Views and Indices

Thomas Schwarz, SJ

Virtual Views

- Relations can be <u>real</u>
  - CREATE TABLE …
- or <u>virtual</u>
  - CREATE VIEW
    - Do not exist physically
    - Defined through a query like expression
    - Can be queried as if they are real tables

Virtual Views

- SQL Programming Language:
  - Table: Relation that exists
  - View: Relation that is virtual
  - Temporary: Created while a query is executed and afterwards discarded

Virtual Views

- Views are defined via CREATE VIEW

```
CREATE VIEW MGMMovies AS
     SELECT title, year
     FROM Movies
     WHERE studioName = 'MGM';
```

Virtual Views

```
movies(title, year, length, genre, studioName, producerC#)
        movieExec(name, address, cert#, netWorth)
```

```
CREATE VIEW MovieProd AS
     SELECT title, name
     FROM movies, movieExec
     WHERE producerC# = cert#;
```

Virtual Views

- Interacting with Views
  - A view, once defined, can be queried just like a real table

```
SELECT title
FROM MGMMovies
WHERE year = 1979;
```

Virtual Views

```
starName(title, year, name)
```

```
SELECT DISTINCT starName
FROM MGMMovies, starsIn
WHERE title = movieTitle AND year = movieYear
```

Virtual Views

- We can rename the attributes in a VIEW

```
CREATE VIEW movieProd(movieTitle, prodName) AS
      SELECT title, name
      FROM movies, movieExec
      WHERE producerC# = cert#;
```

- attribute names in the view are now movieTitle and prodName

Exercises

```
movieStar(name, address, gender, birthday)
movieExec(name, address, cert#, netWorth)
studio(name, address, presC#)
```

- A view RichExec with name address, certificate number, and net-worth of all executives with more than 10 million net-worth

Exercises

```
movieStar(name, address, gender, birthday)
movieExec(name, address, cert#, netWorth)
studio(name, address, presC#)
```

- A view RichExec with name, address, certificate number, and net-worth of all executives with more than 10 million net-worth

```
CREATE VIEW RichExec(execName, execAddress, cert#,
netWorth) AS
     SELECT name, address, cert#, netWorth
     WHERE netWorth > 10000000;
```

Exercises

```
movieStar(name, address, gender, birthday)
movieExec(name, address, cert#, netWorth)
studio(name, address, presC#)
```

- A view StudioPres with name, address, netWorth of studio presidents

Exercises

```
movieStar(name, address, gender, birthday)
movieExec(name, address, cert#, netWorth)
studio(name, address, presC#)
```

- A view StudioPres with name, address, netWorth of studio presidents

```
CREATE VIEW StudioPres AS
      SELECT name, address, netWorth
      FROM movieExec
      WHERE cert# IN (
            SELECT presC#
            FROM studio );
```

Exercises

```
movieStar(name, address, gender, birthday)
movieExec(name, address, cert#, netWorth)
studio(name, address, presC#)
```

- A view ExecutiveStar giving the name, address, gender, birth date and certificate number of movie stars that are also movie executives
  - Assume that executives with the same name and address as a movie star are the movie star
    - Even though there is no reason to assume this

Exercises

```
movieStar(name, address, gender, birthday)
movieExec(name, address, cert#, netWorth)
studio(name, address, presC#)
```

- A view ExecutiveStar giving the name, address, gender, birth date and certificate number of executives that are also movie executives

```
CREATE VIEW ExecutiveStar AS
     SELECT ms.name, ms.address, ms.gender,
        ms.birthdate, me.cert#
     FROM movieStar ms, movieExec me
     WHERE ms.name = ms.name AND ms.address = me.address
```

Modifying Views

- Some views can be used to update the underlying tables
- View Removal
  - `DROP VIEW MGMMovies`
- Just like Table removal
  - `DROP TABLE movies`
  - which would also make the view MGMMovies unusable

Modifying Views

- Updatable views
  - SQL has clear, but complicated definitions when a view can be updated (and an underlying table changed)
    - View must be defined by SELECT
    - There is only one relation R in the definition
    - No subquery involving R in the WHERE clause
    - Enough attributes of R are involved in the view

