These slides are for use with

Database Systems

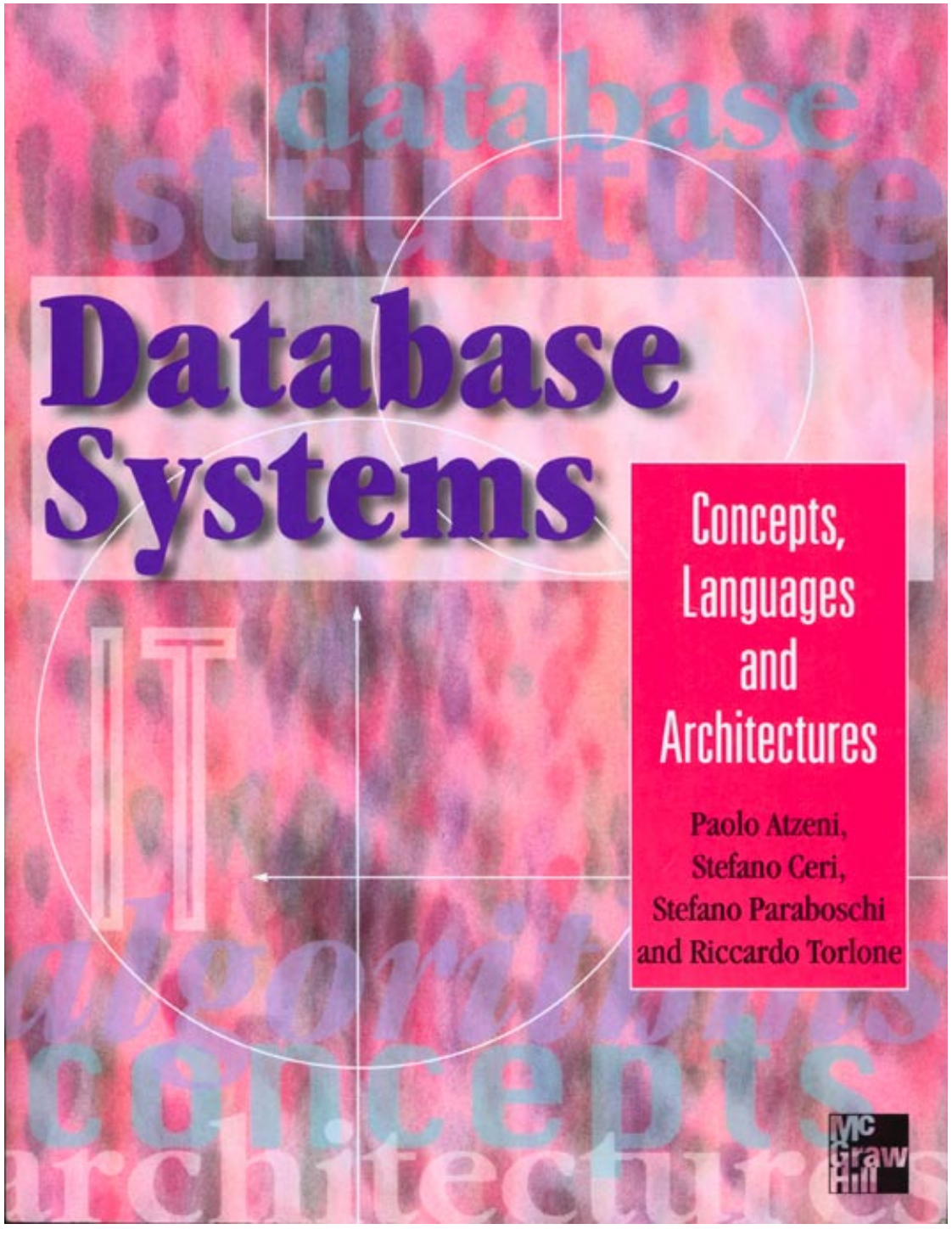
Concepts, Languages and Architectures

Paolo Atzeni • Stefano Ceri • Stefano Paraboschi • Riccardo Torlone

To view these slides on-screen or with a projector use the arrow keys to move to the next or previous slide. The return or enter key will also take you to the next slide. Note you can press the 'escape' key to reveal the menu bar and then use the standard Acrobat controls — including the magnifying glass to zoom in on details.

To print these slides on acetates for projection use the escape key to reveal the menu and choose 'print' from the 'file' menu. If the slides are too large for your printer then select 'shrink to fit' in the print dialogue box.

