RDBMS:

Java is mostly used to develop business applications.

Business Organization:

1. Small scale business organizations (grocery shop, petrol pump)
2. Large scale business organizations (HDFC bank, Indian railway, SBI, PEPSI, MASTERCARD) enterprises.

These BO provides their services to the client/ customer. And to computerized those services whatever application we develop is known as business application.

Common general things in any business organizations:

1.Store and maintain business data in secure and easily retrieval manner (inside the RDBMS)

2.processing that data according to the business rule.

3.presenting the data in user-understandable format.

DATA and INFORMATION:

------------------------------------

Data: It is a collection of raw and isolated facts.

Information: when we process the data, then we get meaningful results, this is called information.

Datastore:

-------------

It is a store where we can store or keep the data.

1.normal books and papers.

2.Flat files in computer system (notepad, excel sheet, word files)

Disadvantages of flat files:

-----------------------------------

* Data maintenance
* Data redundancy
* Data integrity
* Security
* Data retrieval

To overcome these problems we need to store the data inside the DBMS software RDBMS)

Database:

It is an organized collection of interrelated data or structured collection of data.

DBMS:

It is a type of software there we can manage multiple datasets.

RDBMS:

In this model, the data is stored in 2 dimensional tables.

--We have multiple RDBMS software are there:

MySql (oracle): mysql workbench

Oracle

Postgres

Ms-access

Sql-server

Db2 etc.

RDBMS is an extension of DBMS software.

NOTE: every RDBMS is a DBMS but reverse is not true.

--all the RDBMS software have a DB-engine which is the heart of the RDBMS, and it is responsible to execute sql query. (that query could be supplied by either command line client or from mysql workbench)

--In order to work with RDBMS we need to use SQL (structured query language), it is an interface by using which we can work with any kind of RDBMS software.

--sql is a case insensitive language.

--sql language is a collection of predefined commands

These commands are categorized into following categories:

1. DDL (Data Definition Language): create, alter, drop, truncate, rename
2. DML (Data Manipulation Language): insert, update, delete
3. DRL (Data Retrieval Language): select
4. TCL (Transaction Control Language): commit, rollback, savepoint
5. DCL (Decision Control Language): grant, revoke

Java Developer: select, insert, update, delete (CRUD)

Database Administrator: DDL, DML

Practical:

>show databases;

>create database web19sb101db;

After creating the data base inside the mysql rdbms software we need to move inside that DB in order to work with that DB.

>use web19sb101db;

>show tables; --it shows the tables inside the web19sb101db database

DDL Commands:

----------------------

CREATE:

------------

>create table student (roll int, name varchar (12), marks int);

> create table student

(

roll int,

name varchar (12),

marks int

);

>desc student:

Description of student table.

>select \* from student;

Datatypes in mysql:

---------------------------

1. Numeric types
2. String types
3. Date and time types

1.Numeric Type:

------------------------

Tiny int: 1byte

Small int: 2byte

Medium int: 3byte

Int: 4byte

Big int: 8byte

Floating point:

----------------------

Float (6,2): the column can store 6 digits with 2 decimal places.

2.String Type:

-------------------

1.char: fixed length of string range between 0 to 255 characters.

2.varchar: variable length of string between 1 to 65500, here we must define the length.

Eg:

Char (4)

Varchar (4)

Value char (4) storage\_required

‘a’ ----------> 4 bytes

‘ab’ ----------> 4 bytes

‘abcdef’ ----------> error data is too long

Value varchar (4) storage\_required

‘a’ ----------> 1 bytes

‘ab’ ----------> 2 bytes

‘abcdef’ ----------> error data is too long

NOTE: In the term of efficiency, if we are storing with variable length then we should use varchar, and if the length is always then we should use char, here char is slightly faster than varchar.

3.Date and Time:

-----------------------

date yyyy-mm-dd

datetime yyyy-mm-dd hh:mm:ss

ALTER:

---------

It is used to change the structure of the existing table

This command has 4 sub commands

1. Add
2. Modify
3. Drop
4. Change

add: It is used to add the new columns in the existing table.

Eg: >alter table student add address varchar (15);

Modify: It is used to change the column data type of column size.

Eg: >alter table student modify address varchar (20);

Drop: to drop a single or multiple columns

Eg: >alter table student drop address;

Change: to rename a column

Eg: >alter table student change column\_name rename varchar (20);

DROP:

---------

-To drop entire table

>drop table student;

NOTE: It cannot be rolled back. (DDL command cannot be rolled back)

TRUNCATE:

---------------

This command is used to truncate all the records permanently from the table. It will not drop the table. The table structure is present.

--this command also cannot be rolled back.

Eg: >truncate table student;

RENAME:  
-------------

It is used to rename the table.

Eg: rename table student to Student1;

DML Commands:

-----------------------

INSERT:

----------

Inserting all columns values.

>insert into student values(101,’siri’,850);

Inserting partial columns values.

>insert into student (roll, name) values(102,’Dhaanvi’);

UPDATE:  
------------

It is used to update the data with in a table.

Eg1: updating all the marks for all the students.

>update student set marks=marks+50;

Eg2: updating marks for only one student.

>update student set marks=marks+30 where name=’siri’;

>update student set marks=marks+30 where marks>800;

>update student set name=’Krishna’ where roll=103;

DELETE:

-----------

It is used to delete the records/ rows from the table.

>delete from student;

//It will delete entire record from the table like truncate command.

[delete command we can rollback inside the transactional area, whereas truncate command we cannot rollback.]

>delete from student where name=’siri’;

DRL (select)

----------------

This command is used to quiring a database tables.

Syntax:

Select col1, col2, …

From tablename

Where condition

[group by column name]

Having condition

Order by column name [asc/ desc]

Eg1:

>select \* from student;

Eg2: restricting the rows by using where clause.

Select (projecting columns)

From (from table name)

Where (condition)

>select \* from student where roll=10;

>select name from student where marks<800;

Using order by: To sort the records.

-------------------

>select \*from student order by marks desc;

Operations:

1.Arithmetic operation: (\*, /, +, -, %)

NOTE: mostly arithmetic operators are used after select statement (90%) and all other type of remaining operators are used in where clause only.

