Structuration, transfert et analyse de données d'un ouvrage

Bernd Domer, Fabian Boujon, Yohann Schatz, Elie Torri

h e p i a
institut de recherche i n P A C T
paysage, architecture, construction
et territoire

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

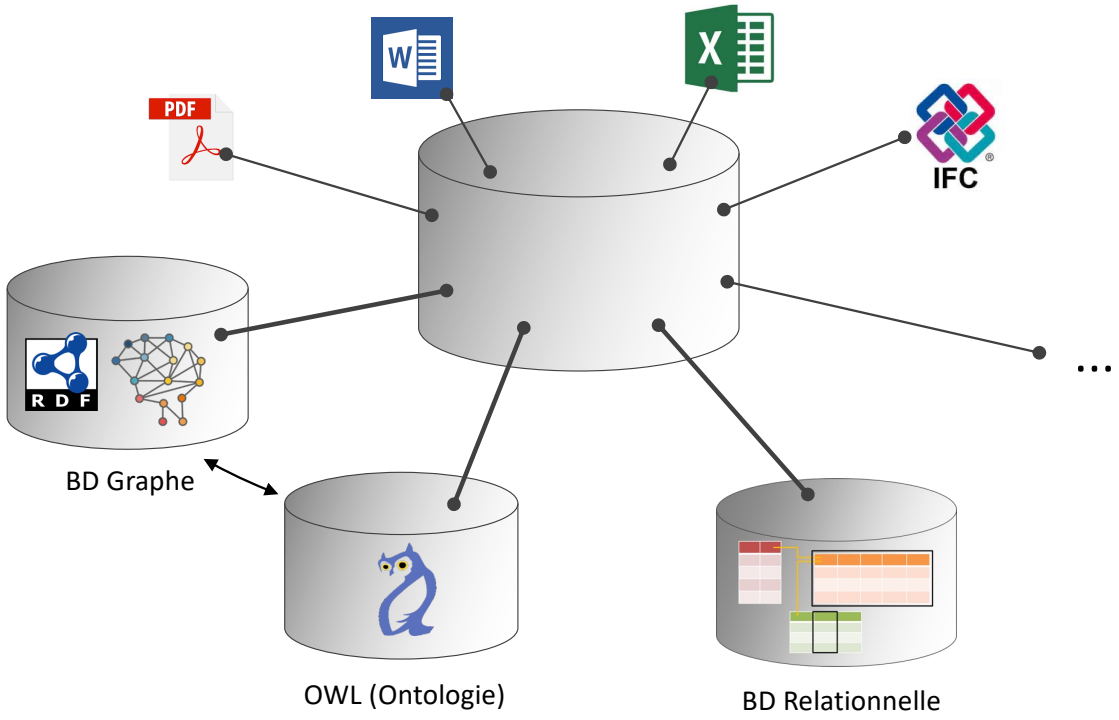
Introduction

Rappels

- Il est important de structurer l'échange d'information tout au long d'un flux de travail BIM
- Les informations sont échangées à travers des **modèles** ou autres **conteneurs** d'information.
- Les informations produites dans un tel projet sont **échangées** entre les différents acteurs, donnant lieu à des **flux d'information structurés**.
- Pour éviter que les informations ne soient pas celles requises, il est indispensable de définir au préalable le contexte des échanges, de les structurer et de les encadrer. C'est l'objectif de la **coordination** et de la gestion du projet BIM.
- Les modèles (qui sont des **conteneurs d'information**) sont créés pour satisfaire les besoins du client (EIR) formulé dans le BEP.

Introduction

Environnement des données



Usages dans la construction

- Données géographiques (www.sitg.ch)
- Données des projets précédents
- Gestion des plans (www.netprojet.ch, www.olmero.ch)
- Coûts des matériaux / des éléments de la construction
- Durée de la construction / séquence de la construction
- Attributs des matériaux liés à leur performance énergétique, à leur durabilité, à l'entretien périodique
- …
- **Le BIM comme base de données gigantesque**

Introduction

Avantages d'une BD

- Fait office de « **single source of truth** » pour les données du projet.
- Permet de **lier/d'agréger** les informations d'un modèle **avec d'autres données** de forme et de source variées.
- Permet d'**interroger, de manipuler et d'exploiter les données de manière assez souple**, ce qui est très avantageux pour certains cas d'usage (ex. établissement automatique des soumissions, introduction des données dans un système de GMAO, etc.)
- **Une BD est pérenne**, contrairement aux ordinateurs/logiciels/formats de données (les bases de données relationnelles, par exemple, existent depuis plus de 50 ans!). Cela permet de garantir l'intégrité des données dans le temps.

