**SQL PROJECT ASSIGNMENT 1**

1)

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

DEPTNO(PK) DNAME

10 AC

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

Table EMP:

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

EMPNO ENAME DEPTNO(FK)

101 ROCK 10

102 JACK 10

103 MARK 10

104 JERRY 10

**Answer:**

In the parent table DEPTNO is used as a primary key and in the child table it is used as a foreign key.

The child table uses the column of the parent table as a primary key and referred as a foreign key.

2. What are the four components of a database management system?

Answer:

A database management system (DBMS) is a computer program designed to manage a large amount of structured data, and run operations on the desired data requested by the users. The best example of DBMS is Banking. All the transactions that take place are based on a defined software program which keeps track of all the data.

There four main components on which the working of a DBMS depends. This includes:

* Data: The main component is the data. The entire database is set based on the data and the information processed based on it. This data acts as a bridge between the software and hardware components of DBMS. This can further be divided into three varieties:
  + User Data – The actual data based on which the work is done
  + Metadata – This is the data of the data, i.e., managing the data required to enter the information
  + Application MetaData – This is the structure and format of the queries

To simplify it, in a table, the information given in each table is the User Data, the number of tables, rows and columns is the MetaData the structure we choose is the Application MetaData.

* Hardware: These are the general hardware devices which help us save and enter the data like hard disks, magnetic tapes, etc.
* Software: The software acts as a medium of communication between the user and the database. Based on the user’s requirement, the database can be modified and updated. To perform operations on the data, query languages like SQL are used.
* Users: No function can be performed without the Users. Thus, they form the fourth most important component of DBMS. The information entered into a database is used by the User or the administrator to perform their business operations and responsibilities.

3. What is the distinction between SQL and SQL plus?

Answer:

SQL follows a standard format wherein the basic syntax and commands used for DBMS and RDBMS remain pretty much the same, whereas MySQL receives frequent updates. SQL supports a single storage engine, but MySQL supports multiple storage engines and also plug-in storage engines. Thus, MySQL is more flexible.

4. What is the definition of normalization?

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.

Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is much easier to implement if that data is stored only in the Customers table and nowhere else in the database.

As with many formal rules and specifications, real world scenarios do not always allow for perfect compliance. In general, normalization requires additional tables and some customers find this cumbersome. If you decide to violate one of the first three rules of normalization, make sure that your application anticipates any problems that could occur, such as redundant data and inconsistent dependencies.

The following descriptions include examples.

**First normal form**

* Eliminate repeating groups in individual tables.
* Create a separate table for each set of related data.
* Identify each set of related data with a primary key.

Do not use multiple fields in a single table to store similar data. For example, to track an inventory item that may come from two possible sources, an inventory record may contain fields for Vendor Code 1 and Vendor Code 2.

What happens when you add a third vendor? Adding a field is not the answer; it requires program and table modifications and does not smoothly accommodate a dynamic number of vendors. Instead, place all vendor information in a separate table called Vendors, then link inventory to vendors with an item number key, or vendors to inventory with a vendor code key.

**Second normal form**

* Create separate tables for sets of values that apply to multiple records.
* Relate these tables with a foreign key.

Records should not depend on anything other than a table's primary key (a compound key, if necessary). For example, consider a customer's address in an accounting system. The address is needed by the Customers table, but also by the Orders, Shipping, Invoices, Accounts Receivable, and Collections tables. Instead of storing the customer's address as a separate entry in each of these tables, store it in one place, either in the Customers table or in a separate Addresses table.

**Third normal form**

* Eliminate fields that do not depend on the key.

Values in a record that are not part of that record's key do not belong in the table. In general, anytime the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table.

For example, in an Employee Recruitment table, a candidate's university name and address may be included. But you need a complete list of universities for group mailings. If university information is stored in the Candidates table, there is no way to list universities with no current candidates. Create a separate Universities table and link it to the Candidates table with a university code key.

EXCEPTION: Adhering to the third normal form, while theoretically desirable, is not always practical. If you have a Customers table and you want to eliminate all possible interfield dependencies, you must create separate tables for cities, ZIP codes, sales representatives, customer classes, and any other factor that may be duplicated in multiple records. In theory, normalization is worth pursing. However, many small tables may degrade performance or exceed open file and memory capacities.

It may be more feasible to apply third normal form only to data that changes frequently. If some dependent fields remain, design your application to require the user to verify all related fields when any one is changed.

