EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

DBOD
DATABASE ON DEMAND

Summer Student Report

Identify threat model and protection offered by different encryption possibilities in MySQL, PostgreSQL and Oracle

Author:
Guilherme Boneti
IT-DA-DB

Supervisors:
Maurizio De Giorgi
Abel Cabezas Alonso

Abstract

This report presents a preliminary assessment of the threats and vulnerabilities encountered in database technology, focusing on encryption solutions available for MySQL, PostgreSQL, and Oracle. Specifically, this research investigates *encryption-at-rest* and *transparent data encryption*. The primary objective is to provide insights into the diverse encryption options and mitigation strategies offered by these database management systems (DBMS). The document details experiments conducted to compare the effectiveness of these solutions. Additionally, the study aims to offer recommendations for a preliminary actionable plan, which can serve as a foundational framework for a comprehensive security strategy tailored to the unique environment and infrastructure of CERN. This research is important in enhancing the security posture of CERN's database systems and ensuring the confidentiality and integrity of sensitive data.

September 29, 2023

Contents

1 Introduction

In today's digital age, data has become as valuable as gold, and its protection is paramount. Storing data securely is essential to prevent unauthorized access and potential data breaches. This challenge becomes even more critical for organizations like CERN, which handle vast amounts of sensitive and confidential data. The exposure of such data can have severe repercussions, not only for CERN but also for its collaborators and stakeholders.

In response to these challenges, encryption emerges as a fundamental solution to mitigate the risks associated with data theft. When data is stored securely through encryption, even if malicious actors gain unauthorized access to a database or storage device, the data remains inaccessible and unusable to them. This protection ensures the confidentiality and integrity of sensitive information, safeguarding both CERN's interests and the privacy of its users.

In this work, we explore different encryption techniques and solutions, with a particular focus on the offerings provided by the database management systems MySQL, PostgreSQL, and Oracle. Our study includes a performance comparison to assist CERN in making informed decisions regarding its infrastructure.

Through this exploration, we aim to provide valuable insights into the significance of data encryption and the diverse options available, ultimately aiding CERN in enhancing its data security measures.

2 Threats and Mitigation Approaches

In the field of data security, our research has identified three main areas of concern: the storage device, client applications, and databases. Each of these parts faces various threats and associated risks. The storage device, for example, could be stolen, which would result in potential data exposure. Client applications may be at risk from issues like malware, which can infect devices and compromise data, phishing, where deceptive tactics are used to steal sensitive information, and ransomware, a type of malicious software that can lock users out of their own systems until a ransom is paid. Databases, in turn, may be vulnerable to problems like backup exposures, where unauthorized access to backups could lead to data breaches, credential theft, which involves someone taking login credentials without permission, SQL injection, a type of cyber attack that can compromise database security, and denial-of-service (DoS) attacks that can disrupt database availability. To address these risks, we use different methods. Disk encryption safeguards the storage device,

