



React

# 15. Introduction to SQL

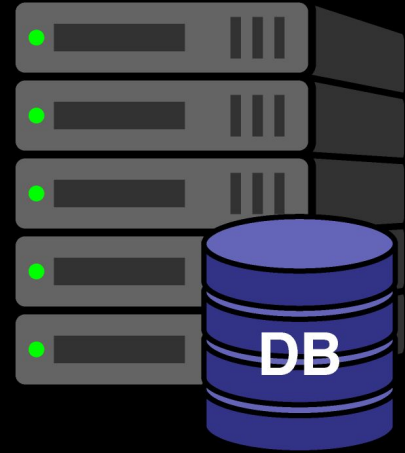
1. What is a DB (Database)
2. Introduction to SQL DB
3. Introduction to NoSQL DB
4. SQL vs NoSQL
5. Installing MySQL
6. Connecting App to DB
7. Creating homes Table
8. Querying homes in App
9. Adding DB in Models
10. Adding Home in Model
11. Implementing Model using Where

**he was forced to use  
SQL**

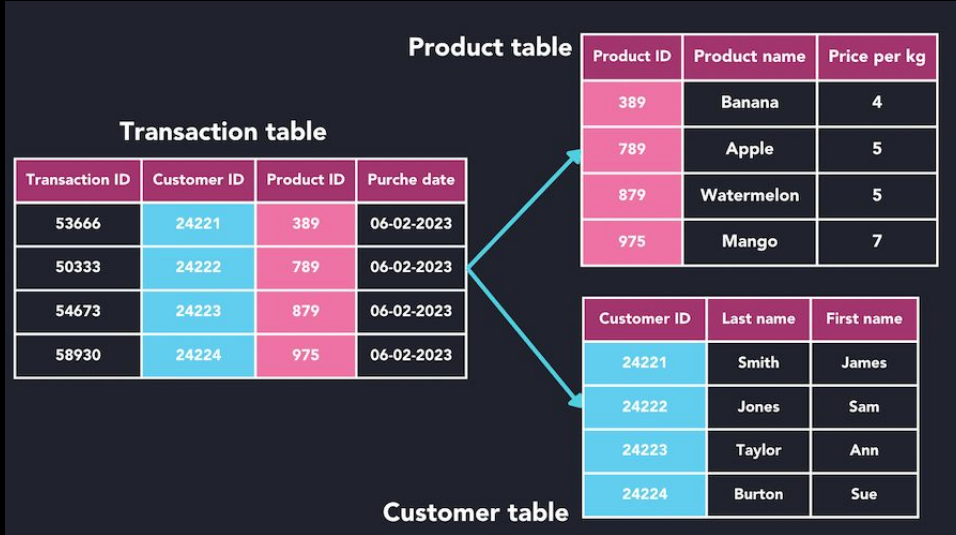


# 15.1 What is a DB (Database)

1. **Store Data:** Keep large amounts of data in a structured format.
2. **Enable Data Management:** Allow for adding, updating, and deleting data easily.
3. **Facilitate Quick Access:** Provide fast retrieval of data through queries.
4. **Ensure Data Integrity:** Maintain accuracy and consistency of data over time.
5. **Support Multiple Users:** Handle concurrent access by many users simultaneously.
6. **Secure Data:** Protect information through access controls and authentication.



# 15.2 Introduction to SQL DB



- **Vertical Scalability**: Typically scaled by increasing the resources of a single server (scaling up).
- **Relationships**: Tables can have multiple types of relationships.

- **Relational Model**: Organize data into tables with rows and columns.
- **Fixed Schema**: Require a predefined schema; the structure of the data must be known in advance.

