

# WPI

## **Comparative Study of Relational Databases for Computer Science Courses**

A Major Qualifying Project Report Submitted to the Faculty of the **WORCESTER  
POLYTECHNIC INSTITUTE** in Partial Fulfillment of the Requirements for the  
Degree of Bachelor of Science in Computer Science

By

Ethan Pollack

Axel Luca

Harrison Taylor

With the Advising of

Professor Wilson Wong  
Department of Computer Science

Professor Rodica Neamtu  
Department of Computer Science

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, see <http://www.wpi.edu/Academics/Projects>

## Abstract

This comparative study assesses three leading database systems: Oracle, Postgres, and MySQL. The aim of this MQP was to create systematic selection criteria for college instructors to assist them in choosing the most suitable database management system for teaching database courses. As many industries rely on databases, the choice of appropriate database systems for the curriculum is crucial to prepare students for real-world applications. Our methodology included a thorough literature review of each system, practical evaluations via installations on multiple operating systems, and syntactical analyses through hands-on projects. The results show unique features of each system, with implications for how they fit within university curricula, depending on course design and goals.

## Acknowledgments

We would like to thank our advisors Professor Wilson Wong and Professor Rodica Neamtu for providing this project opportunity. Their support, insights, and guidance have been instrumental in the success of this research. Additionally, we extend our appreciation to the WPI Computer Science Department for teaching us a lot of topics that equipped us with the necessary skills and knowledge to undertake this project. We would also like to thank all our friends at WPI who have supported us all the way through our education. It would have been so much more difficult without them. Lastly, we would like to acknowledge the unwavering support and encouragement from our parents. They provided the foundation and motivation that allowed us to get to where we are today. Without any of these support systems, none of this would be possible.

## Table of Contents

<b>Abstract .....</b>	<b>ii</b>
<b>Acknowledgments .....</b>	<b>iii</b>
<b>Table of Contents .....</b>	<b>iv</b>
<b>Table of Figures .....</b>	<b>vii</b>
<b>1 Introduction .....</b>	<b>1</b>
<b>2 Background Research .....</b>	<b>2</b>
2.1 What is a database? .....	2
2.2 The importance of teaching about databases .....	4
2.3 Pedagogical approaches to teaching Database Concepts .....	4
2.4 Why Use a Relational Database? .....	6
2.5 Background on Oracle .....	7
2.6 Background on MySQL .....	7
2.7 Background on Postgres .....	8
2.8 Free web-based SQL editors for students and instructors .....	9
<b>3 Methodology .....</b>	<b>12</b>
3.1 Project Methodology .....	12
3.2 Comparative Study .....	12
3.2.1 General Study .....	13
3.2.2 Academic Study .....	14
<b>4 Results and Discussion .....</b>	<b>17</b>
4.1 Installation Complexity .....	17
4.1.1 MySQL and Postgres .....	17
4.1.2 Oracle .....	17
4.1.3 Analysis and Discussion .....	17
4.2 Setup Complexity .....	18
4.2.1 Analysis and Discussion .....	19
4.3 Proprietary IDEs .....	19
4.3.1 Oracle .....	19
4.3.2 MySQL .....	20
4.3.3 Postgres .....	21
4.3.4 Analysis and Discussion .....	21
4.4 Cloud Pricing .....	22
4.4.1 Oracle .....	23
4.4.2 MySQL .....	23
4.4.3 Postgres .....	23
4.4.4 Analysis and Discussion .....	23
4.5 Datatypes .....	24



4.5.1	Numerical Datatypes .....	24
4.5.2	Date and Time Datatypes .....	25
4.5.3	String Datatypes.....	26
4.5.4	Other Notable Datatypes.....	27
<b>4.6</b>	<b>Table Management .....</b>	<b>28</b>
4.6.1	Schema Used .....	28
4.6.2	Oracle.....	28
4.6.3	MySQL and Postgres .....	29
4.6.4	Analysis and Discussion .....	32
<b>4.7</b>	<b>Data Manipulation .....</b>	<b>32</b>
<b>4.8</b>	<b>Query Simplicity .....</b>	<b>33</b>
<b>4.9</b>	<b>Advanced Functionality.....</b>	<b>33</b>
4.9.1	Views.....	34
4.9.2	Procedures .....	34
4.9.3	Functions.....	37
4.9.4	Triggers .....	40
4.9.5	Advanced Functionality Analysis and Discussion .....	42
<b>4.10</b>	<b>Java Database Connectivity (JDBC) .....</b>	<b>43</b>
4.10.1	Schema Used.....	43
4.10.2	Driver Installation.....	43
4.10.3	Registering the Driver .....	44
4.10.4	Establishing a Connection .....	45
4.10.5	Analysis and Discussion.....	45
<b>4.11</b>	<b>Connecting to the database using Python .....</b>	<b>46</b>
4.11.1	Schema Used.....	46
4.11.2	Oracle.....	46
4.11.3	MySQL.....	48
4.11.4	Postgres.....	49
4.11.5	Analysis and Discussion.....	51
<b>5</b>	<b>Future Works .....</b>	<b>52</b>
<b>6</b>	<b>Conclusion .....</b>	<b>53</b>
<b>7</b>	<b>Works Cited .....</b>	<b>54</b>
<b>8</b>	<b>Appendices .....</b>	<b>62</b>
8.1	Appendix A: Oracle Installation for macOS through Docker .....	62
8.2	Appendix B: Using Oracle on macOS through Docker .....	66
8.3	Appendix C: Oracle Installation for Windows .....	68
8.4	Appendix D: Using Oracle on Windows .....	72
8.5	Appendix E: Postgres Installation for macOS.....	76
8.6	Appendix F: Using Postgres on macOS .....	78
8.7	Appendix G: Postgres Installation for Windows.....	80
8.8	Appendix H: Using Postgres on Windows.....	84

<b>8.9</b>	<b>Appendix I: MySQL Installation for MacOS.....</b>	<b>88</b>
<b>8.10</b>	<b>Appendix J: Using MySQL on MacOS.....</b>	<b>90</b>
<b>8.11</b>	<b>Appendix K: MySQL Installation for Windows .....</b>	<b>94</b>
<b>8.12</b>	<b>Appendix L: Using MySQL on Windows.....</b>	<b>99</b>

## Table of Figures

Figure 1: Example relational database schema for users, posts, and followers.	6
Figure 2: Postgres Statistics (Postgres, n.d.-a)	9
Figure 3: Schema used throughout the Table Management section. Note: Authors.email is a unique key. Books.author_id is a foreign key with Authors.author_id. Reviews.book_id is a foreign key with Books.book_id.	28
Figure 4: SQL Code to create an Authors table in OracleSQL.	28
Figure 5: SQL Code to drop the table named Reviews in OracleSQL.	29
Figure 6: SQL Code demonstrating how in MySQL, users can use either the ENUM syntax or the CHECK syntax to limit specific values in a table.	30
Figure 7: SQL Code demonstrating how in PostgreSQL, users can either use a custom defined ENUM type or the CHECK syntax to limit specific values in a table.	30
Figure 8: Valid Table Creation in MySQL and PostgreSQL. Uses ON UPDATE CASCADE.	31
Figure 9: Valid Table Creation in MySQL and PostgreSQL. Uses ON UPDATE SET NULL.	31
Figure 10: The 'IF EXISTS' option that can be used in MySQL and PostgreSQL to drop the 'Reviews' table if it already exists.	32
Figure 11: Example query that is valid in OracleSQL, PostgreSQL, and MySQL.	33
Figure 12: Schema used throughout the procedures section. Note: Employee.locationID is a foreign key with Location.locationID.	34
Figure 13: A procedure defined in OracleSQL used to print out information about an employee's location based on their name.	34
Figure 14: A procedure defined in PostgreSQL used to print out information about an employee's location based on their name.	35
Figure 15: A procedure defined in MySQL used to print out information about an employee's location based on their name.	36
Figure 16: Schema used throughout the functions section. Note: MedicalEquipment.locationID is a foreign key with Location.locationID.	37
Figure 17: A function defined in OracleSQL to get the equipment count based on a location's short name.	37
Figure 18: A function defined in PostgreSQL to get the equipment count based on a location's short name.	38
Figure 19: A function defined in MySQL to get the equipment count based on a location's short name.	38
Figure 20: Schema used throughout the triggers section. Note: TransportRequest.itemID is a foreign key with MedicalEquipment.itemID.	40
Figure 21: A trigger defined in OracleSQL that errors when specific equipment types are inserted into the TransportRequest table.	40
Figure 22: A trigger defined in PostgreSQL that errors when specific equipment types are inserted into the TransportRequest table.	41
Figure 23: A trigger defined in MySQL that errors when specific equipment types are inserted into the TransportRequest table.	41
Figure 24: Schema used throughout the JDBC section.	43
Figure 25: OracleSQL Driver downloads page. Note: the first two .JAR files are typically the correct downloads for an introduction course.	44
Figure 26: MySQL Driver downloads page.	44
Figure 27: Postgres Driver downloads page. Note: the orange 'Download' buttons will download the .JAR file.	44
Figure 28: Java code for locating the Oracle Driver.	45
Figure 29: Java code for locating MySQL Driver.	45
Figure 30: Java code for locating Postgres Driver.	45
Figure 31: Java code to open an Oracle driver connection. Note: <Driver>, <Host Name>, <Port>, <SID>, <User ID>, and <Password> should be replaced with the type of driver, the host name, the port number, the SID of the SQL server, the user ID, and the password respectively.	45
Figure 32: Java code to open a MySQL driver connection. Note: <Host Name>, <Port>, <Database Name>, <User ID>, and <Password> should be replaced with the host name, the port number, the database name, the user ID, and the password respectively.	45

Figure 33: Java code to open a Postgres driver connection. Note: <Host Name>, <Port>, <Database Name>, <User ID>, and <Password> should be replaced with the host name, the port number, the database name, the user ID, and the password respectively.	45
Figure 34: Schema used for the Python Connection section.	46
Figure 35: Terminal command to install the library that allows Python to connect to Oracle.	46
Figure 36: Line of code in a Python project to import the cx_Oracle library.	46
Figure 37: ZIP format available for the Oracle Instant Client.	47
Figure 38: Python code to initialize the Oracle client. Note: '/path/to' should be replaced with the appropriate file location of the Oracle Instant Client.	47
Figure 39: Python code that sets up an Oracle connection with a user ID, password, and a DNS. Note: The DNS should be formatted as "<host name>:<port>/<SID>".	47
Figure 40: Python code that creates a prepared statement which allows SQL queries to be executed in Oracle. Note: the ':1' and ':2' are placeholders for the <Insert Password> and <Insert Employee ID>.	48
Figure 41: Terminal command to install the library that allows Python to connect to MySQL.	48
Figure 42: Line of code in a Python project to import the mysql.connector library.	48
Figure 43: Python code that sets up a MySQL connection with a user ID, password, host, port, and database name.	49
Figure 44: Python code that creates a prepared statement which allows SQL queries to be executed in MySQL. Note: the '%s's are placeholders for the <Insert Password> and <Insert Employee ID>.	49
Figure 45: Terminal command to install the library that allows Python to connect to Postgres.	50
Figure 46: Line of code in a Python project to import the psycopg2 library.	50
Figure 47: Python code that sets up a Postgres connection with a database name, host, and port.	50
Figure 48: Python code that creates a prepared statement which allows SQL queries to be executed in Postgres. Note: the '%s's are placeholders for the <Insert Password> and <Insert Employee ID>.	50
Figure 49: Summary of comparisons across Oracle, Postgres, and MySQL	53

## 1 Introduction

A known impact that the creation of the internet had on society was that it allowed a tremendous amount of data on almost any topic to be accessed, obtained, and stored (Firth et al., 2019). From e-commerce websites to offline sudoku apps, most applications need to create and manipulate data for the benefit of the end-users. For simpler applications that do not have a lot of data and may not be concerned with high-end security, storing data as raw text files on device is not problematic and may suffice. However, as data storage requirements increase and larger applications need increased security, databases provide the most efficient solution.

A database management system (DBMS) is needed to effectively maintain a database. A DBMS references an entire ecosystem of software that is used to create, host, and interact with a database. For instance, before running the database, several steps must occur: (1) a server must be downloaded and hosted, (2) users must be created with permissions, (3) the user must log in and create the framework to store data. After completing these steps, the client can then store data in the database. This process can be very complex, which has resulted in companies creating proprietary DBMS so that server administrators can easily create, host, and interact with their database. Despite this, as of 2023, the most popular type of database being used in general is still the relational database, with Oracle now having been the leader for over 10 years (DB-Engines, 2019; DB-Engines, 2023).

In this project, we explore three relational database management systems (DBMS): Oracle, MySQL, and Postgres. Each DBMS has their proprietary Structured Query Language (SQL) to interact with the respective database. Although they all use the same relational guidelines, the specific implementation differs enough to provide noticeable benefits and deficits in a variety of factors including availability, cost effectiveness, ease of use, performance of key functions, etc. The objective of this research is to generate criteria to assist college instructors in deciding the most appropriate system for their database courses. To accomplish this goal, we conducted an in-depth review of each system, practical evaluations via installations on multiple operating systems, and syntactical analyses through hands-on projects.

With database instruction in academia becoming an increasingly important aspect of computer science, it is essential that professors are aware of the variety of tools at their disposal (Motro, 1993). To this end, highlighting the drawbacks and benefits of each relational database provides professors with critical information to guide them in choosing the best database for their course. In addition, this project provides students with better materials to facilitate access to database technology to help them learn the fundamentals and apply what they have learned in the real-world.

## 2 Background Research

For this MQP report, the Background research addresses multiple aspects of working with databases. These aspects include the importance of teaching about the topic of databases along with some of the pedagogical approaches universities use to do so, reasons why one would choose a relational database as their type of database to use, and some background on the three databases that are being compared for this project. After all of this, the Background research concludes with a section about what are some free web-based SQL editors that students and instructors can use while getting all their database course materials ready.

### 2.1 What is a database?

In general, a database is defined as “an organized collection of structured information, or data, typically stored electronically in a computer system”, and includes 12 major subsets: relational databases, object-oriented databases, distributed databases, data warehouses, NoSQL databases, graph databases, open-source databases, cloud databases, multi-model databases, document/JSON databases, autonomous databases, and hierarchical databases. Each of these databases have similarities and differences in the way they store and handle data, as well as functions and capabilities that are beneficial for specific projects (Oracle, 2022).

Relational databases, which became dominant in the 1980s, organize items in tables with rows and columns, offering the most efficient way to access structured data (Oracle, 2022). Additionally, these databases improve productivity because they offer data independence, structural simplicity, and relational processing (Codd, 1982).

Object-oriented databases represent information as objects, drawing inspiration from the structure in object-oriented programming (Oracle, 2022). They are designed to store and manage complex data objects mirroring those in object-oriented programming languages. These databases ensure object data persistence, allowing stored objects with their properties to persist even after program termination, offering a transparent process for storing and retrieving these complex objects (MongoDB, n.d.-a).

Distributed databases comprise files spread across multiple sites and may exist on numerous computers in varied locations, either locally or over diverse networks (Oracle, 2022). Some key features that these databases provide include transparent management of distributed and replicated data, reliable access to data through distributed transactions, improved performance, and easier system expansion (Özsu et al., 1999).

Data warehouses act as central data repositories and are specialized databases optimized for swift query and analysis tasks (Oracle, 2022). In addition to this, these databases are optimized for retrieving data through multidimensional queries, which allows companies to make important decisions for their business, and ultimately increase their profits (Haxhiu, 2018).

NoSQL databases, also known as "not only SQL", utilize storage designs that differ from the traditional relational tables. Depending on their data model, NoSQL databases can be categorized into primary types like document, key-value, wide-column, and graph. These databases offer adaptable schemas and are adept at handling vast data quantities and significant user traffic (MongoDB, 2019).

Graph databases uniquely represent data by focusing on entities and their interrelationships (Oracle, 2022). Graph databases can utilize native storage specifically designed for managing graphs or rely on relational or object-oriented databases, with the former often being faster. Native graph processing, or "index-free adjacency", efficiently processes data as connected nodes physically reference each other, unlike non-native engines which use alternate methods for Create, Read, Update, and Delete operations. An example of such databases would be Twitter, a graph database that connects over 300 million monthly active users (Sasaki et al., 2018). A subset of graph databases is online transaction processing databases, which are designed to efficiently handle a vast volume of transactions from multiple users (Oracle, 2022). Online Transaction Processing is a type of data processing activity that handles multiple simultaneous transactions, such as online banking, shopping, order placements, or text messaging. Traditionally, these activities are viewed as financial or economic transactions and are documented and safeguarded to ensure businesses can retrieve the data for accounting or reporting needs (Oracle, 2020).

Open-source databases come with open-source code and can encompass both SQL and NoSQL types (Oracle, 2022). Today, open-source databases now support a vast array of modern applications, ranging from popular mobile applications to leading eCommerce platforms. Examples of such databases include MySQL, Postgres, MongoDB, and Redis (Amazon Web Services, n.d.).

Cloud databases store data on cloud platforms and can either be traditional or function as a Database as a Service (DBaaS), with the latter having a service provider overseeing administration and maintenance (Oracle, 2022).

Multi-model databases stand out by integrating various database models, allowing them to cater to a range of data types such as relations, documents, graphs, and objects within a unified management system (Oracle, 2022; Płuciennik et al., 2017).

Document/JSON databases are tailored for document-oriented tasks and modernly save data in JSON format instead of conventional rows and columns (Oracle, 2022). These databases offer flexibility in storing various types of data and easily accommodate changes in a data model (MongoDB, n.d.-b).

Self-driving or autonomous databases are revolutionary in design; being cloud-oriented, they utilize machine learning to automate numerous database management tasks, minimizing manual administrative intervention (Oracle, 2022).

Hierarchical databases, among the earliest database types, structure data in a tree-like model where each record has a singular "parent." This parent-child relationship is represented using pointers, with tables branching out from a central 'root'. A tree-structure diagram resembles a data-structure diagram found in a network model. While the network model arranges records in the layout of a random graph, the tree-structure diagram organizes them in the pattern of a rooted tree (Domdouzis et al., 2021).

## 2.2 The importance of teaching about databases

The topic of databases has now been a part of a core curriculum in computer science for multiple decades. Because of this, in universities, there are courses available at both introductory and advanced levels that teach about a variety of subtopics that relate to Databases. Examples of these subtopics are data modeling and design, query optimization, ER diagrams, views, triggers, procedures, functions, and relational algebra. To teach about these subtopics, commonly used database systems include MySQL, Oracle, Postgres, and even NoSQL (Aggarwal et al., 2020; Bi & Beidler, 2008; Fekete & Röhm, 2022.; Jiang & Nandi, 2015; Udoh, 2006; Fekete, 2005).

There are numerous reasons why teaching the topic of databases is very important in computer science education. One key reason is that technology companies create an increasing number of digital applications that rely on database systems to store data. Example types of such web-based applications include online encyclopedias, social media websites, CRM systems, e-commerce websites, and email systems (MongoDB, n.d.-c). Another important reason for which teaching the topic of databases in a computer science education is important is that the types of industry employing CS graduates may rely on databases for what they need to do even though they do not relate primarily relate to technology. Some examples of such types of industry include commercial, social activities and scientific (Deng et al., 2004). Because of this, no matter the type of industry, having computer science employees that handle data management is essential (Bi & Beidler, 2008). This leads to companies of various types of industry offering various types of positions that require knowledge of databases such as software developers, database administrators, and data analysts (Fekete & Röhm, 2022). Therefore, it is essential for any university that offers a computer science curriculum to give students a solid database system foundation so that they can learn all the skills that they will need in order to be eligible to fill up these kinds of positions (Huang, 2019).

## 2.3 Pedagogical approaches to teaching Database Concepts

Because not all university curricula will approach the topic of databases depending on what their learning objectives are, different classes at different colleges will use different methods to teach it (Aggarwal et al., 2020; Fekete & Röhm, 2022; Mior, 2023). Some universities focus more



on teaching database concepts while others focus more on giving students realistic hands-on experience to prepare them for working in the industry after graduating from college.

One of these methods that some universities implement is the use of visual software to help students understand specific subtopics related to databases better. An example of a university that used this method to help its students understand specific subtopics related to databases better is Ohio State University. To help its students understand the topic of relational algebra, the Ohio State University developed an interactive textbook with features such as being able to connect to a live database and giving users the ability to explore the projection operation by tapping on attributes of a relation, and completely rewriting relational algebraic expressions (Jiang & Nandi, 2015).

Another method some universities such as the University of Wisconsin - Eau Claire, the Macalester College, and the University of Minnesota use to teach the topic of databases is using actual scientific datasets in their course assignments. There are multiple benefits to using this method to teach the topic of databases to students. These include the fact that scientific datasets help students learn to model and work with complex data, which prepares them for future employment in database systems, and that they help students gain an understanding of interdisciplinary work, which can be developed through communication with professionals in the relevant field (Wagner et al., 2003).

One more common method used by universities to teach the topic of databases to students is having students work on large-scale team projects where they can apply all the skills that they learn in class regarding the topic of databases. Universities that use this method include Purdue University and Wentworth Institute of Technology (Rilett & Russo, 2013). As a first example of such projects, in its introductory database course, Purdue University required its students to execute a component in a “client-server (two-tier) architecture, with a client application like Oracle Forms/Reports interacting with the Oracle backend database server” (Udoh, 2006). Another example of those projects that is worth mentioning is one that was conducted in an introductory database course at Wentworth Institute of Technology. For this project, students initially designed “a database which would then be used by the web site to store information such as a shopping cart or payment information. The database itself was initially hosted on AWS using an instance running MySQL. Towards the end of the course, the students migrated their databases to the Relational Data Service (RDS) provided by Amazon and used that to set up a virtual web server on several instances with S3 storage for all static content. Students also deployed an elastic load balancer which directs traffic to instances of the Web server” (Rilett & Russo, 2013).

A similar teaching method to this that also applies to teaching database concepts is Worcester Polytechnic Institute’s signature project-based learning approach used since 1970. With this method, students engage in completing projects that let them use what they have learned in class to solve real-world problems. Students work on these projects either in their own town or

in other parts of the world, collaborating in groups or working alone with guidance from teachers. Through these projects, students hone their critical thinking, research, speaking, and writing skills, which allows them to be better equipped for their post-graduation careers (Center for Project Based Learning» Project-Based Learning at WPI, n.d.; Project-Based Learning at WPI | PBL in Higher Education, n.d.).

## 2.4 Why Use a Relational Database?

Despite the large variety of major subsets of databases that exist, such as the types discussed earlier, relational databases have been widely used in both industry and academic database course settings for decades (Dolezel & McLeod, 2021). Relational database technology offers productivity improvements due to data independence, structural simplicity, and relational processing (Codd, 1982). A relational database is a kind of database that organizes data into tables where each row represents a unique record identified by a key. Based on the relational model, it uses columns to define data attributes, allowing for straightforward representation and easy linkage of related data points through their values (Oracle, 2023). Each table will have a set of rows where each row represents a different record. We can then set up a relation between two of these tables across common rows, such as an id or a name. When manipulating data in the database, a user provides values for each of these rows, where each row constitutes one record for that table. For example, in Figure 1 one would need to add the id, username, role, and created\_at values for each new user that one would want to store.

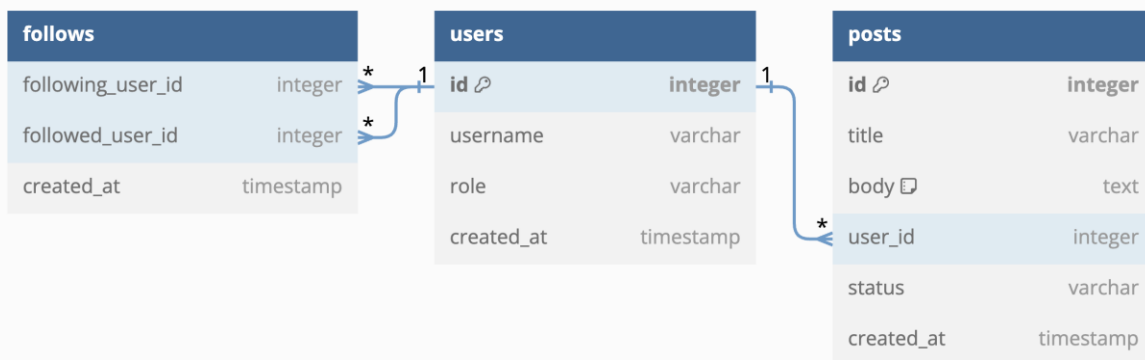


Figure 1: Example relational database schema for users, posts, and followers.

Using a Structured Query Language (SQL), a database developer can interact with their database in multiple ways. SQL supports a multitude of keywords that allow users to add, get, and remove data from any of the tables in the database. In addition to this, SQL has a very structured method of combining tables using the predefined relations on the tables. For example, in Figure 1, a user can search for all posts that match a certain users' id. SQL also has an efficient method of filtering and ordering results. This means that users can perform complex searches. For example, one can search for all users that have between 3 and 24 posts, and the database will return all

appropriate records. The combination of the structured table and relation setup and the SQL queries allow for a robust, secure, fast, effective, and usable database system (Codd, 1982).