Introduction

Objectifs liés à l'utilisation d'une BD

- Fournir des représentations **utiles et intuitives** à ceux qui les utiliseront.
- Éviter les **redondances**.
- Extensibilité et robustesse : ajouter, modifier et supprimer des données avec le moins d'effets secondaires possibles. En d'autres termes, **minimiser les anomalies de mise à jour**.

Introduction

Usages dans la construction

- Une base de données peut être alimentée :
  - A partir d'un logiciel BIM grâce à des canaux/interfaces de connexion spécifiques (ODBC, etc.).
  - A partir d'autres logiciels ou bases de données également dotés d'une interface de connexion (par exemple Excel).
  - Manuellement, en utilisant un système de gestion de base de données (SGBD).

Une constante dans la construction est que les informations sont **modifiées de façon permanente**, surtout pendant les phases de conception et d'exécution.

Introduction

Types de BD

- Hiérarchique
- **Relationnelle → MySQL**
- **Orientée objet → BiMserver**
- **Orientée graphe (Triple store) → GraphDB**
- […]
- Distribué
- Multimédia

Introduction

BiMserver

- Open source (attention, utilisez bien le lien https://bimserver.org/)
- Orienté objet, structuré selon les classes IFC
- Propose un «viewer» intégré
- Possède un langage pour formuler des requêtes sur les données du modèle

Introduction

Triple store

- Adapté pour les graphes RDF (triplets)
- Supporte OWL (Web Ontology Language), un langage de description d'ontologie construit sur le modèle de données de RDF.
- La structure « RDF » (graphe) est avantageuse pour analyser des liens entre objets
- RDF et OWL font partie des standards « en avance sur leur temps », envisagés par buildingSMART dans le futur.

GraphDB™

RDF

Structurée en RDF

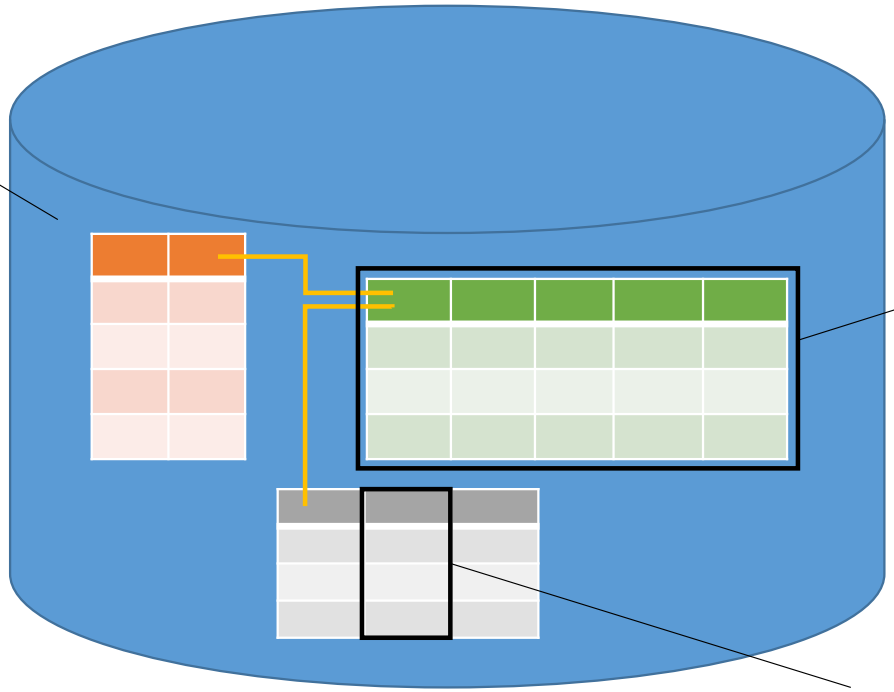
Introduction

BD relationnelle

- Technologie éprouvée depuis des décennies
- La structuration de données en tableau est facile à comprendre
- La plupart des logiciels possèdent une interface «ODBC» pour pouvoir lier une BD relationnelle
- Le langage de requête «SQL» est très puissant et bien documenté

Un peu de vocabulaire...