# 15.2 Introduction to SQL DB

MySQL™

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

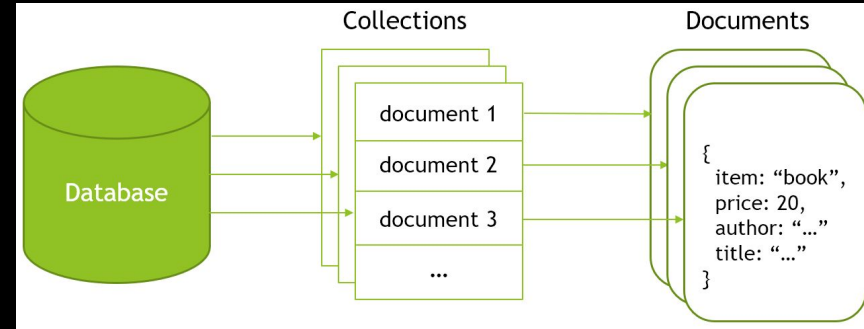
`SELECT age, country  
FROM Customers  
WHERE country = 'USA';`

age	country
31	USA
22	USA

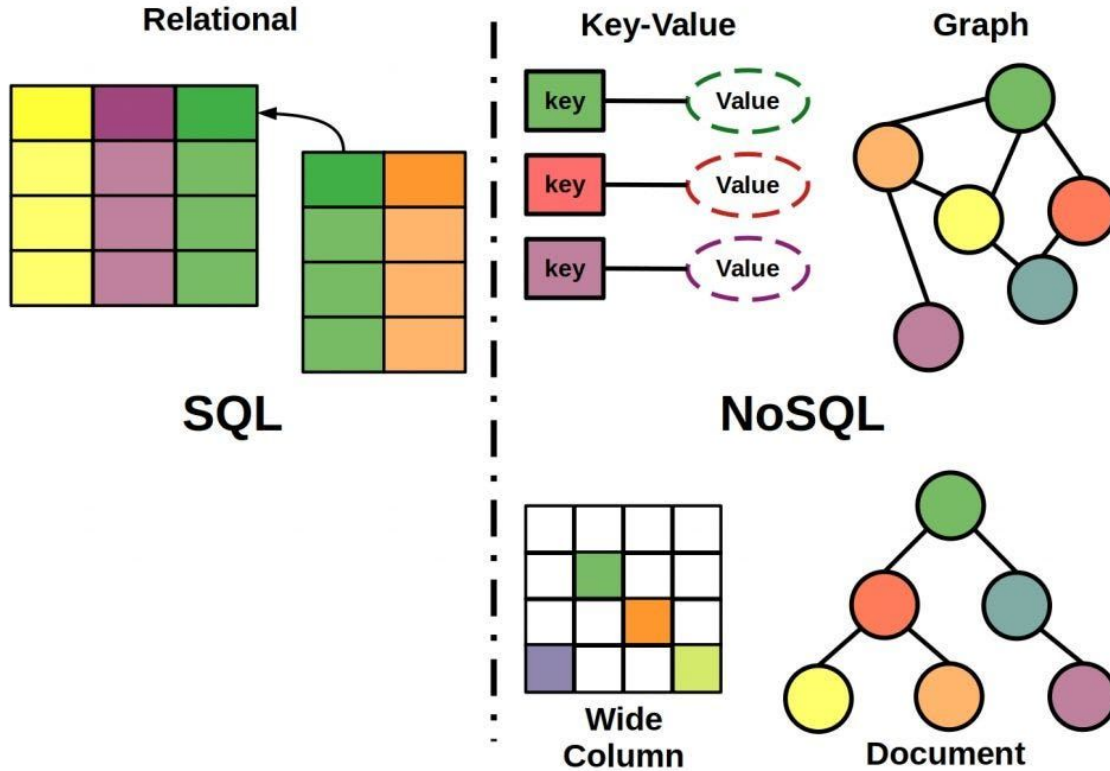
- **Relational Model Use of SQL:** Utilize SQL for querying and managing data, which is a standardized and widely-used language.
- **ACID Compliance:** Support transactions that are Atomic, Consistent, Isolated, and Durable.
- **Complex Queries:** Excel at handling complex queries and relationships between data.

