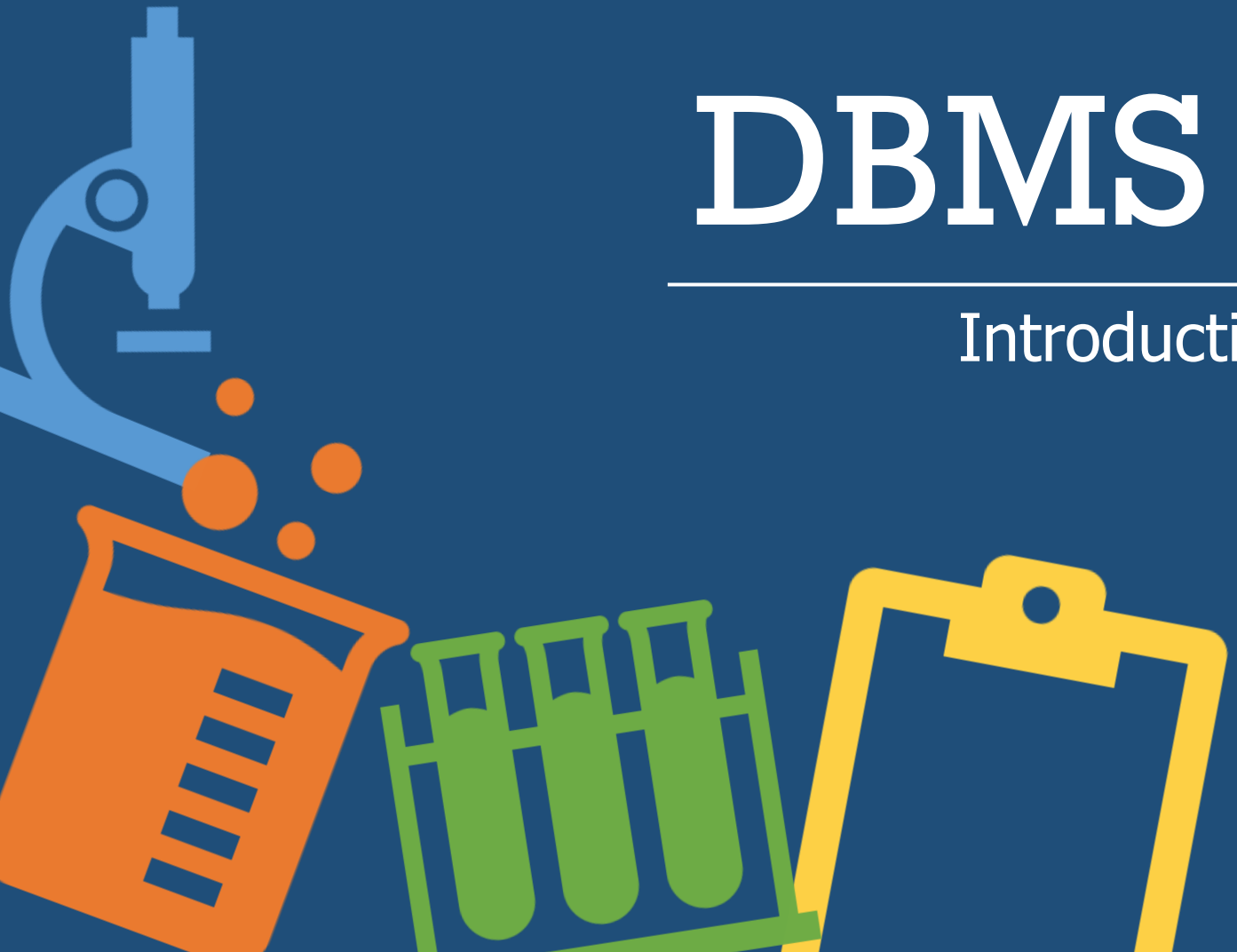
DBMS Lab

Introduction



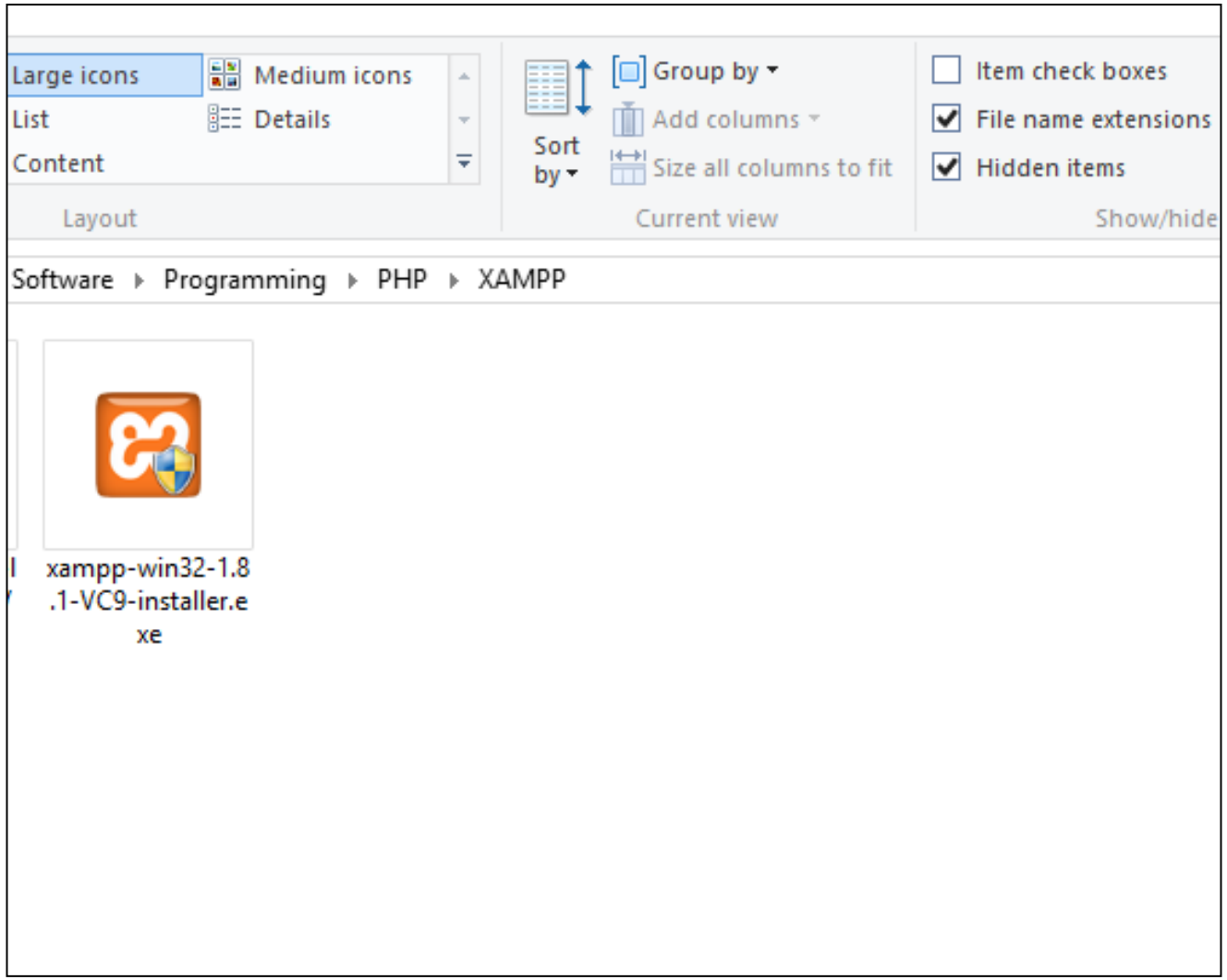
Things we will complete today

1. Database Software Installation
2. Loading a Sample database
3. Run basic SQL query on loaded database
4. Learnabout some additional tools