La BD relationnelle
(contenant les relations)



Une relation
(**ou** table relationnelle)

Un attribut
(colonne)

Un peu de vocabulaire...

Employé	persNo	nom	prénom	dateNaiss	avsNr	deptNr
Multiplet	1					
Multiplet	2					
Multiplet	3					

Schéma relationnel «Employé»

Nombre des multiplets: 3

- Un **schéma relationnel** est l'ensemble des attributs d'une relation («relations sans contenu»)
- Une **relation** est l'ensemble des **multiplets**
- **Attribut**: nom d'une propriété

Un peu de vocabulaire...

Clé primaire

Clé secondaire

Employé	persNo	nom	prénom	dateNaiss	avsNr	deptNr
Multiplet	1					
Multiplet	2					
Multiplet	3					

Schéma relationnel «Employé»

Nombre des multiplets: 3

Clé primaire: le choix d'une clé primaire est une décision de conception. Il est choisi parmi les clés des relations, et les autres clés sont des clés secondaires.

Ex.: Clés de la relation: persNo, nom+prénom+dateNaiss, avsNr
Clé primaire choisi: persNo

Un peu de vocabulaire...

Clé primaire

Clé secondaire

Clé étrangère

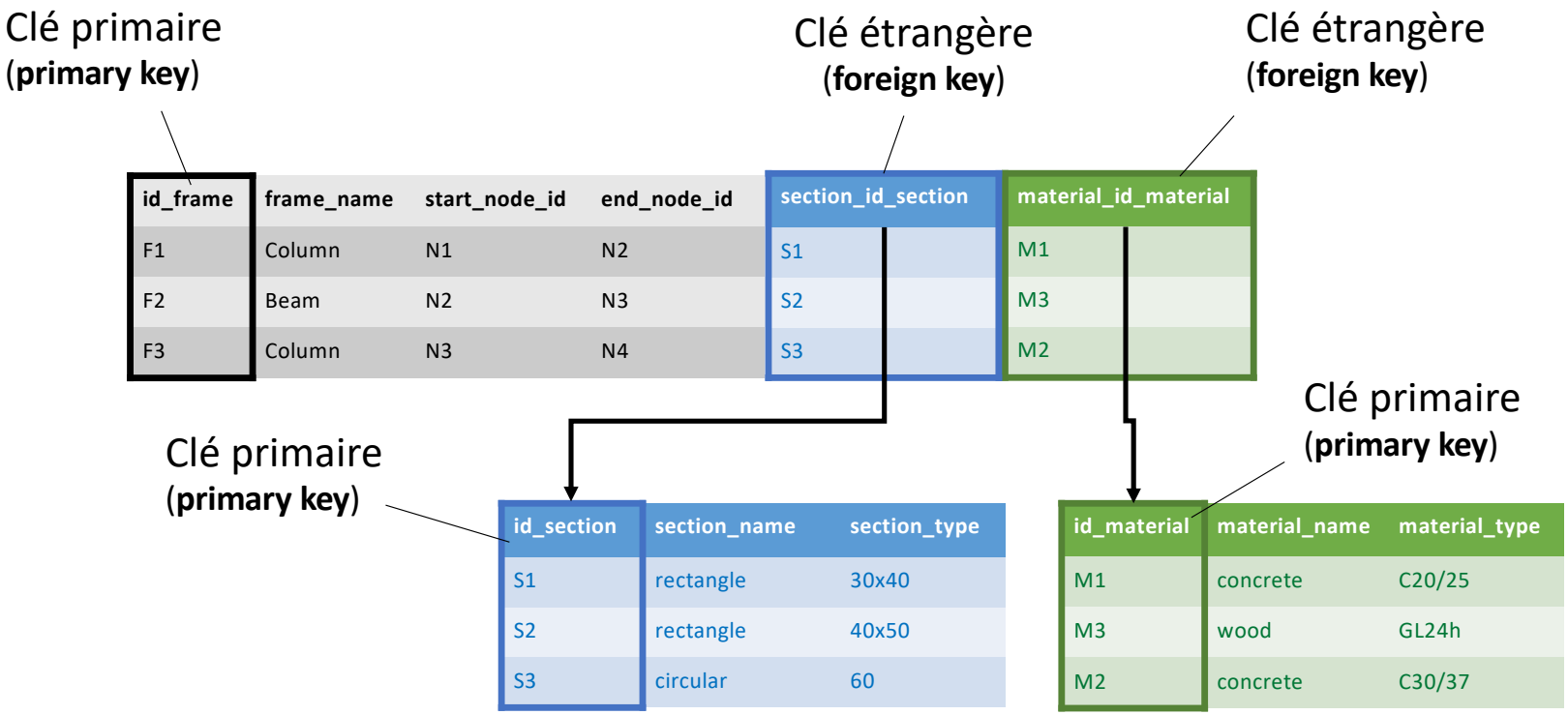
Employé	persNo	nom	prénom	dateNaiss	avsNr	deptNr
Multiplet	1					
Multiplet	2					
Multiplet	3					

