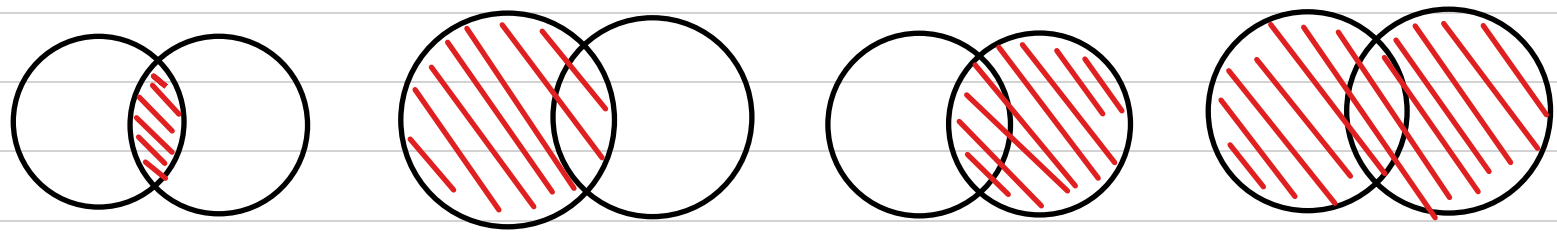
Week 8

- Inner join | Left join | Right join | Full join



- Multiple tables :
  - ```
    FROM actors, roles
    WHERE actors.id = roles.actor_id;
    ```
  - ```
    FROM actors AS a, roles AS r
    WHERE a.id = r.actor_id;
    ```
  - ```
    FROM actors AS a,
         movies AS m,
         roles AS r
    WHERE a.id = r.actor_id
    AND m.id = r.movie_id;
    ```

- Mending pakai JOIN (Innerjoin = Join) :
  - ```
    FROM actors AS a
    JOIN roles AS r ON (a.id = r.actor_id);
    ```
  - ```
    FROM actors AS a
    JOIN roles AS r
    ON (a.id = r.actor_id)
    JOIN movies AS m
    ON (m.id = r.movie_id);
    ```

ON VS USING

```
FROM actors_copy AS a
JOIN roles AS r USING (actor_id)
JOIN movies_copy AS m USING (movie_id);
```

  - ```
    FROM actors_copy AS a
    NATURAL JOIN roles AS r
    NATURAL JOIN movies_copy AS m;
    ```

Muncul 1 kalau sama

Natural join gabung 2 table berdasarkan attribut/kolom nama yang sama & datatype. Kalau inner join gabung 2 table dispecified di ON clause.

Muncul 2 / lebih

  - ```
    FROM actors AS a
    LEFT JOIN roles AS r
    ON (a.id = r.actor_id);
    ```

```
mysql> SELECT CONCAT(a.first_name, ' ', a.last_name) A
    ->        r.role AS Role
    -> FROM actors AS a
    -> LEFT JOIN roles AS r ON (a.id = r.actor_id);
+-------------------------+-----------------------------+
| Actor                   | Role                        |
+-------------------------+-----------------------------+
| Nicholas Cage           | Balthazar Blake             |
| Nicholas Cage           | Benjamin Franklin Gates     |
| Dwayne Johnson          | Jack Bruno                  |
| Dwayne Johnson          | Derek Thompson              |
| Dwayne Johnson          | Hank Parsons                |
| Diane Kruger            | Abigail Chase               |
| Jay Baruchel            | David Stutler               |
| Josh Hutcherson         | Peeta Mellark               |
| Josh Hutcherson         | Sean Anderson               |
| Teresa Palmer           | Rebecca Barnes              |
| Alexander Ludwig        | Cato                        |
| Alexander Ludwig        | Seth                        |
| Vanessa Hudgens         | Kailani                     |
| Ryan Sheckler           | Mick Donnelly               |
| Jennifer Lawrence       | Katniss Everdeen            |
| Ben Stiller             | Larry Daley                 |
| AnnaSophia Robb         | Sara                        |
| Dan Stevens             | Sir Lancelot                |
| Dylan O'Brien           | NULL                        |
| Thomas Brodie-Sangster  | NULL                        |
+-------------------------+-----------------------------+
20 rows in set (0.00 sec)
```

- Ada juga CROSS JOIN yang memakai konsep perkalian Cartesius/matriks
- FULL JOIN = LEFT + RIGHT JOIN
```

```
FROM actors AS a
JOIN roles AS r
ON (a.id = r.actor_id)
RIGHT JOIN movies AS m
ON (m.id = r.movie_id);
```

```
    -> FROM actors AS a
    -> JOIN roles AS r ON (a.id = r.actor_id)
    -> RIGHT JOIN movies AS m ON (m.id = r.movie_id);
+--------------------+-----------------------------------------+--------------------------+
| Actor              | Movie                                   | Role                     |
+--------------------+-----------------------------------------+--------------------------+
| Nicholas Cage      | The Sorcerer's Apprentice               | Balthazar Blake          |
| Jay Baruchel       | The Sorcerer's Apprentice               | David Stutler            |
| Teresa Palmer      | The Sorcerer's Apprentice               | Rebecca Barnes           |
| Josh Hutcherson    | The Hunger Games                        | Peeta Mellark            |
| Alexander Ludwig   | The Hunger Games                        | Cato                     |
| Jennifer Lawrence  | The Hunger Games                        | Katniss Everdeen         |
| Nicholas Cage      | National Treasure                       | Benjamin Franklin Gates  |
| Diane Kruger       | National Treasure                       | Abigail Chase            |
| Dwayne Johnson     | Race to Witch Mountain                  | Jack Bruno               |
| Alexander Ludwig   | Race to Witch Mountain                  | Seth                     |
| AnnaSophia Robb    | Race to Witch Mountain                  | Sara                     |
| Dwayne Johnson     | Tooth Fairy                             | Derek Thompson           |
| Ryan Sheckler      | Tooth Fairy                             | Mick Donnelly            |
| Dwayne Johnson     | Journey 2: The Mysterious Island        | Hank Parsons             |
| Josh Hutcherson    | Journey 2: The Mysterious Island        | Sean Anderson            |
| Vanessa Hudgens    | Journey 2: The Mysterious Island        | Kailani                  |
| Ben Stiller        | Night at the Museum: Secret of the Tomb | Larry Daley              |
| Dan Stevens        | Night at the Museum: Secret of the Tomb | Sir Lancelot             |
| NULL               | Project Almanac                         | NULL                     |
+--------------------+-----------------------------------------+--------------------------+
19 rows in set (0.00 sec)
```

Gabung baris dalam kolom yang sama memakai UNION/ALL

```
SELECT first_name, last_name,
       gender, birth_date
FROM actors
WHERE gender = 'F'
UNION /ALL
SELECT first_name, last_name,
       gender, birth_date
FROM actors
WHERE birth_date > '1992-01-01';
```

```
Administrator: C:\Windows\system32\cmd.exe - mysql -h localhost -u root

mysql> SELECT first_name, last_name, gender, birth_date
    -> FROM actors
    -> WHERE gender = 'F'
    -> UNION
    -> SELECT first_name, last_name, gender, birth_date
    -> FROM actors
    -> WHERE birth_date > '1992-01-01';
+------------+------------+--------+------------+
| first_name | last_name  | gender | birth_date |
+------------+------------+--------+------------+
| Diane      | Kruger     | F      | 1976-07-15 |
| Teresa     | Palmer     | F      | 1986-02-26 |
| Vanessa    | Hudgens    | F      | 1988-12-14 |
| Jennifer   | Lawrence   | F      | 1990-08-15 |
| AnnaSophia | Robb       | F      | 1993-12-08 |
| Josh       | Hutcherson | M      | 1992-10-12 |
| Alexander  | Ludwig     | M      | 1992-05-07 |
+------------+------------+--------+------------+
7 rows in set (0.00 sec)

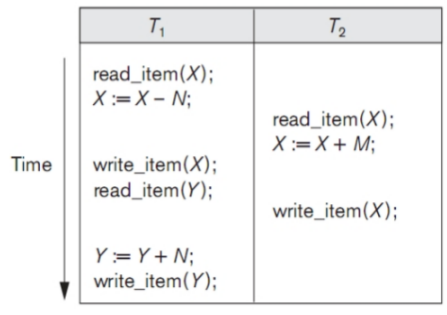
mysql> _
```

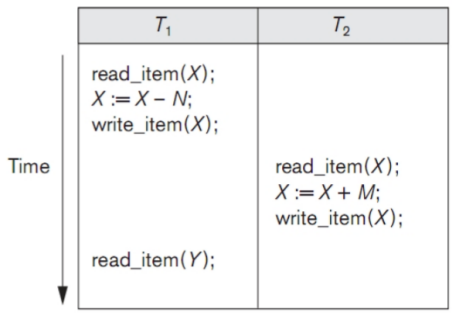
- UNION = Yang sama ambil 1 saja
  UNION ALL = Yang sama ambil semua yang ada

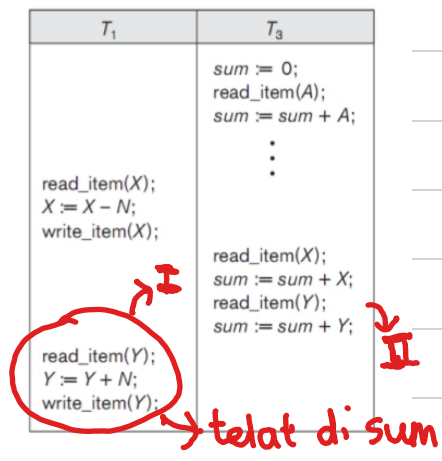
- LEFT JOIN = LEFT OUTER JOIN

- RIGHT JOIN = RIGHT OUTER JOIN

Week 9

- Multi-user kalau mau akses database itu secara concurrently
- Transactions = An executing program that form logical unit of database processing.
- Proses transaksi tentu memakan banyak akses database operations seperti insertion, deletion, modification, or retrieval operations.
- Jika operasi database tidak update dan hanya retriene di transaksi, maka dinamakan read-only transaction. Otherwise namanya read-write transaction
- Basic operation :
  - read_item (X) = Reads a database item X into a program variable
  - write_item (X) = Writes the value of program variable X into database X
- 

  → Needs concurrency control → T1 masalah karena X overwrite
- 

  → Needs concurrency control → Kalau T1 rusak maka harus kembali ke nilai awal tetapi T2 memegang nilai yang temporary, sehingga pembacaan selanjutnya akan salah
- 

  → Needs concurrency control → T_3 reads X after N subtracted and reads Y before N is added sehingga kalkulasi salah. Yang 1 update nilai, yang 1 lagi baca function sum sehingga ada perhitungan nilai sebelum/sesudah update

- Oleh karena itu, perlu recovery karena mungkin ada :

1. Computer failure (System crash)
   - ↳ Hardware, software, or network error sewaktu lagi transaction
   - ↳ Hardware crash are usually media failures seperti main memory failure

2. Transaction / System error
   - ↳ Integer overflow, division by 0, erroneous parameter values, logical programming error

3. Local errors / exception conditions detected by transactions
   - ↳ Data for transaction may not be found
   - ↳ Insufficient account balance in banking database

4. Concurrency control enforcement
   - ↳ Bisa abort transaksi untuk menyelesaikan deadlock dari berbagai transaksi.

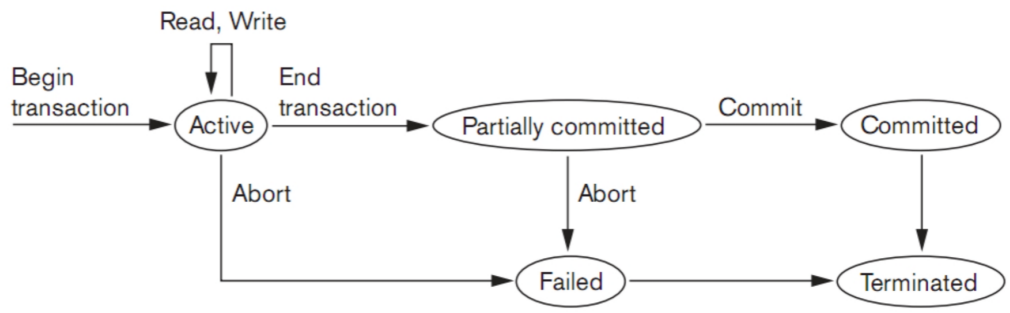
5. Disk failure
   - ↳ Bisa hilang data di disk karena read/write malfunction / head crash

6. Physical problems & catastrophes
   - ↳ Fire, theft, sabotage, salah pasang, etc.

- A transaction is an atomic unit of work that is either completed in its entirely or not done at all.

- Recovery manager keeps track this perintah :

1. BEGIN_TRANSACTION
2. READ OR WRITE
3. END_TRANSACTION
4. COMMIT_TRANSACTION
5. ROLLBACK / ABORT



- Properties of transactions : ACID
  - Atomicity : Transaction itu atomic unit of processing and perform semua / go.
  - Consistency preservation : If its complete execution takes database from one consistent state to another
  - Isolation : Should appear as though it is executed in isolation from other transaction and jgn interfered with by any other transactions executing concurrently.
  - Durability/permanency : Change pke commit hrs persist di database dan must not be lost because of any failure.

- Off autocommit = SET @@AUTOCOMMIT=0
  On autocommit = SET @@AUTOCOMMIT=1

- COMMIT = Make changes permanent
  ROLLBACK = Undo changes

- Undo only a part of current transaction by savepoint.
  - ↳ SAVEPOINT savepoint_name
  - ↳ ROLLBACK TO SAVEPOINT savepoint_name

Week 10

- Database threat :
  - Loss of integrity (Protect from improper modif seperti CRUD)
  - Loss of availability (Make object available yang ada legitimate right)
  - Loss of confidentiality (Protect data from unauthorized disclosure)

- 2 tipe database security mekanisme :

1. Discretionary Security Mechanisms
   ↳ Grant privileges to user untuk CRUD

2. Mandatory Security Mechanisms
   ↳ Multilevel security by classifying data seperti hny lihat data classified at user's own / lower classification level

- Control measures :

1. Access control
   ↳ Include provision for restrict access ke database by creating user account & password to control the login

2. Inference Control
   ↳ Statistical database used to provide statistical information seperti perusahaan besar untuk cari informasi populasi bkn individu.

3. Flow Control
   ↳ Prevent information from flowing yg dpt reach unauthorized users
   ↳ Covert channels are channels that pathways for information to flow implicitly in ways that violate the security policy of an organization

- Data Encryption
  - ↳ Protect data yg dikirim lewat communication network
  - ↳ Encoded using some algorithm

- Database Administrator / DBA is the central authority for managing a database system

- DBA has DBA account in DBMS, sometimes called a system or superuser account which provide powerful capabilities that are not made available to regular database accounts and users

- Types of action :
  - Account Creation
  - Privilege Granting
  - Privilege Revocation
  - Security Level Assignment

- If any tampering with database is suspected, a database audit is performed, which consists of review log to examine all akses dan operasi applied to database during certain time period.

- When an illegal or unauthorized operation is found, the DBA can determine account number used to perform operasi

- View = Single virtual table that is derived from other tables dan disimpan ke database

- ```
  CREATE USER 'username'@'hostname'
  IDENTIFIED BY 'password';
  ```

  Example

  ```
  CREATE USER 'dennis'@'localhost'
  IDENTIFIED BY 'dennis';
  ```

  Show the contents of the users catalog view

  ```
  SELECT host, user, password
  FROM mysql.user;
  ```

  ```
  DROP USER 'username'@'hostname';
  ```

  Example

  ```
  DROP USER 'dennis'@'localhost';
  ```

  ```
  RENAME USER 'username'@'hostname'
  TO 'new_username'@'new_hostname';
  ```

  Example

  ```
  RENAME USER 'dennis'@'localhost'
  TO 'dennisgunawan'@'localhost';
  ```
  ```

```
SET PASSWORD FOR 'username'@'hostname' = PASSWORD('password');
```

Example

```
SET PASSWORD FOR 'dennisgunawan'@'localhost'
= PASSWORD('dennisgunawan');
```

```
GRANT privileges | ALL PRIVILEGES
ON table_name | [database_name.] * | *.*
TO grantees;
```

```
GRANT SELECT
ON actors
TO 'dennis'@'localhost';
```

```
GRANT SELECT
ON entertainment.*
TO 'dennis'@'localhost';
```

```
GRANT CREATE USER
ON *.*
TO 'dennis'@'localhost';
```

```
GRANT INSERT, UPDATE
ON actors
TO 'dennis'@'localhost';
```

```
GRANT ALL PRIVILEGES
ON *
TO 'dennis'@'localhost';
```

```
GRANT CREATE, ALTER, DROP
ON *.*
TO 'dennis'@'localhost';
```

```
GRANT UPDATE (year, rank)
ON movies
TO 'dennis'@'localhost';
```

```
GRANT CREATE, ALTER, DROP, CREATE VIEW
ON entertainment.*
TO 'dennis'@'localhost';
```

```
GRANT SELECT, INSERT
ON actors
TO 'dennis'@'localhost'
WITH GRANT OPTION;
```

'dennis'@'localhost'

```
GRANT INSERT
ON actors
TO 'rico'@'localhost';
```

→ Agar bisa grant kasih orang

```
REVOKE privileges | ALL PRIVILEGES
ON table_name | [database_name.] * | *.*
FROM grantees;
```

REVOKE = Mencabut

```
REVOKE SELECT
ON actors
FROM 'dennis'@'localhost';
```

```
REVOKE GRANT OPTION
ON actors
FROM 'dennis'@'localhost';
```

```
CREATE [OR REPLACE] VIEW view_name [(column_list)]
AS SELECT_statement
[WITH CHECK OPTION];
```

Example

```
CREATE VIEW movie_production
AS SELECT name, year
   FROM movies;
```

```
SELECT name, year
FROM movie_production;
```

```
CREATE OR REPLACE VIEW good_movies
AS SELECT name, rank
   FROM movies
   WHERE rank > 6.5;
```

```
SELECT name, rank
FROM good_movies;
```

```
CREATE OR REPLACE VIEW good_movies
AS SELECT name, rank
   FROM movies
   WHERE rank > 6.5
WITH CHECK OPTION;
```

→ Untuk cek ada perubahan di update, insert, delete

```
DROP VIEW [IF EXISTS] view_name;
```

When IF EXISTS is specified, no error messages appe

Example

```
DROP VIEW movie_production;
```

```
DROP VIEW IF EXISTS good_movies;
```

Week 11

- Functional dependency = Constraint between 2 sets of attributes from the database

- X → Y :-
  - Y depends on X
  - Y is determined by X
  - Values of x determined by values of y
  - Y is functionally dependent on X

- Functional dependency = A property of semantics / meaning of the attributes

- Employee(EmployeeID, EmployeeName)
  - EmployeeID → EmployeeName

  Project(ProjectNumber, ProjectName, ProjectLocation)
  - ProjectNumber → { ProjectName, ProjectLocation }

- Normalisasi dibentuk oleh Codd (1972) yang berguna untuk :
  - Minimizing redundancy
  - Minimizing insertion, deletion, update anomalies

- **Normal Forms**
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain-Key Normal Form (DKNF)

- 1NF : Disallow multivalue, composit attribute

- Domain of attribute must include only atomic (Simple, indivisible) value.

Full Functional Dependency

A functional dependency $X \rightarrow Y$ is a full functional dependency if Y is functionally dependent on X but **not functionally dependent on any proper subset of X**

A functional dependency $X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more

- For any attribute $A \in X$, $(X - \{A\})$ does not functionally determine Y

Partial Dependency

- A functional dependency $X \rightarrow Y$ is a partial dependency if Y is functionally dependent on X and **also functionally dependent on any proper subset of X**
- A functional dependency $X \rightarrow Y$ is a partial dependency if some attribute $A \in X$ can be removed from X and the dependency still holds
  - For some $A \in X$, $(X - \{A\}) \rightarrow Y$

WorksOn(EmployeeID, ProjectNumber, Hours, ProjectName)

- EmployeeID, ProjectNumber → Hours, ProjectName
- ProjectNumber → ProjectName

EmployeeID, ProjectNumber → Hours is a full functional dependency

- Neither EmployeeID → Hours nor ProjectNumber → Hours holds

EmployeeID, ProjectNumber → ProjectName is a partial dependency

- ProjectNumber → ProjectName holds

2NF :
- Harus 1NF
- Hilangkan partial functional dependencies
- Make sure to keep relation with original primary key and any attributes that are fully functionally dependent on it.

A functional dependency $X \rightarrow y$ in a relation schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, and both $X \rightarrow Z$ and $Z \rightarrow y$ hold.

Works-For(SSN, D#, DPhone)

- SSN → D#, DPhone
- D# → DPhone

The dependency SSN → DPhone is transitive through D#

- Both the dependencies SSN → D# and D# → DPhone hold
- D# is neither a key itself nor a subset of the key of Works-For

SSN → D# is not a transitive dependency

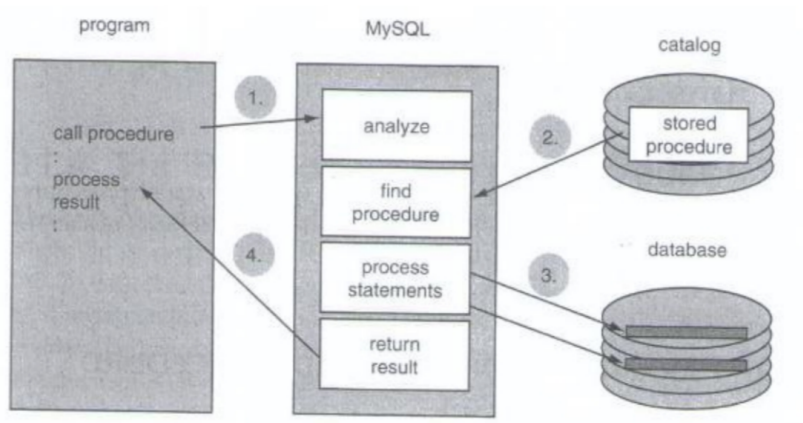
- 3NF :- Horus 2NF
  - Hilangkan transitive functional dependencies
  - Decompose & set up a relation that includes the nonkey attributes that functionally determines other nonkey attributes

- Higher NF eliminates more modification anomalies than lower NF

Week 12

- **Stored procedures** = Certain piece of code consist of declarative and procedural SQL statements stored in the catalog of a database that can activate by call program, trigger, or other stored procedure.

- Each stored procedure consists of at least 3 parts:
  - Name
  - List of parameters
  - Body

- 

  1. The process begins when the procedure is called from the program
  2. The database server receives this call and finds the matching procedure in the catalog
  3. The procedure is executed
  4. When the procedure is finished, the result of the procedure is returned

  - No communication takes place between the database server and the program during the execution of the procedure

- ```
  CREATE PROCEDURE procedure_name
     ( [ [ IN | OUT | INOUT ] parameter_name data_type,
         [ IN | OUT | INOUT ] parameter_name data_type,
         ...] )
  routine_body
  ```

  - `CREATE PROCEDURE delete_actor ( IN actor_id CHAR(5) )`
    `...`

  - `CREATE PROCEDURE show_all_movies ()`
    `...`
    - A stored procedure does not need parameters, but opening and closing brackets are still required

- 3 tipe parameter:
  - Input → Data pass to stored procedure
  - Output → Stove procedure use parameter when answer/result must be returned
  - Input/Output → Act as input & output
  ```

```
DECLARE variable_name, variable_name, ... data_type
   [ DEFAULT default_value ];
```

- `DECLARE score DECIMAL(5,2);`
- `DECLARE name VARCHAR(40);`
- `DECLARE number1, number2 INTEGER;`
- `DECLARE number1 INTEGER DEFAULT 100;`

```
SET variable_name { = | := } scalar_expression,
    variable_name { = | := } scalar_expression,
    ...;
```

- `SET var1 = 1;`
- `SET var1 := 1;`
- `SET var1 = 1, var2 = var1;`

```
CALL [ database_name. ] stored_procedure_name
   ( [ parameter, parameter, ... ] );
```

- `CALL delete_actor ('A0001');`
- `CALL show_all_movies ();`

Untuk fetch data 1 row memakai SELECT INTO, kalau multiple row pakai CURSOR

```
SELECT      ...
INTO        local_variable, local_variable, ...
FROM        ...
WHERE       ...
GROUP BY    ...
HAVING      ...
ORDER BY    ...
LIMIT       ...;
```

=

```
CREATE PROCEDURE count_actors ( OUT counter INTEGER )
BEGIN
   SELECT COUNT(*)
   INTO counter
   FROM actors;
END

CALL count_actors (@number_of_actors);

SELECT @number_of_actors;
```

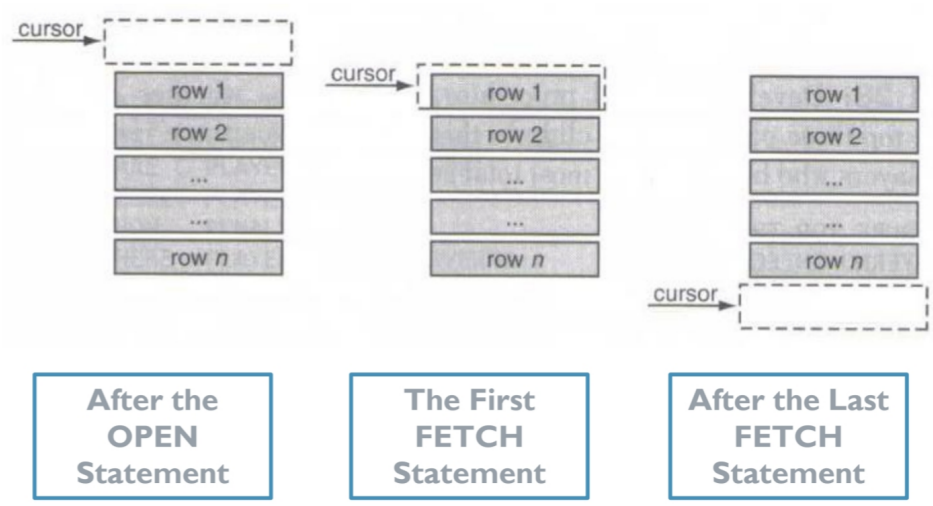
```
DROP PROCEDURE [IF EXISTS] [database_name.] procedure_name;
```

- `DROP PROCEDURE delete_actor;`
- `DROP PROCEDURE show_all_movies;`

Week 13

- 4 special SQL statements with cursor:
  1. DECLARE CURSOR
  2. OPEN CURSOR
  3. FETCH CURSOR
  4. CLOSE CURSOR

- ```
  FETCH cursor_name INTO local_variable, local_variable, ...;
  ```
  - We can use FETCH CURSOR statements to retrieve the created result row by row into the stored procedure
  - At a certain moment, only one row from the result is visible: the current row
  - It is as if an arrow points to one row from the result
  - With the FETCH CURSOR statement, we move this cursor to the next row

- 

- Advantage of stored procedure:
  - Jika specific set of update di database logically form a unit and if set itu use multiple application, better put di 1 procedure.
  - Minimize network traffic (Hrs selesai dulu proceduresnya)
  - Not depend on particular host language

- STORED FUNCTIONS VS STORED PROCEDURES

Have input but no output parameter	Merupakan output parameter
No call	Ada call
Ada return	No return
  ```

```
CREATE FUNCTION function_name
   ( [ parameter_name data_type,
       parameter_name data_type,
       ...] )
   RETURNS data_type
routine_body
```

```
GRANT EXECUTE
ON FUNCTION stored_function_name
TO grantees
[WITH GRANT OPTION];
```

```
DROP FUNCTION [IF EXISTS] [database_name.] function_name;
```

- `DROP FUNCTION count_actors;`

- Trigger = Piece of code consist of procedural & declarative statements stored in the catalog and activated by the database server if a specific operation is executed on database and only when certain condition holds

- Trigger turn passive database server into active

- Trigger ada 3 main elements :
  - Trigger moment
  - Trigger event
  - Trigger action

```
CREATE TRIGGER insert_actor
   AFTER INSERT ON actors FOR EACH ROW
BEGIN
   INSERT INTO actors_copy(id,first_name,last_name,gender,birth_date)
   VALUES(NEW.id,NEW.first_name,NEW.last_name,NEW.gender,NEW.birth_date);
END
```

- NEW is specified in front of the column names
- If a row is inserted, it looks as if there is a table called NEW
- The column names of this NEW table are equal to those of the triggering table (those in which the new row appears)
- As a result of specifying NEW in front of the column names, the actor that is added to the actors table is used

```
CREATE TRIGGER delete_actor
   BEFORE DELETE ON actors FOR EACH ROW
BEGIN
   INSERT INTO actors_history(id,first_name,last_name,gender,birth_date)
   VALUES(OLD.id,OLD.first_name,OLD.last_name,OLD.gender,OLD.birth_date);
END
```

- The keyword OLD is now specified instead of NEW
- After we remove a row, a table called OLD exists with column names that are equal to those of the triggering table, in which the removed row occurs
- When you update rows, the NEW and the OLD tables both exist
  - The row with the old values appears in the OLD table and the new row appears in the NEW table

- DROP TRIGGER trigger_name