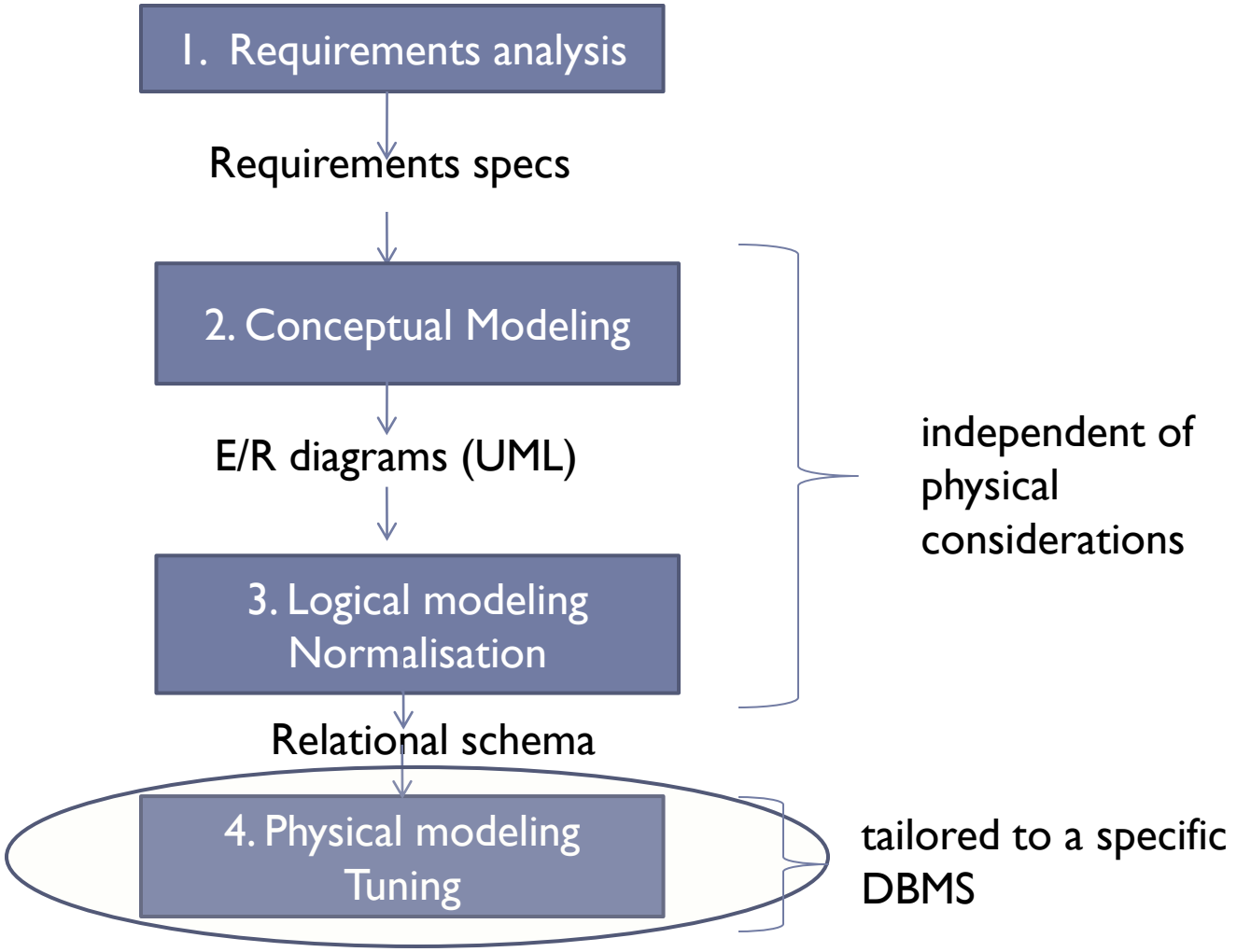
CREATE MATERIALIZED VIEW *lecture8* AS
SELECT *lecture*
FROM **Databases**

WHERE *topic* = ' Physical Design ';