## 2.5 Background on Oracle

Oracle is a company that provides tools for developers in various ways, including Cloud Computing, Customer Relationship Management (CRM), Docker, Kubernetes, and Python. One of the tools within the Cloud Computing category is the Oracle Database. The Oracle Database is a Relational Database Management System (RDBMS). Starting in 1979 with Oracle v2, every few years, or when new features are required, Oracle releases a new version of their database. As of 2023, when this paper was written, Oracle Database 21c is the current version with a very large number of features, efficiency upgrades, and industry breaking implementations (Oracle, n.d. -c). As a result, Oracle Database 21c is the version of the system that we used to conduct our research for this paper.

There are a few editions of Oracle Database available for these versions: Oracle Autonomous Database, Oracle Database Enterprise Edition, and Oracle Database Free. Oracle Cloud Infrastructure (OCI) is a product that allows one to host a variety of Oracle's tools on their servers anonymously. The Oracle Anonymous Database utilizes Oracle Cloud to handle all aspects of hosting, storing, and maintaining the servers that run a database. Although there is a free tier for this cloud version, past a certain threshold the cost goes up with the amount of data that one consumes (Oracle, n.d.-a).

Although only one of the Oracle Database Enterprise Edition and Oracle Database Free has the word 'free' in it, both can be downloaded and run for free at any point (Oracle, n.d.-b). The difference is that the enterprise edition is aiming to support large companies or projects while the free tier is more suitable for smaller applications. Both require the server administrator to maintain their own server to host the database. Setting up and maintaining the hardware and software required can be very time consuming and expensive, which is why in recent years cloud computing has become increasingly popular. However, for the purpose of this research paper, hosting the servers locally will allow us to compare the various DMBSs more effectively.

## 2.6 Background on MySQL

MySQL is an open-source relational database system, that is used for popular applications such as Facebook, Twitter, Netflix, Uber, Airbnb, and more. Since MySQL is open source, it is under continuous development by a large number of users who lend their expertise to MySQL with the collective goal of improving the system (MySQL, n.d.-b). MySQL originated from Scandinavia in 1979, created by Michael "Monty" Widenius who worked for the Swedish company TcX. MySQL includes an SQL server, client programs for accessing the server, administrative tools, and a programming interface for writing new programs. Initially MySQL took off due to its speed and simplicity, however it lacked certain features such as transactions and foreign key support. Since

then, developers have added those features and others such as replication, subqueries, stored procedures, views, and triggers (DuBois, 2009).

One key benefit of MySQL is its reliability, as it has been tested in a wide variety of scenarios and is used in large companies as it can handle complex databases with billions of rows (DuBois, 2009; MySQL, n.d.-b). Additionally, MySQL is easy to install and manage, and it can be easily scaled to meet the demands of users and companies. MySQL has also been proven to be faster and less expensive than other database services based on the industry benchmarks of TPC-H, TPC-DS, and CH-benCHmark. This database also allows developers to develop both SQL and NoSQL schema-free database applications, while also mixing relational data and JSON documents. Another benefit of MySQL is that it is multi-threaded, meaning that multiple clients can connect to it at the same time and a client can use multiple databases simultaneously (MySQL, n.d.-b).

MySQL incorporates features to support the needs of an ever-growing industry. One new product includes MySQL HeatWave which is a fully managed database service for transactions, real-time analytics, and machine learning services. HeatWave is available on OCI, AWS, and Azure, and is known for its simple structure and low cost for ETL duplication. Another product is MySQL Enterprise Edition which has the most features of any MySQL product including MySQL Enterprise Transparent Data Encryption (TDE), MySQL Enterprise Masking and De-identification, MySQL Enterprise Backup, MySQL Enterprise Authentication, and more. These features make MySQL Enterprise Edition the most scalable, secure, and reliable MySQL product. MySQL has also produced MySQL for OEM/ISV which has allowed over 2000 ISVs, OEMs, and VARs to implement MySQL. Finally, MySQL produced MySQL Cluster which allows users to meet the database challenges of cloud and communications services while also allowing it to scale (MySQL, n.d.-a).

Since MySQL is open source, their products are free to download, setup, and maintain. This includes MySQL Server, MySQL Workbench, and other tools needed to operate MySQL on devices. Like Oracle's products, once users begin hosting their databases on MySQL's servers, they will be charged for the data used. There are a variety of plans that depend on the specific types and quantities of data that they are storing. For smaller projects the free downloads for MySQL are a perfect way to start developing a very professional and robust server and database.

## 2.7 Background on Postgres

Like MySQL, Postgres is an open-source relational database system. This RDBMS has been in development for around 35 years (Postgres, n.d.-c). Originally named POSTGRES, this database system has a multitude of features other relational database systems do not offer, making it worth considering whenever choosing a database system (Postgres, n.d.-c; Lerner, 2007). From unique implementations of custom data types to stored procedures, Postgres is a viable and completely free alternative to paid DBMSs. Postgres is available on all major platforms including Windows,

macOS, Linux, and more (Postgres, n.d.-b). On top of this, there are numerous older versions that are still available for download for free (Postgres, n.d.-d, n.d.-c).

All database systems that incorporate SQL will contain numeric, text, and date/time datatypes included with all versions. However, PostgreSQL also implements other important features such as custom datatypes and composite datatypes. These allow users to store complex and custom data related to their project. This can be useful to help with the efficiency loss of splitting up all data into their atomic datatypes. On top of this, there is support for geometrical datatypes: point, line, circle, and polygon (PostgreSQL, n.d.-a).

Like MySQL Workbench, pgAdmin is an integrated development environment (IDE) for Postgres (pgAdmin, n.d.-a). This is a robust open-source tool that is used to interact with Postgres Servers. Included is a great query editor made for PostgreSQL, statistics visualizations of a user's data, file explorers, and much more to ease the development process. The current version is pgAdmin 4, and this can be installed in a variety of ways including Windows, macOS, Linux, Docker, and more (pgAdmin, n.d.-b).

35+ Years Development	700+ Contributors	54,000+ Commits	55+ Local User Groups
1,600,000+ Lines of C	675+ Events	Millions of Happy Users	∞ Data Stored

Figure 2: Postgres Statistics (Postgres, n.d.-a)

## 2.8 Free web-based SQL editors for students and instructors

When teaching any course in general, it is possible that not every student will have all the appropriate tools set up right on the first day of class, and that includes the database server(s)/client(s) that students are required to have to be able to complete an introductory database course. Fortunately, there are multiple ways for students to get access to SQL languages such as OracleSQL, MySQL, and PostgreSQL for free without having to download any software on their computers. However, even though these editors support a lot of key SQL functions such as creating functions and inserting statements, it is expected that students will eventually acquire all the database tools that they need to complete the database course(s).

If a database course uses OracleSQL as the SQL language, then an example of a free web-based SQL editor that one can use is called Oracle Live SQL. This free SQL editor version supports many SQL capabilities/operators such as creating tables, inserting data, triggers, and procedures. An additional benefit to this free SQL editor version is that there is no limit to the number of lines of code that one can type and run individually by highlighting them. Despite all the SQL capabilities/operators that Oracle LiveSQL supports, it does have its drawbacks. To begin, one will

not be able to use Oracle Live SQL until they create an Oracle account. In addition to this, if one tries to run more than approximately 375 lines of code at once by highlighting them, the software will not run those highlighted lines of code at all.

If a database course uses MySQL as the SQL language, then an example of a free web-based SQL editor that one can use is called phpMyAdmin. To begin using this service, one will first need to register by entering an email address that use on a free service called Free SQL Server. Once that is done, the newly registered user will receive an email with a link that will allow them to create their password to finalize their account for the service. Once the new user has their account finalized, they will be required to enter all their account details to log in for good and be able to fully utilize this service. Once logged in, the user simply must click on the “MySQL Hosting” tab in order to be directed to a page where they will need to put in the last bits of information required such as the server location to obtain their free database consisting of 5 megabytes of available space. Once all the required information has been filled in and saved, the user will receive an email consisting of all the information of their newly created database including the database server name, username, password, and port number. Now, all that’s left for them to do is go to the phpMyAdmin website and enter all their obtained database information to use the free SQL editor. Despite all the steps required to access this SQL editor, phpMyAdmin does have a lot of useful features that both students and instructors can use. One of these features is an SQL webpage that students and instructors can use to perform basic SQL tasks such as creating and deleting tables, inserting data into them, and then querying them. Another use of this SQL editor is a “structure” tab that students and instructors can go to manually manipulate their tables by doing things such as dropping them, browsing the data inside them, altering the tables’ column names, and empty them. Other features that this SQL editor provides are the ability for users to both import data from an input file, to export data from their tables into a file, to manually query their database without any SQL code, and to see a visual representation of how their created tables are related to each other.

If a database course uses PostgreSQL as the SQL language, then an example of a free web-based SQL editor that one can use is CoderPad. Even before a user signs up, they will have access to an SQL IDE that is divided into 2 parts. The left part of the IDE is where a user will be able to write PostgreSQL code perform basic SQL tasks like editing the already provided code, creating and deleting tables, inserting data, and querying their database. On the right side of the provided IDE, the user will have 3 different tabs. The first one gives out a set of instructions to solve a sample PostgreSQL interview problem, the second one lets the user see the output of their program when they run their code, and the third one gives them a visual representation of how their created database tables are related to each other. Although this should not be an issue to either students or instructors for the early days of the database course, it should be noted that this free SQL editor runs PostgreSQL 12.4. This means that they will not have access to the newest Postgres features and capabilities introduced in its later versions.

If one is looking for a free web-based SQL editor that supports more than one SQL language, those exist as well. A first example of these free SQL editors is called SQL fiddle (Fadlallah, 2021). SQL Fiddle is an online SQL editor that allows database developers to create, run, and share SQL queries and schemas. Developed in 2012 by Jake Feasel, this web-based SQL editor helps users build representative databases for troubleshooting, comparing SQL statements across different database back-ends, and testing queries in environments they may not have readily available (Feasel, 2012). Although SQL fiddle supports OracleSQL, MySQL and PostgreSQL, it only supports old versions of those languages such as MySQL 5.6 and Oracle 11g. Therefore, if one uses SQL fiddle, they will not have full access to all the SQL capabilities/operators that they will if they use a more modern version of those SQL languages such as Oracle Database 21c. Another option that one can use if they are looking for a free SQL editor if the course uses either MySQL or PostgreSQL as the SQL language is called db-fiddle. Unlike SQL fiddle, db-fiddle supports modern versions of its SQL language options such as MySQL 8.0 and PostgreSQL 15, meaning that a user will have access to a more complete set of SQL capabilities/operators that SQL fiddle does not support (DB-fiddle, n.d.). One of the best free SQL editors that students can use is called db<>fiddle (Fadlallah, 2021). This web-based SQL editor was originally designed to be used for “friendly markdown-based Q&A sites like Stack Overflow, TopAnswers and Codidact” and “readable for longer, multi-step, code”. Just like db-fiddle, db<>fiddle supports modern versions of its SQL language options such as MySQL 8.0 and PostgreSQL 15. However, db<>fiddle gives a user many more options to choose from in terms of SQL languages than db-fiddle. This is because in addition to the SQL languages that db-fiddle already supports, db<>fiddle supports other modern versions such as Oracle 23c, TimescaleDB 2.11, and Firebird 4.0 (DB<>Fiddle, n.d.). Despite all its benefits, db<>fiddle only lets a user create one SQL command per block. This can be very inconvenient if they are trying to copy and paste in large datasets in the form of SQL insert statements into this SQL editor. Luckily, it turns out that one way a user can get around this issue is to take advantage of the fact that modern versions of SQL languages let the user put multiple rows of data to insert in a table into a single SQL insert statement (Fadlallah, 2021). The only catch is that to use this workaround, the user needs to make sure that they are using the most up-to-date version available of their language of choice on this SQL editor. An example of such would be Oracle 23c, the most up-to-date version of Oracle available on db<>fiddle.

## 3 Methodology

### 3.1 Project Methodology

We utilized a well-known and efficient methodology for project management and development to efficiently manage and keep track of our tasks and workload.

Agile was the primary workflow style we used while gathering our research and keeping ourselves on track. Through a software design methodology, Agile proved to be very effective in the organization of tasks, subtasks, and research flow, even for our more research-oriented project. It allowed iterative versions of our research that helped us to see how far we'd come with each progression of our work.

An integral part of our usage of Agile is the project management software JIRA. JIRA captures user stories and objectives and tracks the progress of each issue and shows each user their specific issues during the development cycle(Sarkan et al., 2011). Utilizing this software, it was easy and simple to keep track of our objectives. Each time we generated a new user story or task, we were able to generate a task for it in our JIRA board swiftly and assign it to whoever was ready for it, or simply left it there until we discussed it later.

Our team met 3-4 times a week for a Scrum, which is a short meeting in Agile methodology, where we discuss what we have accomplished since our last meeting, and what we would ideally have done for the next meeting (Srivastava et al., 2017). As such, scrums allowed us to quickly resolve issues that would arise from disagreements or misunderstandings by constantly reviewing and sharing our progress and goals. The scrum is also where we would update our user stories, moving them around as we saw fit, to have our JIRA board as updated as possible.

Once a week we had a sprint, which is an integral part of the Agile Scrum methodology, in which we reviewed the goals we had set for ourselves over the past week, and discussed what we had accomplished, and what we still had to do (Hidalgo, 2019). Each of our sprints began with a sprint planning meeting where we set out our project objectives for the week. At the end of each of these sprints, we conducted a review meeting with our project advisors to present what was accomplished in relation to the outlined project objectives. This meeting was followed by a retrospective meeting to reflect on how closely our accomplishments aligned with those objectives. Sprint is where the overarching goals and user stories of the project are defined and is where most tasks are added to our JIRA board. It allows us an in-depth review of each other's progress while also offering a comprehensive outline of our work for the coming week.

### 3.2 Comparative Study

For this project, we used Oracle Database 21c, PostgreSQL 15, and MySQL 8.0.33 to ensure that our results and conclusions were based on the latest versions of each system available at the time of our research.



### 3.2.1 General Study

When generally comparing which database systems database systems it is important to consider a variety of factors. These factors include:

- Installation Complexity
- Setup Complexity
- Proprietary Integrated Development Environments (IDEs)
- Cloud Pricing
- Datatypes

#### 3.2.1.1 *Installation Complexity*

This criterion focused on the installation of local databases and their respective integrated development environments. Ideally, the installation of the required software is simple and easily manageable. This allows students of any experience level to begin to study relational databases without worrying about initial issues when simply setting up their software. Things such as installing, updating, setting up, learning, and running programming tools can be unnecessary burdens to students (Qiu et al., 2017). This comparison will primarily focus on ease of installation of the three SQL technologies. Any notable problems in the installation or frequent problems associated with the downloading of local databases and their respective IDEs will be considered, such as glitches, frequent errors, and notable complications. Any issues associated with specific operating systems will also be noted, as this process should be as simple as possible for all computers that would be using these technologies.

#### 3.2.1.2 *Setup Complexity*

Similar to installation complexity, this criterion focused on the difficulty of setting up user roles and databases themselves on a local or online server system. Once again, ideally this is simple and should just mean following a set of instructions. As mentioned above, things such as installing, updating, setting up, learning, and running programming tools can be unnecessary burden to students (Qiu et al., 2017). The setup of databases and users should be easy, swift, and simple across all popular operating systems, such as Windows, MacOS, and Linux. This will be evaluated in areas such as ease of access to a SQL console, number of configuration commands, simplicity of syntax, reliability of setup methods, change between operating systems or computer types, and so on.

#### 3.2.1.3 *Proprietary Integrated Development Environments (IDEs)*

To identify the strengths and weaknesses of the language's proprietary Integrated Development Environments (IDEs), we examined specific features supported by each IDE, as well as gauged how simple it is for users to access and use those features. While modern IDEs with lots of features have shown to significantly improve users' productivity in their tasks, if the IDE and its features are too complex to access and use, it can also lead to a decrease in users' productivity in their tasks (Zayour & Hajjdiab, 2013).

#### 3.2.1.4 *Cloud Pricing*

Cloud pricing was an important factor to investigate for our three database systems. This is because in general, good cloud pricing models help in making users buy the cloud services to be able to use them (Al-Roomi et al., 2013). We investigated and conducted the comparison of this aspect for each of these three database systems by researching it.

#### 3.2.1.5 *Datatypes*

Datatypes are the forms in which any given data is stored in a database. Examples of such datatypes include INTEGER or VARCHAR when storing numbers or strings respectively (Melton, 1996). Advanced data types can be incredibly useful for the purpose of efficient data storage, and many languages have a wide variety to choose from. While this is useful in many applications, unfortunately, some languages may utilize datatypes that are less intuitive than others. We weighed the primary datatypes of each popular SQL language to gauge both how useful they are and how intuitive they are.

### 3.2.2 *Academic Study*

The criteria outlined in the Academic Study section are significantly more academically centered. These are integral to the classroom experience while learning SQL, and more specific than the General Study. Many of the criteria focus on simplicity and ease of access for students attempting to learn relational database concepts. Using a syntactically simple language instead of a more complex one facilitates students' learning of programming concepts, allowing students a more streamlined experience (Koulouri et al., 2014). For this reason, many of the criteria focus on the simplicity of the SQL languages being evaluated. For each of the criteria, we utilized hands on projects to fully explore and identify the major similarities and differences between the three SQL languages. These criteria are:

- Table Management
- Data Manipulation
- Query Simplicity
- Advanced Functionality
- Java Database Connectivity (JDBC)
- Python Database Connectivity

#### 3.2.2.1 *Table Management*

The creation and deletion of tables are fundamental when comparing SQL languages. The table is one of the most important concepts in relational databases, as it serves as the structure of data storage, so an SQL language should make it as easy and intuitive as possible to create, edit, and delete tables. In addition, table constraints, which are limitations placed on the table, should be handled easily and intuitively. All of this is important because when students work on projects that rely on databases, creating tables can have roles in the project that are crucial to its

functionality (Gallini et al., 2022). We investigated the creation, updating, and deletion of tables and constraints in all three SQL languages, and evaluated them against each other.

#### *3.2.2.2 Data Manipulation*

In this criterion, we investigated the simplicity of performing basic SQL operations such as retrieving, inserting, altering, and deleting records in tables. In a course that involves the use of databases, students are expected to master such SQL data manipulation operations so that they can apply them to more advanced computer science topics such as web programming (Maiorana, 2014). Once our projects have been completed, we analyzed all the syntactical differences that these operations have presented across the three systems' SQL languages.

#### *3.2.2.3 Query Simplicity*

Querying a relational database means extracting data from it in a specific way to get the data desired by the users. Querying is entirely based on pre-established logic which will remain consistent regardless of language, meaning ideally the SQL language would make it as easy as possible to translate logic mapping into SQL code (Eder, 1992). We intended to find the querying system that aligns most intuitively with the logic behind it, as it will likely represent one of the best choices for students who are learning relational database concepts.

#### *3.2.2.4 Advanced Functionality*

Views, triggers, functions, and procedures are some of the more advanced concepts taught in an introductory database systems course. Triggers are functions that can perform an action when values are inserted or removed to a given table, and functions and procedures act similarly to how they do in other coding languages, executing a block of code when called (Ceri et al., n.d.). Even so, it is important to note that functions and procedures are not part of the universal SQL standard and are specific additions to SQL dialects. Like other language criteria, these should be easily conceptualized and learned by students. They should be formatted in a way that allows the ideas behind them to be easily understood as overly complicated syntax could have a negative impact on the students' overall learning experience. We gauged functionality by testing to see if some languages have triggers, functions, or procedures that are capable of features that other languages do not have, as well as evaluated the simplicity and comprehensibility of these advanced operations.

#### *3.2.2.5 Java Database Connectivity (JDBC)*

When comparing the aspect of Java Database Connectivity (JDBC) for each of these three database systems, we began by delving into the differences within the specific processes they require to successfully establish a connection to the database through Java. Additionally, we also checked whether there were any syntactic differences in the Java code required to iterate through, retrieve, and update data within their associated database. JDBC provides a standardized interface for connecting certain types of web-based applications such as e-businesses to databases and ensuring consistent data management across various platforms and servers (Wang, 2003). We

investigated and conducted the comparison of this aspect for each of these three database systems by first researching and manually going through their respective processes required to successfully connect to the database through Java. Lastly, we analyzed and discussed the differences found in the Java code between the three systems.

#### *3.2.2.6 Python Database Connectivity*

To determine the differences in Python code in each SQL technology, we began by delving into the differences within the specific processes they require to successfully establish a connection to the database through Python. Additionally, we also checked whether there were any syntactic differences in the Python code required to iterate through, retrieve, and update data within their associated database. Connecting Python to a database allows for automated extraction and storage of specific information, such as customers' transactions from bank account statements, for further automated analysis, thereby reducing manual work and potential inaccuracies (Nandi, 2021). We investigated and conducted the comparison of this aspect for each of these three database systems by first researching and manually going through their respective processes required to successfully connect to the database through Python. Lastly, we analyzed and discussed the differences found in the Python code between the three systems.

## 4 Results and Discussion

### 4.1 Installation Complexity

In the case of installation complexity, we found that for the most part, and on most systems, the Installation of the servers and proprietary IDEs are similar, with a couple notable outliers.

#### 4.1.1 MySQL and Postgres

Both MySQL and Postgres have similar and intuitive installation processes. It asks for standard database information, such as port number, file location, and a root username and password. Both installation processes take relatively short amounts of time, while also installing their proprietary IDEs and other useful tools for the user, in a straightforward and intuitive way. The installers are also easy to locate on their respective websites, making the whole process easy to follow in every step. Additionally, the installation processes are almost the exact same between Windows and MacOS, two of the most popular operating systems, meaning this will work smoothly for a wide variety of computer architectures. For the complete MySQL and Postgres installation processes, refer to Appendix E: Postgres Installation for macOS, Appendix G: Postgres Installation for Windows, Appendix I: MySQL Installation for MacOS, and Appendix K: MySQL Installation for Windows.

#### 4.1.2 Oracle

Oracle is a notable outlier when it comes to the ease of product installation. Though the Windows installer is comparable to Postgres and MySQL in ease of use, and installs the server without a hitch, it does not install the proprietary IDE for the user, creating an extra step that could be avoided through the user of a different language. However, this issue is trivial compared to the steps that are required to install Oracle on a macOS system. For Oracle to be installed on an Apple computer, it is required to use either Docker or a Virtual Machine, complicating the process significantly. In addition to this complication, while it is possible to install the Oracle server on Docker using the same terminal commands as with a macOS with an Intel chip with a macOS with an Apple chip, the latter requires a few extra installations and terminal commands such as a tool called Colima to complete the process required to do so. For the complete Oracle installation process, refer to Appendix A: Oracle Installation for macOS through Docker and Appendix C: Oracle Installation for Windows.

#### 4.1.3 Analysis and Discussion

Overall, the installation of the three different servers is typically very easy to access and perform. Postgres and MySQL seem to be the most consistent and easiest to follow, and though Oracle is certainly viable, it has the most difficulties of the three by a significant amount, being substantially more difficult on macOS as opposed to the two other languages.

## 4.2 Setup Complexity

Once the user has successfully installed all the software required to use Oracle, MySQL and/or Postgres, they are then able to begin the steps to set up a SQL server, including authentication for themselves and user privileges. For setting up authentication for themselves and user privileges, all three systems require the user to do all this by connecting to their database server through the terminal using either the systems' default root/admin, or the root/admin user they had to make when installing all the software required to use these systems if one is not already provided. After all that is done, for all three systems, the user can open their IDE environment using both their newly created authentication and details relating to how to connect to their database server.

The first step in creating users is to access the SQL server from the terminal. This will allow the user to run some commands directly in the server to interact with it. Oracle requires the most amount of upfront work to get into the server from the terminal. However, MySQL and Postgres both are more straightforward.

For users using Oracle on macOS, they must use Docker to simplify the process of running the SQL server. This process requires several extra commands that must be run from the command line, or through docker's GUI, to access Oracle through the terminal. Once they have access to the Oracle database through their terminal, both MacOS and Windows users alike will be prompted to enter a username to fully connect to their Oracle server. Oracle's default root/admin username is `"/ as sysdba`, and that's what users must enter to connect to the Oracle database through the terminal for the next steps of the Oracle setup process.

Unlike Oracle, MySQL and Postgres do not need Docker on either macOS or Windows. This means that the number and complexity of the terminal commands used to connect to the database is significantly less. Once in the MySQL server in their terminal, users can start creating authentications for themselves in a very similar method to how it is done in Oracle. Accessing Postgres in the terminal is very similar to accessing MySQL. It just requires running a specific command with the correct credentials to open the connection.

The next steps of set up process revolve around a user setting up authentication for themselves, and user privileges. All three languages follow a very similar process to accomplish this. The first step, once the server is running, is to create a user to interact with the server. Then, they can grant all the privileges that they need for interacting with a database. For the complete setup processes for all three systems, refer to Appendix B: Using Oracle on macOS through Docker, Appendix D: Using Oracle on Windows, Appendix F: Using Postgres on macOS, Appendix H: Using Postgres on Windows, Appendix J: Using MySQL on MacOS, and Appendix L: Using MySQL on Windows.

#### 4.2.1 Analysis and Discussion

Based on the setup processes of the three systems detailed above, it is evident that the complexities involved in each setup differ around the steps to access the database through the terminal, for users to create authentication for themselves, and setting up the systems' proprietary IDEs. For a macOS user navigating Oracle's setup, the distinctive element lies in its Docker-based installation, necessitating various specific terminal commands to access the Oracle database through terminal, while Windows user can just use a single "sqlplus" command to do so. However, MySQL introduces a more complex set of steps to access the database through the terminal due to the requirement to fetch specific file path information. This complexity is especially seen for macOS users as they must interact with Finder in specific ways to access the necessary directories to get that information. In contrast to Oracle and MySQL, Postgres's set of steps to access the database through the terminal is particularly simple for macOS users as it requires just one command. For Windows users on the other hand, Postgres's set of steps to access the database through the terminal is just as complex as MySQL's for both macOS and Windows users alike as it also requires users to follow similar steps to fetch a file path before being able to do so. Now in terms of users creating user authentication for themselves, while Oracle requires a specific session alteration command before allowing users to do so, both MySQL and Postgres allow users to do so without any commands of such.

#### 4.3 Proprietary IDEs

The Oracle, MySQL, and Postgres database systems each have a proprietary IDE that one can use to connect to the database servers. Each of these IDEs support a variety of features. These features range from allowing users to see exactly how their created tables to allowing users to import data from certain file types into the IDE and exporting data from the IDE to certain file types. In order to use the majority of the features available, the user must open a connection to their server. This will involve typing in a series of information from the installation and setup phases including, but not limited to, the host name, port number, username, and password. These are the bare minimum that will show up in all three IDE's that when entered correctly, will permit the tool to interact with the specified server. Refer to Appendices B (Using Oracle on macOS through Docker), D (Using Oracle on Windows), F (Using Postgres on macOS), H (Using Postgres on Windows), J (Using MySQL on MacOS), and L (Using MySQL on Windows) for more information regarding adding a connection within the IDEs.

##### 4.3.1 Oracle

Oracle's default IDE is called "SQLDeveloper". The first feature that a user has access to once they successfully connect to the Oracle database server using SQLDeveloper is a pre-made SQL worksheet. The provided worksheet is what allows users to write their SQL code. However, if users wish to create additional SQL files, all they have to do is click on the "File" tab displayed at the top of the SQLDeveloper, then click on the "New" button to officially be able to create an additional SQL file.



On the left side of the SQL worksheet, the users will see a list of all the connections that they made to the database server. Simply by double-clicking on one of those connections, the user will get access to various things such as all the tables they made using the deployed connection, along with other things such as views, triggers, procedures, and functions. If a user double-clicks any of these options, they will get access to additional features inside their IDEs. An example of such features is that by double-clicking on any of the displayed tables, the user will get access to a menu that allows them to see things like how their tables are structured, the data stored inside of them, and an ER diagram showing exactly how they are related to other tables made by the user.

In addition to all those features, SQLDeveloper gives the users the ability to directly import and export data. In order to import data, the first thing a user must do is right click on the “Tables” label in their connection menu, then click on the “Import Data...” label. Once that is done, the user can select a file by browsing on their computer. Example types of files that can be used to easily import data is Microsoft Excel, .HTML, .JSON, and .CSV. If all the columns are properly formatted, then when the user selects the file, all the columns and the data inside will automatically be formatted correctly, and the user just has to give its new table a name. Once that is done, the user then can select exactly the columns that they wish to include in their table. To complete creating their new table from the imported data, all the user has to do is configure the datatype for each of the columns that they wish to include in their new table. In SQLDeveloper, in addition to being able insert data to an entirely new table, it is also possible to simply insert additional rows to an already existing table. This can be done in the exact same way as inserting data for an entirely new table that a user is creating with the exact same file types. The only differences are that the user will not be able to name the table for which they are importing the data to, and that they will not be able to decide the datatypes for the columns of the table that they wish to import the data to.

In order to successfully export data from SQLDeveloper, users must first query the database so that they can successfully fetch all the data that they wish to export. Once that is done, the user will be prompted to give a table name for the data that they wish to export and select the type of file that they wish to have the data exported to. Examples of the file types available to do this with is Microsoft Excel, .HTML, .JSON and .CSV. To finish exporting the data, all users have left to do after finishing these steps is name the file and decide where on their computer do they want the file saved.

#### 4.3.2 MySQL

MySQL offers MySQL Workbench to access MySQL servers in a graphical user interface. From the home page of this application, the user can add, manage, and remove connections to specific MySQL servers hosted locally or in the cloud. Once a user has created a connection with the appropriate host address, username, and password that correlate with the SQL server, they will be presented with a MySQL Model tab. From here they can interact with database schemas in a variety of ways. They can use advanced database manipulation features such as adding Enhanced



Entity-Relationship (EER) diagrams, adding tables, views, and other items directly, or modifying schema privileges. On top of this, they can open query tab in addition to the one already provided to them when connected to be able to directly interact to SQL.

MySQL Workbench allows users to import and export large entire databases using the Data Import and Data Export tools. These can be found in the 'Server' menu tab. From the Data Export tab, they can select a specific schema or multiple schemas as well as a folder location to dump all the data into. This can later be used to recreate a full database in the Data Import tab.

Like Oracle SQLDeveloper's 'Import Data...' feature, MySQL Workbench supports importing and exporting .CSV and .JSON files. This can help by importing a massive amount of data into a single table as opposed to into the entire database like the Data Import and Data Export tabs. To accomplish this, right click on a table in the active schema, and select Table Data Import Wizard or Table Data Export Wizard.

#### 4.3.3 Postgres

Postgres offers pgAdmin 4 to manage and view tables. The dashboard provides graphs to show the user the active database sessions, transactions per second, tuples in, tuples out, and block I/O. This provides the user an overview of the current usage of the database as time, giving them a very precise idea of its functionality. pgAdmin places significant emphasis on this, with many tabs showing dependencies and intricacies of the database, which could be daunting for newer users.

The user is also able to use a query tool to execute select lines of SQL and can open preexisting SQL files into it. Something important to note is pgAdmin 4 does not show the data in pgAdmin 4 when the user parses through the tables, instead the user must select a table and then select the View Data button in the top left, which is less intuitive than other platforms. Once data is selected in this manner, the user can select a data filter option, and filter the data based on their commands.

pgAdmin 4 supports a PSQL tool, opening a command line style console, which allows the user to input their commands as they would in a terminal. It also supports an entity relational diagram tool, allowing users to create and format ERDs from within the application.

Lastly, pgAdmin 4 supports the importing and exporting of data through text, binary, and CSV files through the tool menu. This allows the user to move significant amounts of data into file format from databases, or vice versa, while also having the option to wipe the preexisting data when importing from files.

#### 4.3.4 Analysis and Discussion

Based on the overview of the three systems' proprietary IDEs, while they all support similar features, there are differences in terms of the steps required to access those features, and the limitations of some of them. All three of these database systems, will require valid connection

information before interacting with any SQL server. Oracle's SQLDeveloper and MySQL's MySQL Workbench are quite similar in how they are set up. Both focus on making it easy to work with SQL and help users design and manage databases smoothly. In Oracle's SQLDeveloper, as soon as users connect, they get direct access to an SQL worksheet to start interacting with SQL right away. If users want new SQL files, there's a clear "File" button at the top of the IDE that allows them to create those new files. On the left side of the IDE, users can see a list of their connections, and if they click on these, they will find things like their created tables, views, and more. Another feature worth mentioning is that users can click on a table and see a picture (an ER diagram) that shows how it connects with other tables. Furthermore, SQLDeveloper provides direct import and export data capabilities. Users can easily import data from formats like Microsoft Excel, .HTML, .JSON, and .CSV by right-clicking on the "Tables" label and then following the "Import Data..." prompt. On the flip side, exporting data requires users to first query the database, after which they can choose from the provided export file formats available such as Microsoft Excel, .HTML, .JSON, and .CSV, name the file, and save it. MySQL's MySQL Workbench is quite user-friendly as well. Using this IDE, users can swiftly set up, manage, or disconnect from MySQL servers from both their local machine and the cloud. A feature that MySQLWorkbench shares with SQLDeveloper is how the user gets access to a graphical user interface that allows them to interact with SQL as soon as they connect. In addition to this, if users want a visual representation of their database's layout, much like Oracle's ER diagrams, MySQLWorkbench also offers advanced features such as Enhanced Entity-Relationship (EER) diagrams. In terms of importing and exporting data, MySQL Workbench has its own functionality that helps with the importing and exporting of .CSV and .JSON files. Although this MySQLWorkbench feature is not supported for as many file types and formats as SQLDeveloper, it still allows users to work with commonly used file types and formats. By right-clicking on a table in the active schema, users can utilize the Table Data Import Wizard or Table Data Export Wizard for these tasks. While SQLDeveloper and MySQL Workbench are both suitable IDEs for students in an introductory database course, Postgres's pgAdmin 4 is not as suitable for such users. To begin, just launching it reveals a dashboard filled with real-time metrics such as active sessions, transactions per second, and granular data movement details like "tuples in" and "tuples out". While these kinds of features might be very useful for professional database users, they can be quite overwhelming and ambiguous for students in an introductory database course. Despite this, the way to view data inside tables in pgAdmin 4 is similar to the way to do so in SQLDeveloper. All users must do to view data inside tables in pgAdmin 4 is first select a table and then finding and clicking the "View Data" button. In addition to this simplicity, all users need to do to access the pgAdmin 4's data import/export functionality is look for the option in the IDE's tools menu.

#### 4.4 Cloud Pricing

In addition to having proprietary IDEs that one can use to connect to the database server, Oracle, MySQL, and Postgres also offer online cloud services that allow users to interact with the database server online. These services can range from automated data backup to giving users

additional databases with powerful tools for data analytics. For all three systems, the pricing of these cloud services depends on the kinds of services available that a user wants to use, and the amount of time that they wish to use the services.

#### 4.4.1 Oracle

Oracle's cloud service is called "Oracle Cloud Infrastructure". While the pricing can vary based on what users use this service for, it does come with a free tier. This free tier provides users various free unlimited services such as "two Oracle Autonomous Databases with powerful tools like Oracle APEX and Oracle SQL Developer", and "up to 4 instances of ARM Ampere A1 Compute with 3,000 OCPU hours and 18,000 GB hours per month" (Oracle, n.d.-a). In addition to the unlimited services, the free tier also comes with a \$300 USD cloud credit that grants users "access to a wide range of Oracle Cloud services for 30 days, including Databases and Analytics", and "up to 5 TB of storage" (Oracle, n.d.-a).

#### 4.4.2 MySQL

MySQL also comes with its own cloud infrastructure called "MySQL HeatWave". Some of the features of this infrastructure include a database that can be used for things like real-time analytics, very fast performance, and the fact that it can be used through other cloud services such as Amazon Web Services (AWS), and Oracle Cloud Infrastructure (OCI). While this cloud infrastructure does not come with an official free tier, users do have the ability to try it out for free if they use it through either AWS or OCI (MySQL, 2019).

#### 4.4.3 Postgres

While Postgres does not have an official Cloud Service like Oracle and MySQL, there are ways that users can use it online without connecting to the server through an IDE. One of such ways is a service called "Amazon RDS for Postgres". Through this service, users have access to fast storage, automated backup and recovery, easy database deployments, and increased security. While this service does not come with an official free tier, users signed up to AWS can start using this service for free (Amazon Web Services, 2019). Another way users can connect to the Postgres database server online is through "Azure Database for Postgres". This service comes with its own benefits such as a flexible server for simplified user experience, and AI-powered intelligent performance optimization and query store to build and scale users' databases. In addition to these benefits, the service comes with extensions for Azure Data Studio and Visual Studio Code, and can be used in languages such as Python, Ruby, Node.js, and Java. While this service does not come with an official free tier, users with Azure account can start using the service for free (Microsoft, n.d.).

#### 4.4.4 Analysis and Discussion

Based on the three systems' cloud offerings detailed above, it's evident that the main differences lie in their pricing structures and accessibility options. To begin, unlike the MySQL and Postgres, Oracle has its own official "Oracle Cloud Infrastructure", free tier accompanied by a \$300

USD cloud credit available for an initial period of 30 days granting a wide range of functionalities from databases to analytics readily available to users. On the other hand, while users can begin using MySQL's "MySQL HeatWave" for free when accessing via partner platforms such as Amazon Web Services and Oracle Cloud Infrastructure, it does not come with an official free tier like Oracle Cloud Infrastructure does. Postgres, though sharing similarities with MySQL's situation of lacking a direct free offering, introduces a slight difference. While it also uses other major platforms such as "Amazon RDS for Postgres" and "Azure Database for Postgres" to provide users cloud services initially for free, it does not have its own direct cloud service. It should also be noted that if a user wants to use these systems' cloud services through other platforms, they must have accounts created for those respective platforms first.

## 4.5 Datatypes

Datatypes are a fundamental aspect of any programming language, especially when interacting with databases. Different forms of content within software applications can be broken down into fundamental data types. An article can be broken down into sections, paragraphs, sentences, words, and eventually characters. A map guiding them from address A to address B can be broken down into a list of coordinates to follow. A date of time can be broken down into years, months, days, hours, minutes, seconds, and more. Characters, coordinates, date/time, and numbers are just a few of the datatypes that most databases will support.

### 4.5.1 Numerical Datatypes

MySQL, PostgreSQL, and OracleSQL all implement the same principles about storing different types of numbers. They support both exact and inexact formats. If users know precisely what size of number they would like to store, they can use an exact type. However, if they do not know this, they would use an approximate format such as a floating-point number.

In OracleSQL, to describe an integer, users will say NUMBER(P) where P(precision) ranges from 1 to 38 and represents the total number of digits that they can store in the datatype. So, NUMBER(3) can represent any number from -999 to 999. This differs from MySQL and PostgreSQL in that these two languages both support named integer sizes. Both implement SMALLINT, INTEGER/INT, BIGINT, and MySQL also supports TINYINT and MEDIUMINT. TINYINT represents 1 byte of storage, SMALLINT represents 2 bytes of storage, MEDIUMINT represents 3 bytes of storage, INT represents 4 bytes of stores, and BIGINT represents 8 bytes of storage.

Another example of an exact format is the fixed-point data type. This represents a number where users know exactly how many numbers there are before and after the decimal point. In OracleSQL, users represent this with NUMBER(P, S) where P(precision) ranges from 1 to 38 and S(scale) ranges from -84 to 127. The precision still represents how many digits to the left of the decimal point, but the scale now represents the total number of digits point. When S is negative, this will round the number at the value S to the left of the decimal. For example: NUMBER(3, 2)

ranges from -9.99 to 9.99. However, `NUMBER(3, -2)` ranges from -900 to 900 but only by multiples of 100 due to the rounding: -900, -800, ..., 800, 900. MySQL and PostgreSQL also implement this same formula swapping out the keyword `NUMBER` in favor of `DECIMAL` or `NUMERIC` for clarity.

However, as developers, users may not always know the exact precision or scale of a number. This is when they would use a floating-point datatype. OracleSQL implements this with three independent datatypes: `FLOAT`, `BINARY_FLOAT`, and `BINARY_DOUBLE`. The `FLOAT(P)` datatype is a subtype of the `NUMBER` datatype, specifically used for base-10 arithmetic. The `P` for precision in this type refers to the binary digits for the mantissa as opposed total number of decimal digits supported. `BINARY_FLOAT` is a datatype made to the IEEE 32-bit floating-point specification, and the `BINARY_DOUBLE` is a datatype made to the IEEE 64-bit floating-point specification. To specify floating-point literals in OracleSQL, users can use 'f' and 'd' attached to the back of any decimal value: '12.3f' or '0.0d'. This differs from PostgreSQL which uses the types `REAL` and `DOUBLE PRECISION` to represent floating-point numbers. `REAL` has a storage size of 4bytes and a minimum precision of 6 decimal digits, whereas the `DOUBLE PRECISION` has a storage size of 8 bytes and a minimum precision of 15 decimal digits. MySQL also has its own implementation of floating-point numbers. They use `FLOAT` and `DOUBLE`. The `FLOAT(p)` and `DOUBLE(p)` syntaxes can be used to specify an optional precision from 0 to 23 in a 4-byte single-precision `FLOAT` column, and 24-53 in an 8-byte double-precision `DOUBLE` column. In older versions of MySQL, there were non-standard representations of floats: `FLOAT(M, D)`, `REAL(M, D)`, and `DOUBLE PRECISION(M, D)`, where values can be stored with up to `M` total digits with `D` digits after the decimal point. However, these non-standard representations have since been deprecated.

#### 4.5.2 Date and Time Datatypes

Managing date and time values is crucial in database systems, as they play a significant role in various applications, from logging events to scheduling tasks. MySQL, PostgreSQL, and OracleSQL offer a range of datatypes to handle these values, each with its own nuances.

All three of these will support basic datatypes to represent an instant in time. In OracleSQL, the `DATE` datatype represents a date and a time by storing the year, month, day, hour, minute, and second. This differs from MySQL's and PostgreSQL's `DATE` datatypes which only represents a date by storing the year, month, and day without time. Both MySQL and PostgreSQL also implement a `TIME` datatype representing the hour, minute, and second of a time. Like OracleSQL's `DATE` type, MySQL and PostgreSQL implement `DATETIME` and `TIMESTAMP` respectively to represent a date and a time. One note to make is that OracleSQL's `TIMESTAMP` datatype represents a `DATE` but with the added precision of a fractional second with up to 9 decimal places.

To save an interval of time as opposed to an instance in OracleSQL, users can use the datatypes `INTERVAL YEAR TO MONTH` or `INTERVAL DAY TO SECOND` which will

represent an interval using years and months or using days, hours, minutes, and seconds respectively. PostgreSQL has a bit more customization when it comes to storing an interval. Specifically, it follows this format: quantity unit [quantity unit...] [direction]. Quantity represents a possibly signed number and unit can be swapped out for ‘microsecond’, ‘millisecond’, ‘second’, ‘minute’, ‘hour’, ‘day’, ‘week’, ‘month’, ‘year’, ‘decade’, and ‘century’. The angle brackets represent an optional input and ... represents one or more of the contents of the brackets. This means [quantity unit...] that users can combine multiple units each with their own quantity to the interval. Lastly, [direction] means that users can optionally add the keyword ‘ago’ to add directionality. Although there is no formal interval datatype in MySQL, users can still use the TIME datatype in a similar way to interval in other systems. MySQL does support the YEAR datatype to represent a 4-digit year as well.

#### 4.5.3 String Datatypes

String datatypes are essential for storing text-based data, such as names, addresses, and descriptions, in databases. MySQL, PostgreSQL, and OracleSQL provide various string datatypes to accommodate different needs and use cases. Similar to how numeric datatypes were split into exact and approximate formats, strings have two main types: fixed-length and variable-length.

All three database systems implement the same base type of CHAR(n) to store exactly n characters in a fixed-length datatype. Often, the database designer does not know exactly how many characters will be saved in any given string. When this is the case the datatype VARCHAR(n) is very useful. When using VARCHAR(n), the string can store up to but not exceeding n characters. Note: in OracleSQL the VARCHAR type has been deprecated in favor of the VARCHAR2(n) datatype which behaves just like VARCHAR(n) in the other two systems.

Sometimes it is necessary to store variable-length strings without an upper limit. MySQL takes advantage of a BLOB type which stands for **binary large object** to store text. There are four sizes of blobs: TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB differing only in the maximum length of values that they can hold. Although these are meant specifically for binary strings, there are four datatypes built on these called TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT that are specifically designed for storing large variable-length strings. PostgreSQL also has a binary large object however it is named BYTEA, but if users want to explicitly store a large variable-length string, they can use the TEXT datatype. OracleSQL also had its own implementations for these types: BLOB and CLOB (**character large object**). Additionally, both PostgreSQL and OracleSQL are able to declare a VARCHAR or VARCHAR2 with no specified length, similar to TEXT, but MySQL does not have access to that functionality. This functionality allows maximum length Strings without using the TEXT field.

String datatypes are pivotal for representing textual information in databases. Understanding the characteristics and proper usage of different string datatypes in MySQL, PostgreSQL, and OracleSQL is crucial for effective database design and data management. By



choosing the appropriate string datatype and applying best practices, developers can optimize storage, enhance performance, and ensure the integrity of text data in database systems.

#### 4.5.4 Other Notable Datatypes

Beyond the standard numerical, date/time, and string datatypes; MySQL, PostgreSQL, and OracleSQL offer a variety of other notable datatypes, each serving unique and specialized purposes. In MySQL, the ENUM datatype is significant, allowing representation of a string object with a value chosen from a list of permitted values defined during table creation, ideal for representing a fixed set of related items. The SET datatype in MySQL is akin to ENUM but permits the storage of one or more values from a predefined list, making it suitable for representing collections of attributes. The BIT datatype in MySQL is crucial for storing bit values, providing an efficient means to store binary data such as flags or states in a compact form. Additionally, MySQL's JSON datatype is pivotal for storing JSON-formatted strings, enabling the manipulation of JSON documents directly in SQL queries, ideal for handling semi-structured data.

In PostgreSQL, the ARRAY datatype is noteworthy, capable of storing ordered sets of elements, facilitating the storage and processing of sequences of values. The HSTORE datatype is a key-value store within a single PostgreSQL value, apt for storing simple object-relational mappings. The UUID datatype in PostgreSQL is essential for storing universally unique identifiers, optimizing the assignment of unique identifiers in a distributed database environment. PostgreSQL also supports JSON and JSONB datatypes for storing JSON data, with JSONB allowing indexing and offering efficiency in searching and processing, while JSON preserves the input format.

OracleSQL introduces unique datatypes like RAW and LONG RAW for storing binary data, suitable for handling small amounts of binary data such as hashes or checksums. The ROWID and UROWID datatypes in OracleSQL represent the unique address of a row in its table, optimizing performance in row-specific operations. The XMLType datatype in OracleSQL is significant for storing XML data, allowing the manipulation of XML documents directly in SQL queries, ideal for handling structured XML data. Lastly, OracleSQL's ANYDATA and ANYTYPE datatypes can store instances of any datatype, including object types and collections, providing flexibility in building generalized routines and data structures.

Understanding and leveraging these specialized datatypes allow developers to design more flexible and efficient database schemas, enabling the representation and manipulation of a diverse range of data structures and formats, thereby enhancing the versatility and capability of database systems.

## 4.6 Table Management

### 4.6.1 Schema Used



Figure 3: Schema used throughout the Table Management section. Note: *Authors.email* is a unique key. *Books.author\_id* is a foreign key with *Authors.author\_id*. *Reviews.book\_id* is a foreign key with *Books.book\_id*.

For this part of the project, we analyzed the syntactic differences and the complexity involved in creating tables and defining constraints across the Oracle, MySQL, and PostgreSQL database systems. Through our analysis, we observed that all three systems employ the CREATE TABLE command, immediately followed by the table name. This is then followed up by a list of column definitions with their respective data types, enclosed within parentheses. However, it's worth noting that the specific data types available and their nuances might vary across these systems. In addition to this, constraints—whether they are primary keys, unique keys, or foreign keys—are delineated within the same set of parentheses. Once all that is done, the user needs to insert a “;” to complete the command to create the table. One of the main differences regarding the aspect of Table management across all three systems is the fact that MySQL and PostgreSQL demonstrate greater flexibility compared to OracleSQL. This is because they offer a broader range of commands and allow for constraints that are not available in OracleSQL. In addition to this, when it comes to dropping tables, OracleSQL lacks the "IF EXISTS" option that is available in both MySQL and PostgreSQL. This option prevents an error if the table targeted to be dropped does not exist.

### 4.6.2 Oracle

When creating a table in OracleSQL, if a user wants to add a constraint to restrict the values allowed for a specific column, they only have one command that they can use to do so:

```

1 CREATE TABLE Authors (
2     author_id  NUMBER,
3     name       VARCHAR2(100),
4     email      VARCHAR2(100),
5     country    VARCHAR2(50),
6     CONSTRAINT Authors_PK      PRIMARY KEY (author_id),
7     CONSTRAINT Authors_email_UQ UNIQUE      (email),
8     CONSTRAINT Authors_country_CHK CHECK     (country IN ('USA', 'UK', 'Canada', 'Australia'))
9 );

```

Figure 4: SQL Code to create an Authors table in OracleSQL.



The only way to restrict the values allowed for a specific table column in OracleSQL is by using the CHECK command. For instance, looking back at our defined schema, in our Authors table, we have used the CHECK command on the country column to ensure that only authors from 'USA', 'UK', 'Canada', or 'Australia' can be added. This is achieved using the line CONSTRAINT Authors\_country\_CHK CHECK (country IN ('USA', 'UK', 'Canada', 'Australia')). In addition to this restriction, OracleSQL presents another challenge. When a user wants to automatically update foreign key values in one table due to changes in primary key values in another table, there's no direct mechanism. For instance, consider our “Books” table from our defined schema. In it, the “author\_id” column is a foreign key that references the “author\_id” primary key in the "Authors" table. If an “author\_id” value in the "Authors" table were to be updated, OracleSQL does not natively provide a mechanism to automatically reflect these changes in the "Books" table where this “author\_id” is used as a foreign key. The same is true for setting them to null when the primary key is updated, as constructs such as “ON UPDATE CASCADE” or “ON UPDATE SET NULL” are not natively supported in OracleSQL. As a result, if users wish to implement such functionality in OracleSQL, they must find other ways to do so, like employing triggers.

```
DROP TABLE Reviews;
```

*Figure 5: SQL Code to drop the table named Reviews in OracleSQL.*

Something else that is different in OracleSQL than in MySQL and PostgreSQL is the restricted syntax used to drop existing tables. The way to drop a table in OracleSQL is to use the “drop table” command with the name of the table the user wishes to drop. For instance, using our previously defined schema, if one wishes to drop the “Reviews” table, the command would be: “drop table Reviews;”. However, unlike MySQL and PostgreSQL, OracleSQL does not have the safeguard of the "IF EXISTS" option, so attempting to drop a non-existent table will result in an error.

#### 4.6.3 MySQL and Postgres

When creating a table in either MySQL or PostgreSQL, if a user wants to add a constraint to restrict the values allowed for a specific column, they have more flexibility to do so than in OracleSQL as they have more commands that they can use:

```

1 CREATE TABLE Authors (
2     author_id INT,
3     name      VARCHAR(100),
4     email     VARCHAR(100),
5     country   ENUM('USA', 'UK', 'Canada', 'Australia'),
6     CONSTRAINT Authors_PK          PRIMARY KEY (author_id),
7     CONSTRAINT Authors_email_UQ    UNIQUE      (email)
8 );
9
10 CREATE TABLE Authors (
11     author_id INT,
12     name      VARCHAR(100),
13     email     VARCHAR(100),
14     country   VARCHAR(50),
15     CONSTRAINT Authors_PK          PRIMARY KEY (author_id),
16     CONSTRAINT Authors_email_UQ    UNIQUE      (email),
17     CONSTRAINT Authors_country_CHK CHECK      (country IN ('USA', 'UK', 'Canada', 'Australia'))
18 );

```

Figure 6: SQL Code demonstrating how in MySQL, users can use either the ENUM syntax or the CHECK syntax to limit specific values in a table.

```

1 CREATE TYPE country_enum AS ENUM ('USA', 'UK', 'Canada', 'Australia');
2
3 CREATE TABLE Authors (
4     author_id INT,
5     name      VARCHAR(100),
6     email     VARCHAR(100),
7     country   country_enum,
8     CONSTRAINT Authors_PK          PRIMARY KEY (author_id),
9     CONSTRAINT Authors_email_UQ    UNIQUE      (email)
10 );
11
12 CREATE TABLE Authors (
13     author_id INT,
14     name      VARCHAR(100),
15     email     VARCHAR(100),
16     country   VARCHAR(50),
17     CONSTRAINT Authors_PK          PRIMARY KEY (author_id),
18     CONSTRAINT Authors_email_UQ    UNIQUE      (email),
19     CONSTRAINT Authors_country_CHK CHECK      (country IN ('USA', 'UK', 'Canada', 'Australia'))
20 );

```

Figure 7: SQL Code demonstrating how in PostgreSQL, users can either use a custom defined ENUM type or the CHECK syntax to limit specific values in a table.

In MySQL and PostgreSQL, users have the flexibility to restrict the values of a specific table column, though the implementation of the “ENUM” type differs between the two. In MySQL, the “ENUM” data type can be directly incorporated into table definitions. For instance, looking back at our defined schema, the syntax for the “country” column in the “Authors” table for MySQL would resemble: “country ENUM('USA', 'UK', 'Canada', 'Australia)”. On the other hand, PostgreSQL requires users to first define an “ENUM” type and then apply it as a column's data type. So, for PostgreSQL, one would first define the type using “CREATE TYPE country\_enum AS

ENUM ('USA', 'UK', 'Canada', 'Australia')” and subsequently, in the “Authors” table definition, specify the column as “country country\_enum”. Despite these differences, similar to OracleSQL, both systems also offer the “CHECK” command as an alternative. Using the “check” command, the value restriction for the “country” column in both systems would look like: “CHECK(country IN ('USA', 'UK', 'Canada', 'Australia'))”

In addition to this flexibility, both MySQL and PostgreSQL offer comprehensive solutions when compared to OracleSQL, particularly regarding the challenge of updating foreign key values in one table when primary key values in another table change:

```

1 CREATE TABLE Books (
2     book_id      INT,
3     title        VARCHAR(200),
4     author_id    INT,
5     CONSTRAINT Books_PK          PRIMARY KEY (book_id),
6     CONSTRAINT Books_FK_Author  FOREIGN KEY (author_id) REFERENCES Authors(author_id)
7     ON UPDATE CASCADE
8 );

```

Figure 8: Valid Table Creation in MySQL and PostgreSQL. Uses ON UPDATE CASCADE.

```

1 CREATE TABLE Reviews (
2     review_id     INT,
3     review_content VARCHAR(500),
4     book_id       INT,
5     CONSTRAINT Reviews_PK          PRIMARY KEY (review_id),
6     CONSTRAINT Reviews_FK_Book    FOREIGN KEY (book_id) REFERENCES Books(book_id)
7     ON UPDATE SET NULL
8 );

```

Figure 9: Valid Table Creation in MySQL and PostgreSQL. Uses ON UPDATE SET NULL.

Both systems provide the “ON UPDATE CASCADE” command, which automatically updates matching foreign key values when the referenced primary key value changes, and the “ON UPDATE SET NULL” command, which sets the foreign key to NULL if its corresponding primary key value is modified. For instance, looking back at our defined schema, in the “Books” table, the “author\_id” column serves as a foreign key referencing the “author\_id” primary key in the “Authors” table. If an “author\_id” in the “Authors” table is updated, these mechanisms ensure changes are reflected in the “Books” table without manual intervention. Similarly, in the “Reviews” table, the “book\_id” column references the “book\_id” primary key in the “Books” table. Should a referenced “book\_id” be deleted or modified, the “ON UPDATE SET NULL” option ensures that the “book\_id” in the “Reviews” table is automatically set to NULL.

Something else that is different in MySQL and PostgreSQL than in OracleSQL is the more flexible syntax used to drop existing tables:

## DROP TABLE IF EXISTS Reviews;

Figure 10: The 'IF EXISTS' option that can be used in MySQL and PostgreSQL to drop the 'Reviews' table if it already exists.

Similar to OracleSQL, the way to drop a table in MySQL and PostgreSQL is to use the “drop table” command with the name of the table the user wishes to drop. For instance, using our previously defined schema, if one wishes to drop the “Reviews” table, the command would be: “drop table Reviews;”. However, what these two systems have that OracleSQL does not is the safeguard of the "IF EXISTS" option, which ensures that users will not encounter an error when trying to drop a non-existent table.

### 4.6.4 Analysis and Discussion

Despite the overwhelming similarity in the table creation statements between the three languages, there are a few notable changes. As seen in the differences described above, when comparing the aspect of Table Management across the OracleSQL, PostgreSQL, and MySQL database systems, the main differences lie in their flexibility regarding the constraints that they allow, and the way tables are dropped. To begin, while OracleSQL only has the CHECK command to restrict the values allowed for a specific table column, both MySQL and PostgreSQL allow users to choose between using the “check” command and using the ENUM datatype to do so. Another example where this applies is while OracleSQL does not have any direct mechanism when a user wants to automatically update foreign key values in one table due to changes in primary key values in another table, both MySQL and PostgreSQL do. Both systems come with the “ON UPDATE CASCADE” command, which automatically updates matching foreign key values when the referenced primary key value changes, and the “ON UPDATE SET NULL” command, which sets the foreign key to NULL if its corresponding primary key value is modified. In terms of the way to drop tables across all three systems, the difference in flexibility can be seen there too. While all three systems use the “DROP TABLE” command to drop tables, both MySQL and PostgreSQL have an added layer of protection that OracleSQL does not. That layer of protection is the safeguard of the "IF EXISTS" option, which ensures that users will not encounter an error when trying to drop a non-existent table.

### 4.7 Data Manipulation

Insert, Update, and Delete statements are the primary tools for data manipulation in all SQL languages. These statements are so fundamental that they remain the exact same between the three languages. Even when using some of the more advanced techniques covered in introductory database courses, such as a nested subquery in a delete or update statement, the syntax stays consistent between the three languages.

## 4.8 Query Simplicity

Queries throughout the three languages are overwhelmingly similar in structure. The three languages utilize the same primary keywords when constructing queries, such as SELECT, FROM, WHERE, and GROUP BY. An example of code that is compatible with all three languages is as follows:

```

1  SELECT firstName, lastName, SUM(price) AS MaxDonatedValue
2  FROM Artwork AR
3  JOIN Artist A ON AR.artistID = A.artistID
4  GROUP BY firstName, lastName
5  HAVING SUM (price) = (
6      SELECT MAX (Sum) AS MaxSum
7      FROM (
8          SELECT SUM(price) AS Sum
9          FROM Artwork AR
10         GROUP BY artistID
11        ) AS sub
12 );

```

Figure 11: Example query that is valid in OracleSQL, PostgreSQL, and MySQL.

All queries in each of the languages follow the same structure of SELECT, FROM, GROUP BY, HAVING, and ORDER BY.

This is one of the more advanced queries that would be seen in an introductory class, featuring aggregation, grouping, and having statements, along with multiple nested subqueries. This is just one example of how many queries are formatted the same way between languages.

There were only some noticeable differences with certain keywords, such as MINUS and EXCEPT. PostgreSQL does not support the MINUS keyword, and instead uses EXCEPT, while the other two languages use MINUS. Additionally, PostgreSQL requires aliasing for subqueries, not allowing their usage if they are not properly aliased.

There are more specialized elements of queries that are only available in certain languages, such as the IF function. Only MySQL supports IF usage in queries, while PostgreSQL and OracleSQL are limited to CASE statements.

## 4.9 Advanced Functionality



It is worth mentioning that procedures and functions are part of proprietary functional programming languages provided by the database system to supplement the existing SQL, which is a declarative language.

### 4.9.1 Views

Views functioned in the exact same ways throughout the three languages. The only difference is that MySQL does not support materialized views, which is a static storage of data based on the schema at the time the view was created. Materialized views are not likely to be covered in an introductory course, so this difference is largely irrelevant for the purpose of this study.

### 4.9.2 Procedures

#### 4.9.2.1 Schema Used

Employee		Location	
employeeID 	number	locationID 	char
username	varchar	xcoord	number
password	varchar	ycoord	number
firstName	varchar	floor	char
lastName	varchar	longName	varchar
salaryGrade	number	shortName	varchar
securityClearance	char	locationTypeID	char
NPI	number		
locationID	char		




Figure 12: Schema used throughout the procedures section. Note: *Employee.locationID* is a foreign key with *Location.locationID*.

The purpose of the procedure was to display an employee's name, location's short name, employee ID, and username based on the inputted first and last names.

#### 4.9.2.2 Oracle

```

1  SET SERVEROUTPUT ON;
2  CREATE OR REPLACE PROCEDURE EmployeeLocation (inFirstName VARCHAR2, inLastName VARCHAR2) AS
3  CURSOR csr IS
4      SELECT * FROM Employee e
5      JOIN Location l ON (e.locationID = l.locationID);
6  BEGIN
7      FOR line IN csr LOOP
8          IF (line.firstName = inFirstName AND line.lastName = inLastName) THEN
9              DBMS_OUTPUT.PUT_LINE(
10                 line.firstName || ' ' || line.lastName
11                 || ' located in ' || line.shortName
12                 || '. EmployeeID: ' || line.employeeId
13                 || '. Username: ' || line.username || '.');
14          END IF;
15      END LOOP;
16  END;
17  /
18
19  EXECUTE EmployeeLocation( inFirstName: "FirstName", inLastName: "LastName");

```

Figure 13: A procedure defined in OracleSQL used to print out information about an employee's location based on their name.



The first important procedure distinction of OracleSQL is the fact that running the line “SET SERVEROUTPUT ON;” is required to be able to display any messages on the console. To complement this distinction, OracleSQL uses its own DBMS\_OUTPUT.PUT\_LINE() function to display messages on the console. To use this function, the line “SET SERVEROUTPUT ON;” must have run first. Following those distinctions, procedures in OracleSQL must end with a “/” at the end to properly compile. To run the procedure in OracleSQL, place the keyword EXECUTE followed by the proper procedure name and the proper inputs.

#### 4.9.2.3 Postgres

```

1 CREATE OR REPLACE PROCEDURE EmployeeLocation (inFirstName IN VARCHAR, inLastName IN VARCHAR)
2 LANGUAGE plpgsql
3 AS $delimiter_name$
4 DECLARE E1 CURSOR FOR
5     SELECT employeeID, username, shortName
6     FROM Employee E
7     JOIN Location L on E.locationID = L.locationID
8     WHERE firstName = inFirstName
9     AND lastName = inLastName;
10 BEGIN
11     FOR  erec in E1 LOOP
12         RAISE NOTICE '% % located in %. EmployeeID: %. Username: %.',
13             inFirstName, inLastName, erec.shortName, erec.employeeID, erec.username;
14     END LOOP;
15 END;
16 $delimiter_name$;
17
18 CALL EmployeeLocation( inFirstName: 'FirstName', inLastName: 'LastName');

```

Figure 14: A procedure defined in PostgreSQL used to print out information about an employee's location based on their name.

PostgreSQL has a few very specific syntactical requirements. The language must be prefaced as plpgsql either before or after the body of the procedure. The body of the procedure must be signaled with a delimiter of the format \$\$, with text between the dollar signs if two or more different delimiters are used. Outputs are raised from procedures with the RAISE NOTICE command, and data is entered into the provided notice string using the percent character. Procedures are called with the CALL command in the format seen above.

#### 4.9.2.4 MySQL

One major difference from this procedure definition in MySQL specifically is the LOOP keyword. In PostgreSQL and OracleSQL, we took advantage of a for loop to iterate over the cursor. In MySQL, however, we need to build out a for loop using the syntax: “custom\_loop\_label: LOOP { insert SQL code to be repeated } END LOOP;”. Since there is no underlying implementation to automatically after the last cursor result, like the for loop has, users must also include “LEAVE custom\_loop\_label;” inside of the loop or else it will loop forever causes errors. In this example, we declared a variable named ‘done’ that will be set to 0 when there is no result left to iterate over. Another key difference regarding procedures in MySQL is how unlike OracleSQL and PostgreSQL, MySQL does not support the “OR REPLACE” clause.

```

1  DELIMITER //
2  CREATE PROCEDURE EmployeeLocation (inFirstName VARCHAR(255), inLastName VARCHAR(255))
3  BEGIN
4      DECLARE done INT DEFAULT 0;
5      DECLARE v_employeeID INT;
6      DECLARE v_username VARCHAR (255) ;
7      DECLARE v_shortName VARCHAR (255) ;
8      DECLARE crsr CURSOR FOR
9          SELECT employeeID, username, shortName
10         FROM Employee E
11         JOIN Location L ON E.locationID = L.locationID
12         WHERE firstName = inFirstName
13         AND lastName = inLastName;
14         DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
15         OPEN crsr;
16         read_loop: LOOP
17             FETCH crsr INTO v_employeeID, v_username, v_shortName;
18             IF done THEN
19                 LEAVE read_loop;
20             END IF;
21         END LOOP;
22         CLOSE crsr;
23     END //
24     DELIMITER ;
25
26     CALL EmployeeLocation( inFirstName: 'FirstName', inLastName: 'LastName');

```

Figure 15: A procedure defined in MySQL used to print out information about an employee's location based on their name.

This means that, in MySQL, if users need to modify an existing procedure, they must manually drop it and then redefine it. Furthermore, also unlike the other two languages, MySQL does not have a straightforward mechanism like to directly display messages on the console, thus making it harder for users to display information using procedures. Despite these drawbacks, MySQL does support an “IF NOT EXIST” clause that users can put in the procedure declaration to be able to avoid encountering an error if they try to create a procedure that they have already created. As for the delimiters, similarly to PostgreSQL’s \$\$ syntax, MySQL uses ‘DELIMITER //’ at the beginning and ‘// DELIMITER’ at the end of the procedure. Just like PostgreSQL, procedures in MySQL are called using the CALL command.

#### 4.9.2.5 Analysis and Discussion

Overall, the procedures documented in each language display similarities and differences. The structure is essentially the same in all, with minor differences being with the delimiters, loop declaration, cursor declaration, MySQL's lack of support for the "OR REPLACE" clause, and procedure call. Every language features some kind of delimiter to section the body of the procedure, and PostgreSQL specifically requires a language declaration statement. The way that messages are printed to the console is notably different in all languages. While PostgreSQL and OracleSQL have methods more in line with other systems, MySQL does not have a direct way to display messages on the console. Additionally, OracleSQL is the only language that requires server messages to be manually enabled. However, what MySQL supports that the other two languages do not is an “IF NOT EXIST” clause that users can put in the procedure declaration to be able to avoid encountering an error if they try to create a procedure that they have already created.



### 4.9.3 Functions

#### 4.9.3.1 Schema Used

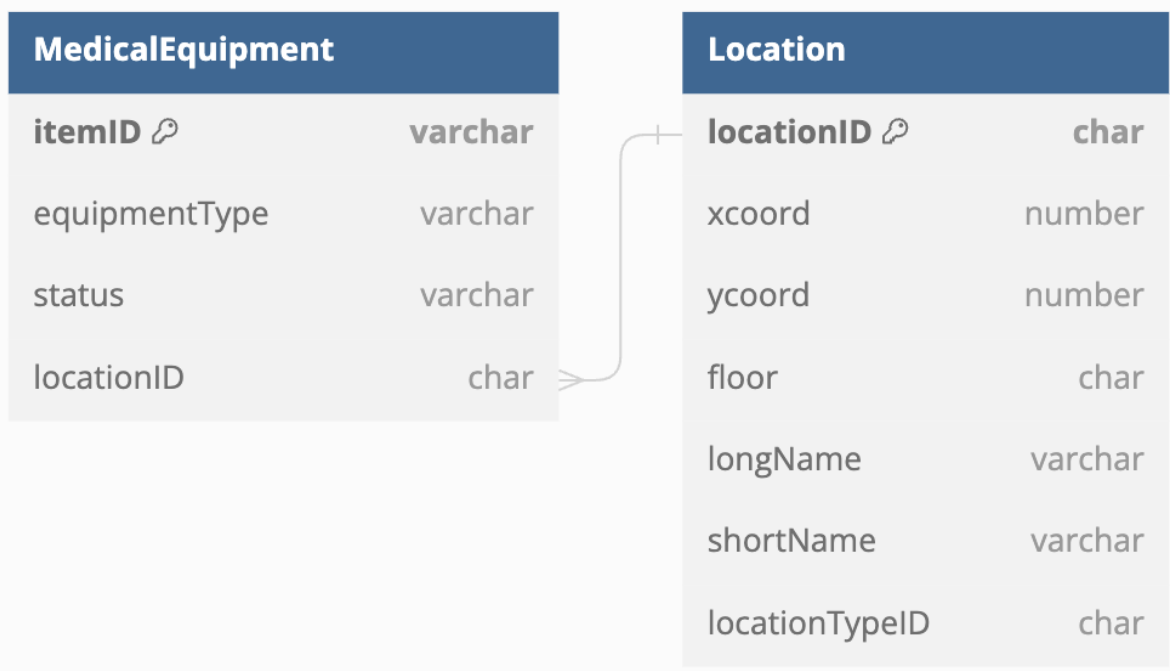


Figure 16: Schema used throughout the functions section. Note: MedicalEquipment.locationID is a foreign key with Location.locationID.

The purpose of this function was to determine and return the count of medical equipment items present in a given location.

#### 4.9.3.2 Oracle

```

1  CREATE OR REPLACE FUNCTION GetEquipmentCountInLocation(p_shortName VARCHAR2) RETURN NUMBER AS
2      v_count NUMBER;
3  BEGIN
4      SELECT COUNT(m.itemID)
5      INTO v_count
6      FROM MedicalEquipment m
7           JOIN Location l ON m.locationID = l.locationID
8      WHERE l.shortName = p_shortName;
9
10     RETURN v_count;
11 END GetEquipmentCountInLocation;
12 /
13
14 SELECT l.shortName, GetEquipmentCountInLocation( p_shortName: l.shortName) AS EquipmentCount
15 FROM Location l
16 ORDER BY GetEquipmentCountInLocation( p_shortName: l.shortName) DESC;

```

Figure 17: A function defined in OracleSQL to get the equipment count based on a location's short name.

Similar to procedures, an important function distinction of OracleSQL is the fact that they require a “/” at the end to properly compile.

#### 4.9.3.3 Postgres

```

1 CREATE OR REPLACE FUNCTION GetEquipmentCountInLocation(inShortName IN VARCHAR)
2 RETURNS NUMERIC AS $$
3 DECLARE
4     m_count NUMERIC;
5 BEGIN
6     SELECT COUNT(m.itemID)
7     INTO m_count
8     FROM MedicalEquipment m
9     JOIN Location l ON m.locationID = l.locationID
10    WHERE l.shortName = inShortName;
11
12    RETURN m_count;
13 END;
14 $$ LANGUAGE plpgsql;
15
16 SELECT l.shortName, GetEquipmentCountInLocation( inShortName: l.shortName) AS EquipmentCount
17 FROM Location l
18 ORDER BY GetEquipmentCountInLocation( inShortName: l.shortName) DESC;

```

Figure 18: A function defined in PostgreSQL to get the equipment count based on a location's short name.

PostgreSQL function syntax is very similar to its procedure syntax, also requiring the delimiter and language specification. However, PostgreSQL differs from OracleSQL in its requirement of a separate Declaration section, as opposed to simply declaring new variables after the AS statement. Other than these, there are not many differences.

#### 4.9.3.4 MySQL

```

1 DELIMITER //
2 CREATE FUNCTION GetEquipmentCountInLocation(inShortName VARCHAR(255))
3 RETURNS INT
4 DETERMINISTIC
5 BEGIN
6     DECLARE v_count INT;
7     SELECT COUNT(m.itemID)
8     INTO v_count
9     FROM MedicalEquipment m
10    JOIN Location l ON m.locationID = l.locationID
11    WHERE l.shortName = inShortName;
12    RETURN v_count;
13 END
14 // DELIMITER ;
15
16 SELECT l.shortName, GetEquipmentCountInLocation( inShortName: l.shortName) AS EquipmentCount
17 FROM Location l
18 ORDER BY GetEquipmentCountInLocation( inShortName: l.shortName) DESC;

```

Figure 19: A function defined in MySQL to get the equipment count based on a location's short name.

MySQL's function definition syntax is also fairly similar to its procedure definition syntax, except there are a few notable differences. Firstly, the delimiters are used equivalently in this definition as the procedure. The first difference is where we specify a return type; it requires a specific datatype before the BEGIN statement. Another key difference regarding functions in MySQL is that similar to procedures, unlike OracleSQL and PostgreSQL, MySQL does not support the "OR REPLACE" clause. This means that, in MySQL, if users need to modify an existing function, they have to manually drop it and then redefine it. Despite this drawback, MySQL does support an "IF NOT EXIST" clause that users can put in the function declaration to be able to avoid encountering an error if they try to create a function that they have already created. Additionally, before the BEGIN statement, users can optionally add characteristics. A characteristic can be one or more of these specific statements: COMMENT 'enter custom comment here', LANGUAGE SQL, DETERMINISTIC, NOT DETERMINISTIC, and a few other options. In this example, the DETERMINISTIC characteristic was added to let the program know that when running this function with the same input parameters it will always return the same result.

#### *4.9.3.5 Analysis and Discussion*

Like procedures, the structure of functions does not see too much change between the three languages other than some intricacies. All languages still use their respective delimiters, and all specify a return type in roughly the same format. One of the most notable differences is the variable declaration, with OracleSQL declaring before the BEGIN statement, PostgreSQL declaring in a separate declaration section, and MySQL declaring within the Begin statement. Moreover, similar to procedures, MySQL has a couple of distinctions from OracleSQL and PostgreSQL. Firstly, unlike OracleSQL and PostgreSQL, MySQL does not support the "OR REPLACE" clause for functions, thus requiring users need to manually drop a function and then redefine it if they wish to modify it. Secondly, MySQL supports an "IF NOT EXIST" clause that the other two languages do not that users can put in the function declaration to be able to avoid encountering an error if they try to create a function that they have already created. Apart from these, overall, the differences between the languages in this section are not very prominent.

## 4.9.4 Triggers

### 4.9.4.1 Schema Used

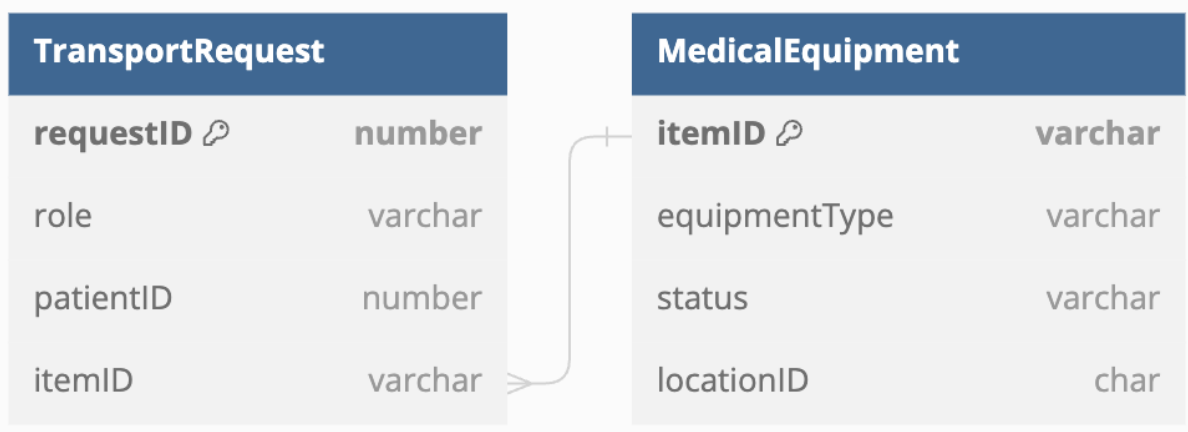


Figure 20: Schema used throughout the triggers section. Note: *TransportRequest.itemID* is a foreign key with *MedicalEquipment.itemID*.

The purpose of the trigger was to require the type of medical equipment of a new transport request to be either a wheelchair or a recliner and display an error if it is not.

