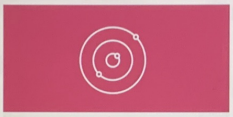
VIEW

What you will learn

Define a view



Describe when to use a view

Explain the syntax for creating a view

Define

- A view is a way of representing data in one or more table

show all column with all table

choose some table of some column

What is a view?

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, ...	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry Ln,	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs,	300	50000	30002	5
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora,	400	60000	30004	5

What is a view?

EMP_ID	F_NAME	L_NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1001	John	Thomas	5631 Rice, ...	100	30001	2
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5
E1004	Santosh	Kumar	511 Aurora,	400	30004	5

another views are included

- Can include specified columns from multiple base tables and existing views
- Once created, can be queried like a table
- Only the definition of the view is stored, not the data

When

When should you use a view?

Views can:

- Show a selection of data for a given table → so it can omit some sensitive data.
- Combine two or more tables in meaningful ways
- Simplify access to data → access to data by granting access to a view without granting access to the underlying tables
- Show only portions of data in the table

Example:

- Create a view to display non-sensitive data from the Employees table

Syntax

CREATE VIEW statement

```
CREATE VIEW <view name> (<column_alias_1>,
<column_alias_2>, … <column_alias_n>)
  AS SELECT <column_1> , <column_2>, … <column_n>
  FROM <table name>
  WHERE <predicate>;
```

Task 2: Update a View

In this exercise, you will update a View to combine two or more tables in meaningful ways.

Assume that the `EMPSALARY` view we created in Task 1 doesn't contain enough salary information, such as max/min salary and the job title of the employees. For this, we need to get information from other tables in the database. You need all columns from `EMPLOYEES` table used above, except for `SALARY`. You also need the columns `JOB_TITLE`, `MIN_SALARY`, `MAX_SALARY` of the `JOBS` table.

The command to be used is as follows:

to update view — view's table

```
1  CREATE OR REPLACE VIEW EMPSALARY AS
2  SELECT EMP_ID, F_NAME, L_NAME, B_DATE, SEX, JOB_TITLE,
3  MIN_SALARY, MAX_SALARY
4  FROM EMPLOYEES, JOBS
5  WHERE EMPLOYEES.JOB_ID = JOBS.JOB_IDENT;
```

syntax

CREATE VIEW statement

```
CREATE VIEW EMPINFO (EMP_ID, FIRSTNAME, LASTNAME, ADDRESS,
JOB_ID, MANAGER_ID, DEP_ID)
  AS SELECT EMP_ID, F_NAME, L_NAME, ADDRESS, JOB_ID,
MANAGER_ID, DEP_ID
  FROM EMPLOYEES;

SELECT * FROM EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1001	John	Thomas	5631 Rice, ...	100	30001	2
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5
E1004	Santosh	Kumar	511 Aurora,	400	30004	5

Working with views

```
CREATE VIEW EMPINFO (EMP_ID, FIRSTNAME, LASTNAME, ADDRESS,
JOB_ID, MANAGER_ID, DEP_ID)
  AS SELECT EMP_ID, F_NAME, L_NAME, ADDRESS, JOB_ID,
MANAGER_ID, DEP_ID
FROM EMPLOYEES
WHERE MANAGER_ID = '30002'

SELECT * FROM EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5

Working with views

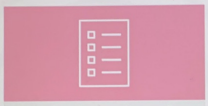
```
DROP VIEW EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5

Stored Procedures

What you will learn

Explain what a stored procedure is



List the benefits of using stored procedures

Describe how to create and use a stored procedure

What is it?

What is a stored procedure?

- A set of SQL statements stored and executed on the database server
  - Write in many different languages
  - Accept information in the form of parameters
  - Return results to the client

Explain

- Instead sending multiple sql statement from client to server, we can encapsulate those statement in a stored procedure on the server
- From Encapsulation (stored procedure), we can use one statement from client to execute them in sever

List the benefits

Benefits of stored procedures

- Reduction in network traffic and latency, because it needed one statement to call
- Improvement in performance
- Reuse of code → multiple app can use the same stored procedure
- Increase in security → we can choose some logic in for security and privacy

How to create?

$$ means to start defining the procedure

CREATE PROCEDURE statement

to start the procedure, it is prudent to change the delimeter

