coding temple

Software Engineering

Module 5:

Relational Databases

Structured Query Language

(SQL) Essentials

Learning Objectives

Students should be able to:

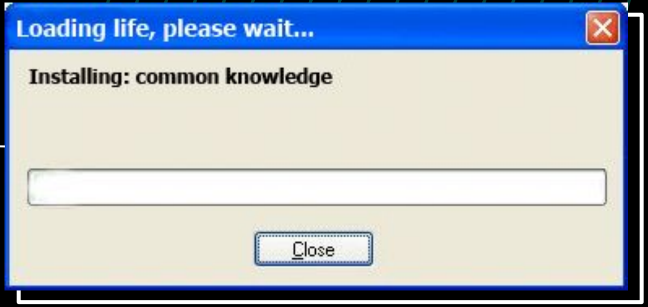
- Identify and select the appropriate MySQL version for their operating system, understanding its relevance to their data management projects.
- Successfully install MySQL and MySQL Workbench demonstrating an understanding of each step in the installation process.
- Execute basic SQL commands such as CREATE, ALTER, and DROP, applying these Data Definition Language (DDL) commands to manage and structure database elements effectively.
- Utilize INSERT, UPDATE, and DELETE commands in SQL, demonstrating proficiency in manipulating data within a database using Data Manipulation Language (DML).
- Construct SQL queries using the SELECT statement, applying this command to retrieve specific data sets from a database.
- Establish and manipulate data relationships showing the ability to connect and interact with data across multiple tables.

Intro

Embark on an exciting journey into SQL and MySQL! Designed for both beginners and those honing their database skills, we'll guide you from setting up your digital workshop to mastering SQL commands like SELECT, UPDATE, INSERT, and DELETE. Explore Data Definition, Manipulation, and Query Language, shaping and linking data like a pro. Get ready to turn data into a canvas for your creativity and analytical skills, crafting stories that inform, engage, and inspire. Let's start this future-focused adventure together!

Installing MySQL - Building Your Data Workshop for Windows and MacOS

Welcome to our step-by-step tutorial on installing MySQL, where we will guide you through setting up your data workshop. Just like constructing a physical workshop in your garage, this process involves selecting the right space, gathering your tools, setting up your workbench, and organizing everything for optimal use.



Loading life, please wait...

Installing: common knowledge

Step 1: Choosing Your Workshop Space

Objective: Select the appropriate MySQL version for your system.

- Choose the right MySQL version. Consider your operating system (Windows, Linux, MacOS) and the specific requirements of your projects.
- **Action Steps:**
  - Visit the MySQL Downloads Page.
  - Choose the version that suits your operating system and data needs.

Step 2: Gathering Your Tools

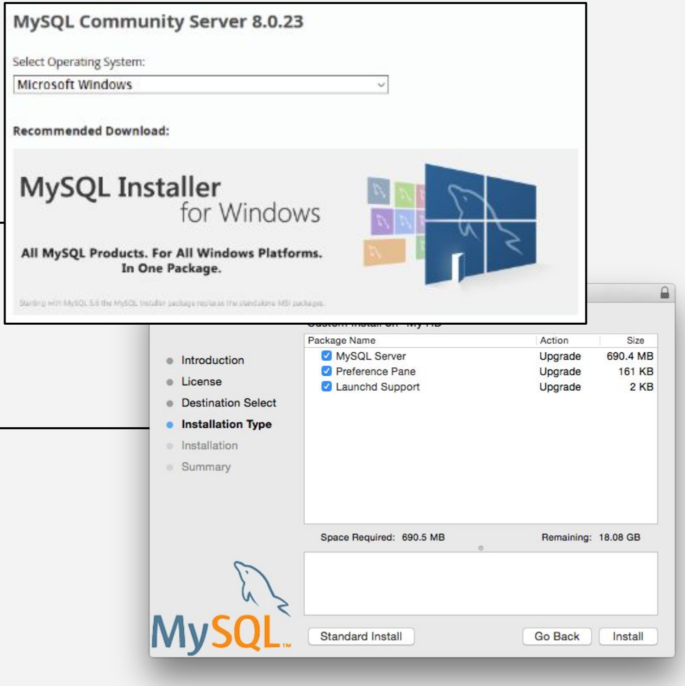
(Downloading MySQL)

Objective: Download the MySQL installation package.

- Think of this step as going to the hardware store to pick up the necessary tools for your workshop.
- **Action Steps:**
    - On the MySQL Downloads page, select the "MySQL Community Server" section.
    - Click the "Download" button for the version you chose.
    - Choose the appropriate installer for your operating system.

Installing MySQL: Further steps to follow

You've selected your MySQL version and downloaded the installer. Now, let's dive into the operating system-specific steps for Windows and MacOS. This stage is like customizing your workshop layout based on the space and resources you have.