INSERT INTO *lecture8* VALUES ('Constraints, Triggers, Views');

Mihaela Elena Breabăn

Relational Database Design Methodology



Outline

- Declaring integrity constraints
- Triggers
- Views

Static integrity constraints (1)

- Restrict the possible states of the database
  - Describe conditions that every instance of a database must satisfy
  - Avoid erroneous insertions, updates, deletions
  - Enforce data consistency
  - Tell the DBMS information that is useful for data storing and retrieval (query optimization)

- Types
  - Domain constraints
  - Non-null constraints
  - Keys – unique constraints
  - Referential integrity
  - General constraints at tuple level
  - Assertions

Static integrity constraints (2)

- Declaration
  - At (table) creation(CREATE TABLE)
  - After (table) creation (ALTER TABLE)
- Validation
  - At every DML statement
  - At the end of the transaction

Integrity constraints over 1 attribute
Inline declaration

CREATE TABLE *table_name* (

a1 type **not null**, -- does not allow null entries

a2 type **unique**, --candidate key consisting of one attribute

a3 type **primary key**, -- primary key consisting of one attribute, implies {not null, unique}

a4 type **references** *table_name2 (b1),* --foreign key consisting of one attribute

a5 type **check** (*condition*) – the condition is a Boolean expression built with attribute a5: (a5<11 and a5>4), (a5 between 5 and 10), (a5 in (5,6,7,8,9,10))…

)

Integrity constraints over several attributes *Out-of-line* declaration

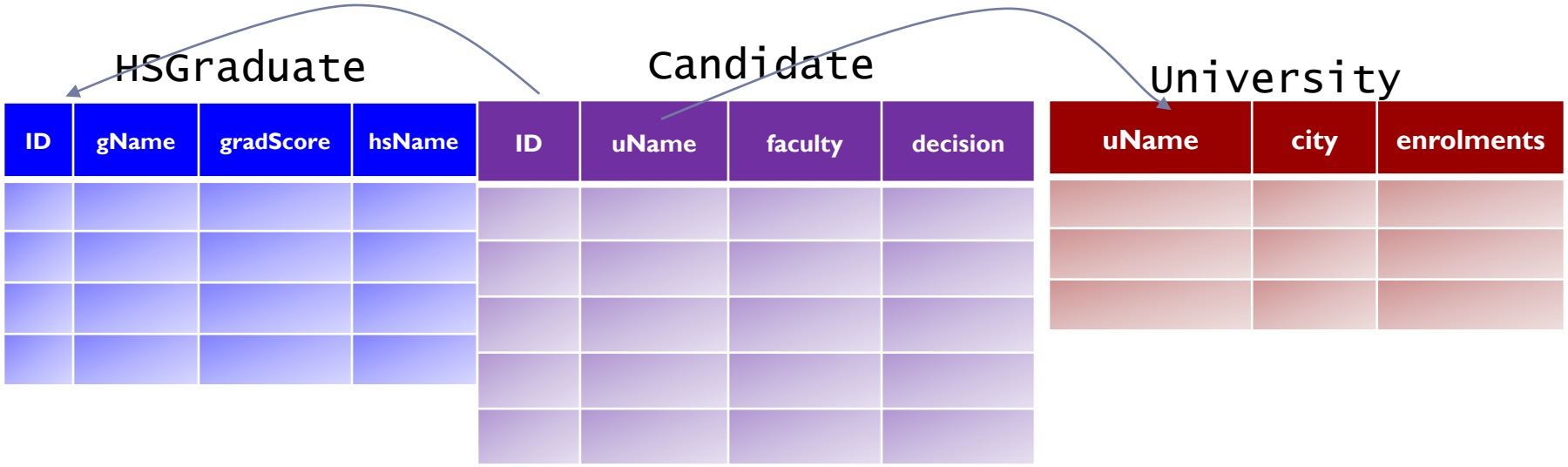
CREATE TABLE *table_name* (
a1 type,
a2 type,
a3 type,
a4 type,

primary key (*a1*,*a2*), -- primary key consisting of two (or more)attributes

unique(*a2*,*a3*), -- candidate key consisting of two (or more)attributes

check (*condiție*), -- a Boolean expression built over several attributes:
((a1+a3)/2>=5)
foreign key (*a3*,*a4*) **references** *table_name2*(*b1*,*b2*) – multi-valued foreign key
)

