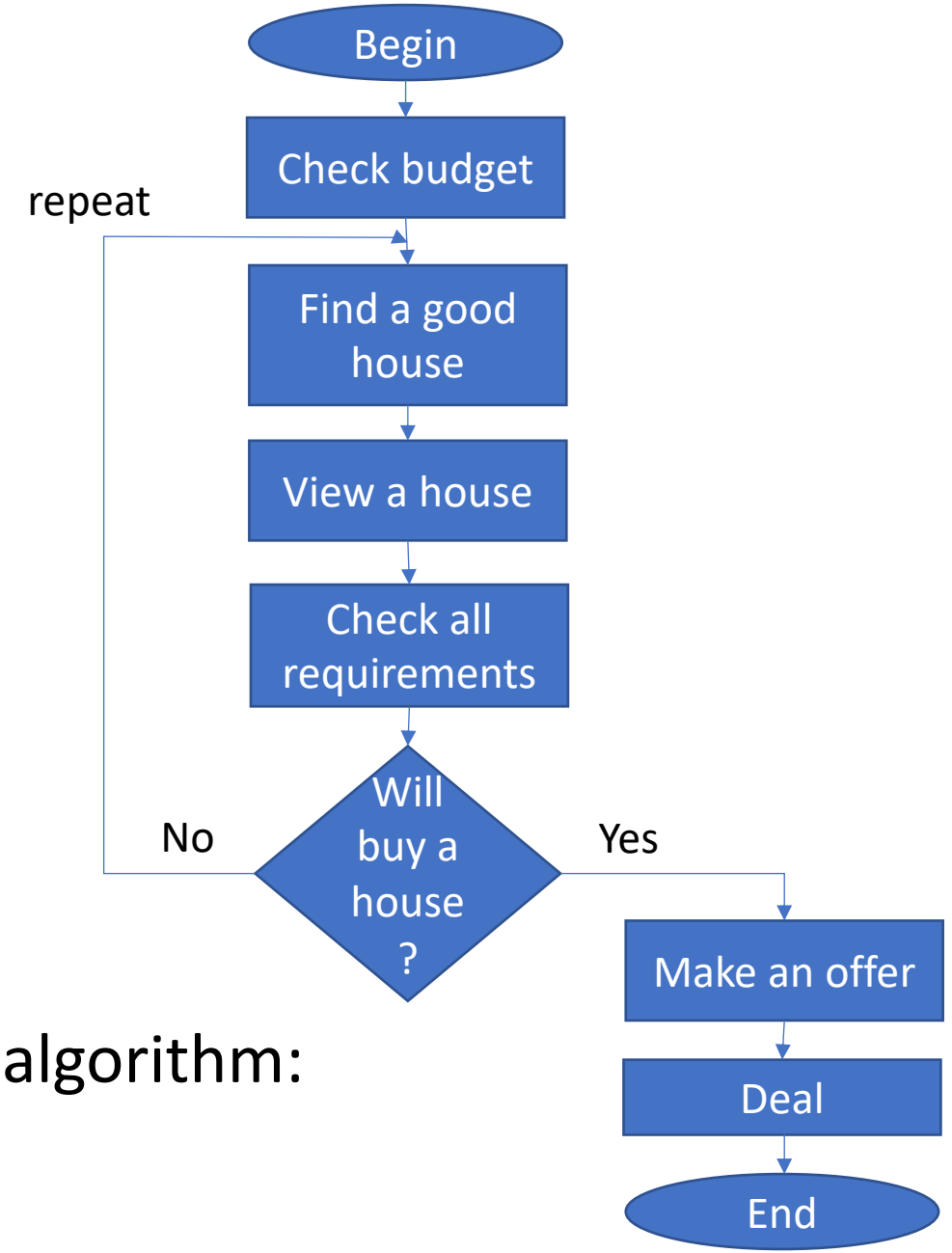
Introduction to databases

What is a Database?

A database is **information that is set up for easy access, management and updating**. Computer databases typically store aggregations of data records or files that contain information, such as sales transactions, customer data, financials and product information.

Example 1: algorithm of buying a house



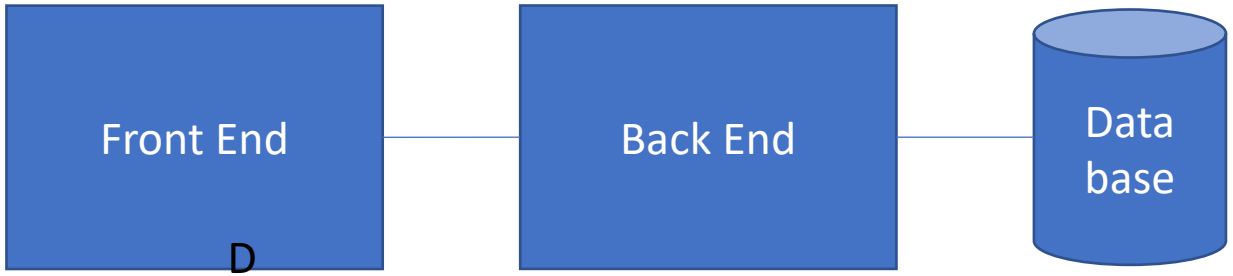
You need to by a house.
How will you do it: will create a simple algorithm: linear steps + loop.

Application for house buying

- Functional requirements :
  want to have an application which will collect information about interesting houses:
  - address,
  - price,
  - bedrooms,
  - bathrooms,
  - floors,
  - area in sq. feet,
  - land (sq feet),
  - comments,
  - score (1..5),
  - status (New, plan to Visit, Viewed, Offer, Cancel, Done).

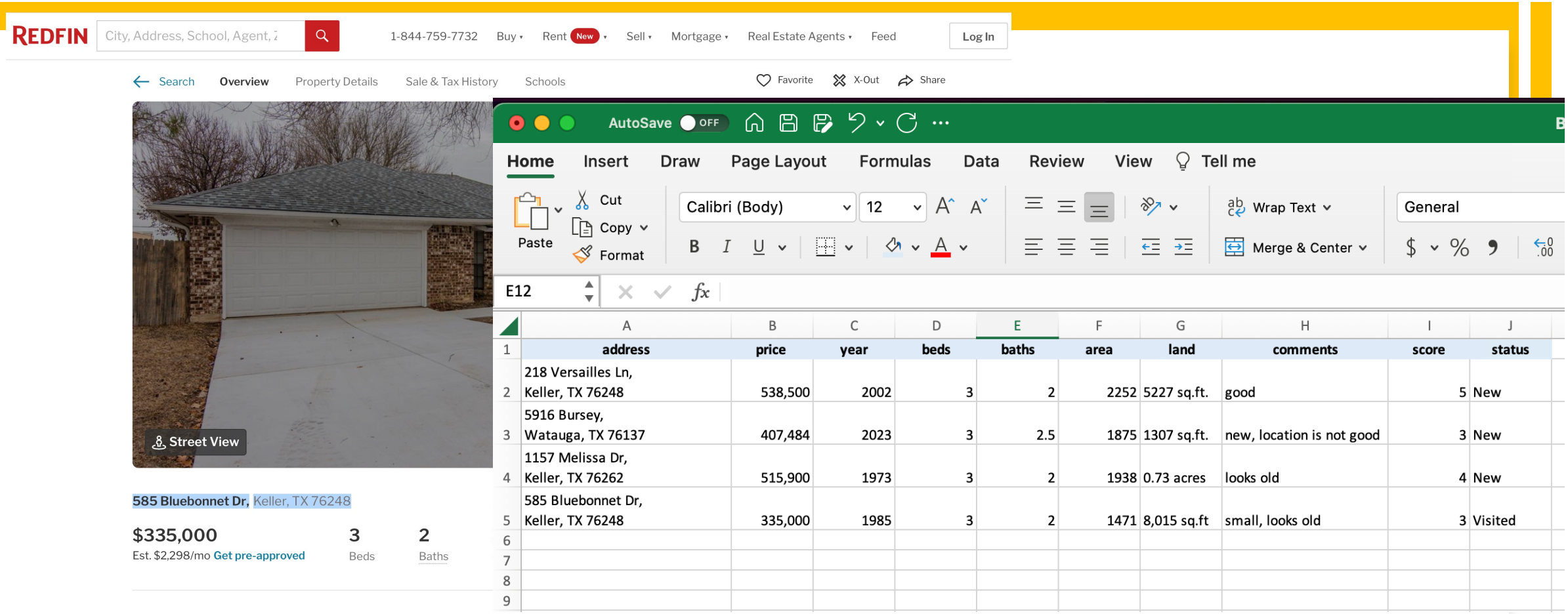
  Want to have a small application with no authorization (anyone could see it by the link), anyone could add/update/delete information about houses.
  Possible to filter by status. Search by address.
- Non-Functional requirements : cloud agnostic application with simple interface.

Application

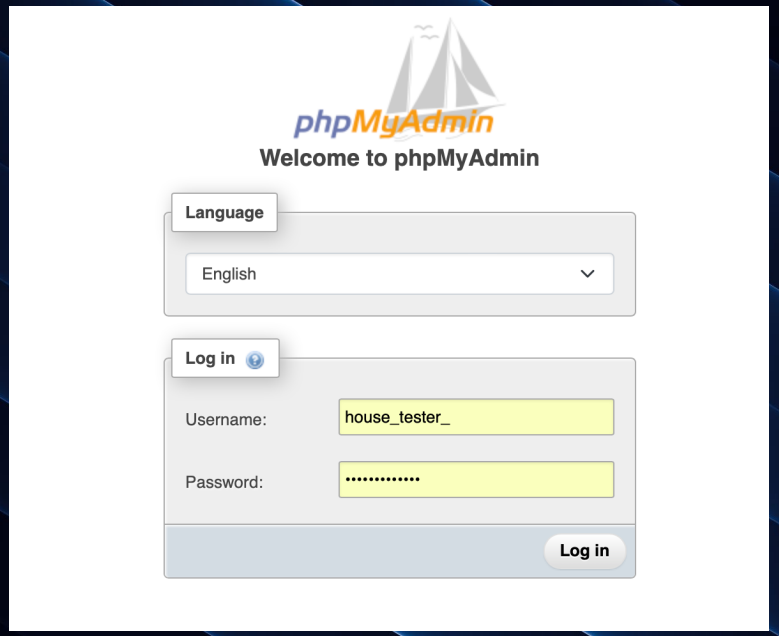


- Analog the application: Excel table with columns

Automation in Excel



	A	B	C	D	E	F	G	H	I	J
1	address	price	year	beds	baths	area	land	comments	score	status
2	218 Versailles Ln, Keller, TX 76248	538,500	2002	3	2	2252	5227 sq.ft.	good	5	New
3	5916 Bursey, Watauga, TX 76137	407,484	2023	3	2.5	1875	1307 sq.ft.	new, location is not good	3	New
4	1157 Melissa Dr, Keller, TX 76262	515,900	1973	3	2	1938	0.73 acres	looks old	4	New
5	585 Bluebonnet Dr, Keller, TX 76248	335,000	1985	3	2	1471	8,015 sq.ft	small, looks old	3	Visited
6										
7										
8										
9										