MySQL Community Server 8.0.23

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.

Package Name	Action	Size
☑ MySQL Server	Upgrade	690.4 MB
☑ Preference Pane	Upgrade	161 KB
☑ Launchd Support	Upgrade	2 KB

Space Required: 690.5 MB   Remaining: 18.08 GB

Step 3: Setting Up the Workbench

(Installing MySQL)

For Windows Users:

Objective: Install MySQL on your Windows machine.

- Assembling your workbench in a garage with specific dimensions and facilities.
- **Action Steps:**
  - Run the downloaded .msi installer.
  - Choose "Full" installation for a comprehensive setup.
  - Follow the installer prompts, clicking "Next" to proceed through each step.
  - Complete the installation wizard and make sure to note down any details like the installation path.

Step 3: Setting Up the Workbench

(Installing MySQL)

For MacOS Users:

Objective: Install MySQL on your MacOS machine.

- Setting up a workbench in a uniquely structured shed.
- **Action Steps:**
  - Open the downloaded .dmg file.
  - Double-click the MySQL .pkg installer from the disk.
  - This will launch the MySQL Installer, which will guide you through the installation process.
  - Follow the on-screen instructions, accept the license agreement, and choose the installation location.
  - During installation, you'll be asked to set a root password. Make sure to note this down as it's crucial for future access to MySQL.
  - Complete the installation process.

Step 4: First Project

(Testing MySQL)

For Windows Users:

Objective: Test MySQL on Windows.

- **Action Steps:**
  - Trying out a simple project in your Windows workshop.
  - **Action Steps:**
  - Search for "MySQL Command Line Client" in the start menu and open it.
  - Enter the root password you set during configuration.
  - Test with the command: SHOW DATABASES; to display a list of databases.

Step 4: First Project

(Testing MySQL)

For MacOS Users:

Objective: Test MySQL on MacOS.

- Conducting an initial project in your MacOS workshop.
- **Action Steps:**
  - Open the Terminal application.
  - Connect to MySQL with `mysql -u root -p` and enter your password when prompted.
  - Execute `SHOW DATABASES`; to view a list of databases.

Installing MySQL Workbench - Crafting Your Data Design Studio for Windows and MacOS

Welcome to the tutorial on installing MySQL Workbench, your tool for visually designing, modeling, and managing databases. Think of MySQL Workbench as the advanced blueprint station in your data workshop, where you can plan and visualize projects before bringing them to life. We'll cover installation steps for both Windows and MacOS and conclude with a primer on creating a sample Entity-Relationship Diagram (ERD).

Step 1: Preparing Your Studio Space

(Downloading MySQL Workbench)

Objective: Download MySQL Workbench.

- Selecting the right blueprint station for your workshop.
- **Action Steps:**
  - Visit the MySQL Workbench download page.
  - Choose the version that corresponds to your operating system.
  - Click "Download" and follow the instructions to complete the download process.

Step 2: Setting Up Your Blueprint Station

(Installing MySQL Workbench)

For Windows Users:

Objective: Install MySQL Workbench on your Windows machine.

- Assembling a blueprint station in a Windows environment.
- **Action Steps:**
  - Locate and run the downloaded .msi installer.
  - Follow the on-screen instructions, proceeding through the installation wizard.
  - Complete the installation by clicking 'Finish'.

Step 2: Setting Up Your Blueprint Station

(Installing MySQL Workbench)

For MacOs Users:

Objective: Install MySQL Workbench on your MacOS machine.

- Assembling a blueprint station in a Windows environment.
- **Analogous Task:** Setting up a blueprint station in a MacOS environment.
- **Action Steps:**
  - Open the downloaded .dmg file.
  - Drag the MySQL Workbench icon to your Applications folder.
  - Follow any additional prompts to complete the installation.

Step 3: Familiarizing with Your Blueprint Station

(Overview and Creating MySQL Connections)

For Both Windows and MacOS Users:

Objective: Understand the MySQL Workbench interface and establish a connection to a MySQL server.

- Learning how to use your new blueprint station and connecting it to your project materials.
- Action Steps:

1. **Opening MySQL Workbench:** Launch the application. You're greeted with the home screen, which is your control panel for database connections and models.
2. **Exploring the Interface:** Notice sections like 'MySQL Connections', 'Models', 'Migration', etc. Each serves a specific purpose - connections for accessing databases, models for design work, and so on.
3. **Creating a New Connection:**
   - Click on the '+' symbol next to 'MySQL Connections'.
   - A setup window appears where you can name your connection and specify details like the hostname (usually 'localhost' for local servers), port (default is 3306), and username.
   - If you've set a password for your MySQL server, enter it here or leave it blank to enter each time you connect.
   - Click 'Test Connection' to ensure everything is set up correctly. If successful, you'll see a confirmation message.