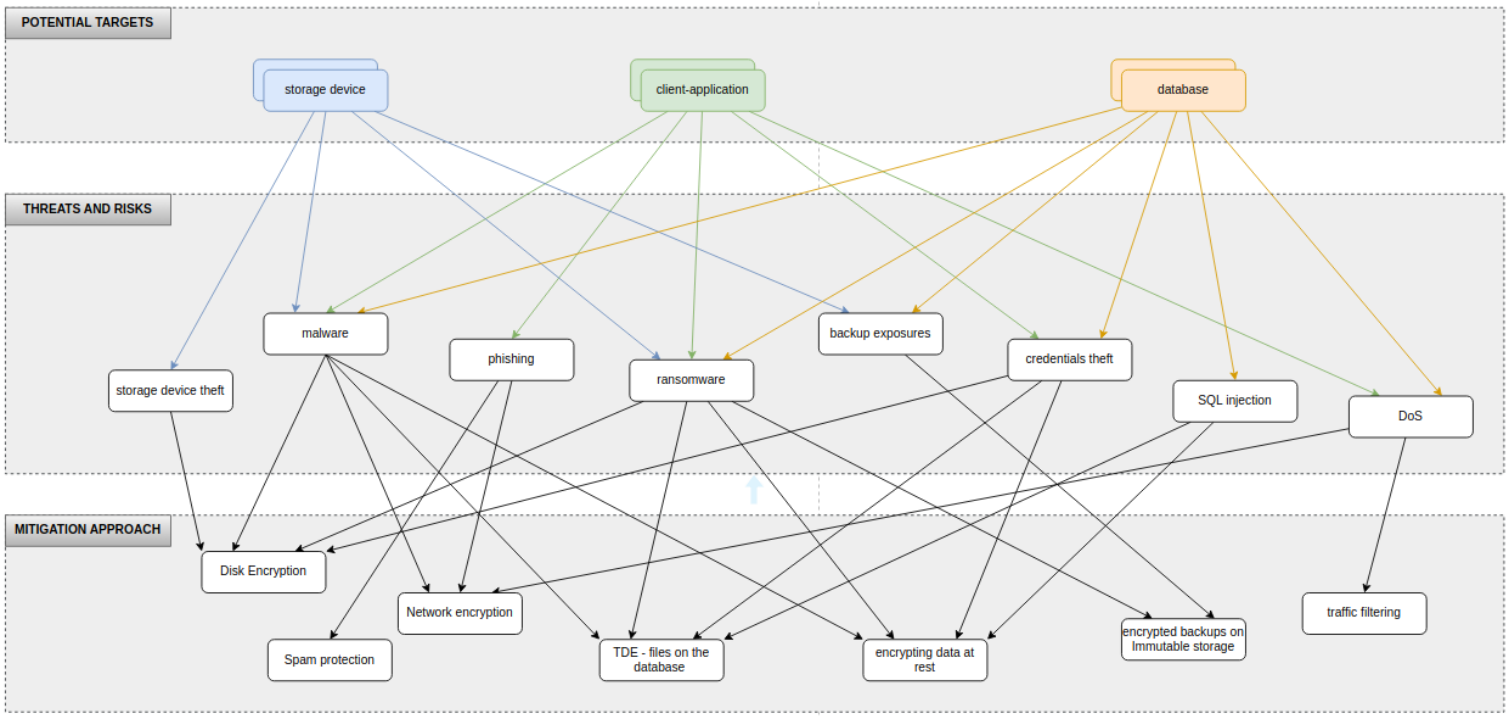

Figure 1: Threats and mitigation approaches

while spam protection helps counter phishing and malware threats. Network encryption enhances the security of client applications, making data transfer more secure. Transparent Data Encryption (TDE) and encrypting data-at-rest strengthen database security. Using encrypted backups stored on immutable storage can prevent backup exposures, and traffic filtering can help mitigate DoS attacks. However, our research focuses mainly on databases as a primary target, exploring the associated risks and solutions, with a specific emphasis on data-at-rest and Transparent Data Encryption (TDE) in the upcoming chapters.

3 Data Encryption

3.1 Types of data

Before diving into the details of data encryption, it is essential to understand the nature of the data that we aim to secure. Data can be categorized into two primary types:

3.1.1 Data-at-Rest

Data-at-rest refers to information that is stored in a storage device and is currently not being accessed or used. This kind of information can face risks from cybercriminals and other harmful activities aimed at either digitally breaking into the data or physically stealing the storage devices containing it. Encrypting data-at-rest, therefore, is a security measure that can mitigate the threats in case of data theft.

3.1.2 Data-in-Motion

Data-in-motion, on the other hand, is data that are actively on the move or being sent over a network. It's the data in transit, including information sent or received by various apps and services. Data-in-motion encryption makes sure that data is protected while it's traveling from one place to another, so it can't be seen or intercepted by unauthorized folks. Common ways to encrypt data-in-motion include using SSL/TLS for secure web connections and VPNs for safe network communication.