2.Relational operator: (=, >, <, <=, >=, [!= or <>])

3.Logical operator: (AND, OR, NOT)

4.Special operator: (IN, LIKE, …)

Arithmetic operator

---------------------------

>select name, marks, marks+100 from student;

>select name, marks, marks+100 Gracemarks from student;

Note: This temporary name of a column we cannot use inside where clause.

>select name, marks, from student where roll!=10;

Using distinct: It will remove the duplicates. Or getting unique data.

-------------------

>select marks from student;

>select distinct marks from student;

Logical Operator:

-----------------------

Special Operator:

------------------------

IN ----------> NOT IN

BETWEEN -----> NOT BETWEEN

IS NULL -------> IS NOT NULL

LIKE --------> NOT LIKE

>select \*from student where marks IN (800, 500, 900);

-- It will display the marks which are equal to specified values.

>select \*from student where marks IS NULL;

>select \*from student where marks BETWEEN 500 AND 800;

Or

>select \*from student where marks>=500 AND marks<=800;

LIKE -----> NOT LIKE

--It is used to retrieve the data based on character pattern.

1.% -- It represent string or group of characters.

2.\_ -- It represents a single character.

Eg:

>select \*from student where name LIKE ‘r%’; --> name should start with r.

>select \*from student where name LIKE ‘%r%’; --> in name r should be any character.

CONSTRAINTS:

--------------------

--constraints are created on the columns.

--It prevents invalid data entry into our tables.

1. Not Null

2. unique

3. primary key

4. foreign key

5. check: MySql does not support this check constraints

NOTE: some constraints we can apply at column level and some constraints we can apply at table level.

Column level: where we define the column

Not null

Unique

Primary key

Table level: after defining all the columns

Composit key (multi-column primary key)

Foreign key

1.not null:

--------------

--null value is not allowed.

2.unique:

------------

--here duplicates are not allowed.

--here we can insert null values multiple time.

NOTE: whenever we define a unique then automatically DB engine will create an index on that column.

--so searching based on unique column will become fast.

3.primary key:

------------------

--here also DB engine create index of that column.

--values cannot be duplicate and null is also not allowed.

\*\*\*another difference with primary key and unique: In one table we can have multiple unique constraints but in one table we can have only one primary key.

--if we want to apply primary key on multiple columns then it will become composit key.

\*\*Note: with the help of primary key column we can uniquely identify one record of a table.

Example:

//primary key

create table student

(

Roll int primary key,

Name varchar (12) unique not null, -->It is also a primary key but only once we use PK.

Address varchar (12) unique,

Marks int not null

);

//composit primary key

Teacher (name, subject, age, phone, email)

Create table teacher

(

Name varchar (12),

Phone varchar (10),

Email varchar (19),

Age int,

Subject varchar (15),

Primary key (name, phone)

);

Here name and phone will become a composit key, this combination cannot be duplicate.

FOREIGN KEY:

-------------------

--with the help of Foreign key we in force the referential integrity.

--with the help of FK we can establish relationship between 2 tables.

--second table FK must refer to first table primary key.

--PK related FK column must belong to the same datatype but column name can be different.

--FK can accept the duplicate and null value also.

Eg:

Create table department

(

Did int primary key,

Dname varchar (12),

Location varchar (12)

);

Create table employee

(

Eid int primary key,

Ename varchar (12),

Address varchar (12),

Salary int,

Deptid int,

Foreign key (deptid) references department (did)

);

--with the help of FK we establish parent and child relationship among tables.

Here department table will act as parent table.

Employee table will act as a child table.

--the table which contains the FK column will be considered as child table.

NOTE: whenever we try to establish a relationship using FK then DB violates following 2 rules:

1.deletion or updation in parent table (even we cannot drop the parent tables also)

2.insertion in child table (we are not allowed any other value which is not there in parent table (PK))

--to overcome this updation and deletion problem we should use

On delete cascade: if we delete the record from parent then it will also delete in child also.

Or

On delete set null: It will remove the parent class value and set child class value as null

Similarly for update also

On update cascade: If we delete the record from parent then it will place the null in child class.

Or

On update set null

Eg: Foreign key (deptid) reference department (did) on delete cascade on delete set null

Composit Foreign Key: It should refer the composit primary key of the parent table.

------------------------------

Create table a1

(

Name varchar (15),

Address varchar (15)

);

Create table b1

(

Nm varchar (15),

Location varchar (15),

Foreign key (nm, location) references a1 (name, address)

);

Adding constraints to an existing table:

----------------------------------------------------

>create table a1 (id int, name varchar (15));

>alter table a1 modify id int primary key;

Adding foreign key:

-------------------------

>create table b1 (bid int);

>alter table b1 add foreign key (bid) references a1 (id) on delete set null;

Functions in mysql:

--------------------------

--It is used to solve a particular task.

--a sql function must return a value.

--in sql we have 2 types of functions:

1.predefined functions

2.user defined functions

Predefined Function:

------------------------------

1.number function

2.character function

3.group functions or aggregate functions

4.date functions

Number function:

------------------------

1. Abs (): It returns the absolute number.

Eg:

>select abs (-50) from dual; //50 //here dual is a one kind of sudo table.

1. Mod (m, n): It returns the remainder of m/ n.

Eg:

>select mod (10,2) from dual; //0

1. Round (m, n):

>select round (12.43483,3) from dual; //12.435

1. Truncate (m, n):

Eg:

>select truncate (12.43483,3) from dual; //12.434

1. Ceil ()
2. Floor ()
3. Greatest () least (): It will return biggest and smallest value from the list of arguments.

EG:

>select greatest (10,12,8,15) from dual; //15

NOTE: from a single column if we want to max and min value then we should use group functions like max () min ();

Character functions:

---------------------------

Upper ()

Lower ()

Length ()

Replace ()

Concat ()

Substring ()

Eg:

>select upper (name) from employee where eid=101;

> select upper (name), length (name) length from employee where eid=101;

>select substr (‘ratan’, 3, 2) from dual; //ta afcz

Date function:

-------------------

1. Sysdate (): It will return the current date and time.

Eg:

>select sysdate() from dual;

2. date\_format(): day month year

>select date\_format (sysdate (), ‘%d %m %Y’);

3. adddate (): adding extra days, months, years etc. to current date.

