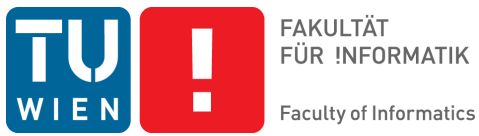
Data Base Systems

VU 184.686, WS 2020

Data Integrity

Anela Lolić

Institute of Logic and Computation, TU Wien



Acknowledgements

The slides are based on the slides (in German) of Sebastian Skritek.

The content is based on Chapter 5 of
(Kemper, Eickler: Datenbanksysteme – Eine Einführung)

and on Chapter 42.9. of the PostgreSQL online documentation
`https://www.postgresql.org/docs/current/static/plpgsql-trigger.html`

Data Integrity

1. Overview
2. Integrity Conditions
3. Referential Integrity
4. Trigger
5. Learning Objectives

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity

4. Trigger

5. Learning Objectives

Overview

1. Overview

2. Integrity Conditions
2.1 Static Integrity Conditions
2.2 Static Integrity Conditions in SQL

3. Referential Integrity

4. Trigger

5. Learning Objectives

Integrity Conditions

DMS are responsible for data consistency

Integrity Conditions

DMS are responsible for data consistency

consistency maintenance after system error (recovery) resp. concurrency control

Integrity Conditions

DMS are responsible for **data consistency**

consistency maintenance after system error (recovery) resp. concurrency control

here (chapter 5, 6): **semantical integrity conditions**, derived from the features in the modelled mini-world

Integrity Conditions

DMS are responsible for data consistency

consistency maintenance after system error (recovery) resp. concurrency control

here (chapter 5, 6): semantical integrity conditions, derived from the features in the modelled mini-world

Example

- matrNr has to be unique; - in AT: 8-ary,
- persNr is a 4-ary digit

Classification of Integrity Conditions

static integrity conditions: constraints on the state of the data base, that have to be satisfied in any state of the data base

Classification of Integrity Conditions

static integrity conditions: constraints on the state of the data base, that have to be satisfied in any state of the data base

Example

- professors have rank C2, C3, C4,
- matrNr of students is unique
- registration for the course only with valid enrolment

Classification of Integrity Conditions

static integrity conditions: constraints on the state of the data base, that have to be satisfied in any state of the data base

Example

- professors have rank C2, C3, C4,
- matrNr of students is unique
- registration for the course only with valid enrolment

dynamic integrity conditions: constraints on state changes in the data base

Classification of Integrity Conditions

static integrity conditions: constraints on the state of the data base, that have to be satisfied in any state of the data base

Example

- professors have rank C2, C3, C4,
- matrNr of students is unique
- registration for the course only with valid enrolment

dynamic integrity conditions: constraints on state changes in the data base

Example

- professors may only be promoted

Overview

1. Overview

2. Integrity Conditions
2.1 Static Integrity Conditions
2.2 Static Integrity Conditions in SQL

3. Referential Integrity

4. Trigger

5. Learning Objectives

Integrity Conditions

known implicit data integrity requirements

keys: unique within a relation

relationship cardinalities: cardinalities defined in the ER model are translated into the relational model such that only data obeying the cardinalities can be inserted

attribute domain: by fixing domains constraints to the data can be set

inclusion at generalization: entities of sub-types have to be contained in super-types as well

Overview

1. Overview

2. Integrity Conditions
2.1 Static Integrity Conditions
2.2 Static Integrity Conditions in SQL

3. Referential Integrity

4. Trigger

5. Learning Objectives

Static Integrity Conditions in SQL

- key candidates: `unique(attribute list)`
- primary key: `primary key(attribute list)`
- foreign key: `foreign key(attribute list)`
- null values not allowed: `not null`
- default value: `default`

Static Integrity Conditions in SQL

- key candidates: `unique(attribute list)`
- primary key: `primary key(attribute list)`
- foreign key: `foreign key(attribute list)`
- null values not allowed: `not null`
- default value: `default`

Example

```
create table student(
                matrNr    integer primary key,
                name      varchar(30) not null,
                semester  integer default 1);
```

`check`

check general static integrity conditions via `check`-clause, followed by a condition

evaluation of `check`-clause at modification/insertion in the table

modifications on a table are only permitted, if `check` does not evaluate to `false`

conditions might be complex; sub queries contained – attention: sometimes implemented only rudimentary!