Installation

1. Database Software Installation

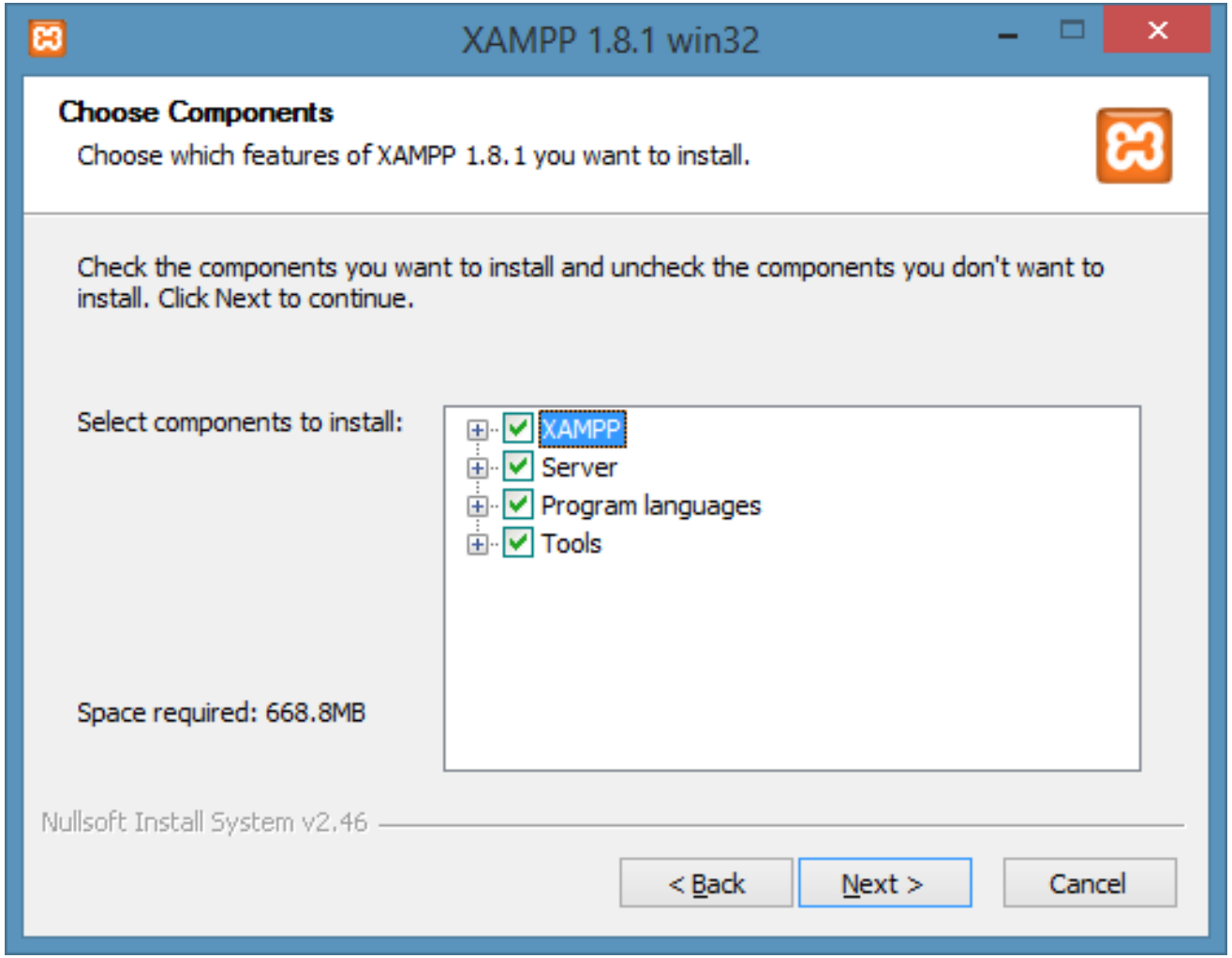
1. Database Software Installation



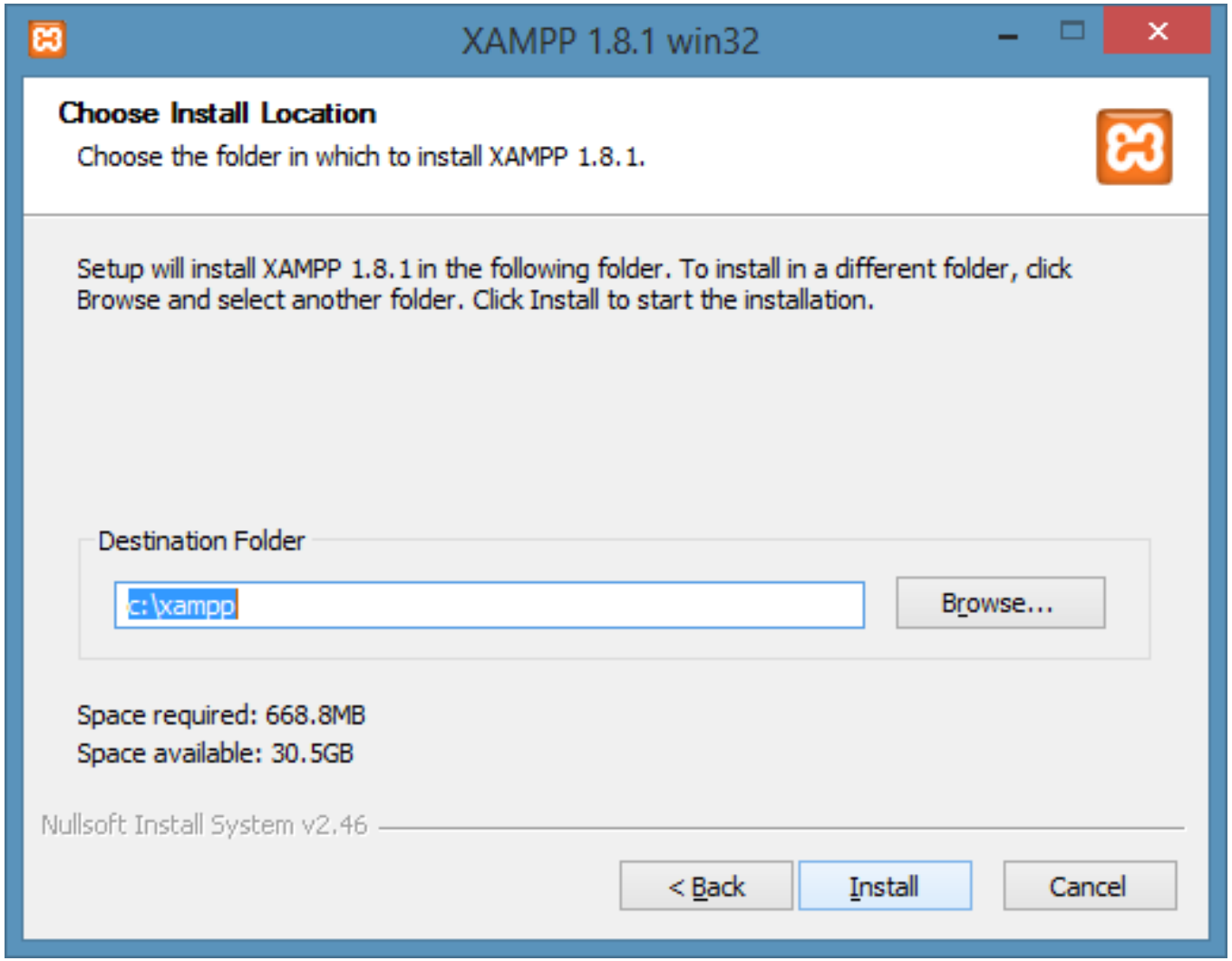
1. Database Software Installation

XAMPP 1.8.1 win32

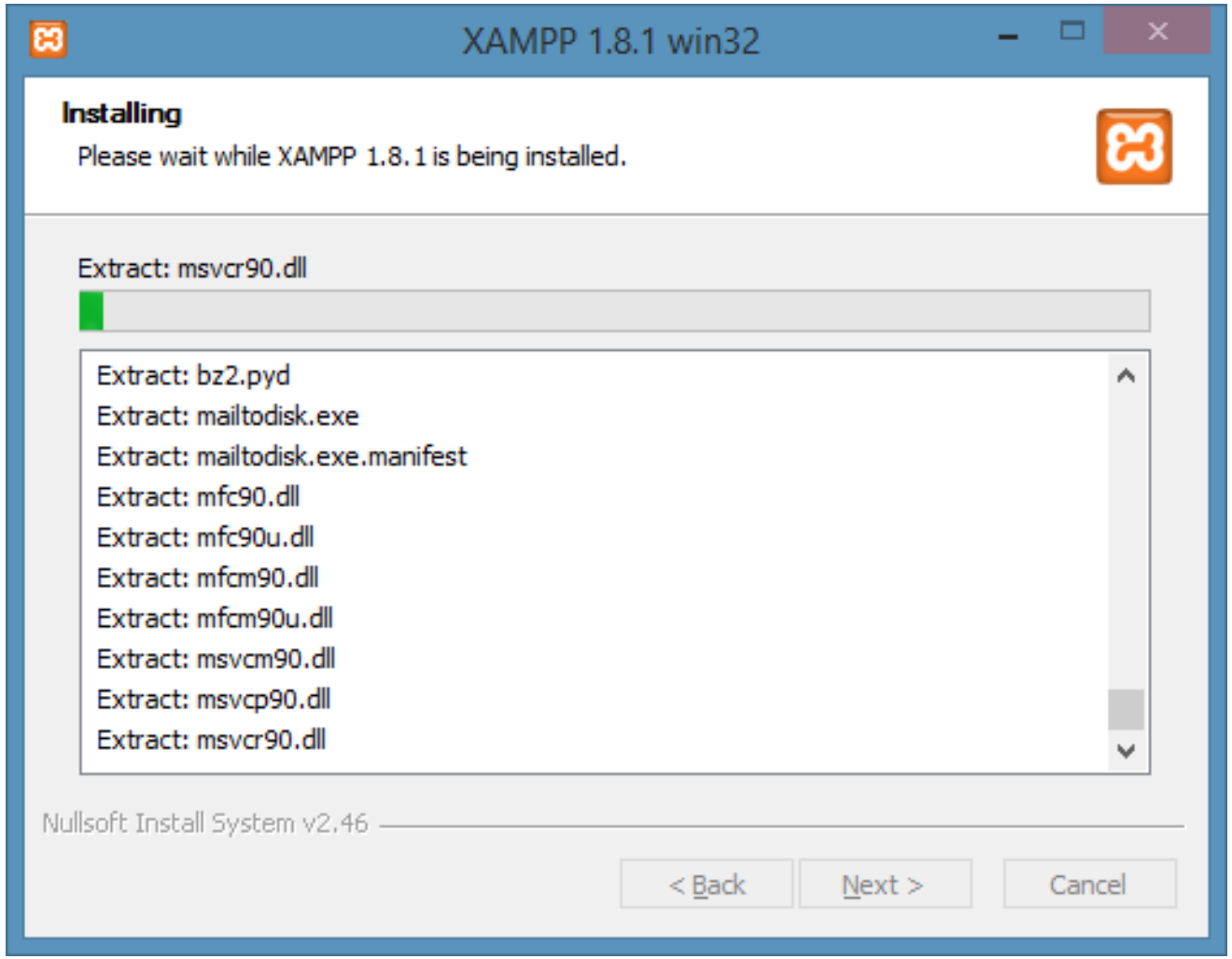
1. Database Software Installation



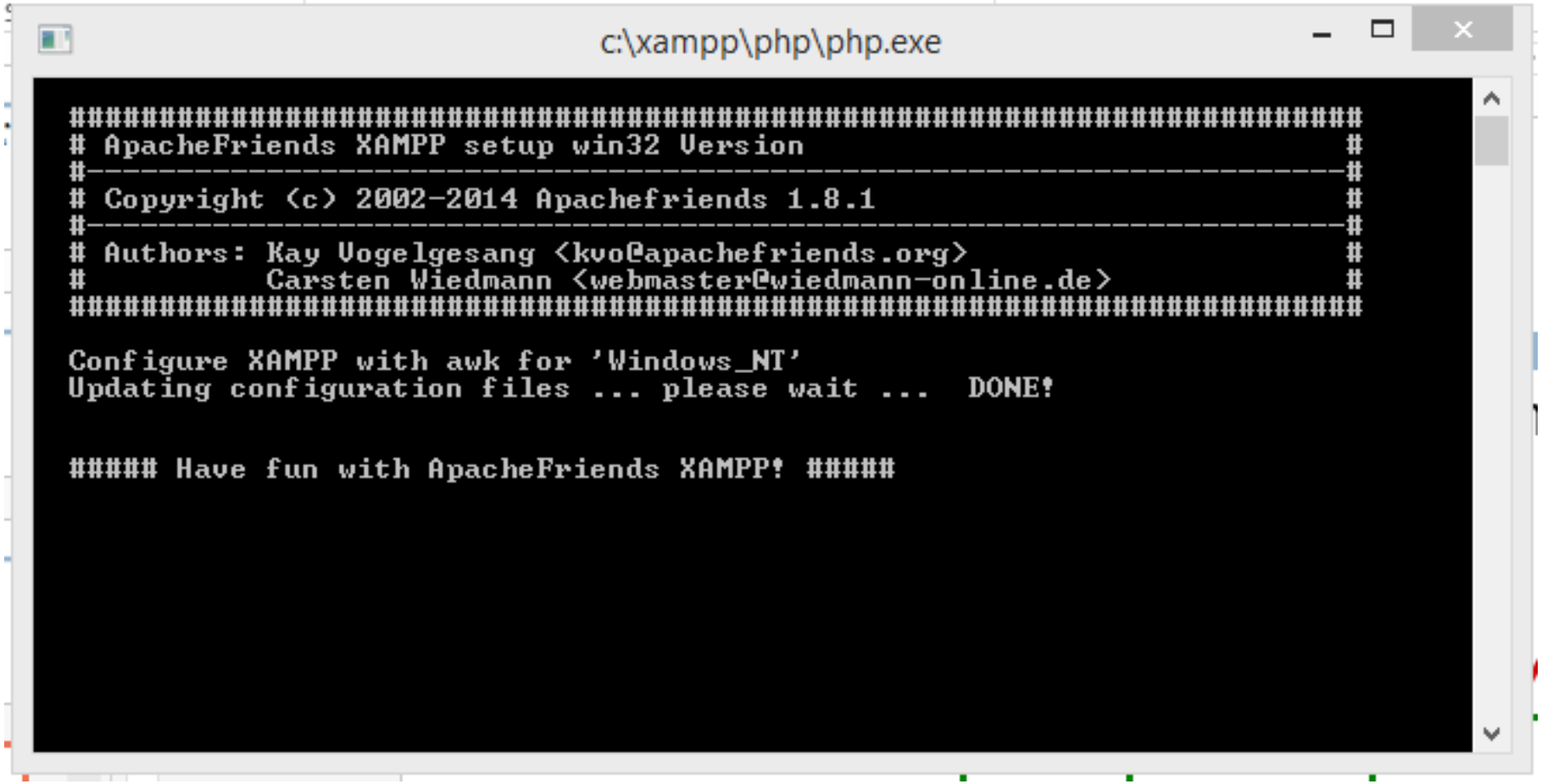
1. Database Software Installation



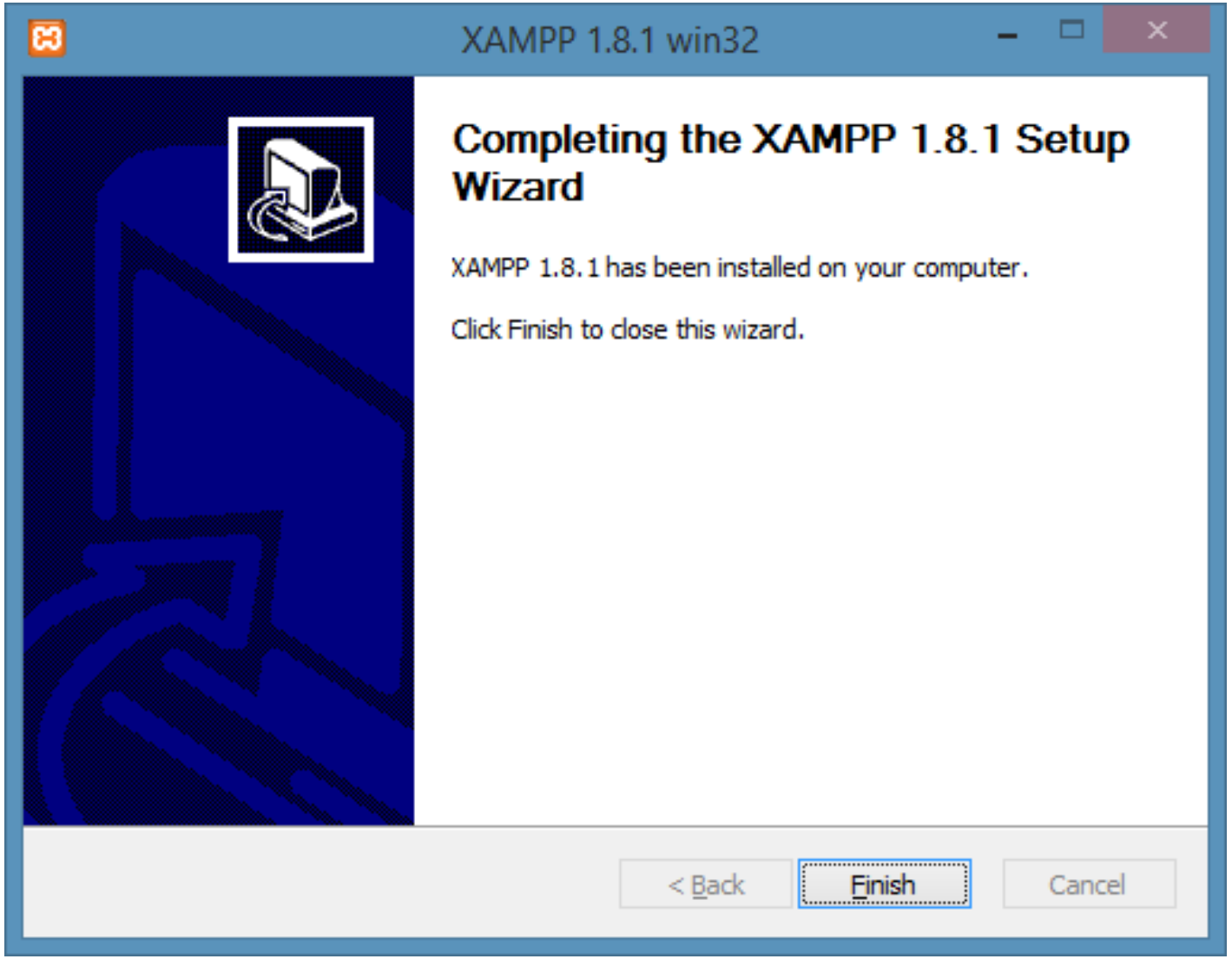
1. Database Software Installation



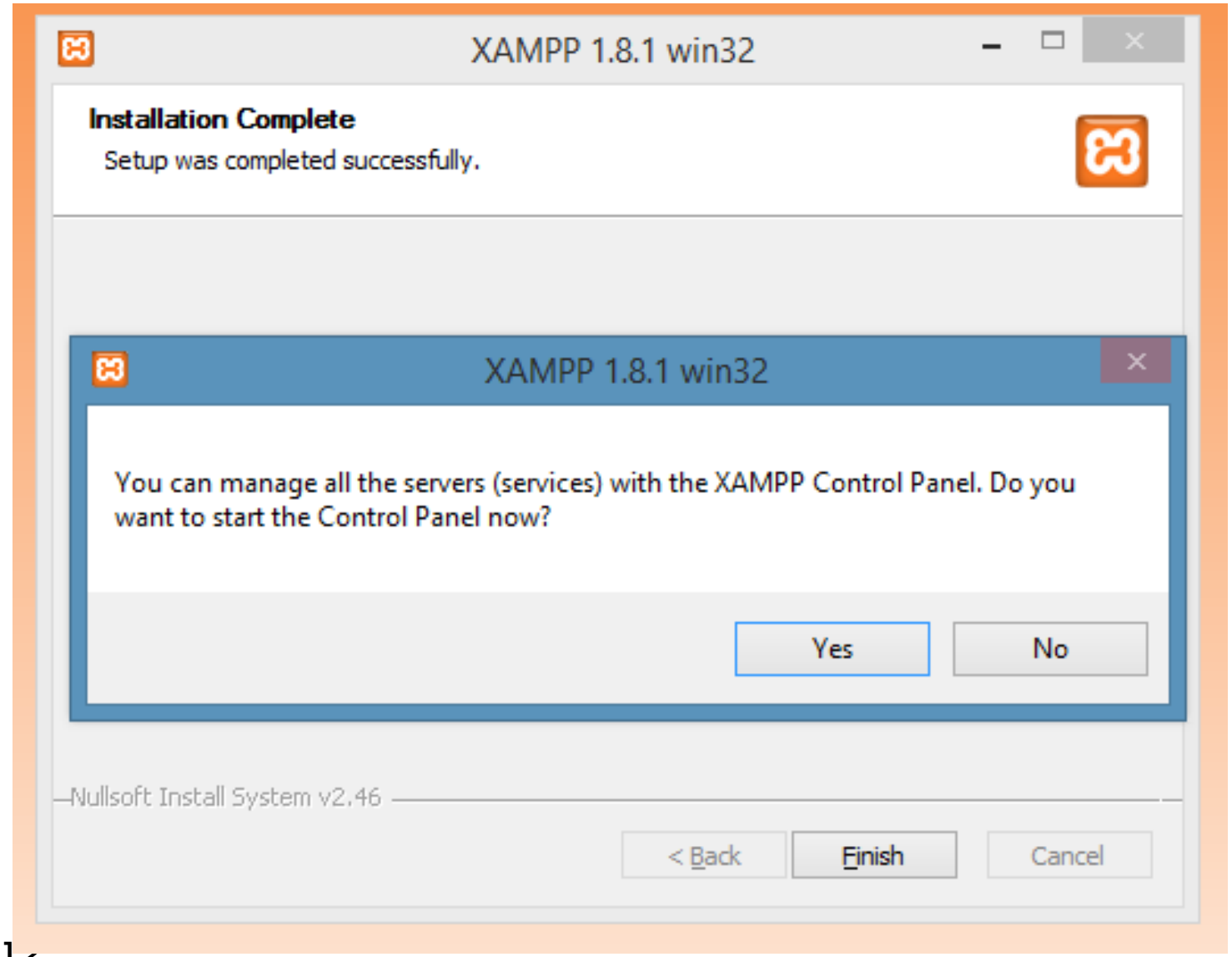
1. Database Software Installation



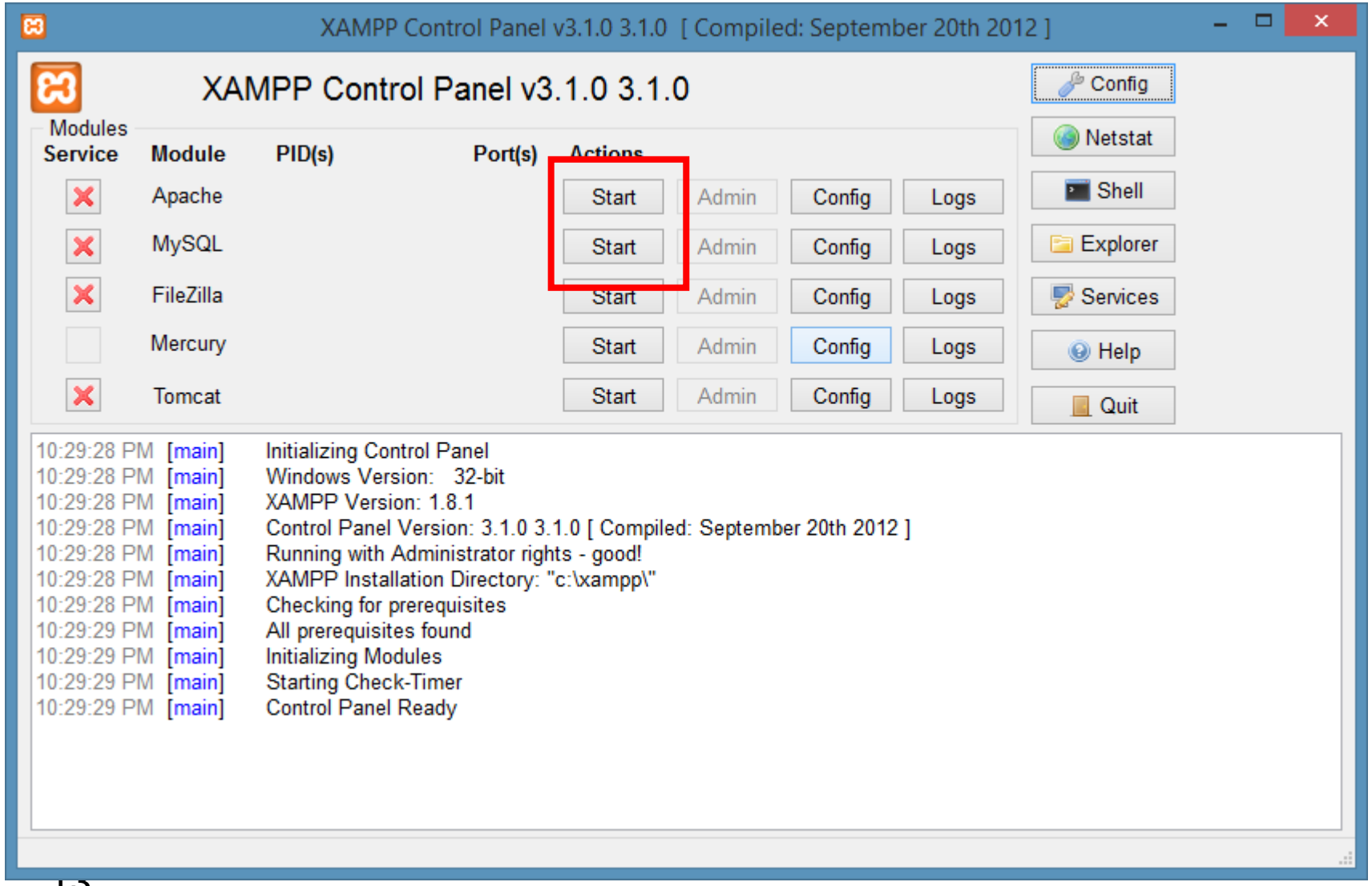
1. Database Software Installation



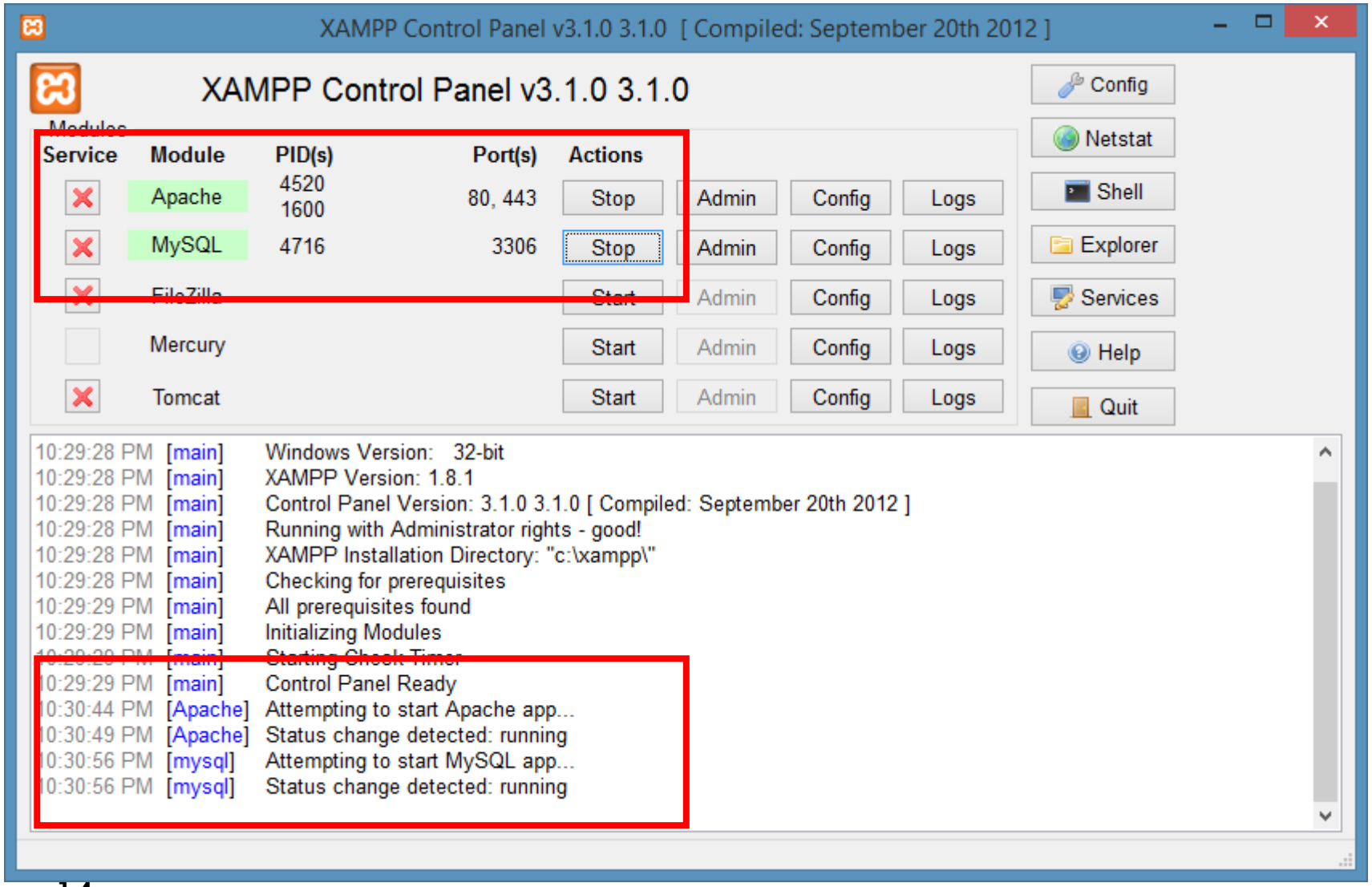
1. Database Software Installation



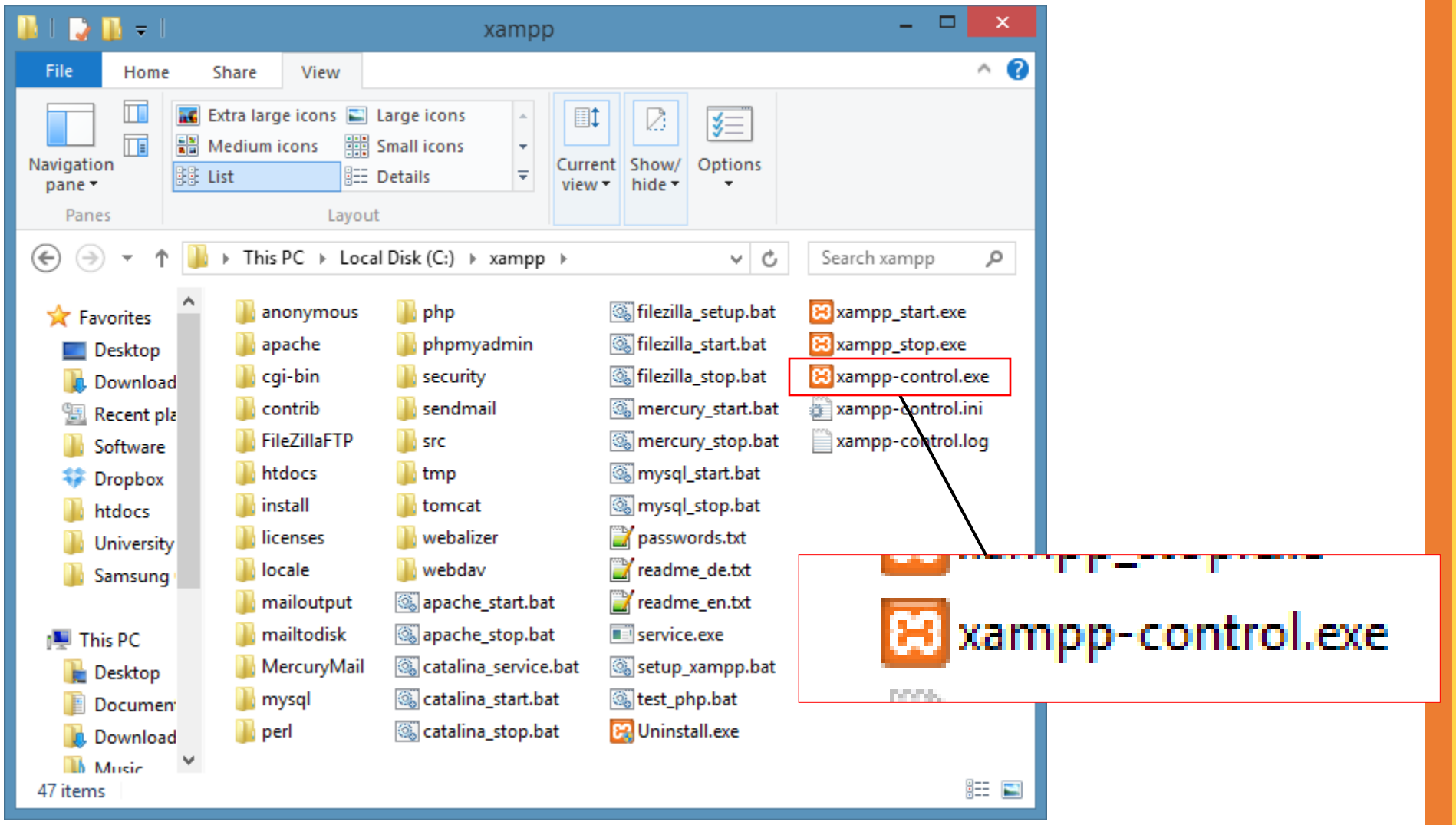
1. Database Software Installation



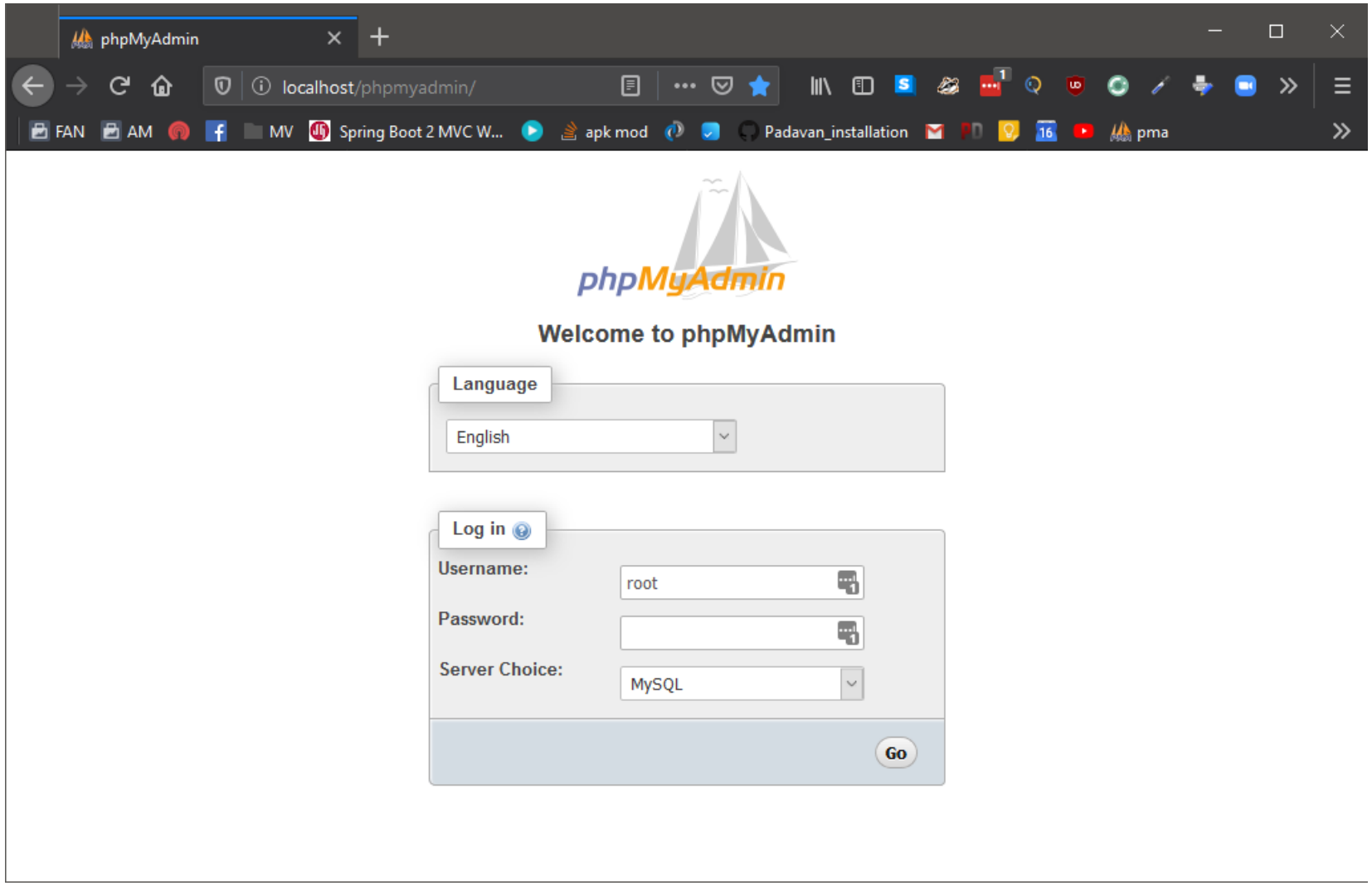
1. Database Software Installation



1. Database Software Installation

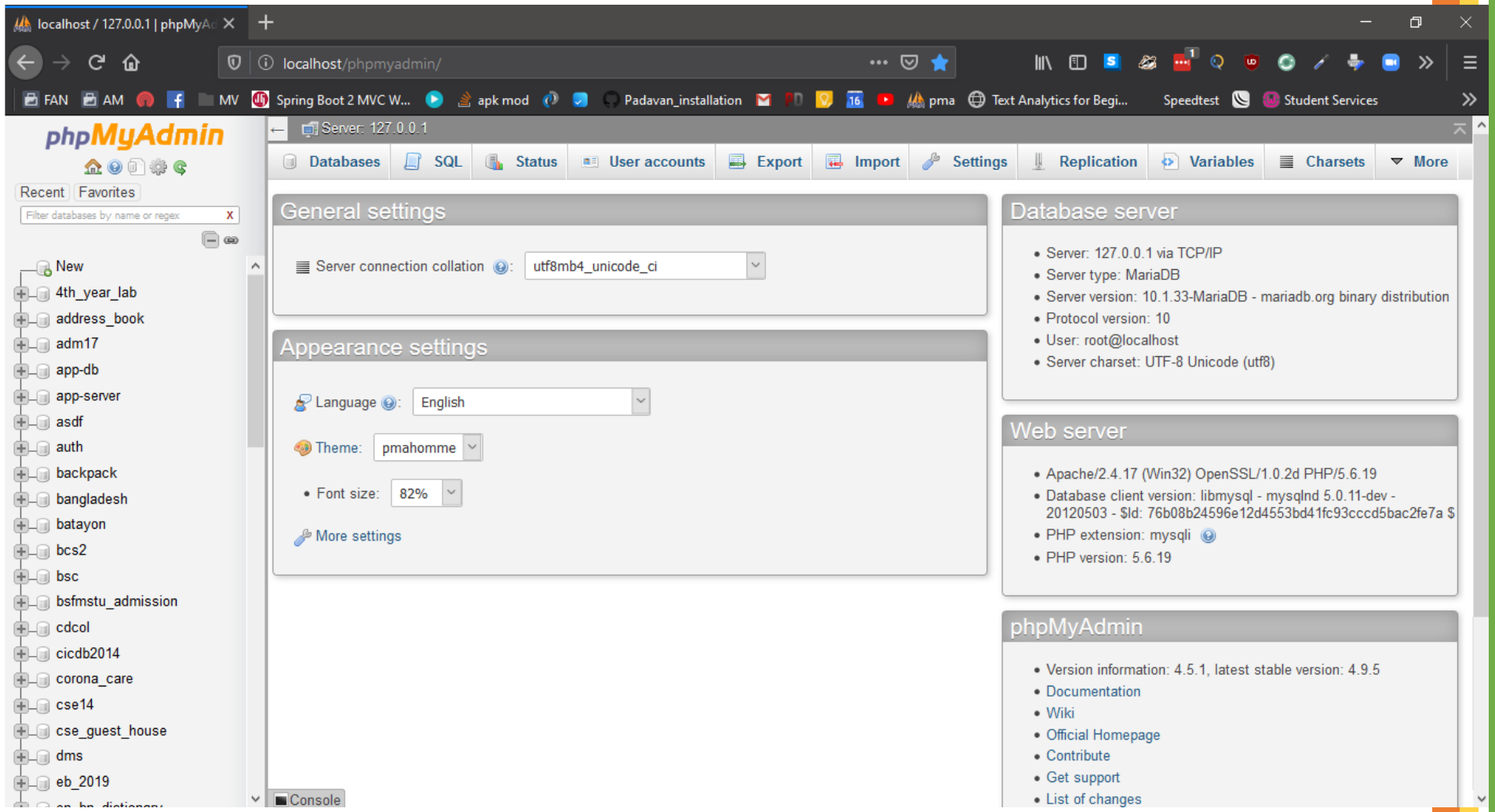


1. Database Software Installation

