

SQL (3)

Contents

- [View](#)
- Query Execution Plans
- Subquery
- Trigger
- Types of Databases
- Normalizations

View

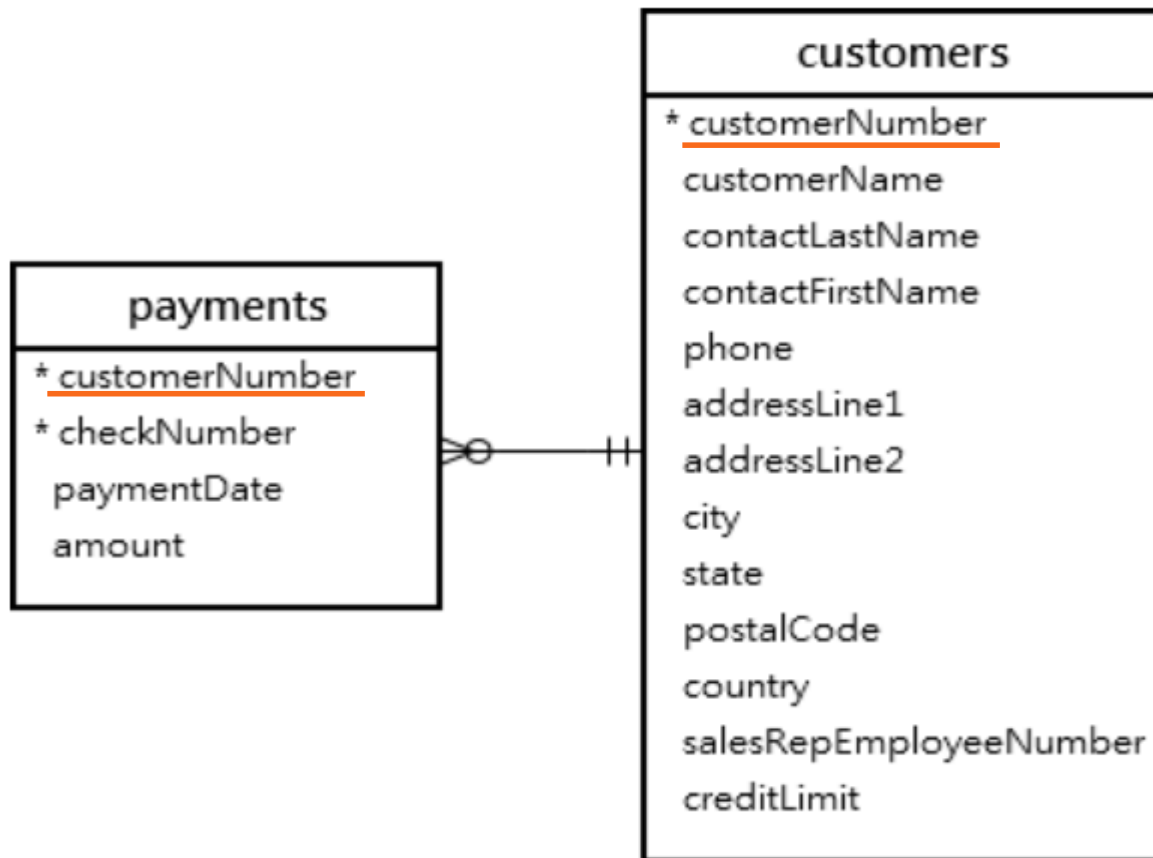
View

- A view is a **database object that has no values**. Its contents are based on the base table. It contains rows and columns similar to the real table. In MySQL, the View is a **virtual table** created by a query by joining one or more tables.
- MySQL supports views, including updatable views.
- **Views are stored queries that when invoked produce a result set.** A view acts as a virtual table.
- Comparison:

	act as	existed in Disk	after running an SQL resident in Memory
Table	base table	yes	yes
View	virtual table	no	yes

The Views in MySQL


- If the following tables customers and payments from your DB



- This query returns data from both tables customers and payments using the inner join:

```
SELECT
    customerName,
    checkNumber,
    paymentDate,
    amount
FROM
    customers
INNER JOIN
    payments USING (customerNumber);
```