While data-in-motion encryption is certainly important, we're going to give most of our attention to data-at-rest encryption in this report. By focusing on data-at-rest encryption, we're tackling the fundamental aspect of data security, creating a strong base for overall protection against unauthorized access and potential data breaches.

3.2 Transparent Data Encryption

TDE is a framework for encrypting data-at-rest. It encrypts databases both on the hard drive and consequently on backup media. It is transparent since the user doesn't notice the encryption. The data is stored encrypted and is automatically decrypted by TDE for authorized users and applications. Therefore, no changes are required in the application side. TDE ensures that sensitive data is encrypted, meets compliance requirements, and provides functionality that streamlines encryption operations.

In the upcoming chapters, we'll dig into the idea of Transparent Data Encryption (TDE). We'll explore the main principles, the keys and key management involved, what parts of the data can be encrypted, approaches for enabling TDE, and how it affects the performance. This exploration will help us understand how data-at-rest encryption is achieved and provide a basis for a comparative analysis between the different implementations in MySQL, PostgreSQL, and Oracle databases.

4 Encryption Architecture

The structure of data encryption plays a critical role in ensuring data security and manageability. It involves various aspects, such as how encryption keys are organized, where they are stored, and how encryption is applied to different parts of a database. In this section, we will explore the essential elements of encryption architecture.

4.1 Encryption Model

Encryption architectures are often categorized into two primary models: the 1-tier approach and the 2-tier approach.

4.1.1 1-Tier Approach

In a 1-tier encryption model, a single encryption key is employed to encrypt all the data. This approach is relatively straightforward and suits scenarios where simplicity and ease of management are priorities.

4.1.2 2-Tier Approach

The 2-tier approach involves a clear separation of duties between two types of encryption keys: the master encryption key and the data encryption keys. In this model, the master encryption key is responsible for encrypting and decrypting the data encryption keys. On the other hand, the data encryption keys are used for encrypting and decrypting the actual data. This segregation enhances security significantly. Even if an attacker gains access to a data encryption key, such as one used to decrypt a specific table, without access to the master key, they cannot decrypt the data encryption key itself. As a result, the data encryption key becomes useless to the attacker, preventing the access to the data.

4.2 Key Management

Encryption keys must be stored securely to prevent unauthorized access. The master key storage can be managed either internally or externally, depending on the chosen encryption solution.

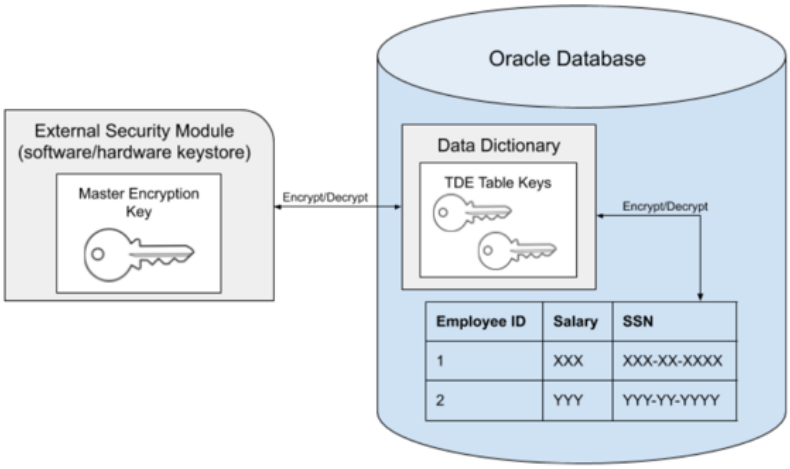

Figure 2: Master Key implementation

- **Internal Key Storage:** Some encryption solutions store keys within the database system itself. While this approach simplifies key management, it may pose security risks if the database system is compromised.
- **External Key Storage:** Alternatively, organizations can opt for external key management solutions. These solutions store keys outside of the database system, often in specialized hardware or dedicated key management platforms, such as the Oracle Key Vault, Thales CipherTrust Manager, Fornetix VaultCore, Google Cloud Key Management Service, and others. External key storage enhances security by isolating keys from potential database breaches.

