SYSTEM TESTING

System testing usually consists of a layered process, including the user interface (UI) layer, the business layer, the data access layer and the database itself. The UI layer deals with the interface design of the database, while the business layer includes databases supporting business strategies. Databases, the collection of interconnected files on a server, storing information, may not deal with the same type of data, i.e. databases may be heterogeneous. As a result, many kinds of implementation and integration errors may occur in large database systems, which negatively affect the system's performance, reliability, consistency and security. Thus, it is important to test in order to obtain a database system which satisfies the ACID properties (Atomicity, Consistency, Isolation, and Durability) of a database management system. One of the most critical layers is the data access layer, which deals with databases directly during the communication process. Database testing mainly takes place at this layer and involves testing strategies such as quality control and quality assurance of the product databases. Testing at these different layers is frequently used to maintain consistency of database systems, most commonly seen in the following examples: Data is critical from a business point of view. Companies such as Google or Symantec, who are associated with data storage, need to have a durable and consistent database system. If database operations such as insert, delete, and update are performed without testing the database for consistency first, the company risks a crash of the entire system. Some companies have different types of databases, and also different goals and missions. In order to achieve a level of functionality to meet said goals, they need to test their database system. The current approach of testing may not be sufficient in which developers formally test the databases. However, this approach is not sufficiently effective since database developers are likely to slow down the testing process due to communication gaps. A separate database testing team seems advisable.

WORKFLOW

The overall workflow in this study is described in each component will be discussed in detail later in this chapter. The general idea of this workflow is: gather useful features from multiple sources and build classifier on top of them. Collect relevant datasets containing examples of normal and abnormal events. Preprocess the data by cleaning, normalizing, and augmenting it as necessary. Split the dataset into training, validation, and test sets. Select appropriate deep learning architectures for abnormal event detection tasks. Design the architecture of the chosen models, including layers, activations, and input/output configurations. Implement the models using a deep learning framework such as TensorFlow or PyTorch. Training and Validation Workflow: Train the models using the training dataset, optimizing model parameters and hyperparameters. Validate the trained models using the validation dataset, monitoring performance metrics such as accuracy, precision, recall, and F1-score. Fine-tune the models based on validation results, adjusting architectures and hyperparameters as needed. 10 Evaluation and Testing Workflow: Evaluate the performance of the trained models using the test dataset, comparing them against baseline methods or existing approaches. Assess metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC) to measure model effectiveness.