```
DELIMITER $$
CREATE PROCEDURE UPDATE_SAL (IN empNum CHAR(6), IN rating
SMALLINT)
BEGIN
UPDATE employees SET salary = salary * 1.10 WHERE emp_id
= empNum AND rating = 1;
UPDATE employees SET salary = salary * 1.05 WHERE emp_id
= empNum AND rating <> 1;
END
$$
DELIMITER ;
```

DELIMITER $$ → name's procedure

BEGIN begin writing statement logic

END closed

$$ → means stop here

DELIMITER ; → change back to ; when we are done

Working with stored procedures

Call from:

- External applications
- Dynamic SQL statements

```
CALL UPDATE_SAL ('E1001', 1)
```

procedure name — ID — rating

Procedure's Lab

Ex 1.

1. You will create a stored procedure routine named **RETRIEVE_ALL**.
   - This **RETRIEVE_ALL** routine will contain an SQL query to retrieve all the records from the PETSALE table, so you don't need to write the same query over and over again. You just call the stored procedure routine to execute the query everytime.
   - To create the stored procedure routine, copy the code below and paste it to the textarea of the **SQL** page. Click **Go**.

Create

```
DELIMITER //

CREATE PROCEDURE RETRIEVE_ALL()

BEGIN
    SELECT * FROM PETSALE;
END //
DELIMITER ;
```

Call

```
CALL RETRIEVE_ALL;
```

Drop

```
DROP PROCEDURE RETRIEVE_ALL;

CALL RETRIEVE_ALL;
```

Stored Procedure: Exercise 2

In this exercise, you will create and execute a stored procedure to write/modify data in a table on MySQL using SQL.

You will create a stored procedure routine named **UPDATE_SALEPRICE** with parameters **Animal_ID** and **Animal_Health**.

- This **UPDATE_SALEPRICE** routine will contain SQL queries to update the sale price of the animals in the PETSALE table depending on their health conditions, **BAD** or **WORSE**.
- This procedure routine will take animal ID and health conditon as parameters which will be used to update the sale price of animal in the PETSALE table by an amount depending on their health condition. Suppose that:
  - For animal with ID XX having BAD health condition, the sale price will be reduced further by 25%.
  - For animal with ID YY having WORSE health condition, the sale price will be reduced further by 50%.
  - For animal with ID ZZ having other health condition, the sale price won't change.
- To create the stored procedure routine, copy the code below and paste it to the textarea of the **SQL** page. Click **Go**.

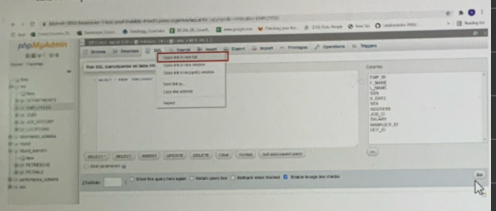
```
DELIMITER @
CREATE PROCEDURE UPDATE_SALEPRICE (IN Animal_ID INTEGER, IN Animal_Health VARCHAR(10))
BEGIN
    IF Animal_Health = 'BAD' THEN
        UPDATE PETSALE
        SET SALEPRICE = SALEPRICE - (SALEPRICE * 0.25)
        WHERE ID = Animal_ID;
    ELSEIF Animal_Health = 'WORSE' THEN
        UPDATE PETSALE
        SET SALEPRICE = SALEPRICE - (SALEPRICE * 0.5)
        WHERE ID = Animal_ID;
    ELSE
        UPDATE PETSALE
        SET SALEPRICE = SALEPRICE
        WHERE ID = Animal_ID;
    END IF;
END @
DELIMITER ;
```

we can use any symbol to change ; to "any symbol"

to signify the end of 'if'

change to any symbol because stored procedure it is a complex structure that contain ';' that means end the statement, if we used ';' to end the stored pro. it would get an Error, so we need to change the semicolon to another symbol to prevent this conflic

1. Let's call the UPDATE_SALEPRICE routine. We want to update the sale price of animal with ID **1** having **BAD** health condition in the PETSALE table. open another **SQL** tab by clicking **Open in new Tab**



Delete the default line which appears so that you will get a blank window.

Copy the code below and paste it to the textarea of the **SQL** page. Click **Go**.

Note if you have dropped RETREIVE_ALL procedure rerun the creation script of that procedure before executing these lines.

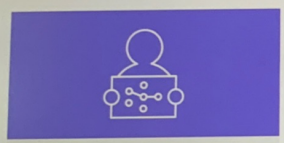
```
CALL RETRIEVE_ALL;