4.3 Master Key Rotation

Key rotation is a security practice that involves periodically changing encryption keys. It is particularly relevant for master encryption keys in the 2-tier encryption model. Regularly rotating master keys strengthens security by limiting the exposure of keys and preventing long-term vulnerabilities.

4.4 Encryption Targets

Encryption can be applied to various components of a database. Common encryption targets include:

- **Cluster**: Encryption at the cluster level provides overarching security for the entire database cluster, ensuring that all data within it is protected.

- **Database**: Database-level encryption extends security to the entire database, encompassing all tables, tablespaces, and data within.
- **Tablespaces**: Encryption can be selectively applied to tablespaces, which are logical storage containers within a database.
- **Tables**: Organizations may choose to encrypt specific tables that contain highly sensitive data.
- **Columns**: For fine-grained control, encryption can be applied at the column level, allowing individual columns within tables to be encrypted.

4.4.1 Tablespace and column encryption

Differences in the implementation of encryption in tablespace and column levels may lead to an impact on performance. Below, Table 1 compares the two implementations.

Tablespace	**Column**
Encrypt all data files belonging to the tablespace	Encrypt individual columns
Tablespace key (same for all data files in a tablespace)	Table key (same for all columns in a table)
Default algorithm: AES 128	Default algorithm: AES 192
Data is decrypted as it is read from data files into the buffer cache or via direct-path read	Data remains encrypted in the buffer cache
Performance impact tends to be consistent and does not change the execution plan for queries	Performance impact varies significantly depending on the query execution plan

Table 1: Tablespace and column encryption comparison

The database buffer cache handles most of the query tasks. Since the data in the buffer cache is already decrypted with tablespace encryption, there's usually no need for additional decryption during query operations. On the other hand, with column encryption, data is encrypted within the buffer cache. This means that each data request necessitates decryption, increasing performance overhead.

5 Methods for data encryption

Working with large amounts of data requires techniques for migrating the data from one place to another. We can have two main cases when encrypting our data:

1. **Reorganizing Data into an Encrypted Instance**: When implementing data-at-rest encryption, one common approach is to reorganize the existing data into an encrypted instance. In this case, the data is initially stored without encryption, and then it is transformed into an encrypted state within a dedicated instance.

    - **Dump and Restore:** This approach involves exporting the unencrypted data from its source and then importing it into the target encrypted instance. Tools like `pg_dump` for PostgreSQL or `mysqldump` for MySQL can be used for this purpose. The data is effectively converted into an encrypted format during this process.
    - **Logical Replication:** Another method is to utilize logical replication, which is a feature provided by some database management systems like PostgreSQL. Logical replication allows you to replicate data from a source to a target database while applying encryption transformations as needed. This can be particularly useful for real-time data replication with encryption.

2. **Converting an Existing Instance:** When you already have an existing database instance that is unencrypted and you want to enable encryption for it, you can employ one of the following approaches:

    - **Offline Conversion:** This method involves taking the database or tablespace offline and performing the encryption operation. This approach typically requires some downtime, as the database needs to be taken offline for the conversion to take place.
    - **Online Conversion:** In contrast to offline conversion, online conversion allows you to enable encryption for an existing database or tablespace without the need for significant downtime. The encryption process occurs in the background while the database remains operational in read-write mode. This method can be more convenient for environments where continuous availability is crucial.

5.1 Online vs offline encryption

When choosing between offline or online encryption conversion, one must consider several critical factors. In Table 2 below, we list some of these factors.

Therefore, offline encryption offers quicker conversion and requires no additional storage space. However, it comes at the cost of downtime, which can disrupt operations. Conversely, online encryption, while avoiding downtime, introduces the need for extra storage and potential performance impacts.

