Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Mini Project
on
“ResearchProjects_DB”

[Code No: COMP232]
(For partial fulfillment of II Year/ II Semester in Computer Science)

Submitted by

Abhinav Adhikari(2)

Submitted to

Sanjog Sigdel
Lecturer
Department of Computer Science and Engineering

Submission Date:

January 1 2025

Problem/ Task

Make a 3NF table on the topic of “ResearchProjects_DB”. Also, show the use of SQL instruction you have learned.

1. Database Design

For us to make a database we need to first decide what kind of organization we are modelling. The topic research projects is vauge. Is it a organization that does research like Bells Labs, or does it publish research like JSTOR, or it funds research through grant like NIST.

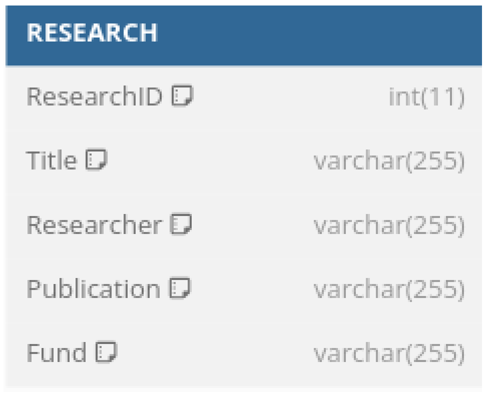
For our purpose we will choose a organizaiton that does research.

Now, what will we have to store. That depends on the type of organization. So, we will suppose a fictional organization and then make our data. Let's say in our organization, we empolyee reseracherss based on a field (i.e. area of study).

A research paper is published under the name the researchers. A single research paper may include multiple people from different fields that we hire for but it must be of one field (to know what kind a journal to publish the letter in), but any resracher isn't required to part of the field.

Suppose we release a math paper. It must be specified as a math paper; the authors can be from any group.

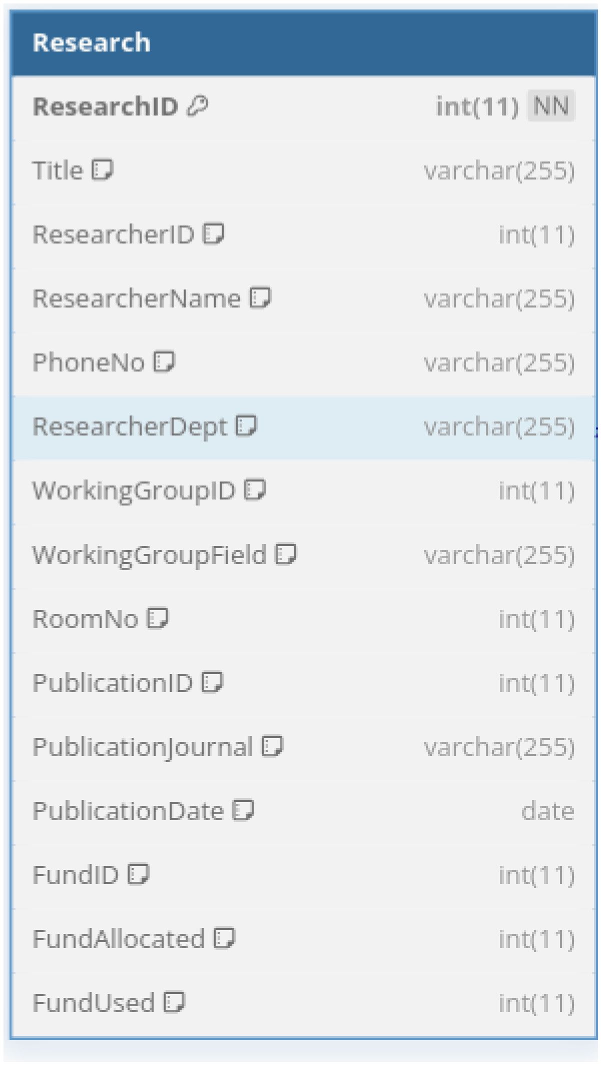
1.1 Unormalized Form



RESEARCH	
ResearchID	int(11)
Title	varchar(255)
Researcher	varchar(255)
Publication	varchar(255)
Fund	varchar(255)

In our unormalized table all the data are grouped into one table. ResearchID and Title are the only atomic values. Other all store all the necessary for the table.

1.2 First Normal Form(1NF)

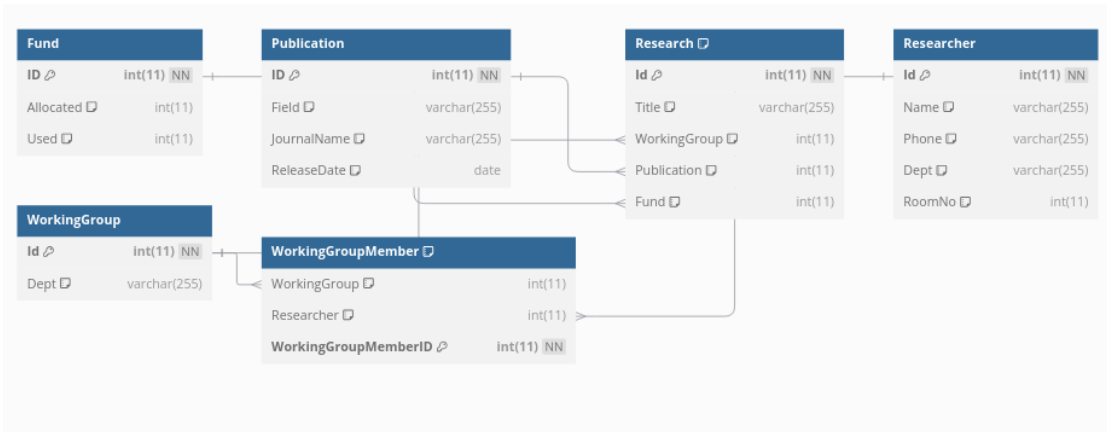


Research	
ResearchID	int(11) NN
Title	varchar(255)
ResearcherID	int(11)
ResearcherName	varchar(255)
PhoneNo	varchar(255)
ResearcherDept	varchar(255)
WorkingGroupID	int(11)
WorkingGroupField	varchar(255)
RoomNo	int(11)
PublicationID	int(11)
PublicationJournal	varchar(255)
PublicationDate	date
FundID	int(11)
FundAllocated	int(11)
FundUsed	int(11)

Now all the attribute are atomic. We are 1NF as no attribute domain has relations as elements. Now the table is in 1NF.

