



UPPSALA  
UNIVERSITET

# F2: Stored Procedures & Triggers

Databaser 2

Jakob Bandelin

Institutionen för informatik och media

# Tidigare

- Lab 0 - Khan SQL
- Lab 1 - Views
- Frågor?
- Kursforum. Email.

# Idag

- MySQL
- Stored Procedures
- Triggers
- Lab 2

# MySQL

- Förr eller senare kommer sqlfiddle inte räcka
- Ni behöver skaffa tillgång till MySQL
- Enklast (tycker jag) MAMP/WAMP/LAMP och phpMyAdmin
- Finns andra lösningar
- Utgår ifrån att de flesta klarar att installera själva.

# En taktik för att lära sig AKA Skriva vs GUI

- För bättre inlärning skriv SQL själv
- GUI är inte dåligt att använda - men förståelse och kunskap blir djupare/bättre om vi kan skriva SQL
- Förståelse för vad GUI kan göra ökar också om vi kan skriva vår SQL själva
- Också enklare att dela kod/exempel/frågor byta DB och miljö
  - Och det behövs för redovisning av laborationerna

# Stored routines

- Eventuellt har ni kört kod mot en DB tidigare:
  - C, PHP, Python, Java, .NET, osv
- Stored procedures/routines kan göra vissa av de saker som externa språk kan göra
- Blir en utökning av SQL
- Olika DBMS har olika syntax. Principerna är samma.
- Vi kommer använda MySQLs sätt på den här kursen.
- Det är inte så avancerad kod (jämfört med vad som kan dyka upp i annan programmering).

# Routines? Procedures? Functions?

- Routines
  - Färdigt mönster för att göra något
  - Functions och procedures är båda routines
  - Stored routines kan vara procedure eller function
  - Färdiga program/kod-delar som kan anropas och köras
- Procedure
  - Färdig SQL-kod att anropa
  - Används rätt ofta som samlingsnamn för stored routines
  - Körs med CALL proc\_name();
- Function
  - Returnerar ett värde
  - Körs i SQL-statement

# Sproc - Fördelar

- Vi kan bygga ett interface eller API mot vår DB
  - Säkerhet
  - Flexibilitet
- Ibland är även DBMS bättre på att göra operationer mot DB än vad externa språk är.
- De är snabbare - mindre kommunikationstid
- Snabbare pga frågeoptimering

# Sproc - nackdelar

- Ökar belastning på DB-server
- Svåra att debugga, tunga att underhålla och uppdatera
- Ev svårare att hitta en kompetent DBA än en programmerare för ex java

# En första procedure

```
DELIMITER //
CREATE PROCEDURE movies()
BEGIN
SELECT * FROM Movies;
END //
DELIMITER ;
```

```
CALL movies();
```

# Delimiter

- Default använder vi semikolon för att avsluta kommando/statement
- Med DELIMITER kan vi ändra från semikolon till andra tecken.
- Behövs när vi ska skriva sproc.

```
DELIMITER //
CREATE PROCEDURE movies()
BEGIN
SELECT * FROM Movies;
END //
DELIMITER ;
```

# Variabler

- Enkelhet

```
SET @topratinglimit = 9;  
SELECT * FROM Movies WHERE rating >= @topratinglimit;
```

```
SELECT @highestrating := (SELECT max(rating) FROM Movies);
```

- Renare kod, ändra på ett ställe  
uppdaterar flera ställen
- Lättare att läsa och förstå
- Lagring/användning av okänd  
data
- Lagring/användning av värden  
som vi räknar ut

# Variabler - scope

- Räckvidd
  - @variable - användardskapad tillgänglig inom samma inlogg/session
  - Variabler deklarerade i sproc har bara räckvidd i sproc.

# Variabler, exempel

```
SET @start = 2010, @end = 2018;  
  
SELECT * FROM Movies WHERE year BETWEEN @start AND @end;
```

```
SELECT @start := 2010, @end := 2017;  
  
SELECT * FROM Movies WHERE year BETWEEN @start AND @end;
```

# Variabler i procedurer

```
DROP PROCEDURE IF EXISTS some_movies;

DELIMITER //
CREATE PROCEDURE some_movies()
BEGIN
    DECLARE some INT DEFAULT 10;
    SET some := 5;
    SELECT * FROM Movies LIMIT some;
END //
DELIMITER ;

CALL some_movies();
```