Offline	**Online**
Requires taking the instance offline	Encrypts instance in background with no downtime
No need for additional storage space	Storage overhead is equal to size of the largest data file
Master key cannot be rotated	Supports live data key rotation
Quicker conversion	Slower conversion

Table 2: Offline and online encryption comparison

6 Available Encryption Implementations

6.1 MySQL encryption options

There are two possibilities for using encryption in MySQL databases: MySQL InnoDB [7] and MySQL Enterprise Edition [8].

6.1.1 MySQL InnoDB

MySQL InnoDB offers a free and robust data-at-rest encryption solution. Key highlights of this option include:

- Support for data-at-rest encryption.
- Implementation of a 2-tier encryption model.
- No noticeable performance overhead.
- User has to rely on a keyring plugin for key management.
- Storage of keyring data in a file local to the server host.
- Encryption capabilities at both the table and tablespace levels.
- Support for master-key rotation.
- Use the AES-256 encryption algorithm.

6.1.2 MySQL Enterprise Edition

MySQL Enterprise Edition is a paid solution offering enhanced security and advanced encryption features. Key aspects of this option include:

- Implementation of a 2-tier encryption model.
- Utilizes a centralized key management solution, supporting several key vaults.
- No noticeable performance overhead.
- Supports data-at-rest encryption at both the table and tablespace levels.
- Offers master-key rotation.
- Utilizes the AES-256 encryption algorithm.

6.2 PostgreSQL encryption options

For PostgreSQL, the available options are the pgsql-hackers TDE [16], the Cybertec PostgreSQL TDE [12], and the EnterpriseDB PostgreSQL TDE [9].

6.2.1 Pgsql-hackers TDE

The PostgreSQL community, often referred to as pgsql-hackers, has developed two patches of TDE solutions, the last one being released in 2019. This option is a free and open-source solution with several notable features:

- Support for multiple AES encryption key lengths, including 128-bit, 192-bit, and 256-bit.
- Offers both single key and 2-tier key architecture options for encryption.
- Provides the capability to encrypt at the cluster, tablespace, and table levels.
- Supports key rotation.
- Employs an internal key management system that stores encryption keys in the database.
- Allows communication with external key management systems to further enhance key security.

6.2.2 Cybertech Postgres TDE

Cybertech offers a PostgreSQL TDE solution, which is available as a paid tool, although older versions may be available for free. Some key aspects of this tool include:

- Provides cluster-level encryption to secure your entire PostgreSQL cluster.
- Utilizes the AES-128 encryption algorithm for data protection.
- Adopts a single key architecture for encryption.
- Imposes no noticeable performance overhead.
- Users have to implement an encrypted key store and manage the master key rotation.

6.2.3 EnterpriseDB Postgres TDE

EnterpriseDB also offers a paid tool for PostgreSQL TDE. Key features of this solution include:

- Employs a 2-tier encryption model.
- Offers database-level encryption.
- Provides support for external key management solutions, supporting several key vaults.
- Imposes no significant performance overhead.
- Utilizes the AES-128 encryption algorithm.
- Supports key rotation.

6.3 Oracle Encryption Options

Oracle offers the built-in Oracle TDE [1] tool.

6.3.1 Oracle TDE

- Oracle TDE is a paid solution.
- It employs a 2-tier encryption model.
- Oracle TDE implementation comes with no noticeable performance overhead.
- It offers encryption capabilities at both the tablespace and column levels.
- Supports master key rotation.

- External key management options, including just Oracle products, such as the Oracle Key Vault.
- Supports various encryption algorithms such as AES-128, AES-192, and AES-256.
- Oracle TDE provides a choice between online and offline encryption options.
- Provides features like direct path and parallelism for efficient data loading.

7 Performance comparison

A 21GB table was used for testing encryption in MySQL InnoDB, Cybertech PostgreSQL, and Oracle TDE. The table was loaded into encrypted instances and the time for the encryption was measured and is shown below.

- **MySQL**: Took 1 hour 19 minutes, speed of 4.4 MB/s.
- **PostgreSQL**: Took 1,5 hours, speed of 4 MB/s.
- **Oracle**: Took 1 hour 21 minutes, speed of 4.2 MB/s.

