Database Requirements for Modern Development

By Gerald Venzl, Oracle

August 2017

Introduction

In the application development community, the function of database technology is well understood. Simply put, databases (also known as data stores) are used to store and retrieve data needed for application processing. Even if there isn't a data store in your initial architectural diagram, data flowing through your application will likely touch one or more databases, sooner or later.

Database software has been widely deployed over a long period of time. First introduced in the 1970s, modern database systems differ greatly from original implementations that evolved from data banks to database systems. Today, the evolution of database technology continues. While every developer knows that databases are useful for managing data, modern day development techniques and technologies require that databases provide much more than just simple SQL statements that can read and write data to disk. In fact, modern database technology must provide three core capabilities to be truly useful to developers:

- Widespread programming language support
- The ability to manage multiple data structures
- Transparent data analysis capabilities across all data structures using SQL

In selecting a database, it's important to consider how these capabilities can impact application development, and the quality, simplicity, and productivity that they can bring to your development projects.

Widespread Programming Language Support

Developers can choose from an ever-increasing number of available programming languages and frameworks. To be useful to developers, given their programming language of choice, modern databases must integrate with as many languages and frameworks as possible. To simplify integration efforts, languages frequently feature standard APIs that can interact directly with databases—such an API implementation is commonly referred to as a database driver. Well-known examples are the Java Database Connectivity (JDBC) API for Java, the ADO.NET framework for Microsoft .NET applications, and the PEP 249—Python Database API specification for the Python scripting language, to name a few.

Almost every programming language has either its own interface specification or can interact with a canonical API that allows the language to talk to a database on the other end. Common canonical APIs are the Open Database Connectivity API (ODBC) and Representational State Transfer (REST) API. The use of REST APIs has mushroomed in the last couple of years, especially for web application development. Although REST was first introduced in 2000, today's architectural designs—such as microservices or stateless cross-web communication in general—make REST a vital component in the toolbox of virtually all modern-day developers.

Providing direct access from within the application code via APIs is one way to achieve database integration. However, some modern databases support hiding database interaction entirely behind a REST call, abstracting the task of data access altogether, which helps to improve developer productivity. This concept is nothing new. Many programming languages have so-called Object-Relational Mapping (ORM) frameworks that translate between the object-oriented data structures in programs and the relational model of databases. Such frameworks usually represent themselves as libraries to the developer that are used to abstract data interaction from the application logic. While abstracting the data access layer in this way can boost developer productivity, the task of managing the abstraction is still the developer's responsibility.

Oracle Supported Programming Languages

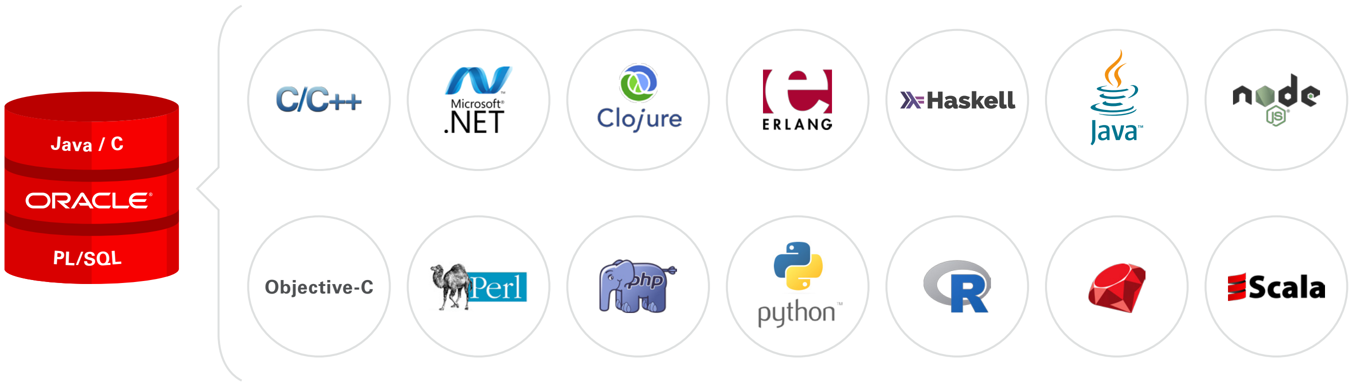

Figure 1: Popular programming languages interface easily with different database technologies from Oracle.

