# RDBMS & MPP

A yellow note on a white sheet of paper

AI-generated content may be incorrect.

RDBMS stands for Relational Database Management System. It is primarily to store the data that is generated by the users of this application using web or mobile application. It permanently stores the data. The data will be stored in the form of tables in the database.

Application servers will channelize the traffic from the users into this database. Also, when the user tries to fetch the data, the request will go to database, the data will be returned from the database. Application server will take care of processing the data and it will send it to web and the data will be rendered as part of the browser using technologies like HTML.

A diagram of data storage

AI-generated content may be incorrect.

MPP stands for massively parallel processing.They are also databases which serve the reports and dashboards for the users within the organization and also outside the organization

|  |  |
| --- | --- |
| RDBMS | Data Warehouse or MPP |
| Not only tables but also columns, data types, rows or records. The data is stored in the form of tables as part of tables.We typically have columns with the data types. The actual data will be rows or records in the tables. | |
| use SQL to perform a wide variety of tasks against the tables in Rdbmss, as well as Data Warehouse or MPP: validation of the data, troubleshooting bugs by running queries against the tables, setting up the databases for development… | |
| used for transactional based systems (OLTP - online transaction processing) | used for OLAP or DSS. DSS stands for Decision Support Systems. OLAP stands for Online Analytical Processing |
| build mobile or web applications. | build reports or dashboards. |

# SQL

## Overview

**Cơ sở dữ liệu là gì?**

Cơ sở dữ liệu là tập hợp các dữ liệu có tổ chức. Ví dụ, một cơ sở dữ liệu của một trường đại học có thể là tập hợp các dữ liệu như: Hồ sơ các nhân của sinh viên, lịch sử hiệu suất của sinh viên, dữ liệu giáo viên, dữ liệu của bộ phận tài chính, v.v.

**Cơ sở dữ liệu quan hệ**

Trong cơ sở dữ liệu quan hệ, dữ liệu được tổ chức dưới dạng bảng. Một bảng chứa các hàng và cột dữ liệu. Bảng có một khóa (key) duy nhất để xác định từng hàng của bảng.

SQL được sử dụng để tương tác với các cơ sở dữ liệu quan hệ. Ta thường xem cơ sở dữ liệu quan hệ là cơ sở dữ liệu SQL.

**SQL là gì?**

SQL là viết tắt của Structured Query Language, là một ngôn ngữ chuẩn hóa để tương tác với RDBMS (Hệ quản trị cơ sỡ dữ liệu quan hệ). Một số ví dụ về cơ sở dữ liệu quan hệ: MySQL, Oracle, mariaDB, postgreSQL, v.v.

SQL được sử dụng để thực hiện các hoạt động C.R.U.D (Tạo, truy xuất, cập nhật và xóa) trên cơ sở dữ liệu quan hệ.

SQL cũng có thể thực hiện các tác vụ quản trị trên cơ sở dữ liệu như bảo mật cơ sở dữ liệu, sao lưu, quản lý người dùng, v.v.

Ta có thể tạo cơ sở dữ liệu và bảng bên trong cơ sở dữ liệu bằng SQL.

## Table

Một cơ sở dữ liệu thường chứa một hoặc nhiều bảng. Trong hệ thống cơ sở dữ liệu quan hệ, một bảng có thể được xem như là một quan hệ (relation).

Bảng dùng để lưu trữ các thông tin của một đối tượng trong thực tế. Mỗi bảng được xác định bằng một tên (ví dụ như actor hay film). Một bảng có các bản ghi (dòng) và các trường (cột).

## Comment

* Comment trên một dòng bắt đầu bằng cú pháp **--**
* Comment trên nhiều dòng bằng /\* \*/

## Data types

TEXT: Text được sử dụng cho các giá trị chuỗi như tên hay là mô tả sản phẩm, không bị giới hạn về chiều dài.