Using direct path and parallel options to load data in Oracle, time decreases considerably. The same table took 13 minutes to be loaded, a speed of approximately 27MB/s.

8 Conclusions

In the scope of database encryption, we've explored the offerings of three database management systems: MySQL, PostgreSQL, and Oracle.

MySQL presents two key options. MySQL InnoDB, a free solution, relies on keyring plugins for key management. The more robust MySQL Enterprise Edition, a paid tool, provides enhanced security through centralized key management solutions supporting several key vaults.

PostgreSQL, doesn't offer built-in TDE like MySQL and Oracle. Instead, users can use various plugins and solutions, such as the pgsql-hackers TDE, which might lead to additional work as the user is responsible for maintenance. Cybertech TDE and EnterpriseDB TDE, are paid options, that provide external key management, supporting several key vaults and more reliability in the long term.

Oracle, a paid solution, provides robust capabilities with TDE. It offers centralized key management but supports only Oracle products for key storage. Oracle TDE also offer online and offline encryption methods, and features like direct path and parallel loading. In the other hand, it is usually more expensive.

We were able to observe minor performance differences with the simple loading test, among the three options, in our setup, heavily relying on virtualization. Greater reliability and security, demand paid solutions for all three DBMS. Each option has its strengths and can be tailored to meet the security and compliance demands of CERN in its diverse environments.

Appendix

In this Appendix, we specify the relevant configuration applied to setup the encryption in MySQL, PostgreSQL and Oracle and the details used on our experiments.

Encryption Setup

MySQL InnoDB

- Component used: `component\_keyring\_file`
- Local keyring directory: `/ORA/dbs03/INSTANCE\_NAME/mysql/keyring/`
- Local keyring file: `/ORA/dbs03/INSTANCE\_NAME/mysql/keyring/component\_keyring\_file`
- Global manifest file: `/usr/local/mysql/mysql-8.0.28/bin/mysqld.my`
- Global configuration file: `/usr/local/mysql/mysql-8.0.28/lib/plugin/component_keyring_file.cnf`
- Local manifest file: `/ORA/dbs03/INSTANCE_NAME/mysql/mysqld.my`
- Local configuration file: `/ORA/dbs03/INSTANCE_NAME/mysql/component_keyring_file.cnf`

```
$ cat /usr/local/mysql/mysql-8.0.28/bin/mysqld.my
{
  "read_local_manifest": true
}
```

```
$ cat /usr/local/mysql/mysql-8.0.28/lib/plugin/component_keyring_file.cnf
Indicating local configuration
{
  "read_local_config": true
}
$ cat /ORA/dbs03/INSTANCE_NAME/mysql/mysqld.my

{
  "components": "file://component_keyring_file"
}
$ cat /ORA/dbs03/INSTANCE_NAME/mysql/component_keyring_file.cnf
{
  "path": "/ORA/dbs03/INSTANCE_NAME/mysql/keyring/component_keyring_file",
  "read_only": false
}
```

Cybertec PostgreSQL

```
$ cat /key/provide_key.sh
#!/bin/sh
echo 882fb7c12e80280fd664c69d2d636913

$$ initdb -D /postgres/db12tde -K /key/provide_key.sh
```

System Configuration

MySQL InnoDB

For the experiments in MySQL we used a stardard DBOD MySQL 8.0.28 instance in a shared VM.

```
Architecture:        x86_64
CPU(s):              16
Operating System:    Red Hat Enterprise Linux release 8.8 (Ootpa)
Model name:          Intel Core Processor (Broadwell, IBRS)
BIOS Model name:     RHEL-8.6.0 PC (Q35 + ICH9, 2009)
RAM:                 30 GB
```

Cybertech PostgreSQL
For the experiments in PostgreSQL we setup the PostgreSQL 12 instance coming with the standard Cybertech TDE patch installation in a custom openstack VM.

```
Architecture:        x86_64
CPU(s):              4
Operating System:    Red Hat Enterprise Linux release 8.8 (Ootpa)
Model name:          Intel Xeon Processor (Skylake, IBRS)
BIOS Model name:     RHEL-8.6.0 PC (Q35 + ICH9, 2009)
RAM:                 7.3 GB
```