1. **Saving the Connection:** After testing, save the connection. It will now appear on your home screen, ready for use whenever you start a new session.

coding temple

Embarking on Our Data Journey: DDL Post Installation

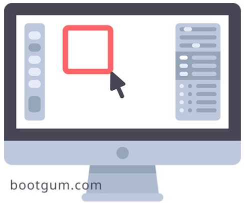
Now that we've successfully installed our essential tools - MySQL and MySQL Workbench - in our digital workshop, it's time to delve into the heart of structuring and managing our data. Think of MySQL as your robust workbench and MySQL Workbench as your detailed blueprint table. With these tools in place, we're perfectly equipped to start exploring Data Definition Language (DDL).

What is DDL?

DDL consists of the commands used to create, modify, and remove database objects like tables, indexes, and users. In our workshop analogy, this is akin to drawing up plans for workbenches, shelves, and storage units - the essential components of your workspace.

Creating

Modifying



Removing

Starting with a Clean Slate:

Creating Our Database

First things first, we need a dedicated space in our workshop to house our b e-commerce data. This is where the **CREATE DATABASE** statement comes in.
This line is akin to designating a specific area of our workshop for all our e-commerce related tools and materials.

```
CREATE DATABASE e_commerce_db;
```

Laying Down the Foundations:

Creating Tables

With our dedicated space ready, it’s time to build the tables - Customers and Orders.

1. Creating the Customers Table

Think of this as setting up a storage unit to organize customer information.

Here, id is like a unique identifier for each customer in our workshop, and AUTO_INCREMENT ensures that each customer gets a unique number automatically.

```sql
USE e_commerce_db;
CREATE TABLE Customers (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NULL
);
```

2. Creating the Orders Table

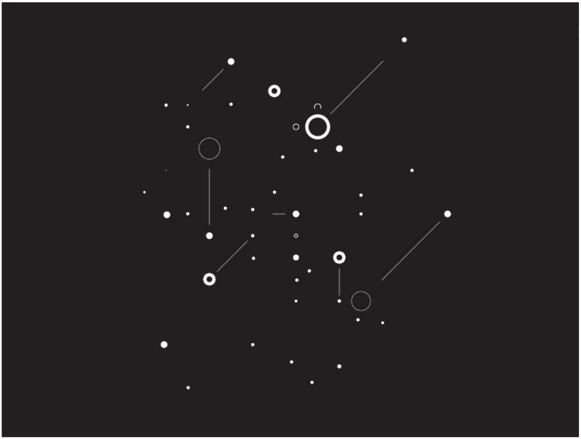
Now, let's build a second storage unit for order details.

In the Orders table, id serves as a unique identifier for each order. The customer_id column here links to the Customers table, establishing a connection between a customer and their orders. This is like labeling each order with the name of the customer who placed it.

```
CREATE TABLE Orders (
  id INT AUTO_INCREMENT PRIMARY KEY,
  date DATE NOT NULL,
  customer_id INT,
  FOREIGN KEY (customer_id) REFERENCES
Customers(id)
);
```

3. **Connecting the Dots:** Establishing Relationships

The FOREIGN KEY in the Orders table creates a one-to-many relationship between Customers and Orders. One customer (from the Customers table) can place many orders (in the Orders table), but each order is linked to only one customer. This is like saying that a customer can have multiple project blueprints (orders), but each blueprint is specifically assigned to one customer.



Tweaking Our Storage Units:

Using ALTER Command

Sometimes, our initial design needs a little tweaking. Maybe we need an extra column in a table or want to change an existing column's properties. This is where the ALTER TABLE command shines.

1. Adding a New Column

Let's say we want to add a new column phone to the Customers table. It's like adding a new compartment to our customer information storage unit. Now, our Customers table has an additional field to store phone numbers.

```sql
ALTER TABLE Customers
ADD phone VARCHAR(15);
```

2. Modifying a Column

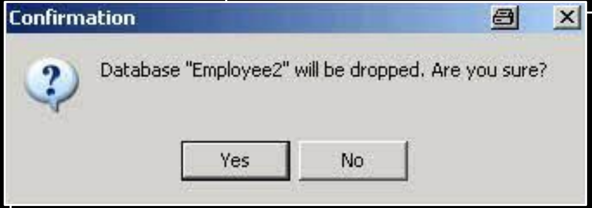
Suppose we realize that the email field needs more space. We can modify it to accommodate longer email addresses.
This change is like adjusting the size of a drawer to fit more items.

```
ALTER TABLE Customers
MODIFY email VARCHAR(320);
```

Clearing Out Unneeded Elements:

Using DROP Command

Over time, we might find that certain elements in our workshop are no longer needed. Similarly, the DROP command in MySQL allows us to remove entire tables or databases.