Syntax: adddate(date, INTERVAL value unit);

DAY

HOUR

YEAR

MONTH

WEEK

Eg:

>select adddate (sysdate(), INTERVAL 4 MONTH);

group function or aggregate function:

-----------------------------------------------------

--these functions operate over several values of a single column and then result in a single value.

Max ()

Min ()

Avg ()

Sum ()

Count (\*) //count all records

Count (column name) //number of all records except null values

Example:

>select sum (marks) result from student;

>select count (\*) from student;

>select count (deptid) from employee; //null values are not considered

Group by clause:

----------------------

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

--the main purpose of group by clause is to group the records.

--this clause is mostly used with the group functions only.

--it is used to divide the similar data items into set of logical groups.

Short syntax:

Select col\_name from table group by col\_name;

Full syntax:

Select col\_names

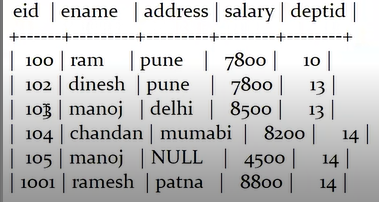
From

Tablename

[where condition] ---optional

Group by col\_names

[having condition] ---optional



--the above data is called as detailed data and after performing the group by, we get the summarized data which is useful for analysis.

>select sum (salary) from employee; //it will calculate the salary from whole table

>select deptid, sum (salary) from employee group by deptid; //dept wise total salary

RULE:

1.group functions we cannot use inside the where clause.

2.other than group function all the columns mentioned inside the select clause must be there after the group by clause otherwise (oracle db will give an error and mysql may give the unexpected result).

>select deptid, ename, max (salary) maximum, min (salary) minimum, avg (salary) from employee group by deptid,ename;

Having:

---------

After group by we are not allowed to use where clause in place of where clause we should use having clause after group by clause.

With where clause:

>select deptid, sum (salary), from employee where deptid IN (13,14) group by deptid having sum (salary) >10000;

Without using where clause:

>select deptid, sum (salary) from employee group by deptid having sum (salary)>10000;

>select deptid, count (\*) from employee where deptid IN (11,12,14) group by deptid having count (\*)> 2;

NOTE: It is not mandatory to put only group function inside the having clause.

JOINS:

-------

Join is used to receive data from multiple tables or by using joins we can combine records from multiple tables.

There are following types of joins:

1.Inner Join

2.Outer Join

Left outer join

Right outer join

Full outer join

3.Self join

4.Cross join (cartesian product)

NOTE: when we try to get the data from more than one table without using joining condition, then it is called cross join, in this case every record of the first table will be mapped with every record of the second table.

--with the cross join we don’t get the meaningful data, in order to get the meaningful data we need to use other types of joins.

INNER JOIN:

-----------------

--here we need to apply joining condition on the common data from both tables.

--if ambiguity is there in column name (both table having the same column name) then we need to use alias support.

--this inner join returns the matching record from the DB table based on common column.

>select \* from department INNER JOIN employee ON department. did= employee. deptid;

Q) given the employee details who is working in ‘marketing’ department.

>select eid, ename, address, salary from department INNER JOIN employee ON department. Did= employee. deptid AND department. dname=’marketing’;

With ALIAS support:

---------------------------

>select e. ename, e. eid, e. address, e. salary, d. location from department d INNER JOIN employee e ON d. did= e. deptid AND d. dname=’marketing’;

Another syntax of INNER JOIN (without using INNER JOIN command):

--------------------------------------------------------------------------------------------

> select e. ename, e. eid, e. address, e. salary, d. location from department d, employee e WHERE d. did= e. deptid AND d. dname=’marketing’;

Left outer join:

--------------------

--to get the unmatched records from the left table use left outer join (it shows the details of left table and null value for the right table).

>select d. dname, e. ename, e. address, e. salary from department d LEFT JOIN employee e ON d. did=e. deptid;

Right outer join:

----------------------

--to get the unmatched records from the right table use right outer join (it shows the details of right table and null value for the left table).

> select d. dname, e. ename, e. address, e. salary from department d RIGHT JOIN employee e ON d. did=e. deptid;

Full outer join:

-------------------

--it is a combination of LEFT outer join and RIGHT outer join.

--it displays the null values both side for all the unmatched records.

NOTE: full outer join is not supported by the mysql DB.

--in order to use full outer join in mysql, then we should use UNION of left join and right join.

> select d. dname, e. ename, e. address, e. salary from department d LEFT JOIN employee e ON d. did=e. deptid UNION select d. dname, e. ename, e. address, e. salary from department d RIGHT JOIN employee e ON d. did=e. deptid;

Self Join:

------------

--here we use joining a table to itself.

--here joining condition column must belongs to same datatype.

Note: if we want to compare two table same column value then we use INNER JOIN where as if we want to compare two different column values with in a single table then we must use self join.

\*\*whenever a table contains hierarchical data then only we allow to use self join.

Ex:

Employee ----> manager

Student -------> monitor

When we use self join, we must take support of alias.

Ex:

Create table emp

(

Eid int primary key,

Ename varchar (12),

Salary int,

Manager int

);

Insertion:

>insert into emp values (100,’ram’, 7800, null); //RAM does not have any manager

>insert into emp values (110,’ravi’, 7200, 100); //RAVI manager is Ram

>insert into emp values (112,’arjun’, 7500, 100); //ARJUN manager is RAM

>insert into emp values (114,’sunil’, 7000, 101); //SUNIL manager is RAVI

Q) display the emp name and their manager name

--here we need to use SELF JOIN

>select e1. Ename EMPLOYEE, e2. Ename MANAGER from emp e1, emp e2 where e1. Manager=e2. Edi;

Subqueries:

----------------

--a query inside another query is called subquery or nested query.

--sub queries are used to retrieve the data from single or multiple tables based on more than one step process.

--here outer query is called parent query and inner query is called child query.

--child query will execute first then only parent query will be executed.

Child Query: It provides values/ data to the parent query.

Parent Query: It receives the values/ data from the child query.

--in child query we cannot use order by clause, but parent query can use.

--group by clause can be used in both queries.

Subqueries we can categories into following categories:

--------------------------------------------------------------------------

1.single row and single column subquery (scalar value SQ)

2.multiple row single column subquery

3.multiple column subquery

1.single row and single column subquery (scalar value SQ):

------------------------------------------------------------------------------

--here child query will return only a single value.