Press the 'return' or 'enter' key to continue . . .

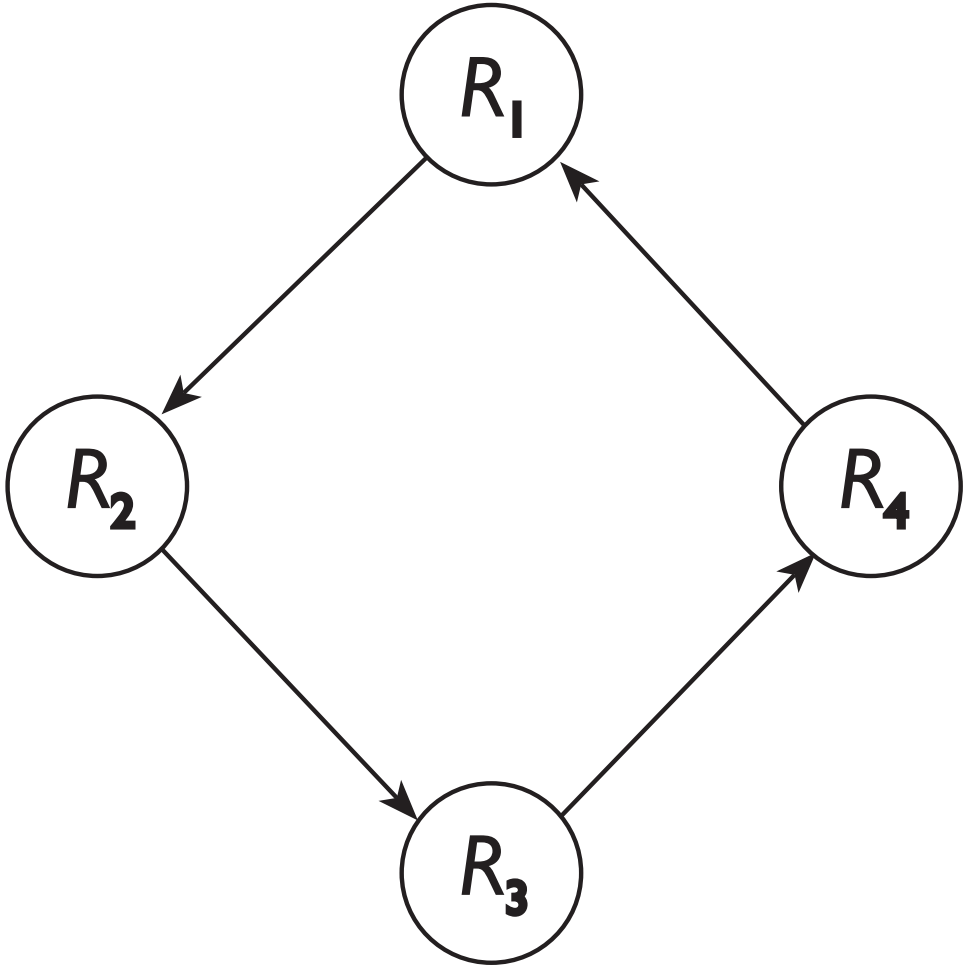


Chapter 12

Active databases

Click here for help

Cyclic activation graph



Active databases

- A DBMS which offers an execution engine for *Event-Condition-Action* rules (production rules) is an *active database*
- Active databases present a *reactive* behaviour (compared with the *passive* behaviour of typical DBMSs): they execute not only user transactions, but also the rules
- Rules are similar to procedures, but their execution is automatic: they permit to share a richer data semantics among all the applications accessing the database
  - "knowledge independence"
- Several research prototypes have been developed
- Most commercial systems offer this functionality, in the form of *triggers* (standardised in SQL-3)

Event-condition-action (ECA) paradigm

- Trigger are defined by a specific DDL statement (`create trigger`)
  - based on the event-condition-action paradigm:
    - event: data update, specified by `insert, delete, update`
    - condition (optional): SQL predicate
    - action: sequence of SQL statements (or a procedure in an extended SQL, like PL/SQL in Oracle)
  - intuition:
    - when the event occurs (triggering)
    - if the condition is satisfied (consideration)
    - then do the action (execution)
  - each trigger is associated with a table (target): it is activated only by events occurring on that table

Granularity and execution mode

- Granularity
  - row-level: the trigger is activated once for every tuple on which the event occurred
  - statement-level: the trigger is activated once for every SQL statement, referring to all the tuples on which the statement operated (set-oriented)
- Execution mode
  - immediate: right after (or even before) the event
  - deferred: at transaction commit

