# Alibaba Cloud ApsaraDB for Cassandra

## Best practices

Issue: 20200521

# Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please contact Alibaba Cloud directly if you discover any errors in this document.

# Document conventions

| Style | Description | Example |
| --- | --- | --- |
|  | A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results. |  **Danger:** Resetting will result in the loss of user configuration data. |
|  | A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results. |  **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance. |
|  | A caution notice indicates warning information, supplementary instructions, and other content that the user must understand. |  **Notice:** If the weight is set to 0, the server no longer receives new requests. |
|  | A note indicates supplemental instructions, best practices, tips, and other content. |  **Note:** You can use Ctrl + A to select all files. |
| > | Closing angle brackets are used to indicate a multi-level menu cascade. | Click **Settings** > **Network** > **Set network type**. |
| **Bold** | Bold formatting is used for buttons, menus, page names, and other UI elements. | Click **OK**. |
| Courier font | Courier font is used for commands. | Run the `cd /d C:/window` command to enter the Windows system folder. |
| Italic | Italic formatting is used for parameters and variables. | `bae log list --instanceid` *Instance_ID* |
| [] or [a\|b] | This format is used for an optional value, where only one item can be selected. | `ipconfig [-all\|-t]` |

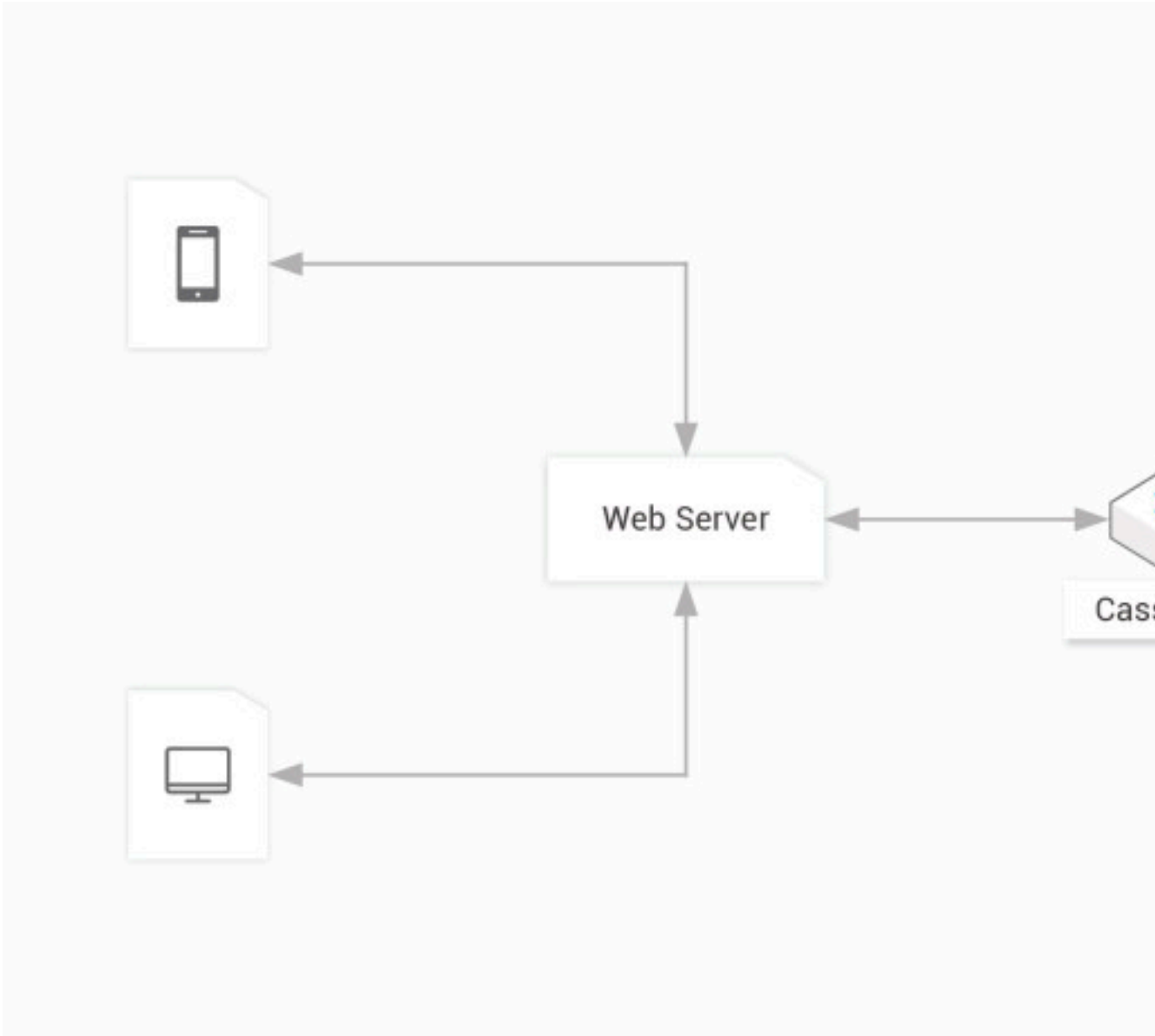
| Style | Description | Example |
| --- | --- | --- |
| {} or {a\|b} | This format is used for a required value, where only one item can be selected. | switch {active\|stand} |