Modifying Views

- MGMMovies fulfills the requirements
- If we insert via the view:

  -
    ```
    INSERT INTO MGMMovies
    VALUES('Get Shorty', 1995)
    ```
  - movies will get a new tuple
    - title: 'Get Shorty', year: 1995
    - Everything else: Null
- Interestingly, because of the latter, the view itself would <u>not be updated</u>

```
movies(title, year, length, genre, studioName, producerC#)
```

Modifying Views

- The view insertion

```
INSERT INTO MGMMovies
VALUES('Get Shorty', 1995)
```

- has the same effect as inserting into the underlying table

```
INSERT INTO movies
VALUES('Get Shorty', 1995)
```

Modifying Views

- To address this anomaly, need to add to the view

```
CREATE OR REPLACE VIEW MGMMovies(name, title, studio) AS
   SELECT name, title, studioName
   FROM movies
   WHERE studio = 'MGM';
```

Modifying Views

- Now it works

```
INSERT INTO MGMMovies
VALUES('Find Shorty', 1995, 'MGM')
```

- which is equivalent to

```
INSERT INTO movies(name, year, studioName)
VALUES ('Find Shorty', 1995, 'MGM')
```

    - and assumes that we do not have any triggers or constraints against NULL values for the other attributes

- but now the view also changes

Modifying Views

- Deletions are also passed through the underlying table

  -
    ```
    DELETE FROM MGMMovies
    WHERE title LIKE '%Shorty%';
    ```
- gets translated into

```
DELETE FROM movies
WHERE title LIKE '%Shorty%' AND studioName = 'MGM';
```

Modifying Views

```
UPDATE MGMMovies
SET year = 1968
WHERE title = 'Get Shorty';
```

- becomes

```
UPDATE movies
SET year = 1968
WHERE title = 'Get Shorty' AND
      studioName = 'MGM';
```

Modifying Views

- Including all properties in a view is a kludge
  - Can use a trigger instead
    - Use the INSTEAD OF syntax

```
CREATE TRIGGER mgmInserts
INSTEAD OF INSERT ON mgmInserts
REFERENCING NEW ROW as newRow
FOR EACH ROW
INSERT INTO movies(title, year, studioName)
VALUES(newRow.title, newRow.year, 'MGM');
```

Modifying Views in MySQL

- MySQL only started to support views in Version 5 (2008)
- Supports updatable views
    - But not the INSTEAD trigger

Try It Out

- Use the employees database in MySQL
  - You might want to turn of automatic commits, then do a commit and at the end of the session a rollback
  - Task 1: Convince yourself that there are no emp_no larger than 500000

Try It Out

```
USE employees;

SELECT *
FROM dept_emp
WHERE emp_no >=500000;
```

Try It Out

- Task 2: Insert three persons into the employees table with employee numbers 600000, 600001, 600002. You can invent the missing dates.
- The hire date should be the day of today
  - In MySQL that is CURDATE( )

Try It Out

```
INSERT INTO employees(emp_no, birth_date, first_name,
last_name, gender, hire_date)
VALUES
    (600000, '1980-01-01', 'Hector', 'Garcia Molinas',
'M', CURDATE()),
    (600001, '1981-01-01', 'Ursula', 'Leyendorf', 'F',
CURDATE()),
    (600002, '1982-01-01', 'Bob', 'Karragher', 'M',
CURDATE());
```

Try It Out

- Create a view of dept_emp that only contains entries with to_date unlimited
  - i.e. '9999-01-01' which is used to indicate an open contract.
  - Call the view v_current_dept_emp
    - Include all attributes so that we can update

Try It Out

```
CREATE OR REPLACE VIEW v_current_dept_emp AS
   SELECT emp_no, dept_no, from_date, to_date
     FROM dept_emp
     WHERE to_date = '9999-01-01';
```

Try It Out

- Now insert the three new employees into the view
    - from_date is today
    - Department is 'd004'

Try It Out

```
INSERT INTO v_current_dept_emp(emp_no, dept_no, from_date,
to_date)
VALUES
    (600000, 'd004', CURDATE(), '9999-01-01'),
    (600001, 'd004', CURDATE(), '9999-01-01'),
    (600002, 'd004', CURDATE(), '9999-01-01');
```