--here mostly same column name which is the where clause of the parent query, will be there inside the select clause of child query.

Q) WAQ to display emp details who is working in marketing department.

Using JOIN:

>select eid, ename, address from department INNER JOIN emp ON dept. did=emp. Deptid AND dname=’marketing’;

Using SQ:

>select eid, ename, address from emp where deptid= (select did from department where dname=’marketing’);

Q) WAQ to display emp details who is working with some person (suresh)

>select \* from emp where deptid= (select deptid from emp where ename=’chandan’);

Q) WAQ to display emp details who are getting more salary than average salary from emp table.

>select \* from emp where salary > (select avg (salary) from emp);

Q) WAQ to display second highest salary employee details

First highest salary emp:

>select \* from emp where salary = (select max (salary) from emp);

Second highest salary emp:

>select \* from emp where salary= (select max (salary) from emp where salary< (select max (salary) from emp)).

Q) WAQ to display details of emp who is working under RAVI

>select \* from emp where manager= (select eid from emp where ename=’ravi’);

2.multiple row single column subquery:

-----------------------------------------------------

--in multi-row, single column SQ, child query will return multiple rows and single column to the parent query.

--in this case in parent query we should use one of following operators:

IN

ANY

ALL

Ex:

IN: It checks equal to any number in the list (using OR)

>select \*from emp where salary IN (select salary from emp where eid>100);

ANY: It compare any value in the list.

ALL: It compares all values in the list.

Ex:

Salary > any (-------); here it checks salary should be greater than any of 4 values in the list.

Salary > all (-------); here it checks salary should be greater than all of 4 values in the list.

>any (10,20,30,40);

Ex:

< any (): less than any – less than maximum

> any (): greater than any – greater than equal to minimum

= any (): It is equal to IN operator

Ex:

5< all (10,20,30,40)

< all (): less than all – less than minimum

>all (): greater than maximum

= all (): equal to all – It is meaningless (because one value cannot be equal to 3 or 4 value)

Ex:

>select \* from emp where salary = ANY (select salary from emp where eid=100);

Q) WAQ to display the emp who is getting max salary in each department?

>select \* from emp where salary IN (select max (salary) from emp group by deptid);

3.multi column subquery:

-----------------------------------

--if we try to compare multiple column values of the childquery with the multiple column values of the parent query then we use this type of SQ

Syntax:

>select \* from tab\_name where (column1, column2, ….) IN (select column1, column2, … from table where condition)

Q) WAQ to display the emp whose salary and did matches with the salary and did of a RAM.

>select \*from emp where (salary, deptid) IN (select salary, deptid from emp where ename=’RAM’);

SQ in DML:

---------------

SQ in insert:

>create table x1 (id int, name varchar (12));

Eg1:

>insert into x1 (select eid, ename from emp);

Eg2:

>insert into x1 values (500, (select ename from emp where eid=1000);

SQ in update:

Here SQ is allowed inside where clause or set clause.

EX:

>update x1 set name=’ramesh’ where id= (select eid from emp where ename=’ram’;

>update x1 set name= (select ename from emp where eid=100) where id= (select eid from emp where ename=’ram’);

SQ in delete:

>delete from x1 where id= (select eid from emp where ename=’ram’);

Autoincrement in MYSQl:

-----------------------------------

--this is for auto generate the ID field.

--mysql supports the auto increment where as Oracle DB use sequence concept to generate the ID field automatically.

--autoincrement starts from 1.

>Create table student1

(

Roll int primary key auto\_increment,

Name varchar (15),

Marks int

);

//autoincrement start from 1001;

>alter table student2 auto\_increment=1001;

Limit:

--------

Q) WAQ to get 4 highest salary paid employee:

>select \* from emp order by salary desc LIMIT 4;

Getting the records from 3 or 6 row

>select \* from emp order by salary desc LIMIT 4,2; //It return 5,6 records.

Relationship among tables:

-------------------------------------

At table level we have 3 types of relationship:

1.one to one (person ---> Aadhaar Card)

2.one to many (Father --- child, teacher --- student)

3.many to many (student --- course, book ---- author, movies --- actors)

One to One:

----------------

Create table person

(

Id int primary key,

Name varchar (12),

Address varchar (12),

Mobile varchar (12),

Email varchar (15)

);

Create table DrivingLicense

(

Dlid int primary key,

issueDate date,

expDate date,

rto varchar (12),

personId int unique,

foreign key (personId) from person(id)

);

One to many:

-----------------

Dept (did, dname, location);

Emp (eid, ename, address, salary, deptId ---- FK refer dept (did));

Many to many:

---------------------

Student (roll, sname, address, mobile);

Course (cid, cname, fee, duration);

NOTE: whenever we have many to many relationship we need to take the help of 3rd linking table.

Student\_course (roll, cid);

Create table student (

Roll int primary key,

Sname varchar (12),

Address varchar (12),

Mobile varchar (10)

);

Create table course (

cid int primary key,

cname varchar (12),

fee int,

duration varchar (12)

);

Create table student\_course (

Roll int,

cid int,

foreign key (roll) references student (roll),

foreign key (cid) references course (cid)

);

//getting the student details who enrolled in java

>select s. roll, s. sname, s. address, s. mobile, c. cname, c. fee, c. duration from student s INNER JOIN course c INNER JOIN student\_course sc ON s. roll=sc. Roll AND c. cid=sc. cid AND c. cname=’Java’;

Using where clause:

>select s. roll, s. sname, s. address, s. mobile, c. cname, c. fee, c. duration from student s, course c, student\_course sc WHERE s. roll=sc. Roll AND c. cid=sc. cid AND c. cname=’Java’;

ER diagram (Entity relationship diagram):

-------------------------------------------------------

Java developers allowed to perform CRUD operations (DML + DRL)

Create a record (insert)

Retrieve a record (select)

Update a record (update)

Delete a record (delete)

**JDBC:**

--------------

Java Data Base Connectivity.

--java mostly used in industry to develop business application.

--common and general things required in a business application.

1.maintaining the business data permanently in a secure and easily retrieval manner.

2.processing the data according to the business rule.

3.presenting the data to the user in user understanding format.

--we store the business data for a typically business application inside the RDBMS software.

Nosql (mongo db --- json data)

Q) why java and DB communication is required?

