Source: https://en.itpedia.nl/2017/11/26/wat-is-een-database/

DATABASES

Prof. Dr. Ulrike Herster
Hamburg University of Applied Sciences

HAW HAMBURG

COPYRIGHT

SUBQUERIES AND VIEWS
VIEWS – UPDATING VIEWS

Repetition

- Views are Relations
  → ... just like tables

- Should make no difference to users

- Question:
  Can we modify the view's data?
  → Depends on type of view!

SUBQUERIES AND VIEWS
VIEWS – UPDATING VIEWS

Repetition

- Classify views based on the select:
  - *Projection View*
    - `SELECT a, b, c …`
  - *Selection View*
    - `… WHERE < condition > …`
  - *Join View*
    - `… FROM tab_a JOIN tab_b …`
  - *Aggregation View*
    - `SELECT MAX(x) …`
- Other types and combinations exist

SUBQUERIES AND VIEWS
VIEWS – UPDATING VIEWS

Repetition

- A view with a single defining table is updatable if
  - the view attributes contain the primary key of the base relation,
  - as well as all attributes with the `NOT NULL` constraint that have a default value specified

- Views defined on multiple tables using joins are only updatable in special cases
  E.g., **`INSERT`** and **`UPDATE`** for Join Views, if join condition is based on PK-FK

- Views defined using grouping and aggregate functions are not updatable

Source: Elmasri, Fundamentals of Database Systems, Page 115ff

SUBQUERIES AND VIEWS
GENERATED TABLES

Repetition

- Syntax:
  `CREATE TABLE <name> AS SELECT …`
- Can create new table based on query
- New table is independent from old table
- Use cases:
  - Copy table
  - Copy parts of table
- Attention: New table does not have all constraints of the parent table!

SUBQUERIES AND VIEWS
GENERATED TABLES

Repetition

Example from before:

```
CREATE TABLE Underpaid ( lname , fname ) AS
       SELECT lname , fname
       FROM Employee
       WHERE salary < 1000 ;
```

ORGANIZATION
OUR JOURNEY IN THIS SEMESTER

Repetition

- □ Integrity, Trigger & Security
- □ Database Applications
- □ **Transactions**
- □ Subqueries & Views
- □ More SQL
- □ Notations & Guidelines
- □ Constraints
- □ Relationships
- □ Simple Entities and Attributes
- □ Basics

Source: Foto von Justin Kauffman auf Unsplash

TRANSACTIONS BASICS

Repetition

- A transaction bundles several operations into one logical unit
  - Unit of Work
  - Includes one or more database access operations
    E.g., **`INSERT`**, **`DELETE`**, **`UPDATE`**, **`SELECT`**
  - Operations must be executed all or none

- Example: Order a hotel room over the internet
  - Choose and reserve room
  - Payment
  - Final booking of the hotel room

TRANSACTIONS
ACID

Repetition

- Key features of transactions
  - *Atomicity*: Transaction is executed in whole or not at all
  - *Consistency*: State of the DB is consistent before and after a transaction
  - *Isolation*: Transactions do not interfere with other concurrent transactions
  - *Durability*: Changes are stored permanently in the database and will not get lost

TRANSACTIONS
ACID - ATOMICITY

Repetition

- Begin of Transaction (*BoT*)
  - SQL99: **`START TRANSACTION`**
  - mySQL: **`BEGIN`**
  - Oracle: transaction is started automatically

- Commit a transaction: **`COMMIT;`**
  - All operations are made persistent
  - All changes are visible to other users

- Rollback transaction: **`ROLLBACK;`**
  - DB is in state at *BoT* again

TRANSACTIONS
ACID - CONSISTENCY

Repetition

- □ DB: in consistent state before transaction
  → Also, in consistent state after transaction

- □ Integrity constraints assure that

- □ Constraints can be defined as
  - **`IMMEDIATE`** (default in mySQL)
    → are checked immediately after operation
  - **`DEFERRED`**
    → Check at time of commit

Not supported by mySQL!

TRANSACTIONS
ACID - ISOLATION

Repetition

- □ Transactions are isolated from other concurrent transactions
- □ Concurrent transactions shall behave well

TRANSACTIONS
ACID – ISOLATION: CONCURRENCY CONTROL

Repetition

- Concurrent operations can lead to problems
  - Lost Update
  - Dirty Read
  - Unrepeatable read
  - Phantom tuples