Try It Out

- Check that these updates made it to the dept_emp table as well as the view

Try It Out

```
SELECT *
FROM v_current_dept_emp
WHERE emp_no >=500000;

SELECT *
FROM dept_emp
WHERE emp_no >=500000;
```

Try It Out

- Change the view v_current_dept_emp to have only three columns: emp_no, dept_no, from_date by recreating it

Try It Out

```
CREATE OR REPLACE VIEW v_current_dept_emp AS
   SELECT emp_no, dept_no, from_date
     FROM dept_emp
     WHERE to_date = '9999-01-01';
```

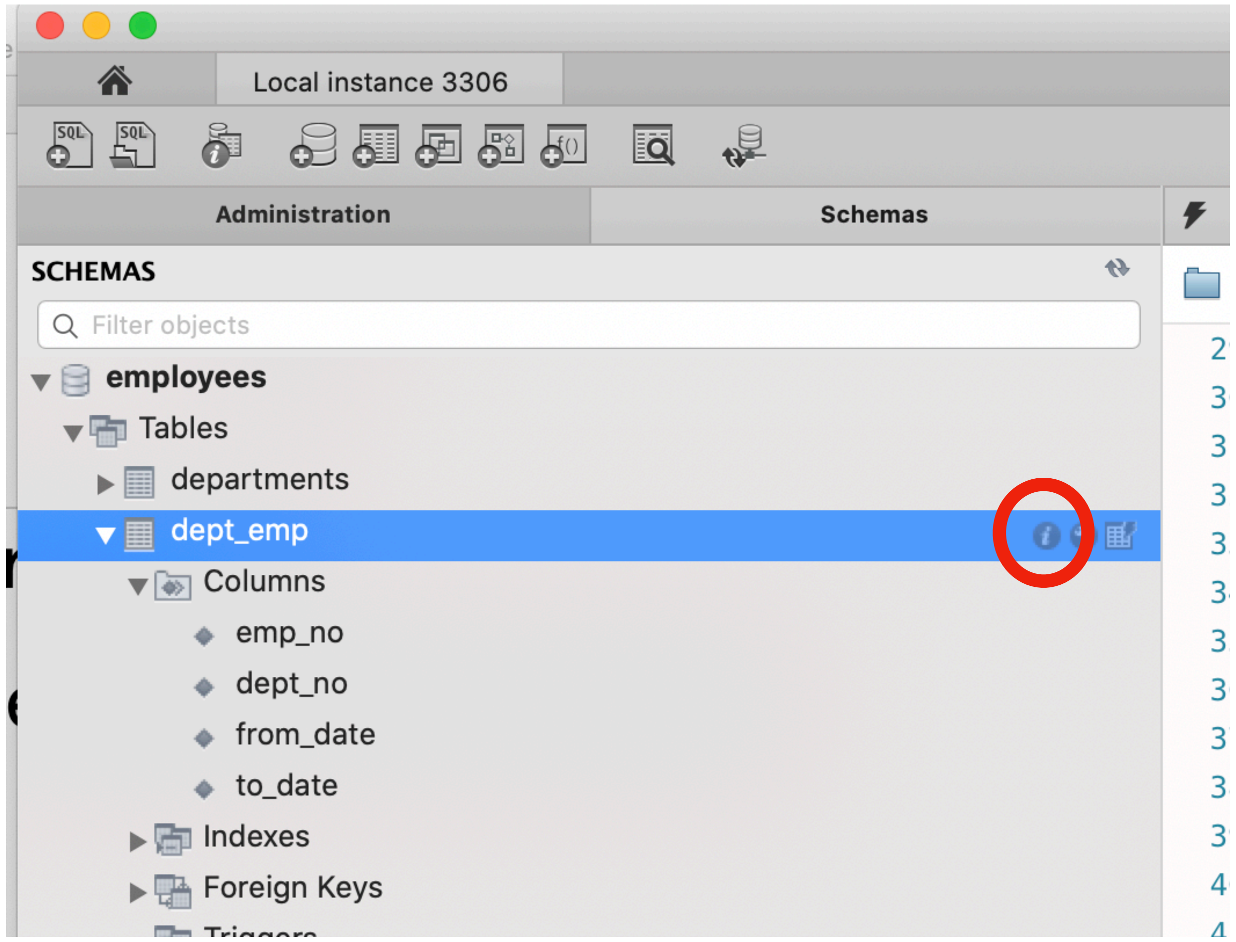
- The CREATE OR REPLACE clause makes it easy.
- You could also say DROP VIEW and then do a CREATE VIEW