# 15.3 Introduction to NoSQL DB

- **Flexible Schema:** Allow for dynamic schemas, accommodating unstructured or semi-structured data without predefined structures.
- **Duplicacy over Relations:** Duplicates data across records (denormalization) to enhance performance and scalability, rather than relying on complex relationships and joins as in relational databases.
- **Horizontal Scalability:** Designed to scale out by adding more servers, handling large volumes of data efficiently.
- **Performance:** Optimized for high throughput and low latency, suitable for real-time applications.



# 15.4 SQL vs NoSQL



# 15.4 SQL vs NoSQL

Feature	SQL Databases	NoSQL Databases
<b>Data Model</b>	Relational (tables with rows and columns)	Non-relational (document, key-value, graph, etc.)
<b>Schema</b>	Fixed schema (predefined structure)	Flexible schema (dynamic structure)
<b>Scalability</b>	Vertically scalable (scale up)	Horizontally scalable (scale out)
<b>Query Language</b>	SQL (Structured Query Language)	Various query languages and APIs
<b>ACID Compliance</b>	Strong ACID compliance	Varies; often prioritizes performance over ACID
<b>Use Cases</b>	Structured data and complex queries	Unstructured data and real-time applications

# 15.5 Installing MySQL

MySQL™

MySQL Community Downloads

MySQL Enterprise Edition for Developers  
Free for learning, developing, and prototyping.  
[Download Now »](#)

MySQL Community Server

General Availability (GA) Releases

Archives

MySQL Community Server 9.1.0 Innovation

Select Version:  

9.1.0 Innovation

Select Operating System:  

Microsoft Windows

<b>Windows (x86, 64-bit), MSI Installer</b> (mysql-9.1.0-winx64.msi)	9.1.0	118.1M	<a href="#">Download</a>
	MD5: 7a26420bb3446eab56f389dba05a9718   <a href="#">Signature</a>		
<b>Windows (x86, 64-bit), ZIP Archive</b> (mysql-9.1.0-winx64.zip)	9.1.0	288.4M	<a href="#">Download</a>
	MD5: 0a2333afc4ef07471bda89232697698e   <a href="#">Signature</a>		
<b>Windows (x86, 64-bit), ZIP Archive Debug Binaries &amp; Test Suite</b> (mysql-9.1.0-winx64-debug-test.zip)	9.1.0	822.6M	<a href="#">Download</a>
	MD5: cb0327a504fc8d983f98c0cbf451a31d   <a href="#">Signature</a>		

We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.



# 15.5 Installing MySQL

MySQL™

dev.mysql.com/downloads/workbench/ Guest (4)

## MySQL Community Downloads

MySQL Workbench

General Availability (GA) Releases Archives

### MySQL Workbench 8.0.38

Select Operating System:  
Microsoft Windows

Recommended Download:

#### MySQL Installer for Windows

All MySQL Products. For All Windows Platforms. In One Package.


Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI

[Go to Download Page >](#)

Other Downloads:

Windows (x86, 64-bit), MSI Installer	8.0.38	41.7M	<a href="#">Download</a>
(mysql-workbench-community-8.0.38-winx64.msi)		MD5: 30ea58c9f40816566ac4ccd2f136f1e2   <a href="#">Signature</a>	