SQL: Check (Example)

Example

```
create table student
       (matrNr    integer primary key,
        name      varchar(30) not null,
        semester  integer default 1
          check (semester between 1 and 13));
```

SQL: Check (Example)

Example

students can be examined in lectures that they have attended

```
create table examine
(matrNr integer,
 lecNr  integer,
 persNr integer,
 grade    numeric(2,1)
          check(grade between 0.7 and 5.0),
 primary key (matrNr, lecNr),
 constraint attended check
  (exists (select * from attend h
           where h.lecNr = examine.lecNr and
                 h.matrNr = examine.matrNr)));
```

Static Integrity Conditions in SQL

- column or table constraints
- can be part of the table definition (`CREATE TABLE`) or later added (`ALTER TABLE ADD ...`)
- can be named (else: name given by system)
- can be removed (`ALTER TABLE DROP ...`)

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity
3.1 Motivation and Definition
3.2 Guarantee of the Referential Integrity
3.3 Referential Integrity in SQL
3.4 Cyclic Dependencies

4. Trigger

5. Learning Objectives

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity
3.1 Motivation and Definition
3.2 Guarantee of the Referential Integrity
3.3 Referential Integrity in SQL
3.4 Cyclic Dependencies

4. Trigger

5. Learning Objectives

Referential Integrity

key identifies a tuple in a relation

foreign key points to tuple in a relation in relationship with

Referential Integrity

key identifies a tuple in a relation

foreign key points to tuple in a relation in relationship with

Example

`persNr` is key of professors, `givenBy` in lectures links to tuples in professors and is foreign key; it has to be ensured that there are no values in `givenBy` that do not occur in `persNr`

Referential Integrity

key identifies a tuple in a relation

foreign key points to tuple in a relation in relationship with

Example

`persNr` is key of professors, `givenBy` in lectures links to tuples in professors and is foreign key; it has to be ensured that there are no values in `givenBy` that do not occur in `persNr`

Definition (referential integrity)

foreign key either have to point to existing tuples in another relation or contain a null value

Referential Integrity

Definition (dangling references)

dangling references are references by foreign keys to not existing data sets

Referential Integrity

Definition (dangling references)

dangling references are references by foreign keys to not existing data sets

Example (dangling references)

insertion of the following data set into the table lectures:

```
insert into lectures
       values(5100, 'Nihilismus', 40, 0007);
```

⇒ what does '0007' reference?

deletion of the following data set in the table professors:

```
delete from professors where persNr=2125;
```

⇒ what does '2125' reference in the lecture table?

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity
3.1 Motivation and Definition
3.2 Guarantee of the Referential Integrity
3.3 Referential Integrity in SQL
3.4 Cyclic Dependencies

4. Trigger

5. Learning Objectives

Guarantee of the Referential Integrity

Theorem (referential integrity)

Let R be a relation with primary key κ and S a relation with foreign key α to R. The referential integrity is guaranteed, if

$$\pi_\alpha(S) \subseteq \pi_\kappa(R)$$

Guarantee of the Referential Integrity

Theorem (referential integrity)

Let R be a relation with primary key κ and S a relation with foreign key α to R. The referential integrity is guaranteed, if

$$\pi_\alpha(S) \subseteq \pi_\kappa(R)$$

allowed modifications are:

- insertion of s in S, if $s.\alpha \in \pi_\kappa(R)$

Guarantee of the Referential Integrity

Theorem (referential integrity)