Ans: DBMS are excellent in data storage in secure and easily retrieval manner but it is very poor in processing and presenting the data in user – understandable format.

--java is excellent in processing the data and presenting the data but java is very poor in storing the data (in java we can store the data using serialization processing).

--so in order to develop a powerful business application we need to communicate our java application with the DB server.

Q) How java DB communication is possible?

--Java app can do any task only through the method calls and objects, these method calls and objects are not directly understandable to the DB software.

--and the DB can understand only sql, java compiler does not accept the sql syntax directly.

--in spite of heterogeneous platform java DB communication is possible through the JDBC.

translator

Chinese ----------------------- Russian

JDBC driver software

Java -------------------------- DB

--JDBC is a technology that enables any kind of java application to communicate with any kind of DB software in a standard manner.

JDBC technology is given by sun microsystem.

JDBC technology comes in the form of a specification (it is documentation which describes rules and guidelines to develop a particular software “JDBC driver software”)

--JDBC driver software is the implementation of the JDBC specification, which will act as a translator software to communicate our java app with the DB software.

--JDBC specification will be implemented by the DB vendor or any 3rd party vendor also and develop the “jdbc driver software’

--each DB software have their own JDBC driver software, this driver software comes in the form of a .jar file.

Mysql: mysql – connector. Jar

Oracle: ojdbc6. Jar

Postgres: postgress. Jar

--Java developer needs to get/ download the JDBC driver related jar file from the internet, and in order to work with that jar file, we need to set that jar file inside the class path of our application.

Java application {

Classes (method)

Interface

}

--compile all the .java files ------> .class files -----> zip all the .class file ------> inside a .jar file

Siriapp {A.java, B.java, Demo.java} --------> make them a1.jar file

Class X {//here we need to set a1.jar file in the class path of our application

Main () {

A a1=new A ();

}

}

--in order to communicate java application and DB server, java developer need the “JDBC driver software” and “JBDC api” to perform the DB operation from the java application.

--JDBC api comes in the form of 2 packages:

1. java.sql package

2. javax.sql package

JDBC Client:

---------------------

--In java DB communication, our java application will act as a client because java application needs the services of the DB server. So our java application is a JDBC client.

Responsibility of the JDBC client:

--------------------------------------------

1. Requesting the DB connection.

2. Submitting the appropriate SQL statement to the DB server in the form of string.

3. Processing the result given by the DB server.

4. Dealing with exceptions if any.

5. Managing the transactions whenever it is required.

6. Closing the connection once done with the DB operation.

JDBC Driver:

---------------------

It is a translation software written in java according to the JDBC specification (it comes in the form of jar file)

Responsibility of JDBC driver software:

----------------------------------------------------

1. Establishing the connection.

2. Receiving the JDBC method calls (java method call) and translate them into DBMS understandable format. (SQL) and forward them to the DB software.

3. Translating the DB software given result into the java format (java objects) and returns that object to the JDBC client.

Step of connect our java application with the DB server (mysql):

-------------------------------------------------------------------------------------

1. Download/ get the JDBC driver related jar file (mysql – connector.jar) and set that jar file inside the class path of our application.

2. Load the driver related main class into the memory.

3. Prepare the connection string.

4. Establish the connection.

5. After performing the CRUD operation close the connection.

Setting jar files in the class path of our application:

-------------------------------------------------------------------

Right click on the project -----> build path ----> configure build path ------> libraries -----> class path --> add external jars ---> select the downloaded jar file -----> apply and clone.

Demo.java

---------------

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**public** **class** Demo {

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

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** {

Connection conn=DriverManager.*getConnection*(url,"root","root");

**if**(conn !=**null**)

System.***out***.println("connected....");

}**catch**(SQLException e) {

e.printStackTrace();

}

}

}

Output:

Connected…….

Performing DML (Data Manipulation Language):

----------------------------------------------------------------

Insert

Update

Delete

Inserting records into the table using java application:

-----------------------------------------------------------------------

--once we get the connection object then we can execute any sql statement from our java application to the DB server

--for that we require a “statement” object.

Java. sql. Statement (I)

--we get the statement object by using:

Statement st= conn. createStatement ();

--once we get the statement object, in order to execute any statement from our java application we need to use following methods:

1. public int executeUpdate (String dml) throws SQLException;

--it will return the number of rows affected due to the supplied sql statement.

2. public ResultSet executeQuery (String drl) throws SQLException

3. public Boolean execute (String anysql) throws SQLException

--using this method we can execute any type of SQL statement (DML, DRL, DDL);

--CRUD

Ex:

create table student (

roll int primary key,

name varchar (12),

marks int

);

Inserting data into table:

----------------------------------

Demo.java

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** Demo {

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

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** {

Connection conn=DriverManager.*getConnection*(url,"root","root");

Statement st=conn.createStatement();

**int** x=st.executeUpdate("insert into student values(10,'Ram',800)");

**if**(x>0)

System.***out***.println("record inserted successfully!....");

**else**

System.***out***.println("not inserted...");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

}

}

Closing the connection solution1:

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** Demo {

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

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

Connection conn=**null**;

**try** {

conn=DriverManager.*getConnection*(url,"root","root");

Statement st=conn.createStatement();

**int** x=st.executeUpdate("insert into student values(10,'Ram',800)");

**if**(x>0)

System.***out***.println("record inserted successfully!....");

**else**

System.***out***.println("not inserted...");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

**finally** {

**try** {

conn.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

Closing the connection solution2:

---------------------------------------------

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** Demo {

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

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

Statement st=conn.createStatement();

**int** x=st.executeUpdate("insert into student values(10,'Ram',800)");

**if**(x>0)

System.***out***.println("record inserted successfully!....");

**else**

System.***out***.println("not inserted...");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

}

}

Using statement object inserting dynamic details in the table:

-----------------------------------------------------------------------------------

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.Scanner;

**public** **class** Demo {

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

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter roll number:");

**int** roll=sc.nextInt();

System.***out***.println("Enter name:");

String name=sc.next();

System.***out***.println("Enter marks:");

**int** marks=sc.nextInt();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

Statement st=conn.createStatement();

**int** x=st.executeUpdate("insert into student values("+roll+",'"+name+"',"+marks+")");

//here every thing is in double quotes(“”) but strings are having extra single quotes (‘’).

