Deep Science Publishing, 2025
https://doi.org/10.70593/978-93-7185-191-6_5

Chapter 5: ETL pipelines and SQL database management

Mohanraju Muppala

1. Introduction to ETL Pipelines

An ETL pipeline comprises a centralized SQL database in which data is consolidated and continuously processed. The database is configured to authenticate visitors and display their data within a frontend framework. In contemporary digital platforms, the role of the frontend is indispensable [1,2]. It serves as the immediate environment for visitor interaction and ensures that data is presented in a manner tailored to different user types. Moreover, it is responsible for elucidating the dynamic conditions of page storage for each visitor.

2. Real-World Use Cases of ETL

ETL serves the connections between applications and the data needed to perform their tasks by providing appropriate access to all data. It is the technology for the applications and tools that are not capable of directly connecting to databases [3-5]. The Extract step involves extracting data from the requested source. In the Transform step, the incoming data is converted into the requested format to ensure seamless integration with the target system. Finally, in the Load step, the data is transferred to the requesting application.In addition to facilitating data transfer between applications, ETL can also be used for basic data duplication between storage systems. A typical scenario involves duplicating operational data into an OLAP system to suit a specific application need. Table 2-1 presents the SQL query that operates on the OLTP system to request its data.

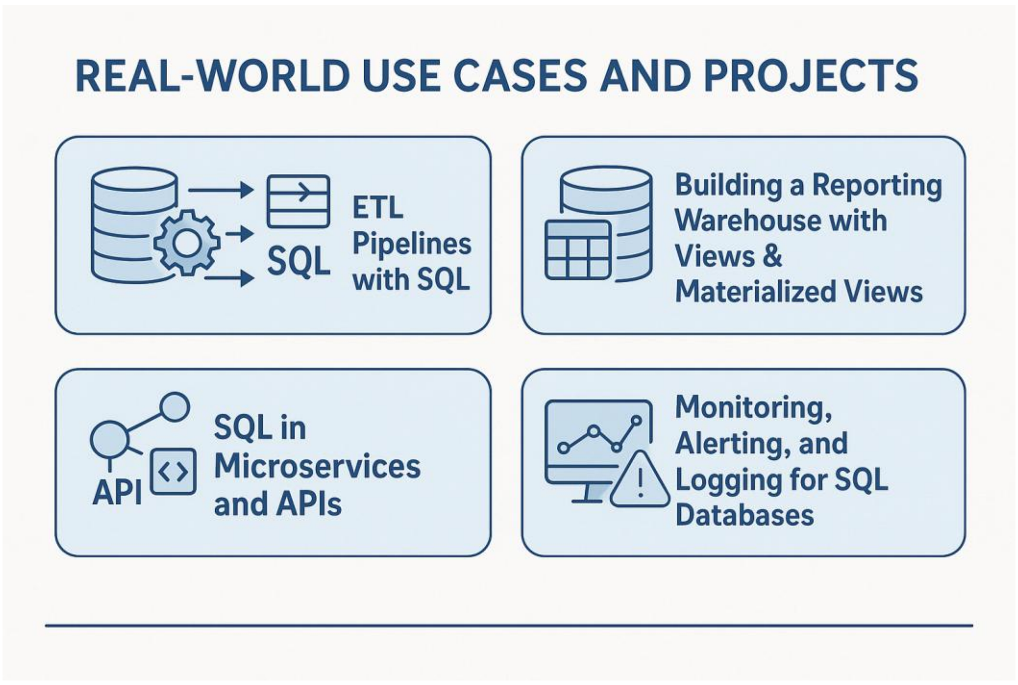
REAL-WORLD USE CASES AND PROJECTS
SQL
ETL Pipelines with SQL
Building a Reporting Warehouse with Views & Materialized Views
API
SQL in Microservices and APIs
Monitoring, Alerting, and Logging for SQL Databases

Fig1. Real world use cases

2.1. Case Study: E-Commerce Data Integration

ETL (Extract-Transform-Load) pipelines are critical for transferring data from the operational world into a database suited for analytics. This use case approaches the problem of transferring data from source files into a SQL database. The source files are taken from an e-commerce company, contained in an S3 bucket as CSV files, showing items, inventory availability, discounts and sales. The destination database contains four tables designed to hold the items' information, prices, sales and stock levels. The built pipeline can be invoked from the command line to perform an initial creation and population of the database, followed by daily updates to keep the data up-to-date.

