Highly-Scalable Relational Database Schema for Curation of Data Breaches with Diverse Data Types

Master Combo List (MCL) Project for University of Illinois

Vivek C. Nair
Authentication
vivekcn2@illinois.edu

Braydon Dudley
Database Systems
bdudley2@illinois.edu

Yang Rong
Database Systems
yr2@illinois.edu

Alex Ackerman
Database Systems
aackrmn2@illinois.edu

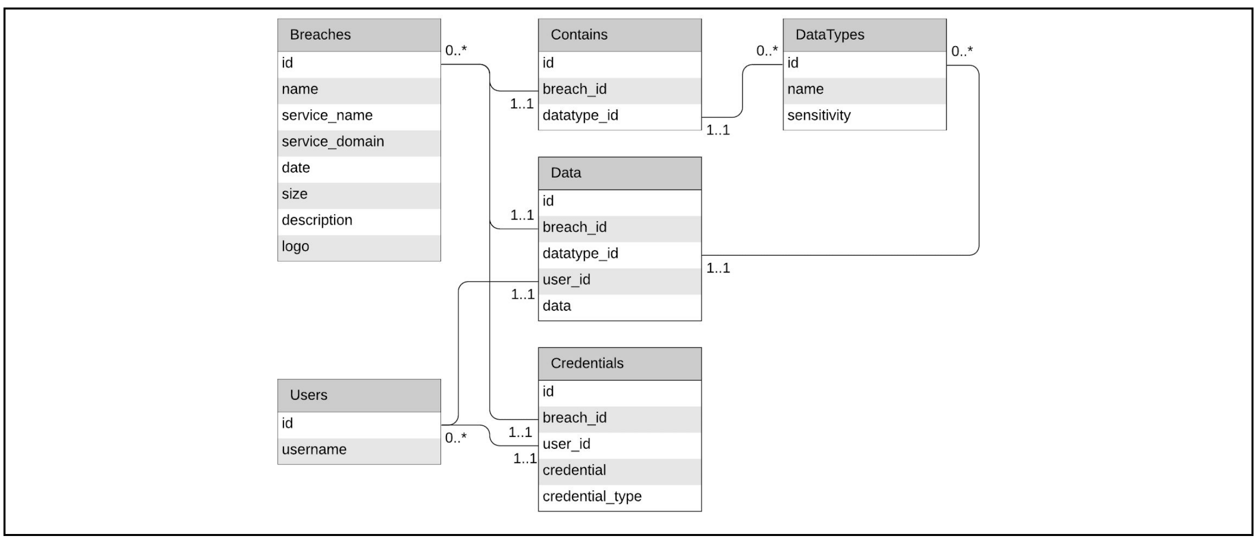

Figure 0: Entity relationship overview of the proposed system.

Overview

In this document we propose a relational database schema for storing data derived from data breaches for research and credential stuffing mitigation purposes. The schema is intended to be highly vertically scalable to the order of hundreds of billions (~10^{11}) of rows and tens of terabytes (~10^{13}) of data. The schema is also intended to be highly horizontally scalable, both locally and globally; given that breach data is relatively static by nature, direct replication can be used without raising concurrency concerns. Liberal use of B-tree indexing enables common operations (eg. credential comparison) to be performed with minimal latency and maximal throughput. We expect the component datasets to contain widely diverse data types and thus aim to maximize flexibility.

MCL Database Schema

The following table outlines our proposed tables in the MCL relational database schema. For each table, we specify the number of columns, estimated row count (based on current curation efforts, and accounting for some linear growth), database engine (InnoDB is suggested but head-to-head comparisons with MyISAM are merited when a final platform/architecture is determined), and default character set and collation, for which latin1 and latin1_bin are suggested as defaults but are frequently overridden for individual columns.

"MCL" Database Tables					
Name	Columns	Estimated Row Count	Engine	Charset	Collation
breaches	8	~10^6 (Tens of Thousands)	InnoDB	latin1	latin1_bin
datatypes	3	~10^2 (Hundreds)	InnoDB	latin1	latin1_bin
contains	3	~10^5 (Hundreds of Thousands)	InnoDB	latin1	latin1_bin
users	2	~10^{10} (Tens of Billions)	InnoDB	latin1	latin1_bin
data	5	~10^{11} (Hundreds of Billions)	InnoDB	latin1	latin1_bin
credentials	5	~10^{11} (Hundreds of Billions)	InnoDB	latin1	latin1_bin

Table 1: Overview of tables in "MCL" database schema.

