1991 | 9988775566 | Caitilin |
| 2032 | 27\_june\_1993 | 9984896566 | Krishna |
| 3033 | 2\_august\_1994 | 9995487566 | Felicity |
| 4032 | 27\_june\_1996 | 9789542316 | Teja |

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is singled valued attribute.

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **CITY** | **ASSETS** |
| tagarapuvalasa | tagrapuvalasa | 5-computers,2-ac |
| kovvur | kovvur | 5-computers, 1-ac |
| kphb | kukatpally | 7-computers, 3-ac |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **STREET** | **CITY** | **EMPLOYEE\_ID** |
| 102 | krishna | 25th street | kphp | 2032 |
| 103 | sannu | 27th street | kovvur | 3033 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCOUNT\_NO** | **NAME** | **RATE** | **ACCOUNT\_TYPE** | **CUSTOMER\_ID** |
| 40 | sannu | 6 | s | 103 |
| 31 | krishna | 0 | c | 102 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **BRANCH\_NAME** | **PAYMENT\_NO** | **P\_DATE** | **AMOUNT** |
| 103 | kovvur | 11201 | 3-12-2011 | 6000 |
| 102 | kphb | 11202 | 7-10-2008 | 70 |

All the above tables are in 1NF as no table consists of mutivalues.

**SECOND NORMAL FORM :-**

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency – If the proper subset of candidate key determines a non-prime attribute, it is called partial dependency.

**1.Branch table:**

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **CITY** | **ASSETS** |
| tagarapuvalasa | tagrapuvalasa | 5-computers,2-ac |
| kovvur | kovvur | 5-computers, 1-ac |
| kphb | kukatpally | 7-computers, 3-ac |

FD set= { BRANCH\_NAME -> CITY , BRANCH\_NAME -> ASSETS }

Candidate key= BRANCH\_NAME

Here there is no partial dependency.

So, this table is in 2NF

**2.Employee table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPLOYEE\_ID** | **START\_DATE** | **TELEPHONE** | **E\_NAME** |
| 1031 | 27\_july\_1991 | 9988775566 | Caitilin |
| 2032 | 27\_june\_1993 | 9984896566 | Krishna |
| 3033 | 2\_august\_1994 | 9995487566 | Felicity |
| 4032 | 27\_june\_1996 | 9789542316 | Teja |

FD set= { EMPLOYEE\_ID -> START\_DATE, EMPLOYEE\_ID -> TELEPHONE, EMPLOYEE\_ID -> E-NAME };

Candidate key= EMPLOYEE\_ID

Here there is no partial dependency. So, this table is in 2NF

**3.Customer table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **STREET** | **CITY** | **EMPLOYEE\_ID** |
| 102 | krishna | 25th street | kphp | 2032 |
| 103 | sannu | 27th street | kovvur | 3033 |

FD set= { CUSTOMER\_ID -> NAME ,CUSTOMER\_ID -> STREET ,CUSTOMER\_ID -> CITY ,CUSTOMER\_ID -> EMPLOYEE\_ID }

Candidate key = CUSTOMER\_ID

Here there is no partial dependency

So, this table is in 2NF

**4.Account table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCOUNT\_NO** | **NAME** | **RATE** | **ACCOUNT\_TYPE** | **CUSTOMER\_ID** |
| 40 | sannu | 6 | s | 103 |
| 31 | krishna | 0 | c | 102 |

FD set= { ACCOUNT\_NO -> NAME ,ACCOUNT\_NO -> RATE ,ACCOUNT\_NO -> ACCOUNT\_TYPE ,ACCOUNT\_NO -> CUSTOMER\_ID }

Candidate key = ACCOUNT\_NO

Here there is no partial dependency

So, this table is in 2NF

**5.Payment table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **BRANCH\_NAME** | **PAYMENT\_NO** | **P\_DATE** | **AMOUNT** |
| 103 | kovvur | 11201 | 3-12-2011 | 6000 |
| 102 | kphb | 11202 | 7-10-2008 | 70 |

FD set= { PAYMENT\_NO -> P\_DATE, PAYMENT\_NO -> AMOUNT, PAYMENT\_NO -> BRANCH\_NAME, PAYMENT\_NO -> CUSTOMER\_ID }

Candidate key = PAYMENT\_NO

Here there is no partial dependency

So, this table is in 2NF

**THIRD NORMAL FORM:-**

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y

1. X is a super key.

2. Y is a prime attribute (each element of Y is part of some candidate key).

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

**1.Branch table:**

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **CITY** | **ASSETS** |
| tagarapuvalasa | tagrapuvalasa | 5-computers,2-ac |
| kovvur | kovvur | 5-computers, 1-ac |
| kphb | kukatpally | 7-computers, 3-ac |

FD set= { BRANCH\_NAME -> CITY , BRANCH\_NAME -> ASSETS }

Candidate key= BRANCH\_NAME

Here there is no transitive dependency.

So, this table is in 3NF.

**2.Employee table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPLOYEE\_ID** | **START\_DATE** | **TELEPHONE** | **E\_NAME** |
| 1031 | 27\_july\_1991 | 9988775566 | Caitilin |
| 2032 | 27\_june\_1993 | 9984896566 | Krishna |
| 3033 | 2\_august\_1994 | 9995487566 | Felicity |
| 4032 | 27\_june\_1996 | 9789542316 | Teja |

FD set= { EMPLOYEE\_ID -> START\_DATE, EMPLOYEE\_ID -> TELEPHONE, EMPLOYEE\_ID -> E-NAME };

Candidate key= EMPLOYEE\_ID

Here there is no transitive dependency. So, this table is in 3NF.

**3.Customer table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **STREET** | **CITY** | **EMPLOYEE\_ID** |
| 102 | krishna | 25th street | kphp | 2032 |
| 103 | sannu | 27th street | kovvur | 3033 |

FD set= { CUSTOMER\_ID -> NAME ,CUSTOMER\_ID -> STREET ,CUSTOMER\_ID -> CITY ,CUSTOMER\_ID -> EMPLOYEE\_ID }

Candidate key = CUSTOMER\_ID

Here there is no transitive dependency

So, this table is in 3NF.

**4.Account table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCOUNT\_NO** | **NAME** | **RATE** | **ACCOUNT\_TYPE** | **CUSTOMER\_ID** |
| 40 | sannu | 6 | s | 103 |
| 31 | krishna | 0 | c | 102 |

FD set= { ACCOUNT\_NO -> NAME ,ACCOUNT\_NO -> RATE ,ACCOUNT\_NO -> ACCOUNT\_TYPE ,ACCOUNT\_NO -> CUSTOMER\_ID }

Candidate key = ACCOUNT\_NO

Here there is no transitive dependency

So, this table is in 3NF.

**5.Payment table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CUSTOMER\_ID** | **BRANCH\_NAME** | **PAYMENT\_NO** | **P\_DATE** | **AMOUNT** |
| 103 | kovvur | 11201 | 3-12-2011 | 6000 |
| 102 | kphb | 11202 | 7-10-2008 | 70 |

FD set= { PAYMENT\_NO -> P\_DATE, PAYMENT\_NO -> AMOUNT, PAYMENT\_NO -> BRANCH\_NAME, PAYMENT\_NO -> CUSTOMER\_ID }

Candidate key = PAYMENT\_NO

Here there is no transitive dependency

So, this table is in 3NF.