# Contents

Legal disclaimer..... I
Document conventions.....I
1 ApsaraDB for Cassandra usage scenarios..... 1
2 Use static columns..... 5

# 1 ApsaraDB for Cassandra usage scenarios

ApsaraDB for Cassandra supports access requests that require high concurrency and low latency, and provides high availability and elastic scaling features. ApsaraDB for Cassandra is applicable to scenarios such as messages, feed streams, orders queries, websites and other online Internet scenarios that process a large amount of data.




**Benefits**

**High availability**

Services are not affected by single point of failures.

**Low latency**

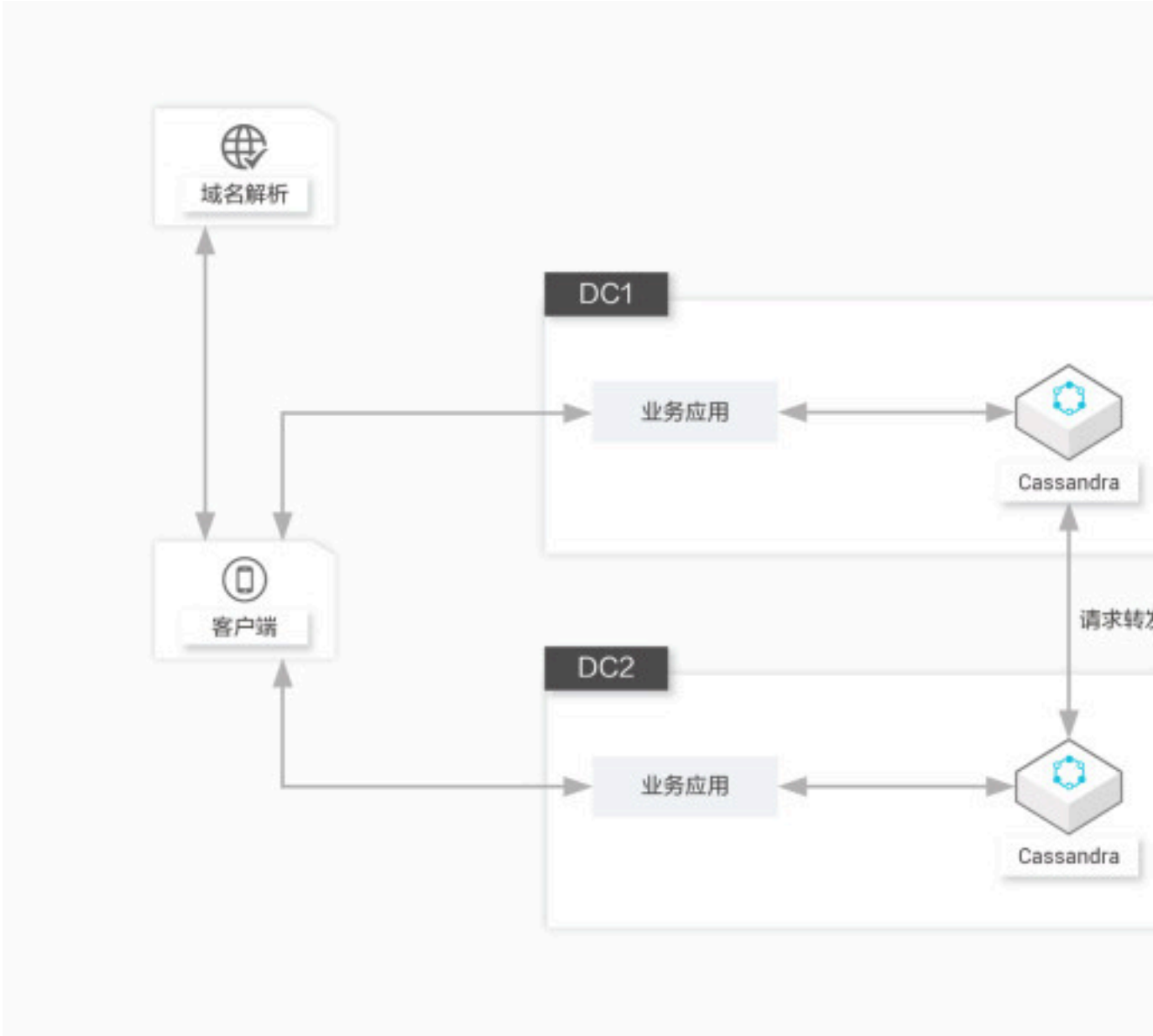
The 99.9% of network latency is solved within milliseconds.