Try It Out

- Check the table dept_emp for its definition

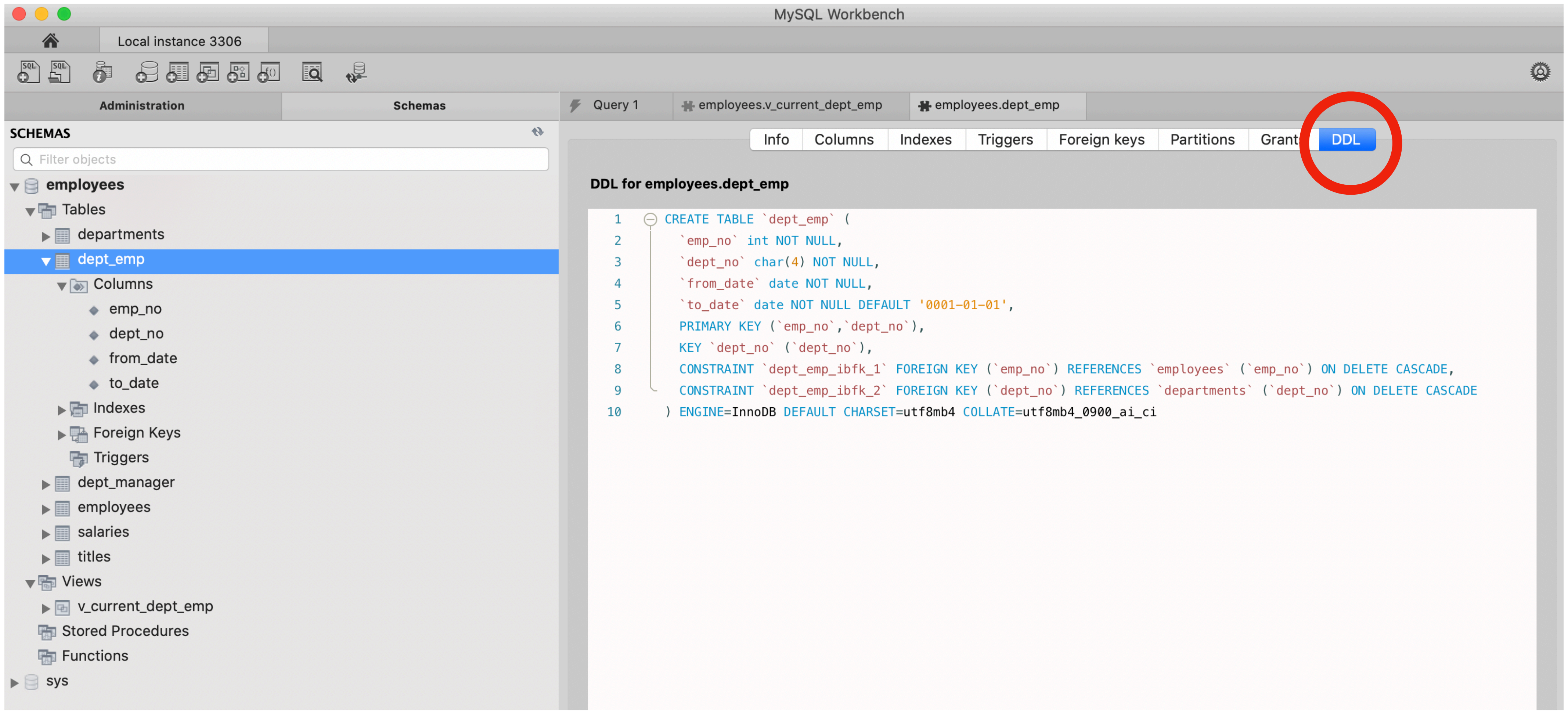
Try It Out

- In MySQLWorkbench:
  - Click on the table and the info tab



Try It Out

- In the view, select DDL, which gives you the definition of the table

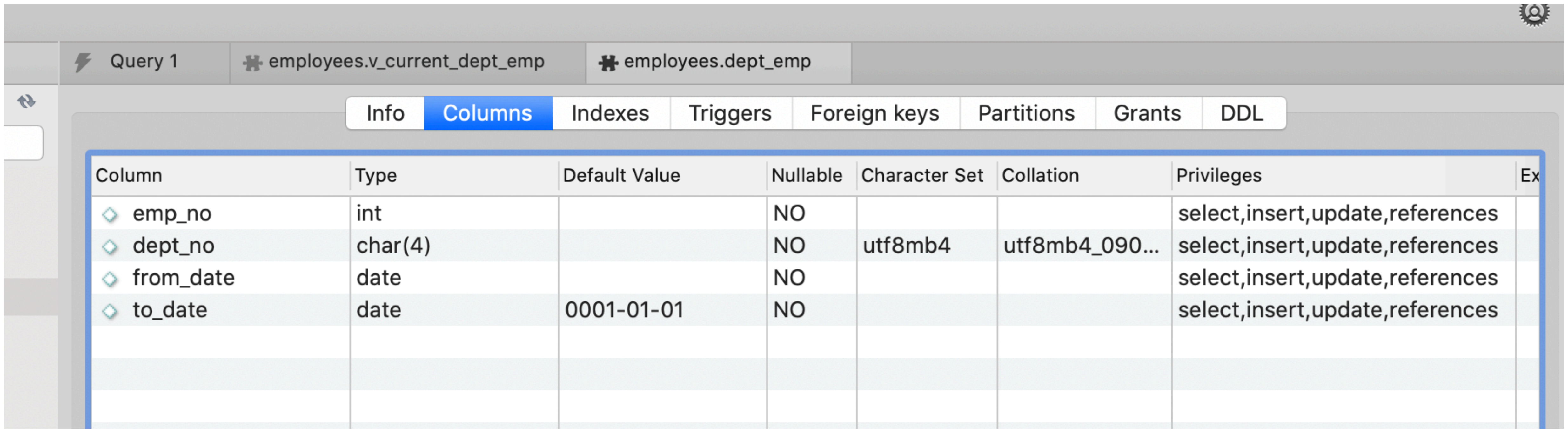


DDL for employees.dept_emp

```
CREATE TABLE `dept_emp` (
  `emp_no` int NOT NULL,
  `dept_no` char(4) NOT NULL,
  `from_date` date NOT NULL,
  `to_date` date NOT NULL DEFAULT '0001-01-01',
  PRIMARY KEY (`emp_no`,`dept_no`),
  KEY `dept_no` (`dept_no`),
  CONSTRAINT `dept_emp_ibfk_1` FOREIGN KEY (`emp_no`) REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,
  CONSTRAINT `dept_emp_ibfk_2` FOREIGN KEY (`dept_no`) REFERENCES `departments` (`dept_no`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Try It Out

- Alternatively, you can select columns



Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Ex
emp_no	int		NO			select,insert,update,references	
dept_no	char(4)		NO	utf8mb4	utf8mb4_090...	select,insert,update,references	
from_date	date		NO			select,insert,update,references	
to_date	date	0001-01-01	NO			select,insert,update,references	

- Both methods show that we have a NOT NULL constraint on to_date

Try It Out

- Alter the table dept_emp to have a default value of '9999-01-01' in the to_date.
    - We could also remove the NOT NULL restriction

Try It Out

```
ALTER TABLE dept_emp
MODIFY COLUMN to_date date NOT NULL DEFAUlT '1-01-01';
```

Try It Out

- If we try to add directly to the table with new values, we violate a foreign key constraint.

```
INSERT INTO v_current_dept_emp(emp_no, dept_no, from_date)
VALUES
   (600003, 'd004', CURDATE()),
     (600004, 'd004', CURDATE()),
     (600005, 'd004', CURDATE());
```

Try It Out

- Create a few more employees in the employee table
    - With emp_no larger than 600000

Try It Out

```
INSERT INTO employees(emp_no, birth_date, first_name,
last_name, gender, hire_date)
VALUES
    (600003, '1980-01-01', 'Javier', 'GPena', 'M',
CURDATE()),
    (600004, '1981-01-01', 'Dick', 'Murphy', 'M',
CURDATE()),
    (600005, '1982-01-01', 'Emilio', 'Zapato', 'M',
CURDATE());
```

Try It Out