VARCHAR:

* lưu trữ chuỗi các ký tự không bị giới hạn chiều dài, còn cho phép người dùng đặt ra giới hạn đối với các giá trị dữ liệu được gọi bằng cách mô tả một số nguyên N, được đặt trong ngoặc đơn liền sau. Ví dụ VARCHAR(50), VARCHAR(100),...
* cho phép các chuỗi có ít hơn N ký tự được lưu trữ trong cột. Việc cố gắng chèn một chuỗi có độ dài lớn hơn N sẽ dẫn đến lỗi.
* Việc khai báo là cột có kiểu dữ liệu là VARCHAR mà không chỉ định N thì tương đương với khai báo một cột có kiểu dữ liệu là TEXT.

CHAR

* được sử dụng để thể hiện một chuỗi các ký tự nhưng nó khác với VARCHAR(N) ở chỗ các giá trị được lưu trữ trong cột CHAR(N) không thay đổi về chiều dài. Nếu một chuỗi được lưu trữ nhỏ hơn độ dài cố định là N, khoảng trắng sẽ được thêm vào cuối (bên phải) để đảm bảo chuỗi có độ dài N.
* Một cột CHAR không được khai báo N mặc định nghĩa là cột đấy chỉ có thể lưu trữ một ký tự. Nghĩa là: CHAR tương đương với CHAR(1).

Numeric: được sử dụng cho các giá trị dữ liệu đại diện cho số lượng và số đo.

SMALLINT là số nguyên có dấu, được lưu trữ bằng 2 byte, có phạm vi từ -32,768 đến 32,767.

INTEGER là số nguyên có dấu, được lưu trữ bằng 4 byte, có phạm vi từ -2,147,483,648 đến 2,147,483,647.

SERIAL

* là kiểu số nguyên, PostgreSQL sẽ tự động tạo và điền các giá trị tăng dần vào cột SERIAL. Tương tự với cột AUTO\_INCREMENT trong cột MySQL hoặc AUTO INCREMENT trong SQLite.
* E.g. PostgreSQL chèn hai hàng vào bảng fruits với các giá trị cho cột id là 1 và 2.

CREATE TABLE fruits(  
 id SERIAL PRIMARY KEY,  
 name VARCHAR NOT NULL  
);

INSERT INTO fruits(name) VALUES('Orange');  
INSERT INTO fruits(id, name) VALUES(DEFAULT,'Apple');

| **id** | **name** |
| --- | --- |
| 1 | Apple |
| 2 | Orange |

FLOAT(n) là một số thực dấu phẩy động có độ chính xác, ít nhất là n, được lưu trữ tối đa là bằng 8 byte.

REAL hoặc FLOAT8 là số thực dấu phẩy động, được lưu trữ bởi 4 byte.

NUMERIC hay NUMERIC(p,s) là một số thực có p chữ số với số s sau dấu thập phân. Trong PostgreSQL NUMERIC và DECIMAL là tương đương nhau và có thể thay thế cho nhau.

Temporal: được sử dụng cho các giá trị dữ liệu đại diện cho ngày và thời gian.

* DATE chỉ lưu trữ giá trị của ngày, tháng năm.
* TIME lưu trữ các giá trị thời gian trong ngày
* TIMESTAMP lưu trữ cả ngày tháng, năm và thời gian trong ngày.
* TIMESTAMPTZ là kiểu dữ liệu dấu thời gian, nhận biết múi giờ. Là viết tắt của dấu thời gian với múi giờ, là phần mở rộng PostgreSQL trên các kiểu dữ liệu tạm thời theo tiêu chuẩn SQL.
* INTERVAL lưu trữ khoảng thời gian.

BOOLEAN or BOOL: có thể có một trong 3 trạng thái true, false hoặc null.

## Show table

SELECT \* FROM link; --Hiển thị tất cả các bản ghi của bảng link

-- Show the information of all restaurants ordering by id.