Breaches

The breaches table stores basic information about all of the breaches in the database. Any data or credential stored in the database is associated with a particular breach.

Datatypes

The datatypes table stores attributes about the various classes of data found across one or more breaches in the database.

Contains

The contains table stores relationships between breaches and datatypes in the form [breach contains datatype].

Users

The users table stores user identifiers (primarily usernames and email addresses) to be associated with one or more credentials or data items.

Data

The data table stores actual leaked data, associated with a particular breach, user, and datatype.

Credentials

The credentials table stores actual leaked credentials, associated with a particular breach and user.

Entity Relationship Diagram
The entity relationships are proposed as follows:

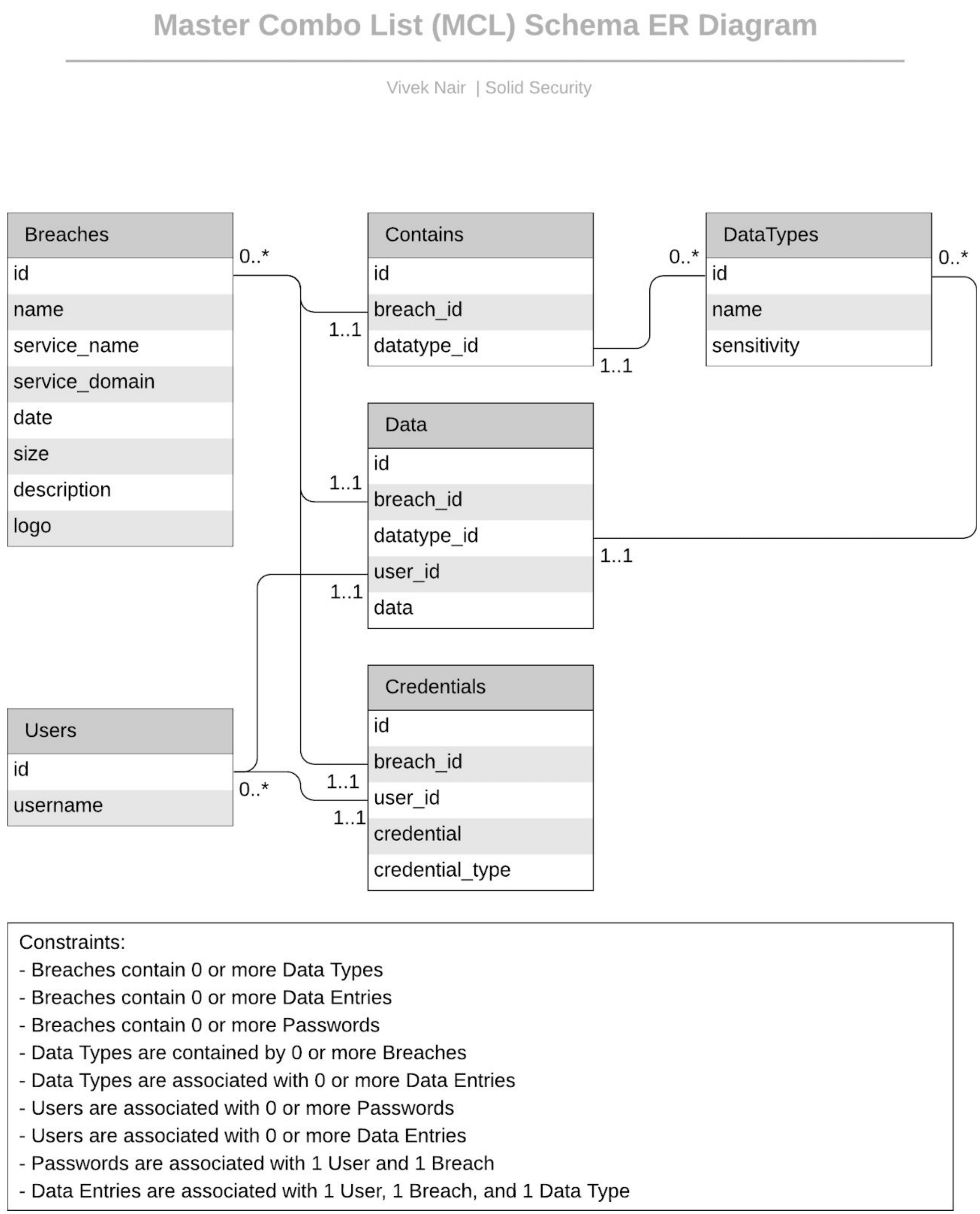

Figure 1: Entity relationship diagram.

Breaches Table Schema

The “breaches” table is proposed as follows:

“Breaches” Table Columns						
Name	Type	Attributes	Null	Default	Charset	Collation
id	bigint(20)	UNSIGNED, PRIMARY, UNIQUE, AUTO_INCREMENT	NO	(None)	(None)	(None)
name	varchar(255)	UNIQUE	NO	(None)	latin1	latin1_general_ci
service_name	varchar(255)		YES	NULL	latin1	latin1_general_ci
service_domain	varchar(255)		YES	NULL	latin1	latin1_general_ci
date	date		NO	(None)	(None)	(None)
size	int(10)	UNSIGNED	NO	(None)	(None)	(None)
description	text		NO	(None)	latin1	latin1_general_ci
logo	char(8)		NO	(None)	latin1	latin1_general_cs

Table 2: Overview of columns in “breaches” table schema.

The “name” column is used to provide a unique, case-insensitive name to each breach in a database (eg. myspace-2008), which could be used as a URL-safe identifier for API queries (eg. /breaches/myspace-2008). The optional “service_name” and “service_domain” columns identify the service which experienced the breach. The required “description” column provides a human-readable description of the breach and the data within. The “date” field can be used to denote the estimated date of the original breach (a separate date field could be added for the date breaches are added to the MCL system). The required “logo” data type provides an 8-character file identifier for a separately-stored logo file.