Let R be a relation with primary key κ and S a relation with foreign key α to R. The referential integrity is guaranteed, if

$$\pi_\alpha(S) \subseteq \pi_\kappa(R)$$

allowed modifications are:

- insertion of s in S, if $s.\alpha \in \pi_\kappa(R)$
- modifying tuples of s to s' in S, if $s'.\alpha \in \pi_\kappa(R)$

Guarantee of the Referential Integrity

Theorem (referential integrity)

Let R be a relation with primary key κ and S a relation with foreign key α to R. The referential integrity is guaranteed, if

$$\pi_\alpha(S) \subseteq \pi_\kappa(R)$$

allowed modifications are:

- insertion of s in S, if $s.\alpha \in \pi_\kappa(R)$
- modifying tuples of s to s' in S, if $s'.\alpha \in \pi_\kappa(R)$
- modifying of $r.\kappa$ in R, if $\sigma_{\alpha=r.\kappa}(S) = \emptyset$, i.e. there is no reference of S to r

Guarantee of the Referential Integrity

Theorem (referential integrity)

Let R be a relation with primary key κ and S a relation with foreign key α to R. The referential integrity is guaranteed, if

$$\pi_\alpha(S) \subseteq \pi_\kappa(R)$$

allowed modifications are:

- insertion of s in S, if $s.\alpha \in \pi_\kappa(R)$
- modifying tuples of s to s' in S, if $s'.\alpha \in \pi_\kappa(R)$
- modifying of $r.\kappa$ in R, if $\sigma_{\alpha=r.\kappa}(S) = \emptyset$, i.e. there is no reference of S to r
- deletion of r in R, if $\sigma_{\alpha=r.\kappa}(S) = \emptyset$

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity
3.1 Motivation and Definition
3.2 Guarantee of the Referential Integrity
3.3 Referential Integrity in SQL
3.4 Cyclic Dependencies

4. Trigger

5. Learning Objectives

Referential Integrity in SQL

possible description for the three key notations as follows:

Referential Integrity in SQL

possible description for the three key notations as follows:

- (candidate)-key: `unique`
- primary key: `primary key(attributelist)`
- foreign key: `foreign key(attributelist)`

Referential Integrity in SQL

possible description for the three key notations as follows:

- (candidate)-key: `unique`
- primary key: `primary key(attributelist)`
- foreign key: `foreign key(attributelist)`

labelling of table to which we reference to:

- `references (table name)`

Referential Integrity in SQL

possible description for the three key notations as follows:

- (candidate)-key: `unique`
- primary key: `primary key(attributelist)`
- foreign key: `foreign key(attributelist)`

labelling of table to which we reference to:

- `references (table name)`

adherence of referential integrity at updates, deletions:

- `on update`
  `{no action | cascade | set null | set default}`
- `on delete`
  `{no action | cascade | set null | set default}`

Referential Integrity in SQL

possible description for the three key notations as follows:

- (candidate)-key: `unique`
- primary key: `primary key(attributelist)`
- foreign key: `foreign key(attributelist)`

labelling of table to which we reference to:

- `references (table name)`

adherence of referential integrity at updates, deletions:

- `on update`
  `{no action | cascade | set null | set default}`
- `on delete`
  `{no action | cascade | set null | set default}`

attention at cyclic dependencies

Referential Integrity in SQL

Example

```
create table professor
(persNr    integer primary key,
 Name      varchar(30) not null,
 ...                   );
```