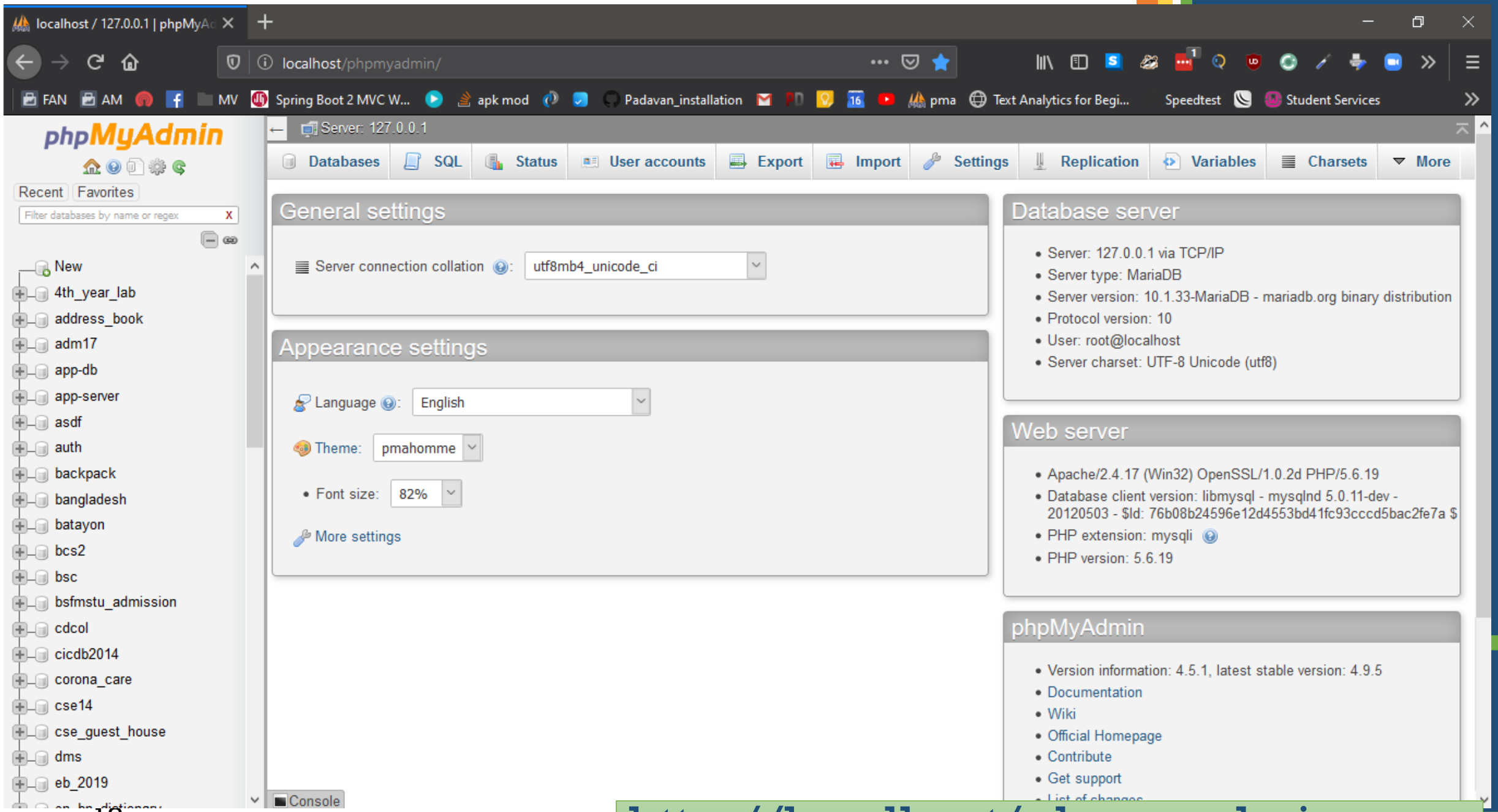


http://localhost/phpmyadmin

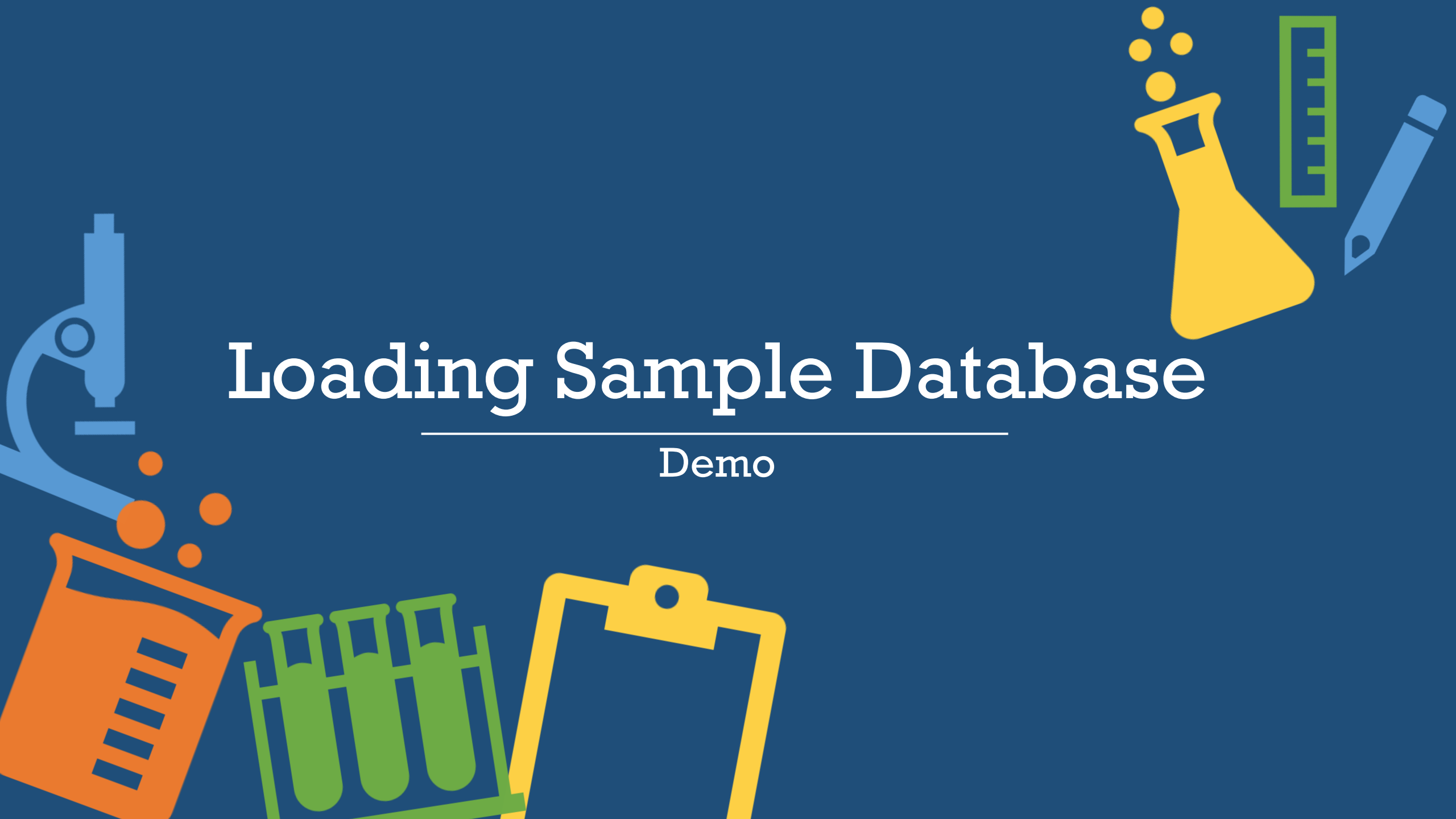
1. Database Software Installation



http://localhost/phpmyadmin



http://localhost/phpmyadmin



Loading Sample Database

Demo

Loading Sample Database

Table: Instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	(NULL)
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

Table: Course

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

http://localhost/phpmyadmin

Run basic SQL query

Demo

SQL

- **S**tructures **Q**uery **L**anguage
- developed at IBM by **Donald D. Chamberlin** and **Raymond F. Boyce** for their database software “System R” in 1970
- **Oracle** made it popular.
- By 1986, ANSI and ISO officially adopted the SQL and standard Language for RDBMS.

SQL


MySQL


ORACLE


Microsoft
SQL Server


SYBASE
An SAP Company


PostgreSQL


INGRES


Apache Derby


MariaDB


HyperSQL


H2


Informix


Amazon Redshift


PROGRESS


IBM
DB2