TRANSACTIONS
ACID – ISOLATION: CONCURRENCY CONTROL – ISOLATION LEVELS IN SQL

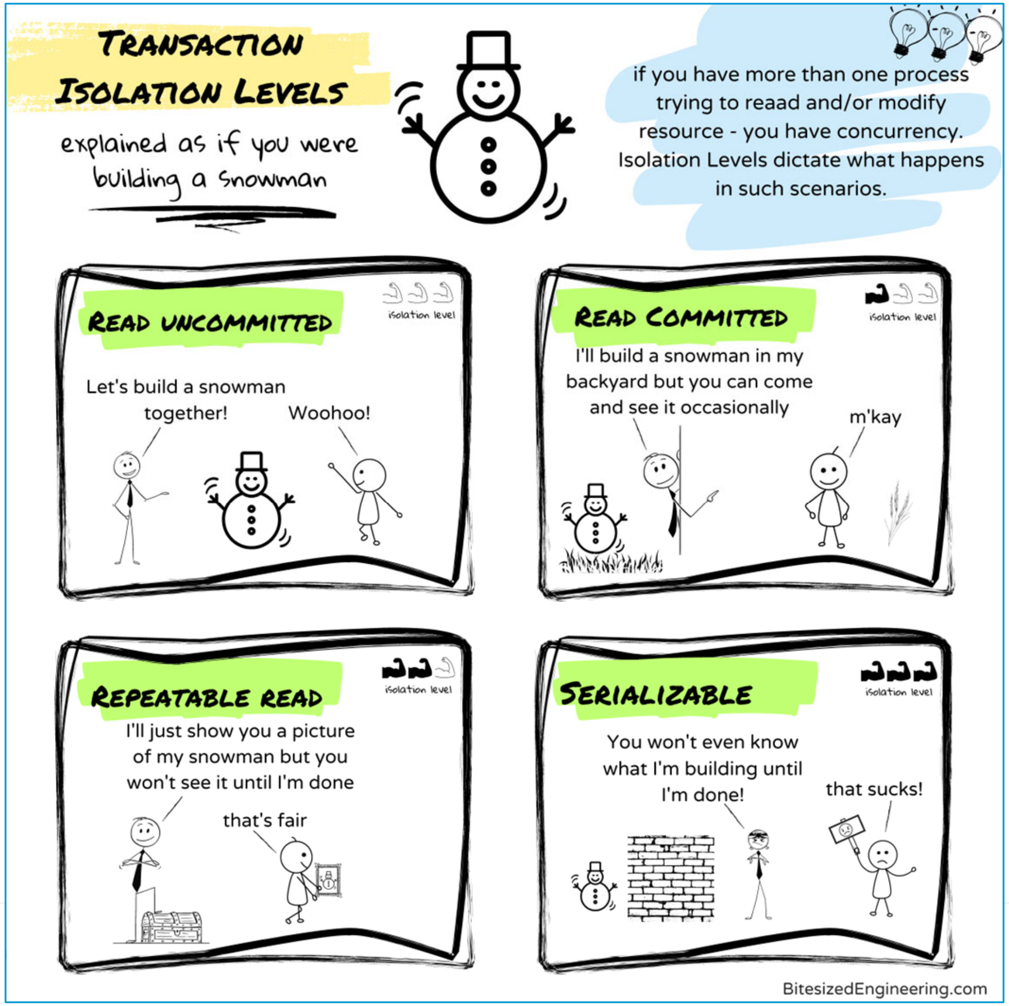
Repetition

- Lost Update is prevented by SQL

- Transactions: may choose *Isolation Level*
  - **`SERIALIZABLE`**
    → no problems
  - **`REPEATABLE READ`** (default in mySQL)
    → Open for phantom tuples
  - **`READ COMMITTED`** (default in Oracle, SQL Server)
    → Open for phantom tuples and unrepeatable read
  - **`READ UNCOMMITTED`**
    → Open for all problems

TRANSACTIONS
ACID – ISOLATION: CONCURRENCY CONTROL – ISOLATION LEVELS IN SQL

Repetition

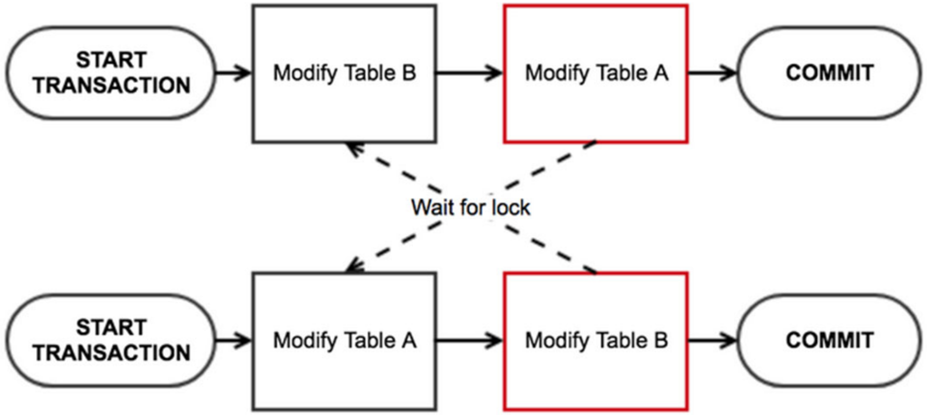

Transaction Isolation Levels
explained as if you were building a snowman
if you have more than one process trying to reaad and/or modify resource - you have concurrency. Isolation Levels dictate what happens in such scenarios.
Read uncommitted
isolation level
Let's build a snowman together!
Woohoo!
Read Committed
isolation level
I'll build a snowman in my backyard but you can come and see it occasionally
m'kay
Repeatable read
isolation level
I'll just show you a picture of my snowman but you won't see it until I'm done
that's fair
Serializable
isolation level
You won't even know what I'm building until I'm done!
that sucks!
BitesizedEngineering.com

Source: https://www.bitesizedengineering.com/p/database-isolation-levels-explained

TRANSACTIONS
ACID – ISOLATION: CONCURRENCY CONTROL – IMPLEMENTATION

Repetition

- Deadlocks may occur!!!
  - Especially when using isolation leven **SERIALIZABLE**
  - Usually are resolved automatically by aborting one transaction



Source: https://blog.nodeswat.com/concurrency-mysql-and-node-js-a-journey-of-discovery-31281e53572e

TRANSACTIONS
ACID - DURABILITY

- □ Once committed, changed data is safe

- □ Error types
  1. Computer failure
  2. Transaction or system error
     (constraint violation, $\frac{x}{0}$, blackout, system crash)
  3. Local Errors
  4. Concurrency control enforcement
  5. Disk error (harddisk broken)
  6. Physical problems and catastrophes
     (fire, earthquake, robbery, ...)

Source: Elmasri, Fundamentals of
Database Systems, Page 750ff

TRANSACTIONS
ACID – DURABILITY: ERROR HANDLING

- Recovery from transaction failures usually means that the database is *restored* to the most recent consistent state just before the time of failure

- Minor damages due to error types 1-4 from slide "ACID – Durability"
  - DBMS provides handling
  - Recovery strategy is to identify any changes that may cause an inconsistency in the database
    - Changes are first written to redo logs (files on disk)
    - Written to database files after commit

Source: Elmasri, Fundamentals of Database Systems, Page 808ff

TRANSACTIONS
ACID – DURABILITY: ERROR HANDLING

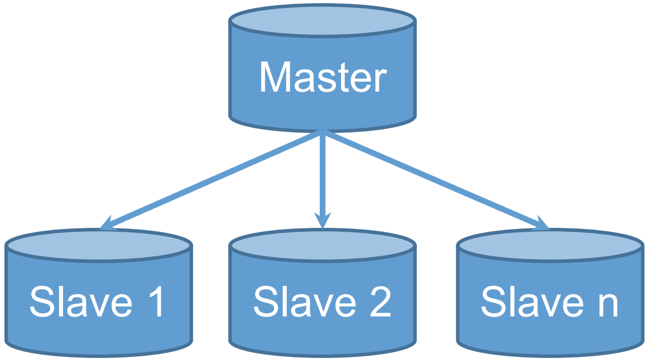
- Extensive damage due to error types 5-6 from slide "ACID – Durability"
  - recovery handling restores a past copy of the database from archival storage
  - reconstructs a more current state by redoing the operations
  - Last transactions are lost!

- Solution: Redundancy
  - RAID
    (**r**edundant **a**rray of **i**ndependent **d**isks)
  - Data Replication by DBMS

Source: Elmasri, Fundamentals of Database Systems, Page 808ff

TRANSACTIONS
ACID – DURABILITY: ERROR HANDLING – DATA REPLICATION

- Changes are performed on (replicated to) several database instances
  - Master/Slave
    - Updates only on one instance (master)
    - Slave: synchronous or asynchronous
  - Multi-Master
    - Updates on different instances
    - Needs conflict resolution strategy



TRANSACTIONS
ACID – DURABILITY: ERROR HANDLING – DATA REPLICATION

BONUS ★★★ BONUS ★★★ BONUS

- *Synchronous*
  - Transaction valid only when committed on all DBs
  - Safest, but performance impact
  - May reduce availability of the system