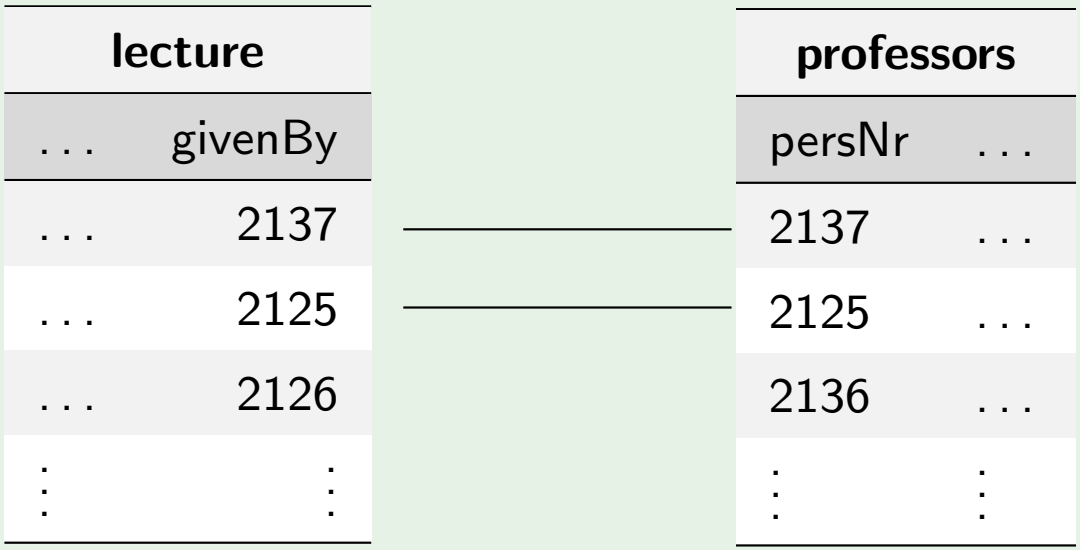
Referential Integrity in SQL

Example

```
create table professor
(persNr    integer primary key,
 Name      varchar(30) not null,
 ...                   );

create table lectures
(lecNr      integer primary key,
 title       varchar(30),
 SWS         integer,
 givenBy integer [foreign key]
             references professors
             on update cascade
             on delete set null);
```

Referential Integrity in SQL

Example (continuation)

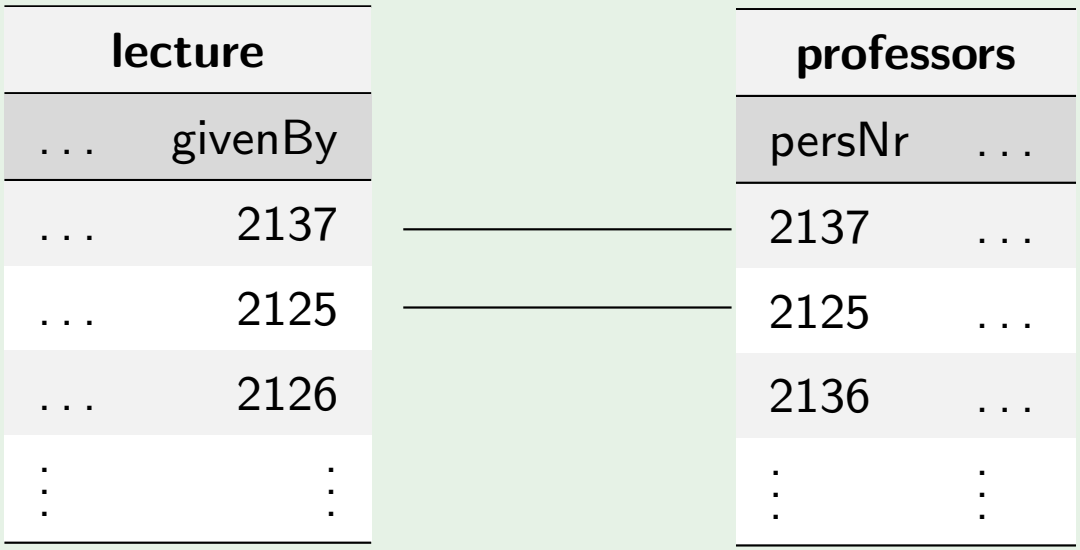


lecture	
...	givenBy
...	2137
...	2125
...	2126
⋮	⋮

professors	
persNr	...
2137	...
2125	...
2136	...
⋮	⋮

Referential Integrity in SQL

Example (continuation)



desired modification:

```
update professors set    persNr=1111
                    where persNr=2137;
delete from professors where persNr=2125;
```