FRONTBASE


SQLite

Basic SQL Query

- For getting all data from a specific table

```
SELECT * FROM <table_name>
```

```
SELECT * FROM instructor
```

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Basic SQL Query

- Showing only selected fields

```
SELECT
        field1,fiel2,…fieldn
FROM
        <table_name>
```

```
SELECT  name, dept_name  FROM instructor
```

name	dept_name
Srinivasan	Comp. Sci.
Wu	Finance
Mozart	Music
Einstein	Physics
El Said	History
Gold	Physics
Katz	Comp. Sci.
Califieri	History
Singh	Finance
Crick	Biology
Brandt	Comp. Sci.
Kim	Elec. Eng.

Basic SQL Query

- Filtering data based on condition

```
SELECT
        field1,fiel2,…fieldn
FROM
        <table_name>
WHERE
        <condition>
```

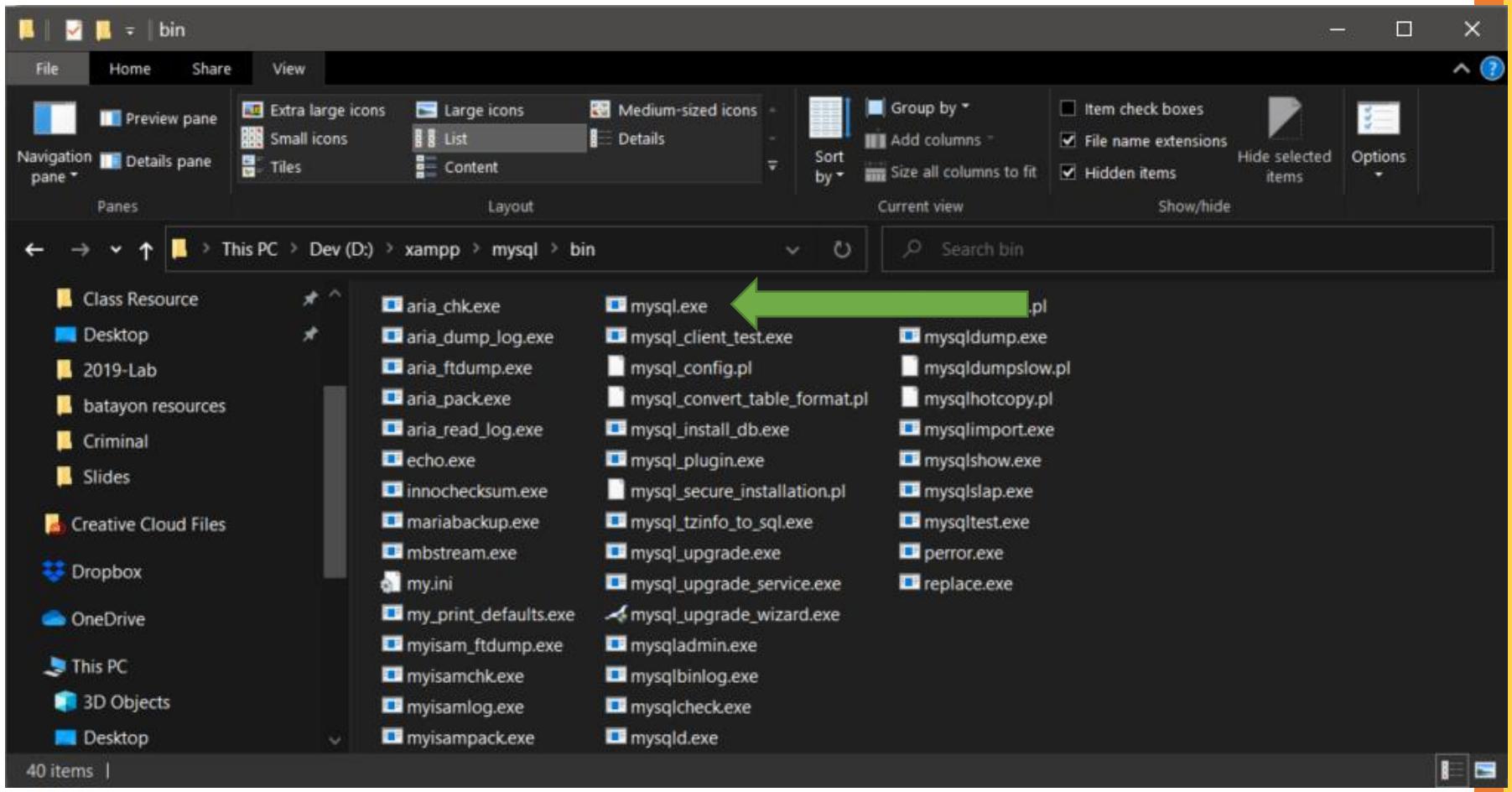
```
SELECT  *  FROM instructor
WHERE dept_name='Comp. Sci.'
```

	ID	name	dept_name	salary
e	10101	Srinivasan	Comp. Sci.	65000.00
e	45565	Katz	Comp. Sci.	75000.00
e	83821	Brandt	Comp. Sci.	92000.00