Schéma relationnel «Employé»

Nombre des multiplets: 3

Clé étrangère: Une clé étrangère n'identifie pas un multiplet à l'intérieur de la relation, mais fais la référence à une clé primaire dans une autre relation. Une clé étrangère met deux multiplets des relations différents en relation.

Un peu de vocabulaire...

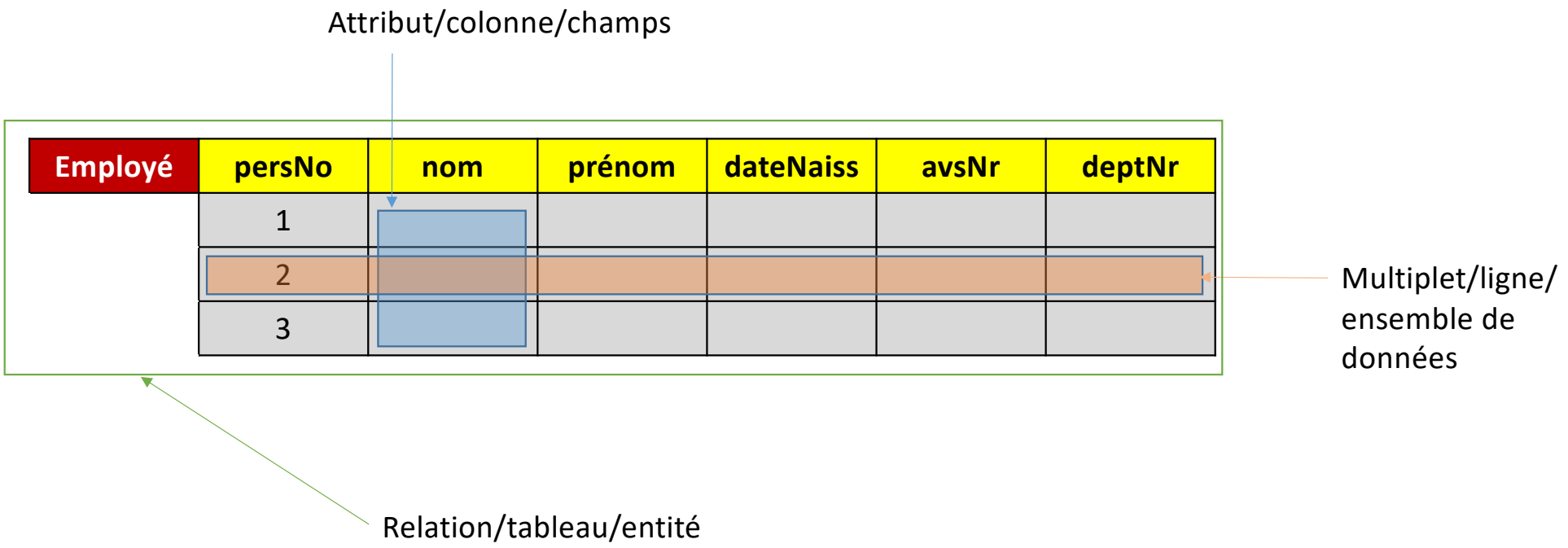


id_frame	frame_name	start_node_id	end_node_id	section_id_section	material_id_material
F1	Column	N1	N2	S1	M1
F2	Beam	N2	N3	S2	M3
F3	Column	N3	N4	S3	M2

id_section	section_name	section_type
S1	rectangle	30x40
S2	rectangle	40x50
S3	circular	60

id_material	material_name	material_type
M1	concrete	C20/25
M3	wood	GL24h
M2	concrete	C30/37

Un peu de vocabulaire...