The indices for the “breaches” table are proposed as follows:

“Breaches” Table Indices			
Name	Type	Column	Unique
PRIMARY	BTREE	id	YES
id	BTREE	id	YES
name	BTREE	name	YES

Table 3: Overview of indices in “breaches” table schema.

Datatypes Table Schema
The “datatypes” table is proposed as follows:

“Datatypes” Table Columns						
Name	Type	Attributes	Null	Default	Charset	Collation
id	bigint(20)	UNSIGNED, PRIMARY, UNIQUE, AUTO_INCREMENT	NO	(None)	(None)	(None)
name	varchar(255)	UNIQUE	NO	(None)	latin1	latin1_general_ci
sensitivity	enum(...)		NO	(None)	latin1	latin1_bin

Table 4: Overview of columns in “datatypes” table schema.

The “name” field shall uniquely identify data types (case-insensitive) for API purposes (eg. /contains/addresses). For the “sensitivity” column, the following enumerated type is proposed:

“Sensitivity” Enum	
Name	Meaning
personal_low	Personal data that is not particularly sensitive (names, usernames, emails, etc.)
personal_medium	Personal data that is somewhat sensitive (dates of birth, addresses, etc.)
personal_high	Personal data that is highly sensitive (passport numbers, social security numbers, financial information, and anything in the Ashley Madison breach)
technical	Technical information (IP addresses, device IDs, IMEI/IMSI numbers, etc.)
security	Data with security implications (password hints, security questions & answers)
usage	Data relating to the usage of a service (search queries, chat logs, etc.)

Table 5: Overview of enum type in “sensitivity” column schema.

The indices for the “datatypes” table are proposed as follows:

“Datatypes” Table Indices			
Name	Type	Column	Unique
PRIMARY	BTREE	id	YES
id	BTREE	id	YES
name	BTREE	name	YES

Table 6: Overview of indices in “datatypes” table schema.

Contains Table Schema
The “contains” table is proposed as follows:

“Contains” Table Columns						
Name	Type	Attributes	Null	Default	Charset	Collation
id	bigint(20)	UNSIGNED, PRIMARY, UNIQUE, AUTO_INCREMENT	NO	(None)	(None)	(None)
breach_id	bigint(20)	UNSIGNED	NO	(None)	(None)	(None)
datatype_id	bigint(20)	UNSIGNED	NO	(None)	(None)	(None)

Table 7: Overview of columns in “contains” table schema.