**if**(x>0)

System.***out***.println("record inserted successfully!....");

**else**

System.***out***.println("not inserted...");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

sc.close();

}

}

Output:

Enter roll number:

20

Enter name:

siri

Enter marks:

850

record inserted successfully!....

NOTE: if we use the Statement (I) object to perform any DB operation then statement object has the following limitations:

1.complexities to concatenate dynamic variables.

2.whenever we pass any query to the DB using statement object, DB engine will perform following tasks every time to execute that query.

a. query compilation

b. query plan generation

c. query optimization

--for the same query doing these operations every time, it will degrade the performance.

--to tell the DB engine to perform above 3 task only at first time with the value or without the value and put that query in the cache, next time on words just add the dynamic values and execute the query.

--for that we need to use “java. Sql. PreparedStatement” object instead of “java. Sql. Statement” object.

Statement

|

PreparedStatement

3.Sql Injection (delete from student)

Note: statement object is relatively faster than PreparedStatement.

--to get the PreparedStatement object we need to call PreparedStatement (String sql) method on the connection object, with the value or without the value.

--in case of without the value, we need to use placeholder (?)

Ex:

PreparedStatement ps = conn. prepareStatement(“insert into student values (10,’Ram’,800)”);

PreparedStatement ps = conn. prepareStatement(“insert into student values (?,?,?)”);

--in case of placeholders, before executing the query we need to bind the appropriate values with the corresponding placeholders by calling various types of set XXX (--) method on the PreparedStatement object.

Ex:

Ps. setInt (1, roll);

Ps. setString (2, name);

Ps. setInt (3, marks);

--these roll, name, marks are the variables which holds the dynamic data.

--after binding the appropriate placeholder we need to execute the query by using one of the following methods of the PreparedStatement object.

Public int executeUpdate ();

Public ResultSet executeQuery ();

Public boolean execute ();

Example: Using PreparedStatement:

--------------------------------------------------

Demo.java

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.util.Scanner;

**public** **class** Demo {

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

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter roll number:");

**int** roll=sc.nextInt();

System.***out***.println("Enter name:");

String name=sc.next();

System.***out***.println("Enter marks:");

**int** marks=sc.nextInt();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("insert into student values (?,?,?)");

ps.setInt(1, roll);

ps.setString(2, name);

ps.setInt(3, marks);

**int** x=ps.executeUpdate();

**if**(x>0)

System.***out***.println("Record Inserted successfully");

**else**

System.***out***.println("Not Inserted..");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

sc.close();

}

}

Output:

Enter roll number:

3

Enter name:

Dhaanvi

Enter marks:

900

Record Inserted successfully

Updating record:

------------------------

Ex: update roll number

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.util.Scanner;

**public** **class** Demo {

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

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter roll number:");

**int** roll=sc.nextInt();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("update student set roll=? where name='Dhaanvi'");

ps.setInt(1, roll);

**int** x=ps.executeUpdate();

**if**(x>0)

System.***out***.println("Record Updated successfully");

**else**

System.***out***.println("Not Updated..");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

sc.close();

}

}

Output:

Enter roll number:

60

Record Updated successfully

Ex2: update marks

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.util.Scanner;

**public** **class** Demo {

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

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter grace marks:");

**int** gmarks=sc.nextInt();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("update student set marks=marks+? where marks<850");

ps.setInt(1, gmarks);

**int** x=ps.executeUpdate();

**if**(x>0)

System.***out***.println(x+"Record Updated successfully");

**else**

System.***out***.println("Not Updated..");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

sc.close();

}

}

Output:

Enter grace marks:

50

2 Records updated successfully

Remove record:

----------------------

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** java.util.Scanner;

**public** **class** Demo {

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

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter roll number to delete:");

**int** roll=sc.nextInt();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("delete from student where roll=?");

ps.setInt(1, roll);

**int** x=ps.executeUpdate();

**if**(x>0)

System.***out***.println(x+"Record deleted successfully");

**else**

System.***out***.println("Record does not exit..");

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

sc.close();

}

}

Output:

Enter roll number to delete:

60

1 Record deleted successfully

Performing DRL operation (select):

-----------------------------------------------

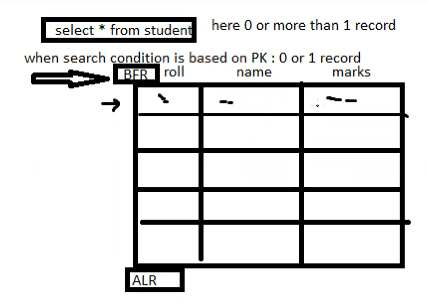
PreparedStatement ps= conn.prepareStatement("select \* from student");

ResultSet rs=ps.executeQuery();

--here we get records from DB, in form of “java. Sql.ResultSet (I)” object.

--in order to get the details from the ResultSet object we should know the structure of the ResultSet object.

Note: ResultSet object structure will depends upon the query what we have executed, it does not depend upon the table structure.



--now we need to move the cursor from the BFR (Before First Row) to the record area, and fetch the data from each column.

--to move the record from the BFR to the record area we have a method in the ResultSet object:

Public Boolean next ();

//It will move the cursor to the record area and if any record is there it returns true and if it points the ALR (After Last Row) then it returns false.

--to access/ get the column field value from the cursor pointed record we need to call following methods on the ResultSet object:

Public XXX getXXX (String columnName); //here XXX will replace with proper datatype of the value.

Public int getInt (“roll”);

Public String getString (“name”);

Public int getInt (“marks”);

Ex:

int r= public int getInt (“roll”);

string n=public String getString (“name”);

int m=public int getInt (“marks”);

Note: To move the cursor from the BFR to the record area we need to follow following strategy.