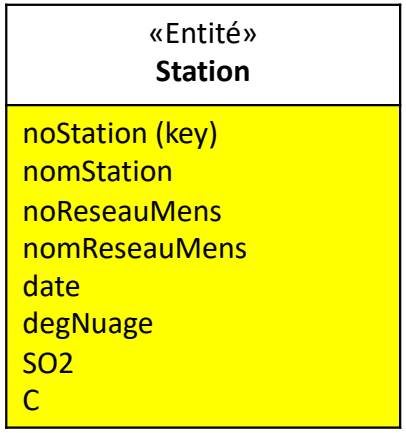
Types de données (pour MySQL)

MySQL est une base de données relationnelle

Type	**Explication / exemple**
INT	Integer – Entier : -2147483648/2147483647
FLOAT	Float – virgule flottante: 1,2345; 34,32; etc.
DATE	AAMMJJ; AA+MM+JJ; etc.
VARCHAR	Alphanumérique: de 1 à 255 caractères

Implémentation d'un modèle relationnel
Point de départ

- Modèle UML simple pour un système de mesure des polluants

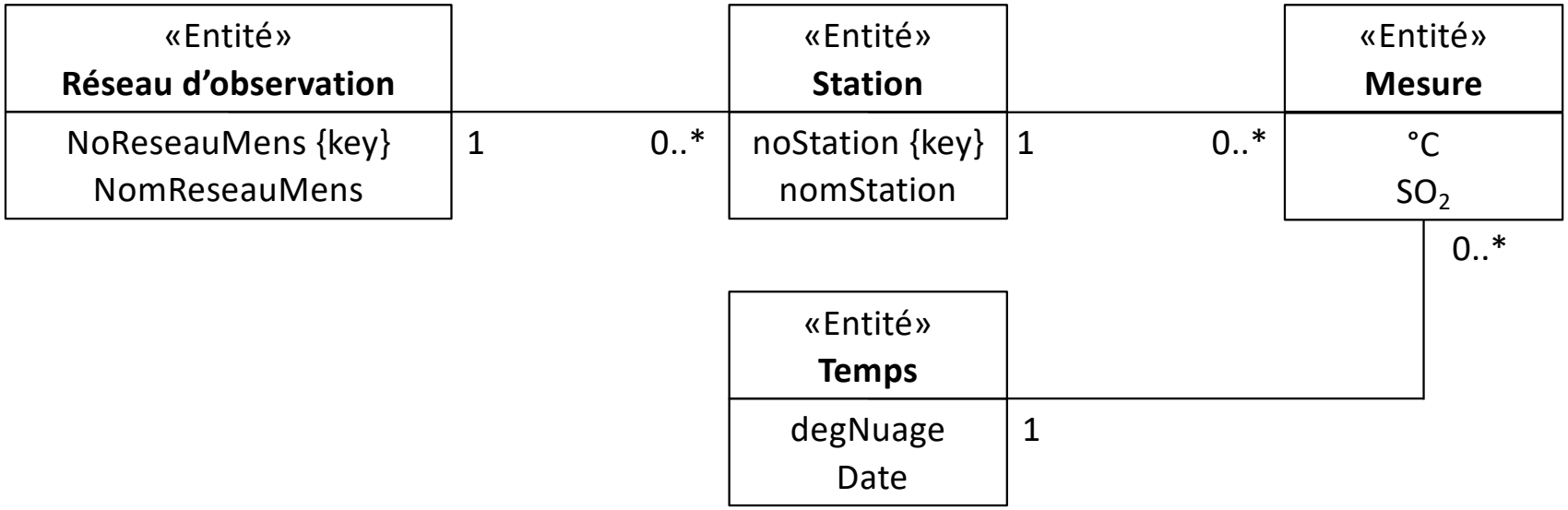


Modèle conceptuel avec UML

Le modèle conceptuel est utile pour montrer

- Le besoin d'information
- Les relations

Exemple: Modèle de données pour l'acquisition des polluants dans l'air aux sites diverses



Une table par classe UML...