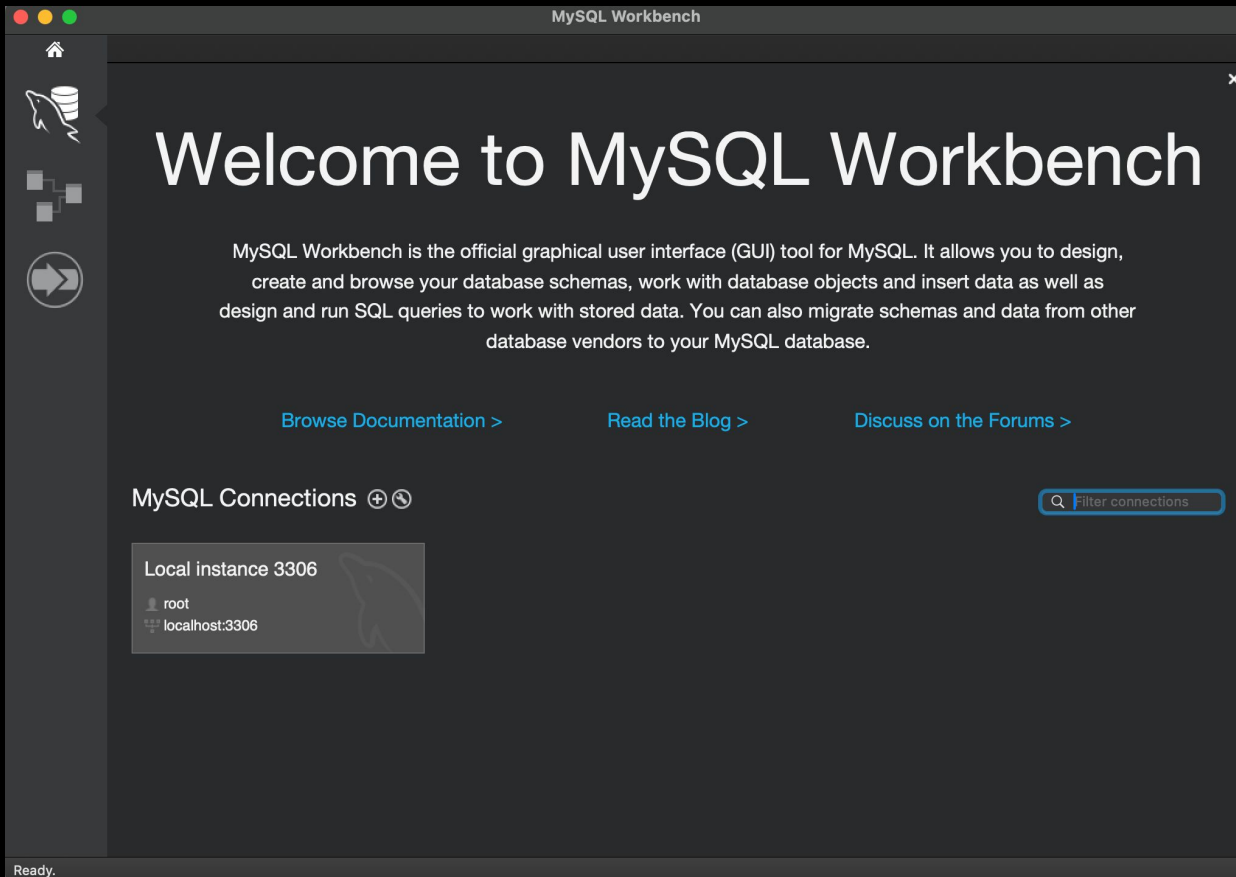
 We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.

ORACLE © 2024 Oracle

[Privacy](#) / [Do Not Sell My Info](#) | [Terms of Use](#) | [Trademark Policy](#) | [Cookie Preferences](#)

# 15.5 Installing MySQL

MySQL™



# 15.5 Installing MySQL

MySQL™

The screenshot displays the MySQL Workbench interface. At the top, a warning message reads "Local instance 3306 - Warning - not supported". The main window is divided into three panes. The left pane, titled "MANAGEMENT", contains a tree view with categories: "MANAGEMENT" (Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, Data Import/Restore), "INSTANCE" (Startup / Shutdown, Server Logs, Options File), and "PERFORMANCE" (Dashboard, Performance Reports, Performance Schema Setup). The middle pane, titled "Query 1", shows a SQL editor with a toolbar and a "Limit to 1000 rows" button. The right pane, titled "Context Help", displays a message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." Below the main panes, there is a "Session" tab and a "No object selected" message. At the bottom, a status bar indicates "SQL Editor Opened."