Here is the output:



	customerName	checkNumber	paymentDate	amount
▶	Atelier graphique	HQ336336	2004-10-19	6066.78
	Atelier graphique	JM555205	2003-06-05	14571.44
	Atelier graphique	OM314933	2004-12-18	1676.14
	Signal Gift Stores	BO864823	2004-12-17	14191.12
	Signal Gift Stores	HQ55022	2003-06-06	32641.98
	Signal Gift Stores	ND748579	2004-08-20	33347.88
	Australian Collectors, Co.	GG31455	2003-05-20	45864.03
	Australian Collectors, Co.	MA765515	2004-12-15	82261.22
	Australian Collectors, Co.	NP603840	2003-05-31	7565.08
	Australian Collectors, Co.	NR27552	2004-03-10	44894.74
	La Rochelle Gifts	DB933704	2004-11-14	19501.82
	La Rochelle Gifts	LN373447	2004-08-08	47924.19

- So, next time, if you want to get the same information including customer name, check number, payment date, and amount, you need to issue **the same query** again.
- One way to do this is to save the query in a file, either .txt or .sql file so that later you can open and execute it from phpmyadmin, MySQL Workbench or any other MySQL client tools.
- By definition, a view is a named **query** stored in the database catalog.
- To create a new view you use the CREATE VIEW statement. This statement creates a view customerPayments based on the above query above:

```
CREATE VIEW customerPayments
AS
SELECT
    customerName,
    checkNumber,
    paymentDate,
    amount
FROM
    customers
INNER JOIN
    payments USING (customerNumber);
```

TABLE_SCHEMA	TABLE_NAME	VIEW_DEFINITION
ksu_db1004-w4	daysofweek	<pre>select 'Mon' AS</pre>
ksu_db1004-w4	test_view1	<pre>select ~ksu_db1004-w4~ ~ ~ksu_std_table~ ~</pre>
mysql	user	<pre>select ~mysql~ ~ ~global_priv~ ~</pre>

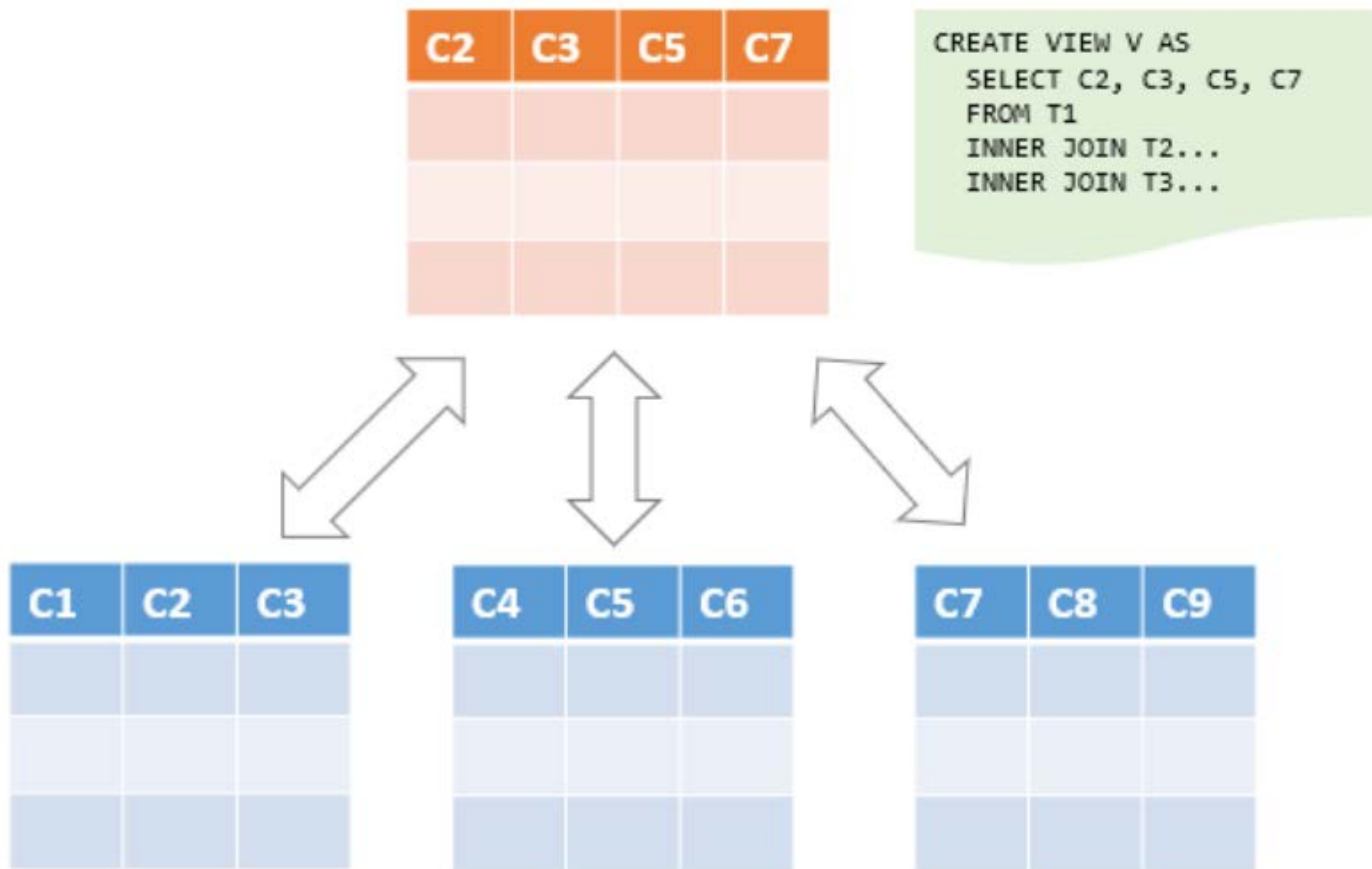
- Which is better **stored procedure** or **view**?
 - In general, a **Stored Procedure** stands a good chance of being faster than a direct SQL statement because the server does all sorts of optimizations when a stored procedure is saved and executed the first time. A view is essentially a saved SQL statement and not optimized.
- When should I use a **view** and when should I use a **stored procedure**?
 - Most simply, a view is used when only a SELECT statement is needed. Views should be used to store commonly-used JOIN queries and specific columns to build virtual tables of an exact set of data we want to see.
 - Stored procedures hold the more complex logic, such as INSERT, DELETE, and UPDATE statements to automate large SQL workflows.