The operational files containing the data are CSVs obtained from an S3 bucket on Amazon Web Services. The source files contain the historic record of sales, prices and availability for each item of the e-commerce company [6,7]. Four tables have been set-up in a SQL database to contain this information: an items table, a prices table, a sales table and a stock levels table. The first one stores information such as item_id, item_name and item_brand, without time dependency. The other three contain a timestamp to allow for the recording of each of the particular information mentioned. Sales contains the volume and value of the sales, prices the price and discount level for each entry and stock_levels the units available in the warehouses.

2.2. Case Study: Financial Data Aggregation

EPAM ETL Plaorm is an internal tool lately being used by fi- nancial systems. It implements several fundamental features: routing data by time, deduplication, parsing data, configuration of the pipeline us- ing Oracle DB, deploying to the EC2 infrastructure with AWS Code- Pipeline & CloudWildlife, and alerting. As a case study, a deployme- nt of such an internal system to aggregate the balance of the dcBalan- ce internal system is described. The goal is to take financial data from the jobs and transfer it to the master schema. The problem is how to turn unstructured data into a structured format, while also keeping the information so that it can be used in the future.

One solution is an ETL pipeline to parse those balances and trans- form them into the Oracle database with the other monetary records. There are many different services that parse and put information di- rectly into the database, so the new service should deploy on the EC2 infra. A new Oracle table is needed to configure the jobs, and YAML ci-files must be updated to be able to create them. This service processes information from the job and merges all the balances into one record.

2.3. Case Study: Healthcare Data Management

Raw healthcare data undergoes a well-planned cascading process. The preparation of a patient health record is considered as a cinematic storyline with fourfold: dramatic composition; process steps that include data extraction, transformation, loading, validation, and verification; life cycle; and design of the data lake and file storage path.

For the extraction, the patient data arrives in JSON format. An Apache Spark job parses the data to ensure that the nested part of the JSON can be handled by the SQL Database. After investigation, the doctor, hospital, and prescription information is extracted to further flatten the nested JSON and enable round-tripping for the patient health record data.

3. Building a Reporting Warehouse

The clarification begins with the data flow, starting from the source files held in a cloud object store such as Amazon S3. These are in a raw, unprocessed form and must be cleaned and transformed before analysis. Once the source data files have been cleaned and transformed, they are copied into the various subjects of the data model. Each subject contains tables that can be combined to find insights related to the subject, such as examining a customer's account history.

An extract, transform, and load (ETL) pipeline is created to perform these steps. It begins by configuring a connection to the cloud object store.

Next, the source data files are copied into a "staging" area within an SQL database. This staging area reflects the structure of the original source tables, making it easy to re-run the initial processing when new data becomes available. The records in the staging area are in a raw and unprocessed state, possibly containing missing, redundant, or inconsistent data.

These pieces are now in place. The next step is to implement the transformation that prepares the data for permanent storage in each of the subject tables. Within each subject table there are query patterns to make it easier to retrieve data from the warehouse, such as creating summary tables for quicker analysis.

3.1. Overview of Reporting Warehouses

The initial repository for raw data in the business context is often a reporting or operational data warehouse [2,8-10]. As operational data changes, updates are scheduled at suitable intervals, implemented efficiently through SQL MERGE statements. These statements integrate data of the same granularity by updating matching rows and inserting non-matching ones. Following the staging in the reporting warehouse, data flows into downstream operational warehouses that support core business functions such as order birth or traffic forecasting.

While SQL MERGE commands offer a convenient update mechanism, they are not immune to isolation anomalies. Therefore, incorporating logic to verify data congruence before issuing update commands can be prudent.

3.2. Designing Views for Data Reporting

In the SAS system, views simplify reporting because one is never required to look behind the scenes. Writing a SELECT statement requires specifying a list of column names for the SELECT clause. Omitting a column name also leaves it out of the report thereby, reducing the size and enhancing readability. Views allow the developer to restrict access to certain columns, which helps to maintain data security and privacy. They enable each user to view bond data according to an individual code-of-conduct and share the bond-acquisition bond sale cost only with those permitted to see it.

A view for report generation needs to be designed and created in such a way that the columns selected can be easily replaced with columns from a table created in the SAS ETL process. SQL scripts can then be rerun, producing the same report with fresh data. The SELECT statement can be saved within the view itself. Creating a report for each bond requires writing a SQL script with the bond name and replacing columns previously selected in the source view. The columns replaced must be the ones generated from SAS, and the data selected from the SAS ETL load table. The initial cost and sale of the bond are excluded from this process. Costs and sales of each bond can be protected by storing prices in the SAS table and omitting these columns when creating the source view.