1.If we sure that only 0 or 1 record will be there (atmost 1): in that case we need to use if-else condition. (when our search condition in on primary key)

2. If there may be a chance of getting more than 1 record then we need to use “while loop”.

Example:

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**public** **class** Demo {

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

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("select \* from student");

ResultSet rs=ps.executeQuery();

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

**int** r=rs.getInt("roll");

String n=rs.getString("name");

**int** m=rs.getInt("marks");

System.***out***.println("Roll is:"+r);

System.***out***.println("Name is:"+n);

System.***out***.println("Marks is:"+m);

System.***out***.println("======================");

}

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

}

}

Output:

Roll is:10

Name is: Ram

Marks is:850

======================

Roll is:20

Name is: Siri

Marks is:850

======================

Roll is:30

Name is: Puri

Marks is:870

======================

Roll is:40

Name is: Naga

Marks is:950

======================

Roll is:50

Name is: Madhu

Marks is:910

======================

Developing a searching application using JDBC:

---------------------------------------------------------------

Demo.java

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.Scanner;

**public** **class** Demo {

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

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter roll number:");

**int** roll=sc.nextInt();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("select \* from student where roll=?");

ps.setInt(1,roll);

ResultSet rs=ps.executeQuery();

**if**(rs.next()) {

**int** r=rs.getInt("roll");

String n=rs.getString("name");

**int** m=rs.getInt("marks");

System.***out***.println("Roll is:"+r);

System.***out***.println("Name is:"+n);

System.***out***.println("Marks is:"+m);

System.***out***.println("======================");

}

**else** {

System.***out***.println("Record does not exist..");

}

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

sc.close();

}

}

Output:

Enter roll number:

20

Roll is:20

Name is: Siri

Marks is:850

======================

Getting only marks:

----------------------------

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.Scanner;

**public** **class** Demo {

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

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter roll number:");

**int** roll=sc.nextInt();

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("select \* from student where roll=?");

ps.setInt(1,roll);

ResultSet rs=ps.executeQuery();

**if**(rs.next()) {

**int** m=rs.getInt("marks");

System.***out***.println("Marks is:"+m);

}

**else** {

System.***out***.println("Record does not exist with roll"+roll);

}

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

sc.close();

}

}

Output:

Enter roll number:

30

Marks is:870

Getting all the record based on marks:

---------------------------------------------------

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**public** **class** Demo {

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

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

String url="jdbc:mysql://localhost:3306/web19sb101db";

**try** (Connection conn=DriverManager.*getConnection*(url,"root","root")){

PreparedStatement ps= conn.prepareStatement("select \* from student where marks<870");

ResultSet rs=ps.executeQuery();

**boolean** flag=**true**;

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

flag=**false**;

String n=rs.getString("name");

**int** m=rs.getInt("marks");

System.***out***.println("Name is: "+n);

System.***out***.println("Marks is: "+m);

System.***out***.println("======================");

}

**if**(flag) {

System.***out***.println("Record does not exist");

}

}**catch**(SQLException e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

}

}

Output:

Name is: Ram

Marks is: 850

======================

Name is: Siri

Marks is: 850

======================

DAO (Data Access Object) pattern:

---------------------------------------------

Student:

------------

create table student (

roll int primary key auto\_increment,

name varchar (15),

marks int,

email varchar (15),

password varchar (10)

);

Course:

----------

create table course (

cid int primary key auto\_increment,

cname varchar (15),

fee int

);

Course\_student:

-----------------------

create table course\_student

(

cid int,

roll int,

foreign key (cid) references course(cid),

foreign key (roll) references student(roll)

);

**Normalization:**

--Normalization is a process of organizing the data in DB to avoid the redundancy (duplication).

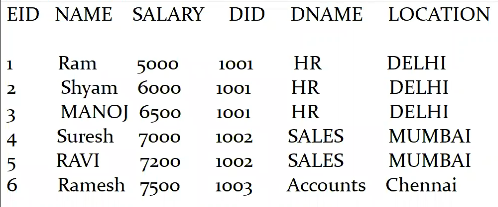
--Because of data redundancy there are several problems in the DB.

Anomalies in DB:

------------------------

--An anomaly is something is different from what is normal or usual (inconsistency) or abnormalities.

--lets try to have a single table to manage employee details.



Here entire department related data is repeated for each employee. (Data redundancy)

Data redundancy: when same data is stored multiple time unnecessary in DB.

Redundancy occurs when we try to keep all the data in a single table.

Problems with data redundancy:

--------------------------------------------

1.Insertion, updation, deletion anomalies

2.inconsistency of DB

3.increases the DB table size and slow the performance while fetching the data.

Insertion anomalies: when certain data cannot be inserted into the table without the presence of other data.

Updation anomalies: when we want to update, a single piece of data, but is must be updated at all its copies.

Deletion anomalies: If we delete some data, it causes deletion of some other data.

--to solve the above problem we need to normalization(decompose) the table into multiple related tables.

NOTE: the main purpose of normalization is to avoid the data redundancy and maximize the efficiency of the DB.

In normalization we should split a table in multiple tables so that each table should contains a single idea/ concept.

--with the normalization we refine a big table into multiple related tables.

So the above table we should split into 2 tables:

1. Department
2. Employee

--to normalize a table we have different types of normal forms:

1.1NF

2.2NF

3.3NF -----> up to 3NF data redundancy is almost minimized or removed

4.BCNF

5.4NF

6.5NF

--each normal form provides a different level of refinement of a DB.

First Normal Form:

-------------------------

--table should not contains any multi value attributes (comma/ space separated values)

--each cell should contains only atomic value

--a table should not have the repeating columns

Ex:

EMPID NAME DEPT\_NAME

1 RAM HR, SALES

2 RAVI marketing

3 AMIT HR, Accounts

So the above schema is in 0NF or unnormalized.

Lets convert it into the 1NF.

Solution1:

EMPID NAME DEPT\_NAME

1 RAM HR

1 RAM SALES

2 RAVI marketing

3 AMIT HR

3 AMIT Accounts

Here empid cannot be PK, here we need to take PK as composite PK (empid, dept\_name)

Solution2:

EMPID NAME DEPT\_NAME1 DEPT\_NAME2

1 RAM HR SALES

2 RAVI marketing Null

3 AMIT HR Accounts

--the above solution is violates the 1NF because of repeating column.

Solution3:

--we can divide the table into 2 tables:

1.as a base table

2.as reference table

Emp table:

EMPID NAME

1. RAM
2. RAVI
3. AMIT

Emp\_dept table:

EMPID DEPT\_NAME //here EMPID will be FK

1 HR

1 SALES

2 marketing

3 HR

3 Accounts

Second Normal Form:

------------------------------

To understand the 2NF or to normalize a table in 2nd NF we need to understand following concepts:

1.functional dependency

2.super key

3.candidate key

4.prime attribute

5.non-prime attribute

Key:

------

--generally we keep in a table, so that latter we can retrieve it easy manner.