Oracle
For the experiments in Oracle we shared the 19c (19.17) instance setup for a study on immutable backups.

```
Architecture:        x86_64
CPU(s):              48
Operating System:    Red Hat Enterprise Linux Server 7.9 (Maipo)
Model name:          Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
RAM:                 528 GB
```

CSV File

The *csv* file contains 4 fields, separated by commas. A sample of a line in the file is below:

```
created updated id json version_id
2018-08-16 17:34:44.65323, 2023-03-04 17:49:02.059818, 6550f68a-9e89-4
4f5-ac31-a776e9f268f8, "{"url": "", "code": "1F31NS043897-01", "title"
: "Estorgen Effects on Place Cells and Navigation Strategy", "funder":
{"$ref": "http://dx.doi.org/10.13039/100000002"}, "$schema": "HTTP://z
enodo.org/schemas/grants/grant-v1.0.0.json", "acronym": "", "enddate":
"", "program": NATIONAL_INSTITUTE_OF_NEUROLOGICAL_DISORDERS_AND_STROKE
","startdate": "2002-06-20", "identifiers": {"oaf":"nih_________::f80a
d5bab6c366b291f2baadca524691", "purl": null, "eurepo": "info:eu-repo/g
rantAgreement/NIH/NATIONAL_INSTITUTE_OF_NEUROLOGICAL_DISORDERS_AND_STR
OKE/1F31NS043897-01/"}, "internal_id": "10.13039/100000002::1F31NS0438
97-01", "remote_modified": "2021-04-27"}", 2
```

Table Description

MySQL InnoDB

```
+------------+------------+------+-----+---------+-------+
| Field      | Type       | Null | Key | Default | Extra |
+------------+------------+------+-----+---------+-------+
| created    | timestamp  | YES  |     | NULL    |       |
| updated    | timestamp  | YES  |     | NULL    |       |
| id         | binary(16) | YES  |     | NULL    |       |
| json_data  | json       | YES  |     | NULL    |       |
| version_id | int        | YES  |     | NULL    |       |
+------------+------------+------+-----+---------+-------+
```

Cybertec PostgreSQL

```
  Column   |            Type             | Nullable | Storage
created    | timestamp without time zone | not null | plain
updated    | timestamp without time zone | not null | plain
id         | uuid                        | not null | plain
json       | jsonb                       |          | extended
version_id | integer                     | not null | plain
```

Oracle

```
 Name        Null?            Type
------------- ----------- ---------------
CREATED       NOT NULL    TIMESTAMP(6)
UPDATED       NOT NULL    TIMESTAMP(6)
ID            NOT NULL    VARCHAR2(36)
JSON_DATA                 CLOB
VERSION_ID    NOT NULL    NUMBER(38)
```

Load Commands

Below, the commands for the loading the table in each technology.

MySQL InnoDB

```
LOAD DATA LOCAL INFILE '/afs/cern.ch/work/g/gwessler/records_metadata.csv'
INTO TABLE records_metadata
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
ESCAPED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(created, updated, id, @json_data, version_id)
SET json_data = NULLIF(@json_data, '');
```

Cybertec PostgreSQL

```
\copy records_metadata(created, updated, id, json, version_id) from
'/afs/cern.ch/work/g/gwessler/records_metadata.csv' delimiter ',' csv header;
```