REST-based access, on the other hand, also moves the management task out of the developer's remit, enabling even greater productivity gains.

Oracle supports a large and varied development community (both internally and externally), so the company recognizes the advantages of broad language support. Today, more than 30 programming languages, including the popular languages shown in Figure 1, can access the various database technologies that Oracle provides. In addition, Oracle actively participates in industry-wide efforts to refine standards for database interfaces, including JDBC, Python PEP 249, and PHP, among others.

Oracle anticipated the popularity of REST and the emerging trend of data access abstraction via REST. In 2014, Oracle introduced a database component that allows RESTful access to data stored in a relational database, document store, or key-value format, making Oracle a pioneer in providing standardized REST interfaces for the data access layer. The component is known as Oracle REST Data Services (ORDS). You can find out more about Oracle REST Data Services here.

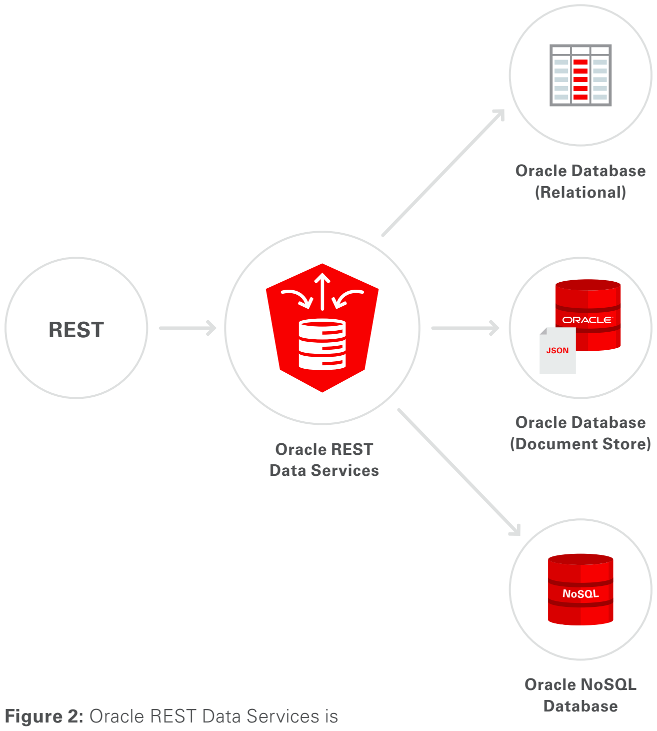
Oracle Database (Relational)
REST
Oracle REST Data Services
ORACLE
JSON
Oracle Database (Document Store)
NoSQL
Oracle NoSQL Database

Figure 2: Oracle REST Data Services is included with both Oracle Database and Oracle SQL Developer installations. It is supported in Weblogic, Tomcat, Glassfish, and as a standalone application running Jetty in embedded mode.

Multi-Model Persistence

Beyond support for a wide range of programming languages, it's critical that a database technology provides easy-to-use capabilities for general-purpose access and data management. Such capabilities are a fundamental strength for databases that organize data using a relational model.

Around 1970, the relational model was conceived to protect database users from having to know how data is explicitly organized within a database. It is this principal that has allowed the relational model to be the most widely used general-purpose data structure for more than 40 years. Given its long history, the relational model is very well understood, and it is still widely selected for development projects today. Most of the time, the relational model is a good fit for modern-day applications, but there are some use cases in which the relational model may not be the optimal fit. For these use cases, other data models—such as data stores based on documents, graphs, or key-value pairs—are often better suited. This has led to a new design pattern known as polyglot persistence, which is the concept of applying different data models to address different data storage requirements, even within the same application. Document-oriented databases, or document stores, are especially popular among developers today. A document database is one that typically stores JSON-based data structures. (Again, the concept is not new. XML databases, a subclass of document databases, were introduced in the early 2000s.) A document data structure is self-describing, meaning that the data is structured by attributes and values that can be hierarchically structured within the same document. Because both the schema and data reside within a document, they offer an advantage in that the document structure can be changed at any given point in time—a schema change within the document has no impact anywhere outside of the document. This characteristic is referred to as schema-on-read, while a relational model is known as schema-on-write.