Referential Integrity in SQL

Example (Continuation)

```
update professors set    persNr=1111
                      where persNr=2137;
```

adherence of referential integrity through cascading
(on update cascade)

Referential Integrity in SQL

Example (Continuation)

```
update professors set    persNr=1111
                      where persNr=2137;
```

adherence of referential integrity through cascading
(`on update cascade`)

lecture	
...	givenBy
...	1111
...	2125
...	2126
⋮	⋮

professors	
persNr	...
1111	...
2125	...
2136	...
⋮	⋮

Referential Integrity in SQL

Example (continuation)

```
delete from professors where persNr=2125;
```

Referential Integrity in SQL

Example (continuation)

```
delete from professors where persNr=2125;
```

adherence of referential integrity through "set to NULL"
(`on delete set NULL`)

lecture	
...	givenBy
...	1111
...	NULL
...	2126
⋮	⋮

professors	
persNr	...
1111	...
2136	...
⋮	⋮

Cascading Delete

attention when using `on delete cascade` because cascading delete might occur

Cascading Delete

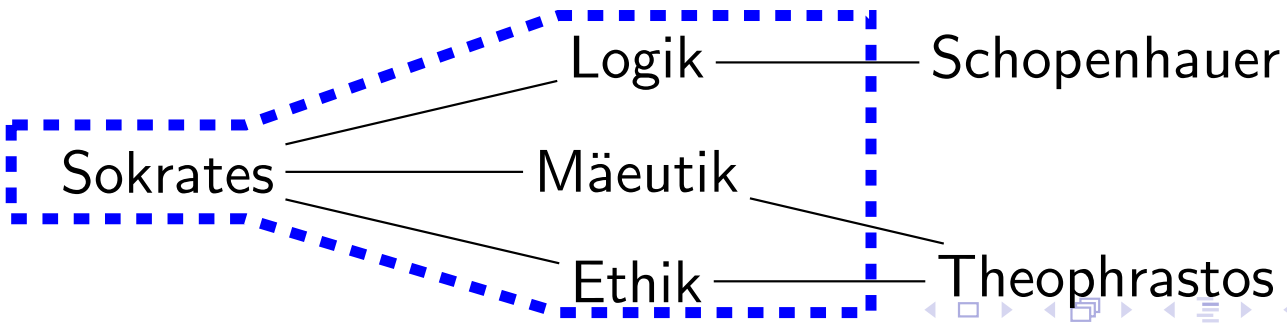
attention when using `on delete cascade` because cascading delete might occur

```
create table lectures
(...
 givenBy integer references professors
              on delete cascade);

create table attend
(...
 lecNr integer references lectures
         on delete cascade);
```

Cascading Delete

attention when using `on delete cascade` because cascading delete might occur

```
create table lectures
(...
 givenBy integer references professors
              on delete cascade);

create table attend
(...
 lecNr integer references lectures
         on delete cascade);
```



University Schema with Integrity Conditions (1/4)

```
create table student
(matrNr    integer primary key,
 Name      varchar(30) not null,
 Semester  integer
           check (Semester between 1 and 13));
```

University Schema with Integrity Conditions (1/4)

```
create table student
(matrNr    integer primary key,
 Name      varchar(30) not null,
 Semester  integer
           check (Semester between 1 and 13));

create table professors
(persNr  integer primary key,
 Name    varchar(30) not null,
 Rang    char(2) check(
                  Rang in ('C2','C3','C4')),
 room    integer unique);
```

University Schema with Integrity Conditions (2/4)

```
create table assistant
(persNr         integer primary key,
 Name           varchar(30) not null,
 field    varchar(30),
 Boss           integer references professors
                on delete set null);
```

