

# I3306: Base de Données II

## Chap 1: Contraintes / Constraints

UL-Nabatieh

October 01, 2018



# Syllabus

- Creation of DB, tables, constraints, *etc.*
- Stored procedures and functions
- Triggers
- Views
- Security
- Transactions
- Indexing
- Object-relational database

# Remarks

## Remark 1

Requires (lots of) practice / reading: some notions cannot be learned thoroughly in this brief course.

# Remarks

## Remark 1

Requires (lots of) practice / reading: some notions cannot be learned thoroughly in this brief course.

## Remark 2

Should not be your first database class: assume you know basic database concepts.

# Outline

# Data integrity

Data integrity falls mainly into 3 categories:

- 1 Entity integrity
- 2 Domain integrity
- 3 Referential integrity



## Data integrity: entity integrity

- Entity integrity defines a row as a **unique entity** for a particular table;
- Through Primary keys, Unique constraints.

### Example

If a student is entered with an idSt value of 123, the database should not allow another student to have an ID with the same value.



## Data integrity: domain integrity

- Domain integrity is the validity of entries for a given column;
- Enforce domain integrity by restricting the type, the format, or the range of possible values;
- Through FOREIGN KEY constraints, CHECK constraints, DEFAULT definitions, NOT NULL definition.

### Examples

- 1 If you have an ratingEmp column intended to have values ranging from 1 to 5, the database should not accept a value of 6.
- 2 If the table employee has a idDep column that stores the department number for the employee, the database should allow only values that are valid for the department numbers in the company.

# Data integrity: referential integrity

- **Referential integrity** preserves the defined **relationships between tables** when records are entered or deleted;
- Referential integrity ensures that key values are **consistent across tables**;
- Such consistency requires:
  - 1 There be **no references to nonexistent values**;
  - 2 And if a **key value changes, all references to it change** consistently throughout the database.



# Outline

# Types of constraints

- 1 Primary keys
- 2 Unique constraints
- 3 Set NULL constraints
- 4 Foreign keys constraints
- 5 Using **check constraints**



# Primary key

- No two tuples can have the same combination of values for all their attributes;
- Relation schema may have more than one key;
- Underline attribute;
- No primary key value can be NULL.



## Primary key

To create a primary key, we use:

For a **single primary key**:

```
1 create table Teacher
2 (
3   int id primary key,
4   fn nvarchar(10),
5   ln nvarchar(10)
6 )
```

For **multiple primary keys**:

```
1 create table Teaches
2 (
3   int idCourse ,
4   int idTeacher ,
5   primary key(idCourse , idTeacher)
6 )
```

# Unique constraint

- UNIQUE constraints enforce the **uniqueness of the values** in a set of columns;
- In a UNIQUE constraint, **no two rows in the table can have the same value for the columns.**



# Unique

To set an attribute as unique, we use:

```
1 create table Relation1
2 (
3 id int UNIQUE
4 )
```

or

```
1 CREATE TABLE Relation2
2 (id int ,
3  num int ,
4  UNIQUE(id , num))
5 )
6 insert into Relation2(id , num) VALUES (1, 1); just fine
7 insert into Relation2(id , num) VALUES (1, NULL); just fine
8 insert into Relation2(id , num) VALUES (1, 1); unique key violation
9 insert into Relation2(id , num) VALUES (NULL, 1); just fine
10 insert into Relation2(id , num) VALUES (NULL, NULL); just fine
```

## Unique vs. primary keys

### ■ **Unique:**

May be NULL

Multiple unique constraints for a table

### ■ **Primary key:**

Should not be null

Only one primary key constraint can be defined for a table



## NOT NULL constraint

NOT NULL specifies that the **column does not accept NULL values.**

```
1 create table Person
2 (
3 id int NOT NULL primary key ,
4 name varchar(20) NOT NULL
5 )
```



# Referential integrity constraint

- Specified between two relations
- **Maintains consistency** among tuples in two relations
- FOREIGN KEY constraints identify and enforce the relationships between tables



## Foreign keys: referential integrity constraints

```

1 REFERENCES relationName [(attribName)]
3 or
5 FOREIGN KEY (attribName [, attribName]* ) REFERENCES relationName [(
    attribName[, attribName]*)]
6 [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
7 [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

```

- **Example:** for table **R**, we can specify:  
foreign key(**x**, **y**) references **S** (**a**, **b**)
- Requires (**a**, **b**) be **unique or primary key** of **S**;
- Consider a row in **R** with values of **x** as **a1**, and **y** as **b1** where **a1**, **b1** are both non-null. There must be a row in **S** with values for (**a** , **b**) as (**a1**, **b1**)

# Dealing with violation constraints

Basic operations that change the **states of relations** in the database: **insert**, **update**, **remove**.