Some Additional Tools

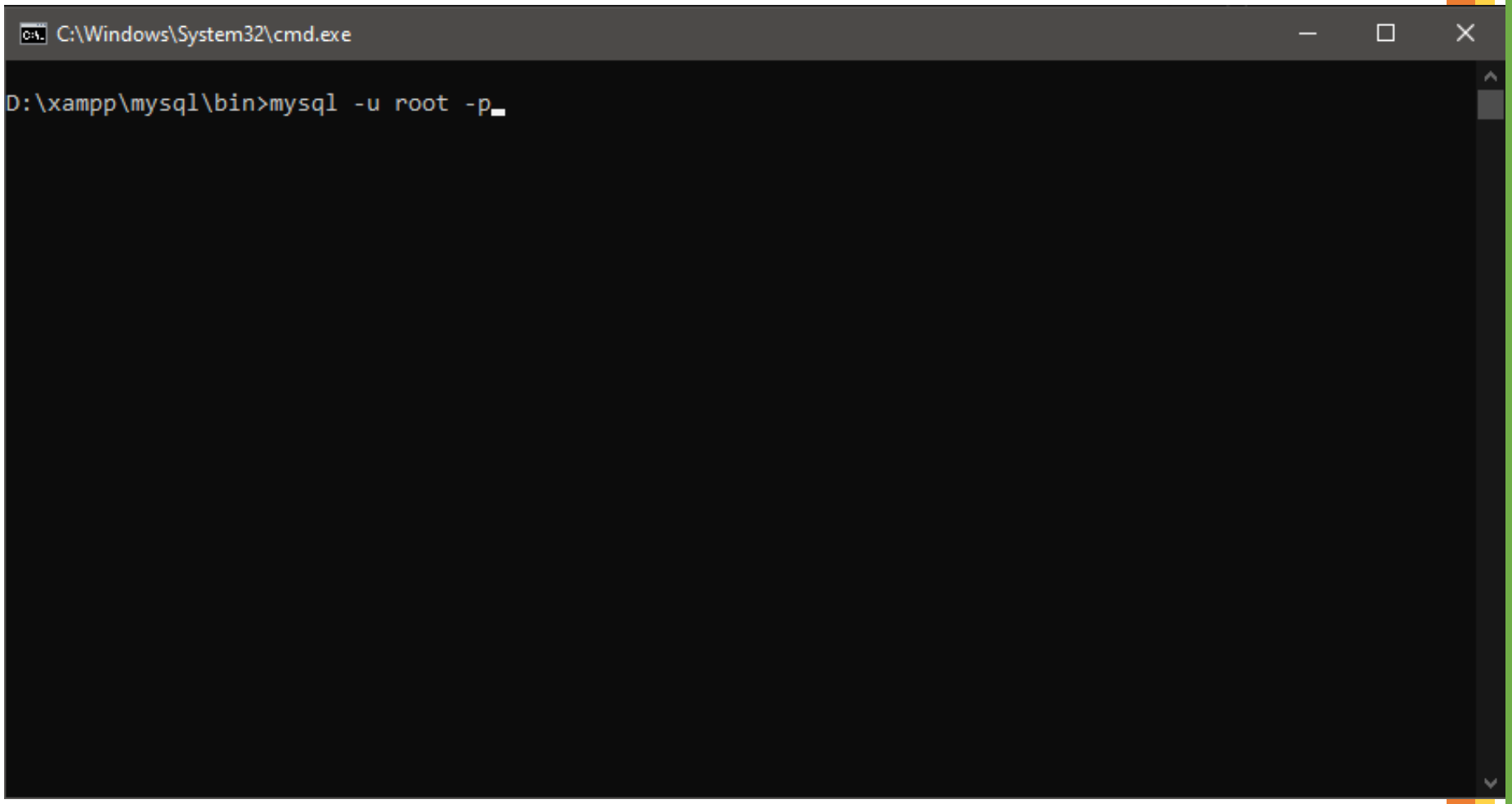
5. Locate mysql within XAMPP

C:\xampp\bin\mysql\bin



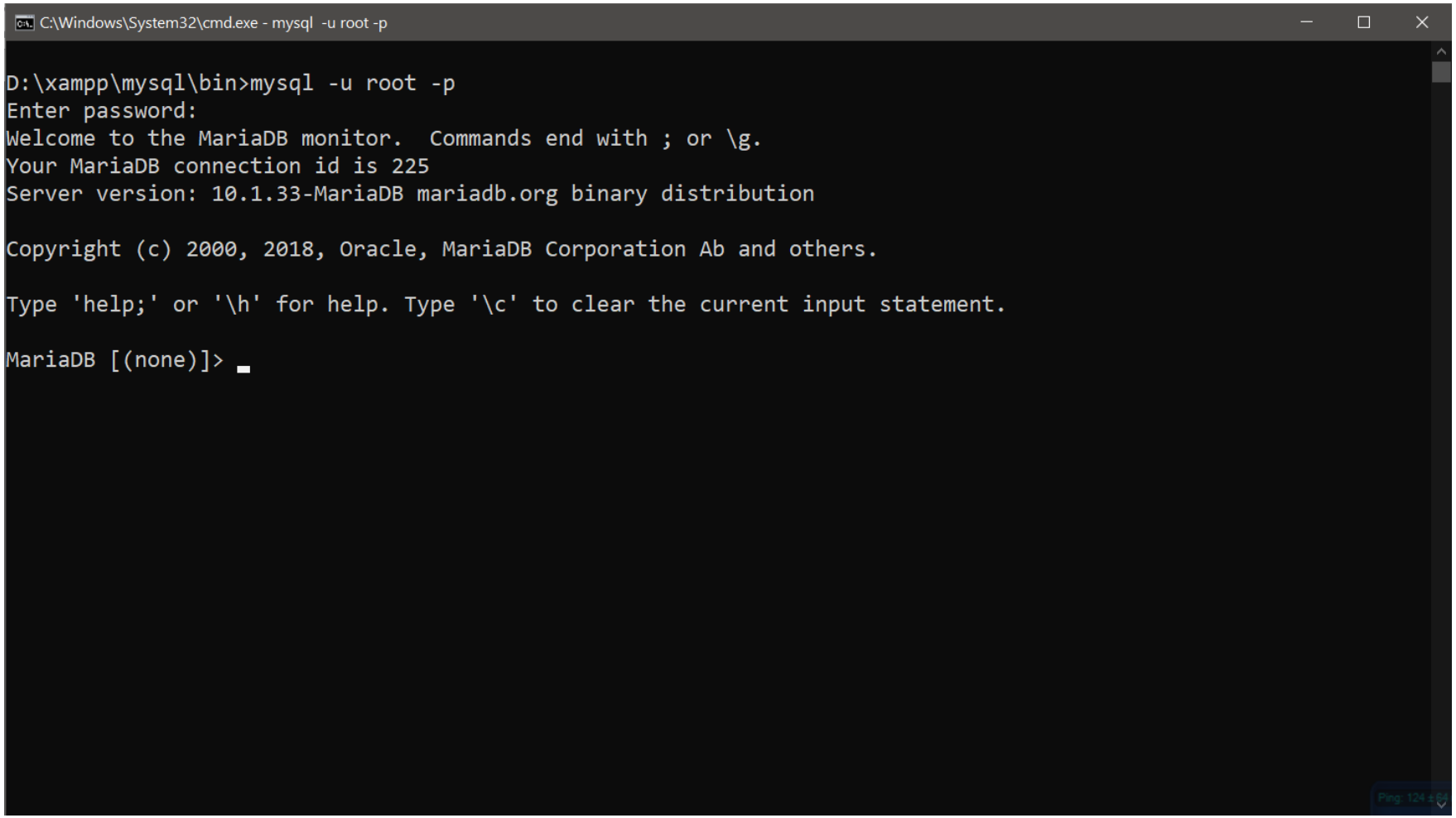
6. Using command prompt run mysql

```
mysql –u root -p
```



6. Using command prompt run mysql

mysql –u root -p



7. Run some query on the db server

- **Show available databases**

```
show  databases;
```

- **Open a database**

```
use <database_name>;
```

- **Show tables on opened database**

```
show  tables;
```

7. Run some query on the db server

- **Show schema of a table**

```
describe   <table_name>;
```

```
desc   <table_name>;
```

←short form

- **Show records from a database**

```
select  *  from <table_name>;
```

7. Run some query on the db server

```
show  databases;
```

```
use <database_name>;
```

```
show  tables;
```

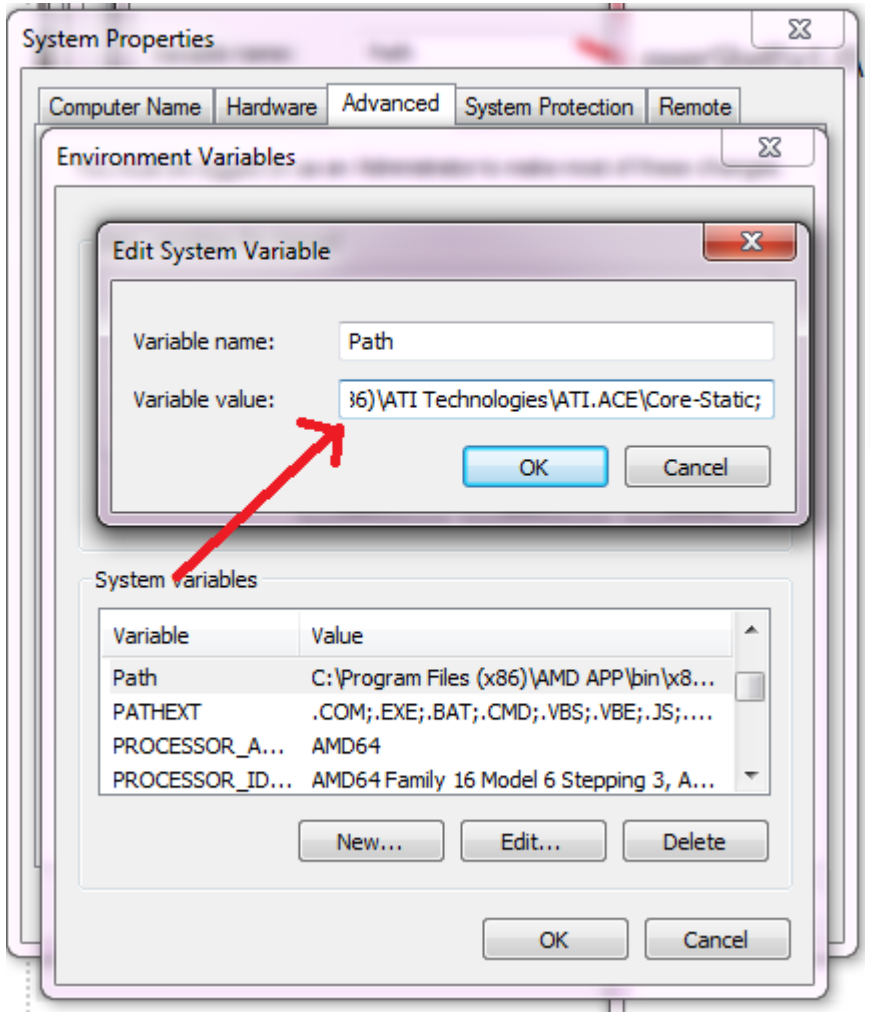
←short form

```
describe   <table_name>;
```

```
desc   <table_name>;
```

```
select  *  from <table_name>;
```

8. Add mysql to windows env path for easy access



Add your mysql path

c:\wamp\bin\mysql\mysql###\bin;

At the end of the Variable value

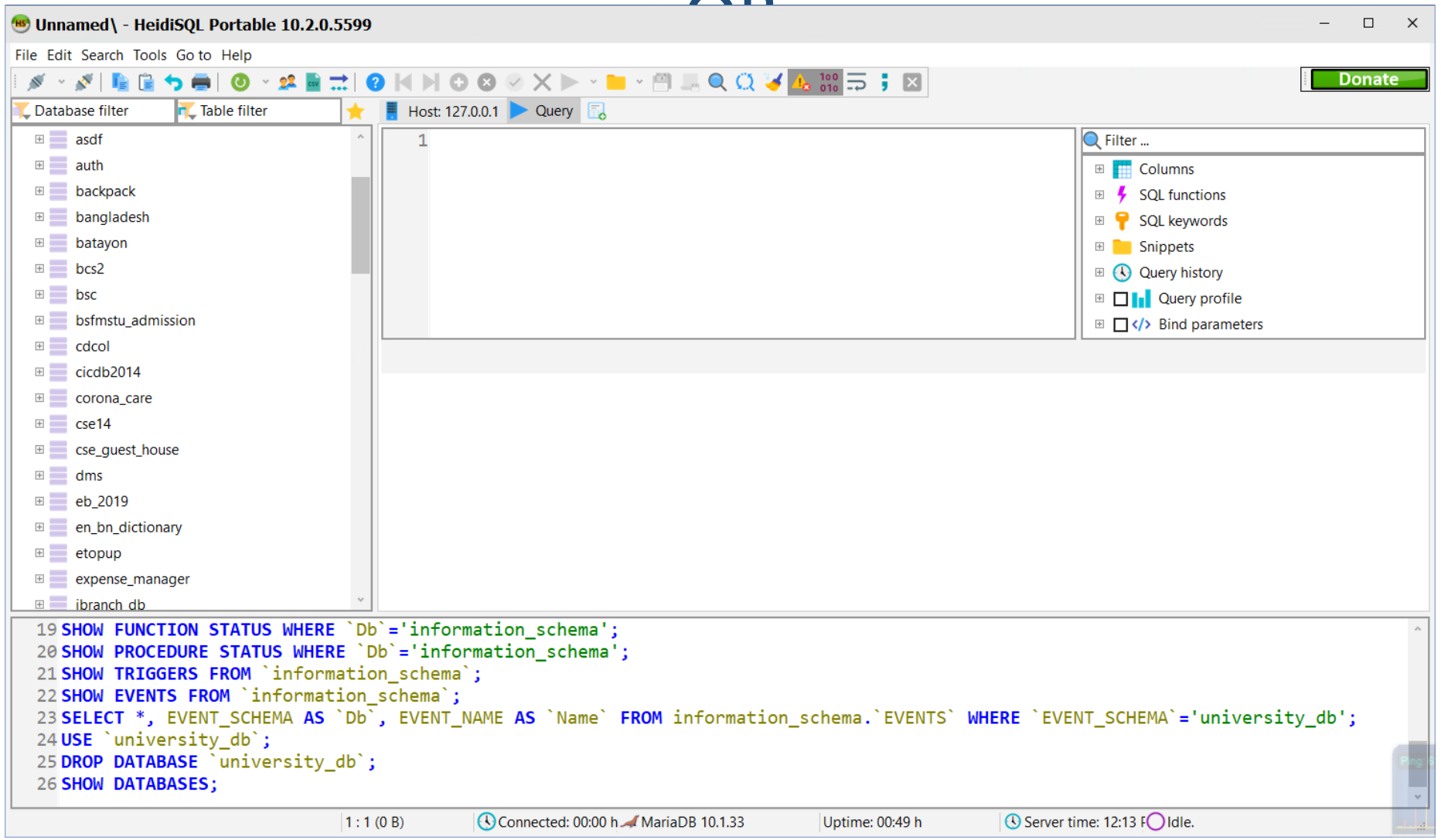
Test if everithing is OK

Open comand prompt anywhere and type

"mysql"

Without the quotation mark

I don't like **mysql command prompt**



11/22/2023 06:25

Demo.

End.

DBMS

Lab-2

Things we will complete today

- Define relations (table)
- SQL data types
- Integrity constraints
- Modifying a relation
- Destroying the relation

Define A Table







Let's Define a table in plain English

Create a table named department. The table should have the following information: department name, name of the building where the department is located, it's budget and the numbers of stuff the department have.

Let's Define a table in SQL

Create a **table** named **department**. The table should have the following information: **department name**, **name of the building** where the department is located, it's **budget** and the **numbers of stuff** the department have.

```
CREATE TABLE department (
        dept_name       varchar(100),
        building        varchar(120),
        budget          numeric(12,2),
        num_staff       int
)
```

SQL DDL

The set of relations in a database must be specified to the system by means of a data-definition language (DDL). The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:

- The names of each attribute of the schema.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk

SQL Data types

char(n)

A fixed-length character string with user-specified length n. The full form, **character**, can also be used instead.

varchar(n)

A variable-length character string with user-specified maximum length n.

int

An integer. The full form, **integer**, is equivalent.

smallint

A small integer (a machine-dependent subset of the integer type).

SQL Data types

numeric(p,d)

A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, numeric(3,1) allows 44.5 to be stored, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

real, **double precision**

Floating-point and double-precision floating-point numbers with machine-dependent precision.

float(n)

A floating-point number, with precision of at least n digits.

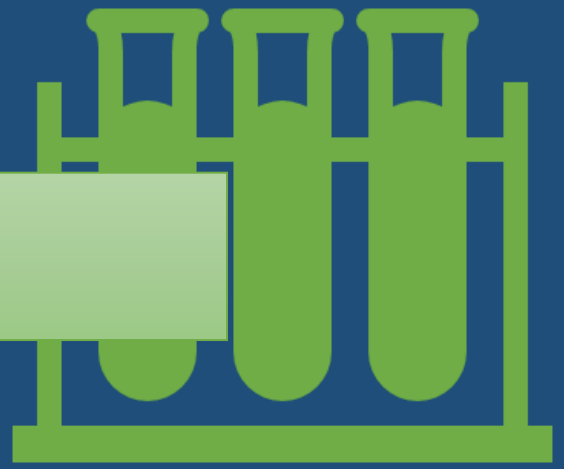
MY SQL Data types

```
INT, TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT,
DECIMAL, FLOAT, DOUBLE, REAL,
BIT, BOOLEAN,
SERIAL,
DATE, DATETIME, TIMESTAMP, TIME, YEAR,
CHAR, VARCHAR,
TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT,
BINARY, VARBINARY,
TINYBLOB, MEDIUMBLOB, BLOB, LONGBLOB,
ENUM, SET,

GEOMETRY, POINT, LINESTRING, POLYGON, MULTIPOINT,
MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION,
```

For More About My SQL Datatypes

https://www.tutorialspoint.com/mysql/mysql-data-types.htm



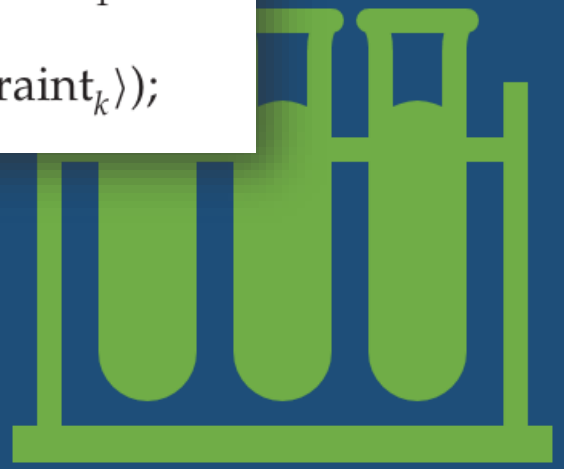
Integrity Constraints in Create Table

- **not null**
- **primary key (A1, …, An)**
- **Many more ….**

```
CREATE  TABLE  department (
        dept_name       varchar(100),
        building        varchar(120) not null,
        budget          numeric(12,2) not null,
        num_staff       int,
        primary key (  dept_name   )
)
```

create table r
$(A_1 \quad D_1,$
$A_2 \quad D_2,$
$\ldots,$
$A_n \quad D_n,$
$\langle \text{integrity-constraint}_1 \rangle,$
$\ldots,$
$\langle \text{integrity-constraint}_k \rangle);$

primary key declaration on an attribute automatically ensures **not null**
in SQL-92 onwards, needs to be explicitly stated in SQL-89