3.3. Implementing Materialized Views

Materialized views can optimize ETL pipelines where only the delta values in a large table need to be read out at every refresh. In this kind of pipeline, a base relation already exists whose tuples need to be updated from an input relation. Typical SQL code reads out the delta values in the input and updates the base relation using the DELETE and INSERT constructs. However, when many tuples are removed and inserted, the DELETE phase may need to read out a large chunk of the relation, thus making the DELETE phase a performance bottleneck. The ULDB methodology thus concentrates the entire update activity inside the DELETE phase, while the INSERT phase remains empty. This is achieved by writing an appropriate expression for the DELETE phase. The criteria that the DELETE expression should satisfy is that any tuple u in the base relation that will not appear in the new database state must appear in the answer of the DELETE expression (so that it can be removed from the base relation).

ETL pipelines typically have local market-specific stores and central revenue accounting store. Understanding the distribution and roll-up of data is important: Extraction happens from local market store; Transform happens at the central store; Load also happens at the central store. The local store can extract only raw transactions, because the extraction often happens on the fly, in real time. However, the central store requires other summary transaction types for its operations, so that it can roll-up information from local sources. Rather than performing a local roll-up, the transformation happens in the central store.

The information is rolled-up into the specific summary transaction types for Drugs and Gaming.

4. SQL in Microservices Architecture

Data flow in a single microservice follows a defined order. Each step in the ETL pipeline relies on the successful processing of the previous step. Despite this, a failed step does not necessarily stop the entire pipeline; it quickly moves on to the next entry, allowing for later backfilling if necessary [1,11-12].

MySQL, a relational database management system, offers robust support and ensures stable database management. It stores user data such as the last comment date, the number of repositories, and the number of organizations. The data is automatically updated through the use of cron jobs.

4.1. Integration of SQL Databases in Microservices

Microservices require data management that is transparent to business logic, accomplished through RESTful APIs that expose the internal business logic. Similarly, integration with a specific data source can be made transparent to the application, encapsulating all data management in an ETL module that exposes a REST API [13-15]. An extractor module interacts with source APIs, requesting the data needed for business logic operation. The retrieved data then feeds into the microservice's business logic, which can process it either in real time or in batch.

A generic extractor can dynamically build extract requests by reading configuration parameters. Once data resides in the processing memory, the transformer and loader modules proceed [16,17]. The transformer applies necessary data transformations, while the loader manages the loading and integration of multiple data sources within the storage phase. A generic loader exploits database management services through a REST API, encapsulating SQL injections and operations such as TRUNCATE, DELETE, INSERT, UPDATE, and MERGE, in accordance with the selected source-table interaction modus operandi.

4.2. Best Practices for SQL APIs

A well designed database API provide a well defined and consistent interface between one component of a software application and another, or between the application and a database management system. A database API typically supports the full range of SQL commands, including those for creating and removing databases and database tables, as well as administrative commands for managing user access. It generally supports selecting and manipulating data, text search and advanced matching operations, querying metadata, creating tablespaces, and bulk loading of data to tables.

While these operations provide a foundation, they alone do not guarantee a straightforward or pleasant development experience. The functions and methods of any database API not only need to be defined correctly, but they should also be designed to be useful, consistent, and expressive. For example, when data is fetched from a table, the returned tuples should be simple to navigate through in order to retrieve the intended values, avoiding the use of numeric indexes or database column names. The environment in which the SQL commands are used should allow developers to write proper SQL statements in a natural way, which implies integration with the programming language and the support provided for operations such as string/number/list formatting or SQL statement composition. Lastly, it is important to protect the user from errors—whether they are common errors or mistakes could be harmful to data integrity—providing properly designed error checking and exceptions.

5. Monitoring SQL Databases

Any SQL database needs to be monitored. PostgreSQL is frequently used, and tools like pgAdmin4 allows inspecting the current state of a database. Querying the pg_stat_activity view shows currently running queries and their respective start times. The list includes other information such as the executing username, source IP address, application name, process ID, transaction ID, query state, and the timestamp of the state. Terminating unnecessary long-running queries can be performed via a SQL command.