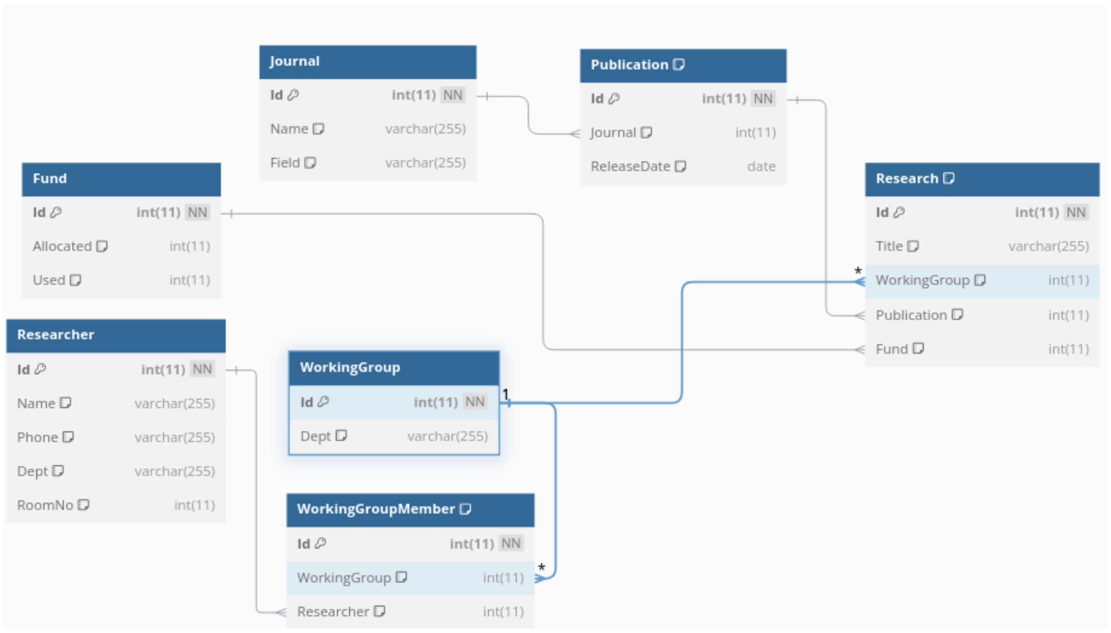
1.3 Second Normal Form(2NF)

For 2NF we divide the single table into mutiple table based on the candidate keys in the 1NF table such that, so that all the attribute of the realtion are dependent on the candidate of the table.

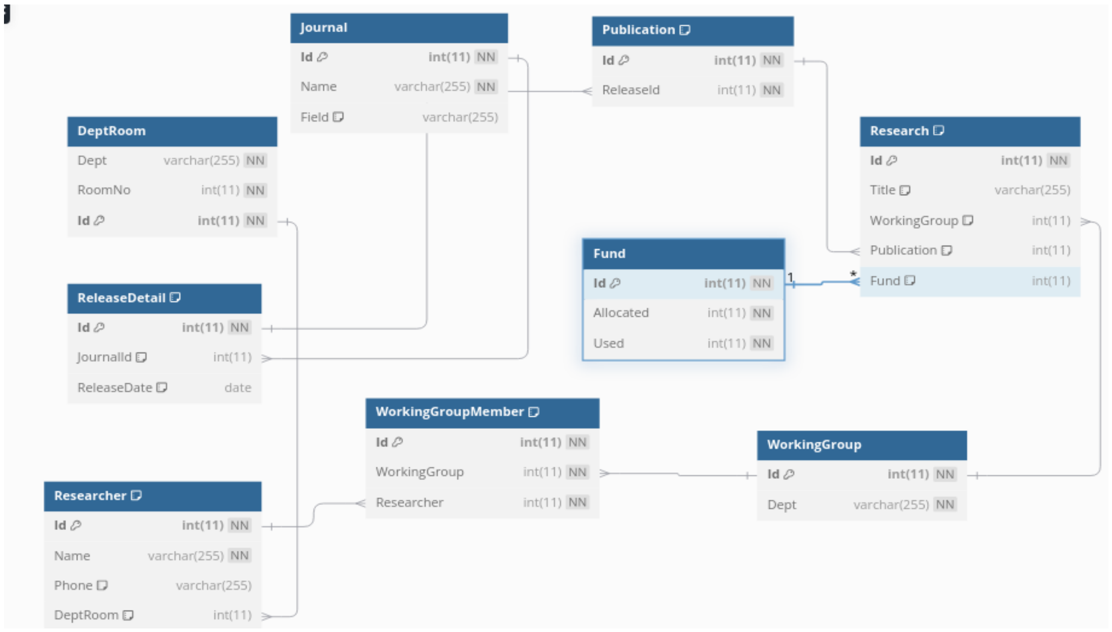


1.4 Third Normal Form(3NF)

To convert to 3NF we take all the instances of transitive dependency and make a seperate table for them all



1.5 Boyce-Codd Normal Form



A relational schema *R* is in Boyce–Codd normal form if and only if for every one of its functional dependencies *X* → *Y*, at least one of the following conditions hold:

- *X* → *Y* is a trivial functional dependency (Y ⊆ X),
- *X* is a superkey for schema *R*.