Signup

By registering for a db4free.net account you agree that:

- db4free.net is a testing environment
- db4free.net is not suitable for production
- if you decide to use your db4free.net database in production despite the warnings, you do that at your own risk (very frequent backups are highly recommended)
- data loss and outages can happen at any time (any complaints about that will likely be ignored)
- the db4free.net team is not granting any warranty or liability of any kind
- the db4free.net team reserves the right to delete databases and/or accounts at any time without notic
- it is up to you to get the latest information from Twitter and the db4free.net blog
- db4free.net provides only a MySQL database, but no web space (there is nowhere to upload any files)

Further:

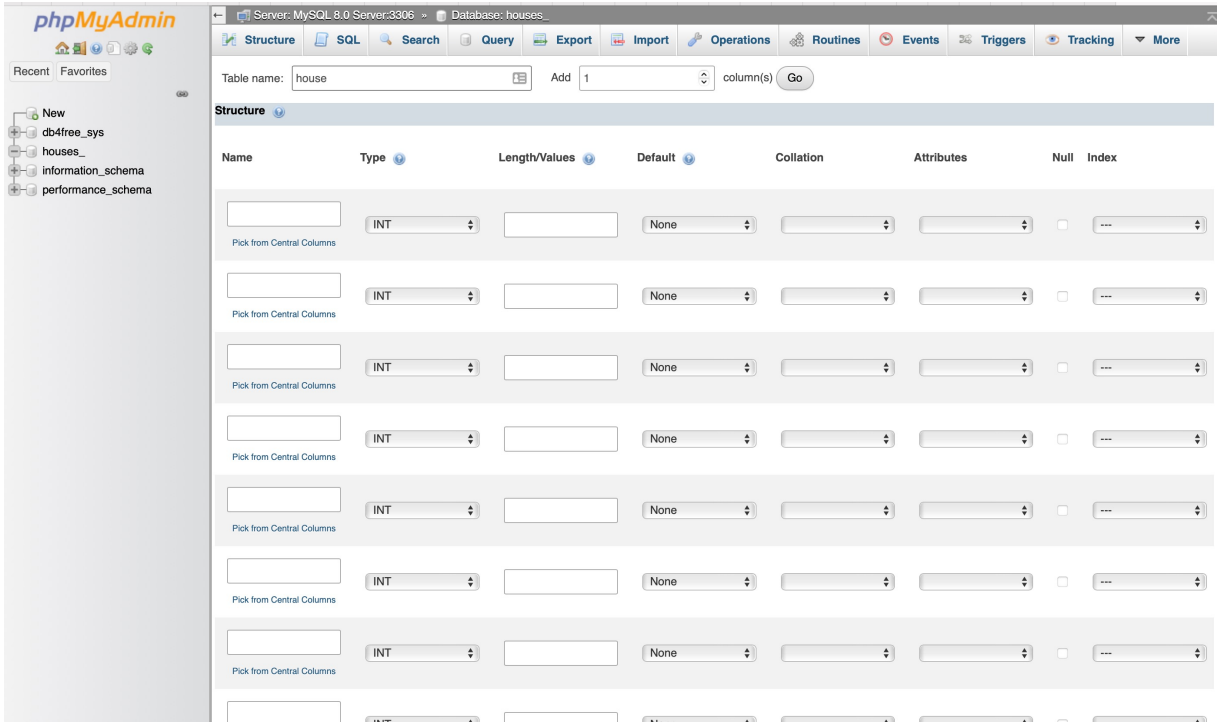
- db4free.net is a service for testing, not for hosting. Databases that store more than 200 MB data will b cleared at irregular intervals without notification
- Please remove data which you no longer need, or delete your no longer needed account. This makes it easier to recover if a server crash occurs.

MySQL database name:	houses
MySQL username:	test
MySQL user password:	••••••••
MySQL user password verification:	••••••••
Email address:	np@cdb.systems

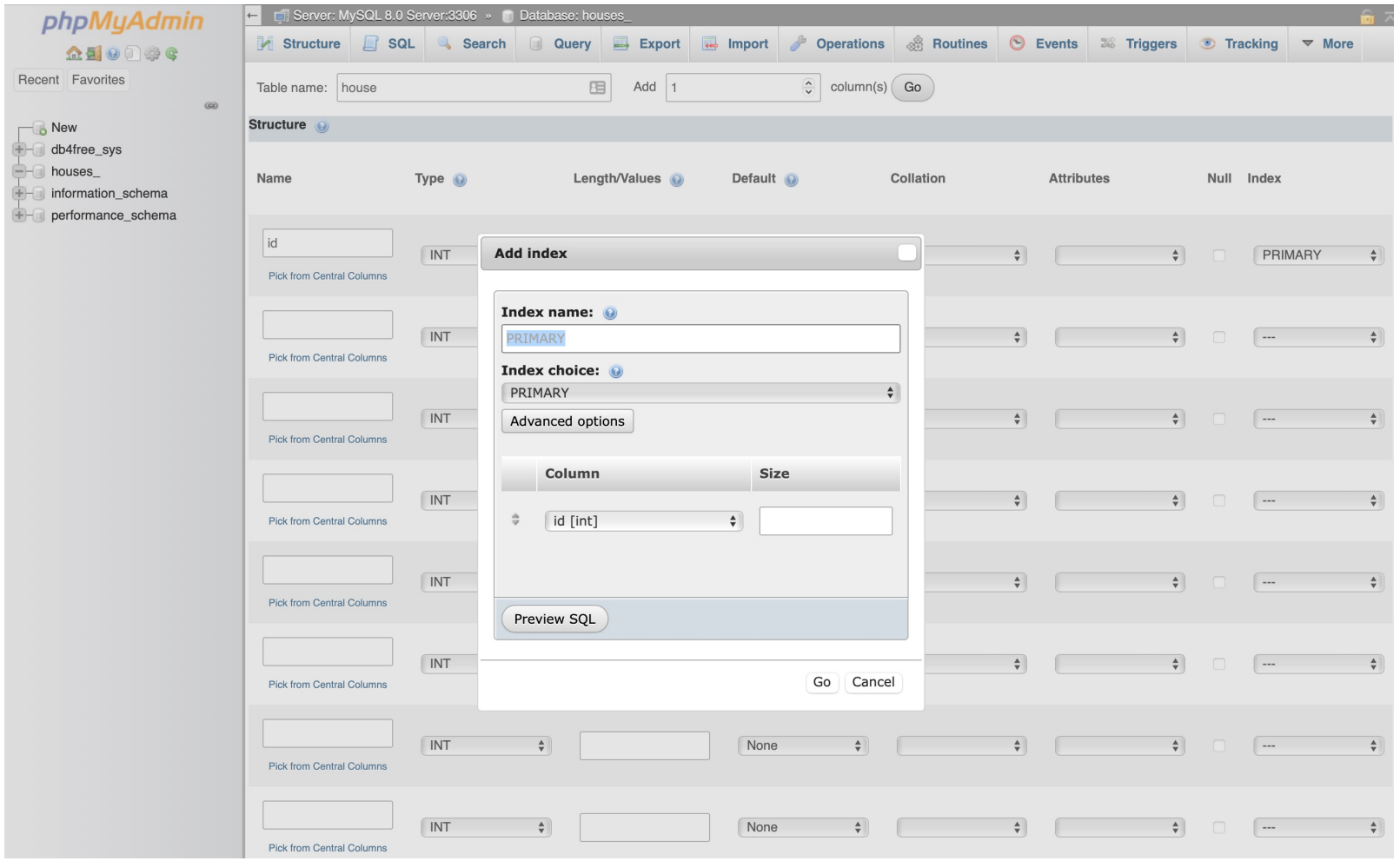
Database

https://www.db4free.net/signup.php

CREATE A TABLE

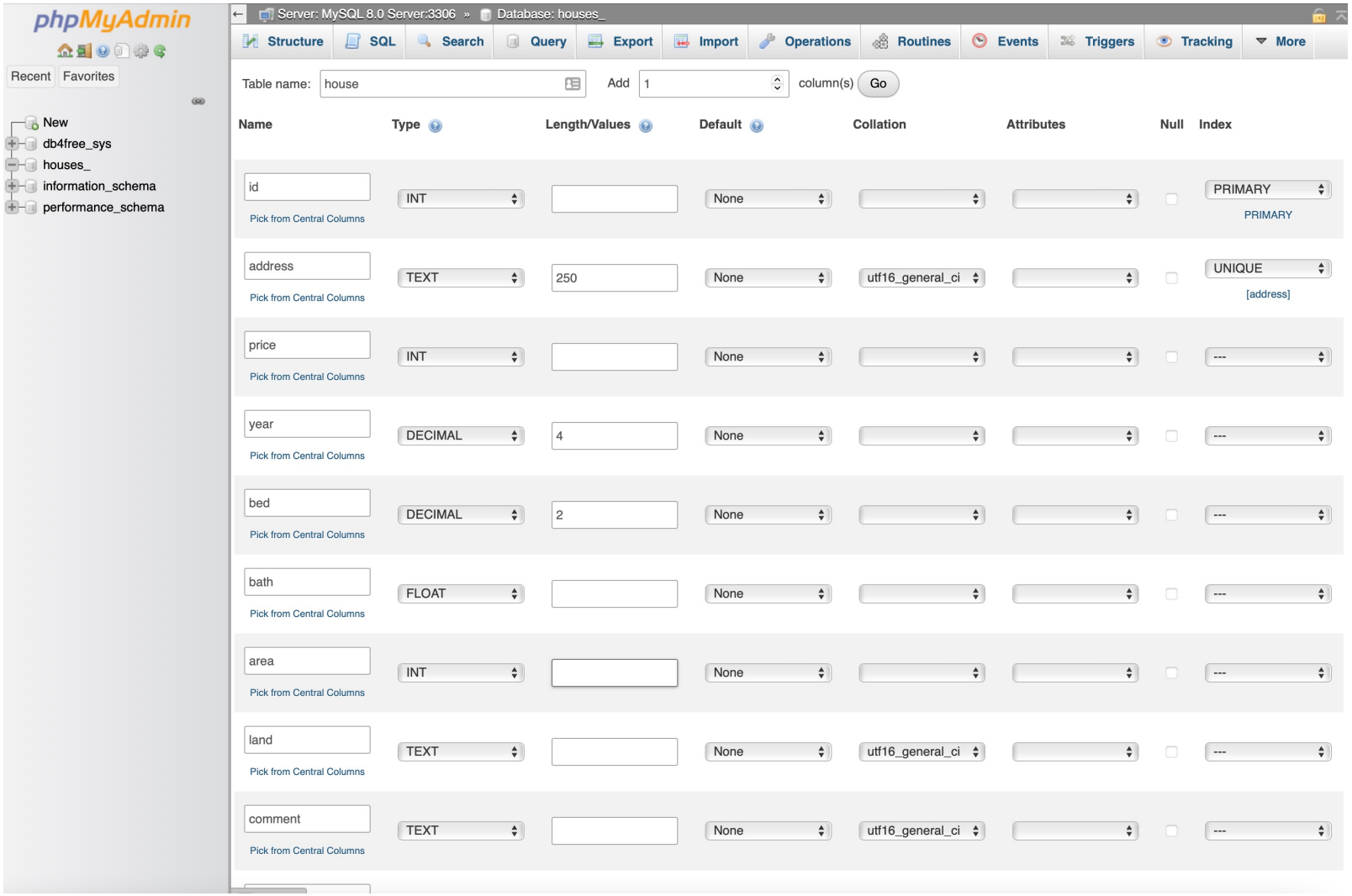


Create a table with the same columns as we have in Excel table.