University Schema with Integrity Conditions (2/4)

```
create table assistant
(persNr        integer primary key,
 Name          varchar(30) not null,
 field    varchar(30),
 Boss          integer references professors
               on delete set null);

create table lectures
(lecNr integer primary key,
 title  varchar(30),
 SWS    integer,
 givenBy integer references professors
             on delete set null);
```

University Schema with Integrity Conditions (3/4)

```
create table attend
(matrNr integer references student
        on delete cascade,
 lecNr integer references lectures
        on delete cascade,
 primary key(matrNr, lecNr));
```

University Schema with Integrity Conditions (3/4)

```
create table attend
(matrNr integer references student
        on delete cascade,
 lecNr integer references lectures
        on delete cascade,
 primary key(matrNr, lecNr));

 create table presuppose
(predecessor   integer references lectures
            on delete cascade,
 successor integer references lectures
            on delete cascade,
 primary key(predecessor, successor));
```

University Schema with Integrity Conditions (4/4)

```
create table examine
(matrNr integer references student
        on delete cascade,
 lecNr  integer references lectures,
 persNr integer references professors
        on delete set null,
 Note   numeric(2,1)
        check(Note between 0.7 and 5.0),
 primary key(matrNr, lecNr));
```

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity
3.1 Motivation and Definition
3.2 Guarantee of the Referential Integrity
3.3 Referential Integrity in SQL
3.4 Cyclic Dependencies

4. Trigger

5. Learning Objectives

Cyclic Dependencies – Create

Example (chicken/egg)

create the table chicken and egg, where for every chicken it is noted from which egg it hatched and vice versa:

Cyclic Dependencies – Create

Example (chicken/egg)

create the table chicken and egg, where for every chicken it is noted from which egg it hatched and vice versa:

```
create table chicken
(cID     integer primary key,
 eID     integer references egg);
```

Cyclic Dependencies – Create

Example (chicken/egg)

create the table chicken and egg, where for every chicken it is noted from which egg it hatched and vice versa:

```
create table chicken
(cID    integer primary key,
 eID    integer references egg);
```

```
ERROR: relation ‘‘egg’’ does not exist
```

Cyclic Dependencies – Create

Example (chicken/egg)

create the table chicken and egg, where for every chicken it is noted from which egg it hatched and vice versa:

```
create table chicken
(cID    integer primary key,
 eID    integer references egg);
```

```
ERROR: relation ‘‘egg’’ does not exist
```

```
create table egg
(eID    integer primary key,
 cID    integer references chicken);
```

Cyclic Dependencies – Insert

Example (chicken/egg)

assuming we have created the table:

```
insert into chicken values (1,11);
```

Cyclic Dependencies – Insert

Example (chicken/egg)

assuming we have created the table:

```
insert into chicken values (1,11);
```

```
ERROR: insert or update on table ‘‘chicken’’ violates
foreign key constraint ‘‘chickenrefegg’’
DETAIL: Key (eid)=(11) is not present in table
‘‘egg’’.
```

Cyclic Dependencies – Delete Table

Example (chicken/egg)

assume we want to delete the table:

```
DROP TABLE egg;
```

Cyclic Dependencies – Delete Table

Example (chicken/egg)

assume we want to delete the table:

```
DROP TABLE egg;
```

```
ERROR: cannot drop table egg because other objects
depend on it
DETAIL: constraint chickenrefegg on table chicken
depends on table egg.
```

Cyclic Dependencies – Three Problems

`CREATE TABLE` How can the foreign key of the first table be defined?

`INSERT` How can new tuples be inserted?

`DROP TABLE` How can tables be deleted?

Cyclic Dependencies – Three Problems

`CREATE TABLE` How can the foreign key of the first table be defined?
Lsolution: subsequent create (`ALTER TABLE`)

`INSERT` How can new tuples be inserted?

`DROP TABLE` How can tables be deleted?