Since Room belongs to Dept is assigned dept we divide Dept and RoomNo

1.6 Implementating the Table

Now we make your Table in SQL. Your SQL for your construction is

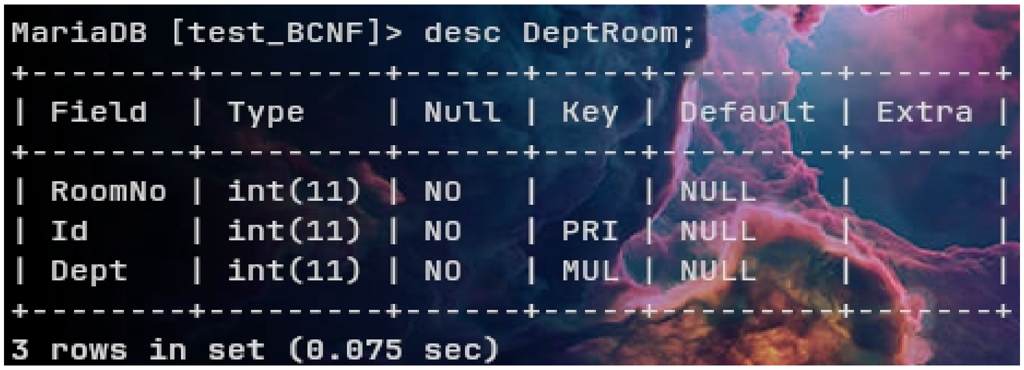
```
CREATE TABLE DeptRoom (

 RoomNo int(11) NOT NULL,

 Id int(11) NOT NULL,

 Dept int(11) NOT NULL,

 PRIMARY KEY (Id),

 KEY Dept (Dept),

 FOREIGN KEY (Dept) REFERENCES Department (Id)

);

CREATE TABLE DeptRoom (

 Dept varchar(255) NOT NULL,

 RoomNo int(11) NOT NULL,

 Id int(11) NOT NULL,

 PRIMARY KEY (Id)

);

CREATE TABLE Fund (
 Id int(11) NOT NULL,

 Allocated int(11) NOT NULL,

 Used int(11) NOT NULL,

 PRIMARY KEY (Id)

);

CREATE TABLE Journal (

 Id int(11) NOT NULL,

 Name varchar(255) NOT NULL,

 Field varchar(255) DEFAULT NULL,

 PRIMARY KEY (Id)

);

CREATE TABLE Publication (

 Id int(11) NOT NULL,

 ReleaseId int(11) NOT NULL,

 PRIMARY KEY (Id),

 KEY ReleaseId (ReleaseId),

 FOREIGN KEY (ReleaseId) REFERENCES ReleaseDetail (Id)

);
```

```
CREATE TABLE ReleaseDetail (

 Id int(11) NOT NULL,

 JournalId int(11) DEFAULT NULL,

 ReleaseDate date DEFAULT NULL,

 PRIMARY KEY (Id),

 KEY JournalId (JournalId),

 FOREIGN KEY (JournalId) REFERENCES Journal (Id)

);

CREATE TABLE Research (

 Id int(11) NOT NULL,

 Title varchar(255) DEFAULT NULL,

 WorkingGroup int(11) DEFAULT NULL,

 Publication int(11) DEFAULT NULL,

 Fund int(11) DEFAULT NULL,

 PRIMARY KEY (Id),

 KEY Publication (Publication),

 KEY WorkingGroup (WorkingGroup),

 KEY Fund (Fund),
 FOREIGN KEY (Publication) REFERENCES Publication (Id),

 FOREIGN KEY (WorkingGroup) REFERENCES WorkingGroup (Id),

 FOREIGN KEY (Fund) REFERENCES Fund (Id)

);

CREATE TABLE WorkingGroup (

 Id int(11) NOT NULL,

 Dept varchar(255) NOT NULL,

 PRIMARY KEY (Id)

);

CREATE TABLE WorkingGroupMember (

 Id int(11) NOT NULL,

 WorkingGroup int(11) NOT NULL,

 Researcher int(11) NOT NULL,

 PRIMARY KEY (Id),

 KEY WorkingGroup (WorkingGroup),

 KEY Researcher (Researcher),

 FOREIGN KEY (WorkingGroup) REFERENCES WorkingGroup (Id),

 FOREIGN KEY (Researcher) REFERENCES Researcher (Id)
```

```
);
```

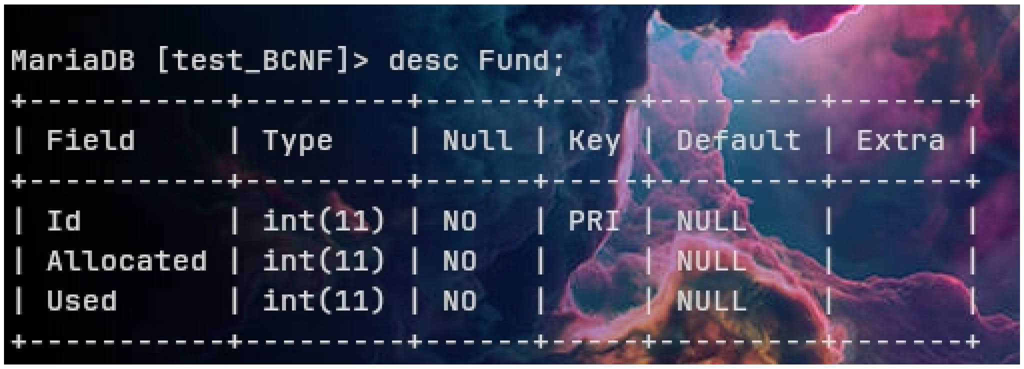
This will result in following table. We can see the description for a table by using the **desc** command. So,

```
desc DeptRoom;
```



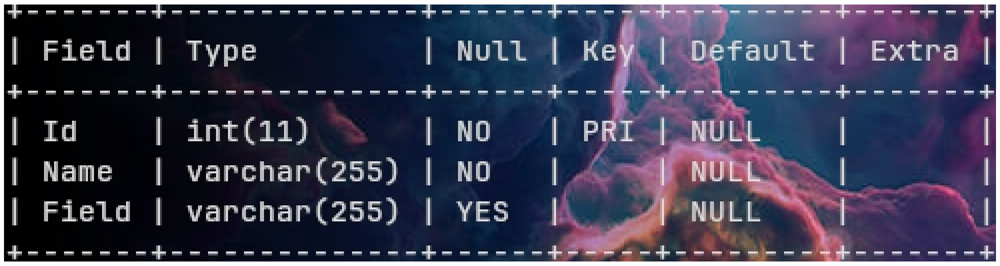
```
MariaDB [test_BCNF]> desc DeptRoom;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| RoomNo | int(11) | NO   |     | NULL    |       |
| Id     | int(11) | NO   | PRI | NULL    |       |
| Dept   | int(11) | NO   | MUL | NULL    |       |
+--------+---------+------+-----+---------+-------+
3 rows in set (0.075 sec)
```

```
desc Fund;
```