SELECT \* FROM restaurants ORDER BY id;

## Create table

Syntax:

CREATE TABLE tên\_bảng (  
 tên\_cột kiểu\_dữ\_liệu ràng\_buộc,  
 tên\_cột kiểu\_dữ\_liệu ràng\_buộc  
);

e.g.

CREATE TABLE account(  
 user\_id SERIAL,  
 username VARCHAR(50),  
 password VARCHAR(50),  
 email VARCHAR(355),  
 created\_on DATE DEFAULT CURRENT\_DATE,  
 last\_login TIMESTAMP  
);

## Insert

Syntax:

INSERT INTO bảng(cột1, cột2, …) VALUES (giá\_trị1, giá\_trị2, …);

e.g.

INSERT INTO link (id, url, name,description, rel)  
VALUES ('1','http://www.postgresqltutorial.com','PostgreSQL Tutorial',NULL,NULL);

INSERT INTO link (id, url, name)  
VALUES ('2','http://www.oreilly.com','O''Reilly Media');

INSERT INTO link  
VALUES('3','http://www.codelearn.io','Codelearn',NULL,NULL);

Nếu bạn bỏ qua bất kỳ cột nào chấp nhận NULL value trong câu lệnh INSERT, cột sẽ lấy giá trị mặc định của nó. Trong trường hợp giá trị mặc định không được đặt cho cột, cột sẽ lấy giá trị NULL.

Để thêm nhiều hàng vào một bảng cùng một lúc, bạn sử dụng cú pháp sau:

INSERT INTO bảng (cột1, cột2, …)

VALUES

(giá\_trị1, giá\_trị2, …),

(giá\_trị1, giá\_trị2, …);

Để chèn dữ liệu đến từ một bảng khác:

INSERT INTO bảng(cột1, cột2,...)

SELECT cột1, cột2,...

FROM bảng\_khác WHERE điều\_kiện;

Nếu bạn muốn chỉ chèn giá trị duy nhất từ một bảng khác, bạn có thể sử dụng từ khóa DISTINCT trong câu lệnh SELECT:

INSERT INTO bảng(cột1,cột2,...)

SELECT DISTINCT cột1, cột2,...

FROM bảng\_khác WHERE điều\_kiện;

e.g.

-- Insert unique professors into the new table

INSERT INTO professors

SELECT DISTINCT

    first\_name, last\_name, university\_shortname

FROM

    university\_professors

ORDER BY

    first\_name DESC;

-- Doublecheck the contents of professors

SELECT \* FROM professors;

A screen shot of a computer

AI-generated content may be incorrect.

A blue and white line

AI-generated content may be incorrect.

## Add column

Để thêm một cột mới vào một bảng hiện có:

ALTER TABLE tên\_bảng ADD COLUMN tên\_cột\_mới kiểu\_dữ\_liệu DEFAULT giá\_trị;

Để thêm nhiều cột vào một bảng hiện có:

ALTER TABLE tên\_bảng  
ADD COLUMN tên\_cột\_mới\_1 kiểu\_dữ\_liệu ràng\_buộc DEFAULT giá\_trị,  
ADD COLUMN tên\_cột\_mới\_2 kiểu\_dữ\_liệu ràng\_buộc DEFAULT giá\_trị,  
...;

## Change column data type

Để thay đổi kiểu dữ liệu của một cột:

ALTER TABLE tên\_bảng ALTER COLUMN tên\_cột [SET DATA] TYPE kiểu\_dữ\_liệu\_mới;

Để thay đổi kiểu dữ liệu của nhiều cột trong một câu lệnh, bạn hãy sử dụng cú pháp sau:

ALTER TABLE tên\_bảng  
ALTER COLUMN tên\_cột\_1 [SET DATA] TYPE kiểu\_dữ\_liệu\_mới,  
ALTER COLUMN tên\_cột\_2 [SET DATA] TYPE kiểu\_dữ\_liệu\_mới,  
...;

## Rename column

Syntax:

ALTER TABLE tên\_bảng RENAME COLUMN tên\_cột TO tên\_cột\_mới;

e.g.