With the schema-on-read approach, you are not required to design the data model before storing the data; you simply write the information to the data store. In contrast, with schema-on-write, you must predefine the data model before loading data into the relational structure. Thus, a schema-on-read approach can help to enhance developer productivity since there's no need to construct the data model first.

However, when it comes to exchanging information between separate systems, schema-on-read provides a degree of autonomy with respect to how data is consumed, and one system can use a different data interpretation than another. While this provides some degree of flexibility, it can also result in added complexity for data analysis, pointing to a potential downside of using the document model. Given that the schema is contained within the data structure itself, different interpretations of the same data can occur. For example, one document can refer to an email address with the attribute "emailAddress", while another document may refer to it as "email_address". Both attributes contain the same information but do not guarantee that the information is interpreted in that way.

As a result, some document databases now add support for schema validation, helping to make sure that attributes are presented correctly. If a document database does not offer this capability, then it is the developer's responsibility to guarantee that the data is interpreted correctly and prevent the return of an inconsistent answer to the user.

Modern-day databases are often multi-model—that is, they give developers the choice of the desired data model without having to worry about which database to use and how to connect to it (databases that support only a single data model are known as single-model databases). Multi-model databases provide developers with a single connection method and a common API for storing and retrieving data, regardless of the data format. Data storage and retrieval is usually performed via standard SQL operations that provide the desired transparency. By using standard SQL operations, a developer can immediately leverage a multi-model database without having to refactor code or interact with a different set of APIs.

Single-model databases are optimally suited for use cases that are well-understood from the beginning, where there is little probability of changing requirements that would alter the data format. In addition, single-model databases typically provide special optimizations for the data format being managed, which is why it can be more difficult for a multi-model database to support different data formats as efficiently.

Oracle provides both single-model and multi-model database technologies. Single-model databases from Oracle include (although not exclusively) Berkeley DB, Berkeley DB XML Data Store, Oracle NoSQL Database, and Oracle Essbase. Other databases from Oracle can manage multiple data models (Figure 4). Oracle Database 12c, for example, supports multi-model database capabilities, enabling scalable, high performance data management and analysis (for more information, see "Multimodel Database with Oracle Database 12c Release 2")

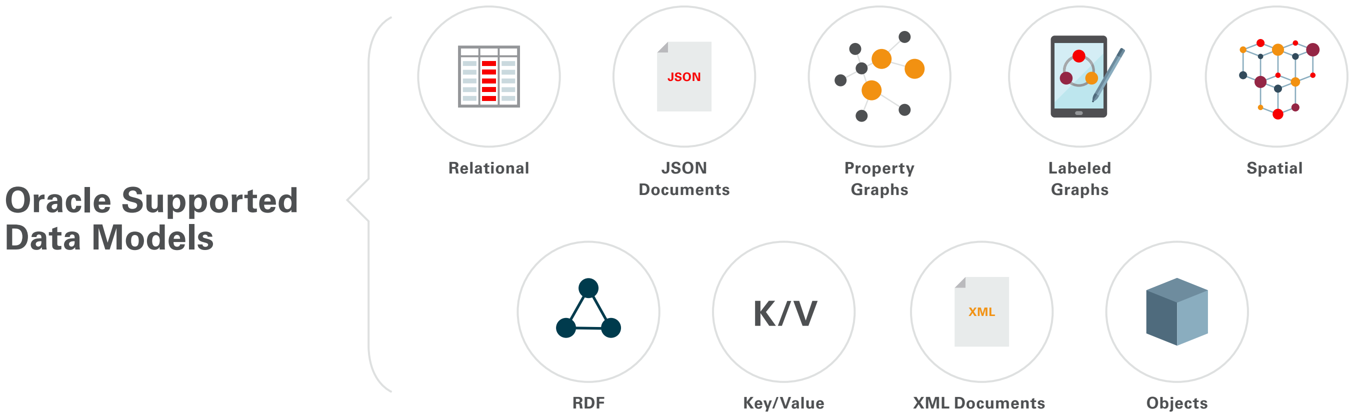

Figure 3: Oracle offers database technologies that can manage multiple data models.