PostgreSQL also tracks statistics for each table — including inserts, updates, reads, and deletes — which can be inspected with SQL. Executing the ANALYZE SQL command at regular intervals updates table statistics and indexes. Queries against the workload statistics catalog provide information about the number of rows read, conflicts, modification origins, locks, and table sizes. Monitoring tools like pg_stat_monitor extend these capabilities by tracking detailed executed query information, accessible through SQL.

5.1. Importance of Monitoring

Many electrical engineers view electrical circuit theory as one of the first lessons in their training. Regardless of an educational background, when starting a new role many engineer tasks remain similar; there is a requirement to analyse signals and associated data, usually in an electrical domain [12,18-20]. As the field of data science progresses, engineers can borrow standard techniques to process large volumes of time-series data; ETL pipelines (extract-transform-load) are ideal for these applications.

Many go unchanged for long periods; only one modification is required with the introduction of new KPI variables. Each table can be defined to retain a specific metric with a timestamp and the row date. The table retains modifications until personnel discover unexpected behaviour or clear new faults. The pipeline is then stopped and restarted once the necessary changes are applied. As KPIs continue to develop, this process becomes tedious: every change requires program modification, testing, and re-deployment. Writing a complex SQL query with hundreds of KPI variables is unwieldy. Furthermore, such a design generates a similar set of input tables over long operational periods, whereas staff would prefer to gain an indication of the pipeline's performance during execution, rather than after the data is written to permanent storage.

5.2. Key Metrics to Monitor

The goal of maintaining an up-to-date SQL database dictates the key metrics to monitor when building ETL pipelines [21-23]. The most common signals of failure can be broken down into Time and Row Count, namely: staleness, volume absence, volume shortage, volume surfeit, and timestamp mismatches.

Considering staleness first, a database can be classified as “up to date” or “stale” depending on the time elapsed since its last successful update. A warning or critical condition is triggered respectively when the time elapsed exceeds one or two expected update intervals. On the other hand, the warning and critical conditions for the Row Count metrics arise when the number of rows in the current update is either excessively small or large. A volume absence warning or critical condition is raised when the current update returns zero rows—a fact that simultaneously corresponds to a volume shortage.

6. Alerting Mechanisms for SQL Databases

Table store and query complexity metrics provide insight into SQL performance and aid in diagnosing performance issues. Other metrics reveal table growth rate or complexity (e.g., number of columns) and help determine whether alert conditions may result from poor modeling or escalating growth.

The following questions can help define and evaluate plausibility alerts.

What Should Be Alerted

What constitutes unusual behavior? Is a particular level of growth atypical or an implementation pattern that will eventually impact performance? The answer depends on the data being collected and the risk profile. Common indicators of problematic growth include very rapid growth, the creation of many very wide tables, and the creation of a large number of tables.

When Should an Alert Be Raised

Determining when tables must be classified as too large or wide to continue normal operation requires definitions of temporal context. Without historical data, the approach is limited to summary statistics and hard thresholds; historical data enable smoothing and the assessment of temporal trends. The key question is whether the table would remain within plausible limits if the current growth rate persisted.

The table can be normalized by the number of business processes supported by the database.

6.1. Setting Up Alerts

Selecting Failure mode: from the Failure dropdown list, choose either Alert if source data isn't imported or Alert if data warehouse table is empty. Selecting "Alert if source data isn't imported" causes alerts to be generated on source load failures. Selecting "Alert if data warehouse table is empty" causes alerts to be generated on empty warehouse table conditions.

Setting Notification Recipients: specify who gets notified by the alert by filling in the Send notification email(s) to input. Enter the email addresses of the alert recipients. Separate multiple email addresses with commas, semicolons, spaces, carriage returns, or line feeds. Alert email notifications are sent:. The SMTP server used is specified in the WD Loader.sws.config file. If you set the Failure mode to Alert if source data isn't imported, alerts will be generated on source load failures. If you set the Failure mode to Alert if data warehouse table is empty, alerts will be generated on empty warehouse table conditions.

6.2. Best Practices for Alert Management

ETL pipelines can be complex entities. They might consist of multiple jobs at once that get executed based on particular dependencies. Furthermore, even individual ETL jobs can be complex: they run for long durations and retrieve, transform, and load enormous volumes of data. Moreover, when ETL jobs have been reused and built upon by various teams over the years, their complexity increases accordingly. When an error occurs in an ETL pipeline, it is crucial to be promptly notified in order to resolve the issue. Proper alert management can thus .