- Once you execute the CREATE VIEW statement, MySQL creates the view and stores it in the database.
- So, now, another example, you can reference the view as a table in SQL statements. For example, you can query data from the customerPayments **view** using the SELECT statement:

```
SELECT * FROM customerPayments;
```

- As you can see, the syntax is much simpler.
- **Note that a view does not physically store the data in Disk. When you issue the SELECT statement against the view, MySQL executes the underlying query specified in the view's definition and returns the result set. For this reason, sometimes, a view is referred to as a virtual table.**
- MySQL allows you to create a view based on a **SELECT** statement that retrieves data from one or more tables. This picture illustrates a view based on columns of multiple tables:

- MySQL allows you to create a view based on a **SELECT** statement that retrieves data from one or more tables. This picture illustrates a view based on columns of multiple tables:



- Can we insert data in view?
 - **We cannot insert or update data using view.** The view is a virtual table. We can do those action, but it's effect on real table data too. View is like a virtual table which enable us to get information of multiple tables.
- What are temporary tables used for?
 - A temporary table exist solely **for storing data within a session.** The best time to use temporary tables are when you need to store information within Database server for use over a number of SQL transactions.
- What is the difference between view and temporary table?
 - **A view exists only for a single query.** Each time you use the name of a view, its table is recreated from existing data. A temporary table exists for the entire database session in which it was created and existed within a session

- In addition, MySQL even allows you to create a view **that does not refer to any table**. But you will rarely find this kind of view in practice.
- For example, you can create a view called `daysofweek` that return 7 days of a week by executing the following query:

```
CREATE VIEW daysofweek (day) AS
  SELECT 'Mon'
  UNION
  SELECT 'Tue'
  UNION
  SELECT 'Web'
  UNION
  SELECT 'Thu'
  UNION
  SELECT 'Fri'
  UNION
  SELECT 'Sat'
  UNION
  SELECT 'Sun';
```



	day
►	Mon
	Tue
	Web
	Thu
	Fri
	Sat
	Sun

- Advantages of MySQL Views:

- **Simplify complex query:** Views help simplify complex queries. If you have any frequently used complex query, you can create a view based on it so that you can reference to the view by using a simple SELECT statement instead of typing the query all over again.
- **Make the business logic consistent:** Suppose you have to repeatedly write the same formula in every query. Or you have a query that has complex business logic. To make this logic consistent across queries, you can use a view to store the calculation and hide the complexity.
- **Add extra security layers:** By using views and privileges, you can limit which data users can access by exposing only the necessary data to them. For example, the table employees may contain SSN and address information, which should be accessible by the HR department only.
- **Enable backward compatibility :** In legacy systems, views can enable backward compatibility. Suppose, you want to normalize a big table into many smaller ones. And you don't want to impact the current applications that reference the table. In this case, you can create a view whose name is the same as the table based on the new tables so that all applications can reference the view as if it were a table. Note that a view and table cannot have the same name so you need to drop the table first before creating a view whose name is the same as the deleted table. **So, suggest to use different name for view.**

How to create a simple MySQL view

- The basic syntax for creating a view in MySQL is as follows:

```
CREATE VIEW [db_name.]view_name [(column_list)]  
AS  
    select-statement;
```

- **[db_name.]** is the name of the database where your view will be created; if not specified, the view will be created in the current database
- **view_name** is a unique name of the view you are creating
- **[(column_list)]** defines the required list of columns that can be indicated in parentheses after the view name; by default, the list of columns is retrieved from the select list of the SELECT statement
- **select-statement** is a specified SELECT statement that can query data from tables or views

- Here is the simplest example.

Databases SQL Status User account

Run SQL query/queries on server "127.0.0.1":

```
1 Create view test_view1 AS
2 select * from ksu_std_table;
```



New

- information_schema
- ksu
 - ksu_database
 - ksu_db0914
 - ksu_db1004-w4
 - Procedures
 - Tables
 - Views



Browse Structure SQL Search

Run SQL query/queries on table ksu_db1004-w4.test_view1:

```
1 select * from test_view1;
```



Filters

Containing the word:

Table	Action	Rows	Type
<input type="checkbox"/> daysofweek	★ Browse Structure Search Insert Edit Drop	~0	View
<input type="checkbox"/> test_view1	★ Browse Structure Search Insert Edit Drop	~0	View

2 tables Sum ~0 InnoDB

```
show full tables;
```

☐ Profiling [Edit inline] [Edit] [Create PHP code]

+ Options

Tables_in_ksu_db1004-w4	Table_type
advisor_detail	BASE TABLE
child_table	BASE TABLE
city_detail	BASE TABLE
customer	BASE TABLE
customer_1	BASE TABLE
customer_2	BASE TABLE
customer_3	BASE TABLE
customer_4	BASE TABLE
daysofweek	VIEW
dept_detail	BASE TABLE
employee_salary_error	BASE TABLE
employee_salary_error1	BASE TABLE
employee_salary_ok	BASE TABLE
employee_salary_ok1	BASE TABLE
employee_salary_ok2	BASE TABLE
employee_salary_ok3	BASE TABLE
geography	BASE TABLE
ksu_std_table	BASE TABLE
parent_table	BASE TABLE
product_detail	BASE TABLE
store_information	BASE TABLE
store_information_1	BASE TABLE
student_detail	BASE TABLE
test_table01	BASE TABLE
test_table1	BASE TABLE
test_table2	BASE TABLE
test_view1	VIEW
testtable	BASE TABLE
tracing_log	BASE TABLE
userinfo	BASE TABLE
議題意見討論版	BASE TABLE

```
SELECT table_schema, TABLE_NAME, view_definition  
from information_schema.views;
```