# Variabler som parametrar

```
DELIMITER //

CREATE PROCEDURE movies_from_years(start_year INT, end_year int)
BEGIN
    SELECT title FROM Movies WHERE year BETWEEN start_year AND finish_year;
END; //

DELIMITER ;

CALL movies_from_years(2000, 2010);
```

```
DROP PROCEDURE IF EXISTS movies_from_years;
```

# Parametrar

- IN (default) - SP jobbar mot kopia av värdet  
(För functions är alla parametrar IN)
- OUT - kan ändras och nya värdet skickas ut från sproc
- INOUT - kombination

```
DROP PROCEDURE IF EXISTS set_counter;

DELIMITER //
CREATE PROCEDURE set_counter (INOUT count INT, IN increase_by INT)
BEGIN
    SET count = count + increase_by;
END //
DELIMITER ;

SET @counter = 1;
CALL set_counter(@counter, 1); -- 2
CALL set_counter(@counter, 2); -- 4
CALL set_counter(@counter, 5); -- 9
SELECT @counter; -- 9
```

# FUNCTION

```
DROP FUNCTION IF EXISTS opinion;

DELIMITER //
CREATE FUNCTION opinion (rating INT)
RETURNS VARCHAR(10) DETERMINISTIC
BEGIN
    DECLARE op VARCHAR(10);
    IF (rating > 9)
        THEN SET op = "Great";
    ELSEIF rating > 5
        THEN SET op = "Average";
    ELSE
        SET op = "Sucks";
    END IF;
    RETURN op;
END//

DELIMITER ;
SELECT title, opinion(rating) FROM Movies;
```

- IF, ELSE IF, ELSE
- REPEAT UNTIL
- WHILE
- CASE

# LAST\_INSERT\_ID()

- Ger INT från senast auto\_increment primary key

```
INSERT INTO Genres (name) VALUES ("Splatter");
SELECT LAST_INSERT_ID();
```

LAST_INSERT_ID()
5

```
INSERT INTO Genres (name) VALUES ("Splatter");

INSERT INTO MoviesGenres (m_id, g_id) VALUES (6, LAST_INSERT_ID());
```

# Triggers

- Aktiva regler
- Aktiva databashanterare
- CREATE TRIGGER <trigger name>  
  { BEFORE | AFTER }  
  { INSERT | UPDATE | DELETE }  
ON <table name>  
FOR EACH ROW  
<triggered action>
- ECA
  - Event
  - Condition
  - Action
- Kontrollerar ett villkor och om det är uppfyllt så görs åtgärd:
  - Avbryta transaktion
  - Ändra innehåll
  - Anropa SP
  - Kommunicera med externa program
- Triggers aktiveras bara av SQL (inte ev kod som kör direkt mot DB)
- Går inte att ha Trigger på en vy
- Olika SQL har olika begränsningar för triggers (och olika versioner av MySQL).

# Trigger, exempel

```
DELIMITER //
CREATE TRIGGER yearfix BEFORE INSERT ON Movies
FOR EACH ROW
IF NEW.year < 1888 THEN SET NEW.year = 1888;
END IF;
```

# Triggers

```
DELIMITER //
CREATE TRIGGER yearfix BEFORE INSERT ON Movies
FOR EACH ROW
IF NEW.year < 1888 THEN SET NEW.year = 1888;
END IF;
```

- Triggers har tillgång till data från den transaktion som körs:
  - All ny data från INSERT
  - All ny data och gammal data i UPDATE
  - Borttagen data från DELETE
  - Utifrån det så kan avgöras vad som ska göras/köras eller inte.

# Triggers

- Sproc kan skicka värden till Trigger med OUT eller INOUT-parametrar.
- Triggers kan inte använda: START TRANSACTION, COMMIT, or ROLLBACK
- Triggers kan använda: ROLLBACK to SAVEPOINT

```
DROP PROCEDURE IF EXISTS validate_movie_year;
DELIMITER //
CREATE PROCEDURE validate_movie_year(
    IN year INT
)
DETERMINISTIC
NO SQL
BEGIN
    IF year < 1888 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No movies made before 1888';
    END IF;
END //
DELIMITER ;
DELIMITER //
CREATE TRIGGER validate_movie_insert
BEFORE INSERT ON Movies FOR EACH ROW
BEGIN
    CALL validate_movie_year(NEW.year);
END //
DELIMITER ;
DELIMITER //

CREATE TRIGGER validate_movie_update
BEFORE UPDATE ON Movies FOR EACH ROW
BEGIN
    CALL validate_movie_year(NEW.year);
END //
DELIMITER ;

INSERT INTO Movies (title, year) VALUES ("Test", 1745);
```