**Elasticity**

The computing and storage capabilities can be flexibly scaled as the business grows.

**Multi-active data centers**

Native Apache Cassandra supports deployment in multiple data centers. Applicatio ns adopt the LOCAL_QUORUM consistency level to access Cassandra, and Cassandra automatically forwards the requests to another data center. This provides better availability and disaster recovery capabilities.



**Benefits**

**Disaster recovery**

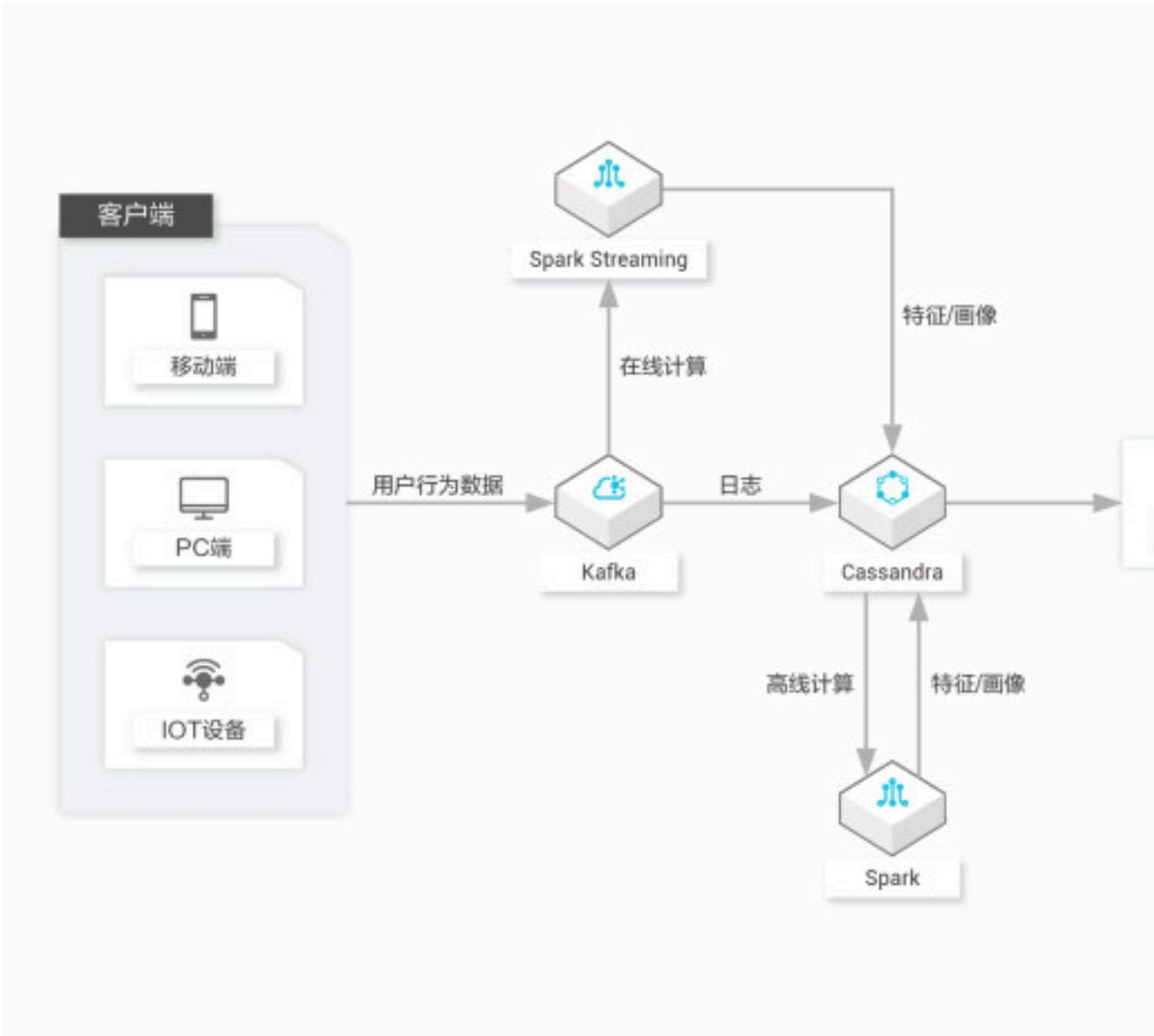
Data and service availability are not affected by physical damage.