Cyclic Dependencies – Three Problems

`CREATE TABLE` How can the foreign key of the first table be defined?
Lsolution: subsequent create (`ALTER TABLE`)

`INSERT` How can new tuples be inserted?
solution: use transactions (`DEFERRED`)

`DROP TABLE` How can tables be deleted?

Cyclic Dependencies – Three Problems

`CREATE TABLE` How can the foreign key of the first table be defined?
Lsolution: subsequent create (`ALTER TABLE`)

`INSERT` How can new tuples be inserted?
solution: use transactions (`DEFERRED`)

`DROP TABLE` How can tables be deleted?
solution: `DROP CONSTRAINT` or `CASCADE`

Cyclic Dependencies – Create

solution: use `alter table` command:

Cyclic Dependencies – Create

solution: use `alter table` command:

Example (chicken/egg)

```
create table chicken(cID integer primary key,
                     eID         integer);

create table egg(
             eID integer primary key,
             cID integer references chicken);
```

Cyclic Dependencies – Create

solution: use `alter table` command:

Example (chicken/egg)

```
create table chicken(cID integer primary key,
                     eID         integer);

create table egg(
             eID integer primary key,
             cID integer references chicken);

alter table chicken
  add constraint chickenrefegg
      foreign key (eID) references egg;
```

Cyclic Dependencies – Insert

solution: transactions; delay checking the conditions until the end of the transaction

Cyclic Dependencies – Insert

solution: transactions; delay checking the conditions until the end of the transaction

Example (chicken/egg)

```
alter table chicken
    add constraint chickenrefegg
        foreign key (eID) references egg
        initially deferred deferrable;
```

Cyclic Dependencies – Insert

solution: transactions; delay checking the conditions until the end of the transaction

Example (chicken/egg)

```
alter table chicken
    add constraint chickenrefegg
        foreign key (eID) references egg
        initially deferred deferrable;

begin;
insert into chicken values(1,11);
insert into egg values(11,1);
commit;
```

Cyclic Dependencies – DEFERRABLE

- determines whether constraints are checked immediately or at the end of transactions

```
[ DEFERRABLE | NOT DEFERRABLE ]
  [INITIALLY DEFERRED|INITIALLY IMMEDIATE]
```

`DEFERRABLE` constraint can be checked only at transaction end

`NOT DEFERRABLE` constraints is checked immediately

`INITIALLY DEFERRED`, `INITIALLY IMMEDIATE` default for transactions, can be changed in transactions

Cyclic Dependencies – Delete Tables

- either constraint is deleted "manually"

```
ALTER TABLE tablename DROP CONSTRAINT ...
```

Cyclic Dependencies – Delete Tables

- either constraint is deleted "manually"

```
ALTER TABLE tablename DROP CONSTRAINT ...
```

- "automatically" via CASCADE

```
DROP TABLE tablename CASCADE;
```

Cyclic Dependencies – Delete Tables

- either constraint is deleted "manually"

```
ALTER TABLE tablename DROP CONSTRAINT ...
```

- "automatically" via CASCADE

```
DROP TABLE tablename CASCADE;
```

Example

```
ALTER TABLE chicken DROP
                    CONSTRAINT chickenrefegg;
DROP TABLE egg CASCADE;
```

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity

4. Trigger

5. Learning Objectives

Trigger

standardized in the SQL-99 standard, but already contained in commercial DBMS before; therefore implementation is different in most DBMS

use to ensure data integrity

work on the event - condition - action model:

- upon arrival of a certain event
- under certain conditions
- are executed automatically by DBMS actions

attention: cyclic trigger or infinite loops!

