2.1 Comparison to Alternative Approaches

Traditional approaches to redacting sensitive data typically relied on application coding or installing third-party software on the database server to modify its behavior. These alternatives have important drawbacks compared to Data Redaction. Approaches that require coding new application logic, modifying existing SQL statements, or authoring custom application scripts are likely to result in disparate solutions that are inconsistent across the enterprise and costly to maintain over their lifetime. In addition, strict controls must be placed on new application development to make sure that custom application code and new objects are properly accessed.

The code also needs to take into consideration multiple factors under which the redaction policies are enforced, while maintaining the performance and semantics of the application. Approaches that add new components to the Oracle Database, overwrite existing components, establish proxies, and modify basic behavior of the database also are fraught with problems [11]. Not only does the new component introduce new attack surfaces, but they also can create performance overhead, impact operational activities of the database, and may fail when attempting to transform complex database queries that are generated by applications.

Approaches that add new components to the Oracle Database, overwrite existing components, establish proxies, and modify basic behavior of the database also are fraught with In contrast, redacting directly in the Oracle Database kernel using Data Redaction has tighter security, superior performance, and better compatibility with a range of database configurations, use cases, and workloads.
2.2 Related work

Multi-tenancy was by no means a new concept, and the initial work was done in the context of Application Service Providers (ASP) in late 90s and early 2000s. However, the topic received much attention when Cloud computing provided concrete use cases. One of the first discussions that made a significant effect on multi-tenancy research was a maturity model for SaaS applications. Chong et al [9] formulated a four-level SaaS maturity model to be employed to determine the configurability, multi-tenant efficiency, and scalability of a SaaS application. Level 1 is the traditional ASP model where each tenant has a separate SaaS application. In level 2, the configurable instances of the same application are delivered to different tenants. Level 3 brings in multi-tenant efficiency to a SaaS application that is a single instance serves every tenant, and separate metadata is used to tailor the application to the needs of each tenant. Level 4 extends the level 3 by making a SaaS application scalable where the SaaS vendor runs the multiple instances of a SaaS application and load balances to scale it up.

Their proposed multi-tenant BPS can be scaled as a cluster and it can be placed at the level 4. Furthermore, a business process deployed in a server instance can be configured and monitored through their multi-tenant aware management portal.

Most of the initial work on multi-tenancy was done in data multi-tenancy. Both Jacobs et al. [42] and Chong [43] have outlined three main approaches for data management in a multi-tenant environment: separate databases, shared database with separate schemas, and shared database with shared schemas. They maintained that the lesser the isolation, the higher the scalability and sharing.
A lot of research has been done to find out ways of implementation of multi-tenancy in SaaS applications. A business based on SaaS application will be successful only if it supports multi-tenancy. Multi-tenancy is an organization approach for SaaS application. Key characteristics of multi-tenancy can be given as 1. Hardware and resource sharing, 2. High degree of reconfigurability, 3. Shared application and database instance [9]. Jinan Fiaidhi et al. [10] proposed general multitenant cloud architecture. Here author has written about managing data of multitenant SaaS application. He specified various approaches as follows

1. Storing tenant data in separate databases

2. Housing multiple tenants in the same database, with tenant specific schema, and

3. Using the same database and a same set of tables to host multiple tenants.