**Simple implementation**

Data synchronization between data centers is implemented by native Cassandra mechanisms with minimum impacts on applications.

**Data-driven services: risk control, recommendation, and IoT**

ApsaraDB for Cassandra stores user behavior data and constructed user profiles integrated from multiple data sources in real time to provide big data risk control and recommenda tion services.



**Benefits**

**Low storage costs**

Uses sparse storage schemes, which are suitable for storing highly compressed user profile data.

**High scalability**

The computing and storage capabilities can be flexibly scaled as the business grows.

**Low latency**

The 99.9% of network latency is solved within milliseconds.

# 2 Use static columns

Assume that a Cassandra table stores the user information (email address and password) and user status. The user information remains constant while the user status changes frequently. If the user information is updated along with each status update, a large amount of storage space will be used.

Cassandra provides the static column feature to solve this problem. The values of a static column in a partition are consistent and only one copy is stored.

**Define a static column**

Add STATIC after a column name to define the column as a static column. The example is as follows:

```
CREATE TABLE "iteblog_users_with_status_updates" (
  "username" text,
  "id" timeuuid,
  "email" text STATIC,
  "encrypted_password" blob STATIC,
  "body" text,
  PRIMARY KEY ("username", "id")
);
```

The preceding statements set email and encrypted_password to STATIC, which means there is only one email value and one encrypted_password value corresponding to a username.

**Limits**

You cannot define static columns in the following situations:

- The table is not configured with clustering columns (also known as clustering keys). For example:

```
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                   ...   "username" text,
                   ...   "id" timeuuid,
                   ...   "email" text STATIC,
                   ...   "encrypted_password" blob STATIC,
                   ...   "body" text,
                   ...   PRIMARY KEY ("username")
                   ... );
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static columns
are only useful (and thus allowed) if the table has at least one clustering column"
```

You cannot define static columns in the iteblog_users_with_status_updates_invalid table because the table has only a primary key and does not have clustering columns. The more rows of data exist in a partition, the better the static column performance will be. If no

clustering columns are defined, a primary key value identifies a unique row in a partition and no static columns are required.

- The table is set to a COMPACT STORAGE table. For example:

```
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                   ...   "username" text,
                   ...   "id" timeuuid,
                   ...   "email" text STATIC,
                   ...   "encrypted_password" blob STATIC,
                   ...   "body" text,
                   ...   PRIMARY KEY ("username", "id")
                   ... )WITH COMPACT STORAGE;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static columns
are not supported in COMPACT STORAGE tables"
```