Database "Employee2" will be dropped. Are you sure?

1. Dropping a Table & 2. Dropping a Database

1. Dropping a Table
Imagine we have a table that's no longer in use. We can remove it to free up space. This command is like removing an unused workbench to make room for new projects.

2. Dropping a Database
In a more significant overhaul, we might need to remove an entire database. It's like clearing out an entire section of our workshop that we plan to repurpose.

```
DROP TABLE UnusedTable;

DROP DATABASE
ObsoleteDatabase;
```

Data Manipulation Language (DML): The Essence of Crafting, Adjusting, and Refreshing Your Data Projects!

DML is our set of essential tools for adding new details, making adjustments, and keeping our workbench clean and organized. We'll dive into the core DML commands - INSERT, UPDATE, and DELETE - and explore how they transform our data workshop into a hub of creativity and efficiency.

The Importance of Caution

Just as in a physical workshop where you need to be cautious not to throw away essential tools, in our data workshop, we must use the DELETE command judiciously. Deleting data is irreversible, like discarding a tool that you later realized was crucial for your project. Always double-check your WHERE clause to ensure you're removing exactly what you intend to.

Data Query Language (DQL):

Unveiling the Treasures of Your Data Workshop with SELECT Statements!

Think of DQL, and specifically the SELECT statement, as the flashlight that helps us find exactly what we need on our cluttered workbench. It's all about shining a light on the specific tools or materials (data) we need at any given moment.

Honing In On The Details: SELECT with WHERE

The Precision Tool in Data Retrieval (SELECT with WHERE)

In a workshop filled with diverse tools and materials, finding exactly what you need requires a bit more than a broad light; it requires a focused beam. The WHERE clause in our SELECT statements acts like a laser pointer, pinpointing the exact data we're interested in.

Targeting Specific Criteria

Need to find customers with specific attributes? Since our Customers table doesn't have a city column, we'll focus on other details like names or emails. This is like focusing your flashlight to find a tool labeled 'Carol White' in your workshop.

```sql
-- Finding customers with a specific name
SELECT * FROM Customers
WHERE name = 'Carol White';
```

Crafting the Perfect Search:

SELECT with Logical Operators

The Art of Combining Criteria (AND, OR, NOT in SQL)

In our data workshop, sometimes we need to find items that meet several specific conditions, or perhaps exclude certain items from our search. This is where our logical operators come into play, acting as the precision tools in our data retrieval toolkit.

Crafting the Perfect Search:

SELECT with Logical Operators (Cont.)

Using AND for Precision

Suppose we want to find customers named 'John Doe' who also have a specific phone number.
This is where we use the AND operator.
This command is like using a filter to find a tool that is both a screwdriver and has a red handle in your workshop.

```sql
-- Selecting customers with a specific name and phone number
SELECT * FROM Customers
WHERE name = 'John Doe' AND phone = '123-456-7890';
```

Crafting the Perfect Search:

SELECT with Logical Operators (Cont.)

Expanding the Search

There are times when we want to find items that meet any one of several conditions. Here, we use the OR operator.

This is similar to looking for either a hammer or a tape measure in your workshop – finding either one will do.

```
-- Selecting customers named Carol White or those with a specific phone number
SELECT * FROM Customers
WHERE name = 'Carol White' OR phone = '123-456-7890';
```

Crafting the Perfect Search:

SELECT with Logical Operators (Cont.)

Excluding Specific Data

Sometimes, we need to exclude certain items from our search. The NOT operator is perfect for this purpose.

This command is like deciding to use every tool except the saw in your project.

```
-- Finding all customers except those named John Doe
SELECT * FROM Customers
WHERE NOT name = 'John Doe';
```

Sorting with Precision:

The ORDER BY Command

Lining Up Your Tools (Introduction to ORDER BY)

In a workshop, sometimes you need your tools or materials sorted in a specific order for efficiency. In SQL, the ORDER BY clause allows us to sort our data in either ascending (ASC) or descending (DESC) order. It's like arranging screwdrivers from smallest to largest or paints from lightest to darkest.

TRUST ME!
THERE'S NOTHING WEIRD ABOUT HAVING EVERYTHING IN IT'S RIGHTFUL PLACE.