Referential integrity Definitions



- *Referential integrity from R.A to S.B:*
  - Every value in column A of table R must appear in column B of table S
  - A is called a foreign key
  - B must be declared as primary key or unique for table S
- There may exist multi-valued foreign keys

Referential integrity Validation

- Statements that may generate violations:
  - Insertions in R
  - Deletions in S
  - Updates of R.A or S.B

- Special actions that can be enforced:
  - At deletions in S:

ON DELETE RESTRICT (by default) | SET NULL | CASCADE

  - At updates on S.B:

ON UPDATE RESTRICT (by default) | SET NULL | CASCADE

Referential integrity
egg or chicken?

```
CREATE TABLE chicken (cID INT PRIMARY KEY,
                      eID INT REFERENCES egg(eID));
CREATE TABLE egg(eID INT PRIMARY KEY,
                      cID INT REFERENCES chicken(cID));
```

Referential integrity
egg or chicken?

```
CREATE TABLE chicken (cID INT PRIMARY KEY,
                      eID INT REFERENCES egg(eID));
CREATE TABLE egg(eID INT PRIMARY KEY,
                 cID INT REFERENCES chicken(cID));
```

```
CREATE TABLE chicken(cID INT PRIMARY KEY,  eID INT);
CREATE TABLE egg(eID INT PRIMARY KEY,   cID INT);
```

```
ALTER TABLE chicken ADD CONSTRAINT chickenREFegg
    FOREIGN KEY (eID) REFERENCES egg(eID)
     DEFERRABLE INITIALLY DEFERRED;  -- Oracle

ALTER TABLE egg ADD CONSTRAINT eggREFchicken
    FOREIGN KEY (cID) REFERENCES chicken(cID)
     DEFERRABLE INITIALLY DEFERRED; -- Oracle
```

```
INSERT INTO chicken VALUES(1, 2);
INSERT INTO egg VALUES(2, 1);
COMMIT;
```

How do you solve insertions if the constraints are validated at each statement?
What about table drops?

Asertions
-defined in the SQL standard -

```
create assertion Key
check ((select count(distinct A) from T) =
       (select count(*) from T)));
```

```
create assertion ReferentialIntegrity
check (not exists (select * from Candidate
                   where ID not in (select ID from HSGraduate)));
```

Integrity constraints
DBMS implementations

- Postgres, SQLite, Oracle, MySQL(innodb) implement and validate all constraints above
- No DBMS allows queries in the check constraint (deviation from the SQL standard)
- No DBMS implements assertions – their functionality can be provided by triggers

…DEMO…

(file *constraints.sql*)

Triggers
Dynamic constraints

- Monitor all the changes in a database, check conditions and initiate actions

- *Event-condition-action* rules
  - Bring within the DBMS elements from the application logic
  - Enforce constraints that cannot be expressed otherwise
  - Are expressive
  - May implement repairing actions

  - The implementation may differ among DBMSs, the examples in the presentation are in accordance with the SQL standard

Triggers Implementation

Create Trigger *name*
Before|**After**|**Instead Of** *event*
[*Referenced-variables*]
[For Each Row] -- the action is executed for each row altered by the event (row vs. statement types)
[When (*conditiion* **)]** – a boolean expression – exactly as within the WHERE clause in SQL queries
action -- în standardul SQL e o comandă SQL, în SGBD-uri poate fi bloc procedural

- *event*:
  - INSERT ON *table*
  - DELETE ON *table*
  - UPDATE [OF *a1,a2,…*] ON *table*