(Almost) Full Definition a table in SQL

```sql
CREATE TABLE course (
  course_id varchar(8) NOT NULL,
  title varchar(50) DEFAULT NULL,
  dept_name varchar(20) DEFAULT NULL,
  credits decimal(2,0) DEFAULT NULL,
  PRIMARY KEY (course_id),
  KEY dept_name (dept_name)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Schema modification

- Adding/ Removing a column
- Modifying column
- Deleting the whole scheme

Adding a column

alter table *r* **add** *A D*;

where *r* is the name of an existing relation, *A* is the name of the attribute to be added, and *D* is the type of the added attribute. We can drop attributes from a relation by the command

Add a column for department code in **department** table

```
ALTER TABLE department ADD dept_code varchar(6) NOT NULL;
```

```
ALTER TABLE department ADD dept_code varchar(6) NOT NULL AFTER dept_name;
```

Removing a column

alter table *r* **drop** *A*;

Remove the column dept_code from **department** table

```
ALTER TABLE DROP dept_code;
```

Modifying a column

ALTER TABLE *table_name* **MODIFY** *column_name datatype*

change column length of **dept_code**

```
ALTER TABLE department MODIFY dept_code varchar(10)
```

Deleting the WHOLE table

```
DROP TABLE department;
```

Deleting only data (Keep table structure)

Two option

```
DELETE FROM department;
```

or

```
TRUNCATE department;
```

08/16/2020 13:32

Demo.

End.

DBMS

Lab-3

Things we will complete today

- Quick recap of previous class
- Insert new data into table
- Update existing data
- Remove data from table
- Query on single relation
- Query on multiple relation

Insert new data into table

Insert new data into table

It is possible to write the INSERT INTO statement in two ways.

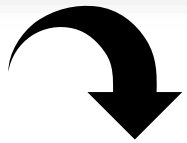
The first way specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Insert new data into table

Example:

```
insert into course (course_id, title, dept_name, credits)
        values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```



```
insert into course (title, course_id, credits, dept_name)
        values ('Database Systems', 'CS-437', 4, 'Comp. Sci.');
```

university_db.course: 13 rows total

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Insert new data into table

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

university_db.course: 13 rows total

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

```
insert into course
        values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

Insert new data from another table

Table1

Id	FirstName
101	Adam
102	Johm

Table2

EmployeeId	EmpName

```
INSERT INTO Table2(EmployeeId,EmployeeName)
SELECT Id,FirstName from Table1;
```

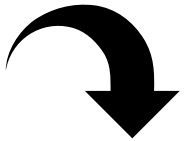


Table2

EmployeeId	EmpName
101	Adam
102	Johm

```
INSERT INTO table_name(column_list)
SELECT
    select_list
FROM
    another_table
WHERE
    condition;
```

Update existing data

Update Syntax

```
UPDATE table
SET column1 = expression-1,
      column2 = expression-2,
   ...
[WHERE conditions];
```

Expression can be a value (eg: 10 , 'John') or a Arithmatic/logical/ string operation resulting in a value (eg: salary*1.15)

```
update instructor
set salary= salary * 1.05;
```

```
update instructor
set salary = salary * 1.05
where salary < 70000;
```

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

Give a 5 percent salary raise to instructors whose salary is less than average

```
update instructor
set salary = salary * 1.05
where salary < (select avg (salary)
                from instructor);
```

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

All instructors with salary over $100,000 receive a 3 percent raise, whereas all others receive a 5 percent raise.

```
update instructor
set salary = salary * 1.03
where salary > 100000;
```

```
update instructor
set salary = salary * 1.05
where salary <= 100000;
```

```
update instructor
set salary = case
                when salary <= 100000 then salary * 1.05
```

```
case
    when pred_1 then result_1
    when pred_2 then result_2
    ...
    when pred_n then result_n
    else result_0
end
```

Delete data from table

SQL delete statement

We can delete only whole tuples; we cannot delete values on only particular attributes

```
DELETE from table_name [ WHERE condition ];
```

delete from *instructor;*

delete from *instructor*
where *dept_name*= 'Finance';

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

Queries on a Single Relation

Basic structure of SELECT

```
SELECT
    column1, column2, columnN
FROM
    table_name
[WHERE condition];
```

```
SELECT *
FROM table_name
[WHERE condition];
```

```
SELECT ID, `name`, dept_name
FROM instructor
```

instructor (3×12)

ID	name	dept_name
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music
22222	Einstein	Physics
32343	El Said	History
33456	Gold	Physics
45565	Katz	Comp. Sci.
58583	Califieri	History
76543	Singh	Finance
76766	Crick	Biology
83821	Brandt	Comp. Sci.
98345	Kim	Elec. Eng.

Basic structure of SELECT

```
1 SELECT *
2 FROM instructor
3 WHERE dept_name='Comp. Sci.'
4
```

instructor (4×3)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
45565	Katz	Comp. Sci.	75,000.00
83821	Brandt	Comp. Sci.	92,000.00

Ordering Result

```
SELECT *
FROM table_name
[WHERE condition];
[ORDER BY
    column1 [ASC|DESC],
    column2 [ASC|DESC],
    ... ]
```

```
1 SELECT *
2 FROM instructor
3 ORDER BY salary DESC
```

instructor (4×12)

ID	name	dept_name	salary
22222	Einstein	Physics	95,000.00
83821	Brandt	Comp. Sci.	92,000.00
12121	Wu	Finance	90,000.00
33456	Gold	Physics	87,000.00
76543	Singh	Finance	80,000.00
98345	Kim	Elec. Eng.	80,000.00
45565	Katz	Comp. Sci.	75,000.00
76766	Crick	Biology	72,000.00
10101	Srinivasan	Comp. Sci.	65,000.00
58583	Califieri	History	62,000.00
32343	El Said	History	60,000.00
15151	Mozart	Music	40,000.00

```
1 SELECT *
2 FROM instructor
3 ORDER BY dept_name, salary DESC
```

instructor (4×12)

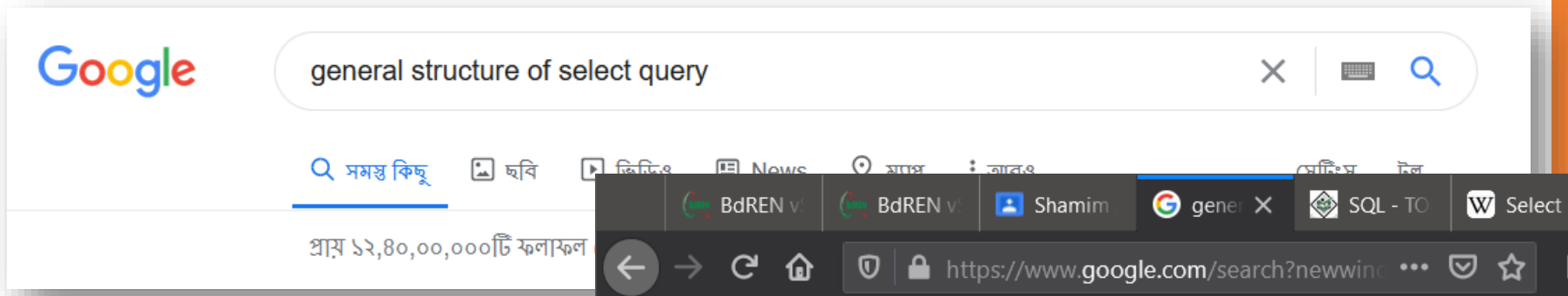
ID	name	dept_name	salary
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
45565	Katz	Comp. Sci.	75,000.00
10101	Srinivasan	Comp. Sci.	65,000.00
98345	Kim	Elec. Eng.	80,000.00
12121	Wu	Finance	90,000.00
76543	Singh	Finance	80,000.00
58583	Califieri	History	62,000.00
32343	El Said	History	60,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
33456	Gold	Physics	87,000.00

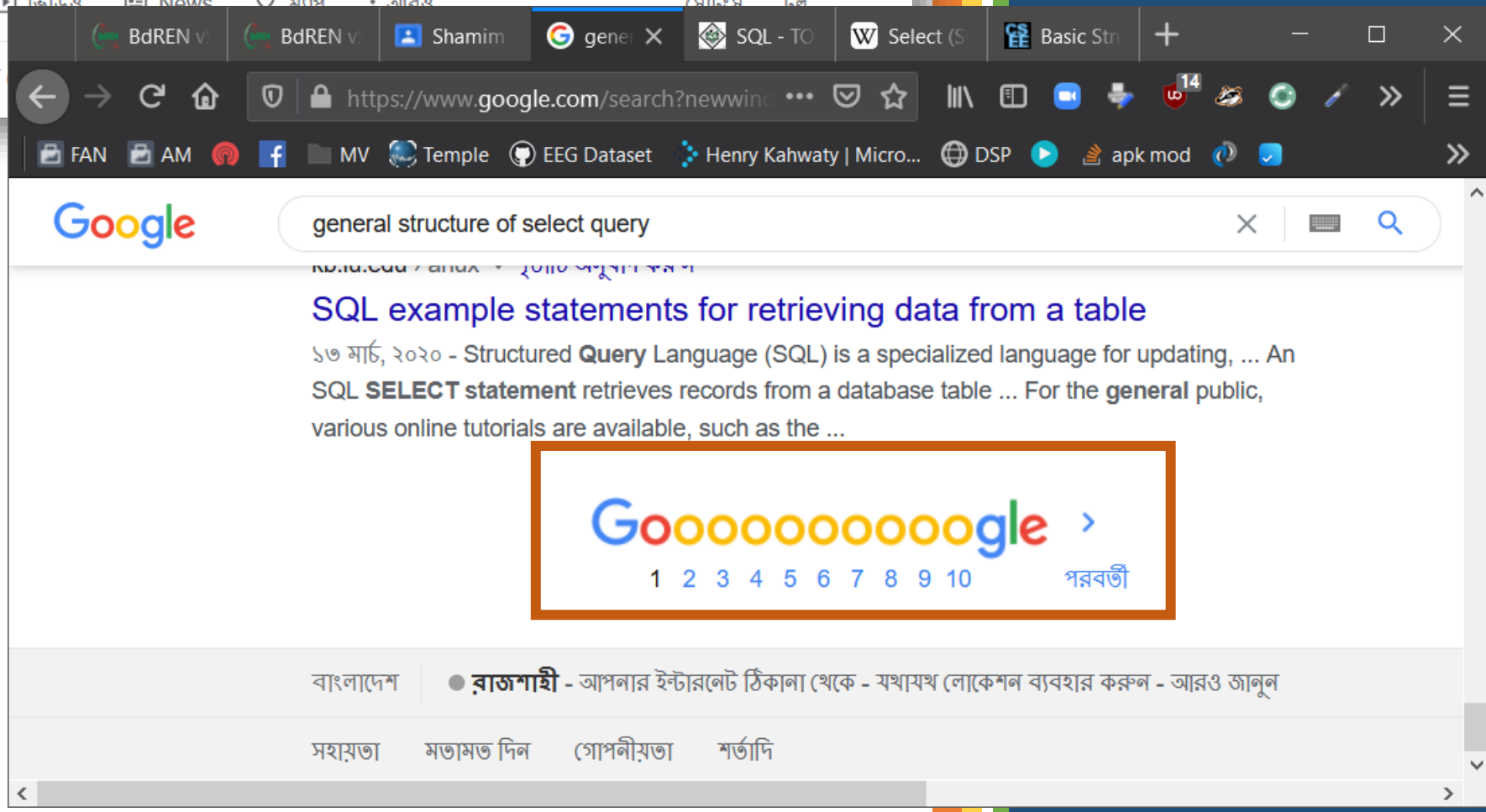
```
1 SELECT * FROM instructor
2 ORDER BY dept_name desc, salary asc
3
```

instructor (4×12)

ID	name	dept_name	salary
33456	Gold	Physics	87,000.00
22222	Einstein	Physics	95,000.00
15151	Mozart	Music	40,000.00
32343	El Said	History	60,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
12121	Wu	Finance	90,000.00
98345	Kim	Elec. Eng.	80,000.00
10101	Srinivasan	Comp. Sci.	65,000.00
45565	Katz	Comp. Sci.	75,000.00
83821	Brandt	Comp. Sci.	92,000.00
76766	Crick	Biology	72,000.00

Limiting number or row in output





Limiting number or row in output

```
SELECT *
FROM table_name
[LIMIT n]
```

```
1 SELECT * FROM instructor LIMIT 3
2
3
```

instructor (4×3)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00

```
SELECT *
FROM table_name
[LIMIT m,n]
```

```
1 SELECT * FROM instructor
2 LIMIT 3,4
3
```

instructor (4×4)

ID	name	dept_name	salary
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00

instructor (4×12)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

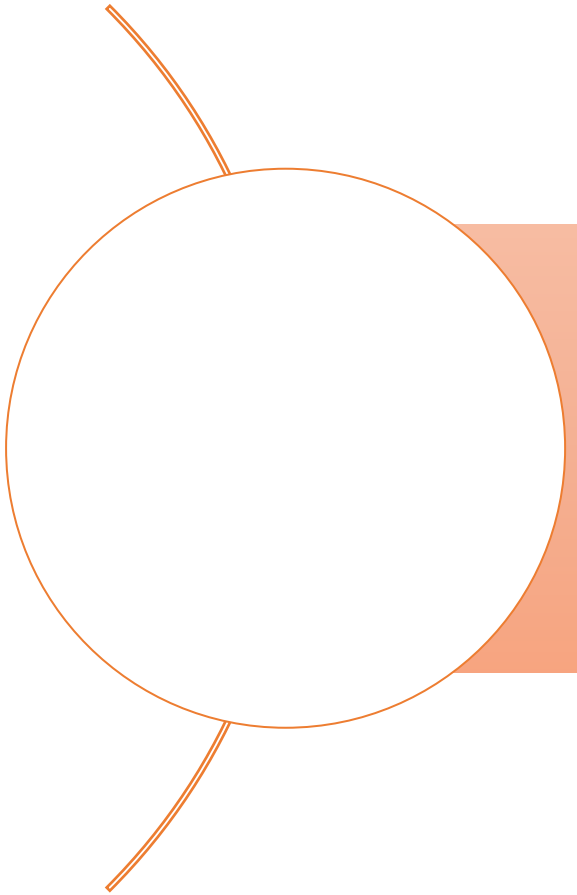
Demo.

End.

DBMS

Lab-4

Things we will complete today



Query on single relation (continue)

MySQL Comparison Operators


MySQL Comparison
Operators

Comparison Operations

Comparison operators are used in the WHERE clause to determine which records to select.