Sorting with Precision:

The ORDER BY Command

Sorting Customers by Name

Let's say we want to view our list of customers, but we want it alphabetically sorted by name.
This command arranges our customer data in alphabetical order, much like lining up chisels in order of size.

```
-- Sorting customers by name in
ascending order
SELECT * FROM Customers
ORDER BY name ASC;
```

Sorting with Precision:

The ORDER BY Command

Sorting Customers by ID

Let's modify our command to sort the customers based on their IDs, from the newest (highest ID) to the oldest (lowest ID).
This command arranges the customer records starting with the one that has the highest ID number, moving down to the lowest. It's akin to lining up customer files on a shelf, with the newest file at the front and the oldest towards the back.

```
-- Sorting customers by ID in
descending order
SELECT * FROM Customers
ORDER BY id DESC;
```

Eliminating Redundancies:

The DISTINCT Command

Sifting Out the Duplicates (Introduction to DISTINCT)

In any workshop, duplicates can be confusing and space-consuming. The DISTINCT keyword in SQL helps us eliminate duplicate entries in our result set. It's like filtering out identical screws or nails, so you only have one of each type.

Eliminating Redundancies:

The DISTINCT Command

Unique Customer Emails

If we want to see all the unique email addresses of our customers, DISTINCT is the tool for the job.
This command filters out any duplicate email addresses, giving us a clear view of each unique one.

```sql
-- Selecting distinct email addresses from the customers
SELECT DISTINCT email FROM Customers;
```

Exploring Patterns in Your Data Workshop:

The Art of LIKE, IN, and BETWEEN in SQL!

Finding What Fits (Introduction to LIKE)

Sometimes in our workshop, we need to find tools or materials that have a specific mark or pattern. In SQL, the LIKE operator allows us to search for data that matches a certain pattern. It's like using a magnifying glass to look for screws with a specific head pattern.

Exploring Patterns in Your Data Workshop:

The Art of LIKE, IN, and BETWEEN in SQL! (Cont.)

Searching for Customer Names with a Pattern

Sometimes in our workshop, we need to find tools or materials that have a specific mark or pattern. In SQL, the LIKE operator allows us to search for data that matches a certain pattern. It's like using a magnifying glass to look for screws with a specific head pattern.

```sql
-- Finding customers whose names start with J
SELECT * FROM Customers
WHERE name LIKE 'J%';
```

Exploring Patterns in Your Data Workshop:

The Art of LIKE, IN, and BETWEEN in SQL! (Cont.)

Filtering Customer Emails Ending with @example.com

Or for example, suppose you want to find all customers whose email addresses end with the domain @example.com.

In these commands, the % symbol acts like a wildcard, representing any number of characters. The queries effectively sift through all the names and emails in the Customers table and pick out only those that start with J or conclude with @example.com. It's akin to having a filter that only lets through screws matching a specific thread pattern.

```
-- Finding customers with email
addresses ending in @example.com
SELECT * FROM Customers
WHERE email LIKE '%@example.com';
```

The Toolbox of Choices
(Introduction to IN)

The Toolbox of Choices
(Introduction to IN)

Imagine you have a list of specific tools you need to pick from your workshop. The IN operator in SQL is similar; it allows us to specify a list of values and find data that matches any of these values.

The Toolbox of Choices (Introduction to IN) (Cont.)

Selecting Customers from a List of IDs

Let's say we want to find customers with IDs 1, 2, and 3.

This command is like selecting tools that have specific serial numbers from a toolbox.

```sql
-- Selecting customers with specific IDs
SELECT * FROM Customers
WHERE id IN (1, 2, 3);
```

The Toolbox of Choices (Introduction to IN) (Cont.)

Selecting Specific Customers by Name

Let's create a query to find customers named 'Alice Smith', 'Bob Johnson', and 'Carol White'. This SQL command is similar to selecting specific materials from a shelf – only the ones that match the names on your list are chosen.

```
-- Selecting customers with specific names
SELECT * FROM Customers
WHERE name IN ('Alice Smith', 'Bob Johnson', 'Carol White');
```

Setting the Boundaries (Introduction to BETWEEN)

In a workshop, you might want to use tools or materials that fall within a certain size range. Similarly, the BETWEEN operator in SQL helps us find data that falls within a specified range.

Setting the Boundaries (Introduction to BETWEEN) (Cont.)

Finding Customers within an ID Range

Suppose you want to find all customers whose IDs fall between 1 and 10.

This command is like looking for all the tools on your workbench that have serial numbers between 100 and 200. It ensures you only pick out the tools within this specified range.

```
-- Finding customers with IDs
between 100 and 200
SELECT * FROM Customers
WHERE id BETWEEN 1 AND 10;
```

Setting the Boundaries (Introduction to BETWEEN) (Cont.)