## Examples of violation constraints:

- Violating referential integrity: if tuple being deleted is referenced by foreign keys from other tuples
- Updating a primary/foreign key



# Maintaining referential integrity: Inserts/Deletes/Updates

- **Default:** reject any modification that violates the constraints;
- We can specify other policies for delete/update as set **null/cascade**;
- **Example:** for Employee relation:

foreign key (idProj) references Project (idProj) ON DELETE SET NULL  
ON UPDATE CASCADE



# On delete

- Let us consider the foreign key on table R referencing table S such as:  
foreign key (x) references S (a)
- SET NULL  
If a delete is performed on S, any rows in R that reference that row in S have their x attributes set to null
- CASCADE  
If a delete is performed on S, any rows in R that reference that row in S are also deleted.



# On Update

- Let us consider the foreign key on table R referencing table S such as:

foreign key (x) references S (a)

- SET NULL

If a update is performed on S, any rows in R that reference that row in S have their x attributes set to null

- CASCADE

If a delete is performed on S, any rows in R that reference that row in S are also updated.



## Foreign key constraints: examples

```
1 create table department
2 (idDep int primary key
3 )
4
5 create table project
6 (idProj int primary key,
7 idDep int references department
8 )
9
10 CREATE TABLE employee(
11 idEmp varchar(20) primary key,
12 idProj int references project
13 )
```



## Foreign key constraints: examples

If we try to remove a department, and there are projects in this department, then the suppression will be refused by DBMS.

### Example

**department** = {1,3,5}      **project** = {(23,1), (65,3), (13,1)}

**delete from department where idDep = 5**

there is no associated project to department 5, then the query will be executed **correctly**.

**delete from department where idDep = 1**

there are 2 associated project to department 1, then the query will be **refused**.

## Foreign key constraints: examples

```
1 create table department
2 (idDep int primary key
3 )
4
5 create table project
6 (idProj int primary key,
7 idDep int references department ON UPDATE CASCADE ON DELETE CASCADE
8 )
9
10 CREATE TABLE employee(
11 idEmp varchar(20) primary key,
12 idProj int references project ON UPDATE SET NULL ON DELETE SET NULL)
```



## Foreign key constraints: ON UPDATE CASCADE

If an update is performed on department (idDep), any rows in projects (idDep) that reference that row in department are also updated.

### Example

**department** = {1,3,5}      **project** = {(23,1), (65,**3**), (13,1)}

update department set idDep=10 where idDep = 3

**department** = {1,**10**,5}      **project** = {(23,1), (65,**10**), (13,1)}



## Foreign key constraints: ON DELETE CASCADE

If a delete is performed on department (idDep), any rows in projects (idDep) that reference that row in department are also deleted.

### Example

**department** = {1,3,5}      **project** = {(23,1), (65,3), (13,1)}

delete from department where idDep = 1

**department** = {3,5}      **project** = {(65,3)}



## Foreign key constraints: ON UPDATE SET NULL

If we modify the attribute idProj of the relation project, the attribute idProj of the relation employee is updated to NULL for the tuples of the concerned employees.

### Example

**project** = {(23,1), (65,3), (13,1)}      **employee** = {(e98,13), (f54,65), (j76,23)}

update project set idProj = 17 where idProj = 13

**project** = {(23,1), (65,3), (17,1)}      **employee** = {(e98,NULL), (f54,65), (j76,23)}



## Foreign key constraints: ON DELETE SET NULL

If we delete the attribute idProj of the relation project, the attribute idProj of the relation employee is updated to NULL for the tuples of the concerned employees.