Add a unique identifier

Will add a column “id” (means identity) type Int (integer)



Add all columns

Create a table with SQL language

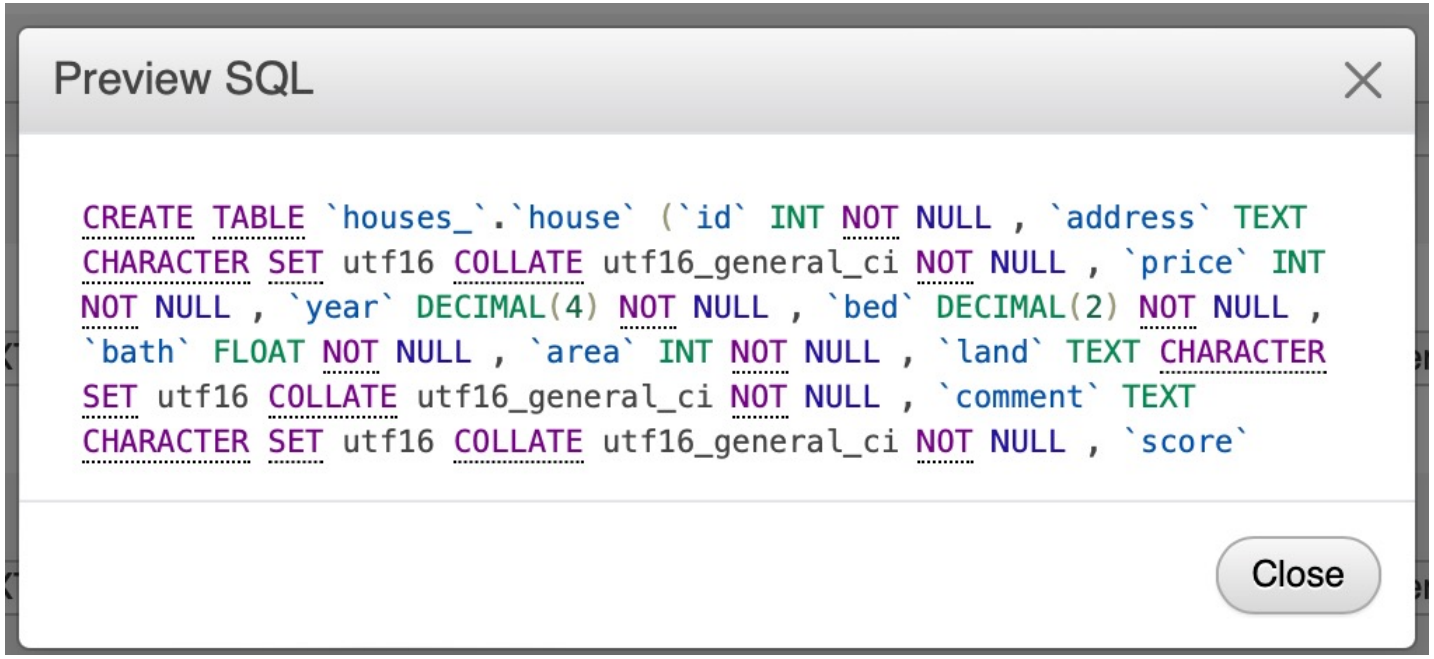
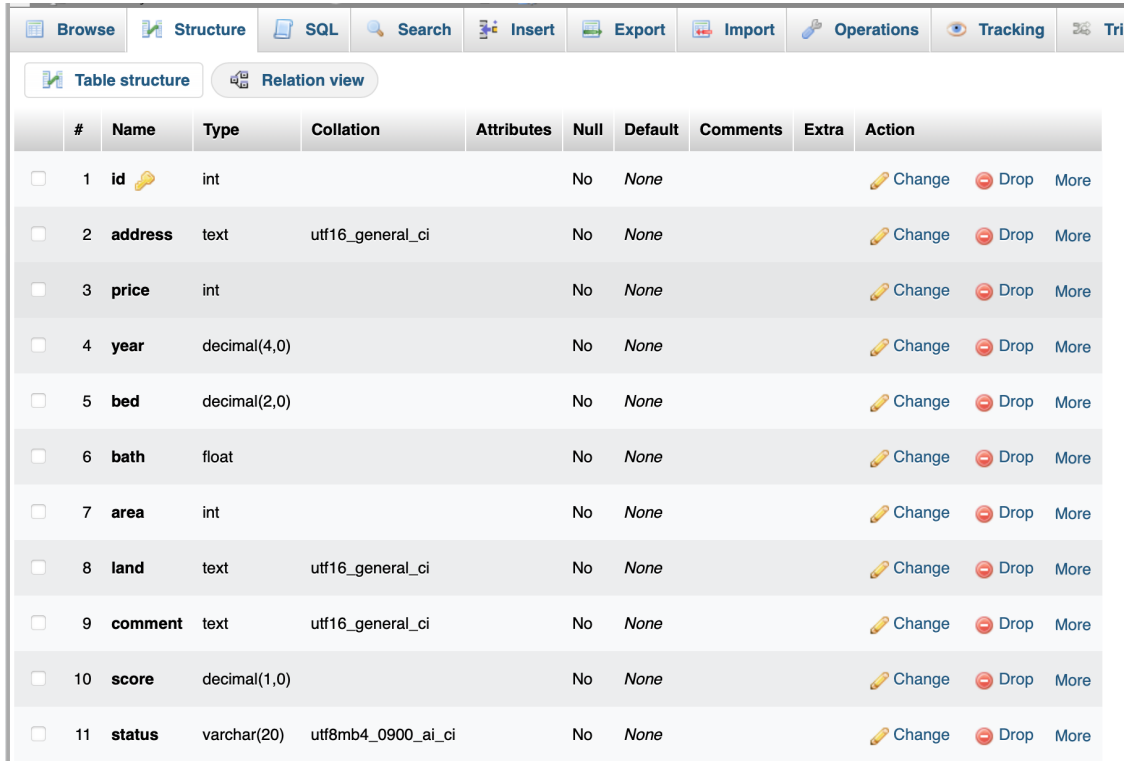


Table is created



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int			No	None			Change Drop More
2	address	text	utf16_general_ci		No	None			Change Drop More
3	price	int			No	None			Change Drop More
4	year	decimal(4,0)			No	None			Change Drop More
5	bed	decimal(2,0)			No	None			Change Drop More
6	bath	float			No	None			Change Drop More
7	area	int			No	None			Change Drop More
8	land	text	utf16_general_ci		No	None			Change Drop More
9	comment	text	utf16_general_ci		No	None			Change Drop More
10	score	decimal(1,0)			No	None			Change Drop More
11	status	varchar(20)	utf8mb4_0900_ai_ci		No	None			Change Drop More

