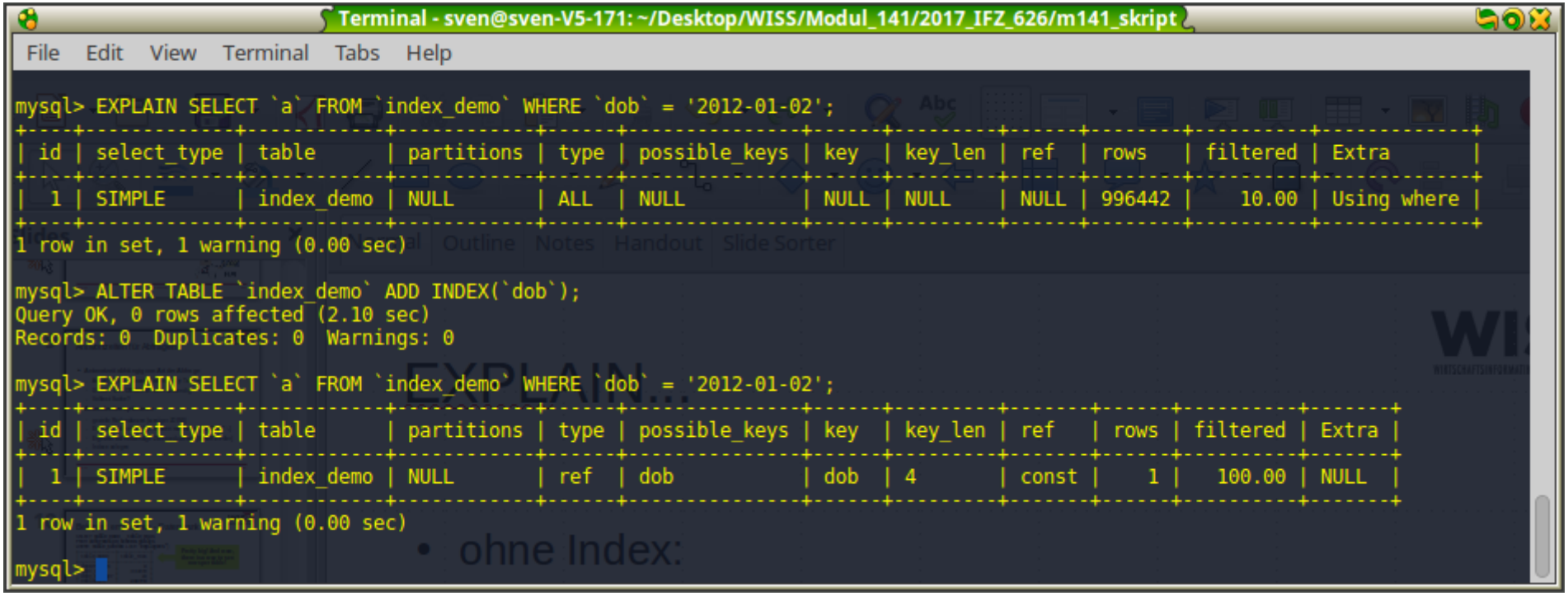
EXPLAIN...

- `EXPLAIN SELECT` ... zur Analyse der Abfrage
- achten auf:
  - type (ALL ist nicht schlau)
  - possible_keys (NULL ist mies)
  - key (NULL ist schlecht)



Warum

- ist die Anwendung so langsam?
- läuft der DB-Service permanent am Anschlag?

- Sind gewisse Abfragen zu langsam?

Tuning über Indexe

- schnelle Suchregister (alà Stichwortverzeichnis)
- vermeiden (langsames) Durchlaufen aller Datensätze
- **sinnvoll für häufig abgefragte Nichtschlüsselattribute**
- Primär- und Fremdschlüssel sind automatisch indiziert (wenn als PK bzw. FK angelegt)

Arten

- UNIQUE
- **INDEX**
- FULLTEXT

SQL-Syntax

- CREATE INDEX i_name ON Kunde (name);
- DROP INDEX i_name ON Kunde;
- SHOW INDEX FROM Kunde;

Stored Procedures (Abk. SP)

- In MySQL seit V5.0
- Ziel: Komplexe Programmierlogik in DB kapseln
  - schnell durch Vorkompilierung und weniger Netzwerkbelastung
  - Sicherheit + Wartbarkeit durch feste Schnittstellen
  - Geschäftsregeln besser umsetzbar (Mindestlänge Name etc.)
  - Direkte Schreibrechte auf Tabellen entfallen (auch für App-Developer)
- z.B. INSERT, UPDATE, DELETE Befehle
- SPs mit SQL Skripten angelegen!