### 4.9.4.2 Oracle

```

1  CREATE OR REPLACE TRIGGER TransportationEquipment
2  BEFORE INSERT ON TransportRequest
3  FOR EACH ROW
4  DECLARE
5      v_equipmentType  VARCHAR2(40);
6  BEGIN
7      SELECT equipmentType INTO v_equipmentType
8      FROM MedicalEquipment
9      WHERE itemID = :NEW.itemID;
10
11     IF (v_equipmentType != 'Recliner' AND v_equipmentType != 'WheelChair') THEN
12         RAISE_APPLICATION_ERROR(-20001,
13             'ERROR: The equipment type for the transportation request is not a recliner or wheelchair!');
14     END IF;
15 END;
16 /

```

Figure 21: A trigger defined in OracleSQL that errors when specific equipment types are inserted into the *TransportRequest* table.

A first important trigger distinction of OracleSQL is the way they raise exceptions. The way OracleSQL raises exceptions is by using a procedure called “`RAISE_APPLICATION_ERROR()`”, which takes in an integer below -20000 and an error message as parameters. In addition to this distinction, there is also the fact that triggers in OracleSQL require a “/” at the end to properly compile.

#### 4.9.4.3 Postgres

```

1 CREATE OR REPLACE TRIGGER TransportationEquipment
2 BEFORE INSERT ON TransportRequest
3 FOR EACH ROW
4 EXECUTE FUNCTION TransportTrigger();
5
6 CREATE OR REPLACE FUNCTION TransportTrigger()
7 RETURNS TRIGGER AS $$
8 DECLARE
9     v_equipmentType VARCHAR(40);
10 BEGIN
11     SELECT equipmentType INTO v_equipmentType
12     FROM MedicalEquipment
13     WHERE itemID = :NEW.itemID;
14
15     IF (v_equipmentType != 'Recliner' AND v_equipmentType != 'WheelChair') THEN
16         RAISE EXCEPTION 'ERROR: The equipment type for the transportation request is not a recliner or wheelchair!';
17     END IF;
18 END;
19 $$ LANGUAGE plpgsql;

```

Figure 22: A trigger defined in PostgreSQL that errors when specific equipment types are inserted into the TransportRequest table.

One important distinction of PostgreSQL triggers is the method in which they are written. PostgreSQL triggers have no bodies of their own; they are only used to call a preexisting function, as is shown above. The function which is called features all the typical features of a normal function, including the language specifier and delimiter, however, it also specifies that it is returning a trigger. In the body of the function, the values of the new row are accessed using the keyword NEW, allowing a user to check the new rows for the purpose of the function, in this case making sure the equipment type is correct. At the end of the function body, but prior to the actual end statement, the value NEW should be returned, signaling the end of the trigger within the function. An additional distinction of PostgreSQL triggers lies in the way errors are raised. The way PostgreSQL raises exceptions is by using a the "RAISE EXCEPTION" statement with an error message inside the quotes.

#### 4.9.4.4 MySQL

```

1 DELIMITER //
2 CREATE TRIGGER TransportationEquipment
3 BEFORE INSERT ON TransportRequest
4 FOR EACH ROW
5 BEGIN
6     DECLARE v_equipmentType VARCHAR(40);
7     SELECT equipmentType INTO v_equipmentType
8     FROM MedicalEquipment
9     WHERE itemID = NEW.itemID;
10
11     IF v_equipmentType NOT IN ('Recliner', 'WheelChair') THEN
12         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
13         'ERROR: The equipment type for the transportation request is not a recliner or wheelchair!';
14     END IF;
15 END
16 // DELIMITER ;

```

Figure 23: A trigger defined in MySQL that errors when specific equipment types are inserted into the TransportRequest table.

The MySQL trigger syntax is similar to OracleSQL's syntax, but with a few notable differences. Firstly, MySQL requires delimiters at the beginning and end of the trigger to separate it from the rest of the query. Secondly, in MySQL, the DECLARE statements happen after the BEGIN statement whereas in OracleSQL and PostgreSQL this happens before the BEGIN. Thirdly, similar to procedures and functions, unlike OracleSQL and PostgreSQL, MySQL does not support the "OR REPLACE" clause. This means that in MySQL, if users need to modify an existing trigger, they have to manually drop it and then redefine it. In addition to these differences, the way MySQL raises exceptions is by using "SIGNAL SQLSTATE" statement, followed by the string value '45000', and the statement "SET MESSAGE\_TEXT = ' ' " with an error message inside the quotes. Another difference worth mentioning is that unlike the other two languages, MySQL supports an "IF NOT EXIST" clause that users can put in the trigger declaration to be able to avoid encountering an error if they try to create a trigger that they have already created.

#### 4.9.4.5 Analysis and Discussion

Triggers see significantly more differences between the three languages than views, procedures, or functions. All three languages feature the first three lines of the trigger being the same, specifying when and how it is called. Additionally, the body and error statements are functionally the same code, other than small syntactical differences. However, a notable difference lies in the fact that all three languages have different ways of raising exceptions. The biggest difference demonstrated is in PostgreSQL's setup. Rather than having a single trigger with a body to be executed when it is called, PostgreSQL requires a separate, preexisting function to be called by the trigger. This adds a layer of complexity to PostgreSQL triggers that are not displayed by MySQL or OracleSQL triggers. Furthermore, another distinction is that similar to procedures and functions, unlike OracleSQL and PostgreSQL, MySQL does not support the "OR REPLACE" clause for its triggers, meaning that any user modification requires dropping and redefining them. Despite this, however, unlike the other two languages, MySQL supports an "IF NOT EXIST" clause that users can put in the trigger declaration to be able to avoid encountering an error if they try to create a trigger that they have already created.

#### 4.9.5 Advanced Functionality Analysis and Discussion

The general structure of most of these functionalities is the same for all three languages, with views seeing no change, and the others most prominently seeing changes to their message outputting or error raising syntax, and notably having distinct delimiters that differ between all three. Among these differences, PostgreSQL's trigger system stands out as it requires a separate function to be called, as opposed to OracleSQL and MySQL which simply have the body within the trigger statement itself. Other pronounced differences lie in MySQL. To begin, unlike OracleSQL and PostgreSQL, MySQL lacks support the "OR REPLACE" clause for procedures, functions, and triggers alike. This means that if a user wishes to modify any of such objects in MySQL, they need to manually drop them and then redefine them. On top of these differences,

MySQL supports an “IF NOT EXIST” clause that the other two languages do not that users can put in procedure, function, and trigger declarations alike to be able to avoid encountering an error if they try to create such objects that they have already created.

## 4.10 Java Database Connectivity (JDBC)

### 4.10.1 Schema Used



Patient		Employee	
patientID 	number	employeeID 	number
firstName	varchar	userName	varchar
lastName	varchar	password	varchar
primaryPhone	varchar	firstName	varchar
city	varchar	lastName	varchar
state	varchar	salaryGrade	varchar
locationID	varchar	NPI	varchar
		LocationID	varchar

Figure 24: Schema used throughout the JDBC section.

What we had to do for this part of the project was to create a java program that would allow users to use the OracleSQL, MySQL, and PostgreSQL database systems to access information regarding specific hospital patients and employees and allow the employees to update their password. While creating this program, the biggest differences noticed were that all three systems had differences regarding the specific processes they require to successfully establish a connection to the database through Java.

### 4.10.2 Driver Installation

The first step for each language is downloading the associated JDBC driver. Each of these downloads can be found on the language’s website. Each language has a respective driver stored in a jar file.

### Oracle Database 23c (23.3.0.23.09) JDBC Driver & UCP Downloads

Supports Oracle Database versions - 23c, 21c, 19c, 18c, and 12.2.

Name	Download	JDK Supported	Description
Oracle JDBC driver	<a href="#">ojdbc11.jar</a>	Implements JDBC 4.3 spec and certified with JDK11, JDK17, and JDK19	Oracle JDBC driver except classes for NLS support in Oracle Object and Collection types. (7,107,784 bytes) - (SHA1: 405bcbcb8d8ab59f562125fad2b7e06d21649f3)
Oracle JDBC Driver	<a href="#">ojdbc8.jar</a>	Implements JDBC 4.2 spec and certified with JDK8 and JDK11	Oracle JDBC driver except classes for NLS support in Oracle Object and Collection types. (6,980,481 bytes) - (SHA1: d36f44a0ed8a07dcff2afef712ccdbd460d053d)
Universal Connection Pool (UCP) - ucp11.jar	<a href="#">ucp11.jar</a>	Certified with JDK11 and JDK17	Universal Connection Pool (UCP) <b>to be used with ojdbc11.jar</b> (1,526,942 bytes) - (SHA1: 2017015e2282f528a7daccf3d459913acb4a3f8)
Universal Connection Pool (UCP)	<a href="#">ucp.jar</a>	Certified with JDK8 and JDK11	Universal Connection Pool (UCP) <b>to be used with ojdbc8.jar</b> (1,485,969 bytes) - (SHA1: 9cdb5a2c3e48b14068671409f94ca98242b5676b)
Zipped JDBC driver (ojdbc11.jar) and Companion Jars	<a href="#">ojdbc11-full.tar.gz</a>	Certified with JDK11 and JDK17	This archive contains <b>ojdbc11.jar</b> , <b>ucp11.jar</b> , Reactive Streams Ingest (rsi.jar), companion jars <sup>1</sup> , JDBC, UCP, RSI Javadoc, and their Readmes. Refer to README.txt in the zip for details. (26,736,640 bytes) - (SHA1: e3b9ad024d611f830ee3def50cca0895d9d93bf2)
Zipped JDBC driver (ojdbc8.jar) and Companion Jars	<a href="#">ojdbc8-full.tar.gz</a>	Certified with JDK8 and JDK11	This archive contains <b>ojdbc8.jar</b> , <b>ucp.jar</b> , Reactive Streams Ingest (rsi.jar), companion jars <sup>1</sup> , JDBC, UCP, RSI Javadoc, and their Readmes. Refer to README.txt in the zip for details. (25,876,480 bytes) - (SHA1: 649db1f6db8b123b1f9e8755a80dc745a54e2f7)

Figure 25: OracleSQL Driver downloads page. Note: the first two .JAR files are typically the correct downloads for an introduction course.

General Availability (GA) Releases
Archives
i

## Connector/J 8.1.0

Select Operating System:

Platform Independent v

<b>Platform Independent (Architecture Independent), Compressed TAR Archive</b>	8.1.0	4.0M	<a href="#" style="background-color: #34495e; color: white; padding: 2px 8px; border-radius: 3px;">Download</a>
(mysql-connector-j-8.1.0.tar.gz)	MD5: 4a95d62b0cfcfbad68b92ffc62ae7ee266		<a href="#">Signature</a>
<b>Platform Independent (Architecture Independent), ZIP Archive</b>	8.1.0	4.8M	<a href="#" style="background-color: #34495e; color: white; padding: 2px 8px; border-radius: 3px;">Download</a>
(mysql-connector-j-8.1.0.zip)	MD5: d745362823ec4d37fa0607746d40a1b9		<a href="#">Signature</a>

Figure 26: MySQL Driver downloads page.

#### Java 8

42.6.0

If you are using Java 8 or newer then you should use the JDBC 4.2 version.

Download
Copy Maven

#### Java 7

42.2.27

If you are using Java 7 then you should use the JDBC 4.1 version.

Download
Copy Maven

#### Java 6

42.2.27

If you are using Java 6 then you should use the JDBC 4.0 version.

Download
Copy Maven

Figure 27: Postgres Driver downloads page. Note: the orange 'Download' buttons will download the .JAR file.

### 4.10.3 Registering the Driver

Each database system will then require the driver to be manually registered with the Driver Manager via the Class.forName function. This line will look different for each database system, though all refer to their driver.



```
Class.forName( className: "oracle.jdbc.driver.OracleDriver");
```

Figure 28: Java code for locating the Oracle Driver.

```
Class.forName( className: "com.mysql.cj.jdbc.Driver");
```

Figure 29: Java code for locating MySQL Driver.

```
Class.forName( className: "org.postgresql.Driver");
```

Figure 30: Java code for locating Postgres Driver.

#### 4.10.4 Establishing a Connection

All database systems will then initialize a connection object to access the databases through. Each system uses the connection object the same way, though the way in which it is initialized takes a different syntax for each.

```
var connection = DriverManager.getConnection(
    "jdbc:oracle:<Driver>:@<Host Name>:<Port>:<SID>",
    <User ID>,
    <Password>
);
```

Figure 31: Java code to open an Oracle driver connection. Note: <Driver>, <Host Name>, <Port>, <SID>, <User ID>, and <Password> should be replaced with the type of driver, the host name, the port number, the SID of the SQL server, the user ID, and the password respectively.

```
var connection = DriverManager.getConnection(
    "jdbc:mysql://<Host Name>:<Port>/<Database Name>",
    <User ID>,
    <Password>
);
```

Figure 32: Java code to open a MySQL driver connection. Note: <Host Name>, <Port>, <Database Name>, <User ID>, and <Password> should be replaced with the host name, the port number, the database name, the user ID, and the password respectively.

```
var connection = DriverManager.getConnection(
    "jdbc:postgresql://<Host Name>:<Port>/<Database Name>"
    <User ID>,
    <Password>
);
```

Figure 33: Java code to open a Postgres driver connection. Note: <Host Name>, <Port>, <Database Name>, <User ID>, and <Password> should be replaced with the host name, the port number, the database name, the user ID, and the password respectively.

#### 4.10.5 Analysis and Discussion

As seen in the differences described above, when comparing the aspect of Java Database Connectivity (JDBC) across the Oracle, Postgres, and MySQL database systems, the main

differences lie in their processes required to successfully connect to the database through Java. To begin, each system has its own version of the file required to get the JDBC driver required to be able to successfully connect to the database. The other additional differences were that across all three systems, the actual drivers required for the connection themselves also had different names, and that the connection strings required for the connection were formatted differently. Despite these differences, apart from the fact that different users may format database queries differently in Java, the Java code required to iterate through, retrieve, and update data within the database is the exact same across all three systems.

## 4.11 Connecting to the database using Python

### 4.11.1 Schema Used



Patient		Employee	
patientID 	number	employeeID 	number
firstName	varchar	userName	varchar
lastName	varchar	password	varchar
primaryPhone	varchar	firstName	varchar
city	varchar	lastName	varchar
state	varchar	salaryGrade	varchar
locationID	varchar	NPI	varchar
		LocationID	varchar

Figure 34: Schema used for the Python Connection section.

What we had to do for this part of the project was to create a python program that would allow users to use the Oracle, MySQL, and Postgres database systems to access information regarding specific hospital patients and employees and allow the employees to update their password. The libraries, though different, are used with overwhelming similar syntax. The biggest differences noticed were that all three systems had differences regarding the specific processes they require to successfully establish a connection to the database through Python, and how each library has different syntax to code up SQL prepared statements.

### 4.11.2 Oracle

The first step to successfully establish a connection to the database was to install and import the library to connect Python to the Oracle database:

```
pip install cx_Oracle
```

Figure 35: Terminal command to install the library that allows Python to connect to Oracle.

```
import cx_Oracle
```

Figure 36: Line of code in a Python project to import the cx\_Oracle library.

The library to connect Python to the Oracle database is called “cx\_Oracle”. The terminal command to install this library is “pip install cx\_Oracle”, and the line of Python code to use this library is “import cx\_Oracle”.

In addition to this difference, this library also requires users to download the Oracle Instant Client basic package:



Figure 37: ZIP format available for the Oracle Instant Client.

The name of this file may vary depending on the operating system used, but for all operating systems, the package can be found on the official Oracle website. Once the package has been installed, users must give the cx\_Oracle library access to all the contents inside it.

```
cx_Oracle.init_oracle_client(
    lib_dir = "/path/to/oracle_instantclient/instantclient-basic_#_#"
)
```

Figure 38: Python code to initialize the Oracle client. Note: '/path/to' should be replaced with the appropriate file location of the Oracle Instant Client.

The line of code to do this consists of a call to the cx\_Oracle library’s init\_oracle\_client() method, with the library directory variable “lib\_dir” assigned to the full pathname of inside the Oracle Instant Client Basic Package formatted as a string.

Another difference in the Python code used to finish establishing the connection to the Oracle database can be found in the parameters used inside the cx\_Oracle’s connect() method which makes the method successfully connect to the database:

```
connection = cx_Oracle.connect(
    user      = # Insert User ID #,
    password  = # Insert Password #,
    dns       = # Insert DNS #
)
```

Figure 39: Python code that sets up an Oracle connection with a user ID, password, and a DNS. Note: The DNS should be formatted as "<host name>:<port>/<SID>".

The first two parameters of this method are the user’s username and password that they use to be able to connect to the Oracle Server. The final parameter is a data source name assigned to a string consisting of the machine that the user is using to host their Oracle server, the port that they are using, and the service ID that they are using.

In terms of differences regarding the Python code required to iterate through, retrieve, and update data within the database, the `cx_Oracle` library has its own way of handling prepared statements.

```
preparedStatement = connection.cursor()
preparedStatement.execute(
    "UPDATE Employee SET password = :1 WHERE employeeID = :2",
    [<Insert Password>, <Insert Employee ID>]
)
```

Figure 40: Python code that creates a prepared statement which allows SQL queries to be executed in Oracle. Note: the `:1` and `:2` are placeholders for the `<Insert Password>` and `<Insert Employee ID>`.

In the `cx_Oracle` library, before executing any SQL statement, users must declare a cursor variable using the `connection.cursor()` method. Once they have the cursor, they can use it to execute SQL statements. For prepared statements, placeholders are denoted by a “:” followed by a sequential number. To properly substitute these placeholders during execution, the user can pass in a list of variables, enclosed by square brackets, as the second argument to the `execute()` method of the cursor. The order of these variables in the list corresponds to the order of the placeholders in the query. For instance, `:1` will be replaced by the first value in the list, `:2` by the second, and so forth.

#### 4.11.3 MySQL

The first step used to successfully establish a connection to the database was to install and import the library to connect Python to the MySQL database:

```
pip install mysql.connector
```

Figure 41: Terminal command to install the library that allows Python to connect to MySQL.

```
import mysql.connector
```

Figure 42: Line of code in a Python project to import the `mysql.connector` library.

The library to connect Python to the MySQL database is called “`mysql.connector`”. The terminal command to install this library is “`pip install mysql.connector`”, and the line of Python code required to use this library is “`import mysql.connector`”.

Another difference in the Python code required to finish establishing the connection to the MySQL database can be found in the parameters used inside `mysql.connector` library’s `connect()` method required to make the method successfully connect to the database:

```

connection = mysql.connector.connect(
    user      = <Insert User ID>,
    password  = <Insert Password>,
    host      = <Insert Host Name>,
    port      = <Insert Port>,
    database  = <Insert Database Name>
)

```

Figure 43: Python code that sets up a MySQL connection with a user ID, password, host, port, and database name.

The first two parameters of this method are the user's username and password that they use to be able to connect to the MySQL Server. For the remaining parameters, the user has to pass in the machine that they are using to host their MySQL server, the port that they are using, and the name of the database that they are using.

In terms of differences regarding the Python code used to iterate through, retrieve, and update data within the database, the `mysql.connector` library handles prepared statements differently than the how `cx_Oracle` library does.

```

preparedStatement = connection.cursor()
preparedStatement.execute(
    "UPDATE Employee SET password = %s WHERE employeeID = %s",
    (<Insert Password>, <Insert Employee ID>)
)

```

Figure 44: Python code that creates a prepared statement which allows SQL queries to be executed in MySQL. Note: the '%s's are placeholders for the <Insert Password> and <Insert Employee ID>.

Similar to the the `cx_Oracle` library, in the `mysql.connector` library, before executing any SQL statement, users must declare a cursor variable using the `connection.cursor()` method. Once they have the cursor, they can use it to execute SQL statements. However, for prepared statements in this library, placeholders are denoted by '%s', irrespective of the datatype. To properly substitute these placeholders during execution, the user can pass in a tuple of variables, enclosed by parentheses instead of square brackets, as the second argument to the `execute()` method of the cursor. The order of these variables in the tuple should correspond to the order of the placeholders in the query. For instance, the first '%s' will be replaced by the first value in the tuple, the second '%s' by the next value, and so on.

#### 4.11.4 Postgres

The first step to successfully establish a connection to the database was to install and import the library to connect Python to the Postgres database:

```
pip install psycopg2
```

Figure 45: Terminal command to install the library that allows Python to connect to Postgres.

```
import psycopg2
```

Figure 46: Line of code in a Python project to import the psycopg2 library.

The library to connect Python to the Oracle database is called “psycopg2”. The terminal command to install this library is “pip install psycopg2”, and the line of Python code needed to use this library is “import psycopg2”.

Another difference in the Python code used to finish establishing the connection to the MySQL database can be found in the parameters used inside psycopg2 library’s connect() method required to make the method successfully connect to the database:

```
connection = psycopg2.connect(
    dbname      = <Insert Database Name>
    host        = <Insert Host Name>,
    port        = <Insert Port>,
)
```

Figure 47: Python code that sets up a Postgres connection with a database name, host, and port.

The first two parameters used in this method are the name of the database and the machine that the user is using to host their Postgres server. For the remaining parameter, the user must pass in the port that they are using. When using the psycopg2 library to connect to the database through python, the user is not required to enter the username and password that they normally need to use to connect to the database.

In terms of differences regarding the Python code used to iterate through, retrieve, and update data within the database, the psycopg2 library handles prepared statements differently than how cx\_Oracle library does.

```
preparedStatement = connection.cursor()
preparedStatement.execute(
    "UPDATE Employee SET password = %s WHERE employeeID = %s",
    (<Insert Password>, <Insert Employee ID>)
)
```

Figure 48: Python code that creates a prepared statement which allows SQL queries to be executed in Postgres. Note: the ‘%s’s are placeholders for the <Insert Password> and <Insert Employee ID>.

Similar to the `cx_Oracle` library, in the `psycopg2` library, before executing any SQL statement, users must declare a cursor variable using the `connection.cursor()` method. Once they have the cursor, they can use it to execute SQL statements. However, for prepared statements in this library, placeholders are denoted by `'%s'`, irrespective of the datatype. To properly substitute these placeholders during execution, the user can pass in a tuple of variables, enclosed by parentheses instead of square brackets, as the second argument to the `execute()` method of the cursor. The order of these variables in the tuple should correspond to the order of the placeholders in the query. For instance, the first `'%s'` will be replaced by the first value in the tuple, the second `'%s'` by the next value, and so on.

#### 4.11.5 Analysis and Discussion

As seen in the differences described above, when comparing the aspect of connecting to the database using Python across the Oracle, Postgres, and MySQL database systems, the main differences lie in their processes required to successfully connect to the database through Java. To begin, each system had its own library used to connect to the database that needed to be installed and imported into Python. Oracle requires a bit more installations as even with the `cx_Oracle` Python library, the user still needs to download the basic Oracle Instant Client package and give Python access to that package to be able to successfully connect to the database. In addition to these differences, while each of the systems' python libraries used to connect to the database use a `connect ()` method to do so, they do not all require the same parameters to be passed into the method. A key example that demonstrates this is while the `cx_Oracle` and `mysql.connector` Python libraries require authentication from the user to connect to the database, the `psycopg2` Python library used to connect to the Postgres database does not. In terms of the Python code used to iterate through, retrieve, and update data within the database, the only differences found were with how each of the three systems' Python connection libraries handled prepared statements. While all three libraries require the user to declare a cursor variable using the `connection.cursor()` method to execute prepared statements, the way placeholders are denoted is not the same across the three libraries. While they are denoted by a `' :` followed by a sequential number in the `cx_Oracle` library, they are denoted by a `'%s'`, irrespective of the datatype, in the `psycopg2` and the `mysql.connector` libraries. The second difference related to the way the three systems' Python connection libraries handle prepared statements is the way to substitute the placeholders in the statements. In the `cx_Oracle` library, the placeholders are sequentially substituted by variables inside a list enclosed by square brackets while they are sequentially substituted by variables inside a tuple enclosed by parentheses.

## 5 Future Works

In the future, this research could be expanded in a variety of ways. There are a few categories that may be relevant to introductory courses that were not discussed within this research, such as some more intricate elements of nested querying and the efficacy of error descriptions. Additionally, because this research only focuses on courses that teach the very fundamentals of relational databases, similar studies could be performed on more technical courses, or simply more advanced database courses. There are several topics that are not discussed in this paper that would be incredibly useful for professors hoping to teach a higher-level course. Therefore, conducting research on these topics would be crucial to help guide them towards the right database system to use for such courses. Furthermore, it would be beneficial to research alternative programming languages in addition to Java and Python as a means of connecting to SQL servers from running code as opposed to IDE's. Many languages have their own methods for connecting to these servers natively or through third-party libraries and package managers. An example of this would be how other popular languages such as C and its extension C++ use ODBC (Open Database Connectivity) as a standard application programming interface (API) in order to connect to databases. This would be beneficial for introductory courses to be aware of as students may have past knowledge of a variety of languages and not necessarily all the same ones.



## 6 Conclusion

From analyzing all the results collected, it is evident that the database systems Oracle, Postgres, and MySQL each have their own strengths and weaknesses in terms of how suitable they are for students in an introductory database course.