Local instance 3306 - Warning - not supported

Administration Schemas Query 1 Context Help Snippets

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

Object Info Session

No object selected

1

Limit to 1000 rows

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

100% 1:1

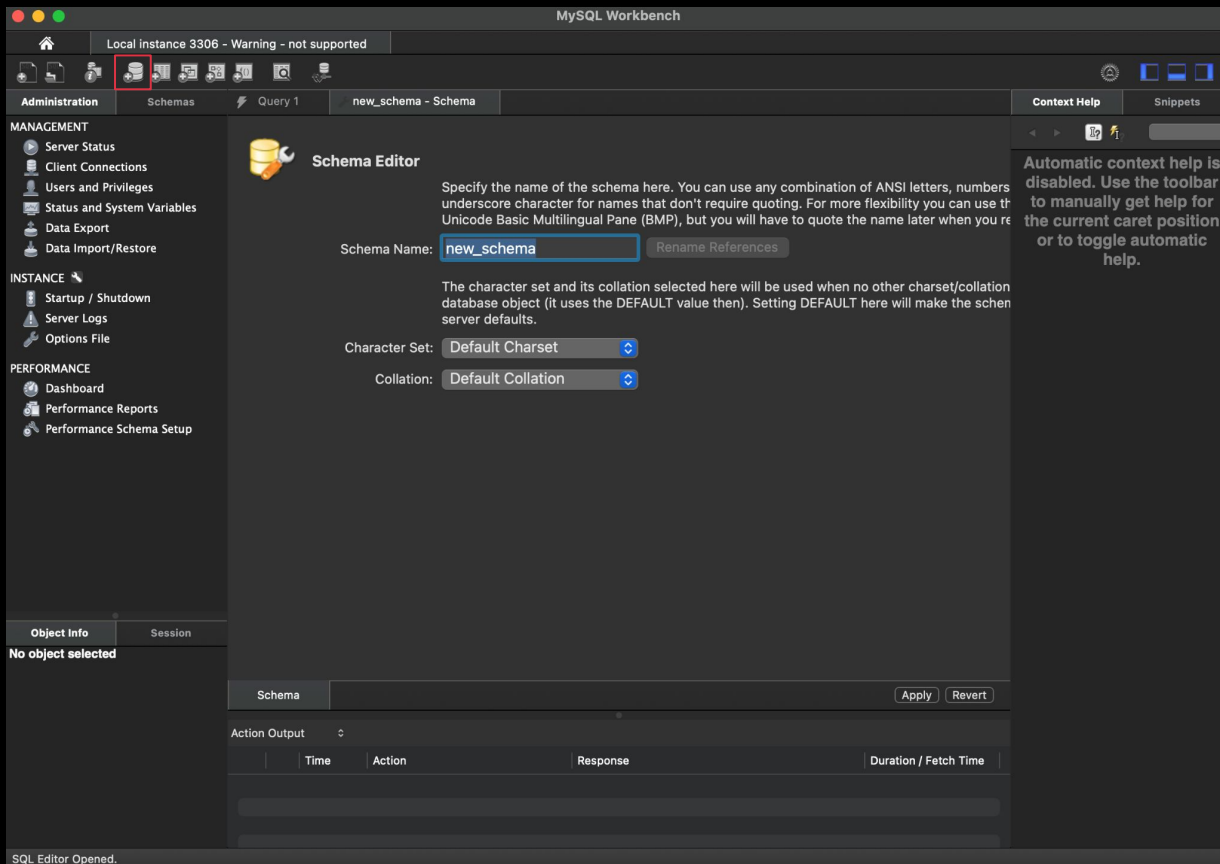
Action Output

Time	Action	Response	Duration / Fetch Time

SQL Editor Opened.

# 15.5 Installing MySQL

MySQL™



The screenshot shows the MySQL Workbench interface with the Schema Editor open. The left sidebar contains a 'MANAGEMENT' section with options like Server Status, Client Connections, Users and Privileges, Status and System Variables, Data Export, and Data Import/Restore. Below this is an 'INSTANCE' section with Startup / Shutdown, Server Logs, and Options File. At the bottom of the sidebar is a 'PERFORMANCE' section with Dashboard, Performance Reports, and Performance Schema Setup. The main area is titled 'Schema Editor' and contains a text input for 'Schema Name' with the value 'new\_schema' and a 'Rename References' button. Below this are dropdown menus for 'Character Set' (Default Charset) and 'Collation' (Default Collation). A status bar at the bottom indicates 'SQL Editor Opened.'