--in a table all column has a unique name but for a row we don’t have a unique name. So in order to find a row uniquely we required a key.

--so a key is an attribute or set of multiple attributes that uniquely identify a row/ record in a table.

School: --- student

Super key:

--------------

All the valid combination of attributes by using we can find a row uniquely in a table.

Eg:

Student (roll, name, marks, address, dob, email)

Combinations:

1.roll

2.roll name

3.roll name address

4.name email

5.name address

6.name dob

7.address email

8.dob email

Candidate key:

--------------------

It is a minimal set of super key.

Note: a candidate key should not have a subset as another super key.

1.roll

2.email

3.name address

4.name dob

Primary key:

-----------------

Here from the candidate key data base administrator will choose a PK, generally it will be minimum number of attribute declared as PK.

Some time we can make other than minimum value attribute also as a PK (composite key).

Prime attribute:

----------------------

Those attribute that are part of any candidate key is called prime attribute.

--and those attributes which are not part of any candidate key is known as non-prime attribute.

In the above example: marks will be a non-prime att ribute.

1NF does not eliminate redundancy.

--2NF applies in a table which is having a composite key, i.e a table with a PK compound with two or more attributes.

NOTE: a table with a single column PK is automatically in 2NF.

--to be in 2NF, a table must be in 1NF and the table must not contains any partial dependency.

Partial dependency:

---------------------------

Note: if the proper subset of a candidate key determines non-prime attribute, then it is called partial dependency.

--the normalization of 1NF table to the 2NF involves the removal of partial dependency.

--if any PD exist, we remove the partial dependency attribute from the table by placing them in a new table.

Eg:

ROLL CID FEE

1 c1 1000

2 c2 1500

1 c3 2000

3 c4 1000

3 c1 1000

2 c5 2000

--here there are many courses having same fee.

--here there will be only one candidate key (roll and cid) that can uniquely identify the record.

Prime attribute: (roll and cid)

Non-prime attribute: (fee)

Here fee is dependent on the CID, which is the example of partial dependency. Because CID is a subset of candidate key.

--the above table is not in 2NF.

Convert above table into 2NF:

Table1

(roll, cid)

Table2

(cid, fee)

Third Normal Form(3NF):

----------------------------------

Although 2NF relations have less redundancies than those in 1NF still have a chance of data redundancy because of transitive dependency.

--in order to make a table in 3NF we need to remove Transitive dependency (TD) from the table.

A relation will be in 3NF if only:

1.It should be in 2NF.

2.There should not be any Transitive dependency.

Transitive Dependency: if a non-prime attribute is transitively depends on primary key.

Eg:

If A---------------->B and B----------> C then A---------> C is called TD

Roll Name Age Country State

1 Ram 25 INDIA MP

2 Ram 35 INDIA UP

3 Ravi 28 INDIA MAHARASTRA

Functional dependencies:

Roll -----> name

Roll -----> age

Roll ------> state

State -----> country

--here the candidate key will be (roll)

Roll----> state and state ----> country ,,, so country is transitively dependent on roll, and it will violates 3NF

--to convert this relation in 3NF we need to decompose this relation in multiple related tables:

Student (roll, name, age, state)

State\_country (state, country)

Transaction management:

----------------------------------

Transaction: It is a set of related operations used to perform a logical unit of work.

Ex:

Withdrawing money from the ATM

Transferring amount from one account to another.

--transaction access data using read and write operations.

--if a transaction fails it should not be resumed, instead it should be restarted.

In sql to manage the transaction we need to use TCL (Transaction control language) commands.

Rollback

Commit

Savepoint

In order to maintain consistency in DB, before and after the transaction certain properties should be followed these are called ACID properties:

A: Atomicity: (All or nothing rule) the entire transaction takes place at once or does not happen at all.

C: Consistency: the DB must be consistent before and after transaction.

I: Isolation: multiple transactions should occurs independently at a time without any interfaces.

D: Durability: The changes of a successful transaction should be permanent even if system failure occurs.

--if we perform any DML operation on a table inside a transaction area, that operation will be partially committed (it will be stored in the local copy)

--and any time we can rollback these operation.

--but once we do commit, then only it will be stored permanently.

Note: In mysql it is by default auto commit is enabled.

Note: In the DB all the DDL statements are by default committed.

--in mysql to disable the autocommit mode:

>set autocommit=0;

To start the transaction area in mysql:

>start transaction;

\*\*\*If we commit the DB, we delete anything after commit we cannot rollback.

>savepoint p1;

>rollback to p1;

INDEX IN DB:

-----------------

Index is a DB object which is used to improve the performance of DB by minimizing the number of disk access required when a query is processed.

--indexes are created on the DB table columns.

--whenever we try to retrieve the data by using index column then DB server retrieve the data from DB very fastly.

--generally indexes are created by data base administrators (DBA) at the time of DB schema design.

--in DB indexes are created in 2 ways:

1.automatically

2.manually

--whenever we are defining PK or unique constraint on a particular column, then DB server automatically create an index on those columns.

We can also create index explicitly by using following syntax:

>create index index\_name ON tables\_name (column1, column2, …);

Book: 1000 pages --------> index (keyword <------> page number) (4 pages)

Ex:

Suppose we have salary like:

1.5000

2.1200

3.7800

4.1000

5.9500

It has 1 lakh records

Salary = 3000

--index maintains order data, by default it ascending order.

--if we create an index on salary column then our salary data will be in sorted order ascending:

4.1000

2.1200

1.5000

3.7800

5.9500

--internally index uses B tree data structure to store the indexed column value.

[1000,1200,5000,7800,9500]

|

-----------------------------------------------------------------

| |

[1000,1200] [5000,7800,9500]

Here 4, 2, 1, 3, 5 is the address of record (rowid) and 1000, 1200, 5000, 7800, 9500 is the real data

The rowid contains 3 parts:

1.file number (table name)

2.data block number

3.record number

--if we try to search the details of the employee based on salary column (indexed column) then before query executes on the real table, it verifies on the index.

Note: whenever “where clause” contains!= or IS NULL or IS NOT NULL operator then DB does not searches for index for those columns even though index is created/ applied on those column.

--so if we frequently verify a particular column value then it is better to create an index on that column.

Dropping index:

---------------------

>drop index index\_name

Autoincrement: