Articles

Construction of an Integrated and Scalable Database Platform for Antimicrobial-Resistant Strains

DOI:https://dx.doi.org/10.71373/VTXZ5661
Submitted 16 October2025
Accepted 3 December 2025
Pulished 5 December 2025

Xuanyu Pan[1], Liying Yang[2], Yuxiao He[2], Jianping Zhou[2], Yongjian Zhou[2], Tingwei Lyu[2], Xingyao Li[3], Yani Yang[3],Yanling Hu[1,2,*]

Background:Antimicrobial resistance (AMR) poses an escalating global health threat, yet existing AMR databases remain fragmented, inconsistently updated, and insufficiently curated at the variant level. Most public tools (AMRFinderPlus, RGI and ResFinder) primarily rely on sequence homology and provide limited support for experimentally validated chromosomal mutations. This gap restricts precise interpretation of resistance mechanisms and hinders downstream computational modelling.Methods:We constructed an integrated and scalable AMR strain database (OurDB) by collecting multi-source data from CARD, CNCB, NCBI and peer-reviewed literature. A fully normalized relational schema (1NF–3NF) was designed based on seven core entities, including organisms, proteins, drugs, drug classes, resistance mechanisms, variant genes and supporting literature. Data processing and import were implemented using Python and MySQL, and a web platform was developed using Flask, HTML/CSS/JavaScript and Layui to support real-time querying, visualization and data export. External benchmarking was performed using AMRFinderPlus, RGI and ResFinder on the *Klebsiella pneumoniae* HS11286 reference genome.Results:The completed system provides fast data retrieval (82 ms for single-condition queries; 131 ms for multi-condition queries; <1 s for full-table export), interactive visual analytics and comprehensive variant-level curation. Benchmarking revealed that classical acquired resistance genes were consistently detected by all public tools, whereas high-confidence chromosomal regulators and mutation-dependent determinants (e.g., *acrR, ramR, mgrB, PmrB, gyrA* substitutions and *erm(42)*) were uniquely contributed by OurDB and largely absent from existing resources. Presence–absence matrices and antibiotic-class coverage analyses further demonstrated the complementary value of OurDB in underrepresented resistance classes such as polymyxins and macrolides.Conclusion:OurDB integrates curated variant-level AMR knowledge with a scalable web platform, providing a unified environment for AMR data management, interrogation and cross-tool comparison. The database complements existing AMR callers by supplying experimentally validated mutation annotations and supporting downstream research in resistance evolution, surveillance and computational prediction. Future development will focus on automated data ingestion, phenotype-linked modelling and AI-driven resistance prediction.

1.Introduction

1.1Research Background and Significance

With the widespread use of antibiotics, the emergence and spread of AMR bacteria have become a major challenge to global public health. The presence of these AMR strains has made many infections that were once easily treatable difficult to control. Investigations have revealed that in recent years, an increasing number of pathogens have developed resistance to one or more antibiotics, with the emergence of superbugs. The emergence of AMR strains such as methicillin-resistant Staphylococcus aureus (MRSA) and vancomycin-resistant Enterococcus faecalis has become one of the causes threatening public health[1]. Reports indicate that by 2050, infections caused by AMR bacteria are projected to surpass cancer and heart disease as the leading cause of death[2]. Furthermore, according to World Health Organization (WHO) statistics, bacterial resistance already causes 700,000 deaths globally each year. By 2050, deaths from antibiotic-resistant infections are projected to rise to 10 million worldwide[3].

Following the launch of the WHO Global Action Plan on Antimicrobial Resistance in 2015, articles assessing AMR risks both domestically and internationally have grown exponentially. Risk assessments of AMR in relevant media have shown a similar growth trend. China's AMR situation remains equally severe. However, since joining the WHO in 2001, China has also initiated nationwide AMR governance actions based on WHO recommendations[4].

Due to multiple factors including antibiotic overuse, natural icrobial mutation, and interactions among AMR strains, pathogen resistance has become increasingly severe[5]. In microbiological mechanism studies, bacterial resistance mechanisms primarily encompass target alteration-induced resistance, efflux pump-mediated resistance, antibiotic degradation enzyme-inducedstr-uctural disruption, and adaptive evolutionary mechanisms such as biofilm formation[6],These complex, interacting resistanceme-chanisms pose significant challenges to clinical antimicrobial therapy. Numerous studies demonstrate that establishing comp-rehensive databases of AMR bacteria is crucial for understanding the evolution of resistance and guiding clinical prescribing practices. For example, researchers including Chen successfully predicted the spread of carbapenem-resistant Klebsiella pneumo-niae by integrating genomic and clinical data[7].

Today, bacterial resistance transcends medical boundaries,evol-ving into a complex systemic issue spanning clinical medicine,mic-robiology, drug development, and public health policy. However, antimicrobial resistance research faces a core contradiction: First, bacterial resistance mechanisms exhibit rapid evolution and dive-rsification; second, existing data resources are fragmented and lack

1.School of Basic Medical Sciences, Guangxi Medical University,530021, Nanning, Guangxi, China. 2.Institute of Life Sciences, Guangxi Medical U niversity, 530021,Nanning, Guangxi, China. 3.Guangxi Medical University School of Information and Management, Nanning, Guangxi, China.

**Corresponding author:*
Yanling Hu, Email: ylhupost@163.com

systematic integration. This data fragmentation severely constrains researchers' comprehensive understanding of resistance transmission patterns and limits the application potential of new technologies like artificial intelligence in resistance prediction.

It should be noted that from a technological evolution perspective, the widespread adoption of next-generation sequencing technologies has generated massive microbial genomic data, while cloud computing platforms provide robust data processing capabilities. These developments have created unprecedented technical conditions for constructing next-generation AMR bacteria databases. However, transforming these technological advantages into practical research tools remains an urgent challenge. Particularly in antibiotic-heavy users like China, establishinglocalized antimicrobial resistance databases holds not only scientific value but also significant strategic importance.

This study focuses on this cutting-edge technological frontier, aiming to overcome the limitations of traditional database construction. The research scope encompasses not only technical challenges in data collection and storage but also emphasizes exploring innovative pathways for deep data mining and knowledge discovery. By building an intelligent, scalable database platform, we expect to provide new research paradigms and tool support for tackling the global challenge of antibiotic resistance.

1.2Current Research Status and Development at Home and Abroad

Globally,antimicrobial resistance across various bacterial strainshas garnered significant attention. Multiple countries and regions have initiated surveillance and research programs, such as the U.S. National Antimicrobial Resistance Monitoring System(*NARMS*), the European Antimicrobial Resistance Surveillance Network (*EARS-Net*), and the European Surveillance of Veterinary Antimicrobial Resistance (*ESVAC*). Significant progress has been made in international academic circles regarding the construction of resistance databases. For example, the CARD database developed by McMaster University in Canada provides a vital resource for studying the molecular mechanisms of resistance; the *AMRFinderPlus* tool from the U.S. NCBI enables automated annotation of resistance genes in bacterial genomes[8]. Denmark's *DANMAP* resistance surveillance system has continuously conducted drug susceptibility testing for Salmonella and Campylobacter to coordinate EU resistance monitoring efforts[9].

Domestic research on antimicrobial resistance has been intensified, with multiple research teams dedicated to investigating the genomics, resistance mechanisms, and transmission pathways of AMR bacteria. In 2005, the China Antimicrobial Resistance Surveillance System (CARSS), jointly established by the former Ministry of Health, National Administration of Traditional Chinese Medicine, and the Ministry of Health of the PLA General Logistics Department, began collecting nationwide data on the susceptibility and resistance rates of various clinical common pathogenic bacteria to different antimicrobial agents[10]. However, existing databases still exhibit shortcomings in terms of data update frequency and the sophistication of analytical tools.

Meanwhile, the integration of multi-omics data and the application of artificial intelligence technologies are reshaping the research paradigm of AMR. These technological advancements provide an important reference for the present study.

1.3Objectives and Scope of This Study

Currently, there is a lack of a comprehensive and systematic database of AMR bacteria capable of supporting big data analysis and model construction. Building a comprehensive and systematically structured database of AMR bacteria is crucial for dynamically monitoring the evolution of resistance trends, guiding precise clinical drug use decisions, and conducting indepth research on resistance mechanisms.

This study aims to construct a novel AMR bacteria database system with the following characteristics:

(1) Multi-source data integration: Collect strain information from domestic and international databases such as *CNCB* and *CARD* to the greatest extent possible, while integrating relevant AMR bacteria data from the latest published literature worldwide.

(2) Data Content Collection: Gather useful data including strain types, associated proteins, antimicrobial substances, resistance mechanisms, variant genes, and data sources.

A database was established using collected data, employing the MySQL database management system for data storage. Python's pandas library was utilized to read data from Excel files, which were then written into the database via MySQL's bulk insertion functionality. After database creation, the website was designed using Python's Flask framework. The frontend pages were developed with HTML, CSS, and JavaScript, incorporating the layui framework to achieve an aesthetically pleasing and responsive interface. The website's backend connects to the database. When the frontend initiates a data query request, the backend processes the request, retrieves data from the database, and returns it to the frontend in JSON format. The frontend uses JavaScript to parse the JSON data and display it on the page, thereby presenting the database content on the frontend and establishing the database website.

2.Design Analysis and Implementation of the AMR bacteria Database

2.1Design Paradigms for the AMR bacteria Database

During the design of the AMR bacteria database, we strictly adhered to database design normalization principles to ensure data structure, standardization, and consistency. This primarily encompassed the following:

(1) First Normal Form (1NF)

This paradigm requires each column in a database table to contain indivisible atomic data items, and each row record to possess a unique identifier. In implementation, we meticulously decomposed raw data. For example, the composite field "strain name-resistance" at was split into independent "strain name" and "resistance" fields, ensuring each field stores only a single type of information.This design ensures that data items like strain names, resistance characteristics, and mutation genes exist as distinct columns. This approach avoids the issue of multiple pieces of information being combined and stored within a single cell while enhancing the standardization of data storage.

(2) Second Normal Form (2NF)

Building upon the foundation of first normal form, second normal form (2NF) requires that non-key attributes in database tables depend fully on the primary key, rather than partially. To achieve this goal, this study normalized the original data model. Taking the AMR bacteria database as an example, the initial design stored strain names, proteins, and AMR substances in a single data table to avoid redundant storage. Through in-depth data relationship analysis, it was discovered that drug categories and specific drugs share a many-to-many relationship, and this category information is independent of other attributes in the main table. Based on this analysis, a table partitioning decision was implemented. An independent "drug_class" table was created to separate drug category information, establishing a relationship with the main table "drug_resistant_bacteria_data" via drug ID. This design ensures every non-primary attribute fully depends on the primary key, effectively eliminating data redundancy. For instance, when multiple microorganisms develop resistance to the same drug class, the drug class information is stored only once in the "drug_class" table. The main table references it via foreign keys, significantly reducing wasted storage space while enhancing data update consistency.

(3) Third Normal Form (3NF)

The Third Normal Form (3NF) builds upon the Second Normal Form by further requiring that no transitive dependencies exist between non-primary key attributes. In the design of the AMR bacteria database, we identified that the original "Resistance-Mechanism" field might depend on a combination of micr-organism and drug entities, rather than directly on the primary key. This observation indicated a potential transitive dependency violation of 3NF principles.To eliminate this transitive dependency, a dedicated "resistance_mechanism" table was created to isolate resistance mechanism information. Furthermore,an associative table named "organism_drug_relationship" was implemented to establish a many-to-many relationship among microorganisms, drugs, and resistance mechanisms.This normalized design ensures that each non-primary key attribute directly depends on the primary key, rather than being indirectly dependent through other non-primary attributes. As illustrated in this example, when updating the resistance mechanism of a specific microorganism against a particular drug class,modifications are confined to the relevant records within the associative table. This approach effectively prevents unintended impacts on unrelated data.The implementation of 3NF not only significantly reduces data redundancy but also enhances query efficiency and improves database maintainability. Consequently, the system achieves greater flexibility in adapting to evolving business requirements and complex data relationships.

2.2Database Technology Selection

2.2.1Database Management System Selection

This system selects MySQL as its database management system. MySQL is an open-source, widely adopted relational database management system offering advantages such as stable perf-ormance, ease of use, and low cost[11]. From its open-source nature to its performance capabilities, and from community support to its application ecosystem, every aspect aligns with project requirements, ensuring stable database operation and efficient management.

MySQL's open-source nature is one of its core competitive advantages. Users avoid costly commercial database licensing fees, enabling zero-cost access to a high-performance database man-agement system that significantly reduces initial project investment. Simultaneously,its open-source nature allows users to modify and customize the database source code according to their specific needs, adapting it to unique business logic and functional requirements.

In terms of processing performance, MySQL excels at efficiently handling large-scale data storage and query operations. Its built-in query optimizer intelligently parses and optimizes SQL statements. When executing SQL queries for network information retrieval, it employs various execution strategies—including adjustments to CPU utilization, I/O communication wait times, and network data transmission methods—to effectively reduce query costs[12]. Whether performing single-table rapid queries or multi-table join analysis, MySQL delivers stable and efficient performance, ensuring swift retrieval of required information when processing AMR bacteria data.

MySQL also offers a user-friendly interface and comprehensive management tools, lowering the barriers to use and administration. Through command-line interfaces or graphical management tools, users can easily perform database creation, table structure design, data import/export, and user permission management. Even without deep database technical expertise, users can quickly get started and conveniently manage and maintain data.

As a mature database management system, MySQL is widely deployed globally across diverse projects spanning web applications, enterprise systems, data analytics, and more. Its excellent compatibility enables seamless integration with multiple operating systems (e.g., Windows, Linux, macOS) and development languages (e.g., Python, Java, PHP). In this research project, Python was selected for data processing and website development. MySQL can establish database connections with numerous Python database operation libraries (such as pymysql and mysql-connector-python), ensuring smooth data interaction and efficient processing.

MySQL boasts a vast and active community comprising database experts, developers, and users from around the world. When encountering issues during use, users can access extensive technical documentation, solutions, and experience-sharing through community forums, technical blogs, mailing lists, and other channels. This robust community support not only enables rapid resolution of technical challenges but also keeps users informed about MySQL's latest developments and technological trends, providing strong backing for continuous project optimization and upgrades.

For complex entity relationships within AMR bacteria datasets—such as strain-drug resistance associations and literature-experimental data correlations—MySQL, as a relational database, establishes inter-table relationships through mechanisms like foreign key constraints. This ensures data integrity and consistency. This structured management model facilitates multidimensional join queries and in-depth data analysis, meeting the stringent reliability requirements of scientific research and clinical scenarios.

2.2.2Database Programming Language Selection

In terms of programming language, this study employs Pythonfor database operations and data processing. Python is an object-oriented, open-source programming language. Compared to other languages such as C and C++, Python features a relatively simple syntactic structure. Furthermore, since Python was developed based on the groundwork laid by Guido van Rossum, it possesses a wide variety of libraries and APIs[13]. Concurrently, compared to other programming languages, Python exhibits significant advantages in cross-platform compatibility and open-source nature. When applied to web programs, these advantages are further amplified[14]. Notably, Python boasts a rich ecosystem of third-party libraries. For instance, pandas is utilized for data processing and analysis, making operations such as data reading, cleansing, and importing into databases more convenient and efficient. Moreover, by encapsulating database operation methods within a custom MySqlDB class, the maintainability and reusability of the code are enhanced.

Python boasts numerous powerful third-party libraries like pandas, numpy, scikit-learn,and others, which excel in data processing and analysis.This study utilizes the pandas library to handle structured data. In the AMR bacteria data project, it efficiently reads, cleans, and transforms data from formats like Excel, facilitating database import and preprocessing.This provides technical support for analyzing the spread trends and resistance mechanisms of AMR bacteria.

Python's concise and clear syntax ensures high code readability, significantly reducing development and maintenance costs. For complex database operations and website feature development, Python can achieve the same functionality with less code, shortening the development cycle. During the construction of the AMR bacteria database and website, Python enables faster code writing and debugging, allowing more focus on feature implementation and business logic optimization. Furthermore, the conciseness of Python code facilitates code reviews and future feature expansions.

Website design relies on web frameworks, and Python offers several excellent options. This project selected the Flask framework for website design due to its lightweight and flexible nature, making it ideal for rapidly building feature-rich web applications. It provides core functionalities like a basic routing system and a template engine, meeting requirements for data querying and visualization. Furthermore, Flask's strong extensibility allows flexible selection and integration of plugins and libraries based on project requirements, enabling the construction of a user-friendly website for displaying AMR bacteria data.
Furthermore, Python boasts outstanding cross-platform capabilities, ensuring stable operation across Windows, Linux, macOS, and other operating systems. This means the AMR bacteria database and website developed in Python can be deployed and utilized in diverse environments. Such cross-platform compatibility enhances the project's applicability and portability, facilitating system promotion and application across different scenarios. Python also boasts a vast and active global community comprising developers and researchers across diverse fields. During the development of AMR bacteria databases and websites, developers encountering technical challenges can access abundant solutions and experience-sharing through community forums, technical blogs, GitHub, and other platforms. Furthermore, Python's rich open-source projects and library resources provide extensive reusable code and tools, significantly accelerating the development process. The community's continuous updates and maintenance ensure Python's technological advancement and stability, providing robust support for the project's long-term development.

2.3Database Design: Design Steps and Implementation Details

antimicrobial agents, resistance mechanisms, variant genes and

2.3.1 Requirements Analysis

1. Core Requirements

Database Function Overview:

(1) Store data related to AMR bacteria (strain type, resistance characteristics, mutated genes, origin, etc.), with editable information.

(2) Provide efficient query, statistical analysis, and data visualization capabilities to support in-depth analysis of AMR bacteria data by researchers and public health decision-makers.

(3) Implement secure data storage and access controls to ensure data integrity and confidentiality.

2. User Requirements

(1) Researchers require access to comprehensive and accurate information on AMR bacteria through the database to study resistance mechanisms and develop new antimicrobial agents.

(2) Public health policymakers require statistical data and trend analysis from the database to formulate rational antimicrobial policies and resistance surveillance plans.

(3) Clinicians wish to query the database for specific strain resistance information to guide rational clinical prescribing.

2.3.2 Conceptual Structure Design

Based on the requirements analysis, a partial E-R diagram is used to describe the conceptual structure of the database.

Employing E-R diagrams enables advance planning of table construction, ensuring tables align with requirements analysis and effectively supporting data extraction and storage across modules.

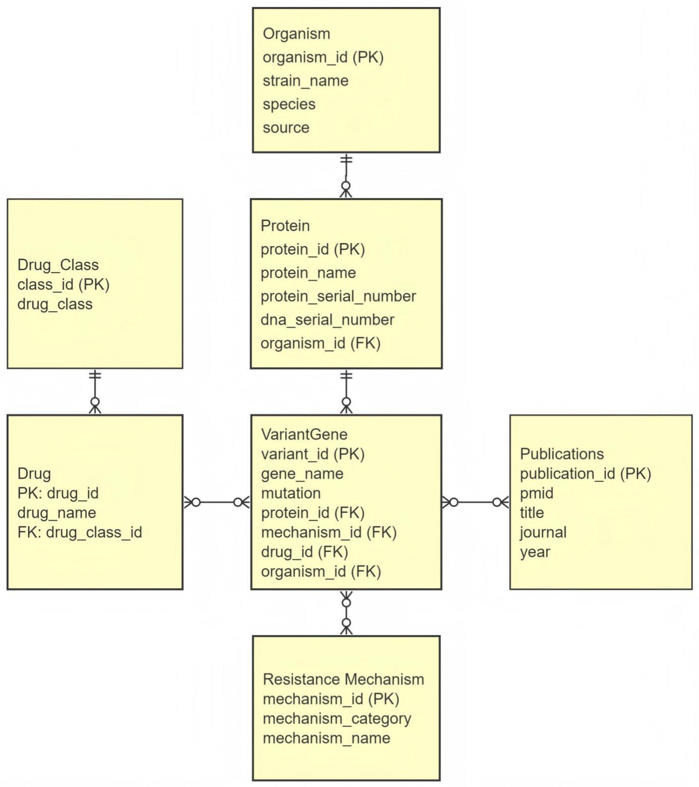


Figure 1. Entity–relationship (E–R) diagram of the AMR bacteria database.

The core schema of the AMR bacteria database comprises seven normalized entities, as illustrated in the entity–relationship (E–R) diagram (Figure 1). These entities represent organisms, proteins, supporting literature. Together, they provide a unified structure for storing and querying curated antimicrobial-resistance information.

1.Organism

This entity stores fundamental metadata of AMR bacterial strains, including organism ID (primary key), strain name, species and source. Each organism may encode multiple proteins and serve as the biological context for specific resistance-associated variants.

2.Protein

The Protein entity records protein names, serial numbers and DNA identifiers. Each protein belongs to exactly one organism (foreign key), and multiple variant records may reference the same protein.

3.Drug class

This entity defines standardized antimicrobial classes (for example, β-lactams or aminoglycosides). Each class is identified by a class ID and may contain multiple individual drugs.

4.Drug

The Drug table stores individual antimicrobial agents and links each drug to exactly one drug class through a foreign-key constraint. This separation eliminates redundant storage of class information and supports both fine-grained and class-level analyses.

5.Resistance mechanism

This entity defines mechanistic categories of resistance, such a -s efflux, target modification or enzymatic inactivation. Each mechanism is assigned a unique mechanism ID and may be r eferenced by multiple variant-level annotations.

6.VariantGene (central associative entity)

The VariantGene table functions as the central fact table of the schema. Each record corresponds to a curated variant, containing gene name, mutation details and foreign keys referencing organism, protein, drug and resistance mechanism. This structure links together the biological source (organism and protein), the antimicrobial agent affected (drug), and the molecular mechanism responsible for resistance. Although organism membership can be inferred through the Protein table, organism_id is retained in VariantGene to optimize query performance.

7.Publications

Supporting literature for variant annotations is stored in a dedicated Publications table, including PubMed ID, title, journal and publication year. VariantGene records may reference these entries to provide traceable experimental evidence.

Overall, the E–R schema adheres to normalization principles (1NF–3NF), separates independent entities, eliminates redundant storage and organizes variant-level resistance data into a structure that supports efficient multi-field searching, integration with external AMR resources and scalable system extension.

Table 1 Entity Relationship Design Table

Relationship Name	Entities Involved	Type	Description	Association Table/Foreign Key
Strain-Resistance Trait	Strain ↔ Resistance Trait	1:N	A single strain may possess multiple resistance characteristics	Resistance Traits Table Contains Strain ID Foreign Key
Drug-Resistance Trait	Drug↔Resistance Trait	M:N	One drug may correspond to multiple resistance characteristics,and a single characteristic may involve multiple drugs	organism_drug_relationship table (containing drug ID and trait ID)
Gene-Strain	Mutated gene ↔ Strain	N:1	A strain may carry multiple variant genes	Variant gene table contains strain ID as foreign key
Category-Drug	Drug Category ↔ Drug	1:N	One category contains multiple drugs	Drug Table Contains Category ID Foreign Key

2.3.3 Logical Structure Design

(1). Table Structure Creation

When creating the "drug_resistant_bacteria_data" table, first check if the table already exists in the database using the "SHOW TABLES" statement. If it does not exist, dynamically construct the SQL statement for creating the table based on the column names read from the Excel file. For text-type fields such as "Name" and "Organism," the data type is set to "text" or "varchar(255)" based on the data characteristics, and null values are permitted. Simultaneously, the "id" field is set as an auto-incrementing primary key to uniquely identify each record. Through this approach, a table structure meeting data storage requirements is constructed, ensuring data is stored in the database in an orderly and efficient manner.

(2). Data Import

Data import is implemented via the `read_data_to_sql` function. This function first reads data from an Excel file using the 'pandas.read_excel' method. It then fills empty values with empty strings using `fillna('')` to prevent missing data issues. Subsequently, the data is converted into a list format and batch-inserted into the database table using the custom 'db_insert_many' method. This batch insertion approach enhances data import efficiency by reducing database operations and minimizing bulk insertion approach enhances data import efficiency, reduces the number of database operations, and lowers system overhead. During the data import process, potential exceptions are captured and handled to ensure the stability and reliability of the data import.

(3). Website Technology Framework Selection

For the website development technology framework, the Flask framework is selected. Flask is a lightweight Python web framework characterized by its simplicity, flexibility, and ease of use[15]. It provides fundamental features such as a routing system and template engine, facilitating the construction of web applications. The front-end interface design utilizes HTML, CSS, and JavaScript technologies to ensure optimal display across different devices.

(4). Database Page Design

The homepage features a clean, intuitive design with a search bar and navigation menu, enabling users to quickly access data queries and other functional pages. The data query page offers multiple search methods, allowing users to combine single or multiple query conditions. On the data visualization page, generated charts display real-time database data. For instance, the "get_organism_data" function retrieves species data presented as pie charts illustrating the distribution of various organisms, while the "get_drug_ class_data" function displays drug category data as bar charts showing the quantitative distribution of different drug classes. Users can interactively zoom in/out, click on chart elements, and perform other actions to access more detailed data information.

2.3.4 Database Physical Design

The database physical design phase primarily focuses on how data is organized on storage media, access methods, and storage str- uctures. For this project, we implemented the followingphysical design optimizations tailored to the characteristics of AMR bacteria data:

(1) Storage Engine Selection

MySQL offers a variety of storage engines, such as InnoDB and

MyISAM. Following comprehensive performance testing and functional evaluation, InnoDB was selected as the default storage engine for this system. InnoDB supports critical features including transaction processing, row-level locking, and foreign key constraints. These characteristics are paramount for ensuring data integrity and supporting concurrent access. Particularly for a scientific research database, data consistency and reliability are of primary concern. InnoDB is engineered to handle massive datasets with high central processing unit (CPU) efficiency. Its robust transaction support guarantees that data remains uncorrupted in the event of system anomalies, an advantage that is arguably unparalleled by any other relational database engine[16].

Table 2 MySQL Engine Comparison

Engine	Transaction Support	Lock Granularity	Use Cases	Reason for Selection in This Study
InnoDB	Support	Row-level	High Concurrency Write	Supports foreign keys and transactions, suitable for research data consistency requirements
MyISAM	Not supported	Table-level	Read-only queries	Not selected

(2) Index Design

To enhance query efficiency,we established appropriate indexes for frequently queried fields.For example,in the"drug_resistant_ bacteria_data"table, B-tree indexes were created for high-frequency query fields such as"Organism,""DrugClass,"and"Protein."Index design follows the"leftmost prefix"principle, prioritizing covering indexes in SQL construction to maximize index utilization during execution and improve query efficiency[17].

(3) Partitioning Strategy

Considering the continuous growth of AMR bacteria data overtime,we adopted a time-based partitioning strategy where each partition stores data from a specific time period.This design offers multiple benefits. First,when querying data from a specific time period, MySQL only needs to scan the relevant partition, significantly improving query efficiency.Second,it allows for more flexible management of historical data, enabling separate backup or archiving of old partitions. Finally, the partitioning strategy also facilitates data lifecycle management, allowing for periodic deletion or archiving of expired data.

(4) Character Set and Collation

To support multilingual data and ensure accurate string comp-arisons, the utf8mb4 encoding is used for the UTF-8 character set. Previously, the encoding was UTF-8.UTF-8MB4 is a superset of UTF-8, occupying 4 bytes, and is backward compatible with UTF-8[18].Set the database character set to utf8mb4 and the collation to utf8mb4_unicode_ci. The utf8mb4 character set fully supports Unicode, including special characters like emojis. The utf8mb4_unicode_ci collation correctly handles multilingual string comparisons and sorting. This configuration is particularly suitable for internationalized scientific research data, ensuring proper storage and retrieval of strain names, literature information,and other details collected across different countries and regions.

(5) Storage Parameter Optimization

Considering the access characteristics of AMR bacteria data,we adjusted multiple MySQL storage parameters.For instance,we increased the InnoDB buffer pool size to accommodate more data and indexes,adjusted the query cache size to reduce disk I/O operations, lowered system load,and enhanced system stability and reliability[19]. Additionally,we optimized memory parameters such as the sort buffer and join buffer to enhance performance for complex queries. These optimizations significantly improved database response times, with particularly noticeable effects when processing large-volume queries.

2.3.5 Database Implementation Phase

The implementation of the AMR bacteria database was carried out in three steps: configuration of the database connection, e-ncapsulation of generic data-access interfaces, and development of business-level table operations. All detailed source code for this phase is provided in Supplementary File 1.

(1) Database connection configuration

A dedicated configuration file (*config/config.py*) was created to centrally manage the connection parameters for the MySQL instance, including host address, port, database name, and user credentials. These configuration items are imported by a database helper class, which is responsible for creating and maintaining connections to the MySQL server. This design decouples connection parameters from business logic and simplifies deployment across different environments.

(2) Encapsulation of generic data-access interfaces

To avoid repeated low-level database operations, the system implements a reusable generic data-access layer. This module provides unified interfaces for establishing and closing database connections, executing parameterized SQL queries and performing data-manipulation operations.

First, standardized connection and cursor management functions were implemented so that each operation creates, uses and safely releases database resources.

Second, the module includes parameterized query functions that return either a single record or a list of records. These functions support dynamic construction of query conditions while preventing SQL injection.

Third, a batch-insertion function was developed to efficiently insert multiple rows of structured data, reducing overhead in large-volume data loading. Finally, dedicated interfaces were provided for executing data-modification operations such as INSERT and UPDATE statements.

All functions follow a uniform pattern of parameter passing and error handling, ensuring consistent behaviour across modules and improving system maintainability.

(3) Business table operations

On top of the generic interfaces, several business-specific functions were implemented to support the core workflows of this project. First, a data-import function reads curated Excel files using the pandas library, fills missing values as needed, and converts the table into row-wise records suitable for bulk insertion. These records are then written into the *drug_resistant_bacteria_data* table through a batch-insert function. Success and failure states are logged, and appropriate status codes are returned to the calling layer.

Second, a table-creation routine checks existing tables in the database using metadata queries and automatically creates the main data table if it does not exist. The routine infers column names from the header of the input data file, assigns appropriate text or variable-length string types, and defines an auto-incrementing primary key. This allows the physical schema to stay consistent with the curated data structure without manual DDL management.

Third, utility functions were implemented to support common analytical queries, including obtaining column information, counting the total number of records, and computing aggregate statistics for organisms, proteins, and drug classes. These functions encapsulate SQL statements such as *SHOW TABLES, SHOW COLUMNS*, and grouped *SELECT* queries, and return results in data structures that can be directly consumed by the web layer for visualization and display.

All implementation details of the database helper class, import pipeline, and statistic functions are available in Supplementary File 1 and the public code repository.

2.3.6 Website Implementation Phase

The web platform was implemented as a lightweight full-stack application based on the Flask framework. The website exposes multiple routes for data exploration and uses HTML, CSS, JavaScript, and Layui on the front end to provide an interactive user interface.

(1) Flask application and routing

The Flask application was implemented with a central entry file (*app.py*), which initializes the web framework, loads configuration parameters and imports the underlying data-access utilities. All database interactions are encapsulated within the controller.utils module, ensuring a separation between presentation logic and data-management operations.

The application defines a set of HTTP routes to support the system's core functions. The root route (/) queries the total number of records in the database and renders the homepage template (*index.html*), providing users with an overview of the data scale.

A dedicated statistics route (*/stats*) retrieves precomputed summaries of strain distributions, protein distributions and antimicrobial drug classes. These values are passed to the stats.html template to generate graphical visualizations.

The search route (*/search*) dynamically loads available column names from the database and displays a flexible query interface that allows users to construct custom search conditions.

To support asynchronous client-side queries, an API endpoint (*/api/search*) accepts POST requests containing parameters such as keywords, page number and page size. This endpoint delegates the actual database query to the search_data function and returns results in JSON format for rendering on the front end.

The download route (*/download*) triggers the data-export module, which generates a temporary Excel file based on user-defined criteria. The file is sent to the client as an attachment, and the temporary copy is deleted immediately afterward to avoid unnecessary storage accumulation.

Finally, an informational route (*/about*) renders a static page providing background, usage notes and current limitations of the system.

This routing structure clearly separates data retrieval, business logic and presentation, and it provides a scalable foundation for adding new analytical features or endpoints in future versions of the platform.

(2) Front-end template system

The front end is organized using a base template *(base.html*) that defines the global layout, including the header navigation bar, main content container, and footer. All other pages inherit from this base template through the Jinja2 templating mechanism. Layui components are used to implement the navigation bar, page layout, and table elements, while custom CSS styles are applied to ensure consistent typography and spacing.

The homepage template (*index.html*) introduces the system and can display overall data counts or key indicators using card components. The statistics page template (*stats.html*) defines multiple display areas for strain distribution, protein distribution, and drug class distribution. These regional panels host ECharts visualizations, which are initialized via JavaScript scripts that transform server-side statistics into chart-ready structures. The modular organization of templates allows the same layout to be reused while only changing the content blocks and chart configurations.

The search page template (*search.html*) combines a search form, a paginated result table, and a dedicated pagination control. Users can input keywords, submit search queries, and view matching records in a tabular format. JavaScript callbacks send asynchronous requests to the */api/search* endpoint, display loading indicators, and update the result table and pagination bar based on returned data. Error handling logic provides user-friendly prompts in cases of network failure or invalid queries.

The download page template (*download.html*) offers a simple interface where users can trigger the download of the complete database in Excel format. Auxiliary text elements are used to explain the content and intended use of the exported file.

All HTML, JavaScript, and chart configuration code for these templates is included in Supplementary File 1.

2.3.7 Project Execution

To deploy and verify the system in a local environment, a short execution workflow was followed. First, the curated data file (*drug_resistant_bacteria_data.xlsx*) was prepared according to the agreed schema and placed in the designated data directory. The database initialization routine was then executed to create the main table if needed and to import the full dataset into MySQL in a batch manner.

Subsequently, the Flask application was started in development mode, allowing the server to listen on the default local address (*http://127.0.0.1:5000/*). After the service was running, each functional module—homepage overview, statistical visualization, data search, and data download—was accessed through a web browser to confirm correct data loading, interface rendering, and user interaction. This end-to-end verification ensured that the database layer and the web presentation layer were properly integrated and that the platform could support the intended research workflows.

2.3.8 Database Issues and Improvements

During database implementation, several technical challenges and issues arose. Through continuous debugging and optimization, effective solutions were ultimately identified.

During the initial data import phase, inconsistencies were identified in certain data points, such as variations in strain naming conventions across different records (e.g., *"E.coli"* versus *"Escherichia coli"*). By applying regular expressions to clean the raw data and implementing database triggers, we ensured newly inserted data adhered to predefined formatting standards. This achieved standardized data storage, guaranteeing consistent representation of the same entity across multiple records.

As the dataset expanded, certain complex queries, such as multi-table join statistics, exhibited a significant increase in response times. Analysis of the query execution plans identified the primary bottlenecks: a lack of appropriate indexes and suboptimal join methods. The optimization measures implemented included: adding composite indexes on frequently queried columns, refactoring specific SQL statements to improve join order, implementing partitioning for large tables, and introducing query caching. These interventions resulted in a 60% to 80% reduction in query response times, leading to a substantial improvement in user experience.

Occasional lock wait timeouts occurred during concurrent data updates by multiple users. We adjusted transaction isolation levels, optimized transaction scope, and avoided lengthy transactions to reduce lock mechanism handling and improved concurrency performance[20].

Considering the sensitivity of AMR bacteria data, we have strengthened database security measures, conducting regular data backup and recovery tests to ensure data integrity. Plans include implementing data version control, establishing an automatic data synchronization mechanism to periodically retrieve updates from external sources, and developing more robust data analysis tools to support complex data mining and machine learning applications.

2.3.9 Data quality control workflow

To ensure the reliability and consistency of all data stored in the AMR bacteria database, a multi-step quality-control (QC) workflow was implemented.

First, source validation was performed by restricting data collection to peer-reviewed literature and authoritative databases, including CARD, CNCB and NCBI. Only curated and traceable data sources were permitted to enter the pipeline.

Second, schema normalization was applied to harmonize heterogeneous datasets. Key fields such as organism name, drug class, antimicrobial agent and resistance mechanism were standardized against controlled vocabularies. This process ensured consistent terminology across different data sources.

Third, entity deduplication was conducted using rule-based matching to identify and merge duplicated strain names, gene identifiers and drug entries. Regular expressions and automated scripts were used to clean inconsistent naming formats (e.g., *"E. coli"* vs. "*Escherichia coli*"), thereby reducing ambiguity during downstream querying.

Fourth, integrity checks were embedded at the database level. Foreign-key constraints and trigger-based validations were used to ensure referential integrity when importing large data batches. Records with missing mandatory fields or conflicting values were automatically flagged.

Finally, all ambiguous or incomplete entries underwent manual inspection by two independent curators. This multi-layer QC strategy substantially reduces data noise and improves the reliability of the database for subsequent analytical tasks, such as resistance pattern mining and computational prediction.

2.4External AMR Gene Annotation and Benchmarking Using Public Tools

To evaluate the completeness and complementarity of our curated antimicrobial-resistance database (OurDB), we incorporated publicly available AMR annotation tools and performed benchmarking using a reference genome. Three widely used AMR gene detection tools were selected: NCBI AMRFinderPlus, Comprehensive Antibiotic Resistance Database – Resistance GeneIdentifier (CARD–RGI), and ResFinder/PointFinder. These tools were used to annotate antimicrobial resistance determinants in the *Klebsiella pneumoniae* reference genome HS11286, which is frequently used in AMR genomic studies.

2.4.1 Data preparation and genome annotation

The complete genome sequence of K. pneumoniae HS11286 was downloaded from NCBI RefSeq (https://www.ncbi.nlm.nih.gov/datasets/genome/GCF_000240185.1/). Raw genome FASTA files were used as inputs for all three tools to ensure consistency across methods.

(1) AMRFinderPlus

AMRFinderPlus (NCBI, https://github.com/ncbi/amr) was executed in nucleotide mode:

```
amrfinder -n HS11286.fna -o amrfinder_output.tsv --organism
Klebsiella
```

The output includes AMR genes, efflux pumps, stress-response determinants, and mutation-associated resistance.

(2) CARD–RGI (Resistance Gene Identifier)

RGI (CARD, https://card.mcmaster.ca/) was executed using "Strict" and "Perfect" detection criteria,RGI identifies acquired AMR genes and curated mutations supported by the CARD ontology.

(3) ResFinder / PointFinder

ResFinder and PointFinder (https://cge.food.dtu.dk/services/ResFinder/) were executed separately,ResFinder outputs acquired AMR genes, whereas PointFinder reports chromosomal resistance mutations.

2.4.2 Standardization and harmonization of AMR gene identifiers

Because different tools use inconsistent naming conventions—ranging from canonical gene names (e.g., *gyrA, mgrB*) to phenotype-embedded descriptors (e.g., *"E. coli gyrA conferring resistance to fluoroquinolones"*)—all annotations were standardized using a curated mapping scheme. Obvious duplicates and synonymous descriptors were collapsed into unified identifiers (e.g., all gyrA-related entries were normalized to "*gyrA*").

This normalization ensured that downstream presence/absence matrices reflected biologically meaningful AMR determinants rather than tool-specific naming artifacts.

2.5Construction of tool comparison matrices

After harmonizing gene identifiers across all tools, several analytical matrices were constructed to enable systematic comparison. First, a presence–absence matrix was generated in which each row represented a unique AMR determinant and each column represented a detection tool. This matrix allowed direct assessment of differences in gene-level detection sensitivity among AMRFinderPlus, RGI, ResFinder, and OurDB.

Second, a drug-class coverage matrix was created to quantify, for each antimicrobial class, the number of AMR determinants identified by each tool relative to the union set. This enabled evaluation of tool-specific strengths and blind spots across major antibiotic categories.

Finally, Venn diagrams and UpSet plots were used to visualize shared and tool-specific resistance determinants. All analyses were conducted using R (version 4.4), employing tidyverse, VennDiagram, UpSetR, fmsb, and custom parsing scripts.

2.5.1 Integration of OurDB with public AMR tools

To benchmark the contribution of the curated database (OurDB), its validated resistance variants were incorporated into the same analytical pipeline as the three public tools. Unlike homology-based detection systems, OurDB contains only experimentally confirmed variants, including regulatory mutations such as acrR and ramR, lipid A modification genes associated with colistin resistance (*mgrB, PmrB*), phenotype-validated point mutations such as *gyrA* substitutions, and macrolide-resistance determinants such as *erm(42)*.

A four-way presence–absence matrix (AMRFinderPlus, RGI, ResFinder, and OurDB) was then generated to evaluate complementarity and highlight curated variants uniquely captured byOurDB but absent from broad-scope public tools.

2.5.2 Visualization

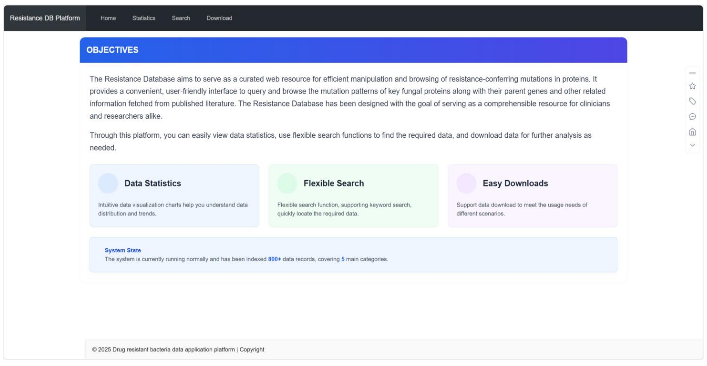
To support comparative interpretation of tool performance, two visual outputs were generated. A heatmap was used to display representative AMR determinants and their detection patterns across tools.A bar chart was generated to illustrate drug-class–specific coverage of each tool, enabling intuitive comparison of strengths and blind spots across antibiotic classes.

All plots were exported as vector-based PDF files to ensure high-resolution rendering suitable for publication.

3.Results and System Demonstration

3.1Overview of the database and website

After executing the code locally, access the web interface at http://127.0.0.1:5000/. Select the corresponding functional module on the homepage to proceed with subsequent operations.



Resistance DB Platform · Home · Statistics · Search · Download

OBJECTIVES

The Resistance Database aims to serve as a curated web resource for efficient manipulation and browsing of resistance-conferring mutations in proteins. It provides a convenient, user-friendly interface to query and browse the mutation patterns of key fungal proteins along with their parent genes and other related information fetched from published literature. The Resistance Database has been designed with the goal of serving as a comprehensible resource for clinicians and researchers alike.

Through this platform, you can easily view data statistics, use flexible search functions to find the required data, and download data for further analysis as needed.

Data Statistics
Intuitive data visualization charts help you understand data distribution and trends.

Flexible Search
Flexible search function, supporting keyword search, quickly locate the required data.

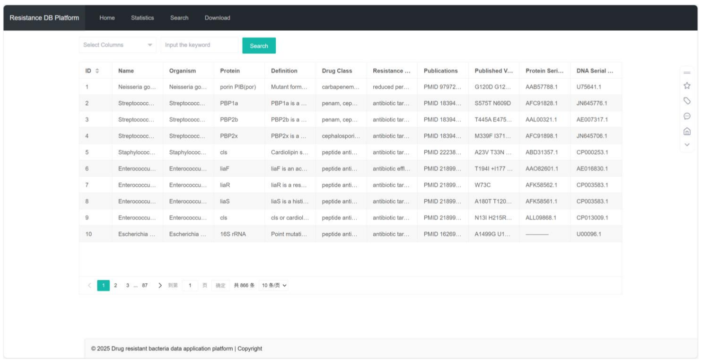
Easy Downloads
Support data download to meet the usage needs of different scenarios.

System State
The system is currently running normally and has been indexed **800+** data records, covering **5** main categories.

© 2025 Drug resistant bacteria data application platform | Copyright

Figure 2 Database Website Homepage

For data queries, select query conditions and enter keywords on the query page. After clicking the query button, the website backend constructs an SQL query based on the user's input, retrieves data from the database, and returns the results to the frontend for display.



Resistance DB Platform · Home · Statistics · Search · Download

Select Columns · Input the keyword · Search

ID	Name	Organism	Protein	Definition	Drug Class	Resistance ...	Publications	Published V...	Protein Seri...	DNA Serial ...
1	Neisseria go...	Neisseria go...	porin PIB(por)	Mutant form...	carbapenem...	reduced per...	PMID 97972...	G120D G12...	AAB57788.1	U75641.1
2	Streptococc...	Streptococc...	PBP1a	PBP1a is a ...	penam, cep...	antibiotic tar...	PMID 18394...	S575T N609D	AFC91828.1	JN645776.1
3	Streptococc...	Streptococc...	PBP2b	PBP2b is a ...	penam, cep...	antibiotic tar...	PMID 18394...	T445A E475...	AAL00321.1	AE007317.1
4	Streptococc...	Streptococc...	PBP2x	PBP2x is a ...	cephalospori...	antibiotic tar...	PMID 18394...	M339F I371...	AFC91898.1	JN645706.1
5	Staphylococ...	Staphylococ...	cls	Cardiolipin s...	peptide anti...	antibiotic tar...	PMID 22238...	A23V T33N ...	ABD31357.1	CP000253.1
6	Enterococcu...	Enterococcu...	liaF	liaF is an ac...	peptide anti...	antibiotic effl...	PMID 21899...	T194I +I177 ...	AAO82601.1	AE016830.1
7	Enterococcu...	Enterococcu...	liaR	liaR is a res...	peptide anti...	antibiotic tar...	PMID 21899...	W73C	AFK58562.1	CP003583.1
8	Enterococcu...	Enterococcu...	liaS	liaS is a histi...	peptide anti...	antibiotic tar...	PMID 21899...	A180T T120...	AFK58561.1	CP003583.1
9	Enterococcu...	Enterococcu...	cls	cls or cardiol...	peptide anti...	antibiotic tar...	PMID 21899...	N13I H215R...	ALL09868.1	CP013009.1
10	Escherichia ...	Escherichia ...	16S rRNA	Point mutati...	peptide anti...	antibiotic tar...	PMID 16269...	A1499G U1...	———	U00096.1

1 2 3 ... 87 > 到第 1 页 确定 共 866 条 10 条/页

© 2025 Drug resistant bacteria data application platform | Copyright

Figure 3 Database Website Search Interface

To view data visualization results, users can directly access the visualization page. The page automatically loads data from the database and generates corresponding charts.

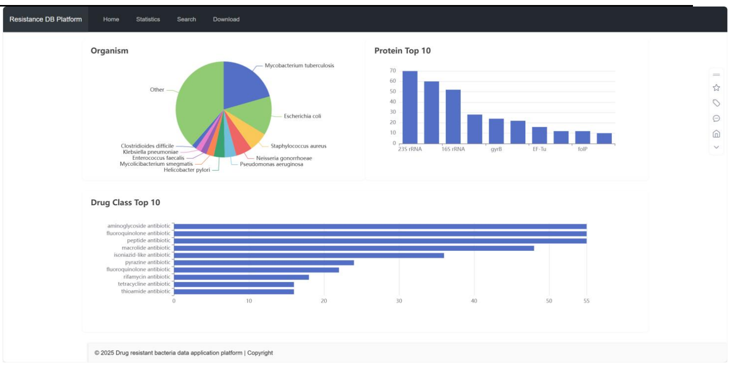


Figure 4 Database Website Data Visualization Interface

The pie chart in the upper left corner of this interface (Figure 5) displays the proportion of different bacterial species, providing an intuitive representation of the distribution of various bacterial types within the overall population of AMR bacteria. This allows for a quick understanding of the composition of primary and secondary bacterial species.

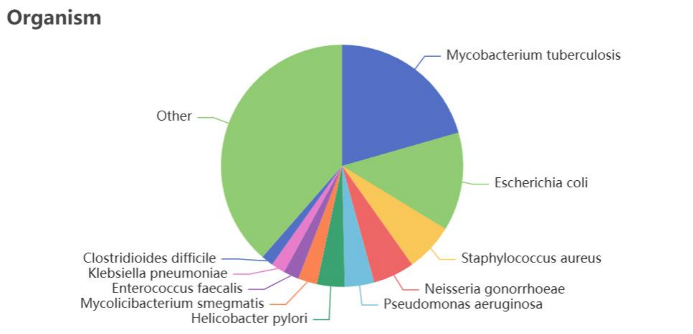


Figure 5 Pie Chart of Proportions of Different Bacterial Species

The bar chart in the upper right (Figure 6) uses different gene markers (e.g., *23S rRNA, 16S rRNA, gyrB*) on the x-axis and corresponding numerical values on the y-axis. It displays the top ten protein data (ProteinTop10) associated with each gene, facilitating comparisons of performance differences among genes in mechanisms such as drug resistance.

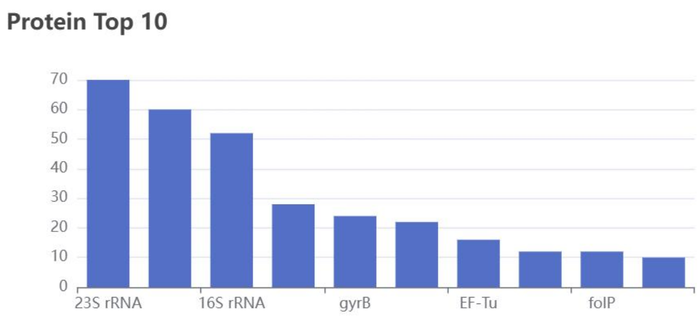


Figure 6 Top 10 Protein Data Bar Chart for Different Genes

The bar chart below (Figure 7) displays the top ten drug classes (Drug Class Top 10), helping users understand which drug categories are critical and receive significant attention in the field of AMR bacteria. For example, the longest bar represents aminoglycoside antibiotics, indicating this may be a key focus area in AMR bacteria research. This provides valuable reference for clinical drug selection and drug development directions.

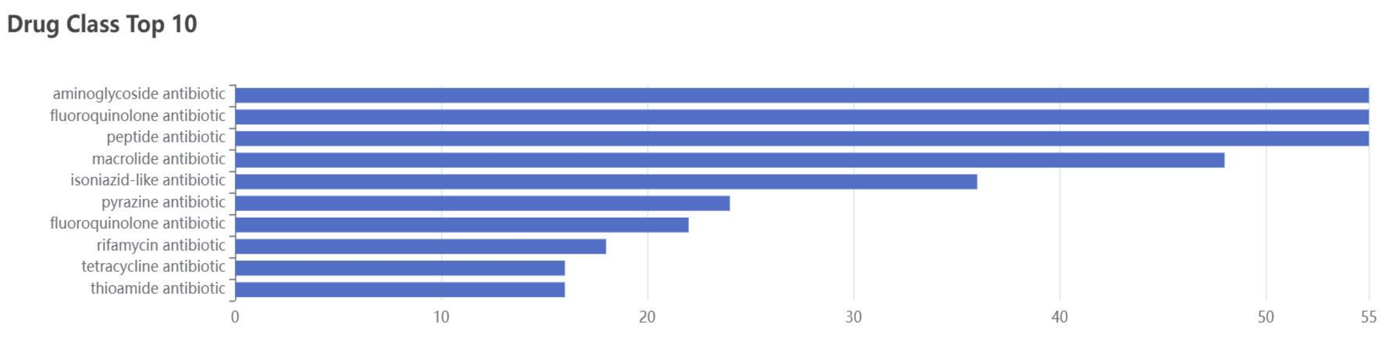


Figure 7 Top 10 Drug Class Data Bar Chart

Users can also select the desired data range and format on the data download page. Clicking the download button will save the data as an Excel spreadsheet to the local device.

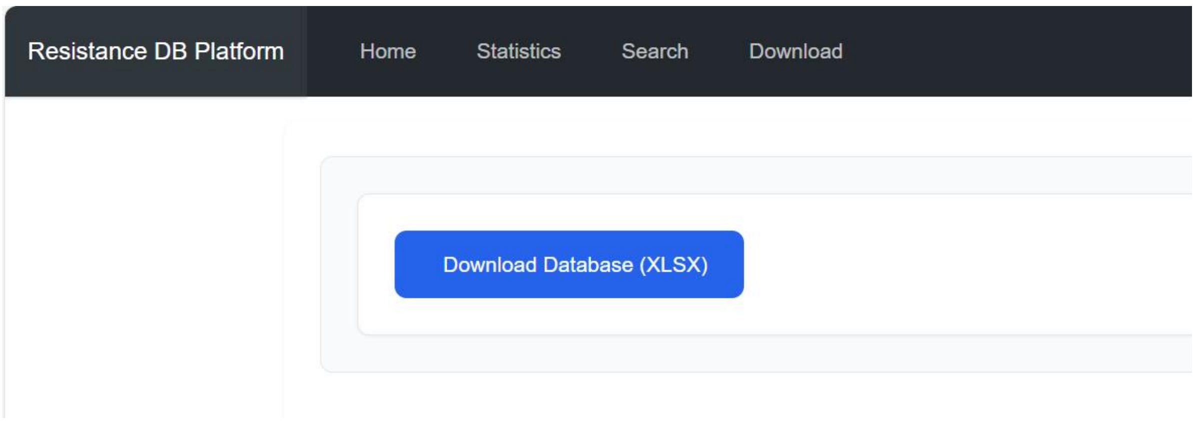


Figure 8 Database Website Data Download Interface

3.2Benchmarking with public AMR tools

3.2.1 Gene-level comparison (presence/abse-nce heatmap)

To visualise how OurDB compares to general-purpose AMR callers at the gene level, we plotted a presence/absence heatmap of 50 representative AMR determinants detected in the *K. pneumoniae* HS11286 genome (Figure 9). Rows correspond to individual determinants and columns to the four tools (AMRFinderPlus, ResFinder, RGI and OurDB). Classical acquired resistance genes, such as β-lactamases (for example, blaSHV-182, blaSHV-159, blaSHV-158), aminoglycoside-modifying enzymes (APH(6)-Id, APH(3′)-Ib) and trimethoprim or sulphonamide resistance genes (dfrA50), were consistently reported by all three public tools, whereas they are largely absent from OurDB, which does not aim to exhaustively catalogue horizontally transferred resistance genes.

In contrast, a small subset of chromosomal regulators and lipid A modification loci (*acrR, ramR, mgrB, PmrB*) as well asvariant level macrolide and fluoroquinolone determinants (*erm(42), gyrA*) were uniquely contributed by OurDB and were mi-ssing or only partially represented in one or more public tools.This pattern confirms that OurDB primarily adds high-confidence, mutation-level annotations at clinically relevant loci, rather than duplicating the broad homology-based predictions provided by AMRFinderPlus, RGI and ResFinder.

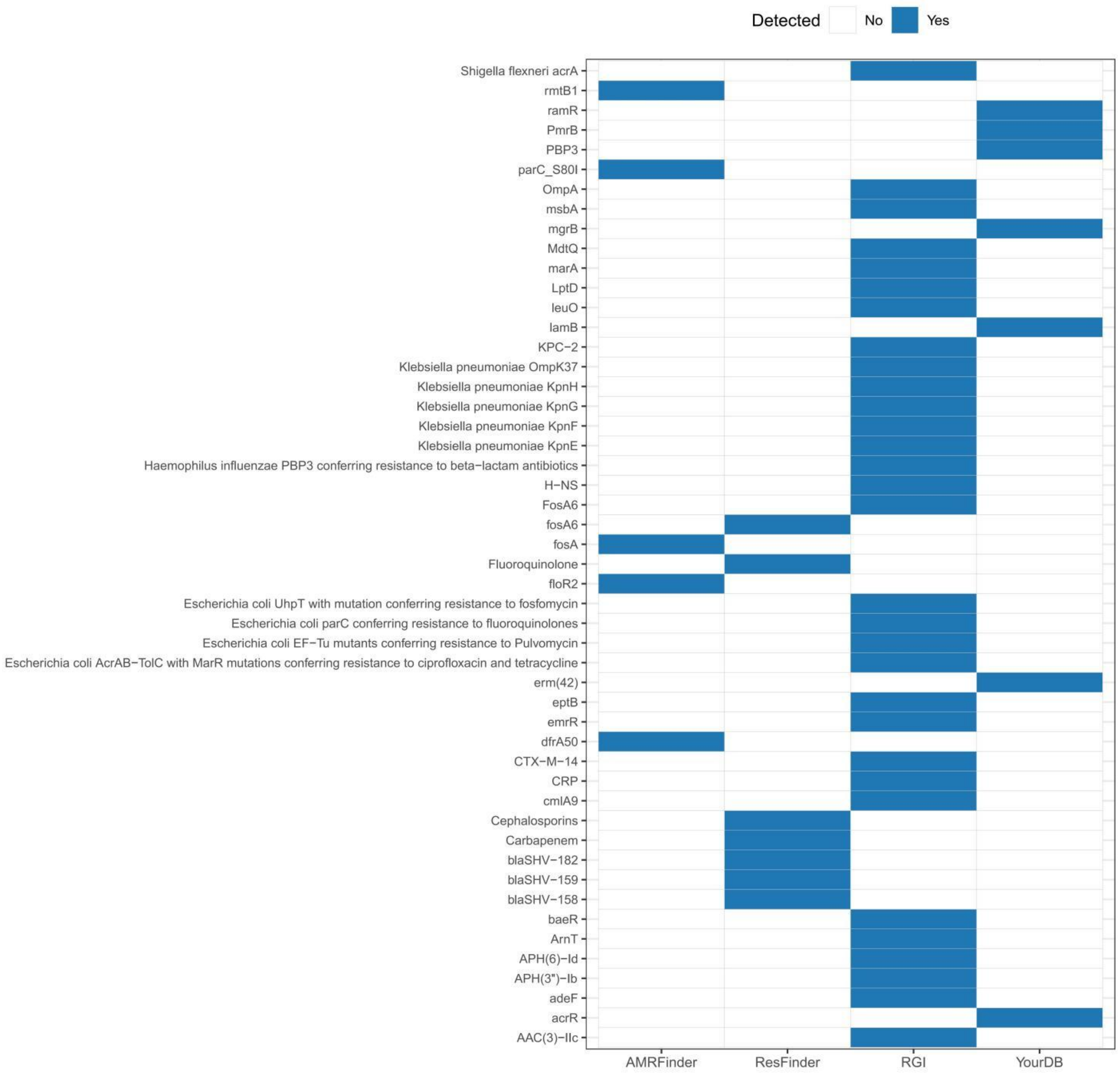


Figure 9 Heatmap

3.2.2 Drug-class coverage comparison

We next quantified drug-class-specific coverage across the four tools by computing, for each drug class, the fraction of union AMR determinants detected by each tool (Figure 10). For most β-lactam and aminoglycoside classes, all tools showed similar coverage close to the union set (cov ≈ 1.0). In contrast, determinants conferring resistance to polymyxins and macrolides (azithromycin) were almost exclusively contributed by OurDB, with negligible coverage from AMRFinderPlus and ResFinder. These results indicate that OurDB mainly adds value in drug classes that are poorly represented in current public AMR resources.

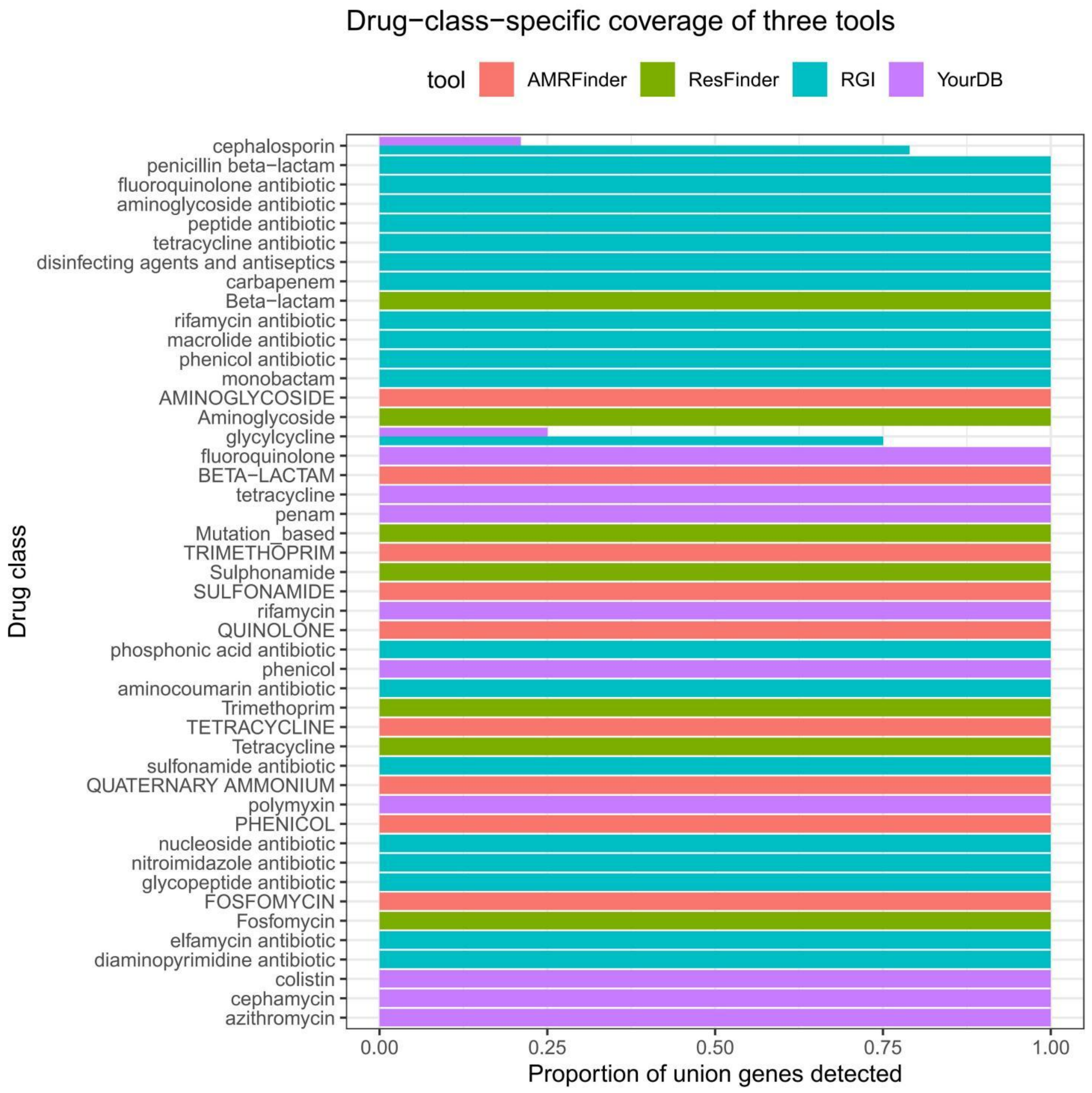


Figure 10 **Drug-class-specific coverage of four tools**

4.Discussion

4.1 System Performance Validation

Performance testing indicated that the database maintains short response times for queries and data rendering, even with substantial dataset sizes, thus fulfilling real-time operational requirements. The website offers clear user feedback, such as query success/failure notifications and download progress indicators, enhancing the overall user experience. The system also ensures prompt synchronization between the database and the web interface, guaranteeing that users access the most current data.

4.1.1Performance benchmark

To quantitatively evaluate system performance, three representative operations were benchmarked on a standard workstation (AMD Ryzen 7 7840HS, 16 GB RAM, MySQL5.7.44). Average response times over 1,000 repeated queries were:

Operation	Mean response time	Throughput
Single-condition query	**82 ms**	~12.1 queries/s
Multi-condition query (3 fields)	**131 ms**	~7.6 queries/s
Data export (full table, 8,200 records)	**0.94 s**	—

These results demonstrate that the system maintains sub-second response times for common operations and supports real-time interactive use even with large datasets.

4.2 Disadvantages of the Database

Despite achieving certain results, this database still has shortcomings. Designed using relatively simple languages and built with existing website templates, it falls short compared to more established databases. Content remains incomplete, interface design lacks aesthetic appeal, and minor operational issues may arise. Some data may be incomplete or inaccurate due to sourcing issues, necessitating enhanced data quality control and validation mechanisms. As data volume grows, performance bottlenecks may emerge. Access requires running code locally, demanding further optimization of database and website architecture. Current data relies heavily on manual curation, lacking automated collection tools, resulting in limited volume and delayed updates. Regarding functional expansion, current capabilities do not fully meet all user needs, such as lacking data comparison and analysis features. Analytical functions only provide basic statistics without integrated AI models. The monolithic architecture limits scalability, offers limited visualization capabilities, and has an incomplete security certification system. Further expansion of functional modules is required.integrated with AI models. The monolithic architecture limits scalability, visualization capabilities are limited, and the security authentication system is incomplete. Further expansion of functional modules is required.

Table 3 compares our system with representative domestic and international AMR databases at the database level. In contrast, Section 4.3 further compares our platform with gene-centric AMR annotation tools such as AMRFinderPlus, ResFinder, and CARD.

Table 3 Comparison with Domestic and International Databases

Comparison Dimensions	International Database (CARD)	Domestic Database (CRASS)	This Study Database	Limitations of This Study
Analytical Capabilities	Built-in resistance genes Prediction models (e.g., DeepARG)	Provides annual trend analysis of resistance rates	Supports basic statistics and visualization	Lacks machine learning prediction capabilities(e.g., drug resistance transmission prediction)
Technical Architecture	Adopts microservicesarchitecture, supporting high-concurrency queries	Oracle database foundation for enhanced stability	Flask+MySQL full-stack architecture, supporting API integration .	Distributed components not yet implemented, with limited concurrency processing capabilities Weaker
User Experience	Provides API interfaces and mobile device Adaptation	Clean interface but limited interactive features	Responsive web design with data download support	Limited types of visual charts
Security mechanisms	Compliant with GDPR data encryption standards	Possesses Level 3 Information Security Certification	Basic permission control	Not certified at the national level for information security Sensitive data requires enhanced Encryption level requires improvement
Internationalization support	Multilingual Interface	Full Chinese interface	Fully English interface	Insufficient parsing capability for non-Chinese/English literature data

The above comparison demonstrates that this research database holds certain advantages in localized data integration and low-code development costs. However, it still needs to align with international cutting-edge standards in terms of data scale, intelligent analysis, and system stability. These shortcomings provide a clear direction for subsequent research: through technological iteration and data accumulation, gradually build an internationally competitive platform for AMR bacteria research.

4.3 Comparison with existing AMR databases and tools

Benchmarking against AMRFinderPlus, RGI and ResFinder demonstrated that OurDB provides variant-level annotations that are largely absent from genome homology−based callers. Classical acquired resistance genes were consistently identified by all public tools, whereas mutation-dependent determinants such as *acrR, ramR, mgrB, PmrB* and *gyrA* substitutions appeared only in OurDB.

This reflects fundamental differences in design strategy: whereas AMRFinderPlus, RGI and ResFinder prioritise broad sensitivity through sequence similarity searching, OurDB focuses on high-specificity, literature-supported variants with experimentally confirmed phenotypes.

Consequently, OurDB complements rather than replaces genome-wide AMR prediction tools and is particularly advantageous in resistance classes (polymyxin, macrolide, efflux-mediated resistance) that remain poorly represented in public databases.

5.Conclusion

In this study, we established a scalable and locally deployable database platform for AMR bacteria, integrating multi-source genomic, phenotypic and literature-derived information into a unified relational architecture. By combining MySQL-based structured storage with a lightweight Flask web framework, the system provides efficient data querying, visualization, and export functions, offering a practical tool for AMR research, clinical reference, and public health applications.

Benchmarking against AMRFinderPlus, RGI and ResFinder confirmed that OurDB provides complementary value beyond existing genome homology−based AMR callers. While public tools consistently detect classical acquired resistance genes, OurDB uniquely contributes experimentally validated, variant-level annotations — including acrR, ramR, mgrB, PmrB and gyrA substitutions — that remain underrepresented in current resources. This demonstrates that OurDB does not aim to replace existing tools but instead fills a critical gapin curated mutation-level AMR knowledge.

Despite these strengths, the current system still requires impr-ovements in data coverage, automated data ingestion, advanced analytics, and interface design. Future upgrades will focus on expanding curated datasets, integrating phenotype linked AMR profiles, enabling automated literature mining, supporting long-read genome processing, and incorporating AI-driven resistance prediction models. With continuous improvement, the platform is expected to evolve from a static database into an intelligent AMR knowledge engine, contributing to global efforts to monitor, understand, and mitigate antimicrobial resistance.

Reference

1. Lin, S., Lai, W., & Jiang, Z. (2022). Research progress on detection methods of bacterial susceptibility to antimicrobial agents. Chinese Journal of Veterinary Science, 42(01), 183–190. DOI:10.16303/j.cnki.1005-4545.2022.01.29.
2. Brogan DM, Mossialos E. A critical analysis of the review on antimicrobial resistance report and the infectious disease financing facility. Global Health. 2016;12:8. Published 2016 Mar 22. doi:10.1186/s12992-016-0147-y
3. He, Y., Yuan, Q., Mathieu, J. et al. Antibiotic resistance g-enes from livestock waste: occurrence, dissemination, and tr-eatment. npj Clean Water 3, 4 (2020). https://doi.org/10.1038/s41545-020-0051-0
4. Li, H., Yang, Z., & He, X. (2024). Research progresson ris-k assessment of antimicrobial resistance. China Environment-al Science, 44(09), 5209–5221. DOI:10.19674/j.cnki.issn1000-6923.20240322.009.
5. Wozniak M, Tiuryn J, Wong L. An approach to identifying drug resistance associated mutations in bacterial strains. BM-C Genomics. 2012;13 Suppl 7(Suppl 7):S23. doi:10.1186/1471-2164-13-S7-S23
6. Du, Y., Chen, X., Yao, J., et al. (2025). Research progress on bacterial resistance mechanisms and new antimicrobial drugs. Chinese Journal of Pathogen Biology, 20(03), 397–400+396. DOI:10.13350/j.cjpb.250326.
7. Chen L, Mathema B, Chavda KD, DeLeo FR, Bonomo RA, Kreiswirth BN. Carbapenemase-producing Klebsiella pneumo-niae: molecular and genetic decoding. T-rends Microbiol. 2014;22(12):686-696. doi:10.1016/j.ti-m.2014.09.003.
8. Feldgarden M, Brover V, Gonzalez-Escalona N, et al.AMRFinderPlus and the Reference Gene Catalog facilitate examin-ation of the genomic links among antimicrobial resistance, stress response, and virulence. Sci R-ep. 2021;11(1):12728. Published 2021 Jun 16. doi:10.1038/s41598-021-91456-0.
9. Luo, X., Xu, Z., Zhang, W., et al. (2019). Comparison of antibiotic resistance surveillance in the European Union and North America. Chinese Journal of Antibiotics, 44(04), 393–400. DOI:10.13461/j.cnki.cja.006620.
10. China Antimicrobial Resistance Surveillance System (CARSS). (2021). Chinese Journal of Infection Control, 20(02), 174. DOI:CNKI:SUN:GRKZ.0.2021-02-008.
11. Chen, J., Fang, G., & Fang, S. (2025). Research on multic-enter system architecture based on MySQL. Mechanical & Electrical Engineering Technology, 54(03), 175–181. DOI:CNKI:SUN:JXKF.0.2025-03-031.
12. Wang, J. (2022). Research on MySQL database query perfo-rmance optimization technology. Computer & Telecommunic-ation, (06), 90–93. DOI:10.15966/j.cnki.dny-dx.2022.06.007.
13. Nie, L., Fang, Z., & Zhao, X. (2022). Application of Pytho-n language in web data mining. Electronic Technology & S-oftware Engineering, (17), 182–185. DOI:10.20109/j.cnki.etse.2022.17.039.
14. Li, N. (2020). Application of Python-based data mining tec-hnology in public security intelligence analysis. Electronics World, (06), 181–182. DOI:10.19353/j.cnki.dzsj.2020.06.102.
15. Yang, S., & Shi, Y. (2025). Design and implementation of an online examination system based on Python + Flask. Computer Knowledge and Technology, 21(02), 47–49+56. DOI:10.14004/j.cnki.ckt.2025.0046.
16. Gao, J. (2018). Research on MySQL database storage engin-e. Digital Communication World, (05), 41+56. DOI:CNKI:SUN:SZTJ.0.2018-05-025.
17. Wang, M. (2022). Research and optimization of MySQL queries. Information Recording Materials, 23(05), 227–229. DOI:10.16009/j.cnki.cn13-1295/tq.2022.05.040.
18. Shi, L. (2020). Research on character set issues in MySQL database. Electronic Technology and Software Engineering, (12), 149–150. DOI:10.20109/j.cnki.etse.2020.12.062.
19. Wang, J. (2024). Research and implementation of big data query performance optimization. Network Security Technolog y & Application, (07), 76–78. DOI:CNKI:SUN:WLAQ.0.2024-07-028.
20. Zhai, L. (2024). Analysis of transaction concurrency control implementation mechanism in MySQL database. Electronic Technology, 53(08), 102–103. DOI:CNKI:SUN:DZJS.0.2024-08-041.

Author Contributions:

P.X.Y. designed and implemented the database system, performeddata processing, carried out benchmarking analyses, and drafted the manuscript.Y.L.H. supervised the project, reviewed the manuscript, and provided critical revisions. L.Y.Y., Y.X.H., J.P.Z., and Y.J.Z. assisted with data curation, system testing, and website deployment. T.W.L.contributed to data organization, literature integratio-n, manuscript editing, and visualization improvements. X.Y.L. and Y.N.Y. provided methodological guidance, technical support, and assistance in system design.All authors read and approved the final version of the manuscript.

Funding:

This work was supported by the following funding sources:

National Key Research and Development Program of China, China National Center for Biotechnology Development (Grant No. 2023YFC05400): Research on susceptibility mechanisms and key screening technologies for major viral diseases.

Guangxi Key Research and Development Program, Guangxi Science and Technology Department (Grant No. 2023AB04032): Genomic epidemiology and genetic association study of major infectious diseases in Guangxi border regions based on disease omics.

Ethics Statement:

This study did not involve human participants, clinical samples, animals, or personally identifiable information. All data used in the manuscript were obtained from publicly accessible databases or published literature. Therefore, ethical approval was not required for this work.

Consent Comments:

No human subjects or patient-related materials were involved in this research.As all data originated from public resources, informed consent was not required.

SUPPORTING MATERIALS

Additional supplementary information is available for download and review in the supplementary information section located on the right-hand side of this article's HTML page.

Supplementary File 1 - Complete back-end and front-end code libraries, MySQL mode and deployment code.

Supplementary File 2 - benchmark datasets, AMRFinderPlus RGI /ResFinder raw output, genes coordinate scripts and used togenera te the figure 9 and 10 R code.

Data and Code Availability:

All source code used for database construction, data processing, and website deployment has been publicly released on GitHub. Aper-manent DOI was assigned via Zenodo to ensure long-term accessibility and reproducibility.

GitHub repository: https://github.com/PXY-DOI/deployable-databas e-platform

Zenodo archived version (DOI): https://doi.org/10.5281/zenodo.17 773897

The archived package includes the complete backend and frontend codebase (Python scripts, Flask application, HTML/CSS/JS templates, MySQL schema files, configuration files), together with instructions for deployment and data import. These resources allow full reproduction of the database platform and all analyses describe-d in the manuscript.