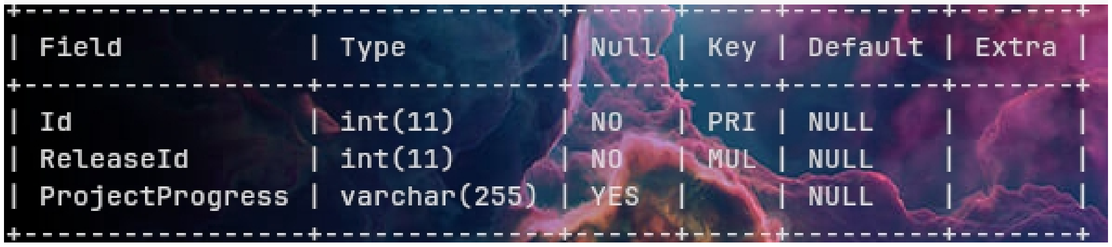
```
MariaDB [test_BCNF]> desc Fund;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| Id        | int(11) | NO   | PRI | NULL    |       |
| Allocated | int(11) | NO   |     | NULL    |       |
| Used      | int(11) | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
```

```
desc Journal;
```



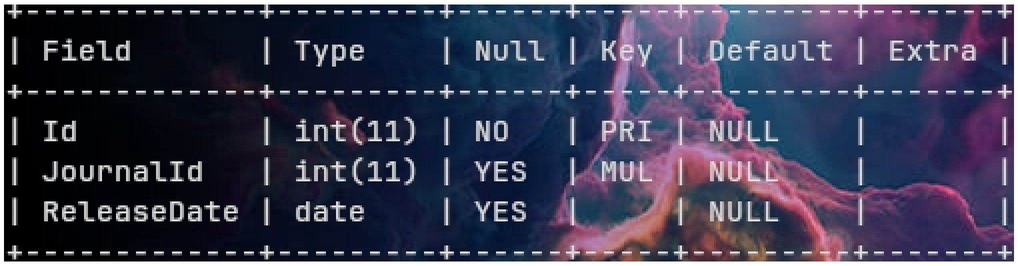
```
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| Id    | int(11)      | NO   | PRI | NULL    |       |
| Name  | varchar(255) | NO   |     | NULL    |       |
| Field | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```

desc Publication;



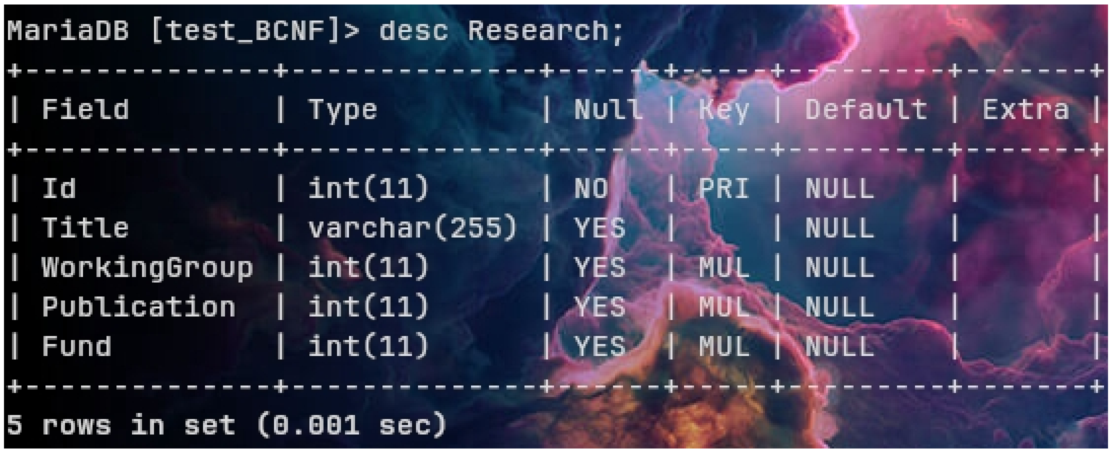
```
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| Id              | int(11)      | NO   | PRI | NULL    |       |
| ReleaseId       | int(11)      | NO   | MUL | NULL    |       |
| ProjectProgress | varchar(255) | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
```

desc ReleaseDetail;



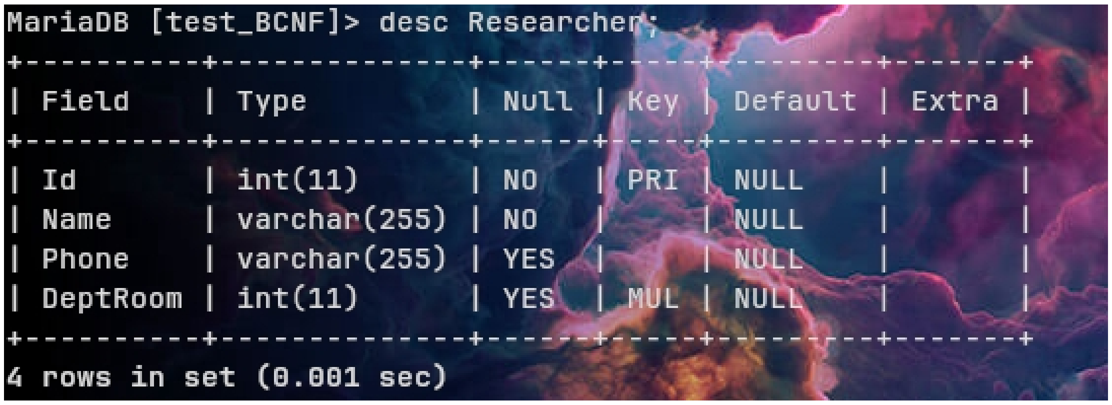
```
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| Id          | int(11) | NO   | PRI | NULL    |       |
| JournalId   | int(11) | YES  | MUL | NULL    |       |
| ReleaseDate | date    | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
```

desc Research;



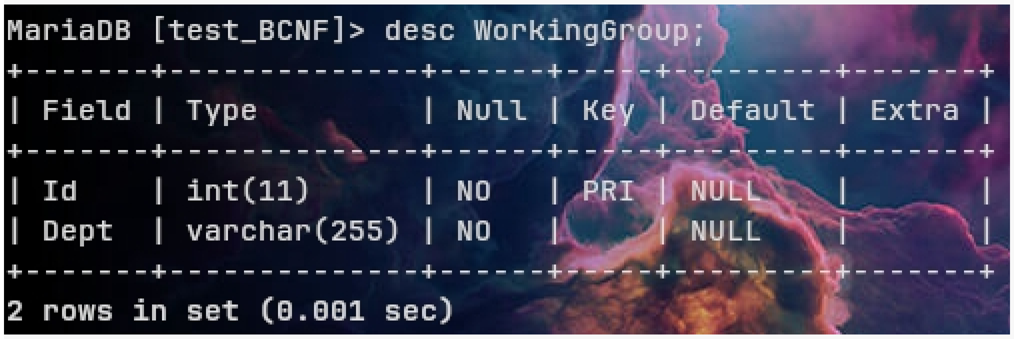
```
MariaDB [test_BCNF]> desc Research;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| Id           | int(11)      | NO   | PRI | NULL    |       |
| Title        | varchar(255) | YES  |     | NULL    |       |
| WorkingGroup | int(11)      | YES  | MUL | NULL    |       |
| Publication  | int(11)      | YES  | MUL | NULL    |       |
| Fund         | int(11)      | YES  | MUL | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
5 rows in set (0.001 sec)
```