## Error

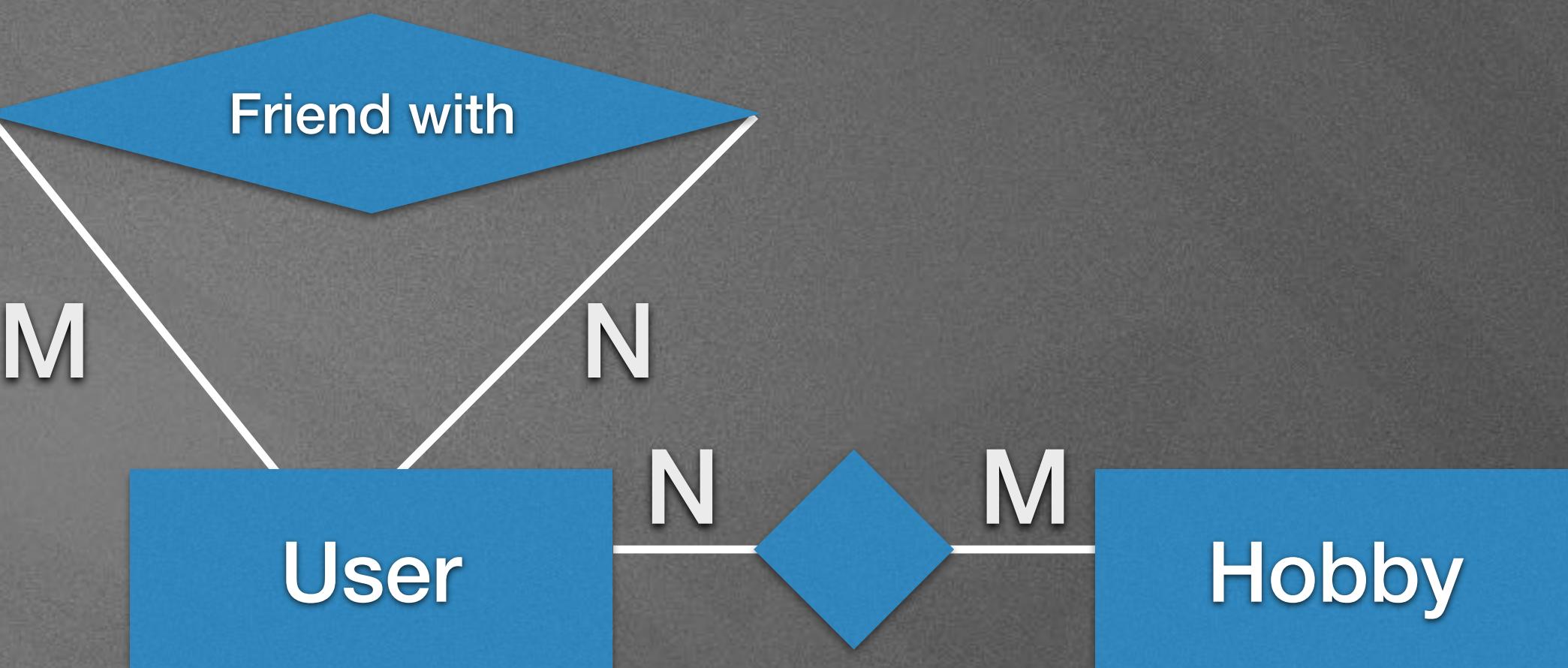
SQL query:

```
INSERT INTO Movies (title, year) VALUES ("Test", 1745)
```

MySQL said:

```
#1644 - No movies made before 1888
```

# Laboration 2



- Ett DB för användare och hobbies
- Användare är kompisar med andra användare och användare har hobbies.
- Finns en lista med vyer, stored procedures och triggers att lägga till för DB.

# Framåt

- Gör klart lab till och med lab 2.
- Läs kap 19 - Tid i databaser
- För planering nya videos och laborationer kommer måndagar och torsdagar
- Mer uppgifter med stored procedures och triggers på kommande laborationer. Ni kommer få mer träning i att använda dem.



UPPSALA  
UNIVERSITET

Tack och lycka till!