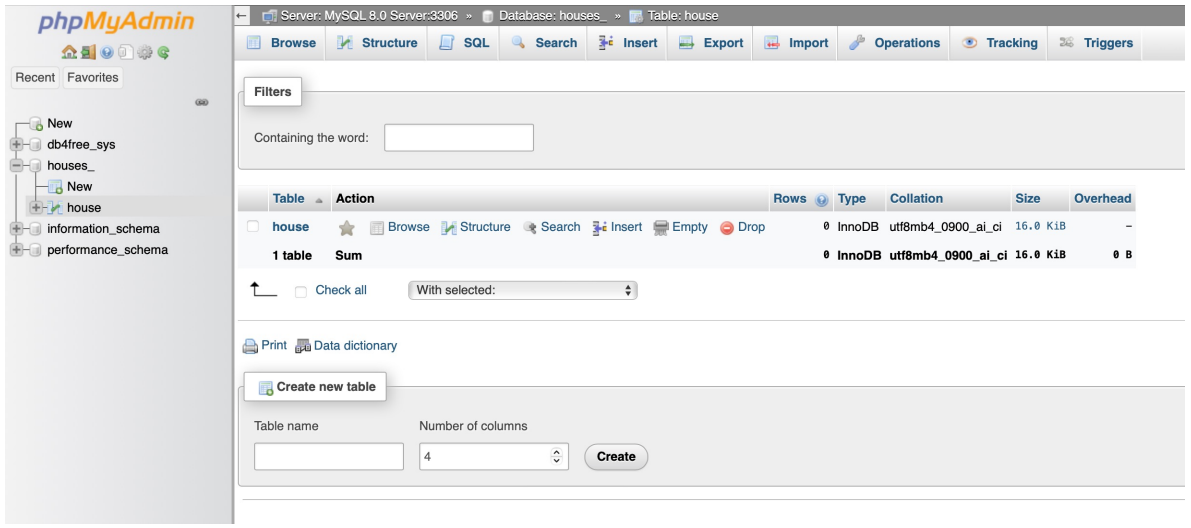
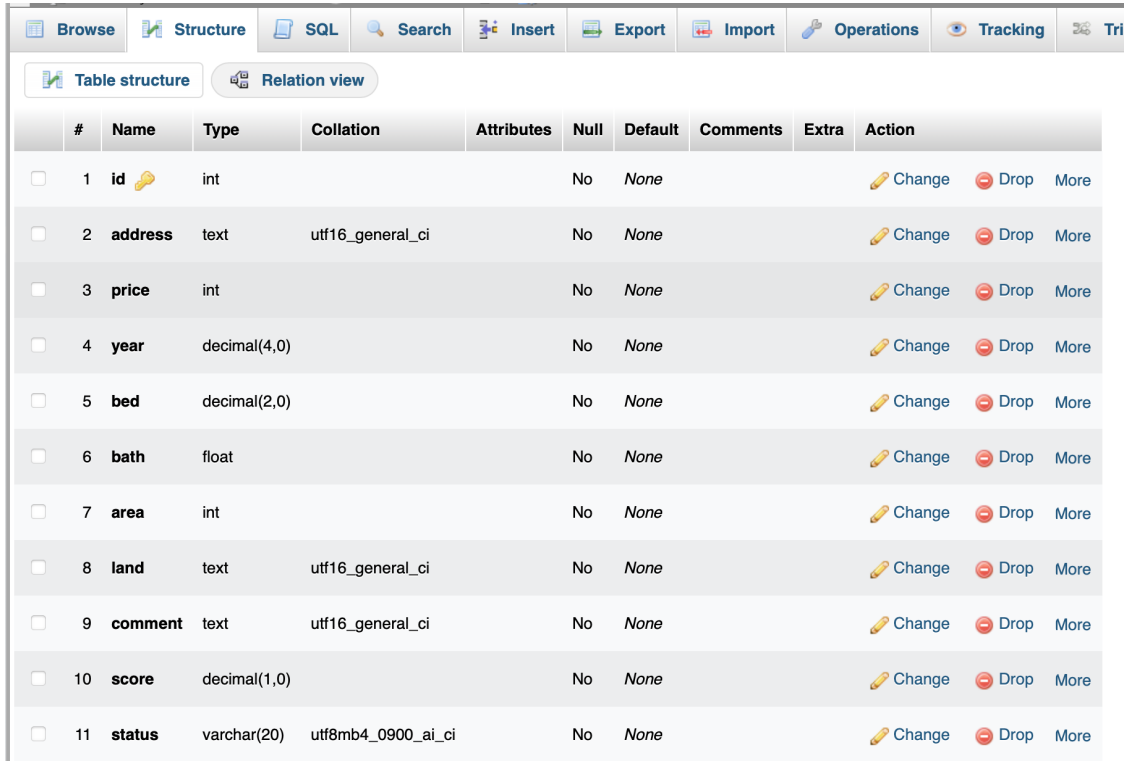
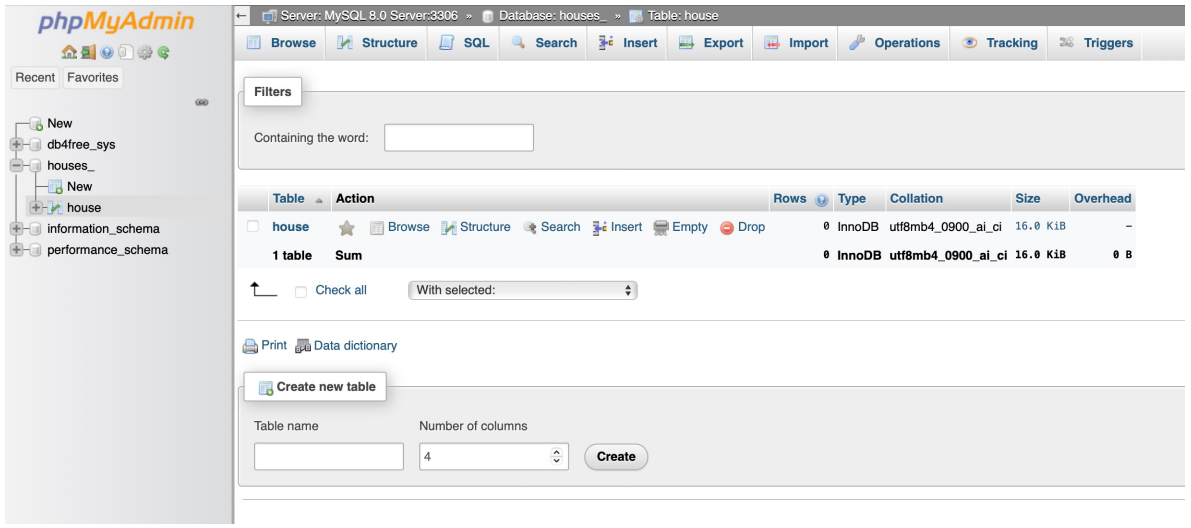
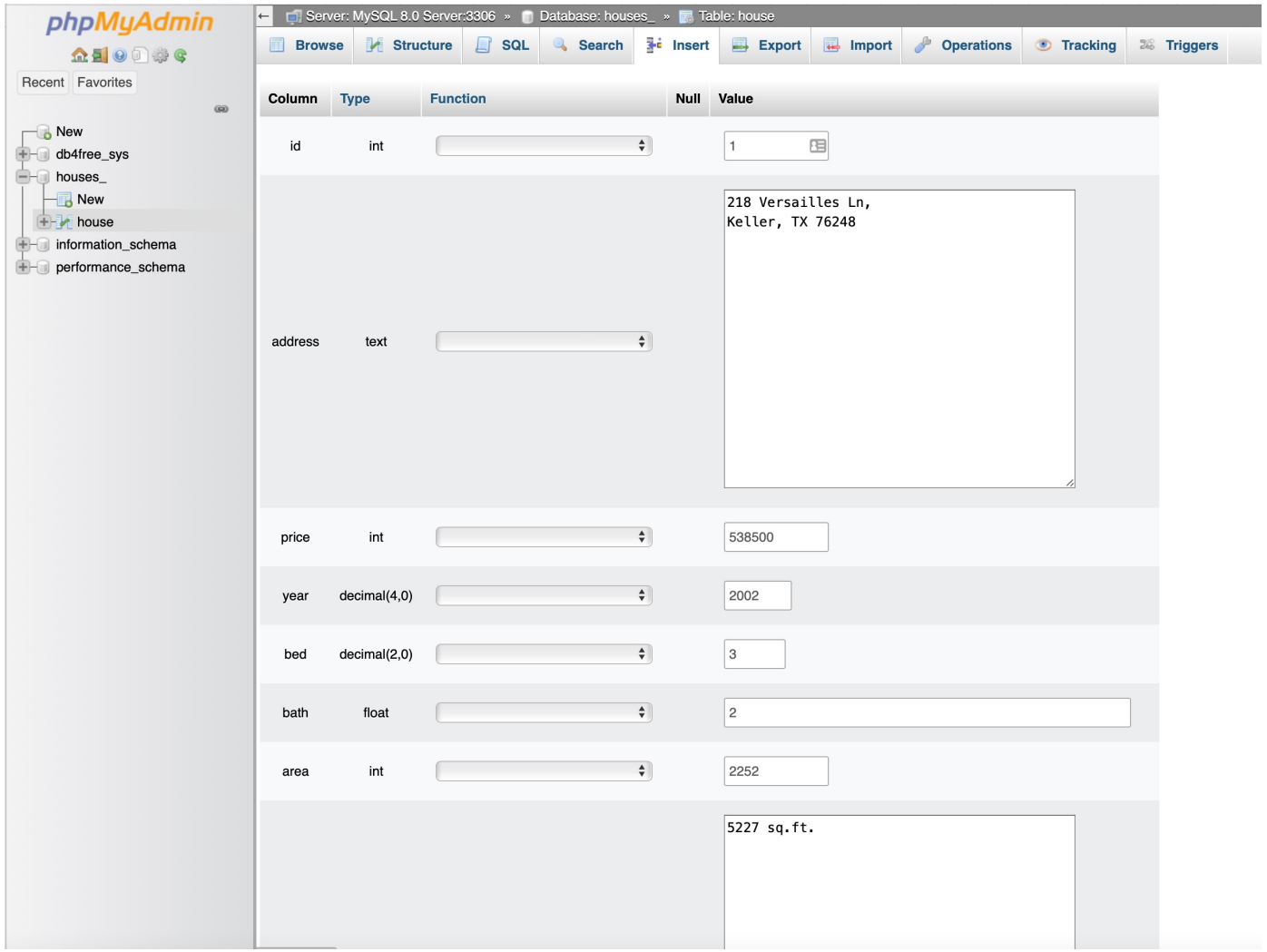