```
desc Researcher;
```



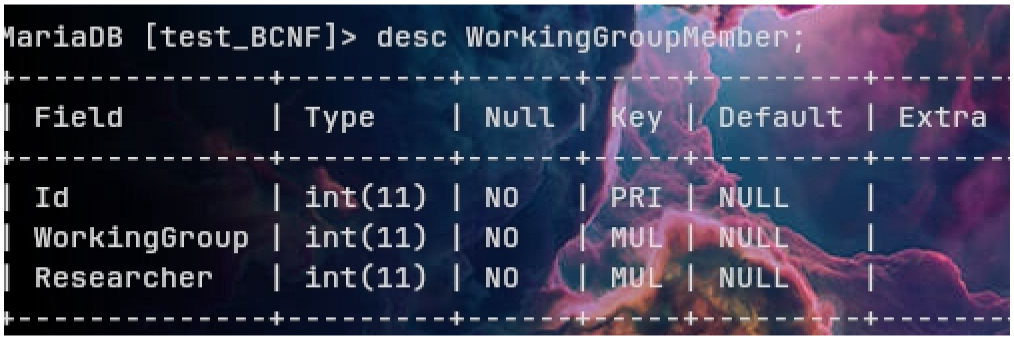
```
MariaDB [test_BCNF]> desc Researcher;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| Id       | int(11)      | NO   | PRI | NULL    |       |
| Name     | varchar(255) | NO   |     | NULL    |       |
| Phone    | varchar(255) | YES  |     | NULL    |       |
| DeptRoom | int(11)      | YES  | MUL | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.001 sec)
```

```
desc WorkingGroup;
```



```
MariaDB [test_BCNF]> desc WorkingGroup;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| Id    | int(11)      | NO   | PRI | NULL    |       |
| Dept  | varchar(255) | NO   |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
2 rows in set (0.001 sec)
```

```
desc WorkingGroupMember;
```



```
MariaDB [test_BCNF]> desc WorkingGroupMember;
+--------------+---------+------+-----+---------+-------
| Field        | Type    | Null | Key | Default | Extra
+--------------+---------+------+-----+---------+-------
| Id           | int(11) | NO   | PRI | NULL    |
| WorkingGroup | int(11) | NO   | MUL | NULL    |
| Researcher   | int(11) | NO   | MUL | NULL    |
+--------------+---------+------+-----+---------+-------
```

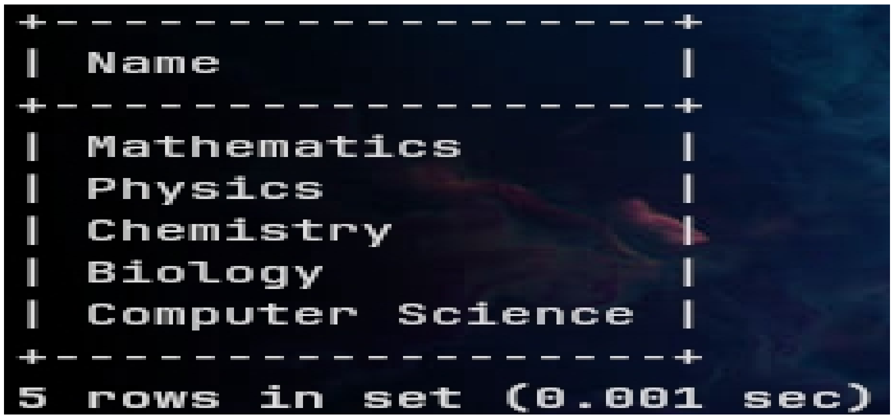
2. Basic SQL Commands

2.1 SELECT

Let us consider that we want the name of every researcher then we can do

```
SELECT Name FROM Department;
```

The result is a relation consisting of a single attribute with the heading *name*.

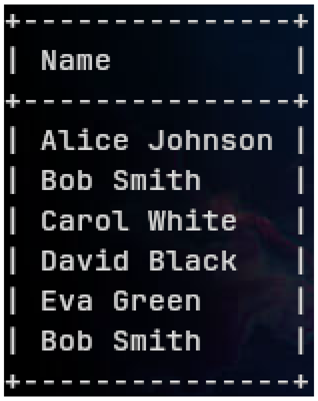


```
+--------------------+
| Name               |
+--------------------+
| Mathematics        |
| Physics            |
| Chemistry          |
| Biology            |
| Computer Science   |
+--------------------+
5 rows in set (0.001 sec)
```

Now consider another query, “Find the department names of all instructors” which can be written as

The **SELECT** command doesn't do duplicate elimination. In case of you need to eliminate duplicate you can use the **DISTINCT** command.

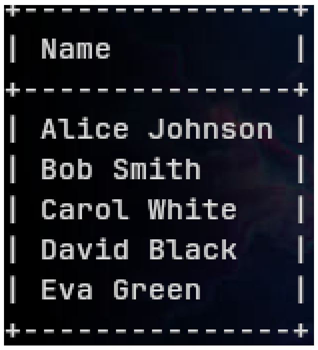
SELECT Name **From** Researcher;



```
+---------------+
| Name          |
+---------------+
| Alice Johnson |
| Bob Smith     |
| Carol White   |
| David Black   |
| Eva Green     |
| Bob Smith     |
+---------------+
```

gives this result. Notice how thier are two Bob Smith. This is because we thier are two people named Bob Simth. But if you don't want duplicates you can do

SELECT DISTINCT Name **From** Researcher;



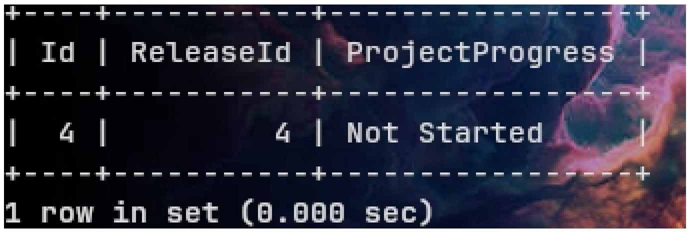
```
+---------------+
| Name          |
+---------------+
| Alice Johnson |
| Bob Smith     |
| Carol White   |
| David Black   |
| Eva Green     |
+---------------+
```

Notice no duplication.

If a command does do duplicate exclusion then you can remove it by using the **ALL** command instead of **DISTINCT**.

You can also make condition using the **WHERE** command like this.

```
SELECT * FROM Publication WHERE ProjectProgress="Not Started";
```

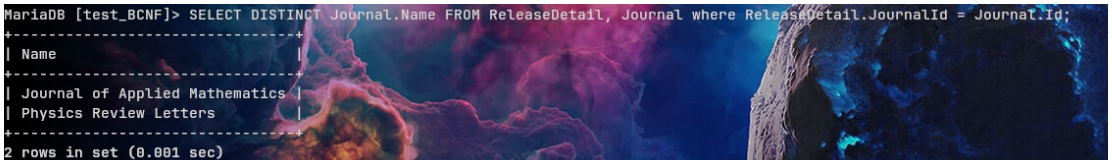


```
+----+-----------+-----------------+
| Id | ReleaseId | ProjectProgress |
+----+-----------+-----------------+
|  4 |         4 | Not Started     |
+----+-----------+-----------------+
1 row in set (0.000 sec)
```

* represents all column in the table.

We can all chain multiple table in a single **SELECT** table like this

```
SELECT DISTINCT Journal.Name FROM ReleaseDetail, Journal where ReleaseDetail.JournalId = Journal.Id;
```



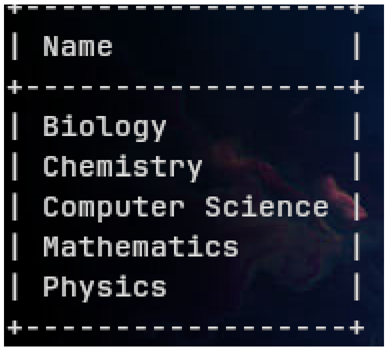
```
MariaDB [test_BCNF]> SELECT DISTINCT Journal.Name FROM ReleaseDetail, Journal where ReleaseDetail.JournalId = Journal.Id;
+--------------------------------+
| Name                           |
+--------------------------------+
| Journal of Applied Mathematics |
| Physics Review Letters         |
+--------------------------------+
2 rows in set (0.001 sec)
```

We can rename the table name temporaliy for this purpose using **AS** command. The above statement is equal to

```
SELECT DISTINCT J.Name FROM ReleaseDetail As R, Journal As J where R.JournalId = J.Id;
```

We can order the list by using the **ORDER BY** command.

```
SELECT Name FROM Department ORDER BY Name;
```

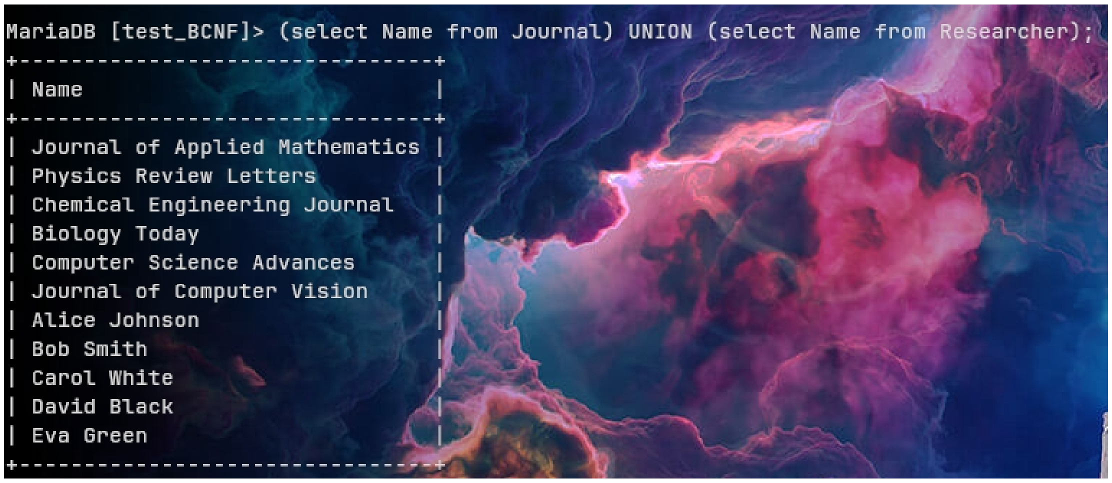


```
+-------------------+
| Name              |
+-------------------+
| Biology           |
| Chemistry         |
| Computer Science  |
| Mathematics       |
| Physics           |
+-------------------+
```

2.2 Set Operations

The SQL operations union, intersect, and except operate on relations and correspond to the mathematical set operations ∪(**UNION**), ∩(**INTERSECT**), and −(**EXCEPT**).

Set operations are duplication elimianting.

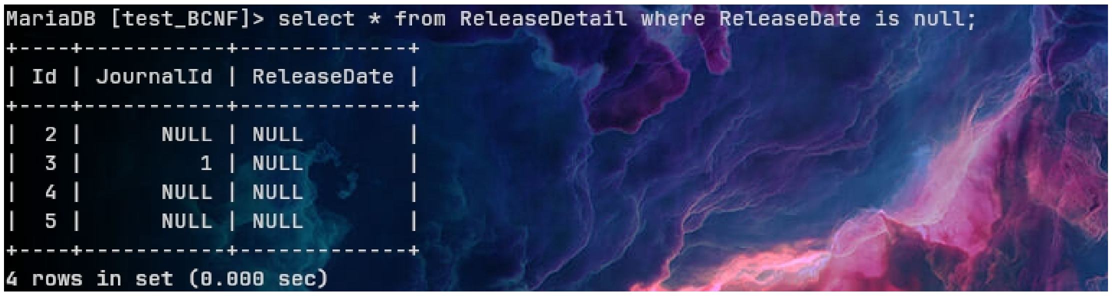


```
MariaDB [test_BCNF]> (select Name from Journal) UNION (select Name from Researcher);
+--------------------------------+
| Name                           |
+--------------------------------+
| Journal of Applied Mathematics |
| Physics Review Letters         |
| Chemical Engineering Journal   |
| Biology Today                  |
| Computer Science Advances      |
| Journal of Computer Vision     |
| Alice Johnson                  |
| Bob Smith                      |
| Carol White                    |
| David Black                    |
| Eva Green                      |
+--------------------------------+
```

To remove Duplication use **UNION ALL**

2.3 Null Values

Null values represent a lack of value. We known whether a value of null or not using the **IS NULL** and **IS NOT NULL** command.