- Columns used as partition keys or clustering columns cannot be set to STATIC. For example:

```
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                   ...   "username" text,
                   ...   "id" timeuuid STATIC,
                   ...   "email" text STATIC,
                   ...   "encrypted_password" blob STATIC,
                   ...   "body" text,
                   ...   PRIMARY KEY ("username", "id")
                   ... );
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static column id
cannot be part of the PRIMARY KEY"
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                   ...   "username" text,
                   ...   "id" timeuuid,
                   ...   "email" text STATIC,
                   ...   "encrypted_password" blob STATIC,
                   ...   "body" text,
                   ...   PRIMARY KEY (("username", "id"), email)
                   ... );
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static column
email cannot be part of the PRIMARY KEY"
```

**Insert data into a table that contains static columns**

Inserting data into a table that contains static columns is similar to inserting data into a standard table. For example, you can execute the following statements to insert data into the iteblog_users_with_status_updates table:

```
cqlsh:iteblog_keyspace> INSERT INTO "iteblog_users_with_status_updates"
                   ... ("username", "id", "email", "encrypted_password", "body")
                   ... VALUES (
                   ...   'iteblog',
                   ...   NOW(),
                   ...   'iteblog_hadoop@iteblog.com',
                   ...   0x877E8C36EFA827DBD4CAFBC92DD90D76,
                   ...   'Learning Cassandra!'
                   ... );
cqlsh:iteblog_keyspace> select username, email, encrypted_password, body from
iteblog_users_with_status_updates;
```

```
 username | email                    | encrypted_password               | body
----------+--------------------------+----------------------------------
+--------------------
  iteblog | iteblog_hadoop@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 |
Learning Cassandra!

(1 rows)
```

A row of data is inserted into the table. The preceding statements have performed the following operations:

- All data in the email and encrypted_password columns corresponding to the iteblog username is set to iteblog_hadoop@iteblog.com and 0x877e8c36efa827dbd4 cafbc92dd90d76 because the email and encrypted_password columns are static.
- A row with the Learning Cassandra! body is added to the iteblog partition. Execute the following statements to insert another row into the table:

```
cqlsh:iteblog_keyspace> INSERT INTO "iteblog_users_with_status_updates"
                    ... ("username", "id", "body")
                    ... VALUES ('iteblog', NOW(), 'I love Cassandra!') ;
cqlsh:iteblog_keyspace> select username, email, encrypted_password, body from
iteblog_users_with_status_updates;

 username | email                    | encrypted_password               | body
----------+--------------------------+----------------------------------
+--------------------
  iteblog | iteblog_hadoop@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 |
Learning Cassandra!
  iteblog | iteblog_hadoop@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 |   I
love Cassandra!

(2 rows)
cqlsh:iteblog_keyspace>
```

The email and encrypted_password columns are not specified. However, the query result shows that the values of the two columns in the new row are consistent with the values specified in the preceding statements.

The user modifies the email address as follows:

```
cqlsh:iteblog_keyspace> UPDATE iteblog_users_with_status_updates SET email = 'iteblog
@iteblog.com'
                    ... WHERE username = 'iteblog';
cqlsh:iteblog_keyspace> select username, email, encrypted_password, body from
iteblog_users_with_status_updates;

 username | email              | encrypted_password               | body
----------+--------------------+----------------------------------+--------------------
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 | Learning
Cassandra!
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 |   I love
Cassandra!
```

```
(2 rows)
```

The query result shows that the emails of all iteblog username rows are synchronized as the new email because the email column is static.

The table stores user information such as the email address and password. If a user modifies the email address and password on the frontend page, the backend system must obtain the current email address and password by executing the following statements:

```
cqlsh:iteblog_keyspace> SELECT "username", "email", "encrypted_password"
                    ... FROM "iteblog_users_with_status_updates"
                    ... WHERE "username" = 'iteblog';

 username | email              | encrypted_password
----------+--------------------+----------------------------------
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76

(2 rows)
```

All rows with the iteblog username are displayed. You can use the SELECT DISTINCT statement to eliminate duplicate rows:

```
cqlsh:iteblog_keyspace> SELECT DISTINCT "username", "email", "encrypted_password"
                    ... FROM "iteblog_users_with_status_updates"
                    ... WHERE "username" = 'iteblog';

 username | email              | encrypted_password
----------+--------------------+----------------------------------
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76

(1 rows)
```

All duplicate rows with the iteblog username are eliminated and a unique row is displayed.

> Cassandra does not perform the DISTINCT operation for all data. A copy of the static column data is already stored in the underlying system.