During the lifecycle of ETL pipelines, it is imperative that the developer is notified about errors, final success, and job execution metrics. As the lifetime of such a pipeline stretches to months and years, notifications become important to ascertain whether the pipeline is performing as expected or not. Putting together an alert management strategy, based on the requirements described above, helps keep track of ETL jobs in a unified fashion. Managing SQL databases is also a significant concern when it comes to monitoring and bookkeeping of ETL pipelines [24,25]. Although it is possible to monitor individual jobs by sending alerts in case of success or failure, it is even better to keep track of the details in a central location so that the developer can check the job status or performance metrics at any time of the day, beyond just looking at the status on email during the periods when the jobs get triggered.

7. Logging Techniques for SQL Databases

During routine updates to SQL database tables, error hooks readily record SQL update errors. By contrast, table creation and dropping scripts for logging at the end of a run only capture errors during the start of an ETL run. A two-level logging mechanism ensures comprehensive error detection at both stages. Firstly, SQL scripts trigger standard error logging routines at the beginning of each ETL run, filtering out runs that have already been logged. Secondly, execution control sheets enable or disable SQL error logging for updates to each table during normal operations.

Combining words and numbers within the WHERE clause of a single SQL statement requires proper syntax and formatting to render the values that the statement compares. When a SQL statement without a parameter marker is prepared, the related object binds any literal value present in the SQL text. Prepared SQL statements can thus compare a field with both character and numeric values. However, if a SQL statement with a parameter marker within quotation marks is prepared, the generated object attempts to bind it to the column type. For instance, for a SQL statement such as SELECT_city_ FROM_pub_locale WHERE _pub_id = '_?,' the code specifies that the parameter marker has a type VARCHAR(1) and has a value of "11".

7.1. Types of Logging

Logging plays an important role in the daily operations of an enterprise. It enables us to track operational activities, to understand what a particular system is doing or has done, and to help with reconstructing or diagnosing problems. Since logging records operations over time, it collects very useful data that can be used also for understanding general past trends, identifying future patterns and demands, and partner or customer analytical activities. From a systems aspect, logging data can assist with auditing, perhaps even certification.

A very easy and convenient form of logging is to copy operational data to a file. Such files are easy to examine, and the file contents can be queried. This facility is also inexpensive to maintain since logging is part of the main system. As a drawback, the logging process can overload the main system. In the private sector, the entire unit may be physically co-located. This can be the case, for example, when the work performed by one or more departments is supported by an internal call center that is responsible for answering queries.

7.2. Implementing Effective Logging Strategies

The implementation of logging within an ETL pipeline allows for the tracking of important events that occur during its operation and the generation of metrics about what it's processing. The logging pipeline type is suitable for recording events that don't transform or represent the data structure loaded into the destination — typically, tables detailing the loading process per entity are loaded into distinct schemas within the target SQL database.

Logging during the construction of an ETL pipeline facilitates monitoring and tracking of crucial events as the pipeline processes data. It also enables the creation of metrics summarizing the operational aspects of the pipeline. For instance, logging stages like parsing, validating, or transforming can capture whether records adhere to expected data types or meet particular constraints. Similarly, logging during the loading phase can indicate the success status of database operations [26-28]. Stages in an ETL pipeline that do not transform data and do not provide a geometric representation often employ this type of logging for structuring the information. Typical use cases include generating audit records that detail how many records are processed in each stage, at what times, and identify those that were successfully loaded versus those that failed.

8. Challenges in ETL Processes

Extract, transform, and load (ETL) processes have been widely used for decades; however, the growth of informational needs, together with the increase in data volumes from heterogeneous sources, makes the use of traditional methodologies difficult. Furthermore, traditional solutions optimized for individual steps do not consider other phases of the process, hindering the construction of complete efficient processes that fit the transforming data flows in output. Several open-source tools have attempted to address these challenges. ETL systems are traditionally divided into four parts: extracting data, transforming data, loading data into a repository, and scheduling jobs.

The increasing volumes of data extracted from multiple sources and the need for more frequent extractions expose the characteristic fragility of scheduling systems, which do not consider dependency relationships between tables [29-31]. This lack of dependency management generates delays in executing jobs and increases the extraction time of information from databases. Moreover, the exponential expansion of database sizes complicates the refreshing process once data have been loaded into the required repository. As a result, cycle times extend, impeding the use of information in accordance with its timeliness. When SQL is used, the transformation step—considered the most important in the ETL process—is highly restricted due to the language's limitations. Finally, scheduling is a delicate aspect of ETL processes because it controls the frequency of data extraction, transformation, and loading; however, many existing schedulers are expensive and lack insight into how jobs affect each other.