Oracle

```
$ cat records_metadata.ctl
OPTIONS (SKIP=1, readsize=200000000, bindsize=200000000, DIRECT=true,
PARALLEL=true)
LOAD DATA
INFILE 'records_metadata.csv'            -- Path to your CSV file
APPEND INTO TABLE records_metadata      -- Target table name
FIELDS TERMINATED BY ','       -- Field delimiter in the CSV
TRAILING NULLCOLS
(
  created TIMESTAMP "YYYY-MM-DD HH24:MI:SS.FF6",
  updated TIMESTAMP "YYYY-MM-DD HH24:MI:SS.FF6",
  id,
  json_data CHAR(4196000) OPTIONALLY ENCLOSED BY '"' AND '"' NULLIF js
  on_data='',
  version_id
)

\$ sqlldr userid=\'/ as sysdba\' control=/ORA/dbs01/oracle/home/record
s_metadata.ctl
```

References

[1] Advanced Security Guide — docs.oracle.com. https://docs.oracle.com/en/database/oracle/oracle-database/19/asoag/introduction-to-transparent-data-encryption. [Accessed 04-10-2023].

[2] Advanced Security Guide — docs.oracle.com. https://docs.oracle.com/en/database/oracle/oracle-database/19/asoag/introduction-to-transparent-data-encryption.html#GUID-B0870B12-E6AD-4254-B4B3-D6A15A637975. [Accessed 04-10-2023].

[3] Chapter 31. Logical Replication — postgresql.org. https://www.postgresql.org/docs/current/logical-replication.html. [Accessed 04-10-2023].

[4] Data at rest - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Data_at_rest. [Accessed 04-10-2023].

[5] Data Protection: Data In transit vs. Data At Rest — digitalguardian.com. https://www.digitalguardian.com/blog/data-protection-data-in-transit-vs-data-at-rest#:~:text=Data%20at%20rest%20is%20safely,while%20it%20is%20being%20transmitted. [Accessed 04-10-2023].

[6] Locked Up: Advances in Postgres Data Encryption - Vibhor Kumar — youtube.com. https://www.youtube.com/watch?v=X5Vfi6qqyHk. [Accessed 04-10-2023].

[7] MySQL :: MySQL 8.0 Reference Manual :: 15.13 InnoDB Data-at-Rest Encryption — dev.mysql.com. https://dev.mysql.com/doc/refman/8.0/en/innodb-data-encryption.html. [Accessed 04-10-2023].

[8] MySQL :: MySQL Enterprise Transparent Data Encryption (TDE) — mysql.com. https://www.mysql.com/products/enterprise/tde.html. [Accessed 04-10-2023].

[9] Transparent Data Encryption — enterprisedb.com. https://www.enterprisedb.com/docs/tde/latest/. [Accessed 04-10-2023].

[10] Transparent Data Encryption - PostgreSQL wiki — wiki.postgresql.org. https://wiki.postgresql.org/wiki/Transparent_Data_Encryption. [Accessed 04-10-2023].

[11] Transparent data encryption - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Transparent_data_encryption. [Accessed 04-10-2023].

[12] Transparent Data Encryption for PostgreSQL — CYBERTEC — cybertec-postgresql.com. https://www.cybertec-postgresql.com/en/products/postgresql-transparent-data-encryption/. [Accessed 04-10-2023].

[13] What is Data at Rest? — techtarget.com. https://www.techtarget.com/searchstorage/definition/data-at-rest#:~:text=Data%20at%20rest%20is%20data,to%20be%20read%20or%20updated. [Accessed 04-10-2023].

[14] What is TDE? — docs.delphix.com. https://docs.delphix.com/docs609/datasets/oracle-environments-and-data-sources/what-is-tde. [Accessed 04-10-2023].

[15] Anirban Ghoshal. EnterpriseDB adds Transparent Data Encryption to PostgreSQL — infoworld.com. https://www.infoworld.com/article/3687813/enterprisedb-adds-transparent-data-encryption-to-postgresql.html. [Accessed 04-10-2023].

[16] Masahiko Sawada. Re: [Proposal] Table-level Transparent Data Encryption (TDE) and Key Management Service (KMS) — postgresql.org. https://www.postgresql.org/message-id/CAD21AoBjrbxvaMpTApX1cEsO%3D8N%3Dnc2xVZPB0d9e-VjJ%3DYaRnw%40mail.gmail.com. [Accessed 04-10-2023].