The indices for the “contains” table are proposed as follows:

“Contains” Table Indices			
Name	Type	Column	Unique
PRIMARY	BTREE	id	YES
id	BTREE	id	YES
breach_id	BTREE	breach_id	NO
datatype_id	BTREE	datatype_id	NO

Table 8: Overview of indices in “contains” table schema.

The “contains” table shall adhere to the following foreign key constraints:

“Contains” Table Foreign Key Constraints			
Column	Foreign Key	On Delete	On Update
breach_id	mcl.breaches.id	RESTRICT	RESTRICT
datatype_id	mcl.datatypes.id	RESTRICT	RESTRICT

Table 9: Overview of foreign key constraints in “contains” table schema.

Users Table Schema
The “users” table is proposed as follows:

“Users” Table Columns						
Name	Type	Attributes	Null	Default	Charset	Collation
id	bigint(20)	UNSIGNED, PRIMARY, UNIQUE, AUTO_INCREMENT	NO	(None)	(None)	(None)
username	varchar(255)	UNIQUE	NO	(None)	utf8mb4	utf8mb4_bin

Table 10: Overview of columns in “users” table schema.

Here, “username” could refer to a username, email address, user ID, or any other unique identifier used for authentication purposes. A separate username_type field could be used to identify which type was used, but may not be necessary for most operations.

The “username” field shall be unique (case-sensitive/utf8) for API query purposes (eg. /breaches/contain/user123).

Indices for the “users” table are proposed as follows:

“Users” Table Indices			
Name	Type	Column	Unique
PRIMARY	BTREE	id	YES
id	BTREE	id	YES
username	BTREE	username	YES

Table 11: Overview of indices in “users” table schema.

Note that practical constraints may require multiple B-tree indices to be maintained for the username column, namely one for enforcing the uniqueness constraint and another for enabling rapid access by username.

Data Table Schema
The "data" table is proposed as follows:

"Data" Table Columns						
Name	Type	Attributes	Null	Default	Charset	Collation
id	bigint(20)	UNSIGNED, PRIMARY, UNIQUE, AUTO_INCREMENT	NO	(None)	(None)	(None)
breach_id	bigint(20)	UNSIGNED	NO	(None)	(None)	(None)
datatype_id	bigint(20)	UNSIGNED	NO	(None)	(None)	(None)
user_id	bigint(20)	UNSIGNED	NO	(None)	(None)	(None)
data	text				utf8mb4	utf8mb4_general_ci

Table 12: Overview of columns in "data" table schema.

The indices for the table are proposed as follows:

"Data" Table Indices			
Name	Type	Column	Unique
PRIMARY	BTREE	id	YES
id	BTREE	id	YES
breach_id	BTREE	breach_id	NO
datatype_id	BTREE	datatype_id	NO
user_id	BTREE	user_id	NO

Table 13: Overview of indices in "data" table schema.

The following foreign key constraints shall be adhered to:

"Data" Table Foreign Key Constraints			
Column	Foreign Key	On Delete	On Update
breach_id	mcl.breaches.id	RESTRICT	RESTRICT
datatype_id	mcl.datatypes.id	RESTRICT	RESTRICT
user_id	mcl.users.id	RESTRICT	RESTRICT

Table 14: Overview of foreign key constraints in "data" table schema.

Credentials Table Schema

The “credentials” table is proposed as follows:

“Credentials” Table Columns						
Name	Type	Attributes	Null	Default	Charset	Collation
id	bigint(20)	UNSIGNED, PRIMARY, UNIQUE, AUTO_INCREMENT	NO	(None)	(None)	(None)
breach_id	bigint(20)	UNSIGNED	NO	(None)	(None)	
user_id	bigint(20)	UNSIGNED	NO	(None)	(None)	
credential	varchar(255)		NO	(None)	utf8mb4	utf8mb4_bin
credential_type	enum(...)		NO	(None)	latin1	latin1_bin

Table 15: Overview of columns in “credentials” table schema.