Finding Orders in a Date Range

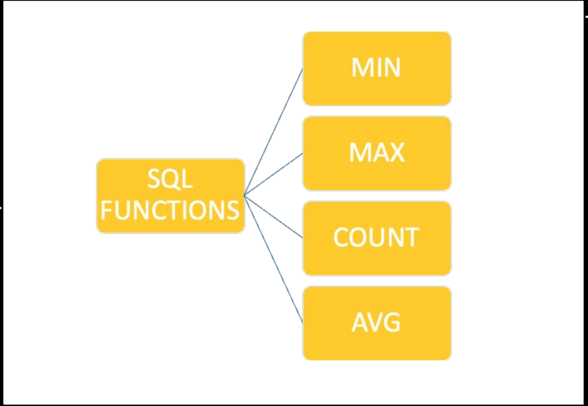
Imagine you want to find all orders placed between two dates (Note: This requires a bit of imagination, as our current Orders table hasn’t yet been populated with data).
This SQL command is like looking for all projects that were started and completed in the month.

```
-- Finding orders placed between
two dates
SELECT * FROM Orders
WHERE date BETWEEN
'2023-01-01' AND '2023-01-31';
```

Uncovering the Numbers:

Mastering Aggregate Functions in Your Data Workshop!

Think of these functions - MIN, MAX, COUNT, AVG - as your go-to measuring tapes and calculators, helping you to quickly assess and summarize various aspects of your data. Whether you're looking to find the smallest or largest value, count items, or calculate an average, these functions are key tools in your data analysis toolbox.



Uncovering the Numbers:

Mastering Aggregate Functions in Your Data Workshop! (Cont.)

Finding Extremes with MIN and MAX

Imagine you're looking for the smallest screw in your collection or the largest sheet of wood in your stock. In SQL, MIN and MAX functions help you find the lowest and highest values in a dataset.

Finding the Range of Customer IDs

To understand the spread of our customer IDs, we can find the lowest and highest IDs.
These commands help us identify the earliest and most recent customers in our database, based on their ID numbers.

```
-- Finding the smallest (oldest)
customer ID
SELECT MIN(id) FROM Customers;

-- Finding the largest (newest)
customer ID
SELECT MAX(id) FROM Customers;
```

Uncovering the Numbers:

Mastering Aggregate Functions in Your Data Workshop! (Cont.)

Discovering Price Extremes

Suppose you want to find the lowest and highest prices among your products. (Note: This requires a bit of imagination, as our current example doesn’t have a Products table structure.)

These commands are like measuring all your materials to find the smallest and largest ones.

```
-- Finding the lowest price
SELECT MIN(price) FROM Products;

-- Finding the highest price
SELECT MAX(price) FROM Products;
```

Counting Your Assets with COUNT

Counting Your Assets with COUNT

COUNT is like taking inventory in your workshop. It helps you determine the number of items in a certain category.

Counting the Number of Customers

If you're curious about how many customers you have, COUNT gives you that number.
This is similar to counting all the tools you have in a specific toolbox.

```
-- Counting all customers
SELECT COUNT(*) FROM Customers;
```

Averaging Out with AVG

Averaging Out with AVG

When you want to find an average size of a set of screws or an average amount of paint used per project, AVG is your go-to function in SQL.

Average Customer-Related Metric

Let's say we want to calculate the average of a numerical attribute related to customers, such as the average number of orders placed. (Note: This requires a bit of imagination, as our current Customers table structure doesn't have a direct numerical attribute like 'number of orders'.)
Here, num_orders would be a hypothetical column representing the number of orders each customer has placed. This command would then give us the average number of orders per customer.

```
-- Calculating the average number of
orders per customer (hypothetical)
SELECT AVG(num_orders) FROM
Customers;
```

Crafting Data Relationships:

INSERTing, UPDATEing, DELETEing and SELECTing Orders in Connection with Customers

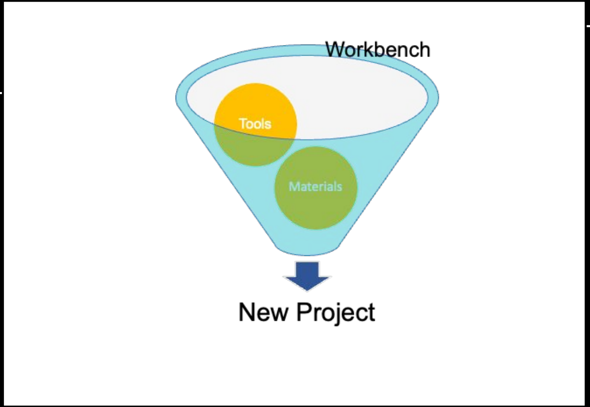
We're focusing on the art of data manipulation into our Orders table, especially in relation to our existing Customers. Just as in a workshop where projects often require different tools and materials to work together, in our database, creating orders involves linking them to existing customers.

Linking Orders to Customers:

The Relational Insert

Creating Connected Data Stories (INSERT with Relationships)

When we add a new order, it's crucial to link it to an existing customer in our Customers table. This is like ensuring that a new project on your workbench is associated with the right tools and materials already present in your workshop.



Linking Orders to Customers:

The Relational Insert (Cont.)