Table is created

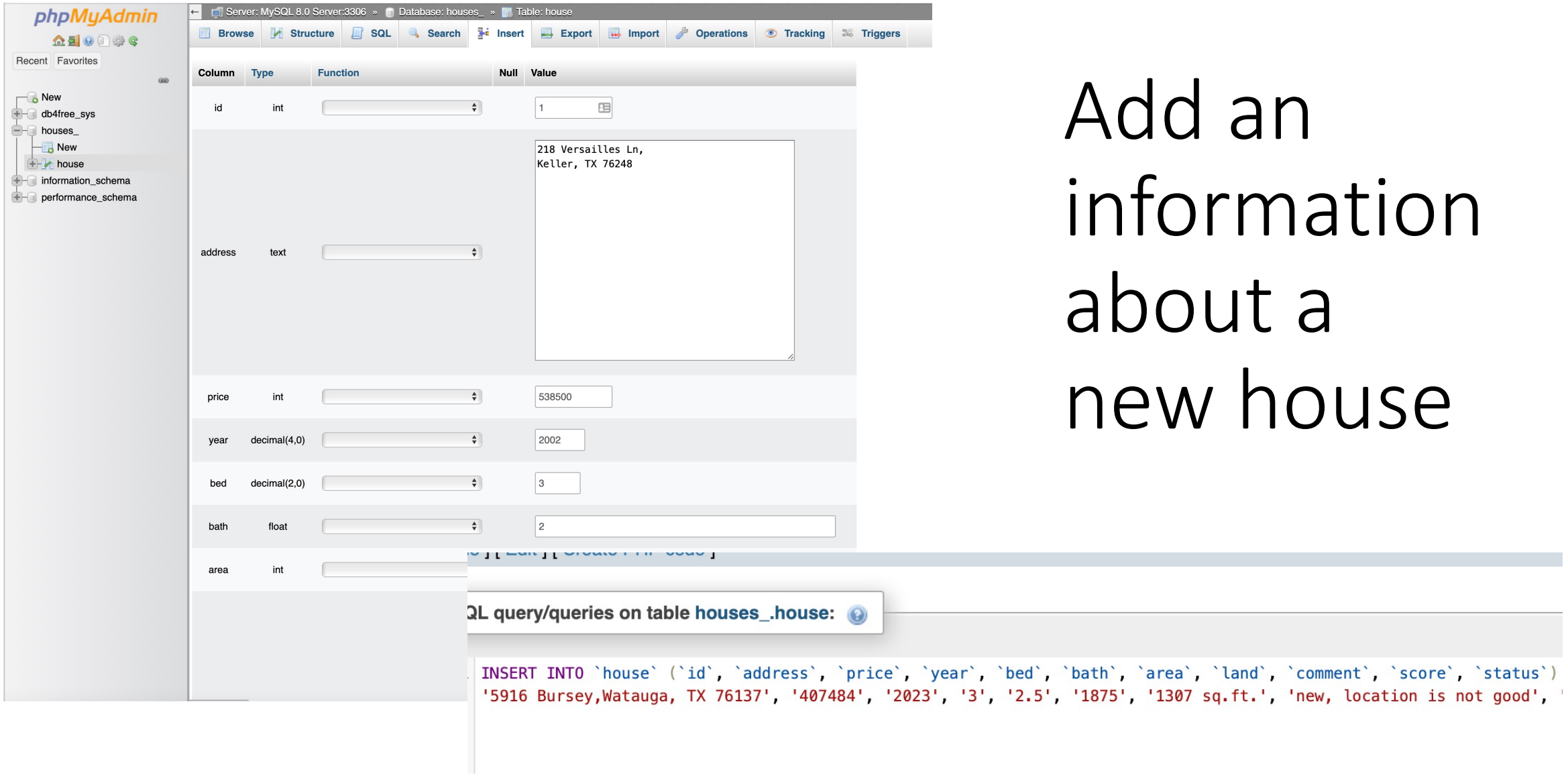


#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int			No	None			Change Drop More
2	address	text	utf16_general_ci		No	None			Change Drop More
3	price	int			No	None			Change Drop More
4	year	decimal(4,0)			No	None			Change Drop More
5	bed	decimal(2,0)			No	None			Change Drop More
6	bath	float			No	None			Change Drop More
7	area	int			No	None			Change Drop More
8	land	text	utf16_general_ci		No	None			Change Drop More
9	comment	text	utf16_general_ci		No	None			Change Drop More
10	score	decimal(1,0)			No	None			Change Drop More
11	status	varchar(20)	utf8mb4_0900_ai_ci		No	None			Change Drop More