What constitutes a “credential” may vary by standard, but here we define a compatible enumerated type which supports plaintext passwords, hashed passwords, and symmetric keys (additions to support asymmetric keys and other credential types may be warranted):

“Credential Type” Enum	
Category	Options
Plaintext Passwords	plaintext
Hashed Passwords	plaintext, md2, md4, md5, sha1, sha224, sha256, sha384, sha512, ripemd128, ripemd160, ripemd256, ripemd320, whirlpool, tiger128,3, tiger160,3, tiger192,3, tiger128,4, tiger160,4, tiger192,4, snefru, snefru256, gost, gost-crypto, adler32, crc32, crc32b, fnv132, fnv1a32, fnv164, fnv1a64, joaat, haval128,3, haval160,3, haval192,3, haval224,3, haval256,3, haval128,4, haval160,4, haval192,4, haval224,4, haval256,4, haval128,5, haval160,5, haval192,5, haval224,5, haval256,5, blowfish, std_des, ext_des
Symmetric Keys	AES-128-CBC, AES-128-CFB, AES-128-CFB1, AES-128-CFB8, AES-128-OFB, AES-192-CBC, AES-192-CFB, AES-192-CFB1, AES-192-CFB8, AES-192-OFB, AES-256-CBC, AES-256-CFB, AES-256-CFB1, AES-256-CFB8, AES-256-OFB, BF-CBC, BF-CFB, BF-OFB, CAST5-CBC, CAST5-CFB, CAST5-OFB, IDEA-CBC, IDEA-CFB, IDEA-OFB, aes-128-cbc, aes-128-cfb, aes-128-cfb1, aes-128-cfb8, aes-128-ofb, aes-192-cbc, aes-192-cfb, aes-192-cfb1, aes-192-cfb8, aes-192-ofb, aes-256-cbc, aes-256-cfb, aes-256-cfb1, aes-256-cfb8, aes-256-ofb, bf-cbc, bf-cfb, bf-ofb, cast5-cbc, cast5-cfb, cast5-ofb, idea-cbc, idea-cfb, idea-ofb, totp

Table 16: Overview of enum type in “credential_type” column schema.

Credentials Table Schema, Continued
The indices for the credential table schema are proposed as follows:

“Credentials” Table Indices			
Name	Type	Column	Unique
PRIMARY	BTREE	id	YES
id	BTREE	id	YES
breach_id	BTREE	breach_id	NO
user_id	BTREE	user_id	NO

Table 17: Overview of indices in “credentials” table schema.

The following foreign key constraints shall be adhered to for the “credentials” table:

“Credentials” Table Foreign Key Constraints			
Column	Foreign Key	On Delete	On Update
breach_id	mcl.breaches.id	RESTRICT	RESTRICT
user_id	mcl.users.id	RESTRICT	RESTRICT

Table 18: Overview of foreign key constraints in “credentials” table schema.

SQL Specification
The schema can alternatively be specified in Structured Query Language (SQL) as follows:

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

CREATE DATABASE IF NOT EXISTS `mcl` DEFAULT CHARACTER SET latin1 COLLATE
latin1_swedish_ci;
USE `mcl`;

CREATE TABLE `breaches` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `service_name` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci
DEFAULT NULL,
  `service_domain` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci
```

```
DEFAULT NULL,
  `date` date NOT NULL,
  `size` int(10) UNSIGNED NOT NULL,
  `description` text CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `logo` char(8) CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin;

CREATE TABLE `contains` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `breach_id` bigint(20) UNSIGNED NOT NULL,
  `datatype_id` bigint(20) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin;