<i>Category</i>	<b>Oracle</b>	<b>Postgres</b>	<b>MySQL</b>
Installation and Setup	Difficult	Moderate	Moderate
Proprietary IDEs	Simple	Moderate	Simple
Datatypes	Simple	Complex	Complex
Table Management	Easy	Easy	Easy
Data Manipulation	Easy	Easy	Easy
Query Simplicity	Easy	Easy	Easy
Advanced Functionality	Easy	Moderate	Moderate
Java/Python Connectivity	Easy	Easy	Easy

Figure 49: Summary of comparisons across Oracle, Postgres, and MySQL

Figure 49 surmises the differences and comparisons we located throughout our project. For instance, when considering the aspects of installation and setup, MySQL and Postgres may be more adequate choices for an introductory database course than Oracle due to their less complex processes required to do so. The same can be said when considering the aspect of flexibility in terms of the SQL code for each of these systems. This is because the SQL dialect of MySQL and PostgreSQL supports certain table commands and constraints that are not directly supported in OracleSQL, giving them more flexibility. On the other hand, when considering the aspect of the systems' proprietary IDEs, Postgres's pgAdmin 4 is the one that may be the least suitable for newer users. The main reason for this is because while pgAdmin 4 supports advanced features that might be very useful for professional database users, it can be quite overwhelming and ambiguous for students in an introductory database course. However, if a third-party IDE such as Datagrip or DBeaver is used instead, then this might not be an issue in deciding which database system to use. With respect to aspects of advanced functionality such as triggers, procedures, and functions, OracleSQL can be considered the most suitable system to use for students in an introductory database. OracleSQL has the most intuitive coding practices and syntaxes, while MySQL and PostgreSQL have more difficult syntactical requirements which could confuse new users.

The varying features and intricacies of Oracle, Postgres, and MySQL illustrate that each system brings its own set of advantages and challenges for use in an introductory database course. Because of this, the most suitable database system to use for an introductory database course largely depends on the course's unique requirements, learning outcomes, and pedagogical approach. Therefore, it is up to the course instructor to select the system that aligns best with the instructional goals and structure of their course.

## 7 Works Cited

- Aggarwal, D., Winstead, C., & Tufte, K. (2020). *Leveraging Industry Benchmarks to Teach Database Concepts*. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* 1410–1410. <https://doi.org/10.1145/3328778.3372562>
- Al-Roomi, M., Al-Ebrahim, S., Buqrais, S., & Ahmad, I. (2013). Cloud Computing Pricing Models: A Survey. *International Journal of Grid and Distributed Computing*, 6(5), 93–106. <https://doi.org/10.14257/ijgdc.2013.6.5.09>
- Amazon Web Services. (n.d.). *Open Source Databases - Amazon Web Services*. Retrieved September 3, 2023, from <https://aws.amazon.com/products/databases/open-source-databases/>
- Amazon Web Services (2019). *Amazon RDS for PostgreSQL – Amazon Web Services (AWS)*. Amazon Web Services, Inc. <https://aws.amazon.com/rds/postgresql/>
- Bi, Y., & Beidler, J. (2008). Teaching database systems with web applications team projects. *Journal of Computing Sciences in Colleges*, 23(3), 82-88. <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/pdf/10.5555/1295109.1295130>
- Center for Project Based Learning» *Project-Based Learning at WPI*. (n.d.). Wp.wpi.edu. Retrieved September 28, 2023, from <https://wp.wpi.edu/projectbasedlearning/proven-pedagogy/project-based-learning-at-wpi/#:~:text=Since%201970%2C%20project%2Dbased%20learning>
- Ceri, S., Cochrane, R. J., Widom, J., & Di Milano, P. (n.d.). *Practical Applications of Triggers and Constraints: Successes and Lingering Issues*.
- Codd, E. F. (1982). Relational database: a practical foundation for productivity. *Commun. ACM*, 25(2), 109–117. <https://doi.org/10.1145/358396.358400>
- DB-Engines. (2019). DB-Engines Ranking. DB-Engines. <https://db-engines.com/en/ranking>

DB-Engines. (February 1, 2023). Ranking of the most popular database management systems worldwide, as of February 2023 [Graph]. In Statista. Retrieved September 04, 2023, from

<https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-system>

*DB-Fiddle*. (n.d.). Db-fiddle.com. <https://www.db-fiddle.com/>

*DB<>Fiddle*. (n.d.). DB<>Fiddle. <https://dbfiddle.uk>

Deng, Y., Frankl, P., & Wang, J. (2004). *Testing web database applications*. *ACM SIGSOFT Software Engineering Notes*, 29(5), 1-10. <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/10.1145/1022494.1022528>

Dolezel, D., & McLeod, A. (2021). Big-Data Skills: Bridging the Data Science Theory-Practice Gap in Healthcare. *Perspectives in Health Information Management*, 18(Winter).

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7883353/>

Domdouzis, K., Lake, P., & Crowther, P. (2021). Hierarchical Databases. In *Concise Guide to Databases: A Practical Introduction* (pp. 205-212). Cham: Springer International Publishing.

[https://link.springer.com/chapter/10.1007/978-3-030-42224-0\\_9](https://link.springer.com/chapter/10.1007/978-3-030-42224-0_9)

DuBois, P. (2009). Writing MySQL Programs Using C. *MySQL, 4th Ed.*, 359–434.

[https://books.google.com/books?hl=en&lr=&id=JgFTUsIC0bUC&oi=fnd&pg=PT13&dq=%22Writing+MySQL+Programs+Using+C%22&ots=Dt\\_JtMLwHm&sig=NAbjkEdF11zJ-zmDguIa1TPkOP4#v=onepage&q=%22Writing%20MySQL%20Programs%20Using%20C%22&f=false](https://books.google.com/books?hl=en&lr=&id=JgFTUsIC0bUC&oi=fnd&pg=PT13&dq=%22Writing+MySQL+Programs+Using+C%22&ots=Dt_JtMLwHm&sig=NAbjkEdF11zJ-zmDguIa1TPkOP4#v=onepage&q=%22Writing%20MySQL%20Programs%20Using%20C%22&f=false)

E. F. Codd. 1982. Relational database: a practical foundation for productivity. *Commun. ACM*

25, 2 (Feb 1982), 109–117. <https://doi.org/10.1145/358396.358400>

Eder, J. (1992). *Logic and Databases*. 95–103. [https://doi.org/10.1007/3-540-55681-8\\_32](https://doi.org/10.1007/3-540-55681-8_32)

- Fadlallah, H. (2021, August 9). *Popular free SQL online compilers*. SQL Shack - Articles about Database Auditing, Server Performance, Data Recovery, and More.  
<https://www.sqlshack.com/popular-free-sql-online-compilers/>
- Feasel, J. (2012). *SQL Fiddle | A tool for easy online testing and sharing of database problems and their solutions*. SQL Fiddle. <http://www.sqlfiddle.com/about.html>
- Fekete, A. (2005, June). Teaching transaction management with SQL examples. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 163-167). <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/pdf/10.1145/1384271.1384382>
- Fekete, A. D., & Röhm, U. (2022). Teaching about Data and Databases: Why, What, How?. *ACM SIGMOD Record*, 51(2), 52-60. <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/pdf/10.1145/3552490.3552504>
- Firth, J., Torous, J., Stubbs, B., Firth, J. A., Steiner, G. Z., Smith, L., ... & Sarris, J. (2019). The “online brain”: how the Internet may be changing our cognition. *World Psychiatry*, 18(2), 119-129. <https://onlinelibrary.wiley.com/doi/full/10.1002/wps.20617>
- Gallini, N. I., Denisenko, A. A., Kazak, A. N., Linnik, I. I., Chetyrbok, P. V., & Sergeeva, E. A. (2022, January). Teaching the Development of Information Web Portals Using ASP. NET Technology. In *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)* (pp. 637-640). IEEE.  
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9755667>
- Haxhiu, V. (2018). Decision making based on data analyses using data warehouses. *International Journal of Business and Technology*, 6(3), 1-6.  
<https://doi.org/10.33107/IJBTE.2018.6.3.04>.

- Hidalgo, E. S. (2019). Adapting the scrum framework for agile project management in science: case study of a distributed research initiative. *Heliyon*, 5(3).  
<https://doi.org/10.1016/J.HELIYON.2019.E01447>
- Huang, C. Y. (2019). Integrative curriculum for teaching databases. *Journal of Computing Sciences in Colleges*, 34(3), 131-131. <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/pdf/10.5555/3306465.3306487>
- Jiang, L., & Nandi, A. (2015). Designing interactive query interfaces to teach database systems in the classroom. *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 1479-1482).  
<https://doi.org/10.1145/2702613.2732900>
- Koulouri, T., Lauria, S., & Macredie, R. D. (2014). Teaching Introductory Programming. *ACM Transactions on Computing Education (TOCE)*, 14(4). <https://doi.org/10.1145/2662412>
- Lerner, R. M. (2007). Open-source databases, part ii: PostgreSQL. *Linux J*, 157, 16.  
<https://www.ecb.torontomu.ca/~courses/coe518/LinuxJournal/elj2007-157-PostgreSQL2.pdf>
- Maiorana, F. (2014, April). Teaching Web Programming. In *Proceedings of the 6th International Conference on Computer Supported Education* (Vol. 2, pp. 49-56).  
<https://pdfs.semanticscholar.org/c789/56a9682473582fedfa769b763a33f3e03247.pdf>
- Melton, J. (1996). SQL Language Summary. *ACM Computing Surveys*, 28(1).  
<https://dl.acm.org/doi/pdf/10.1145/234313.234374>
- Microsoft (n.d.) *Azure Database for PostgreSQL* | Microsoft Azure. [Azure.microsoft.com](https://azure.microsoft.com/en-us/products/postgresql).  
<https://azure.microsoft.com/en-us/products/postgresql>

- Mior, M. J. (2023, June). Relational Playground: Teaching the Duality of Relational Algebra and SQL. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging education practice with education research* (pp. 56-58). <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/abs/10.1145/3596673.3596978>
- MongoDB. (n.d.-a). *What is an Object-Oriented Database?* MongoDB. <https://www.mongodb.com/databases/what-is-an-object-oriented-database>
- MongoDB (n.d.-b) *What Is A JSON Database? | All You Need To Know*. MongoDB. <https://www.mongodb.com/databases/json-database#:~:text=JSON%20databases%20are%20part%20of>
- MongoDB (n.d.-c) *What Is A Database Application?* MongoDB. <https://www.mongodb.com/basics/database-application>
- MongoDB. (2019). *NoSQL Databases Explained*. MongoDB. <https://www.mongodb.com/nosql-explained>
- Motro, A. (1993, June). What to teach about databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data* (p. 420). <https://dl.acm.org/doi/pdf/10.1145/170035.170108>
- MySQL. (n.d.-a). *MySQL :: MySQL Enterprise Edition*. Retrieved August 16, 2023, from <https://www.mysql.com/products/enterprise/>
- MySQL. (n.d.-b). *What Is MySQL? | Oracle*. Retrieved August 16, 2023, from <https://www.oracle.com/mysql/what-is-mysql/>.
- MySQL (2019). *Oracle MySQL Cloud Service*. Mysql.com. <https://www.mysql.com/cloud/>

Nandi, B. K. (2021). *Construct a customer database from PDF bank statements using Python programming and Microsoft SQL* (Doctoral dissertation, Brac University).

[17366002\\_CSE.pdf](#)

Oracle. (n.d.-a). *Cloud Infrastructure* | Oracle. Retrieved August 16, 2023, from <https://www.oracle.com/cloud/>

Oracle. (n.d.-b). *Database Software Downloads* | Oracle. Retrieved August 16, 2023, from <https://www.oracle.com/database/technologies/oracle-database-software-downloads.html>

Oracle. (n.d.-c). *Oracle Database 21c*. Oracle Help Center <https://docs.oracle.com/en/database/oracle/oracle-database/21/>

Oracle. (2020). *What is OLTP?* Oracle.com. <https://www.oracle.com/database/what-is-oltp/>

Oracle. (2022). *What is a database?* Www.oracle.com. <https://www.oracle.com/database/what-is-database/>

Oracle. (2023). *What is a relational database?* Oracle.com. <https://www.oracle.com/database/what-is-a-relational-database/>

Özsu, M. T., & Valduriez, P. (1999). *Principles of distributed database systems* (Vol. 2). Englewood Cliffs: Prentice Hall. <https://link.springer.com/content/pdf/10.1007/978-3-030-26253-2.pdf>

pgAdmin. (n.d.-a). *pgAdmin - PostgreSQL Tools*. Retrieved August 17, 2023, from <https://www.pgadmin.org/>

pgAdmin. (n.d.-b). *pgAdmin Download*. Retrieved August 17, 2023, from <https://www.pgadmin.org/download/>

Pluciennik, E., & Zgorzałek, K. (2017). The multi-model databases—a review. In *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and*

*Knowledge Representation: 13th International Conference, BDAS 2017, Ustroń, Poland, May 30-June 2, 2017, Proceedings 13* (pp. 141-152). Springer International Publishing.

[https://link.springer.com/chapter/10.1007/978-3-319-58274-0\\_12](https://link.springer.com/chapter/10.1007/978-3-319-58274-0_12)

PostgreSQL. (n.d.-a). *PostgreSQL: About*. Retrieved August 17, 2023, from

<https://www.postgresql.org/about/>

PostgreSQL. (n.d.-b). *PostgreSQL: Downloads*. Retrieved August 17, 2023, from

<https://www.postgresql.org/download/>

PostgreSQL. (n.d.-c). *PostgreSQL: The world's most advanced open source database*. Retrieved

August 17, 2023, from <https://www.postgresql.org/>

PostgreSQL. (n.d.-d). *PostgreSQL: Version History*. Retrieved August 17, 2023, from

<https://www.postgresql.org/ftp/source/>

*Project-Based Learning at WPI | PBL in Higher Education*. (n.d.). [www.wpi.edu](http://www.wpi.edu).

<https://www.wpi.edu/project-based-learning>

Qiu, T., Feng, M., Lu, S., Li, Z., Wu, Y., Zoltowski, C. B., & Lu, Y. H. (2017). Online

Programming System for Code Analysis and Activity Tracking. *ASEE Annual Conference and Exposition, Conference Proceedings, 2017-June*. <https://doi.org/10.18260/1-2--28722>

Rilett, D., & Russo, J. P. (2013). Using Amazon web services to teach web-enabled database

concepts. *Journal of Computing Sciences in Colleges*, 28(6), 134-135. <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/pdf/10.5555/2460156.2460181>

Sarkan, H. M., Ahmad, T. P. S., & Bakar, A. A. (2011). Using JIRA and redmine in requirement

development for Agile methodology. *2011 5th Malaysian Conference in Software*

*Engineering, MySEC 2011*, 408–413. <https://doi.org/10.1109/MYSEC.2011.6140707>



- Sasaki, B. M., Chao, J., & Howard, R. (2018). Graph databases for beginners. *Neo4j*.  
[https://go.neo4j.com/rs/710-RRC-335/images/Graph\\_Databases\\_for\\_Beginners\\_v4.pdf](https://go.neo4j.com/rs/710-RRC-335/images/Graph_Databases_for_Beginners_v4.pdf)
- Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM model for agile methodology. *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017, 2017-January*, 864–869.  
<https://doi.org/10.1109/CCAA.2017.8229928>
- Udoh, E. (2006). *Teaching Database in an Integrated Oracle Environment*. In *Working group reports on ITiCSE on Innovation and technology in computer science education* (pp. 71-74).  
<https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/pdf/10.1145/1189215.1189174>
- Wagner, P. J., Shoop, E., & Carlis, J. V. (January 2003). Using scientific data to teach a database systems course. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education* (pp. 224-228). <https://dl-acm-org.ezpv7-web-p-u01.wpi.edu/doi/pdf/10.1145/611892.611975>
- Wang, M. (2003). Teaching Case: E-Business Application Development with Java Technology and Oracle: The Fortune Invest Inc. Case. *Journal of Information Systems Education*, 14(3), 293. <http://jise.org/Volume14/n3/JISEv14n3p293.pdf>
- Zayour, I., & Hajjdiab, H. (2013). How Much Integrated Development Environments (IDEs) Improve Productivity? *Journal of Software*, 8(10), 2425-2431.  
<https://doi.org/10.4304/jsw.8.10.2425-2431>

## 8 Appendices

### 8.1 Appendix A: Oracle Installation for macOS through Docker

# Prerequisites

---

1. Make sure to have an [Oracle account](#) set up.
2. Make sure to have [SQL Developer](#) installed.

If you do not have 2 done, go to slide 9 of this presentation.

Note: There are two types of macOS architecture: Apple chips and Intel chips. Using Apple chips require a few extra steps. These steps are only necessary for Macs using an Apple M1 Chip or an Apple M2 Chip.

# Installation Process: Steps 1-2

---

1. **Apple Chip Only:** Open your Terminal app and enter the command `softwareupdate --install-rosetta` to install Rosetta 2 which will allow Docker and Colima to work properly.
2. Navigate to the [Docker Desktop download site](#). Select the 'Intel Chip' option or the 'Apple Chip' option depending on your computer's architecture and make sure to permit the download from their website by pressing 'Allow'.

## Install Docker Desktop on Mac

This page contains information about system requirements, download URLs, and instructions on how to install Docker Desktop for Mac.

Docker Desktop for Mac with Intel chip

Docker Desktop for Mac with Apple silicon

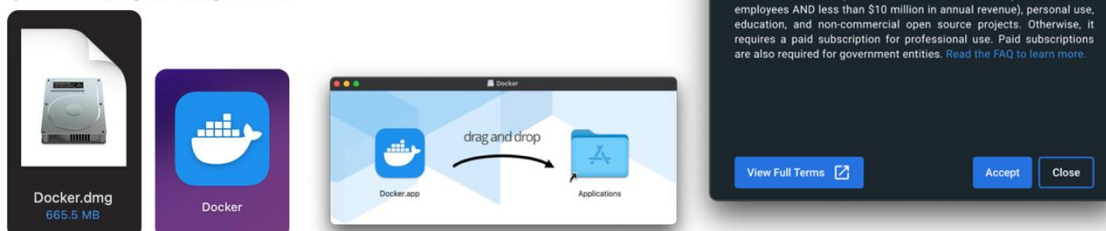
Do you want to allow downloads on  
"www.docker.com"?

You can change which websites can download files in the  
Websites section of Safari Settings.

Cancel Allow

## Installation Process: Steps 3-5

3. Double-click on the [Docker.dmg](#) file in your Downloads folder and drag the [Docker.app](#) into your Applications folder.
4. After that, you can open the **Docker** app just as you would any other app (launchpad, spotlight search, finder, etc.)
5. Press 'Accept', select 'Use recommended settings', press 'Finish', and enter your password to allow Docker access to your machine. You can also skip through the questions about your reasoning for using Docker.



## Installation Process: Steps 6-8

6. Open your preferred terminal emulator or Terminal app to install some command line tools that will allow us to download the Oracle container, make an image of it, start the Oracle server, and eventually access it from SQL Developer.
7. Firstly, if you do not have [Homebrew](#) installed, run the command `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"` to install it.
8. **Apple Chip Only:** You need to install a tool called [colima](#), using the command `brew install colima`, which will enable M1 and M2 support for Docker. This will essentially trick Docker into thinking that it is a 32-bit machine. Then, to start [colima](#), enter the command `colima start --arch x86_64 --memory 4`. We are specifying 4GB of memory in this command, but you can change that depending on your circumstances and project. This command may take a while but wait it out and confirm it finishes.

## Installation Process: Steps 9-10

- Next, the command `docker run --name <container-name> -d -p 1521:1521 -e ORACLE_PASSWORD=<container-password> gvenzl/oracle-xe` will access the Docker Container Hub and pull [gvenzl/oracle-xe](#) into a local image on your computer. After cloning the image, Docker will proceed to create a copy of this image in a container and run the new container. In this step make sure to replace **<container-name>** and **<container-password>** with a name and password, both containing letters and/or numbers only, related to your project. A good example username/password combo for this project could be `OracleContainer/password`. You will use this name to see container statistics, execute commands, and other useful tools that Docker provides.
- After this command succeeds, your Oracle Express container is set up. To start the server, run the command `docker start <container-name>` while replacing `<container-name>` with the name from **Download Process: Step 9**.

## Installing SQLDeveloper: Steps 1-3

- Go to this link:  
<https://www.oracle.com/database/sqldeveloper/technologies/download/>
- If your mac runs on an intel chip, click on the download button associated with "Mac OSX with JDK 11 included". If your mac runs on an m1 or m2 chip, click on the download button associated with "Mac ARM64 with JDK 11 included"
- Make sure to accept the Oracle License Agreement to be able to download this software.

The screenshot shows the Oracle SQL Developer download page. It features two download buttons: "Download (462 MB)" for "Mac OS X with JDK 11 included" and "Download (458 MB)" for "Mac ARM64 with JDK 11 included". To the right of these buttons are links for MD5, SHA1, and Installation Notes for each version. Below the download options is a license agreement dialog box with the text: "You must accept the Oracle License Agreement to download this software." The dialog box contains a checked checkbox labeled "I reviewed and accept the Oracle License Agreement" and a "Required" label. At the bottom of the dialog, it says "You will be redirected to the login screen in order to download the file." and a green button labeled "Download sqldeveloper-23.1.0.0971607-macos-x64.app.zip".

## Installing SQLDeveloper: Step 4

4. Double-click on the SQLDeveloper file now in your Downloads folder and drag the SQLDeveloper IDE into your Applications folder



Favorites	Name	Size	Kind	Date Added
AirDrop	SQLDeveloper 3	678.7 MB	Application	Today at 20:24
Recents	SQLDeveloper 2	678.7 MB	Application	Today at 20:24
Applications	sqldeveloper-23.1.0.097.1607-macos-x64.app (1).zip	462.4 MB	ZIP archive	Today at 20:19
Desktop	11364-18534-1-SM.pdf	834 KB	PDF Document	Today at 18:13
Documents	Oracle Installation for Windows.pptx	1.7 MB	PowerP...(pptx)	Yesterday at 17:07
Downloads	Oracle Installation for ma...S through Docker (1).pptx	574 KB	PowerP...(pptx)	Yesterday at 17:00
	Skeleton-copy	675 KB	Micros...(docx)	Yesterday at 16:34
	free-sample_study_id134_...ra-in-north-america.pdf	230 KB	PDF Document	Yesterday at 16:36

## 8.2 Appendix B: Using Oracle on macOS through Docker

### Creating Users: Steps 1-2

---

1. To create a user to log in with, enter the command `docker exec -it <container-name> sqlplus system/<container-password>@//localhost/<database-name>`. In this step, make sure to replace `<container-name>` with the name of container you created in **Download Process: Step 9**, `<container-password>` with the container password you created in **Download Process: Step 9**, and `<database-name>` with a good name for your database, containing letters and/or numbers only. A good example database name is OracleExpressDB. As a word of caution, you may encounter an error message when running this command, but the command should still execute properly without any issues.
2. Enter `/ as sysdba` exactly (without the quotes). Make sure to have the spaces and spelling correct. This should bypass and password request and put you straight in the server with full admin access. Note: you are now interacting with the SQL Server through Command Line Interface, rather than interacting with your computer's terminal.

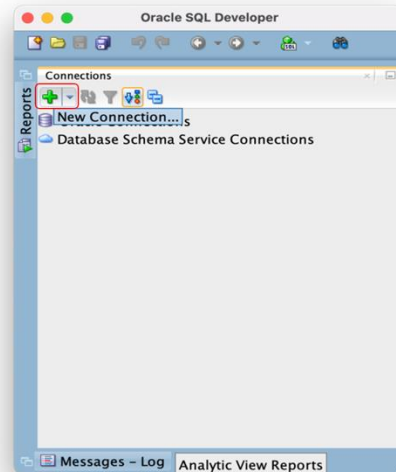
### Creating Users: Steps 3-5

---

3. Now, we need to create a user within the SQL Server for you to access from SQL developer. To do this, enter the command `create user <username> identified by <password>`; while making sure to include the semicolon at the end, replacing `<username>` with your WPI username (without @wpi.edu), and `<password>` with a password containing letters and/or numbers only that you will remember. **This is important as this is the username and password combination that you will use when connecting to the server using an IDE such as SQLDeveloper.**
4. For this user to make changes to the SQL Server, you need to grant all the basic Oracle privileges to the user. We will do this using the command `grant all privileges to <username>`; while making sure to include the semicolon at the end and replacing `<username>` with the username of the user you created in **Creating Users: Step 3**.
5. After all that's done, you can use the command `exit`; to leave SQL mode in your terminal.

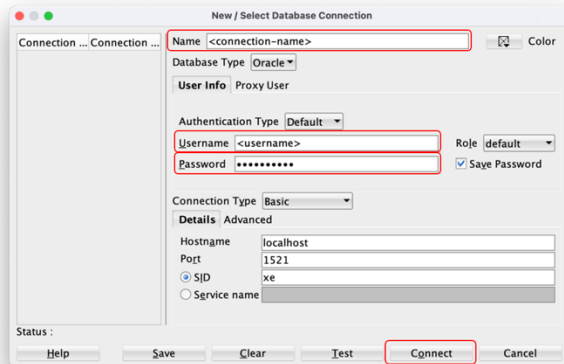
## Connecting to SQLDeveloper: Step 1

1. After this succeeds, open your SQL Developer app, where you will interact with the database. To create a new connection, press on the green plus in the top left of the window.



## Connecting to SQLDeveloper: Step 2


2. In the pop-up window, fill in the 'Name', 'Username', and 'Password' fields. The name should be relevant to what database you are connecting to; A good name could be OracleExpressDBConnection. The Username and password fields should be the same as what you entered in **Creating Users: Step 3**. After doing all this, press 'Connect'




## 8.3 Appendix C: Oracle Installation for Windows
























### Initial download

- ▶ At this [link](#), download the most recent express edition for Windows.