Adding an Order for an Existing Customer

Let's say customer 'John Doe' (whom we previously added) decides to place an order. We know his customer_id is a unique identifier that connects him to this new order.

This command creates a new entry in the Orders table, linked to John Doe's customer record.

```
-- Assuming John Doe has a
customer_id of 1
INSERT INTO Orders (date,
customer_id)
VALUES ('2023-07-15', 1);
```

Linking Orders to Customers:

The Relational Insert (Cont.)

The Importance of Valid References

In a workshop, starting a project without the necessary materials leads to complications. Similarly, in our database, adding an order for a non-existent customer can cause issues, as it violates the integrity of our relational data model.

Linking Orders to Customers:

The Relational Insert (Cont.)

Attempting to Insert an Order for a Non-Existent Customer

Imagine trying to add an order for a customer who isn't in the Customers table.

This would typically result in an error because customer_id 999 doesn't exist. It's like trying to use a tool that isn't in your workshop – it just doesn't work.

```sql
-- Trying to add an order for a
non-existent customer (assuming no
customer_id 999)
INSERT INTO Orders (date,
customer_id)
VALUES ('2023-07-15', 999);
```

Linking Orders to Customers:

The Relational Insert (Cont.)

Inserting Multiple Orders for Existent Customers

Let's say we want to add two new orders for customer 1 and one new order for customer 3.
This command is like setting up three different projects on your workbench, each carefully tagged to either customer 1 or customer 3's specific requirements or specifications.
Here's how we can do it efficiently:

```sql
-- Adding multiple orders for customers with IDs 1 and 3
INSERT INTO Orders (date, customer_id)
VALUES
('2023-08-01', 1),  -- First order for customer 1
('2023-08-02', 1),  -- Second order for customer 1
('2023-08-03', 3);  -- Order for customer 3
```

Fine-Tuning Existing Projects:

Using UPDATE in Orders

Precise Adjustments for Enhanced Outcomes (UPDATE)

In the workshop, updating a project might involve resizing a component or changing its color. Similarly, in SQL, we can update specific details of an order, such as changing the order date or linking it to a different customer.

Fine-Tuning Existing Projects:

Using UPDATE in Orders (Cont.)

Modifying an Order Date for a Specific Order

Suppose we need to update the order date for a specific order in our Orders table.
This command is like altering the deadline on one of your workshop projects – specific and targeted.

```
-- Updating the order date for
order_id 5
UPDATE Orders
SET date = '2023-09-01'
WHERE id = 5;
```

Clearing Space for New Creations:

Using DELETE in Orders

Removing Outdated or Unnecessary Elements (DELETE)

Just as you might clear out old or completed projects from your workshop to make space for new ones, the DELETE command allows us to remove certain orders from our database.

Deleting an Order Linked to a Specific Customer

Let's say a customer cancels an order, and we need to remove it from our database.
This command is akin to removing a specific project associated with a particular material or tool from your workspace. It's precise and ensures that only the intended item is removed.

```sql
-- Deleting an order for customer_id 1
DELETE FROM Orders
WHERE customer_id = 1 AND id = 1;
```

Clearing Space for New Creations:

Using DELETE in Orders (Cont.)

Attempting to Delete a Customer with Orders

What happens if we try to remove a customer who has orders linked to them? This is like trying to remove a toolbox that still contains tools. In SQL, this action can result in a conflict due to the existing relationship between the Customers and Orders tables.

Deleting a Customer with Existing Orders

If customer 1 has placed orders, this command might fail or cause integrity issues, depending on the database's referential integrity constraints.
Let's take a look at this example in the code.

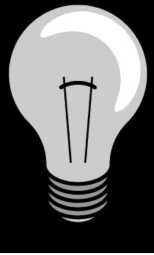
```sql
-- Attempting to delete a customer
who has placed orders
DELETE FROM Customers
WHERE id = 1;
```

Clearing Space for New Creations:

Using DELETE in Orders (Cont.)

IMPORTANT

In our data workshop, think of the Customers and Orders tables as interdependent sections - customers are like the main toolbox, and their orders are the tools inside it. These tools (orders) depend on the toolbox (customer) for their existence and relevance.



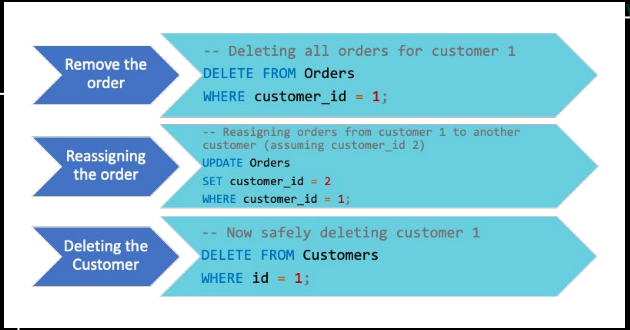
Clearing Space for New Creations:

Using DELETE in Orders (Cont.)