MySQL Workbench

Local Instance 3306 - Warning - not supported

Administration Schemas Query 1 new\_schema - Schema Context Help Snippets

**MANAGEMENT**

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

**INSTANCE**

- Startup / Shutdown
- Server Logs
- Options File

**PERFORMANCE**

- Dashboard
- Performance Reports
- Performance Schema Setup

**Schema Editor**

Specify the name of the schema here. You can use any combination of ANSI letters, numbers underscore character for names that don't require quoting. For more flexibility you can use the Unicode Basic Multilingual Plane (BMP), but you will have to quote the name later when you rename it.

Schema Name:  [Rename References](#)

The character set and its collation selected here will be used when no other charset/collation database object (it uses the DEFAULT value then). Setting DEFAULT here will make the schema use the server defaults.

Character Set:

Collation:

**Object Info** Session

No object selected

**Schema**

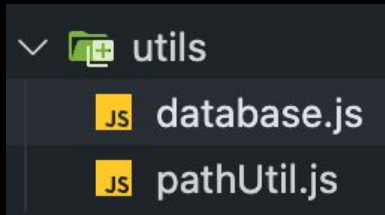
Action Output

Time	Action	Response	Duration / Fetch Time

SQL Editor Opened.

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

# 15.6 Connecting App to DB



```
prashantjain@Prashants-Mac-mini Chapter 13 - MVC % npm install --save mysql2
added 12 packages, and audited 222 packages in 586ms
49 packages are looking for funding
  run `npm fund` for details
```

A code editor window with a tab labeled 'database.js'. The editor shows the following JavaScript code:

```
utils > JS database.js > ...
1  const mysql = require("mysql2");
2
3  const pool = mysql.createPool({
4    host: "localhost",
5    user: "root",
6    password: "CompleteCoding@01",
7    database: "airbnb",
8  });
9
10 module.exports = pool.promise();
```

# 15.7 Creating homes Table

Query 1

homes - Table

Name: homes

Schema: airbnb

Column	Datatype	PK	NN	UQ	B...	UN	ZF	AI	G	Default / Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
price	DOUBLE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
description	LONGTEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
imageUrl	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
location	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
rating	DOUBLE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<click to edit>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column details "

Column Name:

Datatype:

Charset/Collation:

Expression

Comments:

Storage: ☒ VIRTUAL ☐ STORED

☒ Primary Key ☒ Not NULL ☒ Unique

☐ Binary ☐ Unsigned ☐ ZeroFill

☒ Auto Increment ☒ Generated

Columns

Indexes

Foreign Keys

Triggers

Partitioning

Options

Apply

Revert

## 15.7 Creating homes Table

```
⊖ CREATE TABLE `airbnb`.`homes` (  
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(255) NOT NULL,  
  `price` DOUBLE NOT NULL,  
  `description` LONGTEXT NOT NULL,  
  `imageUrl` VARCHAR(255) NOT NULL,  
  `location` VARCHAR(45) NOT NULL,  
  `rating` DOUBLE NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE);
```

# 15.7 Creating homes Table

The screenshot shows a database management interface. On the left, a sidebar lists the database structure: 'airbnb' (Tables) and 'sys'. Under 'airbnb', there is a 'homes' table with sub-items for 'Columns', 'Indexes', 'Foreign Keys', and 'Triggers'. The main area displays the 'homes' table with the following columns: id, name, price, description, imageUrl, location, and rating. The table is currently empty, with all cells showing 'NULL'. Above the table, a query editor shows the SQL statement: `SELECT * FROM airbnb.homes;`. The interface also includes a 'Result Grid' button and a 'Form Editor' button.