CREATE TABLE customers (  
    id serial PRIMARY KEY,  
    name VARCHAR NOT NULL,  
    phone VARCHAR NOT NULL,  
    email VARCHAR  
);

ALTER TABLE customers RENAME COLUMN name TO customer\_name;  
ALTER TABLE customers RENAME phone TO contact\_phone;

## Constraint

Ý tưởng của cơ sở dữ liệu là sắp xếp dữ liệu theo một cấu trúc nhất định - một mô hình được xác định trước, nơi bạn sử dụng các kiểu dữ liệu, các mối quan hệ và các quy tắc lên cấu trúc đấy. Nhìn chung, những quy tắc này được gọi là các ràng buộc toàn vẹn (integrity constraints).

Các ràng buộc toàn vẹn có thể được chia thành ba loại:

* Ràng buộc thuộc tính, ví dụ kiểu dữ liệu trên cột.
* Các ràng buộc khóa, ví dụ các khóa chính.
* Các ràng buộc tham chiếu, được thi hành thông qua các khóa ngoại.

Các ràng buộc định hình cấu trúc của dữ liệu. Dữ liệu được nhập bởi người dùng thường dài dòng và không theo nguyên tắc nào dẫn đến việc sẽ gặp khó khăn trong quá trình xử lí dữ liệu. Vì vậy, việc cung cấp các ràng buộc sẽ mang đến tính nhất quán cho dữ liệu, có nghĩa là một hàng trong một bảng nhất định có cấu trúc chính xác giống như hàng tiếp theo, v.v. Nói chung, chúng giúp giải quyết rất nhiều vấn đề về chất lượng của dữ liệu.

### Ràng buộc thuộc tính đặc biệt

1. NULL

- Giá trị NULL nghĩa là một giá trị không được xác định hoặc hoàn toàn không tồn tại. Ràng buộc NOT NULL không cho phép các giá trị (bị) NULL trong một cột nhất định. Điều này phải đúng với trạng thái hiện tại của cơ sở dữ liệu, nhưng cũng đúng với bất kỳ trạng thái nào trong tương lai.

- e.g.

CREATE TABLE students(  
     ssn INTEGER NOT NULL,  
     lastname VARCHAR(64) NOT NULL,  
     home\_phone INTEGER  
);

cũng có thể thêm ràng buộc NOT NULL đối với cột home\_phone cho bảng students đã được tạo rồi với syntax như sau:

ALTER TABLE students ALTER COLUMN home\_phone SET NOT NULL;

2. UNIQUE

- Ràng buộc UNIQUE trên một cột đảm bảo rằng không có sự trùng lặp trong cột đấy. Cũng giống như với ràng buộc NOT NULL, bạn chỉ có thể thêm một ràng buộc UNIQUE nếu cột không tồn tại một sự trùng lặp nào nào trước khi bạn đặt ràng buộc lên cột.

- syntax:

CREATE TABLE tên\_bảng(  
     tên\_cột kiểu\_dữ\_liệu UNIQUE  
);

* Hoặc cho bảng đã tồn tại rồi như sau:

ALTER TABLE tên\_bảng ADD CONSTRAINT tên\_ràng\_buộc UNIQUE(tên\_cột);

### Key

Thông thường, bảng trong cơ sở dữ liệu có một thuộc tính hoặc kết hợp nhiều thuộc tính mà giá trị của chúng là giá trị duy nhất trên toàn bộ bảng. Các thuộc tính như vậy xác định một bản ghi duy nhất.

