

# Databases

## More SQL: Domains, Constraints, Triggers, Views, Authorization

Hans Philippi

March 5, 2019

# SQL DDL: domains & constraints

- primary key constraints
- foreign key constraints
- attribute constraints
- tuple constraints
- domain definitions
- general constraints: assertions

# Attribute constraints

```
gender char(1)
    CONSTRAINT CheckGender
        CHECK (gender IN ('F','M','N'))

booknr integer
    CONSTRAINT CheckBookRef
        CHECK (booknr IN
            (SELECT booknr FROM Book))
```

# Tuple constraints

```
CREATE TABLE Person (  
    ...  
    CHECK (hasDrivingLicense = 'no' OR age >= 17)
```

# Domain definition

Auto increment key:

```
CREATE DOMAIN serial_number AS integer  
    CHECK (VALUE BETWEEN 1 AND 9999999)
```

```
CREATE TABLE Product (  
    id serial_number NOT NULL AUTO_INCREMENT=1000001  
    ...
```

# Domain definition

String mask definition:

```
CREATE DOMAIN postcode
  AS varchar(7)    NOT NULL
  CHECK (postcode LIKE
    ' [1-9] [0-9] [0-9] [0-9] [A-Z] [A-Z] '
  );
```

# Domain definition

String mask definition:

```
CREATE DOMAIN postcode
    AS varchar(7)    NOT NULL
    CHECK (postcode LIKE
        ' [1-9] [0-9] [0-9] [0-9] [A-Z] [A-Z] '
    );
```

```
CONSTRAINT postcode_invalid_substring
CHECK (
    (postcode NOT LIKE '%[SA]') AND
    (postcode NOT LIKE '%[SD]') AND
    (postcode NOT LIKE '%[SS]')
);
```

# Assertions

General constraints: *assertions*

```
CREATE ASSERTION BudgetCheckProject123 CHECK
  (100000 >=
    SELECT sum(Article.price)
    FROM Order, Article
    WHERE Order.artno = Article.artno
    AND Order.projectno = 123
  )
```

Possibly at high performance penalty



# Triggers

## Triggers or ECA rules

ON <event> IF <condition> THEN <action>

- *event*: insert, delete, update (possibly restricted to some attributes), transaction start, transaction end, temporal event, system event
- *condition*: evaluated on database (by query)
- *action*: database operation and/or general action

SQL3 triggers: we distinguish

- *row level triggers*

the action is repeated for each tuple satisfying the condition

*old* refers to the old value of the tuple

*new* refers to the new value of the tuple

- *statement level triggers*

the action is executed once

*old\_table* refers to the old value of the table

*new\_table* refers to the new value of the table

# Triggers

Example trigger (row level)

```
CREATE TRIGGER RegisterResultUpdates
AFTER UPDATE OF grade ON Results
REFERENCING
    OLD AS oldt
    NEW AS newt
WHEN (newt.grade <> oldt.grade)
    INSERT INTO UpGrades
    VALUES (oldt.studentno, oldt.course,
            oldt.date, oldt.grade, newt.grade)
FOR EACH ROW
```

# Triggers

Example trigger (statement level)

```
CREATE TRIGGER LogResults
AFTER INSERT ON Results
REFERENCING
    OLD_TABLE AS oldt
    NEW_TABLE AS newt
WHEN
    DECLARE @cnt1, @cnt2 integer;
    SELECT @cnt2 = count(*) FROM newt;
    SELECT @cnt1 = count(*) FROM oldt;
    INSERT INTO ResultsLog
    VALUES (@sysDate, @cnt2 - @cnt1)
FOR EACH STATEMENT
```

# Triggers: beware!

A trigger ...

# Triggers: beware!

A trigger ...  
may trigger ...

# Triggers: beware!

A trigger ...  
may trigger ...  
another trigger ...

# Triggers: beware!

A trigger ...  
may trigger ...  
another trigger ...  
which may trigger ...



# Triggers: beware!

A trigger ...  
may trigger ...  
another trigger ...  
which may trigger ...  
yet another trigger ...

# Triggers: beware!

A trigger ...  
may trigger ...  
another trigger ...  
which may trigger ...  
yet another trigger ...  
or even the first trigger again ...  
...  
...  
...  
(ad infinitum)

## SQL views

- define virtual relations on *base tables*
- are defined by a query
- define areas of interest for different users
- define areas of authorization for different users

## View definition

```
CREATE VIEW Late AS
SELECT abno, name, address, city, count(*) AS number
FROM Reader, Loan
WHERE Reader.abno = Loan.abno
AND loan_date < '01.01.2017'
GROUP BY abno, name, address, city
```

## The View Update Problem propagation of insert/update/delete on

- select view
- project view
- join view

## View updates: select view

```
CREATE VIEW InfStudent AS  
SELECT * FROM Student  
WHERE opleiding = "informatica"  
OR opleiding = "informatiekunde"
```

- delete
- update
- insert

## View updates: project view

```
CREATE VIEW MedewerkerPubl AS  
SELECT naam, adres, postcode, woonplaats, telefoon  
FROM Medewerker
```

```
-- attributen schaal, salaris weggelaten
```

- delete
- update
- insert

## View updates: join view

```
CREATE VIEW Uitleenoverzicht AS  
SELECT naam, datum, titel, auteur  
FROM Lezer, Lening, Boek  
WHERE Lezer.id = Lening.id  
AND Lening.boekid = Boek.boekid
```

- delete



# SQL Authorization: privileges

## Notions:

- user ID
- owner of data
- granting privileges to users

## Type of privileges:

- SELECT / SELECT(ATTR1,...,ATTRk)
- INSERT / INSERT(ATTR1,...,ATTRk)
- DELETE
- UPDATE / UPDATE(ATTR1,...,ATTRk)

# Granting privileges

Examples:

```
GRANT SELECT, INSERT, UPDATE  
ON StudentData TO annelies;
```

```
GRANT SELECT  
ON StudentData TO lennart  
WITH GRANT OPTION;
```

```
GRANT SELECT, INSERT, UPDATE  
ON StudentData TO jannie  
WITH GRANT OPTION;
```

# Revoking privileges

```
REVOKE <privileges>  
ON <data elements>  
FROM <users>  
[CASCADE | RESTRICT]
```

<privileges>:  
SELECT, ..., GRANT OPTION FOR <data>

# Revoking privileges

Examples:

```
REVOKE SELECT, INSERT, UPDATE  
ON StudentData TO arthur;
```

```
REVOKE GRANT OPTION  
FOR SELECT ON StudentData  
FROM jeroen CASCADE;
```

*Cascading effects!*

# Views and privileges

```
CREATE VIEW InfStudent AS  
SELECT * FROM Student  
WHERE opleiding = "informatica"  
OR opleiding = "informatiekunde";
```

```
GRANT SELECT ON InfStudent  
TO jeroen, lennart  
WITH GRANT OPTION;
```