- *Referenced-variabiles* (they are declared and then used in *condition* and *action*):
  - OLD TABLE AS *var*
  - NEW TABLE AS *var*
  - OLD ROW AS *var* – *only for DELETE, UPDATE*
  - NEW ROW AS *var* –*only for INSERT, UPDATE*

Only for ROW triggers

Triggers Example

- Referential integrity from R.A to S.B implementing cascade deletes

```
Create Trigger Cascade_Deletes
After Delete On S
Referencing Old Row As O
For Each Row
[ no conditions ]
Delete From R Where A = O.B
```

```
Create Trigger Cascade_Deletes
After Delete On S
Referencing Old Table As OT
[ For Each Row ]
[ no condition ]
Delete From R Where
    A in (select B from OT)
```

Triggers
Problems to consider

- Several triggers are activated simultaneously: which one is the first to be executed?

- The trigger action activates other triggers: chaining that can create cycles

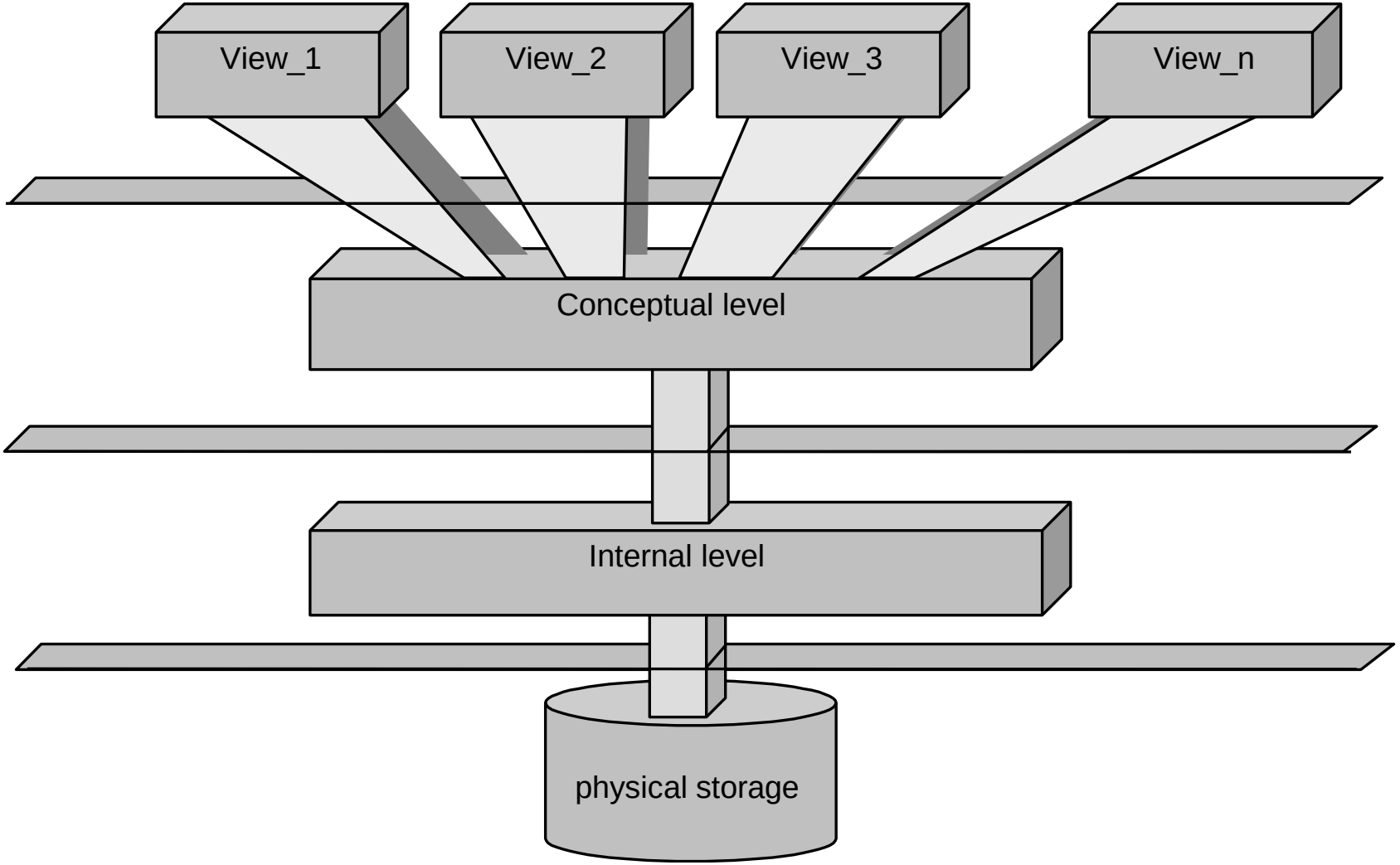
Triggers Implementation

- Postgres
  - The closest to the standard
  - implements row+statement -> {old,new}x{row,table} variables
  - The syntax suffers some changes
- SQLite
  - Only row (no old/new table)
  - Are executed after each row modification
- MySQL
  - Only row (no old/new table)
  - Are executed after each row modification
  - Allows only one trigger per event per table
- Oracle
  - Implements the standard: row+statement, some syntax changes
  - Instead-of triggers are allowed only for views
  - Allows the use of procedural blocks
  - Introduce restrictions to avoid cycling
- Studied at the lab in the second semester

…DEMO…

(file *triggers.sql*)

Views


View_1
View_2
View_3
View_n
Conceptual level
Internal level
physical storage

Motivation

- Controlled access to the data:
  - hide data from specific users
  - restrict DML statements
- Reduce the complexity of queries

- Real applications tend to use many views

Definition and use

- A view is actually a stored query formulated over tables or other views
- Its schema is generated based on the schema of the query result

- Conceptually, a view is queried just like a table
- In reality, a query over a view is rewritten by replacing the name of the view with the query defining the view; query optimization takes place, as implemented by the DBMS

- Syntax