### Example

**project** = {(23,1), (65,3), (13,1)}      **employee** = {(e98,13), (f54,65), (j76,23)}

delete project where idProj = 13

**project** = {(23,1), (65,3)}      **employee** = {(e98,NULL), (f54,65), (j76,23)}



## Foreign key constraints: example

### Example

**department** = {1,3,5}

**project** = {(23,1), (65,3), (13,1)}

**employee** = {(e98,13), (f54,65), (j76,23)}

delete department set idDep = 1

**department** = {3,5}

## Foreign key constraints: example

### Example

**department** = {1,3,5}

**project** = {(23,1), (65,3), (13,1)}

**employee** = {(e98,13), (f54,65), (j76,23)}

delete department set idDep = 1

**department** = {3,5}

**project** = {(65,3)}

## Foreign key constraints: example

### Example

**department** = {1,3,5}

**project** = {(23,1), (65,3), (13,1)}

**employee** = {(e98,13), (f54,65), (j76,23)}

delete department set idDep = 1

**department** = {3,5}

**project** = {(65,3)}

**employee** = {(e98,NULL), (f54,65), (j76,NULL)}

# The check clause

check(P): P is a **predicate**

## Example 1

Ensure that semester is one of fall, winter, spring or summer:

```
1 create table course
2 (idCourse varchar(20) primary key ,
3  semester varchar(20) check (semester in ( 'Fall ' , 'Winter ' , 'Spring ' , 'Summer' ) )
4 )
```



# The check clause

`check(P)`: P is a **predicate**

## Example 2

Ensure that gender is 'M' or 'F' and salary is greater than 1000:

```
1 create table employee
2 (idEmp int primary key,
3  gender char(1) check (gender in ('M', 'F')),
4  salary int check(salary >1000)
5 )
```



## Remark: Sub-Query in CHECK Constraints

### Problem

Using sub-queries is not allowed within the CHECK constraint expression in **T-SQL** language.

### Example

```
1 create table product(  
2 id int primary key,  
3 qte int check (select SUM(qte) from product >100)  
4 )
```

Not allowed.

## Remark: Sub-Query in CHECK Constraints

**Solution 1:** putting the sub-query within a **scalar function** and use it within the CHECK's expression:

**Example:**

```
1 create function getSum()  
2   returns int  
3 as  
4   begin  
5       declare @res int;  
6       select @res = SUM(qte) from product  
7       return @res  
8   end  
  
10 alter table product  
11 add constraint chSum check (getSum() > 100)
```

**Solution 2:** triggers

Scalar functions and triggers will be presented in next lectures.



## Creating check constraint

Creating CHECK constraints is limited to the same column scope:

### Example:

```
1 create table product(  
2 id int primary key,  
3 qte int check (qte>id)  
4 )
```

Leads to the following error:

Column CHECK Constraint for column qte references another column



## Creating check constraint

**Solution:** putting the definition of check constraint at the end of the all columns definitions.

### Example:

```
1 create table product(  
2 id int primary key,  
3 qte int,  
4 check (qte>id)  
5 )
```



# Outline

# Creating Constraints

Two situations:

- 1 Creating constraint when **creating table**;
- 2 Creating constraint when **table exists**.



## Creating constraint when creating table

### ■ Without constraint name:

```
1 create table product(  
2 id int primary key,  
3 qte int check (qte>0)  
4 )
```

### ■ With constraint name:

```
1 create table product(  
2 id int primary key,  
3 qte int constraint cns_Qte check (qte>0)  
4 )
```



## Creating constraint when table exists

Using the **alter table** syntax:

### Example:

```
1 create table product(  
2   id int primary key,  
3   qte int  
4 )  
  
6 alter table product  
7 add constraint cons_Qte_ID check (qte>id)
```

**Remark:** if there are products whose the quantities are smaller than id, an error message will be appeared.



## Creating constraint when table exists

If we do not want to enforce new condition for the existing data, we use the **nocheck** option:

### Example:

```
1 create table product(  
2 id int primary key,  
3 qte int  
4 )  
  
6 alter table product with nocheck  
7 add constraint cons_Qte check (qte>0)
```



## Creating constraint when table exists

To disable a CHECK constraint, we can use **NOCHECK CONSTRAINT** option:

### Example:

```
1 alter table product  
2 nocheck constraint cons_Qte
```



## Creating check constraint when table exists

To disable all constraints on a table, we can use **NOCHECK CONSTRAINT ALL**:

### Example:

```
1 alter table product  
2 nocheck constraint all
```