**Other normalization forms**

Fourth normal form, also called Boyce Codd Normal Form (BCNF), and fifth normal form do exist, but are rarely considered in practical design. Disregarding these rules may result in less than perfect database design, but should not affect functionality.

**Normalizing an example table**

These steps demonstrate the process of normalizing a fictitious student table.

1. Unnormalized table:

| **Student#** | **Advisor** | **Adv-Room** | **Class1** | **Class2** | **Class3** |
| --- | --- | --- | --- | --- | --- |
| 1022 | Jones | 412 | 101-07 | 143-01 | 159-02 |
| 4123 | Smith | 216 | 101-07 | 143-01 | 179-04 |

1. First normal form: No repeating groups

Tables should have only two dimensions. Since one student has several classes, these classes should be listed in a separate table. Fields Class1, Class2, and Class3 in the above records are indications of design trouble.

Spreadsheets often use the third dimension, but tables should not. Another way to look at this problem is with a one-to-many relationship, do not put the one side and the many side in the same table. Instead, create another table in first normal form by eliminating the repeating group (Class#), as shown below:

| **Student#** | **Advisor** | **Adv-Room** | **Class#** |
| --- | --- | --- | --- |
| 1022 | Jones | 412 | 101-07 |
| 1022 | Jones | 412 | 143-01 |
| 1022 | Jones | 412 | 159-02 |
| 4123 | Smith | 216 | 101-07 |
| 4123 | Smith | 216 | 143-01 |
| 4123 | Smith | 216 | 179-04 |

1. Second normal form: Eliminate redundant data

Note the multiple Class# values for each Student# value in the above table. Class# is not functionally dependent on Student# (primary key), so this relationship is not in second normal form.

The following tables demonstrate second normal form:

Students:

| **Student#** | **Advisor** | **Adv-Room** |
| --- | --- | --- |
| 1022 | Jones | 412 |
| 4123 | Smith | 216 |

Registration:

| **Student#** | **Class#** |
| --- | --- |
| 1022 | 101-07 |
| 1022 | 143-01 |
| 1022 | 159-02 |
| 4123 | 101-07 |
| 4123 | 143-01 |
| 4123 | 179-04 |

1. Third normal form: Eliminate data not dependent on key

In the last example, Adv-Room (the advisor's office number) is functionally dependent on the Advisor attribute. The solution is to move that attribute from the Students table to the Faculty table, as shown below:

Students:

| **Student#** | **Advisor** |
| --- | --- |
| 1022 | Jones |
| 4123 | Smith |

Faculty:

| **Name** | **Room** | **Dept** |
| --- | --- | --- |
| Jones | 412 | 42 |
| Smith | 216 | 42 |

**BCNF:**

BCNF is the advance version of 3NF. It is stricter than 3NF.

* A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
* For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

**Example:** Let's assume there is a company where employees work in more than one department.

**EMPLOYEE table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** | **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| 264 | India | Designing | D394 | 283 |
| 264 | India | Testing | D394 | 300 |
| 364 | UK | Stores | D283 | 232 |
| 364 | UK | Developing | D283 | 549 |

**In the above table Functional dependencies are as follows:**

1. EMP\_ID  →  EMP\_COUNTRY
2. EMP\_DEPT  →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate key: {EMP-ID, EMP-DEPT}**

The table is not in BCNF because neither EMP\_DEPT nor EMP\_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

**EMP\_COUNTRY table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_COUNTRY** |
| 264 | India |
| 264 | India |

**EMP\_DEPT table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_DEPT** | **DEPT\_TYPE** | **EMP\_DEPT\_NO** |
| Designing | D394 | 283 |
| Testing | D394 | 300 |
| Stores | D283 | 232 |
| Developing | D283 | 549 |

**EMP\_DEPT\_MAPPING table:**

|  |  |
| --- | --- |
| **EMP\_ID** | **EMP\_DEPT** |
| D394 | 283 |
| D394 | 300 |
| D283 | 232 |
| D283 | 549 |

**Functional dependencies:**

1. EMP\_ID   →    EMP\_COUNTRY
2. EMP\_DEPT   →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate keys:**

**For the first table:** EMP\_ID  
**For the second table:** EMP\_DEPT  
**For the third table:** {EMP\_ID, EMP\_DEPT}