Trigger: Applications

trigger are used in:

- calculation of stored derived attributes

Example

calculation the gross price with net price and sales tax rate given

Trigger: Applications

trigger are used in:

- calculation of stored derived attributes

Example

calculation the gross price with net price and sales tax rate given

- general conditions

Example

for every given lecture with more than 6SWS there is a bonus professors cannot be degraded

- logging of data base modifications

Syntax (Definition of Triggers)

- definition of a trigger: `create trigger`
- triggering events: `insert`, `update`, `delete`
- "fires" before or after execution: `before`/`after insert`
- tuple or query level: `for each row`/`statement`
- restricted conditions via `when`
- access to values before and after the event: `referencing old`/`new` (only at `for each row`)
- use of procedures in respective DBMS syntax

Trigger (Example)

Example

for every given lecture with more than 6SWS there is a bonus .

```
CREATE TRIGGER lect
AFTER INSERT ON lectures
FOR EACH ROW
WHEN NEW.sws > 6
update professors
set professors.salary=professors.salary+100
where professors.persNr=NEW.lecturedBy;
```

Trigger (Example)

Example

for every given lecture with more than 6SWS there is a bonus .

```
CREATE TRIGGER lect
AFTER INSERT ON lectures
FOR EACH ROW
WHEN NEW.sws > 6
update professors
set professors.salary=professors.salary+100
where professors.persNr=NEW.lecturedBy;
```

attention: in PostgreSQL not available **like this**

CREATE TRIGGER (PostgreSQL)

```
CREATE TRIGGER name { BEFORE | AFTER}
  {INSERT | UPDATE | DELETE}
  ON table_name
  [ REFERENCING { { OLD | NEW } TABLE
                  [ AS ] trans_name } ]
  [ FOR [ EACH ] { ROW | STATEMENT } ]
  [ WHEN ( condition ) ]
  EXECUTE PROCEDURE function_n ( arguments )
```

PL/pgSQL Functions as Triggers

function . . .

- . . . is not allowed to have arguments
- . . . has to have the return value `trigger`

```
CREATE FUNCTION name() RETURNS trigger
 AS $$ ... $$ LANGUAGE plpgsql;
```

PL/pgSQL Functions as Triggers

function ...

- ... is not allowed to have arguments
- ... has to have the return value `trigger`

```
CREATE FUNCTION name() RETURNS trigger
 AS $$ ... $$ LANGUAGE plpgsql;
```

- ... special variables (for instance `NEW`/`OLD`) available
- ... has to return `NULL` or a value that corresponds to the structure of the table

Per-Row Before Trigger

`INSERT` `OLD` undefinable, `NEW` contains the new row

`UPDATE` `OLD` and `NEW` are defined

`DELETE` `OLD` contains the row to be removed, `NEW` undefined

Per-Row Before Trigger

`INSERT` `OLD` undefinable, `NEW` contains the new row

`UPDATE` `OLD` and `NEW` are defined

`DELETE` `OLD` contains the row to be removed, `NEW` undefined

return values

- with `NULL` processing of line is aborted
  (no further trigger, no `INSERT`/`UPDATE`/`DELETE`)
- else: processing is continued with returned value $\Rightarrow$ return of last trigger inserted
  (no effect for `DELETE`)

Trigger (Example)

Example

professors cannot be degraded (PostgreSQL syntax)

```
CREATE OR REPLACE FUNCTION ouf()
   RETURNS TRIGGER AS $$
BEGIN
 IF (OLD.Rang = 'C3' AND NEW.Rang='C2') THEN
  RETURN OLD;
 END IF;
 RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER noDegrading
BEFORE UPDATE ON professor
FOR EACH ROW WHEN (OLD.Rang is not NULL)
EXECUTE PROCEDURE ouf();
```

Overview

1. Overview

2. Integrity Conditions

3. Referential Integrity

4. Trigger

5. Learning Objectives

Learning Objectives

- Which kinds of integrity conditions are there?
- static integrity conditions in SQL
- in particular: foreign key:
  - referential integrity in SQL
- handling of cyclic dependencies
- What are triggers?
- Which information has to be given when defining triggers?
- How does this information influence the behaviour of the trigger?
- What has to be considered with the return value of the trigger?