```
INSERT INTO employees(emp_no, birth_date, first_name,
last_name, gender, hire_date)
VALUES
    (600003, '1980-01-01', 'Javier', 'Pena', 'M',
CURDATE()),
    (600004, '1981-01-01', 'Dick', 'Murphy', 'M',
CURDATE()),
    (600005, '1982-01-01', 'Emilio', 'Zapato', 'M',
CURDATE());
```

Try It Out

- Check that this updates the dept_emp table correctly

Try It Out

- MySQL trigger mechanisms are not so great!

Example

Example

- We create a sub-scheme of the classic models database

```
CREATE database HW7;

USE HW7;

DROP  TABLE If EXISTS clients;

CREATE TABLE clients(
    client_no INT PRIMARY KEY AUTO_INCREMENT,
    client_name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL
);
```

Example

```
DROP TABLE IF EXISTS orders;

CREATE TABLE orders(
   ord_no INT PRIMARY KEY AUTO_INCREMENT,
     client_no INT,
     date_received DATE NOT NULL,
     date_shipped DATE DEFAULT '9999-01-01',
     CONSTRAINT order_needs_client
        FOREIGN KEY (client_no)
        REFERENCES clients(client_no)
     );
```

Example

```
DROP TABLE IF EXISTS items;

CREATE TABLE items (
    item_no INT PRIMARY KEY AUTO_INCREMENT,
    item_name VARCHAR(64) NOT NULL,
    description VARCHAR(255) DEFAULT ''
);
```

Example

```
DROP TABLE IF EXISTS order_details;

CREATE TABLE order_details (
    ord_no INT NOT NULL,
    item_no INT NOT NULL,
    quantity INT,
    price DECIMAL(10 , 2 ),
    PRIMARY KEY (ord_no , item_no),
    CHECK (price > 0),
    CHECK (quantity > 0),
    CONSTRAINT od_needs_order FOREIGN KEY (ord_no)
        REFERENCES orders (ord_no),
    CONSTRAINT od_needs_item FOREIGN KEY (item_no)
        REFERENCES items (item_no)
);
```

Example

```
INSERT INTO items(item_name) VALUES
('F14'),
('F16'),
('F35'),
('Eurofighter Typhoon'),
('Trump'),
('Saab Gripen'),
('Dassault Rafaele');
```

Example

```
INSERT INTO clients(client_name, address) VALUES
('Canada', 'Ottawa, ONT'),
('Portugal', 'Lisboa'),
('Netherlands', 'Den Haag'),
('Norway', 'Oslo'),
('France', 'Paris'),
('Belgium', 'Bruxelles');
```

Example

```
INSERT INTO orders(client_no, date_received)
SELECT client_no, DATE(NOW())
FROM clients
WHERE client_name = 'Portugal';
```

Example

```
SELECT ord_no, item_no, 15, 35000
FROM orders JOIN clients USING (client_no) JOIN items
WHERE client_name = 'Portugal' and date_received =
DATE(NOW()) and item_name LIKE 'Das%';

INSERT INTO order_details(ord_no, item_no, quantity,
price)
SELECT ord_no, item_no, 15, 3500
FROM orders JOIN clients USING (client_no) JOIN items
WHERE client_name = 'Portugal' and date_received =
DATE(NOW()) and item_name = 'F35';
```

Example

```
INSERT INTO orders(client_no, date_received)
SELECT client_no, '2025-01-01'
FROM clients
WHERE client_name = 'BELGIUM';

INSERT INTO orders(client_no, date_received)
SELECT client_no, '2025-02-01'
FROM clients
WHERE client_name = 'Netherlands';
```

Example

```
INSERT INTO order_details(ord_no, item_no, quantity,
price)
SELECT ord_no, item_no, 13, 2400
FROM orders JOIN clients USING (client_no) JOIN items
WHERE client_name = 'Belgium' and date_received =
'2025-01-01' and item_name LIKE 'Das%';

INSERT INTO order_details(ord_no, item_no, quantity,
price)
SELECT ord_no, item_no, 25, 3000
FROM orders JOIN clients USING (client_no) JOIN items
WHERE client_name = 'Netherlands' and date_received =
'2025-02-01' and item_name LIKE 'Das%';
```

Example