```
MariaDB [test_BCNF]> select * from ReleaseDetail where ReleaseDate is null;
+----+-----------+-------------+
| Id | JournalId | ReleaseDate |
+----+-----------+-------------+
|  2 |      NULL | NULL        |
|  3 |         1 | NULL        |
|  4 |      NULL | NULL        |
|  5 |      NULL | NULL        |
+----+-----------+-------------+
4 rows in set (0.000 sec)
```

2.4 Aggregate Function

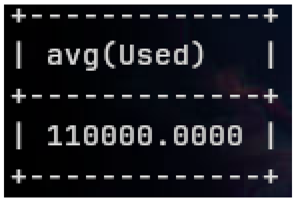
Aggregate functions are functions that take a collection (a set or multiset) of values as a input and return a single value.

We have the following five standard built-in aggregate functions:

- Average: **avg**
- Minimum: **min**
- Maximum: **max**
- Total: **sum**
- Count: **count**

So we can do

```sql
SELECT avg(Used) From Fund;
```



```
+-------------+
| avg(Used)   |
+-------------+
| 110000.0000 |
+-------------+
```

2.5 Join Expressions

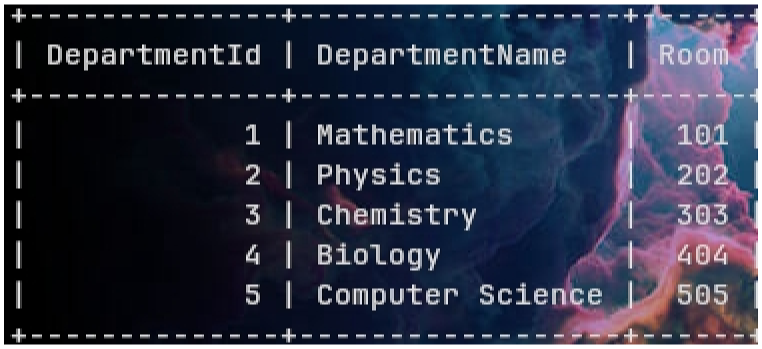
JOIN syntax for the table_references part of **SELECT** statements and multiple-table **DELETE** and **UPDATE** statements:

It takes a two table an merges them together. Join statements should be used with caution as they check all the required row in a table and may check the left table in (**LEFT JOIN**), right table in (**RIGHT JOIN**) and all the table in (other **JOIN** statements).

It is to be noted that **JOIN**, **NATURAL JOIN**, **INNER JOIN** and **OUTER JOIN** are syntactically equivalent in MySQL.

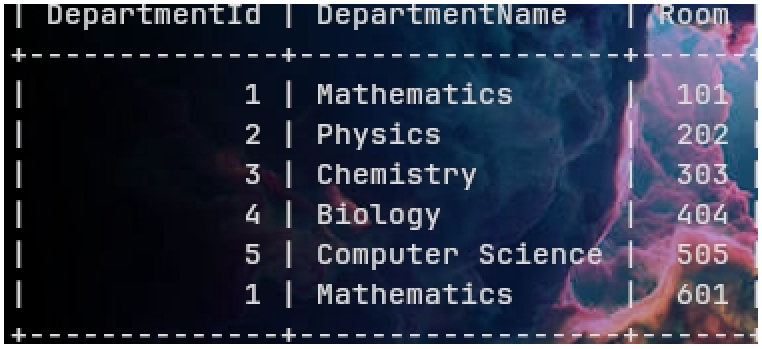
So, we want to list all the department and their room we can do:

SELECT DISTINCT Department.ID **As** DepartmentId, Department.Name **As** DepartmentName, DeptRoom.RoomNo **As** Room **FROM** Department **INNER JOIN** DeptRoom **ON** Department.Id = DeptRoom.Dept;



```
+--------------+-------------------+------
| DepartmentId | DepartmentName    | Room
+--------------+-------------------+------
|            1 | Mathematics       |  101
|            2 | Physics           |  202
|            3 | Chemistry         |  303
|            4 | Biology           |  404
|            5 | Computer Science  |  505
+--------------+-------------------+------
```

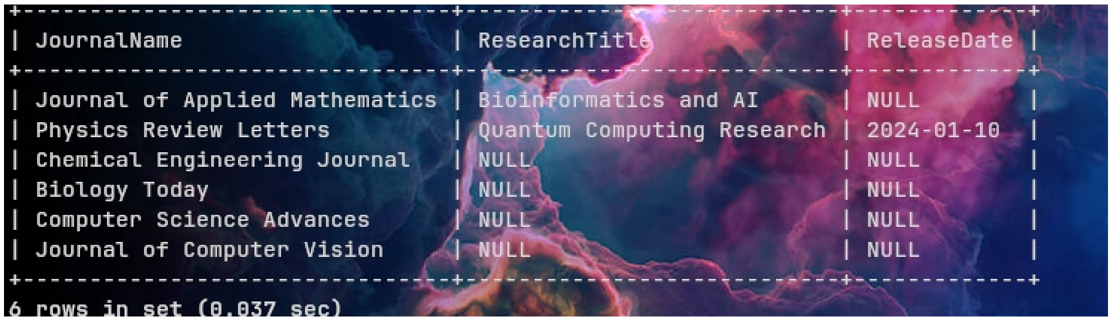
It should be noted that **JOIN** don’t exclude duplicity by default.



```
| DepartmentId | DepartmentName   | Room
+--------------+------------------+------
|            1 | Mathematics      |  101 |
|            2 | Physics          |  202 |
|            3 | Chemistry        |  303 |
|            4 | Biology          |  404 |
|            5 | Computer Science |  505 |
|            1 | Mathematics      |  601 |
+--------------+------------------+------
```

We can also do fancy things like list research projects associated with a journal and release details:

```
SELECT j.Name AS JournalName, r.Title AS ResearchTitle, rd.ReleaseDate FROM Journal j LEFT JOIN ReleaseDetail rd ON j.Id = rd.JournalId LEFT JOIN Publication p ON rd.Id = p.ReleaseId LEFT JOIN Research r ON p.Id = r.Publication;
```



```
+---------------------------------+----------------------------+-------------+
| JournalName                     | ResearchTitle              | ReleaseDate |
+---------------------------------+----------------------------+-------------+
| Journal of Applied Mathematics | Bioinformatics and AI      | NULL        |
| Physics Review Letters          | Quantum Computing Research | 2024-01-10  |
| Chemical Engineering Journal    | NULL                       | NULL        |
| Biology Today                   | NULL                       | NULL        |
| Computer Science Advances       | NULL                       | NULL        |
| Journal of Computer Vision      | NULL                       | NULL        |
+---------------------------------+----------------------------+-------------+
6 rows in set (0.037 sec)
```

3 Transcation

We need to make some operation atomic. In our case, suppose that a researcher has been determined to be better in another department. Since a reseracher can belong to only one department, we need to remove him from his original department and then add to the new department. This posits us with a challenge. If we remove the user from a Department and the system crashes he is in no Department. So, we need to make it so that we do the operation in full or the operation never occurs.

This can be done using tarnscations.