Triggers in Oracle, syntax

`create trigger` *TriggerName*
Mode Event {`,` *Event*}
`on` *TargetTable*
[[`referencing` *Reference*]
[`for each row`]
[`when` *SQLPredicate*]]
PL/SQLBlock

- *Mode:* `before` or `after`
- *Event:* `insert, update, delete`
- `for each row` specifies the granularity
- *Reference:* permits to introduce variable names (only for row-level triggers):

  `old as` *OldVariable* | `new as` *NewVariable*

Example: automatic reorder management

1. event:

   `update(QtyAvbl)` in Warehouse

2. condition: the new available quantity is less than the (new) reorder quantity:

   `new.QtyAvbl < new.QtyReord`

3. action:

   if there are no pending order for the part, issue a new order

Active rule

```
create trigger Reorder
after update of QtyAvbl on Warehouse
when (new.QtyAvbl < new.QtyLimit)
for each row
  declare
    X number;
  begin
    select count(*) into X
    from PendingOrders
    where Part = new.Part;
    if X = 0
    then
      insert into PendingOrders
      values (new.Part, new.QtyReord, sysdate);
    end if;
  end;
```

Application execution

```
update Warehouse
  set QtyAvbl = QtyAvbl - 50
  where Part = 4
```

Part	QtyAvbl	QtyLimit	QtyReord
4	100	80	50

Trigger execution

event : **`update(QtyAvbl) on Warehouse`**

condition : TRUE

action : **`insert into PendingOrders values (new.Part, new.QtyReord, sysdate)`**

Part	QtyReord	Date
4	50	10-10-00

Triggers in Oracle, semantics

- *Immediate* execution mode (both for `after` and `before`)
- Execution schedule:
  - `before statement` triggers
  - for each tuple on which the event occurred:
    - `before row` triggers
    - operation
    - `after row` triggers
  - `after statement` triggers
- If an error is detected, everything is undone
- No explicit user-defined priorities
- At most 32 triggers can be activated *in cascade*

Triggers in DB2, syntax

```
create trigger TriggerName
  Mode Event on TargetTable
  [referencing Reference]
  for each Level
   [when (SQLPredicate)]
    SQLProceduralStatement
```

- *Mode:* `before` or `after`
- *Event:* `insert, update, delete`
- `for each` *Level* specifies the granularity, `row` or `statement`
- *Reference:* permits to define variable names (depending on rule granularity):

```
        old as OldTupleVar | new as NewTupleVar
old_table as OldTableVar | new_table as NewTableVar
```

Triggers in DB2, semantics

- *Immediate* execution mode (both for `after` and `before`)
- `before` triggers cannot modify the database, except to introduce changes on the update introduced by the event (thus, they cannot trigger other rules)
- If an error occurs, everything is undone (executed transaction and triggers)
- No priorities among triggers (order defined by the system)
- Interaction with reactions to violations of referential integrity constraints
- At most 16 triggers *in cascade*

Example of trigger in DB2

```
foreign key (Supplier)
    references Distributor
    on delete set null

create trigger SoleSupplier
    before update of Supplier on Part
    referencing new as N
    for each row
    when (N.Supplier is not null)
    signal sqlstate '70005' ('Cannot change supplier')

create trigger AuditPart
    after update on Part
    referencing old_table as OT
    for each statement
    insert into Audit
    values(user, current_date, (select count(*) from OT))
```

Extensions (usually not available)