## Apply SQL Script to Database

### Review the SQL Script to be Applied on the Database

Please review the following SQL script that will be applied to the database.  
Note that once applied, these statements may not be revertible without losing some of the data.  
You can also manually change the SQL statements before execution.

```
1 INSERT INTO `airbnb`.`homes` (`name`, `price`, `description`, `imageUrl`, `location`) VALUES ('Utsav', '999', 'the best holiday home', '/images/house1.png', 'delhi');
2
```



## 15.8 Querying homes in App

```
const db = require("../utils/database");
```

```
db.execute("SELECT * FROM homes").then(([rows, fields]) => {  
  console.log(rows);  
  console.log(fields);  
}).catch((error) => {  
  console.log("Error Fetching Homes", error);  
});
```

Server running on address http://localhost:3000

```
[  
  {  
    id: 1,  
    name: 'Utsav',  
    price: 999,  
    description: 'the best holiday home',  
    imageUrl: '/images/house1.png',  
    location: 'delhi',  
    rating: 4.5  
  }  
]  
[  
  `id` INT UNSIGNED NOT NULL PRIMARY KEY UNIQUE_KEY AUTO_INCREMENT,  
  `name` VARCHAR(255) NOT NULL,  
  `price` DOUBLE NOT NULL,  
  `description` LONGTEXT NOT NULL,  
  `imageUrl` VARCHAR(255) NOT NULL,  
  `location` VARCHAR(45) NOT NULL,  
  `rating` DOUBLE NOT NULL  
]
```

## 15.9 Adding DB in Models

1. Remove the test code from app.js
2. Change the Home.js model file to remove all code related to file operations.
3. Import the DB from the utils.
4. Change photoUrl to imageUrl and houseName to name in the entire project.
5. Implement fetchAll:
  - a. Using the query we used while testing.
  - b. fetchAll will not take a callback but return a promise.
6. Go to StoreController and use the promise to get the data here.
7. Fix all the usages of fetchAll.

# 15.9 Adding DB in Models

2,3.

```
const db = require("../utils/database");  
const Favourites = require("../favourites");
```

```
module.exports = class Home {  
  constructor(houseName, price, location, rating, photoUrl) {  
    this.houseName = houseName;  
    this.price = price;  
    this.location = location;  
    this.rating = rating;  
    this.photoUrl = photoUrl;  
  }  
  
  save() {    
  }  
  
  static fetchAll() {  
  }  
  
  static findById(id) {  
  }  
  
  static deleteById(id) {  
  }  
};
```

## 15.9 Adding DB in Models

5. 

```
static fetchAll() {  
  return db.execute("SELECT * FROM homes");  
}
```

6. 

```
exports.getHomes = (req, res, next) => {  
  Home.fetchAll()  
    .then(([rows, fields]) => {  
      res.render("store/home-list", {  
        registeredHomes: rows,  
        pageTitle: "Homes List",  
        currentPage: "Home",  
      });  
    })  
    .catch((error) => {  
      console.log("Error Fetching Homes", error);  
    });  
};
```

# 15.9 Adding DB in Models

7.

```
exports.getIndex = (req, res, next) => {
  Home.fetchAll()
    .then(([rows, fields]) => {
      res.render("store/index", {
        registeredHomes: rows,
        pageTitle: "airbnb Home",
        currentPage: "index",
      })
    })
    .catch((error) => {
      console.log("Error Fetching Homes", error);
    });
};
```

```
exports.getHostHomes = (req, res, next) => {
  Home.fetchAll().then(([rows, fields]) => {
    res.render("host/host-home-list", {
      registeredHomes: rows,
      pageTitle: "Host Homes",
      currentPage: "hostHomes",
    });
  }).catch((error) => {
    console.log("Error Fetching Homes", error);
  });
};
```

```
exports.getFavouriteList = (req, res, next) => {
  Favourites.getFavourites((favourites) => {
    Home.fetchAll().then(([registeredHomes, fields]) => {
      const favouritesWithDetails = favourites.map((homeId) => {
        registeredHomes.find((home) => home.id === homeId)
      });
      res.render("store/favourite-list", {
        favourites: favouritesWithDetails,
        pageTitle: "My Favourites",
        currentPage: "favourites",
      });
    }).catch((error) => {
      console.log("Error Fetching Homes", error);
    });
  });
};
```

## 15.10 Adding Home in Model

1. Add the **description** field in home. Change constructor and usage.
2. Make changes in UI to input and show it everywhere.
3. Implement the **save method** using the **insert query**.
4. Change the **usages of save method** to use the **promise**.