- *Asynchronous*
  - Transaction valid when committed locally

TRANSACTIONS
ACID – DURABILITY: ERROR HANDLING – DATA REPLICATION METHODS

BONUS

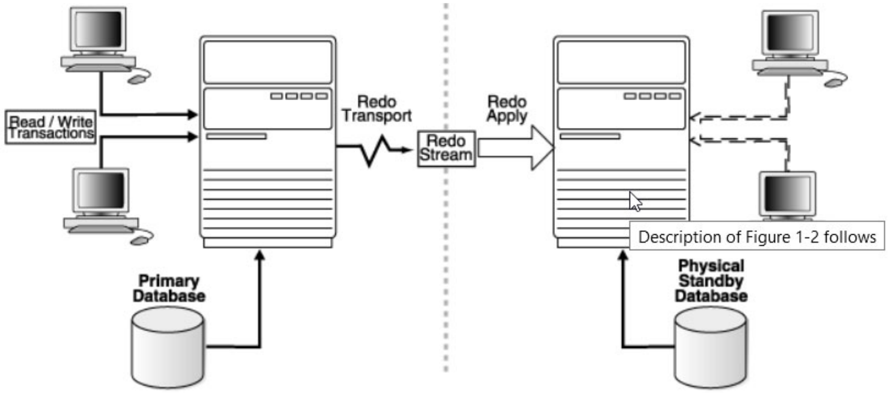
- Low level (disk device)

- Trigger based
  → Update triggers the replication (SQL level)

- Logfile shipping
  - Changes are stored in redo logs (as usual)
  - redo logs are copied to standby DB

TRANSACTIONS
ACID – DURABILITY: ERROR HANDLING – DATA REPLICATION METHODS

- Oracle
  - Data Guard
    → Replication on second server, can be used to answer Read-Only queries

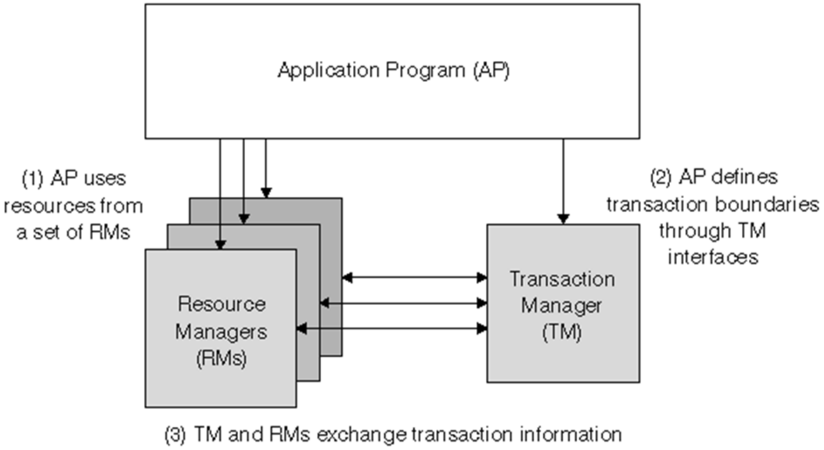

Read / Write Transactions
Redo Transport
Redo Stream
Redo Apply
Description of Figure 1-2 follows
Primary Database
Physical Standby Database

Source: https://docs.oracle.com/cd/B19306_01/server.102/b14239/concepts.htm#i1033808

  - Real Application Cluster (RAC)
    → Several servers share the same DB

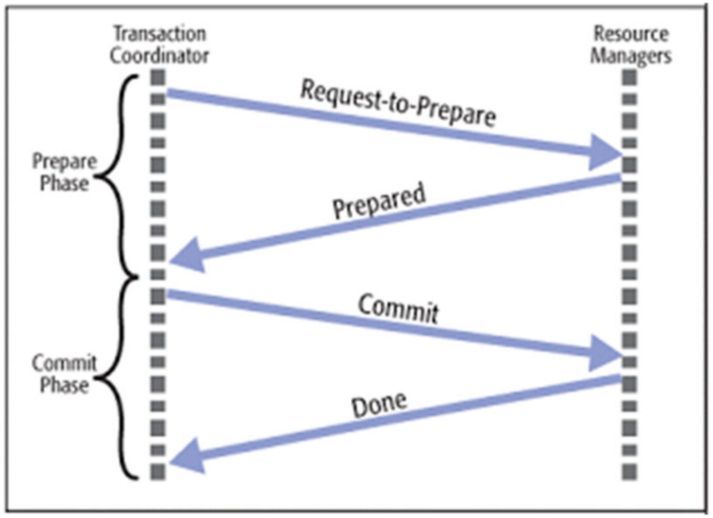
TRANSACTIONS
DISTRIBUTED TRANSACTIONS

- Transactions not only in a single DBS
- Standardized by X/Open
  - Transaction Manager:
    A software component that guarantees transaction properties
  - Resource Manager:
    Every resource
    (e.g., DBS, GUI) that is able
    to work in a transactional mode
    without providing a transaction control structure itself
- The Transaction manager coordinates the Resource Manager that take part in the transaction. E.g., different DBS (distributed transactions) that appear as one DBS from outside (transparency!)