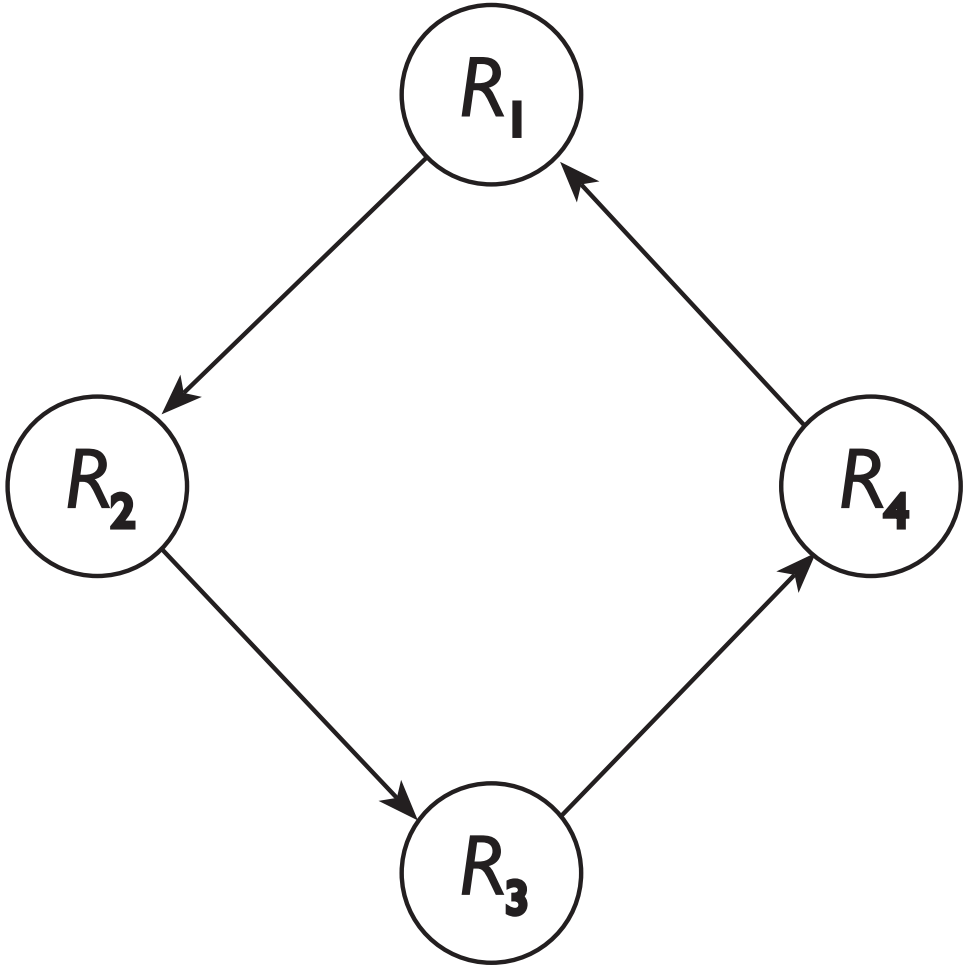
- Temporal events (even periodic) or “user-defined”
- Boolean combinations of events
- `instead of`: clause: it is not executed the operation that triggered the event, but another one in its place
- “Detached” execution mode: a separate transaction is started to manage the triggers
- Explicit user-defined priorities
- Rule sets, that can be made activated and deactivated with a single command

Properties of active rules

- Termination (fundamental property)
- Confluence
- Identical observable behaviour

- Termination is assessed studying rule interaction
- An important conceptual tool is the triggering graph
  - acyclicity of the triggering graph guarantees that the triggers cannot exhibit an infinite cascading execution
  - cycles in the triggering graph identify possible sources of non-termination

Cyclic activation graph



Applications of active databases

- Internal services
  - checking and maintenance of integrity constraints
  - replication
  - view management
    - materialised views: update propagation
    - virtual views: query rewriting
- External services
  - description of the dynamics (behaviour) of the database

Referential integrity management

- An exemplary active rule application

- The constraint is first expressed as an SQL predicate
  - the negation of the predicate becomes the rule condition
- The events that can introduce constraint violations are identified and monitored by active rules
- The rule actions represent the (arbitrary) integrity maintenance policy that the designer considers appropriate

Example of referential integrity management by active rules

- Integrity constraint

```
exists (select * from Department
        where DeptName = Employee.Dept)
```

- Condition:

```
not exists (select * from Department
            where DeptName = Employee.Dept)
```

- Events:
  - insert into Employee
  - delete from Department
  - update to Employee.Dept
  - update to Department.DeptName

Example of referential integrity management by active rules

- First active rule (of four):

```
create trigger DeptRef1
after insert on Employee
for each row
when (not exists
  (select * from Department
   where DeptName = New.Dept))
signal sqlstate '70006' (’employee without department');
```

Business rules

- Business rules express the strategy of a company in pursuing its objectives
- Active rules are the most flexible and powerful tool to implement business rules in information systems
- For instance, business rule "*an employee must not have a salary greater than that of the manager of the department to which he or she belongs*":

```
create trigger ExcessiveSalary
after update on Salary of Employee
for each row
when New.Salary > select Salary from Employee
                  where EmpNum in
                    (select Director from Department
                     where DeptNum = New.DeptNum)
then signal sqlstate '70005' ('Salary too high')
```