```
1  SELECT
2      column1, column2, columnN
3  FROM
4      table_name
5  [WHERE condition];
```

Comparison Operator	Description
=	Equal
<=>	Equal (Safe to compare NULL values)
<>	Not Equal
!=	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=	Less Than or Equal
IN ( )	Matches a value in a list
NOT	Negates a condition
BETWEEN	Within a range (inclusive)
IS NULL	NULL value
IS NOT NULL	Non-NULL value
LIKE	Pattern matching with % and _
EXISTS	Condition is met if subquery returns at least one row

The = operator

Example:

contact_id	last_name	website1	website2
1	Johnson	techonthenet.com	<NULL>
2	Anderson	<NULL>	<NULL>
3	Smith	TBD	TDB
4	Jackson	checkyourmath.com	digminecraft.com

```
SELECT *
FROM contacts
WHERE website1 = website2;
```

Because we used the `=` operator, we would get the following results:

contact_id	last_name	website1	website2
3	Smith	TBD	TDB

The <=> operator

Example:

contact_id	last_name	website1	website2
1	Johnson	techonthenet.com	<NULL>
2	Anderson	<NULL>	<NULL>
3	Smith	TBD	TDB
4	Jackson	checkyourmath.com	digminecraft.com

```
SELECT *
FROM contacts
WHERE website1 <=>website2;
```

Because we used the `<=>` operator, we would get the following results:

contact_id	last_name	website1	website2
2	Anderson	<NULL>	<NULL>
3	Smith	TBD	TDB

The Inequality Operator <> and !=

contact_id	last_name	website1	website2
1	Johnson	techonthenet.com	<NULL>
2	Anderson	<NULL>	<NULL>
3	Smith	TBD	TDB
4	Jackson	checkyourmath.com	digminecraft.com

Example:

```
SELECT *
FROM contacts
WHERE last_name <> 'Johnson';
```

```
SELECT *
FROM contacts
WHERE last_name != 'Johnson';
```

contact_id	last_name	website1	website2
2	Anderson	<NULL>	<NULL>
3	Smith	TBD	TDB
4	Jackson	checkyourmath.com	digminecraft.com

08/30/2020 19:04

The >, >=, < and <=

contact_id	last_name	website1	website2
1	Johnson	techonthenet.com	<NULL>
2	Anderson	<NULL>	<NULL>
3	Smith	TBD	TDB
4	Jackson	checkyourmath.com	digminecraft.com

Example:

```
SELECT *
FROM contacts
WHERE contact_id > 50;
```

```
SELECT *
FROM contacts
WHERE contact_id >= 50;
```

```
SELECT *
FROM inventory
WHERE product_id < 300;
```

```
SELECT *
FROM inventory
WHERE product_id <= 300;
```

08/31/2020 09:34

The in(...) operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

```
1  SELECT column_name(s)
2  FROM table_name
3  WHERE column_name = value-1 or column_name=value-2, ...column_name=value-n);
4
```

```
1  SELECT column_name(s)
2  FROM table_name
3  WHERE column_name IN (value-1, value-2, ... value-n);
4
```

The in(…) operator

selects the names of instructors whose names are either “Mozart” or “Einstein”

```
1 SELECT *
2 FROM instructor
3 WHERE `name` IN ('Mozart', 'Einstein');
```

instructor (4×2)

ID	name	dept_name	salary
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The in(…) operator

Using ‘in’ selects the names of instructors whose ids are less than 30000

```
1 SELECT *
2 FROM instructor
3 WHERE ID IN (SELECT id FROM instructor WHERE id < 30000);
```

instructor (4×4)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The not in(…) operator

selects the names of instructors whose names are neither “Mozart” nor “Einstein”

```
1 SELECT *
2 FROM instructor
3 WHERE `name` NOT IN ('Mozart', 'Einstein');
```

instructor (4×10)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The **between** operator

between comparison operator is used to simplify **where** clauses that specify that a value be less than or equal to some value and greater than or equal to some other value

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The **between** operator

find the names of instructors with salary amounts between $90,000 and $100,000

```
select name
from instructor
where salary <= 100000 and salary >= 90000;
```

```
1 SELECT `name`, salary
2 FROM instructor
3 WHERE salary BETWEEN 90000 AND 100000;
```

instructor (2×3)

NAME	salary
Wu	90,000.00
Einstein	95,000.00
Brandt	92,000.00

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The **is null** operator

Null values present special problems in relational operations, including arithmetic operations, comparison operations, and set operations.

The result of an arithmetic expression (involving, for example +, −, ∗, or /) is null if any of the input values is null. For example, if a query has an expression

- **and**: The result of *true* **and** *unknown* is *unknown*, *false* **and** *unknown* is *false*, while *unknown* **and** *unknown* is *unknown*.
- **or**: The result of *true* **or** *unknown* is *true*, *false* **or** *unknown* is *unknown*, while *unknown* **or** *unknown* is *unknown*.
- **not**: The result of **not** *unknown* is *unknown*.

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The **is null** operator

SQL uses the special keyword **null** in a predicate to test for a null value

Find the names of instructors whose salary is unknown.

```
1 SELECT * FROM instructor
2 WHERE salary = NULL
```

instructor (4×0)

ID	name	dept_name	salary

```
1 SELECT * FROM instructor
2 WHERE salary IS NULL
3
4
```

instructor (4×1)

ID	name	dept_name	salary
76766	Crick	Biology	(NULL)

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The **is not null** operator

The opposite of **is null**

Find the names of instructors whose salary we know.

```
1 SELECT * FROM instructor
2 WHERE salary IS NOT NULL
```

instructor (4×11)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

The like operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

There are two wildcards often used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

The like operator

Find the names of all courses whose title includes the substring '**to**'.

```
1 SELECT * FROM course
2 WHERE title LIKE '%to%'
3
```

course (4×4)

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
CS-101	Intro. to Computer Science	Comp. Sci.	4
EE-181	Intro. to Digital Systems	Elec. Eng.	3
HIS-351	World History	History	3

university_db.course: 13 rows total

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

The like operator

- 'Intro%' matches any string beginning with "Intro".
- '%Comp%' matches any string containing "Comp" as a substring, for example, 'Intro. to Computer Science', and 'Computational Biology'.
- '___' matches any string of exactly three characters.
- '___%' matches any string of at least three characters.

university_db.course: 13 rows total

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

The like operator

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	
WHERE CustomerName LIKE '%a'	
WHERE CustomerName LIKE '%or%'	
WHERE CustomerName LIKE '_r%'	
WHERE CustomerName LIKE 'a_%'	
WHERE CustomerName LIKE 'a__%'	
WHERE ContactName LIKE 'a%o'	

The exists operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns true if the subquery returns one or more records

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

The exists operator

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20

```
SELECT
    SupplierName
FROM
    Suppliers
WHERE
    EXISTS (SELECT ProductName
            FROM Products
            WHERE
                Products.SupplierID = Suppliers.supplierID
                AND
                Price < 20
            );
```

Result:

Number of Records: 1

SupplierName
New Orleans Cajun Delights

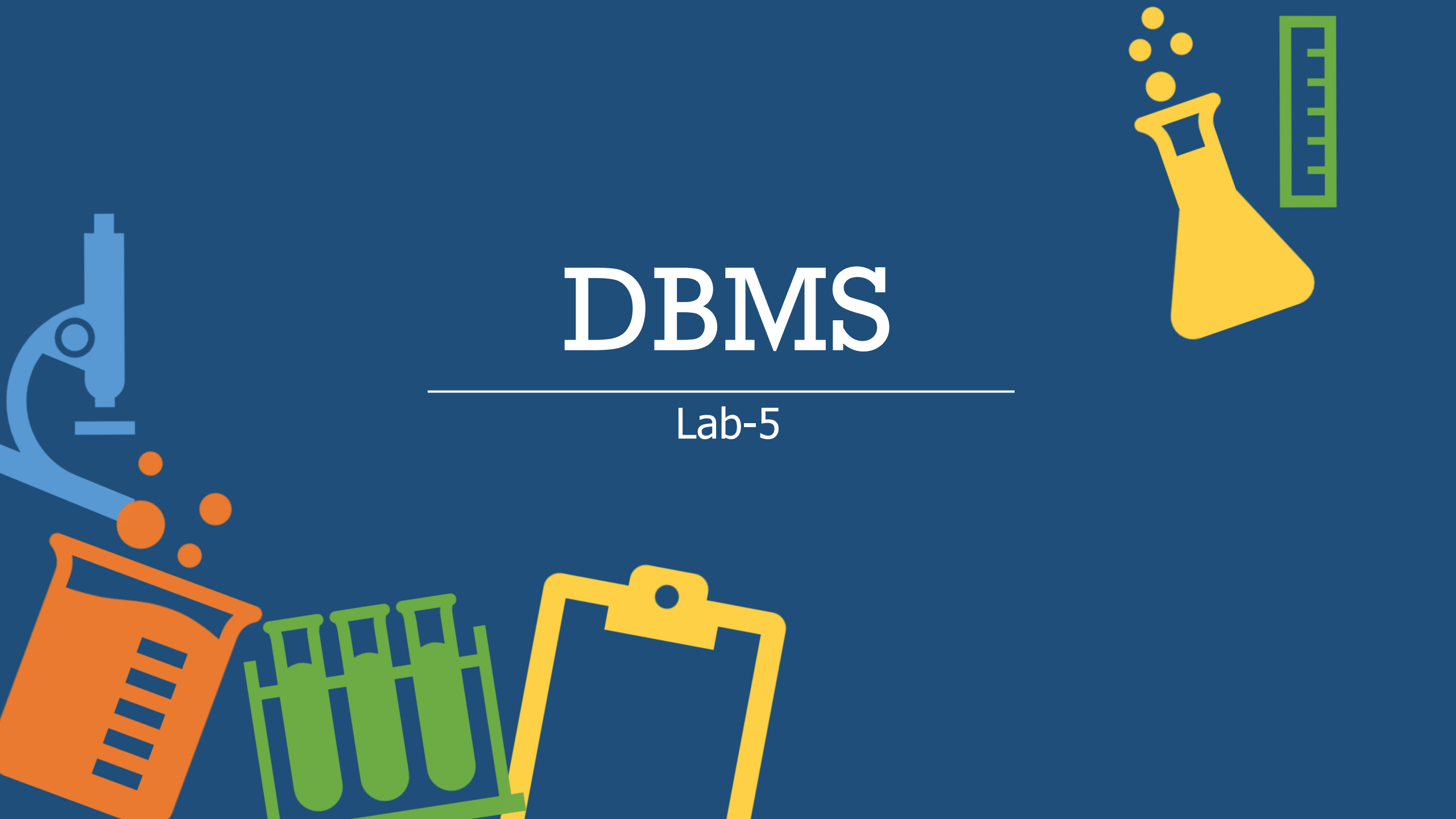
ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA
4	Tokyo Traders	Yoshi Nagase	9-8 Sekimai Musashino-shi	Tokyo	100	Japan

08/31/2020 13:06

Demo.

End.



DBMS

Lab-5

Things we will complete today

- Aggrigate functions
- Query on Multiple relation






Aggregate Functions

Aggregate functions

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

SQL offers five built-in aggregate functions:

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

The input to **sum** and **avg** must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well

Aggregate functions

- Find the average salary of instructors in the Computer Science department

```
1 SELECT avg (salary)
2 FROM instructor
3 WHERE dept_name= 'Comp. Sci.'
4
```

Result #1 (1×1)

avg (salary)
77,333.333333

- Find the number of departments in *instructor* relation

```
1 SELECT COUNT(distinct dept_name)
2 FROM instructor
```

Result #1 (1×1)

COUNT(distinct dept_name)
7

- Find the number of tuples in the *instructor* relation

```
1 SELECT COUNT(*) AS num_row
2 FROM instructor
```

Result #1 (1×1)

num_row
12

Aggregate functions - Group By

- The **GROUP BY** statement groups rows that have the same values into summary rows like *"Find the average salary of instructors in each department"*

```
SELECT dept_name, avg (salary) as avg_salary
FROM instructor
GROUP BY  dept_name;
```

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregate functions - Group By

- "Find the number of of instructors in each department"

```
1 SELECT dept_name, COUNT(id) AS num_teacher
2 FROM instructor
3 GROUP BY  dept_name;
4
```

instructor (2×7)

dept_name	num_teacher
Biology	1
Comp. Sci.	3
Elec. Eng.	1
Finance	2
History	2
Music	1
Physics	2

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Aggregate Functions – Having Clause

- "Find the number of of instructors in each department which **have more than one instructors**"

```
1 SELECT dept_name, count(id) as num_teacher
2 FROM instructor
3 GROUP BY  dept_name
4 HAVING num_teacher > 1
5
```

instructor (2×4)

dept_name	num_teacher
Comp. Sci.	3
Finance	2
History	2
Physics	2

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Aggregate Functions – Having Clause

- *The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.*

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Aggregate Functions – Having Clause

The meaning of a query containing aggregation, **group by**, or **having** clauses is defined by the following sequence of operations:

1. As was the case for queries without aggregation, the **from** clause is first evaluated to get a relation.
2. If a **where** clause is present, the predicate in the **where** clause is applied on the result relation of the **from** clause.
3. Tuples satisfying the **where** predicate are then placed into groups by the **group by** clause if it is present. If the **group by** clause is absent, the entire set of tuples satisfying the **where** predicate is treated as being in one group.
4. The **having** clause, if it is present, is applied to each group; the groups that do not satisfy the **having** clause predicate are removed.
5. The **select** clause uses the remaining groups to generate tuples of the result of the query, applying the aggregate functions to get a single result tuple for each group.

Page 88

Null Values and Aggregates

- Total all salaries

  select sum (*salary*)
  from *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

Query on Multiple relation

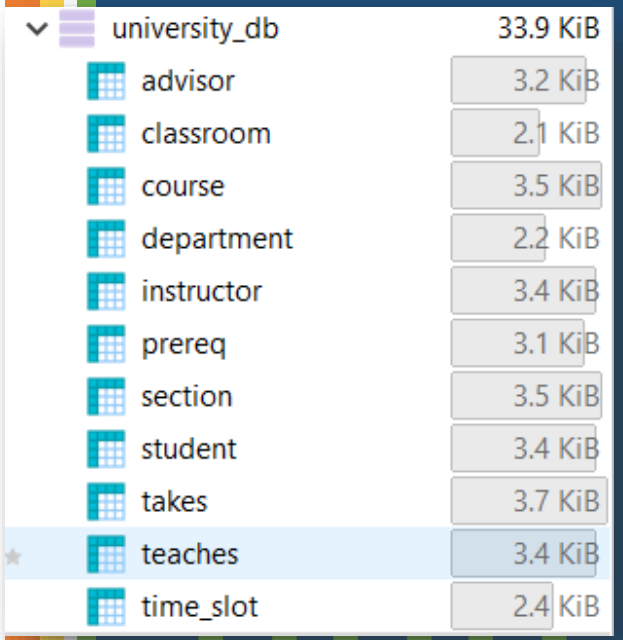
Show me a list containing all teachers name, their respective department and the building the department is situated.

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.department: 7 rows total

dept_name	building	budget
Biology	Watson	90,000.00
Comp. Sci.	Taylor	100,000.00
Elec. Eng.	Taylor	85,000.00
Finance	Painter	120,000.00
History	Painter	50,000.00
Music	Packard	80,000.00
Physics	Watson	70,000.00



10/05/2020 10:04






Cartesian Product

Cartesian Product

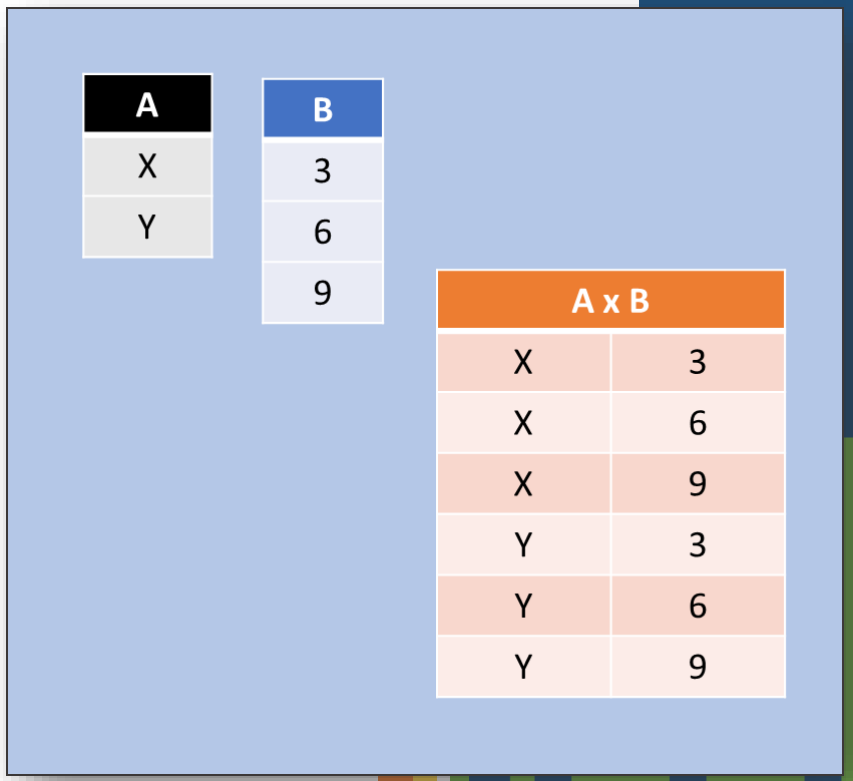
The Cartesian product of two sets A and B, denoted by A × B, is defined as the set consisting of all ordered pairs (a, b) for which

$$a \in A \quad \text{and} \quad b \in B.$$

For example, if

A = {x, y} and B = {3, 6, 9}, then

A × B = {(x, 3), (x, 6), (x, 9), (y, 3), (y, 6), (y, 9)}



Cartesian Product

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.department: 7 rows total

dept_name	building	budget
Biology	Watson	90,000.00
Comp. Sci.	Taylor	100,000.00
Elec. Eng.	Taylor	85,000.00
Finance	Painter	120,000.00
History	Painter	50,000.00
Music	Packard	80,000.00
Physics	Watson	70,000.00

instructor X department

Result #1 (7×84)

ID	name	dept_name	salary	dept_name	building	budget
10101	Srinivasan	Comp. Sci.	65,000.00	Biology	Watson	90,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Comp. Sci.	Taylor	100,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Elec. Eng.	Taylor	85,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Finance	Painter	120,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	History	Painter	50,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Music	Packard	80,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Physics	Watson	70,000.00
12121	Wu	Finance	90,000.00	Biology	Watson	90,000.00
12121	Wu	Finance	90,000.00	Comp. Sci.	Taylor	100,000.00
12121	Wu	Finance	90,000.00	Elec. Eng.	Taylor	85,000.00
12121	Wu	Finance	90,000.00	Finance	Painter	120,000.00
12121	Wu	Finance	90,000.00	History	Painter	50,000.00
12121	Wu	Finance	90,000.00	Music	Packard	80,000.00
12121	Wu	Finance	90,000.00	Physics	Watson	70,000.00
15151	Mozart	Music	40,000.00	Biology	Watson	90,000.00
15151	Mozart	Music	40,000.00	Comp. Sci.	Taylor	100,000.00
15151	Mozart	Music	40,000.00	Elec. Eng.	Taylor	85,000.00
15151	Mozart	Music	40,000.00	Finance	Painter	120,000.00
15151	Mozart	Music	40,000.00	History	Painter	50,000.00
15151	Mozart	Music	40,000.00	Music	Packard	80,000.00
15151	Mozart	Music	40,000.00	Physics	Watson	70,000.00
22222	Einstein	Physics	95,000.00	Biology	Watson	90,000.00

History

Cartesian Product

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.department: 7 rows total

dept_name	building	budget
Biology	Watson	90,000.00
Comp. Sci.	Taylor	100,000.00
Elec. Eng.	Taylor	85,000.00
Finance	Painter	120,000.00
History	Painter	50,000.00
Music	Packard	80,000.00
Physics	Watson	70,000.00

department X

dept_name	building	budget	ID	name	dept_name	salary
Biology	Watson	90,000.00	10101	Srinivasan	Comp. Sci.	65,000.00
Comp. Sci.	Taylor	100,000.00	10101	Srinivasan	Comp. Sci.	65,000.00
Elec. Eng.	Taylor	85,000.00	10101	Srinivasan	Comp. Sci.	65,000.00
Finance	Painter	120,000.00	10101	Srinivasan	Comp. Sci.	65,000.00
History	Painter	85,000.00	10101	Srinivasan	Comp. Sci.	65,000.00
Music	Packard	80,000.00	10101	Srinivasan	Comp. Sci.	65,000.00
Physics	Watson	70,000.00	10101	Srinivasan	Comp. Sci.	65,000.00
Biology	Watson	90,000.00	12121	Wu	Finance	90,000.00
Comp. Sci.	Taylor	100,000.00	12121	Wu	Finance	90,000.00
Elec. Eng.	Taylor	85,000.00	12121	Wu	Finance	90,000.00
Finance	Painter	120,000.00	12121	Wu	Finance	90,000.00
History	Painter	50,000.00	12121	Wu	Finance	90,000.00
Music	Packard	80,000.00	12121	Wu	Finance	90,000.00
Physics	Watson	70,000.00	12121	Wu	Finance	90,000.00
Biology	Watson	90,000.00	15151	Mozart	Music	40,000.00
Comp. Sci.	Taylor	100,000.00	15151	Mozart	Music	40,000.00
Elec. Eng.	Taylor	85,000.00	15151	Mozart	Music	40,000.00
Finance	Painter	120,000.00	15151	Mozart	Music	40,000.00
History	Painter	50,000.00	15151	Mozart	Music	40,000.00
Music	Packard	80,000.00	15151	Mozart	Music	40,000.00
Physics	Watson	70,000.00	15151	Mozart	Music	40,000.00
Biology	Watson	90,000.00	22222	Einstein	Physics	95,000.00

Cartesian Product in SQL

This result is not very heplfull.

We need to filter out the records that make sense based on some **common fields** between two relations