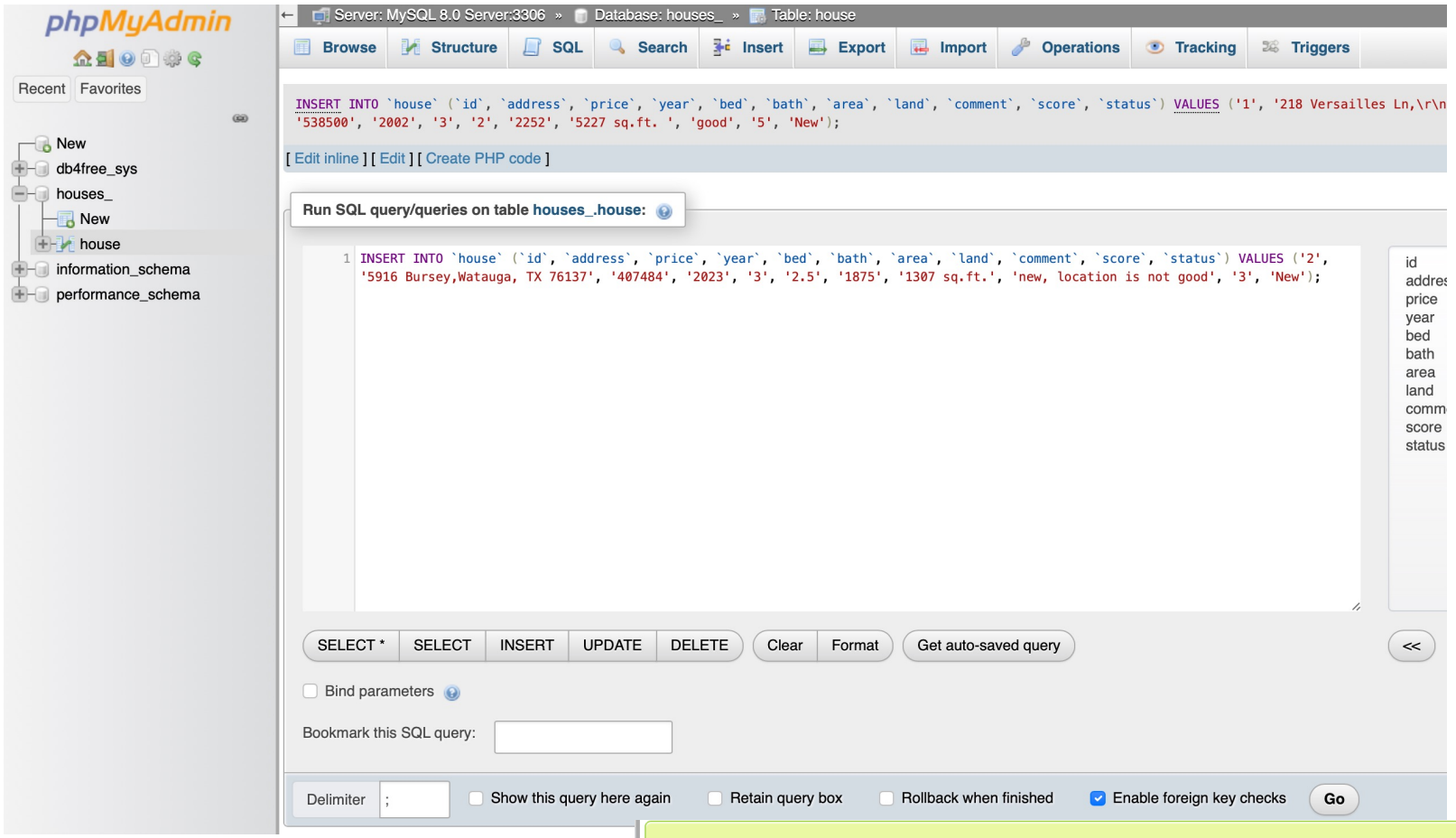




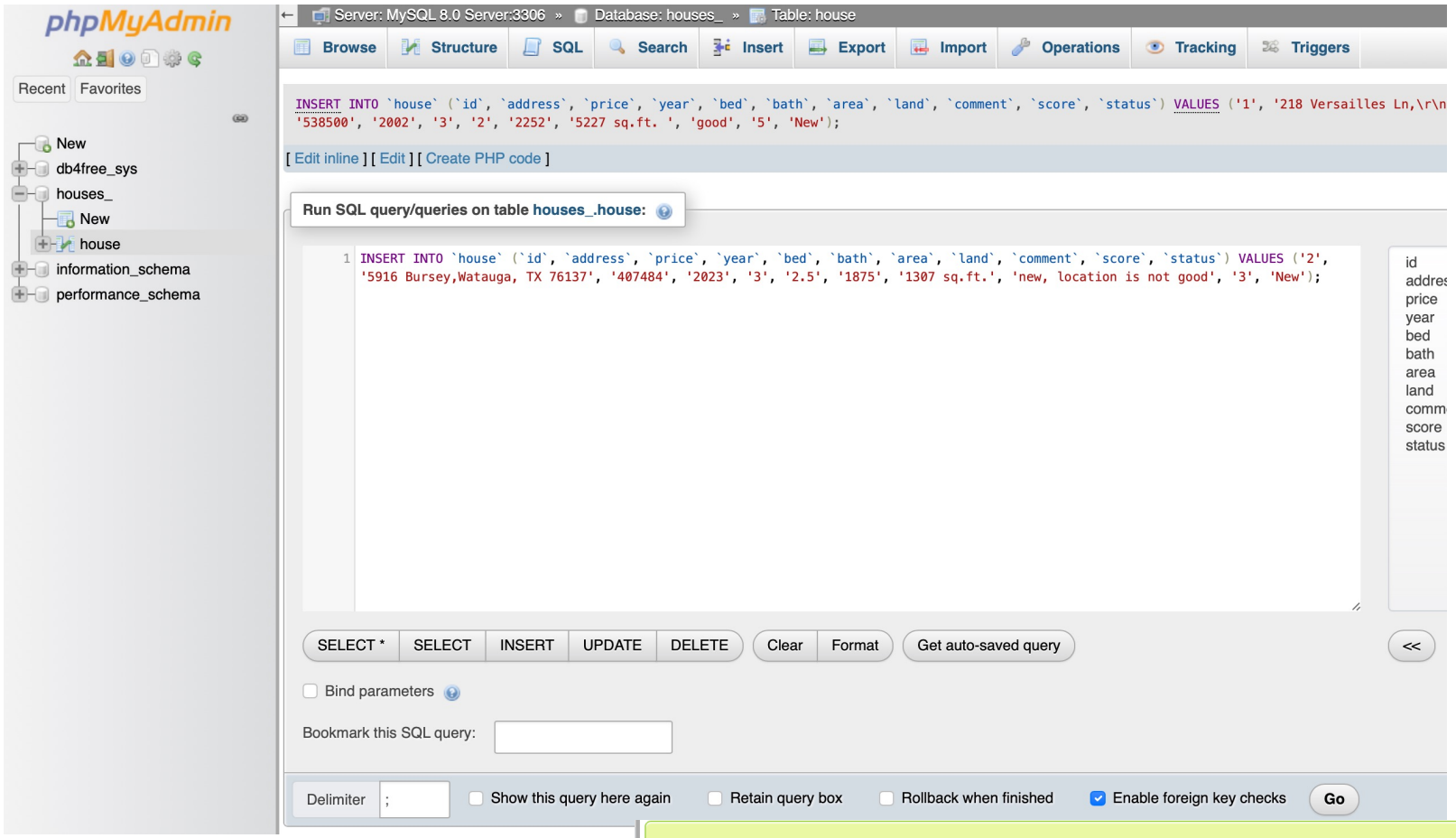
Add an information about a new house



Add an information about a new house



Add an information about a new house using SQL



Add an information about a new house using SQL

Table with data

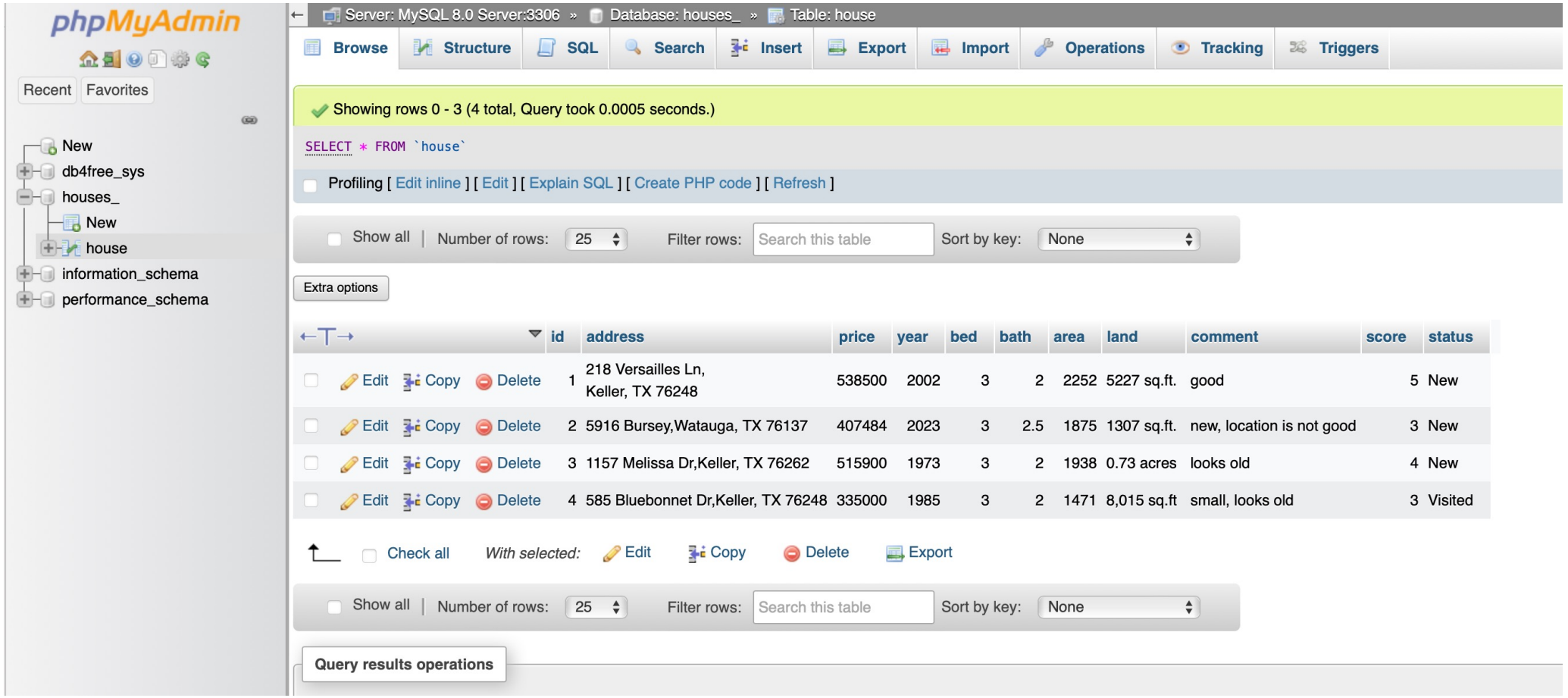
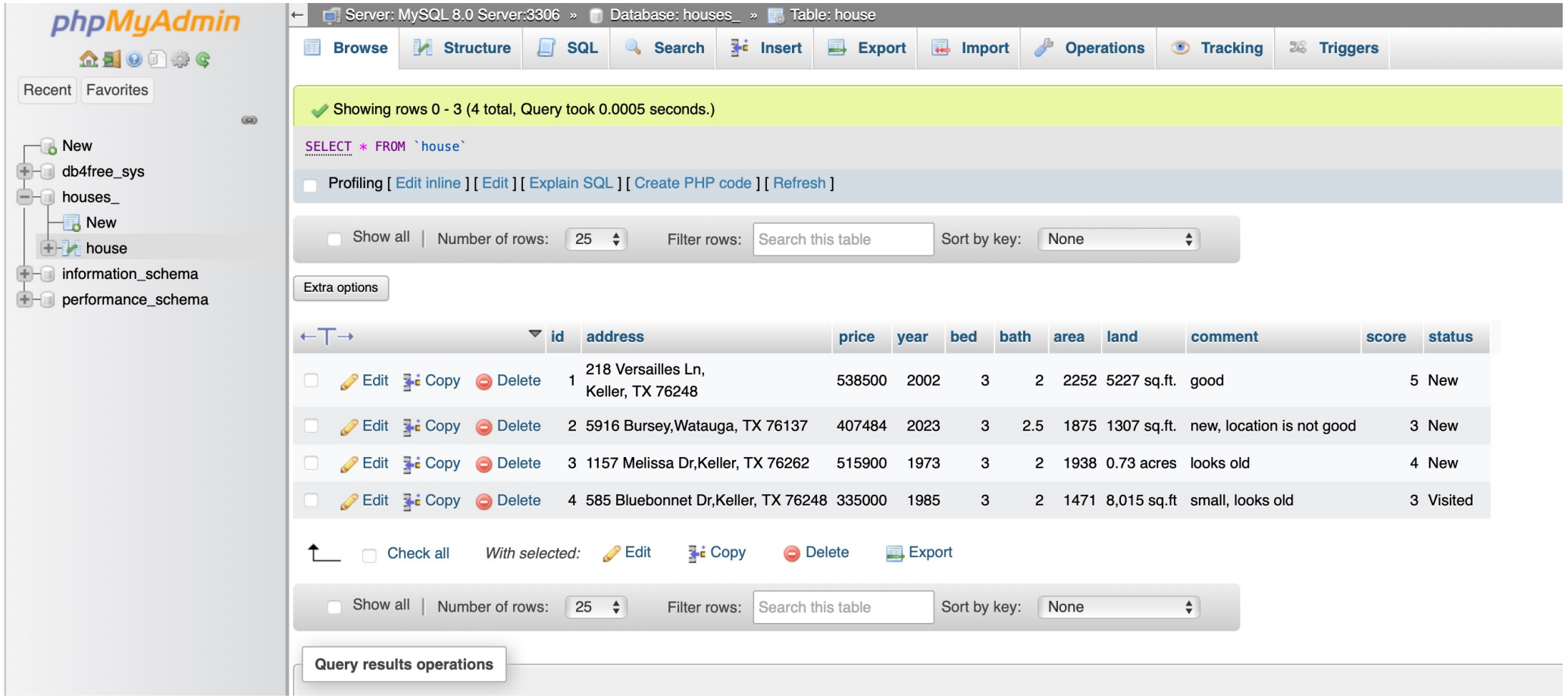
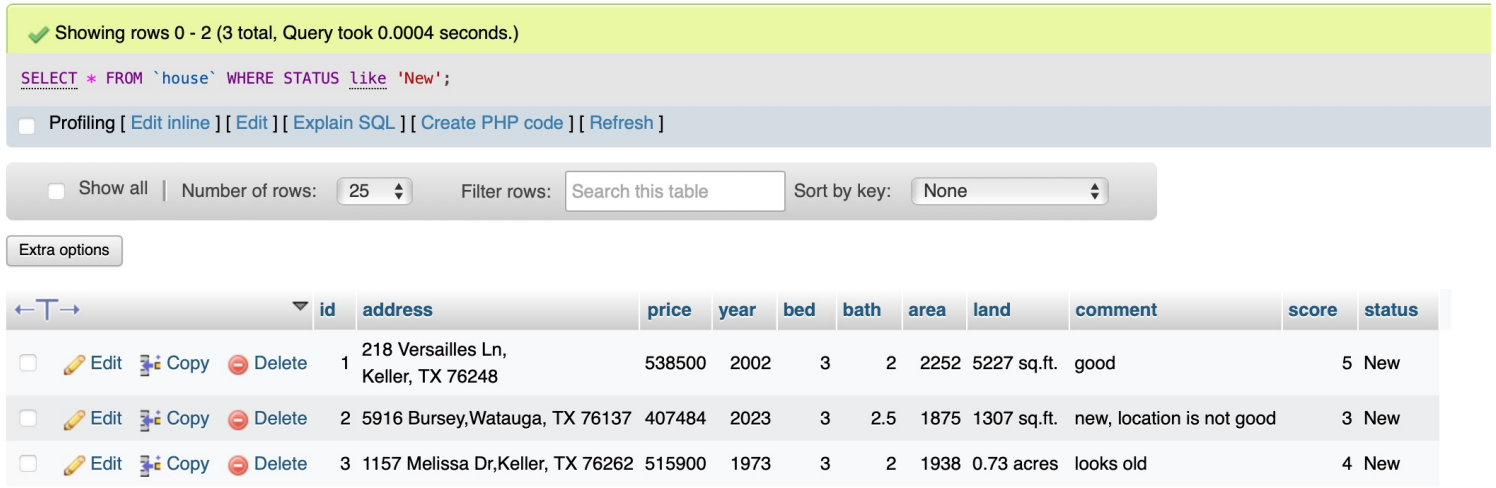
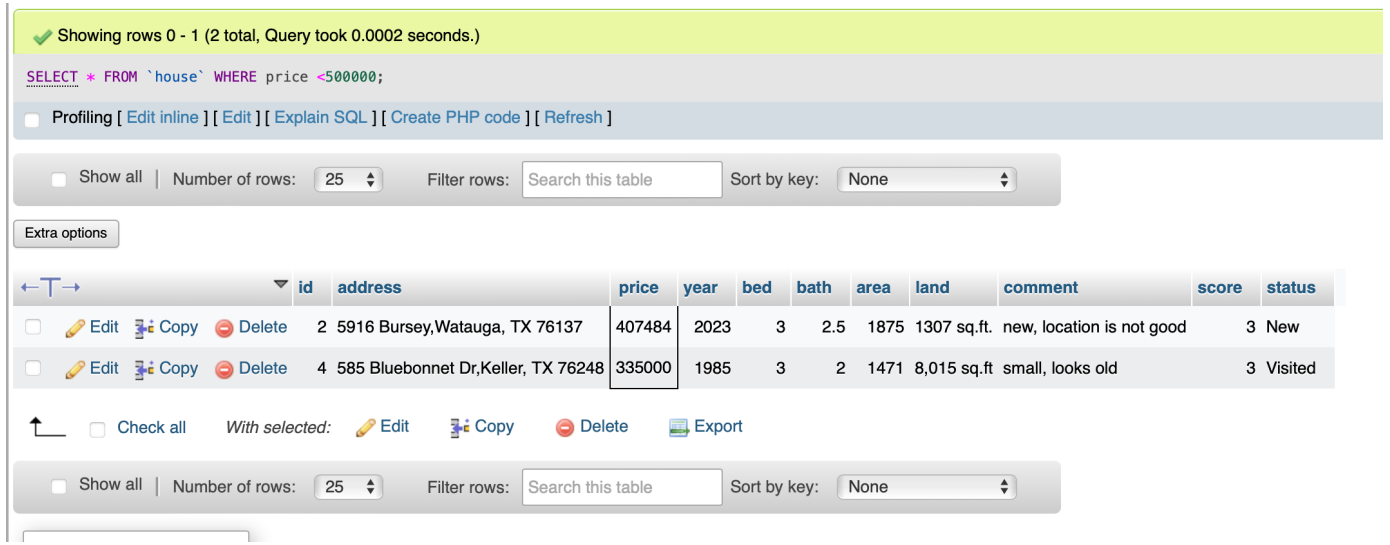