Safeguarding Data Integrity: Preventing Unintended Deletions

To safely remove a customer who has orders, we need a two-step approach:

Before deleting the customer, ensure their orders are either deleted (if appropriate) or reassigned to another customer. This is similar to either removing the tools from the toolbox or transferring them to a different toolbox. Once the orders are handled, you can safely delete the customer.



Remove the order

```sql
-- Deleting all orders for customer 1
DELETE FROM Orders
WHERE customer_id = 1;
```

Reassigning the order

```sql
-- Reasigning orders from customer 1 to another
customer (assuming customer_id 2)
UPDATE Orders
SET customer_id = 2
WHERE customer_id = 1;
```

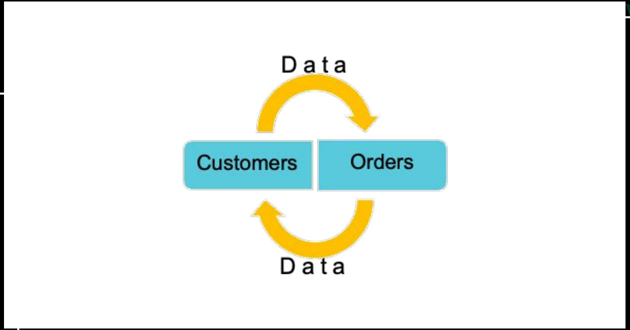
Deleting the Customer

```sql
-- Now safely deleting customer 1
DELETE FROM Customers
WHERE id = 1;
```

Crafting Queries Across Tables

Bridging Data from Separate Tables

We can query data from both Customers and Orders tables in a single SELECT statement by matching the id from Customers with the customer_id from Orders ensuring that we can view related information in a cohesive manner.



Crafting Queries Across Tables

Viewing Orders with Customer Details

Suppose we want to see details of orders along with information about the customers who placed them. This command allows us to see which customer placed which order, along with the order details and customer information. It's like laying out all the tools and their corresponding manuals side by side for a complete view.

```sql
-- Selecting order details along
with customer information
SELECT o.id AS OrderID, o.date AS
OrderDate, c.id AS CustomerID,
c.name, c.email
FROM Customers c, Orders o
WHERE c.id = o.customer_id;
```

Crafting Queries Across Tables (Cont.)

Filtering Specific Orders and Their Customers

We can also refine this approach to look for specific orders or customers.

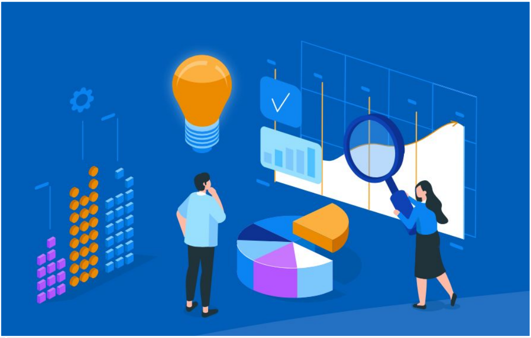
This command filters the orders to show only those placed by customers whose names start with 'Carol'. It's like selecting only those projects in your workshop that use a specific brand of tools.

```
-- Selecting details for orders
placed by customers with specific
criteria
SELECT o.id AS OrderID, o.date AS
OrderDate, c.id AS CustomerID,
c.name, c.email
FROM Customers c, Orders o
WHERE c.id = o.customer_id AND
c.name LIKE 'Carol%';
```

Navigating the Maze of SQL:

Best Practices and Common Pitfalls in Our Data Workshop

In our journey through the SQL workshop, we've learned to handle various tools - from DDL to DML, and DQL. Like any skilled craftsman, it's essential to know not just how to use these tools but also the best practices and common pitfalls associated with them. Let's dive into some key insights that will keep your data projects running smoothly and efficiently.



In-Class Exercises: Hands-on Coding!

Hey there! Before we unveil our solutions, how about taking a shot at it yourself? Keep in mind that we've provided some broad strokes, so a detailed approach will help fine-tune the solution. But, this is all in good fun and a chance for you to flex those problem-solving muscles. Give it a go, and remember, it's all about learning and improving!

Conclusion

Just as each tool in a physical workshop serves a specific purpose and demands careful handling, each SQL command has its designated role in the database realm. From creating structures with DDL to manipulating and querying data with DML and DQL, we've equipped ourselves with a comprehensive toolkit for diverse data tasks. As we proceed with building and managing our data projects, let's prioritize precision, organization, and understanding relationships within our data workshop.

Thank you for joining me on this enlightening journey through SQL and MySQL. May the skills and insights gained serve you well in all your future data endeavors!