```
INSERT INTO orders(client_no, date_received)
SELECT client_no, '2026-01-01'
FROM clients
WHERE client_name = 'Portugal';

INSERT INTO order_details(ord_no, item_no, quantity,
price)
SELECT ord_no, item_no, 5, 5000
FROM orders JOIN clients USING (client_no) JOIN items
WHERE client_name = 'Portugal' and date_received =
'2026-01-01' and item_name LIKE 'F14';
```

Example

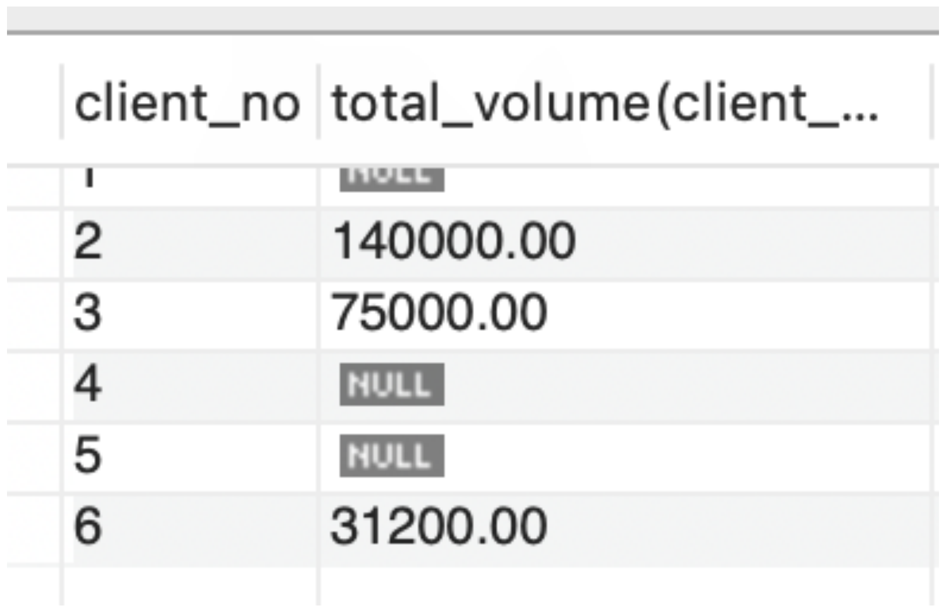
```
DELIMITER $$

CREATE FUNCTION total_volume (
   my_client_no INT
)
  RETURNS DECIMAL(10,2)
  READS SQL DATA
  BEGIN
       DECLARE tv DECIMAL(10,2);
       SELECT SUM(quantity*price)
       INTO tv
       FROM orders JOIN order_details USING(ord_no)
       WHERE client_no = my_client_no;
       RETURN tv;
  END $$

 DELIMITER ;
```

Example

```
SELECT client_no, total_volume(client_no)
FROM clients;
```



client_no	total_volume(client_...
1	NULL
2	140000.00
3	75000.00
4	NULL
5	NULL
6	31200.00

Example

- Creating a trigger for inserts into order details
- We create a table instead of a materialized view

```
DROP TABLE IF EXISTS gold;

CREATE TABLE gold (
   client_no INT PRIMARY KEY
);
```

```
DELIMITER //
CREATE TRIGGER gold_trigger
AFTER INSERT
ON order_details
FOR EACH ROW
BEGIN
     DECLARE client_number INT;

    SELECT client_no
    INTO client_number
    FROM order_details JOIN orders USING (ord_no)
    WHERE ord_no = NEW.ord_no;

    IF total_volume(client_number) > 1000
    THEN
    INSERT INTO gold(client_no)
    VALUES (client_number);
    END IF;

END;
//
```

Example

```
INSERT INTO orders(client_no, date_received)
SELECT client_no, '2022-04-01'
FROM clients
WHERE client_name = 'Norway';

INSERT INTO order_details(ord_no, item_no, quantity,
price)
SELECT ord_no, item_no, 100, 4000
FROM orders JOIN clients USING (client_no) JOIN items
WHERE client_name = 'Norway' and date_received =
'2022-04-01' and item_name LIKE 'Saab%';
```