Clé primaire

«Table» **Station**
«PK» noStation nomStation noReseauMens nomReseauMens date degNuage SO_2 °C

Station	no Station	nom Station	no Reseau Mens	nom Reseau Mens	date	degNuage	°C	SO_2
	1	Aare	1	Uni	1.6.01	fortement	21.3	10
					2.6.01	moyen	18.2	12
					3.6.01	faible	12.8	14
	2	Rathaus	2	Canton	1.6.01	fortement	21.8	21
					2.6.01	moyen	18.9	20
					3.6.01	faible	13.8	22
	3	Loeb	2	Canton	1.6.01	fortement	21.0	23
					2.6.01	moyen	17.2	26
					3.6.01	faible	14.8	28

La normalisation

Évite...

- des redondances et ainsi les contradictions potentielles
- des effets secondaires non désirés, comme des anomalies lorsque des données sont insérées, modifiés ou effacés

Permet...

- d'établir le principe de «one fact in one place»
- d'établir un modèle de données qui est compris et interprété par chaque développeur
- que les informations sont disponibles

Les formes normales

0NF Chaque table possède une clé primaire

1NF Les domaines des toutes les attributs sont atomiques

2NF Dans des tableaux avec plusieurs clés primaires, chaque attribut qui ne fait pas partie de la clé, doit être dépendante de l'intégralité des clés primaires. Des dépendances partielles ne sont pas admise.

3NF Entre deux attributs, qui ne font pas partie de la clé, il ne doit y avoir aucune dépendance. Les dépendances *transitives* ne sont pas permises.

Forme 1NF

Clé primaire

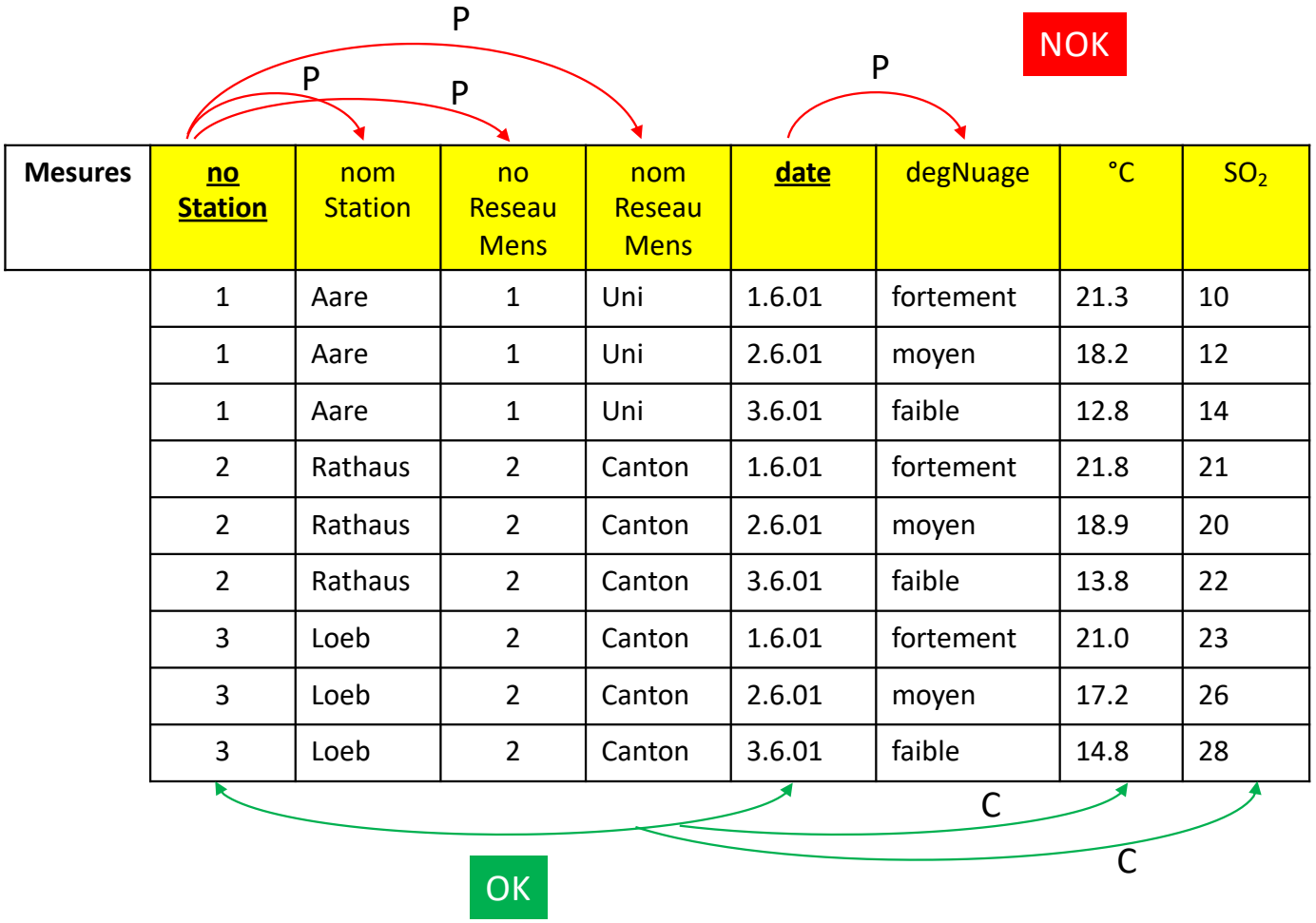
Il s'agit d'une clé primaire ***composée***, car plus qu'un attribut est utilisé.