Table with data



Data by conditions



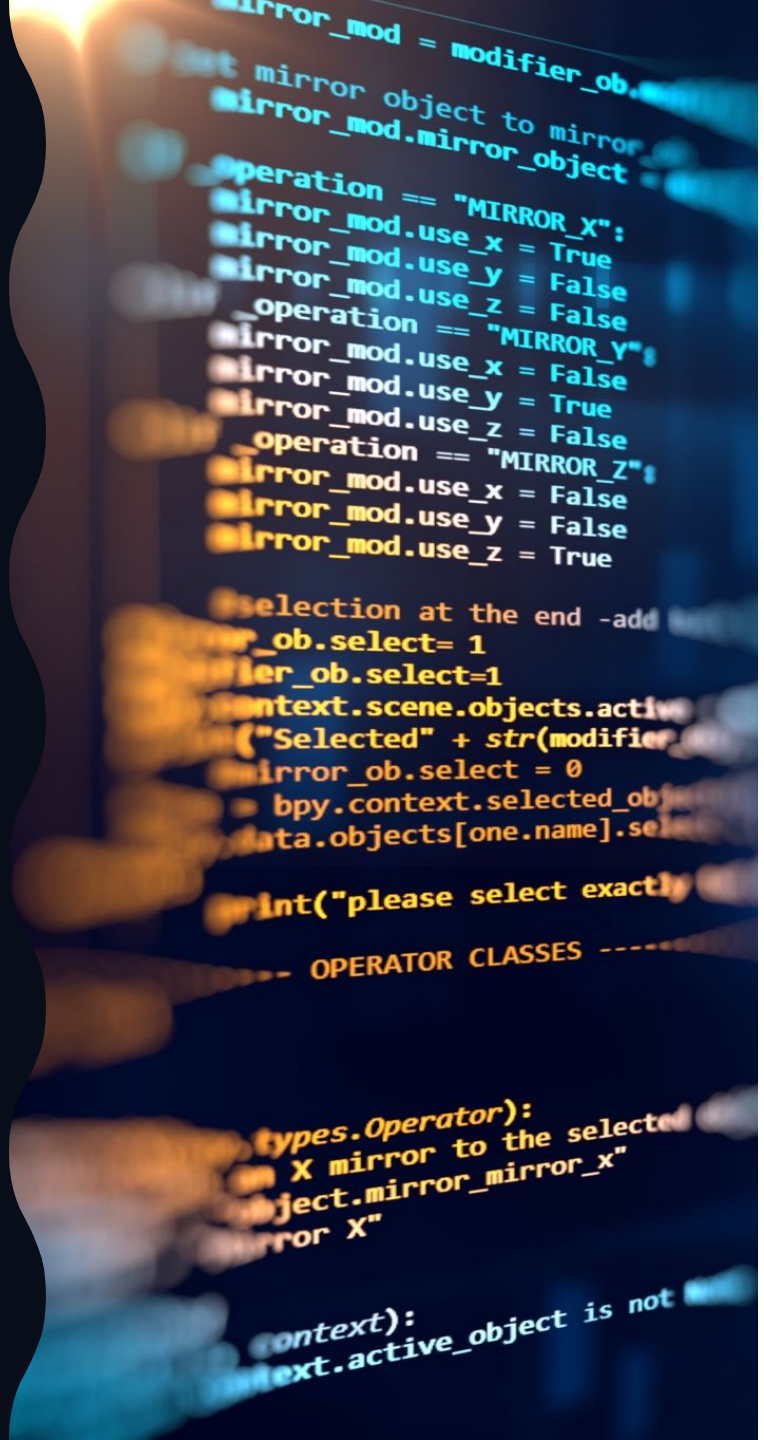


SQL databases

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

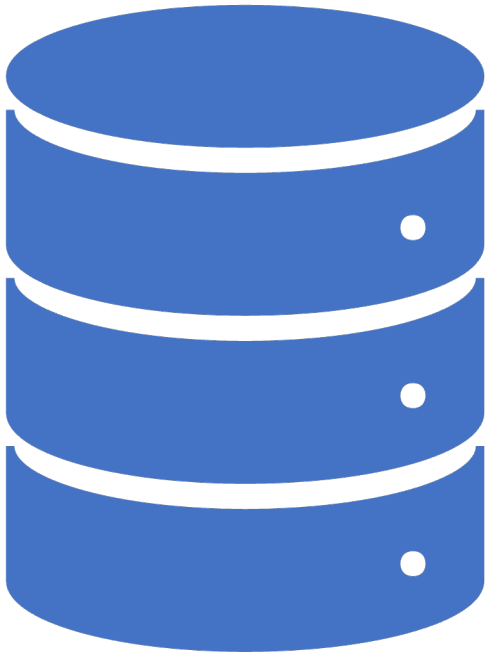
SQL can do

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views



Using SQL in simple Web Site

- To build a web site that shows data from a database, you will need:
- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like PHP or ASP
- To use SQL to get the data you want
- To use HTML / CSS to style the page



RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.
- Look at the "Customers" table:
  `SELECT * FROM Customers;`

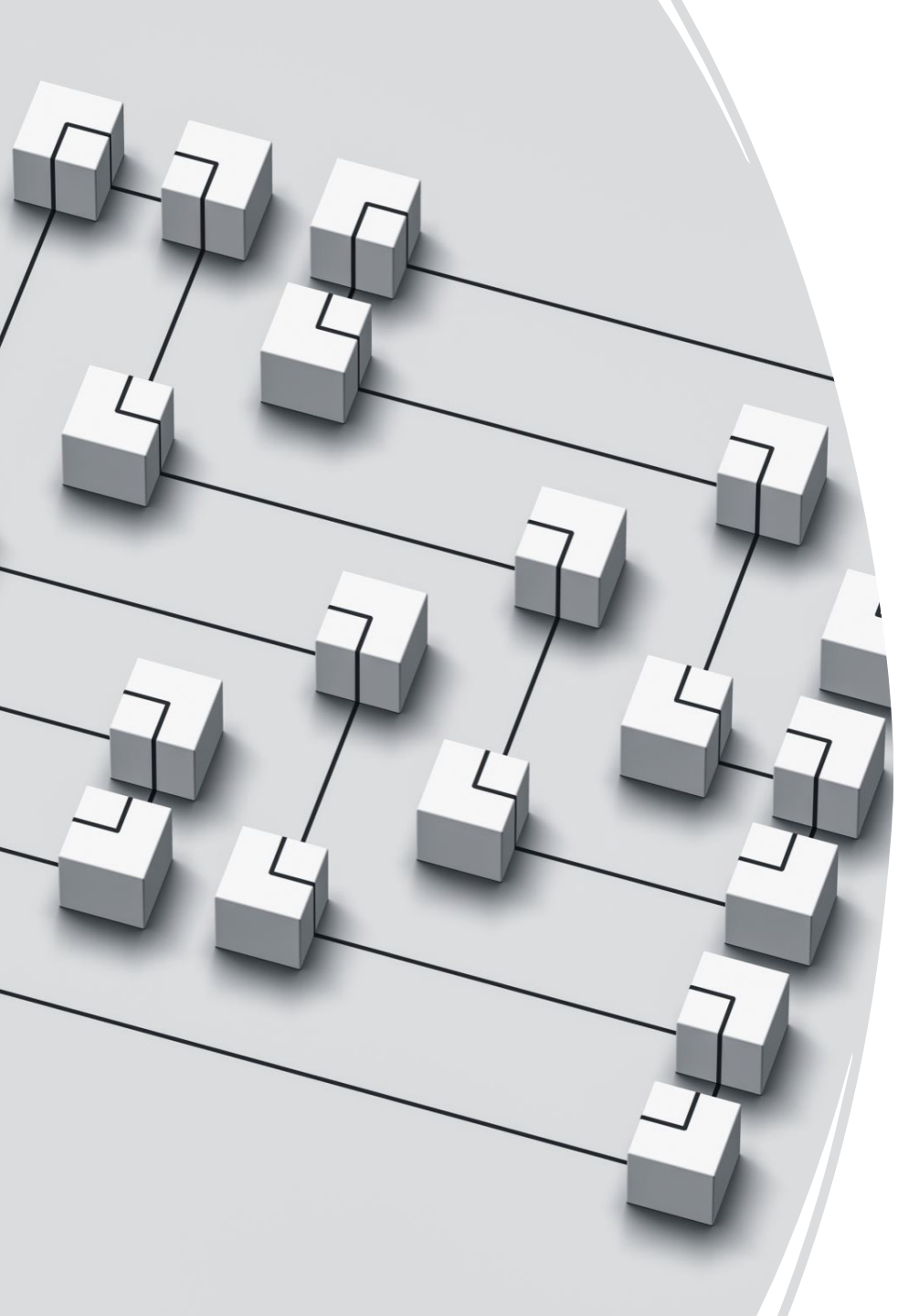
Columns

- Every table is broken up into smaller entities called fields. The fields in the Customers table consist of CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country. A field is a column in a table that is designed to maintain specific information about every record in the table.
- A record, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.
- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

SQL Statements

- Most of the actions you need to perform on a database are done with SQL statements.
- The following SQL statement selects all the records in the "Customers" table:
- Example

```
SELECT * FROM Customers;
```



Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2, ...
FROM table_name;
```

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

```
SELECT CustomerName, City FROM Customers;
```

DISTINCT

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

```
SELECT distinct Country FROM Customers;
```

COUNT

The following SQL statement lists the number of different (distinct) customer countries:

Example

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

WHERE

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

Text Fields vs. Numeric Fields

Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

Operators in The WHERE Clause

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

AND, OR and NOT Operators

The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

•The AND operator displays a record if all the conditions separated by AND are TRUE.

•The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

ORDER BY Keyword

The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2,
... ASC|DESC;
```

```
SELECT * FROM Customers
ORDER BY Country;
```

INSERT INTO Statement

he INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

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

1. Specify both the column names and the values to be inserted:

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

2. 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. Here, the INSERT INTO syntax would be as follows:

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

Null

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

UPDATE

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

DELETE

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement.
The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

TOP, LIMIT, FETCH FIRST or ROWNUM

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST *n* ROWS ONLY and ROWNUM.