Oracle Database 21c Express Edition	
Download	Description
 Oracle Database 21c Express Edition for Windows x64	(1,967,615,483 bytes - October 08, 2021) [Sha256sum: 939742c3305c466566a55f607638621b6aa7033a183175f6bcd6cffb48e6bc3f]

### Zip Extraction

- ▶ Right-click the downloaded folder and extract it. Open the extracted folder, it should look like this 

 0x0404.ini	6/7/2023 11:59 AM	Configuration sett...	11 KB
 0x0407.ini	6/7/2023 11:59 AM	Configuration sett...	26 KB
 0x0409.ini	6/7/2023 11:59 AM	Configuration sett...	22 KB
 0x0410.ini	6/7/2023 11:59 AM	Configuration sett...	25 KB
 0x0411.ini	6/7/2023 11:59 AM	Configuration sett...	15 KB
 0x0412.ini	6/7/2023 11:59 AM	Configuration sett...	14 KB
 0x0416.ini	6/7/2023 11:59 AM	Configuration sett...	24 KB
 0x0804.ini	6/7/2023 11:59 AM	Configuration sett...	11 KB
 1028.mst	6/7/2023 11:59 AM	MST File	76 KB
 1031.mst	6/7/2023 11:59 AM	MST File	104 KB
 1033.mst	6/7/2023 11:59 AM	MST File	20 KB
 1034.mst	6/7/2023 11:59 AM	MST File	104 KB
 1036.mst	6/7/2023 11:59 AM	MST File	104 KB
 1040.mst	6/7/2023 11:59 AM	MST File	104 KB
 1041.mst	6/7/2023 11:59 AM	MST File	96 KB
 1042.mst	6/7/2023 11:59 AM	MST File	92 KB
 1046.mst	6/7/2023 11:59 AM	MST File	96 KB
 2052.mst	6/7/2023 11:59 AM	MST File	76 KB
 DB.cab	6/7/2023 12:00 PM	Cabinet File	1,907,646 ...
 Oracle Database 21c Express Edition...	6/7/2023 12:00 PM	Windows Installer ...	4,145 KB
 Setup.exe	6/7/2023 12:00 PM	Application	1,271 KB
 Setup.ini	6/7/2023 12:00 PM	Configuration sett...	6 KB
 XEinstall.rsp	6/7/2023 12:00 PM	RSP File	1 KB



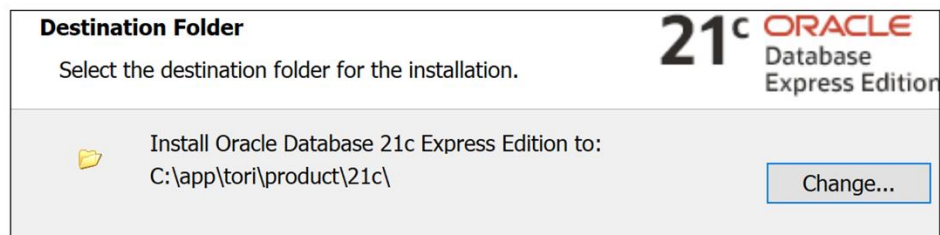
## Beginning Installation

- ▶ Run the setup.exe file in the folder to begin the installation and allow it to make changes to your device. The installer will launch and look like this →



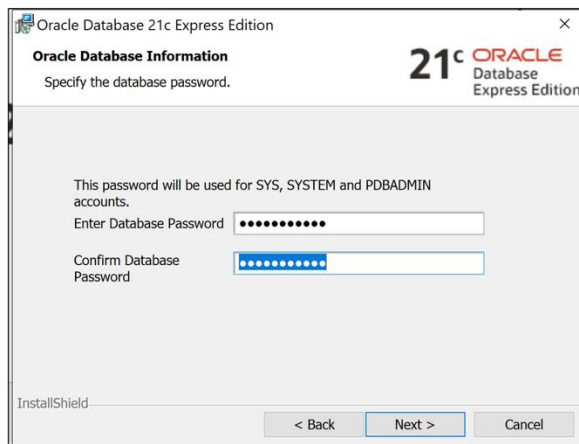
## Local File Location

- ▶ You will be prompted for the location of your local files. Select a location or leave it as default and hit next.



## Password Selection

- ▶ Enter a password for your superuser. You will use this to access the database initially and in case you lose access to other passwords, so write it down. Hit next.



Oracle Database 21c Express Edition

**Oracle Database Information**  
Specify the database password.

21c ORACLE Database Express Edition

This password will be used for SYS, SYSTEM and PDBADMIN accounts.

Enter Database Password

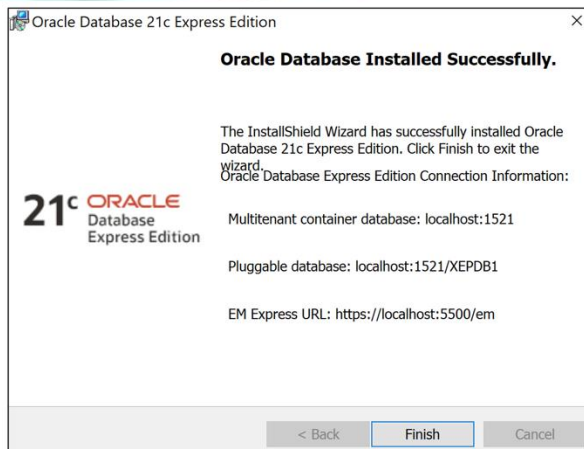
Confirm Database Password

InstallShield

< Back Next > Cancel

## Final Installation

Hit Install and wait for the installation. Once the server is installed, select Finish.



Oracle Database 21c Express Edition

**Oracle Database Installed Successfully.**

The InstallShield Wizard has successfully installed Oracle Database 21c Express Edition. Click Finish to exit the wizard.

Oracle Database Express Edition Connection Information:

Multitenant container database: localhost:1521

Pluggable database: localhost:1521/XEPDB1


EM Express URL: https://localhost:5500/em

21c ORACLE Database Express Edition

< Back Finish Cancel

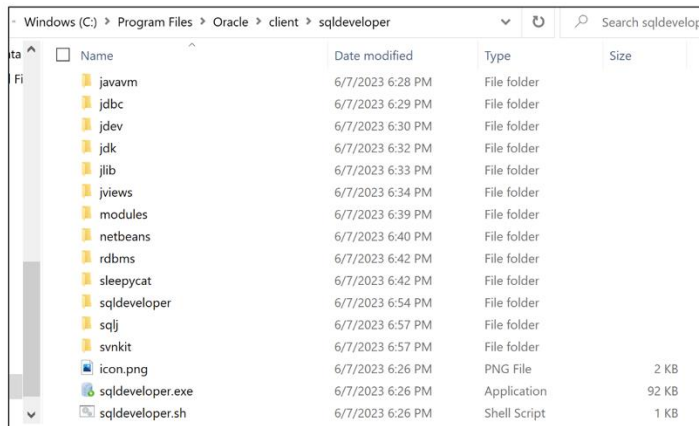
## Client Download

- ▶ At this [link](#), download the most recent version of the Windows client in the basic package.

Platform	Download
Windows 64-bit with JDK 11 included	 Download (459 MB)

## Client Extraction

Once again, extract the downloaded Zip file. The resulting folder should look like this 



Name	Date modified	Type	Size
javavm	6/7/2023 6:28 PM	File folder	
jdbc	6/7/2023 6:29 PM	File folder	
jdev	6/7/2023 6:30 PM	File folder	
jdk	6/7/2023 6:32 PM	File folder	
jiib	6/7/2023 6:33 PM	File folder	
jviews	6/7/2023 6:34 PM	File folder	
modules	6/7/2023 6:39 PM	File folder	
netbeans	6/7/2023 6:40 PM	File folder	
rdbms	6/7/2023 6:42 PM	File folder	
sleepycat	6/7/2023 6:42 PM	File folder	
sqldeveloper	6/7/2023 6:54 PM	File folder	
sqj	6/7/2023 6:57 PM	File folder	
svnkit	6/7/2023 6:57 PM	File folder	
icon.png	6/7/2023 6:26 PM	PNG File	2 KB
sqldeveloper.exe	6/7/2023 6:26 PM	Application	92 KB
sqldeveloper.sh	6/7/2023 6:26 PM	Shell Script	1 KB

## 8.4 Appendix D: Using Oracle on Windows

### Opening Command Prompt

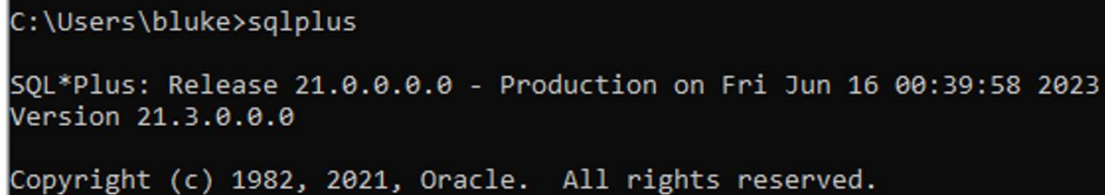
- ▶ Open command prompt by searching for it in the Windows search



```
Command Prompt
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Harrison>
```

### Running `sqlplus`

- ▶ Type in and run the command "`sqlplus`". It should look like this 



```
C:\Users\bluke>sqlplus
SQL*Plus: Release 21.0.0.0.0 - Production on Fri Jun 16 00:39:58 2023
Version 21.3.0.0.0
Copyright (c) 1982, 2021, Oracle. All rights reserved.
```

## Logging into Database

- ▶ You will be prompted for a username. Enter `/ as sysdba` and hit enter. You will now be logged into the `sql` console.

```
Enter user-name: / as sysdba

Connected to:
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
```

## Altering Session

- ▶ Now, enter the command `alter session set "_ORACLE_SCRIPT"=true;` and hit enter.

```
SQL> alter session set "_ORACLE_SCRIPT"=true;

Session altered.

SQL>
```

## Creating User

- ➔ Now enter the command "create user <username> identified by <password>;" and enter.
- ➔ Finally, enter the command "grant all privileges to <username>;" and hit enter.

```
SQL> grant all privileges to yourUsername;
Grant succeeded.
SQL>
```

```
SQL> create user yourUsername identified by yourPassword;
User created.
SQL>
```

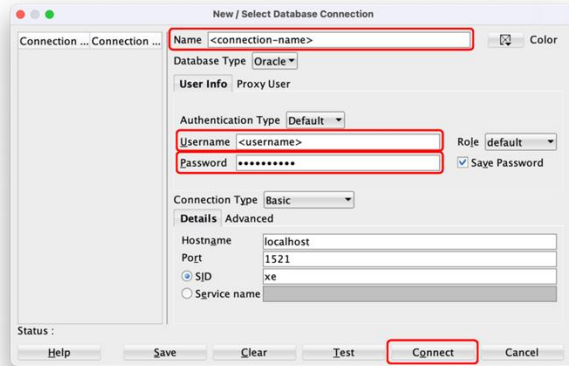
## Client Launch

Run sqldeveloper.exe and select no import preferences if prompted.

Name	Date modified	Type	Size
javavm	6/7/2023 6:28 PM	File folder	
jdbc	6/7/2023 6:29 PM	File folder	
jdev	6/7/2023 6:30 PM	File folder	
jdk	6/7/2023 6:32 PM	File folder	
jlib	6/7/2023 6:33 PM	File folder	
jviews	6/7/2023 6:34 PM	File folder	
modules	6/7/2023 6:39 PM	File folder	
netbeans	6/7/2023 6:40 PM	File folder	
rdbms	6/7/2023 6:42 PM	File folder	
sleepycat	6/7/2023 6:42 PM	File folder	
sqldeveloper	6/7/2023 6:54 PM	File folder	
sqlj	6/7/2023 6:57 PM	File folder	
svnkit	6/7/2023 6:57 PM	File folder	
icon.png	6/7/2023 6:26 PM	PNG File	2 KB
sqldeveloper.exe	6/7/2023 6:26 PM	Application	92 KB
sqldeveloper.sh	6/7/2023 6:26 PM	Shell Script	1 KB

# Client Connection

2. In the pop-up window, fill in the 'Name', 'Username', and 'Password' fields. The Username and Password fields should be the same as what you entered in the command line. Name the connection and hit Connect.



## 8.5 Appendix E: Postgres Installation for macOS

# Installation Process: Step 1

1. Follow the [macOS postgres.app download link](#) which will bring you to a download page that will allow you to install **Postgres.app with PostgreSQL 15(Universal)**. Here you will find various versions of this system including beta versions and previous releases. For this instruction set, we will be using version **15 Universal**. Make sure to press 'Allow' to confirm the download.

Postgres.app with PostgreSQL 15 (Universal)

Postgres.app v2.6.2 · Requires macOS 10.13 · Download Size 100MB  
PostgreSQL 15.3 / PostGIS 3.3.2 · Universal

Download

Do you want to allow downloads on "postgresapp.com"?

You can change which websites can download files in the Websites section of Safari Settings.

Cancel Allow

Installation Process: Step 3

# Installation Process: Steps 2-4

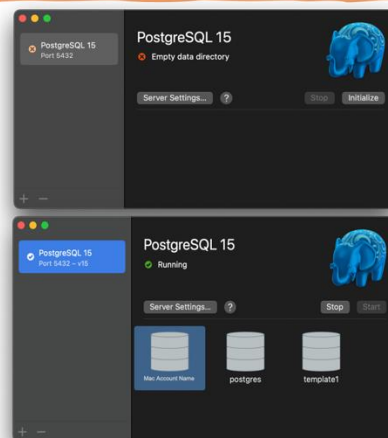
2. Go to your "Downloads" folder and double-click on the downloaded **dmg** file.
3. Then, you can drag the **app** into the Applications folder.
4. Then, you can open the app just as you would any other app (launchpad, spotlight search, finder, etc.)





## Initialization Process: Steps 1-2

1. On the first open, you will see a window like this. Then press on the **Initialize** button. This creates a server named 'PostgreSQL 15' along with three databases that you can attach future servers to.
2. Once the server has been initialized, you will see the second screen. Pay attention to the database names that you see there as they will be useful when creating users.



## Initialization Process: Step 3

Initialization Process: Steps 1-2

3. Now, we need to give tell your computer where Postgres is. To do this, follow the following steps in the following order:
  - a. Open your preferred terminal emulator or Terminal app.
  - b. Enter the command `sudo mkdir -p /etc/paths.d && echo /Applications/Postgres.app/Contents/Versions/latest/bin | sudo tee /etc/paths.d/postgresapp` which will request your mac's password. This creates a folder on all accounts(hence the admin request) and places the path to your Postgres app in that folder, so your computer knows how to access Postgres properly.

## 8.6 Appendix F: Using Postgres on macOS

### Creating Users

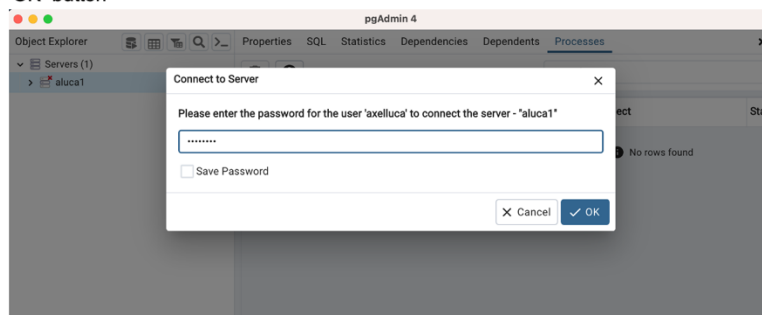
---

1. Enter the command `psql postgres://<database>@localhost:5432/postgres` while replacing **<database>** with any of the database names from the second screen of the Initialization process slide to start the CLI.
2. Following that, enter the command `create user <username> with password '<password>';` to create a user. Replace **<username>** and **<password>** with your WPI username (minus @wpi.edu) and a password that you will remember. If you forget this password, you most likely will need to repeat this step. **This is important as this is the username and password combination that you will use when connecting to the server using an IDE such as pgAdmin 4.**
3. Next, we need to assign some privileges to your user to let it interact with the server. To do that enter the command `grant all on database <database> to <username>;` replacing **<database>** with the same database name used in **Creating Users: Step 1** and **<username>** with the same username used in **Creating Users: Step 2**.
4. If all is working, you can enter `exit;` to leave PostgreSQL mode in your Terminal.


### Connecting to pgAdmin 4: Steps 1-3

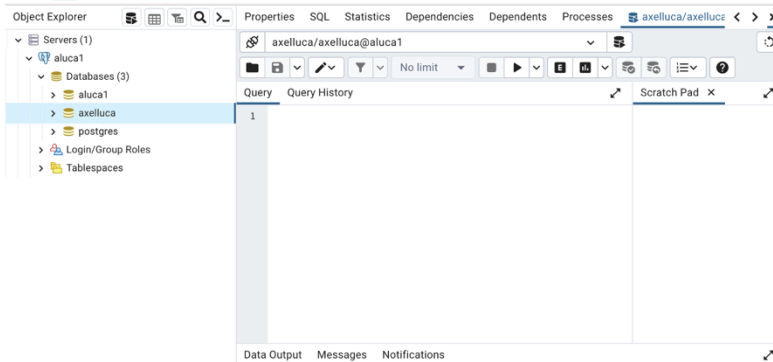
---

1. On your mac, open the "pgAdmin 4" app from your launchpad.
2. If prompted, enter the password of the username you created in Creating Users: Step 2
3. Click the "OK" button



## Connecting to pgAdmin 4: Steps 4-5

4. Click "Servers", and select any of the database names that were identified in Initialization Process:  
Step 2
5. Click on the  Button to get access to an SQL worksheet.



## 8.7 Appendix G: Postgres Installation for Windows

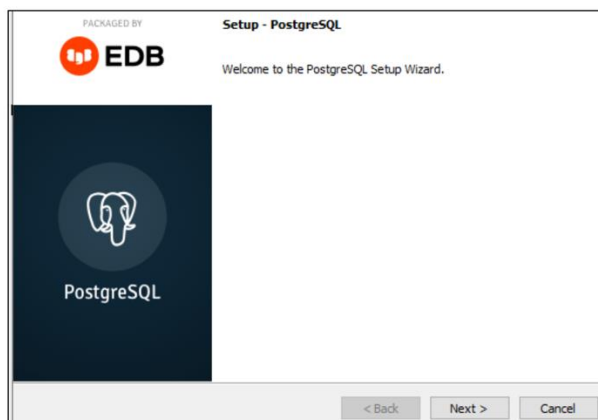
### Initial download

- ▶ At this [link](#), download the most recent express edition for Windows by clicking the button under Windows.

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
15.3	<a href="https://www.postgresql.org">postgresql.org</a>	<a href="https://www.postgresql.org">postgresql.org</a>			Not supported

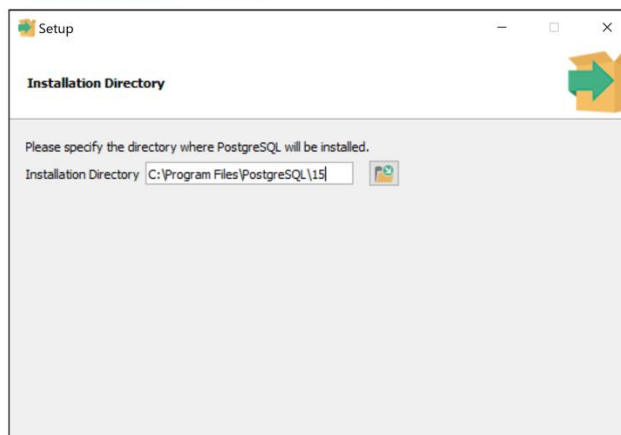
### Starting the Installer

- ▶ Run the executable that was downloaded, allowing it to make changes to your device if prompted. The installer will launch, hit next.



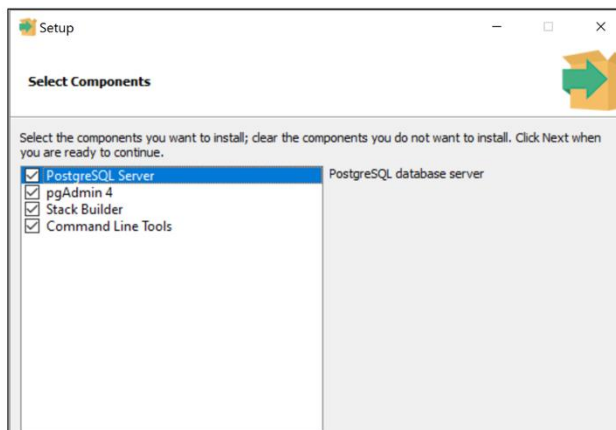
## Local File Location

- ▶ You will be prompted for the location of your local files. Select a location or leave it as default and hit next.



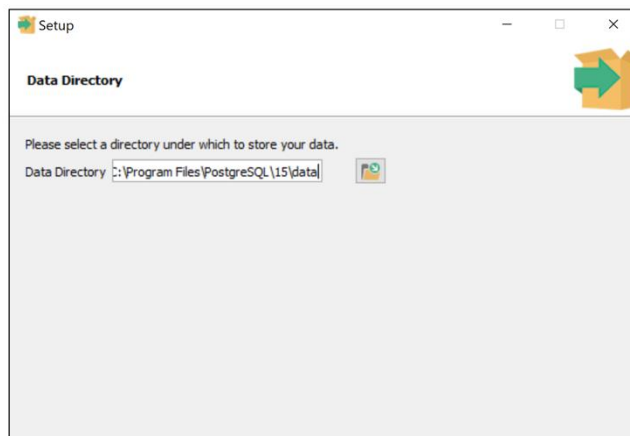
## Installing Additional Components

- ▶ Leave all components checked off and hit next.



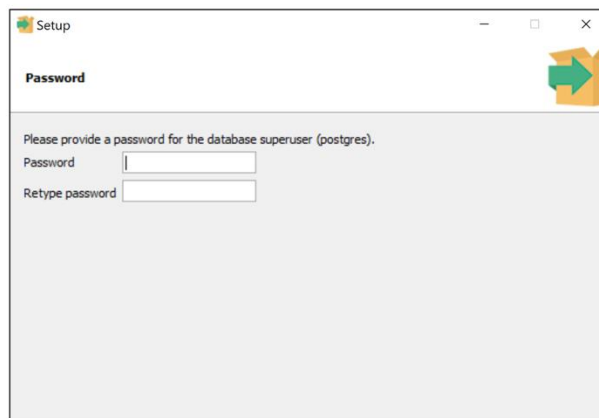
## Data Location

- ▶ It will now prompt you for a location for the storage of the data. Specify a location or leave it default and hit next.



## Password Selection

- ▶ Enter a password for your superuser. You will use this to access the database initially and in case you lose access to other passwords, so write it down. Hit next.



## Locale and Port Selection

When prompted for a locale, leave it default and hit next. Then leave the port default as well and hit next.

**Advanced Options**

Select the locale to be used by the new database cluster.

Locale

**Port**

Please select the port number the server should listen on.

Port

## Final Installation

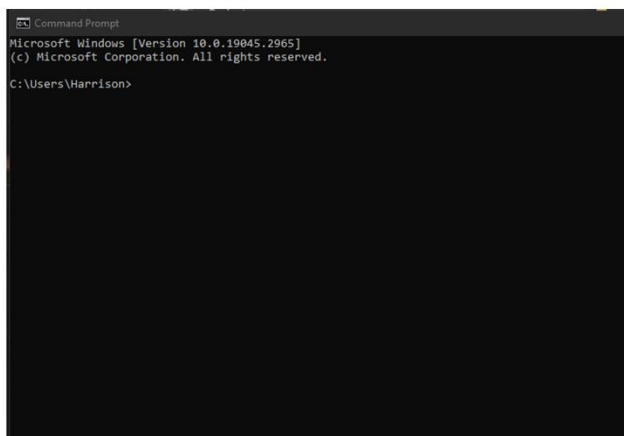
- ▶ Hit next and begin the installation. It will be fully downloaded within a few minutes and ready to go.



## 8.8 Appendix H: Using Postgres on Windows

### Opening Command Prompt

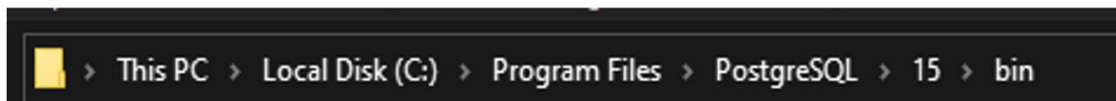
- ▶ Open Command Prompt by searching for it in the Windows search bar



```
Command Prompt
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.
C:\Users\Harrison>
```

### Locating the File Path

- ▶ Locate the file path to the bin folder in your installed Postgres server. The default path is shown here.