CALL UPDATE_SALEPRICE(1, 'BAD');

CALL RETRIEVE_ALL;
```

2. Let's call the UPDATE_SALEPRICE routine once again. We want to update the sale price of animal with ID **3** having **WORSE** health condition in the PETSALE table. copy the code below and paste it to the textarea of the **SQL** page. Click **Go**. You will have all the records retrieved from the PETSALE table.

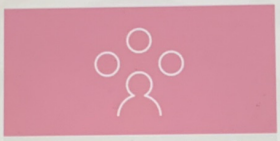
```
CALL RETRIEVE_ALL;

CALL UPDATE_SALEPRICE(3, 'WORSE');

CALL RETRIEVE_ALL;
```

ACID Transaction

What you will learn



Explain what an ACID transaction is

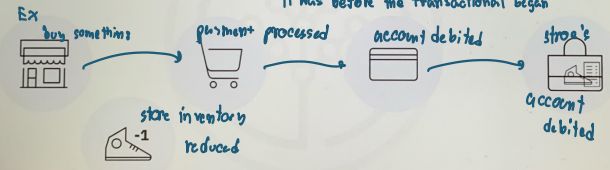


Give an example of an ACID transaction

Describe commits and rollbacks

What is a transaction?

- Indivisible unit of work → ต้องทำงานร่วมกันทั้งหมด
- Consists of one or more SQL statements
- Either all happens or none → ต้องเกิดทั้งหมดหรือไม่เกิดเลย and roll back to it was before the transactional began



Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE BankAccounts
SET Balance = Balance-200
WHERE AccountName = 'Rose';
```

Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE BankAccounts
SET Balance = Balance+200
WHERE AccountName = 'Shoe Shop';
```

Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE ShoeShop
SET Stock = Stock-1
WHERE Product = 'Boots';
```

Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

If any of these UPDATE statements fail, the whole transaction must fail ** to keep the data in a consistent state

What is an ACID transaction?

ACID stands for

Atomic

All changes must be performed successfully or not at all.

Consistent

Data must be in a consistent state before and after the transaction.

Isolated

No other process can change the data while the transaction is running.

Durable

The changes made by the transaction must persist.

ACID commands

- BEGIN
  - Start the ACID transaction
- COMMIT
  - All statements complete successfully
  - Save the new database state

success

```
BEGIN
    UPDATE BankAccounts
    SET Balance = Balance – 200
    WHERE AccountName = 'Rose';

    UPDATE BankAccounts
    SET Balance = Balance + 200
    WHERE AccountName = 'Shoe Shop';

    UPDATE ShoeShop
    SET Stock = Stock - 1
    WHERE Product = 'Boots';
COMMIT
```

ACID commands

- BEGIN
  - Start the ACID transaction
- COMMIT
  - All statements complete successfully
  - Save the new database state
- ROLLBACK
  - One or more statements fail
  - Undo changes

fail

```
BEGIN
    UPDATE BankAccounts
    SET Balance = Balance – 200
    WHERE AccountName = 'Rose';

    UPDATE BankAccounts
    SET Balance = Balance + 200
    WHERE AccountName = 'Shoe Shop';

    UPDATE ShoeShop
    SET Stock = Stock - 1
    WHERE Product = 'Boots';
ROLLBACK
```

ACID must be

Calling ACID commands

- Can also be issued by some languages
  - Java, C, R, and Python
  - Requires the use of database specific APIs or connectors
- Use the EXEC SQL command to execute SQL statements from code

```
void main ()
{
    EXEC SQL  UPDATE BankAccounts
      SET Balance = Balance – 200
      WHERE AccountName = 'Rose';
    EXEC SQL  UPDATE BankAccounts
      SET Balance = Balance + 200
      WHERE AccountName = 'Shoe Shop';
EXEC SQL UPDATE ShoeShop
      SET Stock = Stock - 1
      WHERE Product = 'Boots';
  FINISHED:
    EXEC SQL  COMMIT WORK;
    return;

  SQLERR:
    EXEC SQL  WHENEVER SQLERROR CONTINUE;
    EXEC SQL  ROLLBACK WORK;
    return;
}
```