MySQL Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

MIN() and MAX() Functions

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

LIKE Operator

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

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

The percent sign (%) represents zero, one, or multiple characters

The underscore sign (_) represents one, single character

Note: MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).

Tip: You can also combine any number of conditions using AND or OR operators.

Like

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL 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.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

or:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

```
SELECT column_name AS alias_name
FROM table_name;
```

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

SQL Joins

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

ORDERS:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CUSTOMERS:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

JOIN TWO TABLES

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

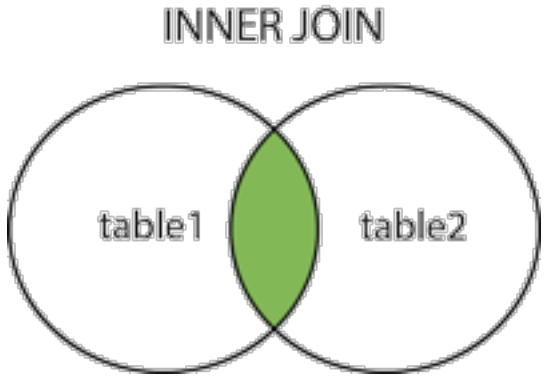
Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

```
SELECT Orders.OrderID,
Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID
=Customers.CustomerID;
```

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.



INNER JOIN Syntax

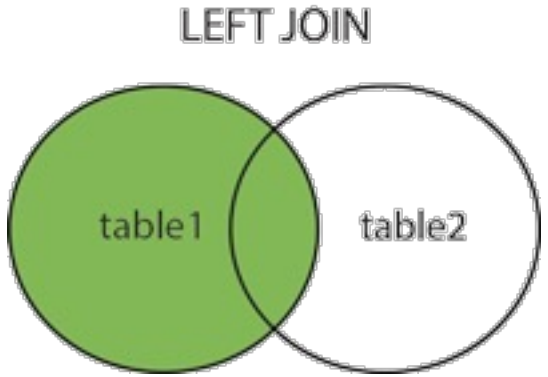
```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.



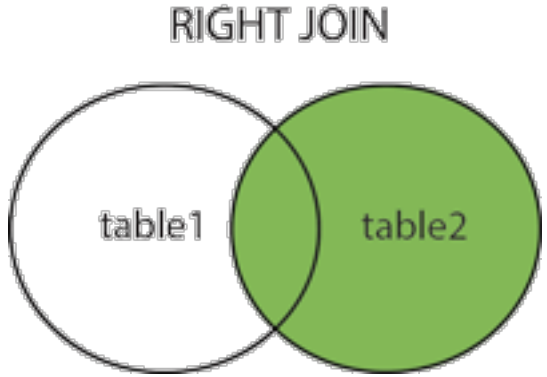

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

RIGHT JOIN Keyword

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.
- RIGHT JOIN Syntax
- ```
  SELECT column_name(s)
  FROM table1
  RIGHT JOIN table2
  ON table1.column_name = table2.column_name;
  ```



- **Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.
- **Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).
  ```

UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Every SELECT statement within UNION must have the same number of columns

The columns must also have similar data types

The columns in every SELECT statement must also be in the same order

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Note: The column names in the result-set are usually equal to the column names in the first SELECT statement.

GROUP BY Statement

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

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

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

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot 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);
```

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

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);
```

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

UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

FOREIGN KEY Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Person table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

FOREIGN KEY

- Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.
The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

Data Types for Strings

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum string length in characters - can be from 0 to 65535
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them

Numeric Data Types

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size*parameter can hold a value from 1 to 64. The default value for *size* is 1.
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d*parameter
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.

Date and Time Data Types

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

NOSQL DATABASE

NoSQL can be defined as a database which is employed for managing the massive collection of unstructured data and when your data is not piled up in a tabular format or relations like that of relational databases. The term NoSQL came from the word non SQL or nonrelational. There are a wide variety of existing Relational Databases that have been unsuccessful in solving several complex modern problems such as:

A dynamic change in the nature of data - i.e., nowadays data are in structured, semi-structured, nonstructured as well as polymorphic in type.

The variety of applications and the type of data feed into them for analysis has now become more diverse and distributed and is approaching cloud-oriented.

Also, modern applications and services are serving tens of thousands of users in diverse geo-locations, having diverse time zones. So data integrity needs to be there at all the time.

Structured Data Vs. Unstructured Data

Structured data are in a proper format, usually text files or which can be represented in a tabular format. Also, such types of data can be smoothly represented in chart-like form, and data mining tools can be used to process them efficiently.

Whereas unstructured data are haphazard data formats (such as document files, image files, video files, icons, etc.) where structured data can be pulled out or mine from unstructured data, but this process usually takes a lot of time. Modern-day data generated from different applications, services, or sources are a combination of structured and unstructured both. So, you will need something to store such data to make your application work properly. NoSQL based languages and scripts can help in this regard.

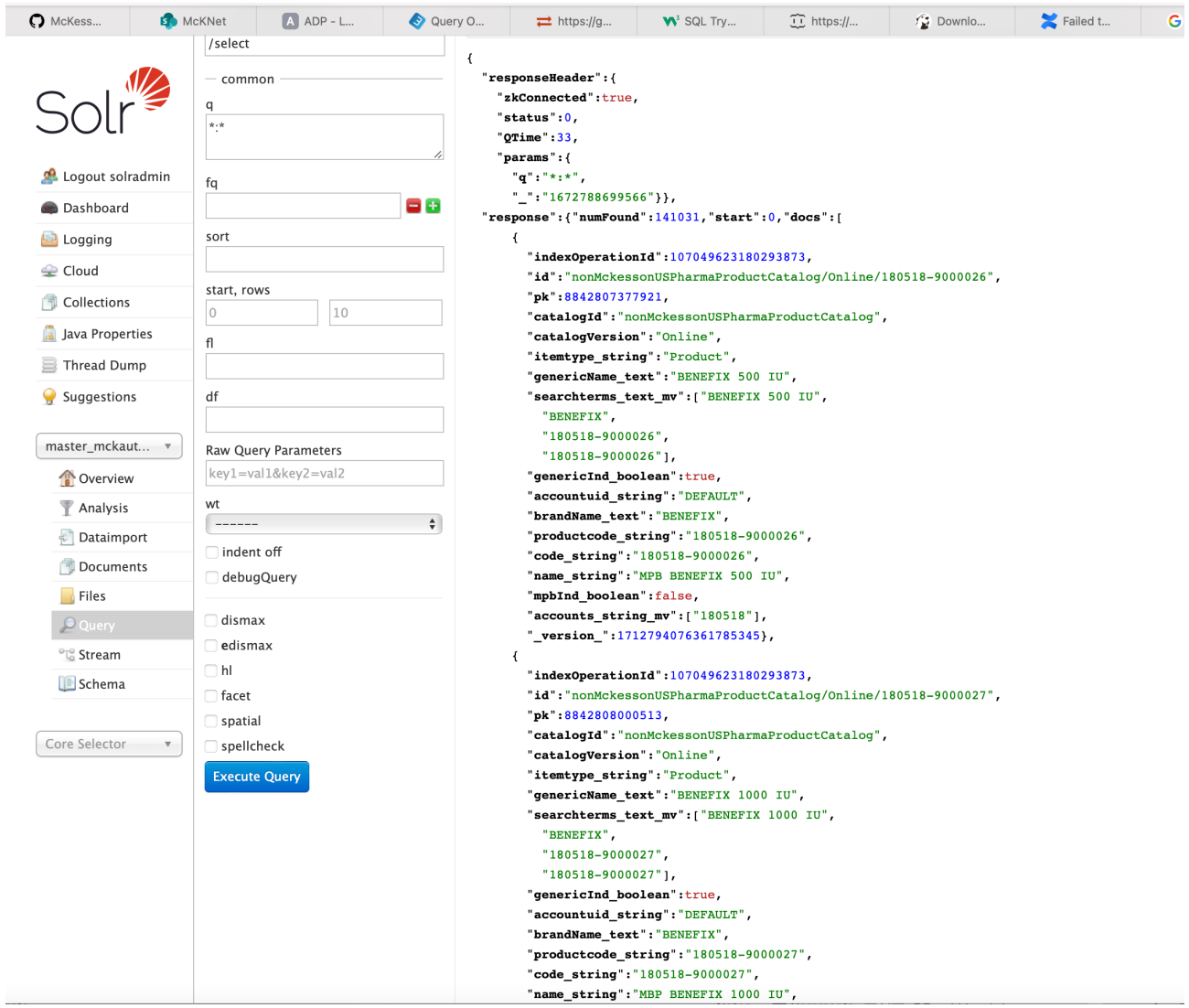
Types of Database in NoSQL

Here are some of the common database types that come under NoSQL:

1. Document type databases: Here, the key gets paired with a compound data structure, i.e., document. MongoDB is an example of such type.
2. Key-Value stores: Here, each unstructured data is stored with a key for recognizing it.
3. Graph stores: In this type of database, data is stored mostly for networked data. It helps to relate data based on some existing data.
4. Wide-column stores: This type of data stores large data sets Cassandra (Used by Facebook), HBase are examples of such type.

Benefits of NoSQL Type Database

- It allows developers to create large volumes of structured, semi-structured as well as unstructured data for making the application diverse and not restricting its use because of the type of data being used within the application.
- It also allows agile development; rapid iteration along with frequent code pushes, which makes it more popular.
- It can be used with object-oriented programming (OOP), which makes it easy to use with flexibility.
- Data can be stored more efficiently, making it less expensive, providing massive architecture.



SOLR as NOSQL database

Solr is a search engine at heart, but it is much more than that. **It is a NoSQL database with transactional support**. It is a document database that offers SQL support and executes it in a distributed manner.

/select?q=code_string%3A180518*

HOMEWORK

- **https://www.db4free.net/signup.php**
- Create database / user/ password
- Confirm in email – go to the conformation link
- Go to https://www.db4free.net/phpMyAdmin/
- Input user/password
- Create a table for houses (see an example in the presentation)
- Add 10 items from https://www.redfin.com as INSERT INTO house statement.
- Do search for the price less then 400000 $
- Do group by beds: count how many houses presented by each type of bedroom.
- Count for only 3 bedrooms.

HOMEWORK, part 2.

- Create additional table: realtor
- Columns: First name, Last name, id.
- Add realter column (RealtorID) in houses table.
- Add items into realtor table.
- Add information about realtor in few house rows.
- Do left join to show realtor First name and realtor last name for each house item if realtor id is presented.

HOMEWORK, part 3.

- Register **https://opensolr.com/users/login?registered=yes**
- Log in
- **https://opensolr.com/search/fresh**
- Try to find in NoSQL database Solr information – articles about Floride state
- Think about Google search – is it similar to thing?