Create View *view_name [(a1,a2,…)]* As <*select_statement*>

Modifying views

- Although the main operation/statement executed on a view is querying it, they may allow DML operations
- Modification commands on views must be rewritten as modification commands over base tables
  - Usually is simple
  - Sometimes several ways exist

- Example 1
  - R(A,B), V(A)=R[A];
  - INSERT INTO V VALUES(3);
- Example 2
  - R(N), V(A)=avg(N),
  - UPDATEV SET A=7;

Modifying views
Approaches

1. The view owner must rewrite all DML statements launched on the view as DML statements on the tables using the INSTEAD OF trigger
   - Covers all the cases
   - Guarantees data consistency

2. The SQL standard defines the existence of (inherently) **updatable views**:
   - The view must be created based on a single table T
   - The attributes in T that are not used in the view may have NULL entries or DEFAULT values
   - There is no aggregation used: no GROUP BY, no DISTINCT keyword

Materialized views

Create Materialized View *V [a1,a2,…]* As *<select_statement>*

- A new table V is created with schema defined by the select statement
- The tuples result of the query are inserted into V
- Any query on V is executed directly on the V table

- Advantages:
  - Specific to regular views + increased query execution speed
- Ddisadvantages:
  - V may increase in size
  - Any DMLs on the base tables require modifications on V
  - Any DMLs on V still must be translated into DMLs on base tables

How decide if materialize?

- Data size
- Query complexity
- The number of queries on the view
- The number of modifications on the base tables and the possibility of incrementally updating the view

- Trade-off the time for query processing over the time for propagating updates on the view when DML commands occur the base tables

…DEMO…

(file *views.sql*)

Bibliografie

- Hector Garcia-Molina, Jeff Ullman, **Jennifer Widom**: *Database Systems: The Complete Book (2nd edition)*, Prentice Hall; (June 15, 2008)

- *Oracle:*
  - *http://docs.oracle.com/cd/B28359_01/server.111/b28310/general005.htm*
  - *http://www.oracle-base.com/articles/9i/MutatingTableExceptions.php*
  - *http://www.dba-oracle.com/t_avoiding_mutating_table_error.htm*