TABLE_SCHEMA	TABLE_NAME	VIEW_DEFINITION
--------------	------------	-----------------

ksu_db1004-w4	daysofweek	<pre>select 'Mon'</pre> <pre>AS</pre>
---------------	------------	--

ksu_db1004-w4	test_view1	<pre>select `ksu_db1004-w4` . `ksu_std_table` .</pre>
---------------	------------	---

mysql	user	<pre>select `mysql` . `global_priv` .</pre>
-------	------	---

How to update a MySQL view

- If you need to update tables through views (**not suggest to do this if you not familiar with view creation!**), you can use the INSERT, UPDATE, and DELETE statements to perform the corresponding operations with the rows of the underlying table. However, please note that in order to be updatable, **your view must not include any of the following:**
 - Aggregate functions, e.g. MIN, MAX, COUNT, AVG, or SUM
 - Such clauses as GROUP BY, DISTINCT, HAVING, UNION or UNION ALL
 - Left or outer JOINS
 - Multiple references to any column of the base table
 - Subqueries in the SELECT or WHERE clause referring to the table appearing in the FROM clause
 - References to non-updatable views in the FROM clause
 - References to non-literal values
- **Suggest to drop your view, then recreate it if you are not familiar with view creation!**

How to update a MySQL view

```
CREATE VIEW warehouse_details AS  
  SELECT warehouse_id, phone, city  
  FROM warehouses;
```

Now we can query data from this view:

```
SELECT * FROM warehouse_details;
```

Let's say we want to change the phone number of the warehouse with the `warehouse_id` '55' through the `warehouse_details` view using the following UPDATE statement.

```
UPDATE warehouse_details  
SET  
  phone = '(555) 555-1234'  
WHERE  
  warehouse_id = 55;
```

Finally, we can check whether the change has been applied using the following query:

```
SELECT * FROM warehouse_details  
WHERE  
  warehouse_id = 55;
```

How to drop a MySQL view

- If we no longer need a certain view, we can delete it with a simple DROP statement:

```
drop view test_view2
```

Example

- Check the view if you want to use



```
SELECT table_schema, table_name, view_definition
FROM   information_schema.views
where  table_name = "test_view1"
```

table_schema	table_name	view_definition
ksu_db1004-w4	test_view1	<pre>select `ksu_db1004-w4` . `ksu_std_table` .</pre>

```
show full tables;
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Create PHP](#)]

+ Options

Tables_in_ksu_db1004-w4	Table_type
advisor_detail	BASE TABLE
child_table	BASE TABLE
city_detail	BASE TABLE
customer	BASE TABLE
customer_1	BASE TABLE
customer_2	BASE TABLE
customer_3	BASE TABLE
customer_4	BASE TABLE
daysofweek	VIEW
dept_detail	BASE TABLE
employee_salary_error	BASE TABLE
employee_salary_error1	BASE TABLE
employee_salary_ok	BASE TABLE
employee_salary_ok1	BASE TABLE
employee_salary_ok2	BASE TABLE
employee_salary_ok3	BASE TABLE
geography	BASE TABLE
ksu_std_table	BASE TABLE
parent_table	BASE TABLE
product_detail	BASE TABLE
store_information	BASE TABLE
store_information_1	BASE TABLE
student_detail	BASE TABLE
test_table01	BASE TABLE
test_table1	BASE TABLE
test_table2	BASE TABLE
test_view1	VIEW
testtable	BASE TABLE
tracing_log	BASE TABLE
userinfo	BASE TABLE
議題意見討論版	BASE TABLE















































Show the view to verify it

```
select * from test_view1;
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Number of rows: Filter rows:

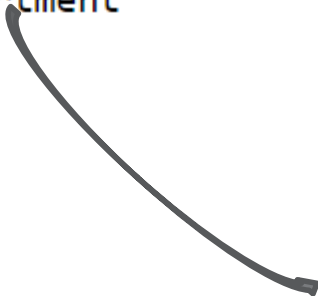
Options

				ksu_std_id	ksu_std_name	ksu_std_age	ksu_std_department	ksu_std_signin	ksu_std_grade
<input type="checkbox"/>				2323E1	John1	33	QQ	2020-04-01	100
<input type="checkbox"/>				4040w1	John1	22	CS	2020-04-01	100
<input type="checkbox"/>				D01	John Sieg	22	CS	2019-12-05	100
<input type="checkbox"/>				D02	John Sieg	44	IE	2019-12-04	99
<input type="checkbox"/>				IE01	Canning	33	IE	2019-11-12	100
<input type="checkbox"/>				IE02	Mike Fire	32	IE	2019-12-11	77
<input type="checkbox"/>				IE03	Mary Wee	34	IM	2019-12-02	80
<input type="checkbox"/>				IM01	WuBer Eat	22	IM	2019-11-12	33
<input type="checkbox"/>				IM02	Foot Penny	27	CS	2019-10-10	100
<input type="checkbox"/>				IM05	John Sieg	24	CS	2019-12-16	100
<input type="checkbox"/>				ss	1John	22	CS	2020-04-01	100
<input type="checkbox"/>				33	33	0		0000-00-00	100
<input type="checkbox"/>				9898	Mike	0		0000-00-00	100
<input type="checkbox"/>				777	Taiwan	0		0000-00-00	100
<input type="checkbox"/>				s	sss	0		0000-00-00	100
<input type="checkbox"/>				ddd	ddd	100		0000-00-00	100

Console

- Creating a view based on another view example

```
create view test_view2 AS
select ksu_std_department, count(*)
from test_view1
where ksu_std_age > 20
group by ksu_std_department
```



```
SELECT * FROM `test_view2`
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)]

☐ Show all | Number of rows: 25 ▾

+ Options

ksu_std_department	count(*)
	1
CS	5
IE	3
IM	2
QQ	1

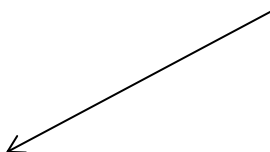
```
select * from information_schema.views where table_name = "test_view2";
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Number of rows: 25 ▾ | Filter rows:

+ Options

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	VIEW_DEFINITION
def	ksu_db1004-w4	test_view2	<pre>select `test_view1` . `ksu_std_department` AS</pre>



Creating a view with explicit view columns example

```
create view test_view3
(DEPARTMENT,
 NUMBER_OF_STUDENTS) AS
select distinct (ksu_std_department), count(*)
from ksu_std_table
where ksu_std_age > 20
```