Một bảng chỉ chứa các bản ghi khác biệt nhau (tính duy nhất của các bản ghi), có nghĩa là sự kết hợp của tất cả các thuộc tính là một khóa trong chính nó. Tuy nhiên, nó vẫn chưa được gọi là khóa chính, mà được gọi là một siêu khóa (superkey). Nếu xóa một hoặc một số thuộc tính trong tập hợp các thuộc tính đó cho đến khi không thể xóa được nữa mà tập hợp thuộc tính đấy vẫn xác định tính duy nhất của các bản ghi thì tập hợp đấy là khóa. **Vì vậy, khóa luôn là tối giản nhất.**

**Khóa chính** là một trong những khái niệm quan trọng nhất trong thiết kế cơ sở dữ liệu.

* Hầu như mọi bảng cơ sở dữ liệu có một khóa chính. Mục đích chính của khóa chính là xác định tính duy nhất của các bản ghi trong một bảng.
* Các khóa chính cần được xác định trên các cột không chấp nhận giá trị trùng lặp hoặc NULL.
* Các ràng buộc khóa chính là bất biến theo thời gian. Do đó, nên chọn các cột trong đó các giá trị sẽ luôn là duy nhất và không rỗng.

Syntax:

CREATE TABLE TABLE (  
    cột\_1 kiểu\_dữ\_liệu PRIMARY KEY,  
    cột\_2 kiểu\_dữ\_liệu,  
     …  
);

Trong trường hợp khóa chính bao gồm hai hoặc nhiều cột, bạn xác định ràng buộc khóa chính như sau:

CREATE TABLE TABLE (  
     cột\_1 kiểu\_dữ\_liệu,  
     cột\_2 kiểu\_dữ\_liệu,  
     …   
     PRIMARY KEY (cột\_1, cột\_2)  
);

Thêm ràng buộc PRIMARY KEY vào bảng hiện có:

ALTER TABLE tên\_bảng

ADD CONSTRAINT tên\_ràng\_buộc PRIMARY KEY(tên\_cột)

**Khóa ngoại**

Mối quan hệ giữa các bảng được thực hiện bằng các khóa ngoại - các cột của bảng này được chỉ định trỏ đến khóa chính của bảng khác.

Có một số quy định đối với khóa ngoại:

* Đầu tiên, tên và kiểu dữ liệu phải giống như với khóa chính được trỏ đến.
* Thứ hai, chỉ các giá trị khóa ngoại được phép tồn tại dưới dạng giá trị trong khóa chính của bảng được tham chiếu. Đây là ràng buộc khóa ngoại, còn được gọi là "tính toàn vẹn tham chiếu" (referential integrity)
* Cuối cùng, khóa ngoại không nhất thiết phải là khóa, vì các giá trị trùng lặp và giá trị NULL được cho phép đối với cột khóa ngoại.

Để xác định khóa ngoại, bạn nên sử dụng một trong các cách dưới đây:

--Cách 1  
CREATE TABLE bảng\_con(  
    c1 kiểu\_dữ\_liệu PRIMARY KEY,  
    c2 kiểu\_dữ\_liệu REFERENCES bảng\_bố(p2)  
);

--Cách 2  
CREATE TABLE bảng\_con(  
    c1 kiểu\_dữ\_liệu PRIMARY KEY,  
    c2 kiểu\_dữ\_liệu ,  
    FOREIGN KEY (c2) REFERENCES bảng\_bố(p2)  
);

Trong trường hợp khóa ngoại là một nhóm cột, chúng ta có thể xác định ràng buộc khóa ngoài bằng cú pháp sau:

CREATE TABLE bảng\_con(  
    c1 kiểu\_dữ\_liệu PRIMARY KEY,  
    c2 kiểu\_dữ\_liệu,  
    c3 kiểu\_dữ\_liệu,  
    FOREIGN KEY (c2, c3) REFERENCES bảng\_bố(p2, p3)  
);

**Toàn vẹn tham chiếu**

Một bản ghi tham chiếu một bảng khác phải luôn luôn tham chiếu đến một bản ghi hiện có. Nói cách khác: Một bản ghi trong bảng A không thể trỏ đến một bản ghi không tồn tại ở trong bảng B. Tính toàn vẹn của tham chiếu là một ràng buộc luôn liên quan đến hai bảng và được thi hành thông qua các khóa ngoại.