Application Program (AP)
(1) AP uses resources from a set of RMs
(2) AP defines transaction boundaries through TM interfaces
Resource Managers (RMs)
Transaction Manager (TM)
(3) TM and RMs exchange transaction information

TRANSACTIONS
DISTRIBUTED TRANSACTIONS

- To ensure interoperability between the participating resource managers the *2-phase commit protocol* is realized



- It defines the final synchronization of different parts of a transaction of a global transaction
- In the first phase the transaction manager asks participating resource managers to announce the results of their local transaction part
- This leads to a global result (commit or rollback) that is then in the second phase announced to the participants

Source: https://medium.com/@balrajasubbiah/consensus-two-phase-and-three-phase-commits-4e35c1a435ac

TRANSACTIONS
SAVEPOINTS

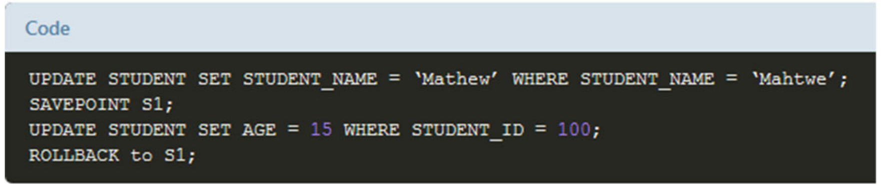
BONUS

- □ There are operations that may be expensive to execute
  → time consuming
- □ If certain constraints fail within transaction execution, then maybe these constraints may not fail in a second attempt
  (e.g., time dependent)
- □ So "fall back" points can be defined, which are called *savepoints*
- □ It is possible to rollback up to a savepoint and restart transaction execution from this point on

TRANSACTIONS
SAVEPOINTS

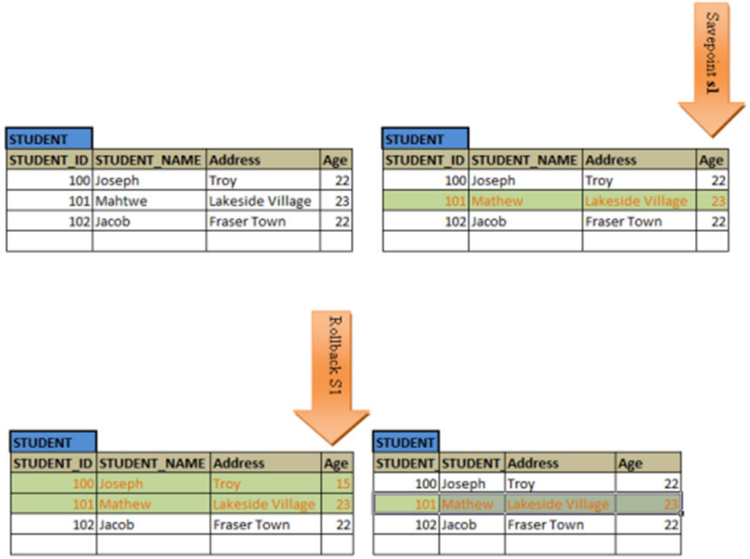
BONUS
BONUS
BONUS

Example:



Code

```
UPDATE STUDENT SET STUDENT_NAME = 'Mathew' WHERE STUDENT_NAME = 'Mahtwe';
SAVEPOINT S1;
UPDATE STUDENT SET AGE = 15 WHERE STUDENT_ID = 100;
ROLLBACK to S1;
```



STUDENT

STUDENT_ID	STUDENT_NAME	Address	Age
100	Joseph	Troy	22
101	Mahtwe	Lakeside Village	23
102	Jacob	Fraser Town	22

STUDENT

STUDENT_ID	STUDENT_NAME	Address	Age
100	Joseph	Troy	22
101	Mathew	Lakeside Village	23
102	Jacob	Fraser Town	22

STUDENT

STUDENT_ID	STUDENT_NAME	Address	Age
100	Joseph	Troy	15
101	Mathew	Lakeside Village	23
102	Jacob	Fraser Town	22

STUDENT

STUDENT_	STUDENT_	Address	Age
100	Joseph	Troy	22
101	Mathew	Lakeside Village	23
102	Jacob	Fraser Town	22

Source: https://www.tutorialcup.com/
dbms/transaction-control-language.htm

ORGANIZATION
OUR JOURNEY IN THIS SEMESTER

- □ **Integrity, Trigger & Security**
- □ Database Applications
- □ Transactions
- □ Subqueries & Views
- □ More SQL
- □ Notations & Guidelines
- □ Constraints
- □ Relationships
- □ Simple Entities and Attributes
- □ Basics

Source: Foto von Justin Kauffman auf Unsplash

INTEGRITY, TRIGGER & SECURITY
INTEGRITY CONSTRAINTS

- Static Constraints
  - Conditions on states
  - Conditions must be fulfilled before and after operations
  - Used until now
    - Primary Key
    - Foreign Key
    - `UNIQUE`, `NOT NULL`, `CHECK`

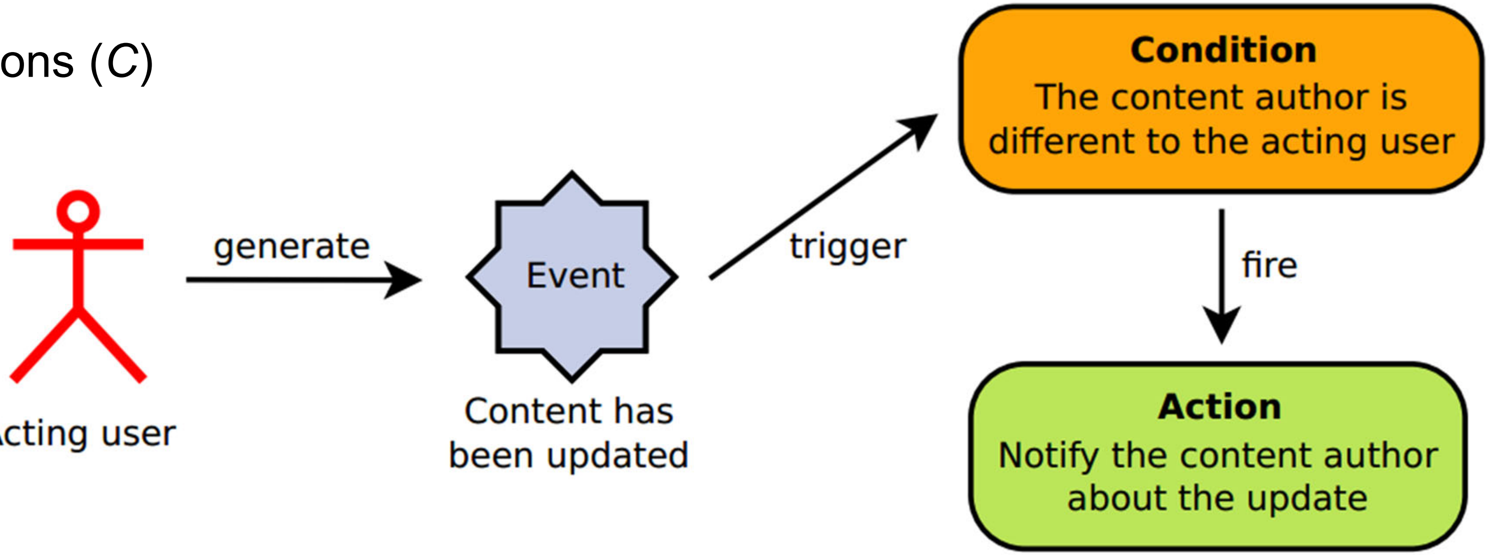
- Dynamic Constraints (*Assertions*)
  - Integrity conditions that affect multiple tables
  - Conditions on state transitions
    → Example: status of order
    new → payed → processing → shipped

INTEGRITY, TRIGGER & SECURITY
INTEGRITY CONSTRAINTS

- Assertions have been part of the SQL since SQL-92 (DDL)
- Not supported by most DBMS
  (e.g., MySQL, Postgres and Oracle)
- If the concept of assertions is to be simulated
  → **`TRIGGER`**
- Concept:
  - Whenever anything is modified in the database,
    the assertion checks its condition
  - If the **`SELECT`**-statement gives a non-empty result,
    the operation that has triggered the assertion is denied

INTEGRITY, TRIGGER & SECURITY
TRIGGER – ECA RULE

- *ECA* rules
  - on an event (*E*)
  - under certain conditions (*C*)
  - perform actions (*A*)



Acting user
generate
Event
Content has been updated
trigger
Condition
The content author is different to the acting user
fire
Action
Notify the content author about the update

Quelle: https://dev.acquia.com/blog/drupal-8-module-of-the-week/drupal-8-module-of-the-week-rules/15/06/2016/15681

INTEGRITY, TRIGGER & SECURITY
TRIGGER

Source: https://www.youtube.com/
watch?v=gpthfJnvzY8

INTEGRITY, TRIGGER & SECURITY
TRIGGER – SYNTAX IN MYSQL

```
CREATE
   [DEFINER = user]
   TRIGGER trigger_name
   trigger_time trigger_event
   ON tbl_name FOR EACH ROW
   [trigger_order]
   trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXCURSION: DELIMITER

- □ A MySQL client program such as MySQL Workbench or mysql program uses the delimiter (";") to separate statements and executes each statement separately
- □ However, a stored procedure consists of multiple statements separated by a semicolon (";")
- □ If you use a MySQL client program to define a stored procedure that contains semicolon characters, the MySQL client program will not treat the whole stored procedure as a single statement, but many statements.
- □ Therefore, you must redefine the delimiter temporarily so that you can pass the whole stored procedure to the server as a single statement.
- □ To redefine the default delimiter, you use the delimiter command

Source: https://www.mysqltutorial.org/
mysql-stored-procedure/mysql-delimiter/

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXCURSION: DELIMITER

□ Shortly: A delimiter is a separator between commands

□ For example:

```
delimiter |
…
|
delimiter ;
```

□ In the code block between "`delimiter`" and "`delimiter;`" the delimiter is changed to "`|`" (instead of "`;`")

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXAMPLE IN MYSQL

What is the goal of this trigger?

```
delimiter |
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
   BEGIN
      IF NEW.SALARY > (SELECT SALARY
                    FROM EMPLOYEE
                    WHERE SSN = NEW.SUPER_SSN )
      THEN SET NEW.Salary = (SELECT SALARY
                          FROM EMPLOYEE
                          WHERE SSN = NEW.SUPER_SSN )-1;
      END IF;
END;
|
delimiter ;
```

Source: Elmasri, Fundamentals of Database Systems

INTEGRITY, TRIGGER & SECURITY
TRIGGER – SYNTAX IN ORACLE

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER } triggering_event ON table_name
[FOR EACH ROW]
[FOLLOWS | PRECEDES another_trigger]
[ENABLE / DISABLE ]
[WHEN condition]
DECLARE
    declaration statements
BEGIN
    executable statements
EXCEPTION
    exception_handling statements
END;
```

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EVENTS

- Triggers can react on events
  - DML: `INSERT`, `UPDATE`, `DELETE`
    - Most common trigger types
  - DDL: `CREATE`, `ALTER`, `DROP`
  - DB: startup, shutdown, logon of a user

- No `COMMIT` triggers

INTEGRITY, TRIGGER & SECURITY
TRIGGER – TYPES

- Time of execution, relative to event
  - **`BEFORE`**
  - **`AFTER`**
  - **`INSTEAD OF`**
- Statement trigger
  - Once per statement
  - Even if no row is affected!
  - Default trigger type
- Row trigger
  - For every affected row
  - Syntax: **`FOR EACH ROW`**

`INSTEAD OF` for views not supported by mySQL!

INTEGRITY, TRIGGER & SECURITY
TRIGGER – ORDER OF TRIGGER EXECUTION

- Before Statement Trigger (once!)

- For every row affected:
  - Before row trigger
  - DML operation
  - Immediate integrity checks
  - After row trigger

- After Statement Trigger (once!)

INTEGRITY, TRIGGER & SECURITY
TRIGGER – TRANSITION VARIABLES

- Row triggers can access old and new tuples
  - MySQL
    - :old or old → `NULL` for **`INSERT`**s
    - :new or new → `NULL` for **`DELETE`**s
  - Oracle
    - NEW and OLD

- Before row triggers:
  - Can even modify new!

INTEGRITY, TRIGGER & SECURITY
TRIGGER – USE CASES

- Constraints on state transitions
- Audit
  → When was a record last modified?
- Integrity checks with error correction
  → Change `:new`
- Maintain redundant data
- Updateable views
  → `INSTEAD OF`

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXAMPLE IN MYSQL

Example: Audit insertion of new persons

```
DROP TRIGGER IF EXISTS emp_insert;

CREATE TRIGGER emp_insert
AFTER INSERT ON employee
FOR EACH ROW
   INSERT INTO EMPLOYEE_LOG (ESSN, INSERT_DATE)
   VALUES ( NEW.ssn , NOW() ) ;
```

Source: Elmasri, Fundamentals of Database Systems

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXAMPLE IN ORACLE

Example: Audit insertion of new persons

```
CREATE OR REPLACE TRIGGER emp_insert
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
   INSERT INTO EMPLOYEE_LOG (ESSN, INSERT_DATE)
   VALUES( :NEW.Name , current_timestamp ) ;
END ;
```

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXAMPLE IN MYSQL

Example: Salary of new persons

```
delimiter |
CREATE PROCEDURE output
   (in ssn char(9), in old_sal DECIMAL(10,2),
    in new_sal DECIMAL(10,2), in diff_sal DECIMAL(10,2))
BEGIN
   INSERT INTO EMPLOYEE_SALDIFF VALUES ( ssn , old_sal , new_sal, diff_sal);
END
|
delimiter ;
```

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXAMPLE IN MYSQL

Example: Salary of new persons

```
delimiter |
CREATE TRIGGER Print_salary_changes
BEFORE UPDATE ON EMPLOYEE
FOR EACH ROW
   BEGIN
      DECLARE sal_diff DECIMAL(10,2);
      IF (NEW.salary != OLD.salary)
      THEN
         BEGIN
         SET sal_diff = NEW.salary - OLD.salary ;
         CALL output(NEW.ssn, OLD.salary, NEW.salary, sal_diff);
         END;
      END IF;
   END;
|
delimiter ;
```

INTEGRITY, TRIGGER & SECURITY
TRIGGER – EXAMPLE IN ORACLE

Example: Salary of new persons

```
CREATE OR REPLACE TRIGGER Print_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON Emp_tab
FOR EACH ROW
WHEN (NEW.empno > 0)
DECLARE
     sal_diff number ;
BEGIN
     sal_diff := :NEW.sal - :OLD.sal ;
     dbms_output.put ('Old salary : ' || :old.sal ) ;
     dbms_output.put ('New salary : ' || :new.sal ) ;
     dbms_output.put_line ('Difference ' || sal_diff ) ;
END;
/
```

INTEGRITY, TRIGGER & SECURITY
TRIGGER – PROBLEMS

- Problems
  - Cascading triggers
    → Trigger actions cause other triggers to fire
  - Execution order
    → Result of high-level operation must be independent hereof!
  - "Mutating Tables"

INTEGRITY, TRIGGER & SECURITY
TRIGGER – PROBLEMS

- Problems
  - Hard to implement
    - Transaction save!
    - Multi-session save
  - Hard to debug
    - Update may lead to insert in another table
    - ... can cause for example constraint violation
    - Which statement failed?

INTEGRITY, TRIGGER & SECURITY
TRIGGER – ASSIGNMENT WEBSHOP

- Suppose the following relations in your database
- In the table `Price_History` we want to track on how the prices of the products of table `Product` develop over time. Table `Price_History` has four attributes:
  - The record ID `PHID`
  - The reference to table `Product` with the foreign key `PID`
  - The current price `Price`
  - The date `Change_Date`, where we store the date of the change

Table `Product`

PID	Price	Description
1	0.50	red apple
2	0.60	green apple
3	1.20	red pepper
4	1.10	green pepper
…	…	…

Table `Product_History`

PHID	PID (FK)	Price	Change_Date
1	1	0.50	02.06.2021
2	3	1.20	02.06.2021
3	2	0.60	03.06.2021
4	4	1.10	04.06.2021
…	…	…	…

INTEGRITY, TRIGGER & SECURITY
TRIGGER – ASSIGNMENT WEBSHOP

1. **INSERT** trigger:
   We want to get an **INSERT** with the current (start) price in table Price_History when we do an **INSERT** in the table Product. This is triggered when an **INSERT** on our table product is done (**AFTER**).

2. **DELETE** trigger:
   Furthermore, in case of a **DELETE**, all records of the deleted product in the table Price_History should be deleted as well.

3. **UPDATE** trigger:
   If a price of a product is changed, this change should also result in an entry in the table Price_History.

Table Product

PID	Price	Description
1	0.50	red apple
2	0.60	green apple
3	1.20	red pepper
4	1.10	green pepper
…	…	…

Table Product_History

PHID	PID (FK)	Price	Change_Date
1	1	0.50	02.06.2021
2	3	1.20	02.06.2021
3	2	0.60	03.06.2021
4	4	1.10	04.06.2021
…	…	…	…

INTEGRITY, TRIGGER & SECURITY
PERMISSIONS – BASICS

- DBMS are multi-user systems

- You need permissions to do anything with the DB:
  - login
  - **`CREATE`** `table`, **`DROP`** `table`, etc.
  - **`SELECT`**
  - **`INSERT`**, **`UPDATE`**, **`DELETE`**

- Permissions can be **`GRANT`**ed and **`REVOKE`**d

INTEGRITY, TRIGGER & SECURITY
PERMISSIONS – BASICS

Source: https://www.youtube.com/watch?v=QmRQ9OvBVZQ

INTEGRITY, TRIGGER & SECURITY
PERMISSIONS – GRANT AND REVOKE

- Permissions can be **GRANT**ed and **REVOKE**d

- Syntax:

```
GRANT <privilege_name> ON <object_name>
TO { <user_name> | PUBLIC | <role_name>} [ WITH GRANT OPTION ] ;
```

- Example: **GRANT**

```
GRANT SELECT ON tab_a TO user_a ;
GRANT UPDATE ON tab_b TO user_a ;
```

- Example: **REVOKE**

```
REVOKE SELECT ON tab_a FROM user_a ;
```

INTEGRITY, TRIGGER & SECURITY
PERMISSIONS - LEAST PRIVILEGE PRINCIPLE

- A user should have exactly the permissions necessary to do the work
  - … and not more!

- Important for web applications
  - anonymous end users
  - not trustworthy

- Limit the possible damage of attacks

INTEGRITY, TRIGGER & SECURITY
PERMISSIONS – ASSIGNMENT WEBSHOP

1. Create a user `student` which is allowed to query and insert the table `Product`.
2. Revoke the insert privilege from a user `student.`