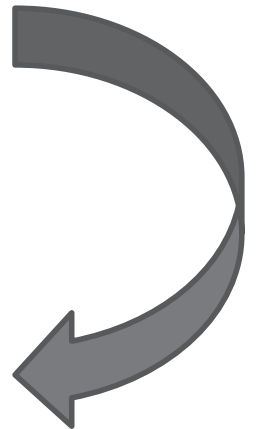
```
SELECT * FROM `test_view3`
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Number of rows: 25  Filter rows:

+ Options

DEPARTMENT	NUMBER_OF_STUDENTS
QQ	12



Query Execution Plans

Types of Databases

- A company should use the type of database that fit in its requirements and needs. There are various types of database structures:
 - **Relational databases**-Relational databases have been around since the 1970s. The name comes from the way that data is stored in multiple, related tables. Within the tables, data is stored in rows and columns. Organizations that have a lot of **unstructured** or **semi-structured** data should not be considering a relational database. (Examples of unstructured data are: Rich media. Media and entertainment data, surveillance data, geo-spatial data, audio, weather data. Semi Structured Data Examples: Email 、 CSV 、 XML and JSON documents 、 HTML, and so on)
 - Microsoft SQL Server, Oracle Database, MySQL, PostgreSQL and IBM DB2

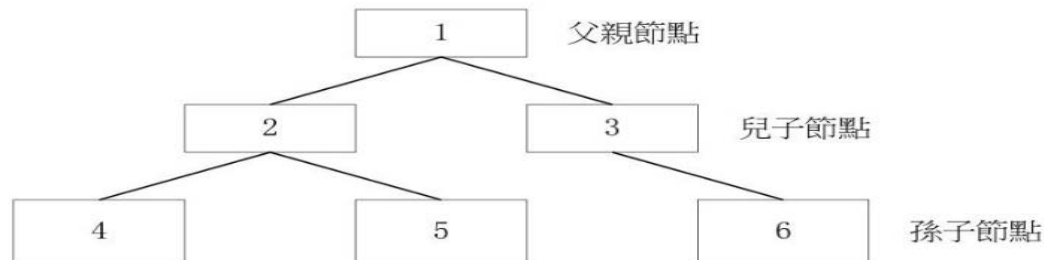
Types of Databases

- A company should use the type of database that fit in its requirements and needs. There are various types of database structures: (**We just introduce few types here!**)
 - **NoSQL** is a broad category that includes any database that doesn't use SQL as its primary data access language.
 - These types of databases are also sometimes referred to as non-relational databases. Unlike in relational databases, data in a NoSQL database doesn't have to conform to a pre-defined schema, so these types of databases are great for organizations seeking to store **unstructured** or **semi-structured** data.
 - One advantage of NoSQL databases is that developers can make changes to the database on the fly, without affecting applications that are using the database.
 - Apache Cassandra, MongoDB, CouchDB, and CouchBase

Types of Databases

- A company should use the type of database that fit in its requirements and needs. There are various types of database structures: (**We just introduce few types here!**)
 - **Hierarchical databases** use a parent-child model to store data. If you were to draw a picture of a hierarchical database, it would look like a family tree, with one object on top branching down to multiple objects beneath it.
 - **Examples:** IBM Information Management System (IMS), Windows Registry

■ 除了樹根以外的節點均只有一個直屬的「父親」節點



Types of Databases

- A company should use the type of database that fit in its requirements and needs. There are various types of database structures: (**We just introduce few types here!**)
 - **Document databases** Document databases, also known as document stores, use JSON-like documents to model data instead of rows and columns.
 - Sometimes referred to as document-oriented databases, document databases are designed to store and manage document-oriented information, also referred to as semi-structured data. Document databases are simple and scalable, making them useful for mobile apps that need fast iterations.
 - **Examples:** MongoDB, Amazon DocumentDB, Apache CouchDB

Why SQL Statements

What is a Relational Database (RDBMS)

- A relational database is a type of database that stores and provides access to data points that are related to one another.
- Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables.
- In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

Industry's Best RDBMS

- Oracle database products offer customers cost-optimized and high-performance versions of Oracle Database, the world's leading converged, multi-model database management system, as well as in-memory, NoSQL and MySQL databases.
- Oracle Autonomous Database enables customers to simplify relational database environments and reduce management workloads.
- Good procedural computer language: PL/ SQL



Benefits of RDBMS

- The simple yet powerful relational model is used by organizations of all types and sizes for a broad variety of information needs.
- Relational databases are used to track inventories, process ecommerce transactions, manage huge amounts of mission-critical customer information, and much more.
- A relational database can be considered for any information need in which data points relate to each other and must be managed in a secure, rules-based, consistent way.
- Relational databases have been around since the 1970s. Today, the advantages of the relational model continue to make it the most widely accepted model for databases.

Relational model and data consistency

- The relational model is the best at maintaining data consistency across applications and database copies (called instances). For example, when a customer deposits money at an ATM and then looks at the account balance on a mobile phone, the customer expects to see that deposit reflected immediately in an updated account balance. Relational databases excel at this kind of data consistency, ensuring that multiple instances of a database have the same data all the time.
- It's difficult for other types of databases to maintain this level of **timely consistency** with large amounts of data. Some recent databases, such as NoSQL, can supply only “**eventual consistency**.” Under this principle, when the database is scaled or when multiple users access the same data at the same time, the data needs some time to “catch up.” Eventual consistency is acceptable for some uses, such as to maintain listings in a product catalog, but for critical business operations such as shopping cart transactions, the relational database is still the gold standard.

The Differences between MySQL Workbench VS phpMyAdmin

Feature	MySQL Workbench	phpMyAdmin
Database	✓	✓
GUI Designer	✓	✓
Server Management	✓	✓
Spell checking	✓	✓
Data export/import	✓	✓
Autocompletion	✓	✓
Database Management	✓	✓
ER Diagrams	✓	✓
Lightweight	✓	✓
Data Report Wizard	✓	✓
MySQL Support	✓	✓
User interface	✓	✓
PostgreSQL support	✓	✓
Schema editor	✓	✓
Import CSV data	✓	✓

The Differences between MySQL Workbench VS phpMyAdmin

Feature	MySQL Workbench	phpMyAdmin
Data-management	✓	✓
Night mode/Dark Theme	✓	✓
Local based GUI	✓	✗
MariaDB	✓	✓
Backend	✗	✓
Charts	✗	✓
Code completion	✗	✓
Intellisense	✗	✓
Backup	✗	✓
Built-in editor	✗	✓
PhpMyAdmin	✗	✓

Q&A