Mesures	**no Station**	nom Station	no Reseau Mens	nom Reseau Mens	**date**	degNuage	°C	SO_2
	1	Aare	1	Uni	1.6.01	fortement	21.3	10
	1	Aare	1	Uni	2.6.01	moyen	18.2	12
	1	Aare	1	Uni	3.6.01	faible	12.8	14
	2	Rathaus	2	Canton	1.6.01	fortement	21.8	21
	2	Rathaus	2	Canton	2.6.01	moyen	18.9	20
	2	Rathaus	2	Canton	3.6.01	faible	13.8	22
	3	Loeb	2	Canton	1.6.01	fortement	21.0	23
	3	Loeb	2	Canton	2.6.01	moyen	17.2	26
	3	Loeb	2	Canton	3.6.01	faible	14.8	28

La table (la relation) est dans sa forme ***atomique***, car les valeurs «1», «Aaare», «Uni», etc., ne peuvent plus être divisées dans des données pertinentes. Par exemple, les types de données comme «Integer», «Real», ou des «Char» avec une longueur fixe sont considérés comme atomiques.
«Lists», «Arrays» (un étalage d'objets) ne sont pas atomiques.

Transfer en 2NF
Détecter des dépendances fonctionnelles

P: dépendance partielle
C: dépendance complète



Mesures	**no Station**	nom Station	no Reseau Mens	nom Reseau Mens	**date**	degNuage	°C	SO_2
	1	Aare	1	Uni	1.6.01	fortement	21.3	10
	1	Aare	1	Uni	2.6.01	moyen	18.2	12
	1	Aare	1	Uni	3.6.01	faible	12.8	14
	2	Rathaus	2	Canton	1.6.01	fortement	21.8	21
	2	Rathaus	2	Canton	2.6.01	moyen	18.9	20
	2	Rathaus	2	Canton	3.6.01	faible	13.8	22
	3	Loeb	2	Canton	1.6.01	fortement	21.0	23
	3	Loeb	2	Canton	2.6.01	moyen	17.2	26
	3	Loeb	2	Canton	3.6.01	faible	14.8	28

La table (la relation) est dans sa deuxième forme normale, ***2NF***, lorsque les valeurs dépendent intégralement de la clé primaire (composée).

Transfer en 3NF

T

T: dépendance transitive

Station	**no Station**	nom Station	no Reseau Mens	nomReseau Mens
	1	Aare	1	Uni
	2	Rathaus	2	Canton
	3	Loeb	2	Canton

Mesures	**no Station**	**date**	°C	SO_2
	1	1.6.01	21.3	10
	1	2.6.01	18.2	12
	1	3.6.01	12.8	14
	2	1.6.01	21.8	21
	2	2.6.01	18.9	20
	2	3.6.01	13.8	22
	3	1.6.01	21.0	23
	3	2.6.01	17.2	26
	3	3.6.01	14.8	28

Les dépendances partielles ont étés éliminés

Temps	**Date**	degNuage
	1.6.01	fortement
	2.6.01	moyen
	3.6.01	faible

La table (la relation) est dans sa troisième forme normale, ***3NF***, lorsque les valeurs qui ne sont pas des clés primaires sont mutuellement indépendantes.