8.1. Data Quality Issues

Data Quality Issues

The ETL process that populates the SQL database includes numerous transformations designed to correct data quality issues. The available online data are continually changing, and at any point in time the available data may be incomplete. Missing data cause the SQL table to be incomplete because a complete record needs all OR-AND-ZIP columns to be populated.

For some county/ZIP code combinations, Federal Reserve Economic Data (FRED) does not supply information for all requested fields. This causes the These treatment-status folders to be temporarily empty, just as when a folder has been created in anticipation of completing staging for a new month. Several CSV files contain only dashes or periods representing unavailable data, which causes some of the associated table cells to be empty. NBER use codes C15 and C16 are not fully populated with outbound summary reports, and their C-rates are filled with dashes.

8.2. Performance Bottlenecks

For the past few years, it has been possible to detect bottlenecks and optimize the performance of web applications [3,32,33]. This development is due to the broad implementation of professional tools that allow the recording of Important Application Specific Events—for example, event tracking in Google Analytics in order to know the user steps on a web page or recording calls of web services in order to see the bottlenecks on a web application's back-end. Since nearly all distributed systems and especially big data systems are based on reliable distributed file systems, these files open a visibility window into the functioning of the system. A professional—mainly developer or system administrator—can analyze the processed application-specific events with the help of simple SQL commands in a tool like Apache Hive, then detect performance bottlenecks and formulate solutions for the big data systems. Provided that a user recognizer detects the user rights correctly, the user can realize the visibility and change the system behavior in order to increase the speed and reduce their costs

The ELK stack is a state-of-the-art tool that allows users to transform, aggregate, and analyze different data types in an efficient way. Similar to Apache Hadoop, it can be used for big data issues, but it offers a lower cost of entry and a faster learning curve. Furthermore, the combination of powerful components such as Elasticsearch, Logstash, and Kibana allows for the collection, processing, storage, and visualization of data and can thus be used in many fields of business and administration.

9. Future Trends in SQL Database Management

Looking ahead, many of the processes and workflows that have long been commonplace in SQL database management and daily operations are likely to evolve or disappear over time, replaced by new database and engine architectures, abstractions, and AI-enabled business analysts. While it's possible to imagine a future in which AI knows just as much about your business domain as you do—from hiring to budgeting and financial forecasting—there are still many gaps in AI data intelligence that it is working hard to close.

The basic tasks include extracting, transforming, and loading data into a data warehouse or a transaction processing SQL database, then managing operations business as usual: writing SQL queries for auditing and controls, querying tables for real-time analytics, adding new domain tables and columns, optimizing for performance, and so on.

AI-assisted tools can now generate extract, transform, and load routines, including in-line judging and validation, query transactional or warehouse data tables, create advanced analytics intelligence, and perform enterprise business-as-usual for SQL engines.

9.1. Emerging Technologies

Data management in higher education must move beyond traditional storage and maintenance of academic and financial information. Emerging technologies and new infrastructure are necessary to support higher education research, teaching and learning, and decision-making [4,34-35].

The Automated Learning Informatics Network, an initiative of the Sloan Consortium's Information Technology in Education Special Interest Group, assesses and compares institutional-level learning data. LAK16 organizations also address evidence-based practices, decision-making, business intelligence, learning analytics and educational data mining research, project funding, and funding sources. Resources include from higher education business intelligence, international organizations, web portals, and professional organizations.

Proposed ETL pipelines address the curation and standardization of Open Data, assisting the in-depth analysis of datasets for surprising or counterintuitive correlations and comparisons by international media and analysts. Using commonplace pictograms and color-coding, analysts and media can reveal education achievement versus the expenditure in order to raise awareness among citizens and politicians.

9.2. The Role of AI and Machine Learning

Several examples of ETL tools capable of employing AI and machine learning could be named, but the following are two of the most interesting ones. One is Fivetran, a platform of data integration tools that offers automated pipelines based on machine learning, capable of adapting to changes in source systems and ensuring reliable replication in the destination. The second is IBM InfoSphere Information Server, which includes AI-powered metadata management features to enhance its data integration capabilities and simplify their management. The use of AI in ETL systems is evolving rapidly and continues to draw much attention.