CREATE TABLE `credentials` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `breach_id` bigint(20) UNSIGNED NOT NULL,
  `user_id` bigint(20) UNSIGNED NOT NULL,
  `credential` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT
NULL,
  `credential_type`
enum('plaintext','md2','md4','md5','sha1','sha224','sha256','sha384','sha512',
'ripemd128','ripemd160','ripemd256','ripemd320','whirlpool','tiger128,3','tige
r160,3','tiger192,3','tiger128,4','tiger160,4','tiger192,4','snefru','snefru25
6','gost','gost-crypto','adler32','crc32','crc32b','fnv132','fnv1a32','fnv164'
,'fnv1a64','joaat','haval128,3','haval160,3','haval192,3','haval224,3','haval2
56,3','haval128,4','haval160,4','haval192,4','haval224,4','haval256,4','haval1
28,5','haval160,5','haval192,5','haval224,5','haval256,5','blowfish','std_des'
,'ext_des','AES-128-CBC','AES-128-CFB','AES-128-CFB1','AES-128-CFB8','AES-128-
OFB','AES-192-CBC','AES-192-CFB','AES-192-CFB1','AES-192-CFB8','AES-192-OFB','
AES-256-CBC','AES-256-CFB','AES-256-CFB1','AES-256-CFB8','AES-256-OFB','BF-CBC
','BF-CFB','BF-OFB','CAST5-CBC','CAST5-CFB','CAST5-OFB','IDEA-CBC','IDEA-CFB',
'IDEA-OFB','aes-128-cbc','aes-128-cfb','aes-128-cfb1','aes-128-cfb8','aes-128-
ofb','aes-192-cbc','aes-192-cfb','aes-192-cfb1','aes-192-cfb8','aes-192-ofb','
aes-256-cbc','aes-256-cfb','aes-256-cfb1','aes-256-cfb8','aes-256-ofb','bf-cbc
','bf-cfb','bf-ofb','cast5-cbc','cast5-cfb','cast5-ofb','idea-cbc','idea-cfb',
'idea-ofb','totp') COLLATE latin1_bin NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin;

CREATE TABLE `data` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `breach_id` bigint(20) UNSIGNED NOT NULL,
  `datatype_id` bigint(20) UNSIGNED NOT NULL,
  `user_id` bigint(20) UNSIGNED NOT NULL,
  `data` text CHARACTER SET utf8mb4 NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin;

CREATE TABLE `datatypes` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `sensitivity`
enum('personal_low','personal_medium','personal_high','technical','security','
usage') COLLATE latin1_bin NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin;
```

```
CREATE TABLE `users` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `username` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin DEFAULT
NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin;


ALTER TABLE `breaches`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `id` (`id`),
  ADD UNIQUE KEY `name` (`name`);

ALTER TABLE `contains`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `id` (`id`),
  ADD KEY `breach_id` (`breach_id`),
  ADD KEY `datatype_id` (`datatype_id`);

ALTER TABLE `credentials`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `id` (`id`),
  ADD KEY `breach_id` (`breach_id`),
  ADD KEY `user_id` (`user_id`);

ALTER TABLE `data`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `id` (`id`),
  ADD KEY `breach_id` (`breach_id`),
  ADD KEY `datatype_id` (`datatype_id`),
  ADD KEY `user_id` (`user_id`);

ALTER TABLE `datatypes`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `id` (`id`),
  ADD UNIQUE KEY `name` (`name`);

ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `id` (`id`),
  ADD UNIQUE KEY `username_2` (`username`),
  ADD KEY `username` (`username`);


ALTER TABLE `breaches`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

ALTER TABLE `contains`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

ALTER TABLE `credentials`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE `data`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

ALTER TABLE `datatypes`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;

ALTER TABLE `users`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT;


ALTER TABLE `contains`
  ADD CONSTRAINT `contains_ibfk_1` FOREIGN KEY (`breach_id`) REFERENCES
`breaches` (`id`),
  ADD CONSTRAINT `contains_ibfk_2` FOREIGN KEY (`datatype_id`) REFERENCES
`datatypes` (`id`);

ALTER TABLE `credentials`
  ADD CONSTRAINT `credentials_ibfk_1` FOREIGN KEY (`breach_id`) REFERENCES
`breaches` (`id`),
  ADD CONSTRAINT `credentials_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES
`users` (`id`);

ALTER TABLE `data`
  ADD CONSTRAINT `data_ibfk_1` FOREIGN KEY (`breach_id`) REFERENCES `breaches`
(`id`),
  ADD CONSTRAINT `data_ibfk_2` FOREIGN KEY (`datatype_id`) REFERENCES
`datatypes` (`id`),
  ADD CONSTRAINT `data_ibfk_3` FOREIGN KEY (`user_id`) REFERENCES `users`
(`id`);
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Structured Query Language (SQL) Representation