```
DELIMITER ;;

CREATE DEFINER=`specialist`@`localhost` PROCEDURE `ChangeResearcherDepartment`(IN researcher_id INT, IN new_room INT, IN new_department_name VARCHAR(255))

BEGIN
```

```
DECLARE deptroom_id INT;

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN
```

```
ROLLBACK;

SHOW ERRORS;

END;


DECLARE EXIT HANDLER FOR NOT FOUND

BEGIN

ROLLBACK;

SHOW ERRORS;

END;


START TRANSACTION;


SELECT DeptRoom As deptroom_id FROM Researcher WHERE Id =
researcher_id;

UPDATE DeptRoom SET RoomNo = new_room where Id = deptroom_id;

UPDATE DeptRoom SET Dept = new_department_name where Id =
deptroom_id;
```

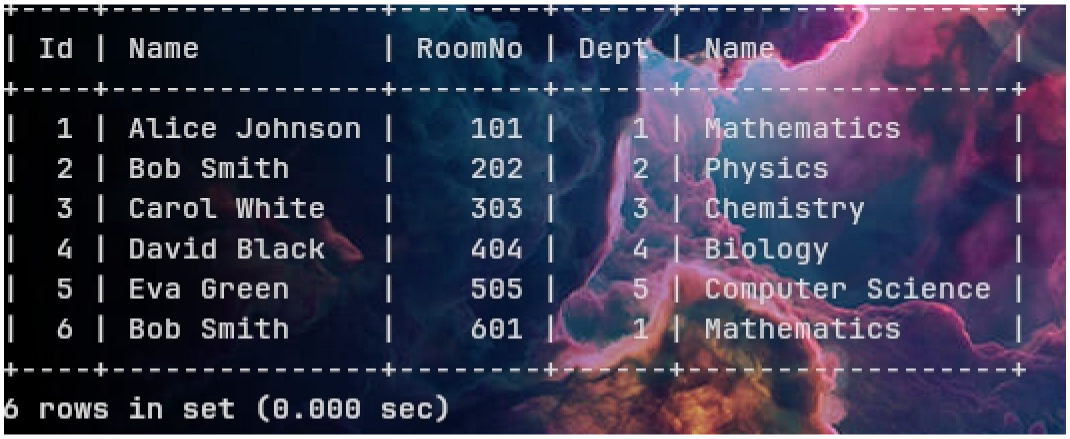
```
COMMIT;


SELECT 'Researcher department updated successfully.' AS message;

END ;;

DELIMITER ;
```

This will cause the transcation to be not continued if thier is any error. If we call the function

```
CALL ChangeResearcherDepartment(1, 526, 5);
```

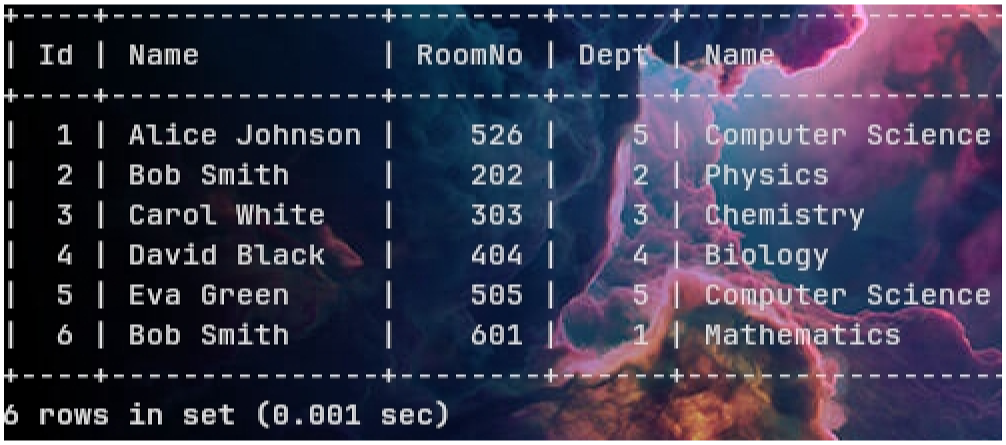
We can check for the result using

```
Select R.Id, R.Name, Dr.RoomNo, Dr.Dept, Department.Name FROM Researcher R JOIN DeptRoom Dr ON R.DeptRoom = Dr.Id LEFT JOIN Department ON Dr.Dept = Department.Id;
```



```
+----+---------------+--------+------+-------------------+
| Id | Name          | RoomNo | Dept | Name              |
+----+---------------+--------+------+-------------------+
|  1 | Alice Johnson |    101 |    1 | Mathematics       |
|  2 | Bob Smith     |    202 |    2 | Physics           |
|  3 | Carol White   |    303 |    3 | Chemistry         |
|  4 | David Black   |    404 |    4 | Biology           |
|  5 | Eva Green     |    505 |    5 | Computer Science  |
|  6 | Bob Smith     |    601 |    1 | Mathematics       |
+----+---------------+--------+------+-------------------+
6 rows in set (0.000 sec)
```

turns into this



```
+----+---------------+--------+------+------------------
| Id | Name          | RoomNo | Dept | Name
+----+---------------+--------+------+------------------
|  1 | Alice Johnson |    526 |    5 | Computer Science
|  2 | Bob Smith     |    202 |    2 | Physics
|  3 | Carol White   |    303 |    3 | Chemistry
|  4 | David Black   |    404 |    4 | Biology
|  5 | Eva Green     |    505 |    5 | Computer Science
|  6 | Bob Smith     |    601 |    1 | Mathematics
+----+---------------+--------+------+------------------
6 rows in set (0.001 sec)
```

4. Dump File

The dump file is at