After extracting and transforming the data and loading it into a database, the data is ready for use. Typically, a series of SQL requests against the database is performed to generate meaningful information in the form of reports or graphs. Data analysts generate series of requests to satisfy reporting requirements but experience limitations due to the potential complexity and large volume of the database. Users require an interface that allows them to pose questions in natural language and receive meaningful answers expressible as a series of simple SQL requests, as detailed in an example discussed in Section 7.3.

10. Conclusion

Extract-transform-load (ETL) pipelines are essential for consolidating data from multiple sources—from web and app platforms to third-party integrations–into a single data warehouse. Once in the warehouse, a series of views convert this data into business methods and properties crafted to address the precise questions answered by a business intelligence dashboard. These views transform the data from its raw, often complex form into easy-to-read tables and graphs. Their architecture supports additional materialized views that can be queried directly for real-time decision-making.

During the past five years, the SQL database has emerged as the lynchpin of the microservices architecture model. Providing all the services with a highly granular API enables every operation to proceed asynchronously, delivering results promptly while circumventing bottlenecks. No one entity can limit the database's efficiency. The four dimensions of SQL database state—

monitoring, alerting, logging, and troubleshooting—are detailed here. Executing real-world ETL projects reveals a new set of problems that warrant a fresh perspective on these state aspects. As artificial intelligence and SQL databases converge, daily operations are primed to become smarter, faster, and less demanding.

References:

[1] Walha A, Ghozzi F, Gargouri F. Data integration from traditional to big data: main features and comparisons of ETL approaches. The Journal of Supercomputing. 2024 Dec;80(19):26687-725.

[2] Dhummad S, Patel T. Advanced SQL Techniques for Efficient Data Migration: Strategies for Seamless Integration across Heterogeneous Systems. International Journal of Computer Trends and Technology.;72(12):38-50.

[3] Tanimura C. SQL for Data Analysis: Advanced Techniques for Transforming Data Into Insights. " O'Reilly Media, Inc."; 2021 Sep 9.

[4] Brunner U, Stockinger K. Valuenet: A natural language-to-sql system that learns from database information. In2021 IEEE 37th International Conference on Data Engineering (ICDE) 2021 Apr 19 (pp. 2177-2182). IEEE.

[5] Panda SP. Relational, NoSQL, and Artificial Intelligence-Integrated Database Architectures: Foundations, Cloud Platforms, and Regulatory-Compliant Systems. Deep Science Publishing; 2025 Jun 22.

[6] Lawson JG, Street DA. Detecting dirty data using SQL: Rigorous house insurance case. Journal of Accounting Education. 2021 Jun 1;55:100714.

[7] Zhang B, Ren R, Liu J, Jiang M, Ren J, Li J. SQLPsdem: A proxy-based mechanism towards detecting, locating and preventing second-order SQL injections. IEEE Transactions on Software Engineering. 2024 May 14;50(7):1807-26.

[8] Panda SP. Artificial Intelligence Across Borders: Transforming Industries Through Intelligent Innovation. Deep Science Publishing; 2025 Jun 6.

[9] Gandhi N, Patel J, Sisodiya R, Doshi N, Mishra S. A CNN-BiLSTM based approach for detection of SQL injection attacks. In2021 International conference on computational intelligence and knowledge economy (ICCIKE) 2021 Mar 17 (pp. 378-383). IEEE.

[10] Dhanaraj RK, Ramakrishnan V, Poongodi M, Krishnasamy L, Hamdi M, Kotecha K, Vijayakumar V. Random forest bagging and x-means clustered antipattern detection from SQL query log for accessing secure mobile data. Wireless communications and mobile computing. 2021;2021(1):2730246.

[11] Panda SP, Muppala M, Koneti SB. The Contribution of AI in Climate Modeling and Sustainable Decision-Making. Available at SSRN 5283619. 2025 Jun 1.

[12] Shivadekar S. Artificial Intelligence for Cognitive Systems: Deep Learning, Neuro-symbolic Integration, and Human-Centric Intelligence. Deep Science Publishing; 2025 Jun 30.

[13] Davidson L. Pro SQL Server Relational Database Design and Implementation: Best Practices for Scalability and Performance. Apress; 2021.

[14] Zhang W, Li Y, Li X, Shao M, Mi Y, Zhang H, Zhi G. Deep Neural Network-Based SQL Injection Detection Method. Security and Communication Networks. 2022;2022(1):4836289.