# 15.10 Adding Home in Model

1.

```
module.exports = class Home {  
  constructor(name, description, price, location, rating, imageUrl) {  
    this.name = name;  
    this.description = description;  
    this.price = price;  
    this.location = location;  
    this.rating = rating;  
    this.imageUrl = imageUrl;  
  }  
}
```

```
exports.postAddHome = (req, res, next) => {  
  const { name, description, price, location, rating, imageUrl } = req.body;  
  const home = new Home(name, description, price, location, rating, imageUrl);  
}
```

```
exports.postEditHome = (req, res, next) => {  
  const { id, name, description, price, location, rating, imageUrl } = req.body;  
  const home = new Home(name, description, price, location, rating, imageUrl);  
}
```

# 15.10 Adding Home in Model

2.

```
<input
  type="text"
  name="description"
  value="<%= home ? home.description : '' %>"
  placeholder="Enter your House Description"
  class="w-full px-4 py-2 mb-4 border rounded-md focus:outline-none focus:ring-2
  focus:ring-red-500"/>

<div class="border-b pb-4">
  <h3 class="text-2xl font-semibold mb-2">Description</h3>
  <p class="text-gray-600"><%= home.description %></p>
</div>

<div class="border-b pb-4">
  <h3 class="text-2xl font-semibold mb-2">Location</h3>
  <p class="text-gray-600"><%= home.location %></p>
</div>
```



# 15.10 Adding Home in Model

3.

```
save() {  
  return db.execute(  
    "INSERT INTO homes (name, price, location, rating, imageUrl) VALUES (?, ?, ?, ?, ?)",  
    [this.name, this.price, this.location, this.rating, this.imageUrl]  
  );  
}
```

4.

```
exports.postAddHome = (req, res, next) => {  
  const { name, description, price, location, rating, imageUrl } = req.body;  
  const home = new Home(name, description, price, location, rating, imageUrl);  
  home.save().then(() => {  
    res.render("host/home-added", {  
      pageTitle: "Home Added Successfully",  
      currentPage: "homeAdded",  
    });  
  }).catch((error) => {  
    console.log("Error Adding Home", error);  
  });  
};  
  
exports.postEditHome = (req, res, next) => {  
  const { id, name, description, price, location, rating, imageUrl } = req.body;  
  const home = new Home(name, description, price, location, rating, imageUrl);  
  home.id = id;  
  home.save().then(() => {  
    res.redirect("/host/host-home-list");  
  }).catch((error) => {  
    console.log("Error Editing Home", error);  
  });  
};
```

## 15.11 Implementing Model using Where

```
static findById(id) {  
  return db.execute("SELECT * FROM homes WHERE id = ?", [id]);  
}  
  
static deleteById(id) {  
  return db.execute("DELETE FROM homes WHERE id = ?", [id]);  
}
```

```
exports.postDeleteHome = (req, res, next) => {  
  const homeId = req.params.homeId;  
  Home.deleteById(homeId).then(() => {  
    res.redirect("/host/host-home-list");  
  }).catch((error) => {  
    console.log("Error Deleting Home", error);  
  });  
};
```

```
exports.getHome = (req, res, next) => {  
  const homeId = req.params.homeId;  
  Home.findById(homeId).then(([rows]) => {  
    const home = rows[0];  
    if (!home) {  
      return res.redirect("/homes");  
    }  
    res.render("store/home-detail", {  
      home: home,  
      pageTitle: home.name,  
      currentPage: "homes",  
    });  
  }).catch((error) => {  
    console.log("Error Fetching Home", error);  
  });  
};
```