You can copy the file path, if necessary, by Control+Shift+Right-clicking the bin folder and selecting "Copy as path"



## Navigating to bin

- ▶ In your command prompt, type "cd " and then enter the pathway to the bin folder, as shown here.

```
C:\Users\Harrison>cd C:\Program Files\PostgreSQL\15\bin
```

## Logging into Database

- ▶ Now, enter the command "psql -U postgres" which will log you into the admin user that was created during database installation. Enter the password you set and hit enter.

```
C:\Program Files\PostgreSQL\15\bin>psql -U postgres
Password for user postgres:
psql (15.3)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

## Creating User

- ➔ Now that you are logged into the database, create your user and password by entering the command "create user <username> with password '<password>';"

```
postgres=# create user yourUsername with password 'database';  
CREATE ROLE
```

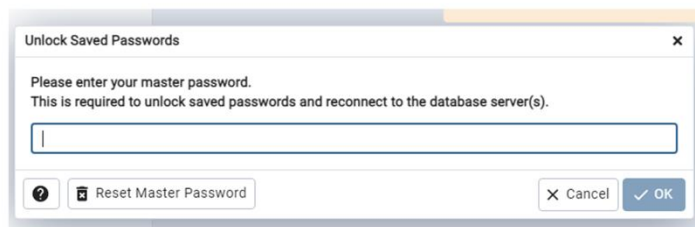
## Creating Personal Data Base and Permissions

- ➔ Create a database and assign yourself roles with the following commands: "create database <databaseName>;"
- ➔ Finally, run "grant all on <databaseName> to <username>;" You now have a user on your new database!

```
postgres=# create database myDatabase;  
CREATE DATABASE  
postgres=# grant all on database MyDatabase to yourUsername;  
GRANT  
postgres=#
```

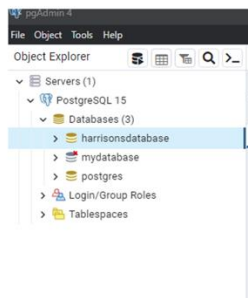
## Logging into pgAdmin

- ➔ Launch pgAdmin and enter the master password you created during the installation



## Accessing Database

- ➔ Use the dropdown to access your server that you created, and if prompted for a password, enter the password you created in the command line



## 8.9 Appendix I: MySQL Installation for MacOS

# Helpful links

[Supported Platforms: MySQL Database](#)

[Find out which version of macOS you are using](#)

[Find out which architecture your macOS is using](#)

# Download Options

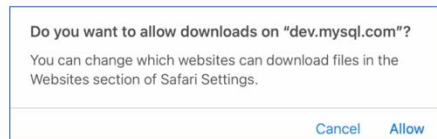
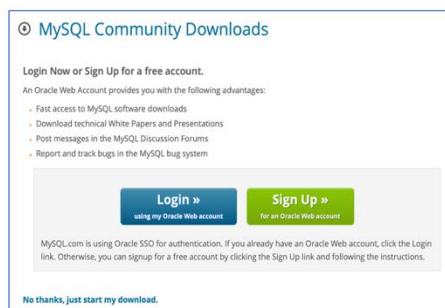
You can download MySQL as a DMG which is the native macOS installer [here](#).

If you would like to download the raw binaries as a compressed file you can see the steps required [here](#). This requires Terminal commands, so it is less ideal if you do not feel comfortable with that.

The full documentation regarding installing MySQL on macOS can be found [here](#). This includes directions for using the [MySQL Launch Daemon](#) (which is automatically set up using the DMG above) and the [MySQL Preference Pane](#).

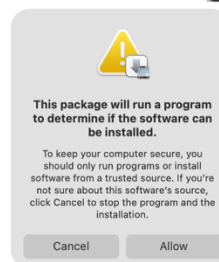
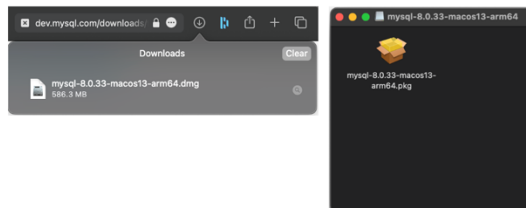
# Downloading the .dmg

1. You should figure out which [version](#) of macOS you are using and which [architecture](#) you are using to download the proper file at the [download page](#).
2. After selecting the proper version for your computer, you will be prompted to login or signup. This isn't a bad idea just to be able to save information regarding your MySQL account. But this step is not required. You can select '**No thanks, just start my download.**'
3. You may see a prompt asking for your trust, to download files. You should press '**Allow**' to continue.



## Installing the .dmg

4. After the download finishes you can double-click on it from your browser's downloads page or from Finder.
5. This will open a new window that shows a package that you can double-click to open.
6. You may not be allowed to download this due to the source of the download. You can ignore this and try again. The program should install on the second or third try by pressing 'Allow.'



## Installing MySQL(using the .dmg)

7. Follow the installer's instructions following these steps:
  - a. Press '**Continue**' through the Introduction page and the License page, making sure to press '**Agree**' to continue.
 

You can specify a custom location by pressing '**Change Install Location...**' but normally leaving it as default is best.
  - b. Press '**Continue**' and confirm the installation and allow it to install the program.
  - c. You can now select '**Use Strong Password Encryption**' before pressing '**Next**'. Most importantly, the installer will prompt you to enter a password for the "root" user. This is something that you should not forget as it will be complicated to fix that.
 

In the real world, this is kept hidden and never shared with anyone. Enter a password that makes sense for your situation.
  - d. Now, make sure to select '**Start MySQL Server once the installation is complete**' after pressing '**Finish**'. After waiting for it to finish, press '**Close**' to exit the installer.

For a more detailed documentation of this process please check out the [MySQL docs](#).

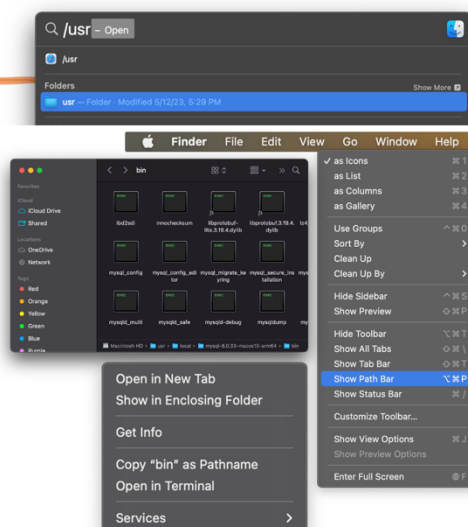
After searching in your apps, you should see one called '**MySQL Workbench**.' This will be your primary interface for interacting with your MySQL databases.



## 8.10 Appendix J: Using MySQL on MacOS

## Creating Users: Step 1

1. Now, hit both the **Space** and **⌘** keys at the same time to activate Spotlight Search. From here type `/usr` and navigate to folder that shows up.
2. With Finder open, click on the **'View'** tab in the Menu Bar, followed by the **'Show Path Bar'** button.
3. Back in your `/usr` folder, navigate to `local/<your MySQL version>/bin`. From here, right click on the `bin` icon at the bottom Path Bar. From the menu that appears, select **'Open in Terminal'**.

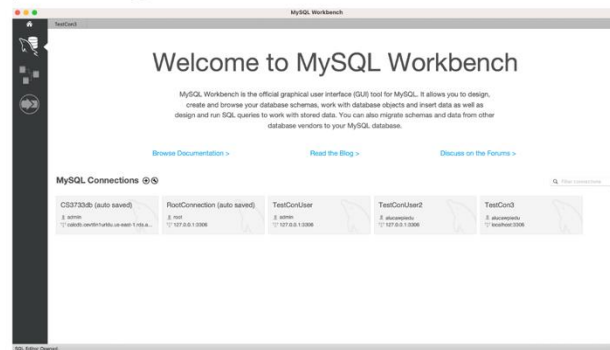


## Creating Users: Steps 4-7

4. From inside the Terminal app, run the command `./mysql -u root -p` and enter the original root password from setting up the MySQL Server.
5. Then, enter `create user '<username>' identified by '<password>';`. Replace `<username>` with your WPI username (minus `@wpi.edu`) and `<password>` with a password that you will remember, as if you forget, you will need to redo most of these steps. **This is important as this is the username and password combination that you will use when connecting to the server using an IDE such as MySQL.**
6. Next, we need to assign this user privileges which allow the user to interact with the server. To do this, enter the command `grant all privileges on *.* to '<username>';` replacing `<username>` with the username from **Creating Users: Step 5**.
7. After confirming that this is working, enter `exit;` to leave SQL mode in your Terminal.

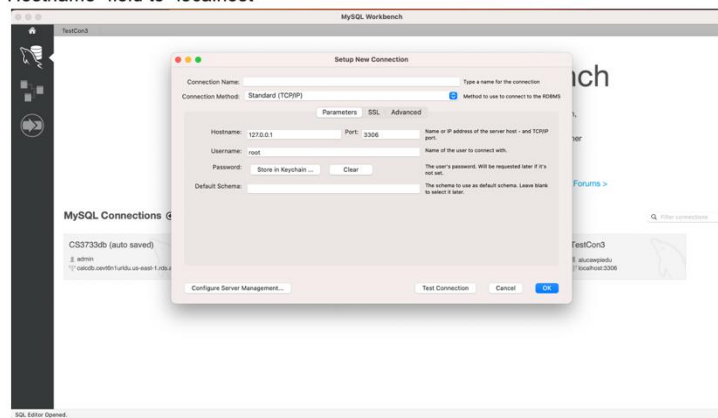
## Connecting to MySQL Workbench: Steps 1-2

1. On your mac, open the "MySQLWorkbench" app from your launchpad.
2. Once the app opens, click on the **+** button to start a new connection



## Connecting to MySQL Workbench: Steps 3-4

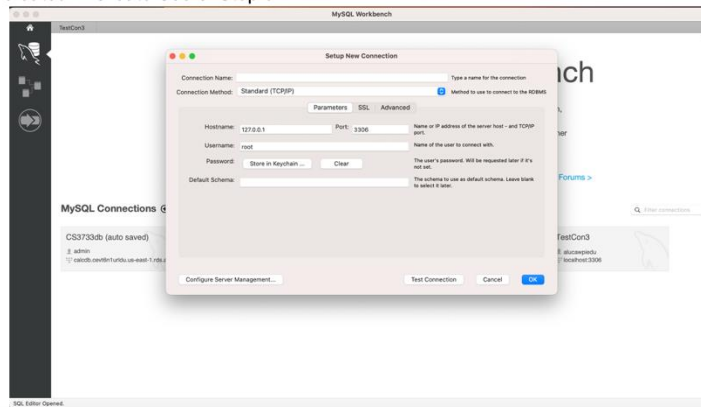
3. On the page that appears, give your new connection a name without spaces
4. Change the "Hostname" field to "localhost"





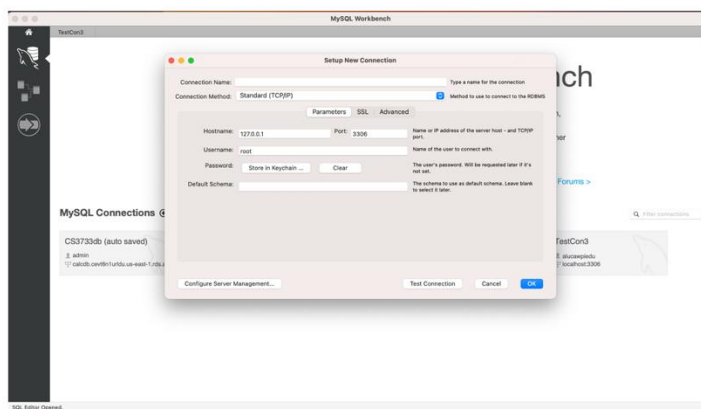
## Connecting to MySQL Workbench: Steps 5-6

5. Change the "Username" field to the username you created in Create Users: Step 5
6. To enter your password, first click on the "Store in Keychain" button, then enter the password you created in Create Users: Step 5



## Connecting to MySQL Workbench: Step 7

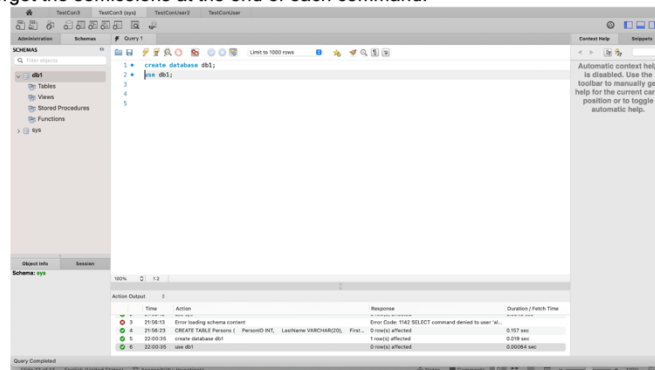
7. Once all the details have been entered, click on the "OK" button





## Connecting to MySQL Workbench: Step 8

8. After the window from **Connecting to MySQL**, you should see a new connection appear in the Connections tab, as well as a new Worksheet tab. To create and use a new database schema, simply type the commands:
- “create database <name>,” and “use <name>.” In here, <name> can be anything consisting of letters and numbers only. Do not forget the semicolons at the end of each command.



## 8.11 Appendix K: MySQL Installation for Windows

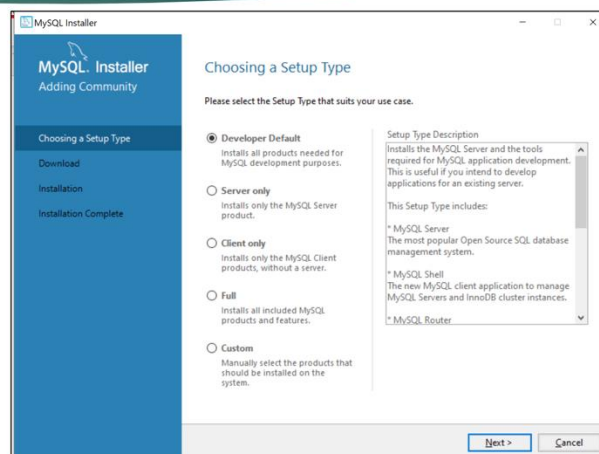
### Installer download

- ▶ At this [link](#), download the first installer. If prompted to sign into Oracle, select No and continue the download. Launch the file that is downloaded.

<b>Windows (x86, 32-bit), MSI Installer</b>	8.0.33	2.4M	<a href="#">Download</a>
(mysql-installer-web-community-8.0.33.0.msi)			MD5: 2a330cf24915964cca87e04dbb34e5d3   <a href="#">Signature</a>

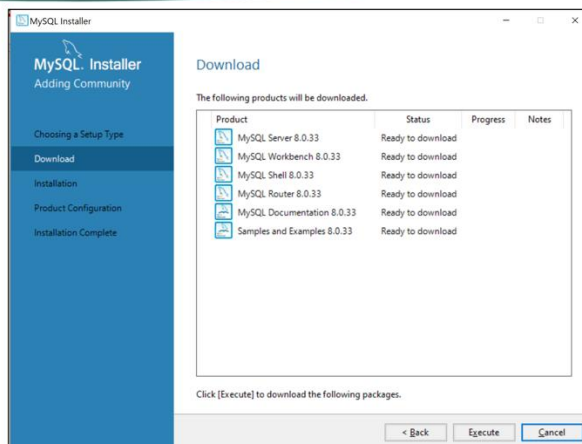
### Initial Installer Setup

- ▶ You will now be in the installer. Select Developer Default and hit next.



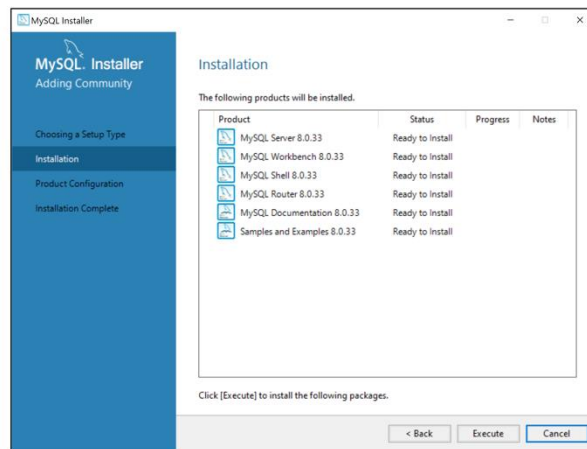
# Download

- ▶ You will be prompted with a download, execute it and hit next.



# Installation

- ▶ You will now be prompted with the installation, execute it and hit next.



# Networking Configuration

- You will be brought to a configuration screen. Hit next to continue to the networking configuration, and then hit next on that to continue, leaving all fields default

The screenshot shows the 'MySQL Installer' window for 'MySQL Server 8.0.33'. The left sidebar has 'Type and Networking' selected. The main area is titled 'Type and Networking' and contains the following sections:

- Server Configuration Type:** Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance. Config Type:
- Connectivity:** Use the following controls to select how you would like to connect to this server.
  - TCP/IP Port:  X Protocol Port:
  - Open Windows Firewall ports for network access
  - Named Pipe Pipe Name:
  - Shared Memory Memory Name:
- Advanced Configuration:** Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.
  - Show Advanced and Logging Options

Buttons at the bottom: Next > and Cancel

# Password

- It will prompt you for a security option, select strong password. Then, enter and repeat your password, and write it down as it will be needed for database access. Hit next.

The screenshot shows the 'MySQL Installer' window for 'MySQL Server 8.0.33'. The left sidebar has 'Accounts and Roles' selected. The main area is titled 'Accounts and Roles' and contains the following sections:

- Root Account Password:** Enter the password for the root account. Please remember to store this password in a secure place.
  - MySQL Root Password:
  - Repeat Password:
  - Password strength: **Weak**
- MySQL User Accounts:** Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.
 

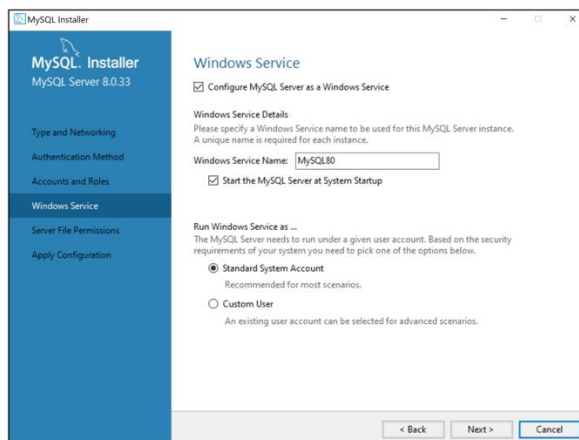
MySQL User Name	Host	User Role

  - Buttons: Add User, Edit User, Delete

Buttons at the bottom: < Back, Next >, Cancel

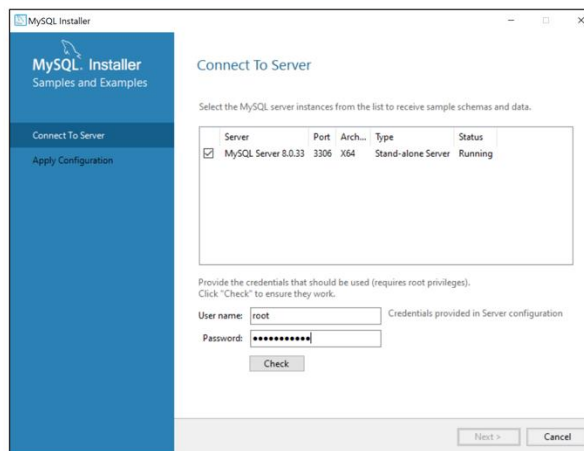
## Windows Service

- ▶ Leave this page default and hit next



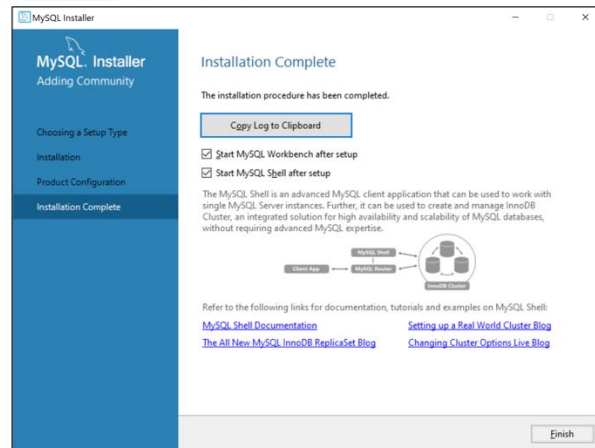
## Password Configuration

- ▶ You'll be brought back to the configuration screen, hit next again. You will then enter the password you created earlier, check it, and hit next



# Final Setup

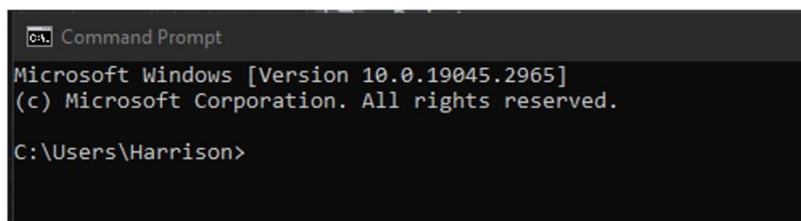
Hit next on the configurations and finally hit finish.



## 8.12 Appendix L: Using MySQL on Windows

### Opening command prompt

- ▶ Open Command Prompt by searching for it in the Windows search bar

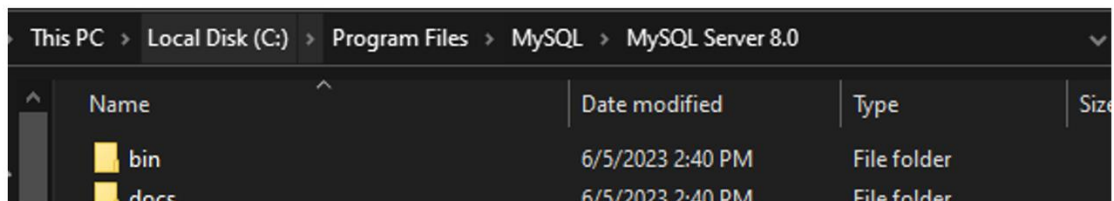
A screenshot of the Windows Command Prompt window. The title bar reads "CA. Command Prompt". The main text area shows the following text:

```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Harrison>
```

### Finding the File Path

- ▶ Locate the file path to the bin folder in your installed MySQL server. The default path is show here. You can copy the file path, if necessary, by Control+Shift+Right-clicking the bin folder and selecting "Copy as path"

A screenshot of a Windows File Explorer window showing the path: This PC > Local Disk (C:) > Program Files > MySQL > MySQL Server 8.0. The main area shows a table of files and folders.

Name	Date modified	Type	Size
bin	6/5/2023 2:40 PM	File folder	
docs	6/5/2023 2:40 PM	File folder	

## Navigating to the Server

- ▶ In your command prompt, type "cd " and then enter the pathway to the bin folder, as shown here 

```
C:\Users\Harrison>cd C:/Program Files/MySQL/MySQL Server 8.0/bin  
C:\Program Files\MySQL\MySQL Server 8.0\bin>
```

## Logging into Database

- ▶ Now enter the command "mysql.exe -u root -p" and then enter the password you created during installation

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql.exe -u root -p  
Enter password: *****
```



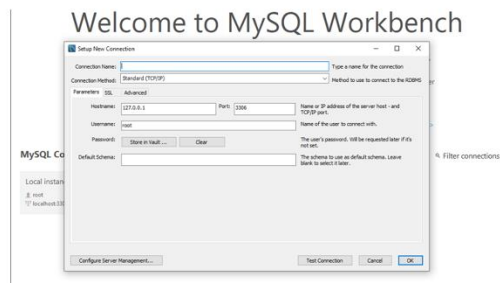
## Creating a User

- ▶ Now enter the following commands to create and allow your user to access the database: "create user '<username>' identified by '<password>';" and "grant all privileges on \*.\* to '<username>';" Your new user will now have access to the database!

```
mysql> create user 'username' identified by 'password';  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> grant all privileges on *.* to 'username';  
Query OK, 0 rows affected (0.02 sec)  
  
mysql>
```

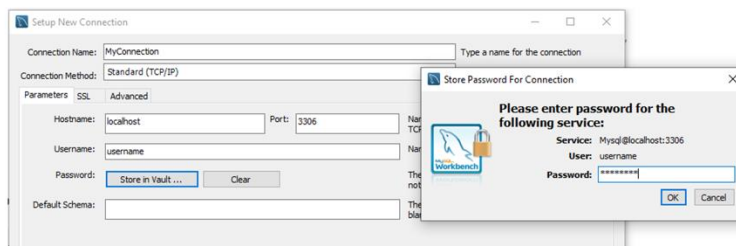
## Logging Into MySQL Workbench

- ▶ Open MySQL Workbench, and hit the plus button to start a new connection



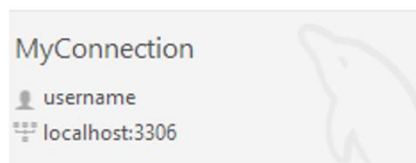
## Logging Into MySQL Workbench

- ▶ Enter a name for your connection, and then enter "localhost" for the Hostname, the username you set in the command line for the Username field, and select "Store in Vault" and enter your password from the command line



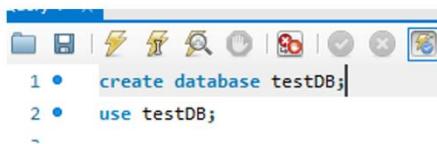
## Connection Successful

- ▶ Hit Test Connection, and if it says the connection is successful, apply your connection. If everything went well it should look like this. If the connection was not successful, recheck the data you entered. Click your new connection!



# Database Creation

- ▶ Create and use a database by running these two lines of code in the query that just opened. Run them with the Lightning shaped buttons at the top. Now select schemas instead of Administration in the bottom left



```
1 • create database testDB;
2 • use testDB;
```

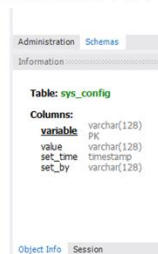


Table:	sys_config
Columns:	
variable	varchar(128)
value	PK varchar(128)
set_time	timestamp
set_by	varchar(128)