[15] Roy P, Kumar R, Rani P. SQL injection attack detection by machine learning classifier. In2022 International conference on applied artificial intelligence and computing (ICAAIC) 2022 May 9 (pp. 394-400). IEEE.

[16] Katsogiannis-Meimarakis G, Koutrika G. A survey on deep learning approaches for text-to-SQL. The VLDB Journal. 2023 Jul;32(4):905-36.

[17] Khan W, Kumar T, Zhang C, Raj K, Roy AM, Luo B. SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review. Big Data and Cognitive Computing. 2023 May 12;7(2):97.

[18] Hong Z, Yuan Z, Zhang Q, Chen H, Dong J, Huang F, Huang X. Next-generation database interfaces: A survey of llm-based text-to-sql. arXiv preprint arXiv:2406.08426. 2024 Jun 12.

[19] Islam S. Future trends in SQL databases and big data analytics: Impact of machine learning and artificial intelligence. Available at SSRN 5064781. 2024 Aug 6.

[20] de Oliveira VF, Pessoa MA, Junqueira F, Miyagi PE. SQL and NoSQL Databases in the Context of Industry 4.0. Machines. 2021 Dec 27;10(1):20.

[21] Rockoff L. The language of SQL. Addison-Wesley Professional; 2021 Nov 4.

[22] Shivadekar S, Halem M, Yeah Y, Vibhute S. Edge AI cosmos blockchain distributed network for precise ablh detection. Multimedia tools and applications. 2024 Aug;83(27):69083-109.

[23] Fotache M, Munteanu A, Strîmbei C, Hrubaru I. Framework for the assessment of data masking performance penalties in SQL database servers. Case Study: Oracle. IEEE Access. 2023 Feb 22;11:18520-41.

[24] Panda SP. Augmented and Virtual Reality in Intelligent Systems. Available at SSRN. 2021 Apr 16.

[25] Karwin B. SQL Antipatterns, Volume 1: Avoiding the Pitfalls of Database Programming. The Pragmatic Programmers LLC; 2022 Oct 24.

[26] Nasereddin M, ALKhamaiseh A, Qasaimeh M, Al-Qassas R. A systematic review of detection and prevention techniques of SQL injection attacks. Information Security Journal: A Global Perspective. 2023 Jul 4;32(4):252-65.

[27] Chakraborty S, Aithal PS. CRUD Operation on WordPress Database Using C# SQL Client. International Journal of Case Studies in Business, IT, and Education (IJCSBE). 2023 Nov 28;7(4):138-49.

[28] Panda SP. The Evolution and Defense Against Social Engineering and Phishing Attacks. International Journal of Science and Research (IJSR). 2025 Jan 1.

[29] Choi H, Lee S, Jeong D. Forensic recovery of SQL server database: Practical approach. IEEE Access. 2021 Jan 18;9:14564-75.

[30] Thalji N, Raza A, Islam MS, Samee NA, Jamjoom MM. Ae-net: Novel autoencoder-based deep features for sql injection attack detection. IEEE access. 2023 Nov 28;11:135507-16.

[31] Chakraborty S, Paul S, Hasan KA. Performance comparison for data retrieval from nosql and sql databases: a case study for covid-19 genome sequence dataset. In2021 2nd International Conference on Robotics, electrical and signal processing techniques (ICREST) 2021 Jan 5 (pp. 324-328). IEEE.

[32] Crespo-Martínez IS, Campazas-Vega A, Guerrero-Higueras ÁM, Riego-DelCastillo V, Álvarez-Aparicio C, Fernández-Llamas C. SQL injection attack detection in network flow data. Computers & Security. 2023 Apr 1;127:103093.

[33] Antas J, Rocha Silva R, Bernardino J. Assessment of SQL and NoSQL systems to store and mine COVID-19 data. Computers. 2022 Feb 21;11(2):29.

[34] Shivadekar S, Kataria DB, Hundekar S, Wanjale K, Balpande VP, Suryawanshi R. Deep learning based image classification of lungs radiography for detecting covid-19 using a deep cnn and resnet 50. International Journal of Intelligent Systems and Applications in Engineering. 2023;11:241-50.

[35] Ashlam AA, Badii A, Stahl F. Multi-phase algorithmic framework to prevent SQL injection attacks using improved machine learning and deep learning to enhance database security in real-time. In2022 15th International Conference on Security of Information and Networks (SIN) 2022 Nov 11 (pp. 01-04). IEEE.