Stored Procedures (SP)

```
DELIMITER //

CREATE PROCEDURE sp_ausleihen(IN p_kdnr INT,IN p_buch INT)
SQL SECURITY DEFINER
BEGIN

  INSERT INTO ausleihe (idmitglied, idbuch, ausleihedatum)
     VALUES (p_kdnr, p_buch, CURRENT_DATE());

END;

//
DELIMITER ;
```

```
mysql> SET @p_mitglied=33; SET @p_buch=4;
mysql> CALL sp_ausleihen(@p_mitglied,@p_buch);
Query OK, 1 row affected (0.02 sec)
```

SP – Syntax I: Signatur

- Parameterliste nach SP Name:
  - { IN | OUT | INOUT } Parameter mit definiertem SQL Datentyp
  - Sinnvolle Parameternamen benützen, z.B. p_idkunde
- **delimiter // → notwendig für SQL-Parser,**
  - **damit das Ende der Prozedur gefunden wird (nach END;)**
  - **um Statements wie üblich mit ';' beenden zu können**
  - **SP-Code abschliessen mit `//`**
  - **am Ende Delimiter zurücksetzen: `DELIMITER ;`**

Weitere Quellen:
https://www.w3resource.com/mysql/mysql-procedure.php

SP – Syntax II: body

- Zwischen BEGIN und END;
- Lokale Hilfsvariablen:
  - deklarieren: **DECLARE var_name *DATATYPE* [DEFAULT *value*];**
  - Zuweisungen: **SET var_name=(alpha*beta);**
- Variablen in SQL Statements lesen und schreiben:
  - **INSERT INTO tbl1 (col2, col3) VALUES (p_var1, pvar2);**
  - **SELECT COUNT(*) INTO var_name FROM tbl1;**
- Kontrollstrukturen in SPs:
  - **IF … THEN … ELSE … END IF;**
  - **CASE** u.v.m.

SP – Syntax III: Beispiel Schleifen

```
DELIMITER //
CREATE PROCEDURE `my_proc_LOOP` (IN num INT)
BEGIN
  DECLARE x INT;
  SET x = 0;
  loop_label: LOOP #Sprungmarke
    INSERT INTO number VALUES (rand());
    SET x = x + 1;
    IF x >= num  #Abbruchbedingung
       THEN LEAVE loop_label;
    END IF;
  END LOOP;
END;
//
DELIMITER ;
```

Benutzerdefinierte Funktionen

- Erweiterung von StoredProcedures
- Rückgabewert anstatt Übergabe als OUT Parameter in Parameter-Liste
- damit wird Anwendung direkt im SELECT möglich:
  - SELECT **PI**();
  - SELECT **NOW**();
  - SELECT **CONCAT**("Hallo Welt, ", " heute ist ",
    DATE_FORMAT(DATE(NOW()), '%d.%m.%Y'));

Benutzerdefinierte Funktionen

- Parameterliste enthält nur noch IN Parameter:

```
DELIMITER $$

CREATE FUNCTION mysumfun (x1 INT,x2 INT)
RETURNS INT DETERMINISTIC
BEGIN
  DECLARE sum INT DEFAULT 0;
  SET sum = x1 + x2;
  RETURN sum;
END $$

DELIMITER ;
```

- Aufruf via: `SELECT myssumfun(123,234);`

Funktion getPreis für Videothek-Projekt

```
DELIMITER $$

CREATE FUNCTION getPreis(idKunde INT, idMedium INT) RETURNS INT

BEGIN
    DECLARE vdays INT DEFAULT 0; DECLARE vprice DOUBLE DEFAULT 0;
    SELECT DATEDIFF(NOW(), Ausleihe) INTO vdays FROM Ausleihe
    WHERE Kundennummer=idKunde AND Mediumnummer=idMedium AND ??;

    SELECT PreisProTag INTO vprice FROM Film
    JOIN Medium ON Medium.Videonummer=Film.Videonummer
    WHERE Medium.Mediumnummer=idMedium;

    RETURN vprice*vdays;
END $$

DELIMITER ;
```

Trigger

- Zum Ausführen zusätzlicher SQL-Befehle **vor** oder **nach** **INSERT, UPDATE, DELETE** auf bestimmter Tabelle
- z.B. Medium-Ausleihezähler inkrementieren nach Rückgabe des Mediums

```
DELIMITER $

CREATE TRIGGER counter_increment BEFORE DELETE ON
Ausleihe

FOR EACH ROW

UPDATE Medium SET Medium.Zaehler = Medium.Zaehler+1
WHERE Medium.Mediumnummer = OLD.Mediumnummer;

$

DELIMITER ;
```

Trigger und SPs im DataDictionary

- ```
  SELECT routine_name, routine_schema
  FROM information_schema.routines
  WHERE routine_schema = 'm141_Videothek';
  ```
- ```
  SHOW TRIGGERS FROM m141_Videothek;

  SELECT trigger_name, trigger_schema
  FROM information_schema.triggers
  WHERE trigger_schema = 'm141_Videothek';
  ```