Data Access and Analysis via SQL

The true power of databases is not how fast they can store a piece of data onto a disk or in which model the data is stored. Instead, the true power of databases lies within the retrieval and analysis of the data. One of the most important success factors of a database (if not the most important success factor) is the support of a simple-to-understand, universal query language that can be applied over all data. Today, the de facto industry standard is SQL (Structured Query Language). SQL allows developers to interact with databases while providing the same capability to business users, allowing them to extract value from the data without writing lines and lines of code.

The advent of NoSQL databases seemingly calls into question the usefulness of SQL. However, every mainstream NoSQL database vendor has either adopted SQL or an SQL-like engine to interact with its database, validating the requirement for SQL language support.

Developer surveys reinforce the necessity of offering SQL support, showing year over year that SQL is one of the most popular languages in daily use by developers. Most developers throughout the world are familiar with SQL, and those who are not can typically pick up the basics quickly because the syntax is English language-based.

While SQL's basic functions concern data storage and retrieval, the language is much more powerful than that. SQL provides a rich set of analytical capabilities to users, whether that user is a developer, data scientist, or business analyst. SQL supports running complex computations against data, and runs them as close as possible to the data. Databases that do not provide SQL (or that only provide marginal SQL support) force developers to deal with complex business data analysis requirements directly in application code, decreasing developer productivity. Indeed, using SQL allows developers to respond to analytical changes in business requirements more quickly and efficiently, so proficiency with SQL is a desirable programming skill.

Oracle sees SQL as an important component of modern-day data management, and actively participates in the ISO SQL standard committee. The SQL language continues to evolve since its original release more than 30 years ago, with the latest SQL:2016 standard revision released in December 2016. Because other databases only provide support for the standard's SQL-92 or SQL:1999 revisions, they may not include important analytical capabilities that have been added over time.