```
1 SELECT * FROM instructor , department
2
```

Result #1 (7×84)

ID	name	dept_name	salary	dept_name	building	budget
10101	Srinivasan	Comp. Sci.	65,000.00	Biology	Watson	90,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Comp. Sci.	Taylor	100,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Elec. Eng.	Taylor	85,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Finance	Painter	120,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	History	Painter	50,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Music	Packard	80,000.00
10101	Srinivasan	Comp. Sci.	65,000.00	Physics	Watson	70,000.00
12121	Wu	Finance	90,000.00	Biology	Watson	90,000.00
12121	Wu	Finance	90,000.00	Comp. Sci.	Taylor	100,000.00
12121	Wu	Finance	90,000.00	Elec. Eng.	Taylor	85,000.00
12121	Wu	Finance	90,000.00	Finance	Painter	120,000.00
12121	Wu	Finance	90,000.00	History	Painter	50,000.00
12121	Wu	Finance	90,000.00	Music	Packard	80,000.00
12121	Wu	Finance	90,000.00	Physics	Watson	70,000.00
15151	Mozart	Music	40,000.00	Biology	Watson	90,000.00
15151	Mozart	Music	40,000.00	Comp. Sci.	Taylor	100,000.00
15151	Mozart	Music	40,000.00	Elec. Eng.	Taylor	85,000.00
15151	Mozart	Music	40,000.00	Finance	Painter	120,000.00
15151	Mozart	Music	40,000.00	History	Painter	50,000.00
15151	Mozart	Music	40,000.00	Music	Packard	80,000.00
15151	Mozart	Music	40,000.00	Physics	Watson	70,000.00
22222	Einstein	Physics	95,000.00	Biology	Watson	90,000.00
22222	Einstein	Physics	95,000.00	Comp. Sci.	Taylor	100,000.00
22222	Einstein	Physics	95,000.00	Elec. Eng.	Taylor	85,000.00

Cartesian Product in SQL

```
1 SELECT * FROM instructor , department
2 WHERE instructor.dept_name = department.dept_name
3
```

Result #1 (7×12)

ID	name	dept_name	salary	dept_name	building	budget
10101	Srinivasan	Comp. Sci.	65,000.00	Comp. Sci.	Taylor	100,000.00
12121	Wu	Finance	90,000.00	Finance	Painter	120,000.00
15151	Mozart	Music	40,000.00	Music	Packard	80,000.00
22222	Einstein	Physics	95,000.00	Physics	Watson	70,000.00
32343	El Said	History	60,000.00	History	Painter	50,000.00
33456	Gold	Physics	87,000.00	Physics	Watson	70,000.00
45565	Katz	Comp. Sci.	75,000.00	Comp. Sci.	Taylor	100,000.00
58583	Califieri	History	62,000.00	History	Painter	50,000.00
76543	Singh	Finance	80,000.00	Finance	Painter	120,000.00
76766	Crick	Biology	(NULL)	Biology	Watson	90,000.00
83821	Brandt	Comp. Sci.	92,000.00	Comp. Sci.	Taylor	100,000.00
98345	Kim	Elec. Eng.	80,000.00	Elec. Eng.	Taylor	85,000.00

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.department: 7 rows total

dept_name	building	budget
Biology	Watson	90,000.00
Comp. Sci.	Taylor	100,000.00
Elec. Eng.	Taylor	85,000.00
Finance	Painter	120,000.00
History	Painter	50,000.00
Music	Packard	80,000.00
Physics	Watson	70,000.00

Show me a list containing all teachers name, their respective department and the building the department is situated.

```sql
SELECT
    name, instructor.dept_name, building
FROM
    instructor , department
WHERE
    instructor.dept_name = department.dept_name
```

Result #1 (3×12)

name	dept_name	building
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Califieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.department: 7 rows total

dept_name	building	budget
Biology	Watson	90,000.00
Comp. Sci.	Taylor	100,000.00
Elec. Eng.	Taylor	85,000.00
Finance	Painter	120,000.00
History	Painter	50,000.00
Music	Packard	80,000.00
Physics	Watson	70,000.00

Find the names of all instructors who have taught some course and the course_id of those courses, and year

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.teaches: 15 rows total

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2,009
10101	CS-315	1	Spring	2,010
10101	CS-347	1	Fall	2,009
12121	FIN-201	1	Spring	2,010
15151	MU-199	1	Spring	2,010
22222	PHY-101	1	Fall	2,009
32343	HIS-351	1	Spring	2,010
45565	CS-101	1	Spring	2,010
45565	CS-319	1	Spring	2,010
76766	BIO-101	1	Summer	2,009
76766	BIO-301	1	Summer	2,010
83821	CS-190	1	Spring	2,009
83821	CS-190	2	Spring	2,009
83821	CS-319	2	Spring	2,010
98345	EE-181	1	Spring	2,009

```
1 SELECT `name`, course_id, year
2 FROM
3     instructor , teaches
4 WHERE
5     instructor.id = teaches.ID
```

Result #1 (3×15)

name	course_id	year
Srinivasan	CS-101	2,009
Srinivasan	CS-315	2,010
Srinivasan	CS-347	2,009
Wu	FIN-201	2,010
Mozart	MU-199	2,010
Einstein	PHY-101	2,009
El Said	HIS-351	2,010
Katz	CS-101	2,010
Katz	CS-319	2,010
Crick	BIO-101	2,009
Crick	BIO-301	2,010
Brandt	CS-190	2,009
Brandt	CS-190	2,009
Brandt	CS-319	2,010
Kim	EE-181	2,009

Find the names of all instructors who have taught some course and the course_id of those courses

What if I also need course name?

Find the names of all instructors who have taught some course and the course_id, course name of those courses, and year

university_db.instructor: 12 rows total

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	(NULL)
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

university_db.course: 13 rows total

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

university_db.teaches: 15 rows total

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2,009
10101	CS-315	1	Spring	2,010
10101	CS-347	1	Fall	2,009
12121	FIN-201	1	Spring	2,010
15151	MU-199	1	Spring	2,010
22222	PHY-101	1	Fall	2,009
32343	HIS-351	1	Spring	2,010
45565	CS-101	1	Spring	2,010
45565	CS-319	1	Spring	2,010
76766	BIO-101	1	Summer	2,009
76766	BIO-301	1	Summer	2,010
83821	CS-190	1	Spring	2,009
83821	CS-190	2	Spring	2,009
83821	CS-319	2	Spring	2,010
98345	EE-181	1	Spring	2,009

```sql
SELECT *
FROM
   instructor , teaches, course
WHERE
   instructor.ID = teaches.ID AND teaches.course_id = course.course_id

```

Filter ...

- Columns in instruct
- SQL functions
- SQL keywords
- Snippets

Result #1 (13×15)

ID	name	dept_name	salary	ID	course_id	sec_id	semester	year	course_id	title	dept_name	credits
10101	Srinivasan	Comp. Sci.	65,000.00	10101	CS-101	1	Fall	2,009	CS-101	Intro. to Computer Science	Comp. Sci.	4
10101	Srinivasan	Comp. Sci.	65,000.00	10101	CS-315	1	Spring	2,010	CS-315	Robotics	Comp. Sci.	3
10101	Srinivasan	Comp. Sci.	65,000.00	10101	CS-347	1	Fall	2,009	CS-347	Database System Concepts	Comp. Sci.	3
12121	Wu	Finance	90,000.00	12121	FIN-201	1	Spring	2,010	FIN-201	Investment Banking	Finance	3
15151	Mozart	Music	40,000.00	15151	MU-199	1	Spring	2,010	MU-199	Music Video Production	Music	3
22222	Einstein	Physics	95,000.00	22222	PHY-101	1	Fall	2,009	PHY-101	Physical Principles	Physics	4
32343	El Said	History	60,000.00	32343	HIS-351	1	Spring	2,010	HIS-351	World History	History	3
45565	Katz	Comp. Sci.	75,000.00	45565	CS-101	1	Spring	2,010	CS-101	Intro. to Computer Science	Comp. Sci.	4
45565	Katz	Comp. Sci.	75,000.00	45565	CS-319	1	Spring	2,010	CS-319	Image Processing	Comp. Sci.	3
76766	Crick	Biology	(NULL)	76766	BIO-101	1	Summer	2,009	BIO-101	Intro. to Biology	Biology	4
76766	Crick	Biology	(NULL)	76766	BIO-301	1	Summer	2,010	BIO-301	Genetics	Biology	4
83821	Brandt	Comp. Sci.	92,000.00	83821	CS-190	1	Spring	2,009	CS-190	Game Design	Comp. Sci.	4
83821	Brandt	Comp. Sci.	92,000.00	83821	CS-190	2	Spring	2,009	CS-190	Game Design	Comp. Sci.	4
83821	Brandt	Comp. Sci.	92,000.00	83821	CS-319	2	Spring	2,010	CS-319	Image Processing	Comp. Sci.	3
98345	Kim	Elec. Eng.	80,000.00	98345	EE-181	1	Spring	2,009	EE-181	Intro. to Digital Systems	Elec. Eng.	3

Find the names of all instructors who have taught some course and the course_id, course name of those courses, and year

```
1 SELECT
2    `name`, teaches.course_id, course.title, year
3 FROM
4    instructor , teaches, course
5 WHERE
6    instructor.ID = teaches.ID AND teaches.course_id = course.course_id
```

Result #1 (4×15)

name	course_id	title	year
Srinivasan	CS-101	Intro. to Computer Science	2,009
Srinivasan	CS-315	Robotics	2,010
Srinivasan	CS-347	Database System Concepts	2,009
Wu	FIN-201	Investment Banking	2,010
Mozart	MU-199	Music Video Production	2,010
Einstein	PHY-101	Physical Principles	2,009
El Said	HIS-351	World History	2,010
Katz	CS-101	Intro. to Computer Science	2,010
Katz	CS-319	Image Processing	2,010
Crick	BIO-101	Intro. to Biology	2,009
Crick	BIO-301	Genetics	2,010
Brandt	CS-190	Game Design	2,009
Brandt	CS-190	Game Design	2,009
Brandt	CS-319	Image Processing	2,010
Kim	EE-181	Intro. to Digital Systems	2,009

Combining a relation with itself

Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

instructor (4×12)

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65,000.00
12121	Wu	Finance	90,000.00
15151	Mozart	Music	40,000.00
22222	Einstein	Physics	95,000.00
32343	El Said	History	60,000.00
33456	Gold	Physics	87,000.00
45565	Katz	Comp. Sci.	75,000.00
58583	Califieri	History	62,000.00
76543	Singh	Finance	80,000.00
76766	Crick	Biology	72,000.00
83821	Brandt	Comp. Sci.	92,000.00
98345	Kim	Elec. Eng.	80,000.00

Find the names of all instructors who have a higher salary than some instructor in ‘Comp. Sci’.

```
2 distinct S.name, S.salary, S.dept_name, T.name, T.dept_name, T.salary
3 FROM
4    instructor AS S, instructor AS T
5 WHERE
6    S.salary > T.salary AND T.dept_name='Comp. Sci.'
```

instructor (6×14)

name	salary	dept_name	name	dept_name	salary
Wu	90,000.00	Finance	Srinivasan	Comp. Sci.	65,000.00
Wu	90,000.00	Finance	Katz	Comp. Sci.	75,000.00
Einstein	95,000.00	Physics	Srinivasan	Comp. Sci.	65,000.00
Einstein	95,000.00	Physics	Katz	Comp. Sci.	75,000.00
Einstein	95,000.00	Physics	Brandt	Comp. Sci.	92,000.00
Gold	87,000.00	Physics	Srinivasan	Comp. Sci.	65,000.00
Gold	87,000.00	Physics	Katz	Comp. Sci.	75,000.00
Katz	75,000.00	Comp. Sci.	Srinivasan	Comp. Sci.	65,000.00
Singh	80,000.00	Finance	Srinivasan	Comp. Sci.	65,000.00
Singh	80,000.00	Finance	Katz	Comp. Sci.	75,000.00
Brandt	92,000.00	Comp. Sci.	Srinivasan	Comp. Sci.	65,000.00
Brandt	92,000.00	Comp. Sci.	Katz	Comp. Sci.	75,000.00
Kim	80,000.00	Elec. Eng.	Srinivasan	Comp. Sci.	65,000.00
Kim	80,000.00	Elec. Eng.	Katz	Comp. Sci.	75,000.00

The Rename Operation

- The SQL allows renaming relations and attributes using the as clause:

  old-name **as** new-name

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

```
select
        distinct  S.name
from
        instructor as S, instructor as T
where
        S.salary > T.salary and T.dept_name = 'Comp. Sci.'
```

- Keyword as is optional and may be omitted

  instructor **as** T ≡ instructor T

The Natural Join

- To make the life of an SQL programmer easier for this common case, SQL supports an operation called the **natural join**

```
select name, course_id
from instructor, teaches
where instructor.ID= teaches.ID;
```

```
select name, course_id
from instructor natural join teaches;
```

```
select A1, A2, . . . , An
from r1 natural join r2 natural join . . . natural join rm
where P;
```

The Natural Join

- To make the life of an SQL programmer easier for this common case, SQL supports an operation called the **natural join**

```
SELECT
    `name`, teaches.course_id, course.title, `year`
FROM
    instructor, teaches, course
WHERE
    instructor.ID = teaches.ID AND teaches.course_id = course.course_id
```

```
SELECT
    `name`, teaches.course_id, course.title, `year`
FROM
    FROM instructor NATURAL JOIN  teaches NATURAL JOIN course
```

Demo.

End.