Tính toàn vẹn tham chiếu có thể bị vi phạm bởi hai cách:

* Giả sử bảng A tham chiếu bảng B. Vì vậy, nếu một bản ghi trong bảng B đã được tham chiếu từ bảng A bị xóa, bạn vi phạm tính toàn vẹn tham chiếu.
* Mặt khác, nếu bạn cố gắng chèn một bản ghi vào bảng A nhưng nó không tồn tại trong bảng B, bạn cũng vi phạm nguyên tắc.

Đó là lý do chính cho các khóa ngoại sẽ gây lỗi và ngăn bạn làm những việc này. Tuy nhiên, thông báo lỗi không phải là cách duy nhất. Nếu bạn chỉ định khóa ngoại trên một cột, bạn thực sự có thể cho hệ thống cơ sở dữ liệu biết điều gì sẽ xảy ra nếu một mục trong bảng được tham chiếu bị xóa. Theo mặc định, từ khóa **ON DELETE NO ACTION** được tự động gắn vào định nghĩa khóa ngoại:

CREATE TABLE a   
   id INTEGER PRIMARY KEY,  
   column\_a VARCHAR(50),  
   ...,  
   b\_id INTEGER REFERENCES b (id) ON DELETE NO ACTION  
);

Điều này có nghĩa là nếu bạn cố xóa một bản ghi trong bảng B được tham chiếu từ A, hệ thống sẽ đưa ra lỗi. Tuy nhiên, có những lựa chọn khác. Ví dụ: có tùy chọn CASCADE, trước tiên sẽ cho phép xóa bản ghi trong bảng B, sau đó sẽ tự động xóa tất cả các bản ghi tham chiếu trong bảng A. Vì vậy, việc xóa đó được xếp tầng (cascaded).

CREATE TABLE a   
  id INTEGER PRIMARY KEY,  
   column\_a VARCHAR(50),  
   ...,  
   b\_id INTEGER REFERENCES b (id) ON DELETE CASCADE  
);

### Check

Ràng buộc CHECK là một loại ràng buộc cho phép bạn chỉ định nếu một giá trị trong cột phải đáp ứng một yêu cầu cụ thể. Ràng buộc CHECK sử dụng biểu thức Boolean để đánh giá các giá trị trước khi chèn hoặc cập nhật vào cột. Nếu các giá trị vượt qua kiểm tra, PostgreSQL sẽ chèn hoặc cập nhật các giá trị này vào cột.

e.g.

CREATE TABLE employees (  
 id SERIAL PRIMARY KEY,  
 first\_name VARCHAR (50),  
 last\_name VARCHAR (50),  
 birth\_date DATE CHECK (birth\_date > '1900-01-01'),  
 joined\_date DATE CHECK (joined\_date > birth\_date),  
 salary NUMERIC CHECK(salary > 0)  
);

e.g. sử dụng câu lệnh ALTER TABLE để thêm các ràng buộc CHECK vào bảng price\_list rằng price và discount phải lớn hơn 0 và discount nhỏ hơn price. Như vậy chúng ta cần sử dụng biểu thức Boolean chứa toán tử AND để thêm ràng buộc như sau:

ALTER TABLE prices\_list ADD CONSTRAINT price\_discount\_check CHECK (

price > 0 AND discount >= 0 AND price > discount

);

### Drop

Câu lệnh DROP CONSTRAINT dùng để xóa các ràng buộc UNIQUE, PRIMARY KEY, FOREIGN KEY, hay CHECK.

Cú pháp để loại bỏ một ràng buộc trong PostgreSQL là:

ALTER TABLE tên\_bảng DROP CONSTRAINT tên\_ràng\_buộc;

ALTER TABLE tên\_bảng   
DROP CONSTRAINT tên\_ràng\_buộc1,  
DROP CONSTRAINT tên\_ràng\_buộc2,  
...;