SQL Integration

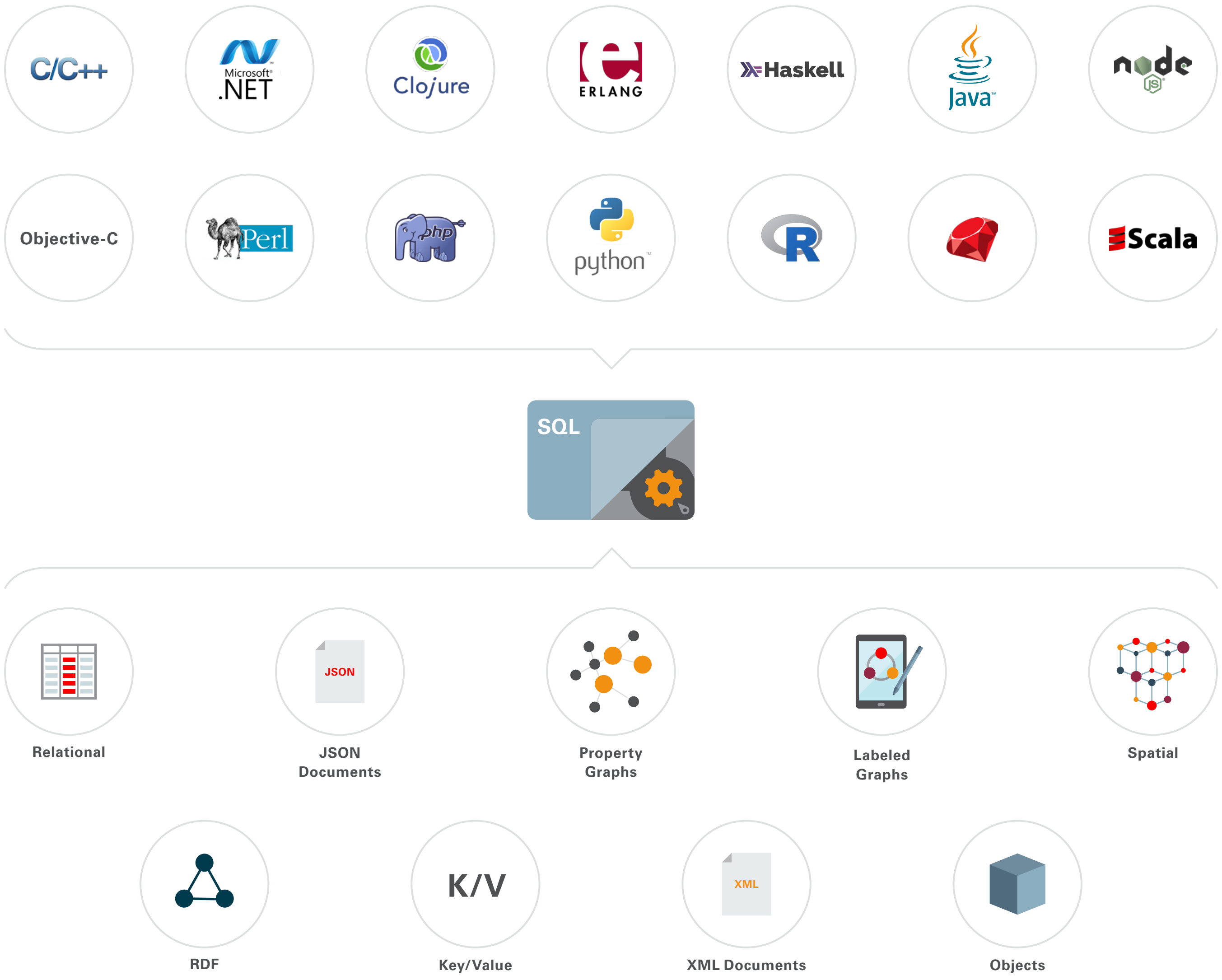

Figure 4: Oracle SQL is integrated across the different data models and accessible from all programming languages.

Conclusion

As highlighted in this paper, it is paramount for modern databases to provide wide spread programming language support and the ability to manage multiple data structures and transparent data analysis capabilities across all those data structures. With all of those combined, a database is best fit for managing modern day requirements that go beyond the original tasks of a relational database and will provide a rock solid data management solution as compared to just a mere data storage solution.

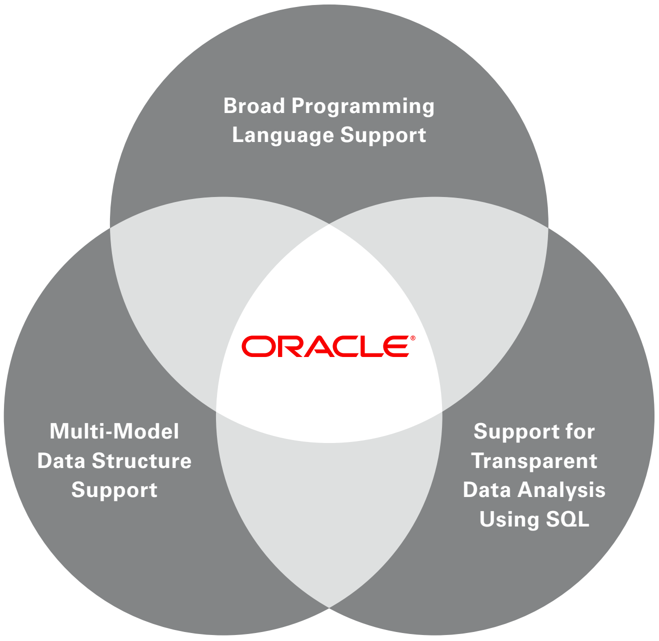
Broad Programming Language Support
ORACLE
Multi-Model Data Structure Support
Support for Transparent Data Analysis Using SQL

Figure 5: Modern development requirements for databases.

Try Oracle Cloud for Free

Get Oracle Cloud now > **Visit developer.oracle.com >**