Forme 3NF

Les dépendances transitives ont été éliminés

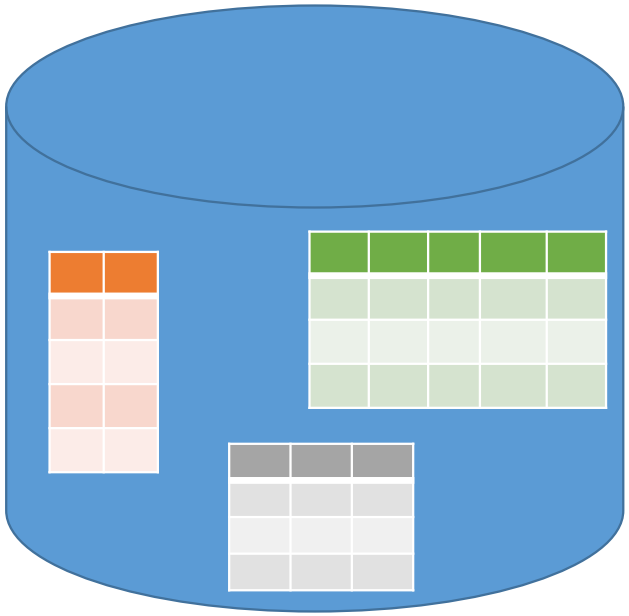
Station	**noStation**	nomStation	noReseau Mens
	1	Aare	1
	2	Rathaus	2
	3	Loeb	2

Mesure	**noStation**	**Date**	°C	SO_2
	1	1.6.01	21.3	10
	1	2.6.01	18.2	12
	1	3.6.01	12.8	14
	2	1.6.01	21.8	21
	2	2.6.01	18.9	20
	2	3.6.01	13.8	22
	3	1.6.01	21.0	23
	3	2.6.01	17.2	26
	3	3.6.01	14.8	28

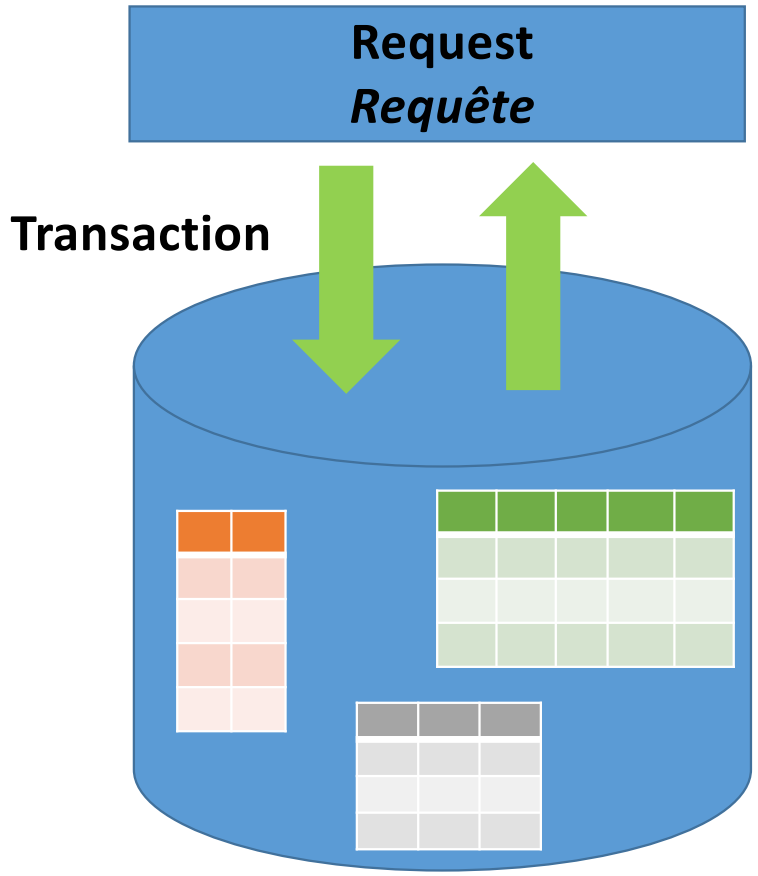
Reseau mensuration	**no Reseau Mens**	nomReseau Mens
	1	Uni
	2	Canton

Temps	**Dates**	degNuage
	1.6.01	fortement
	2.6.01	moyen
	3.6.01	faible

Database / Database management system



Database
Base de données



Database management system
Système de gestion de données

Transaction

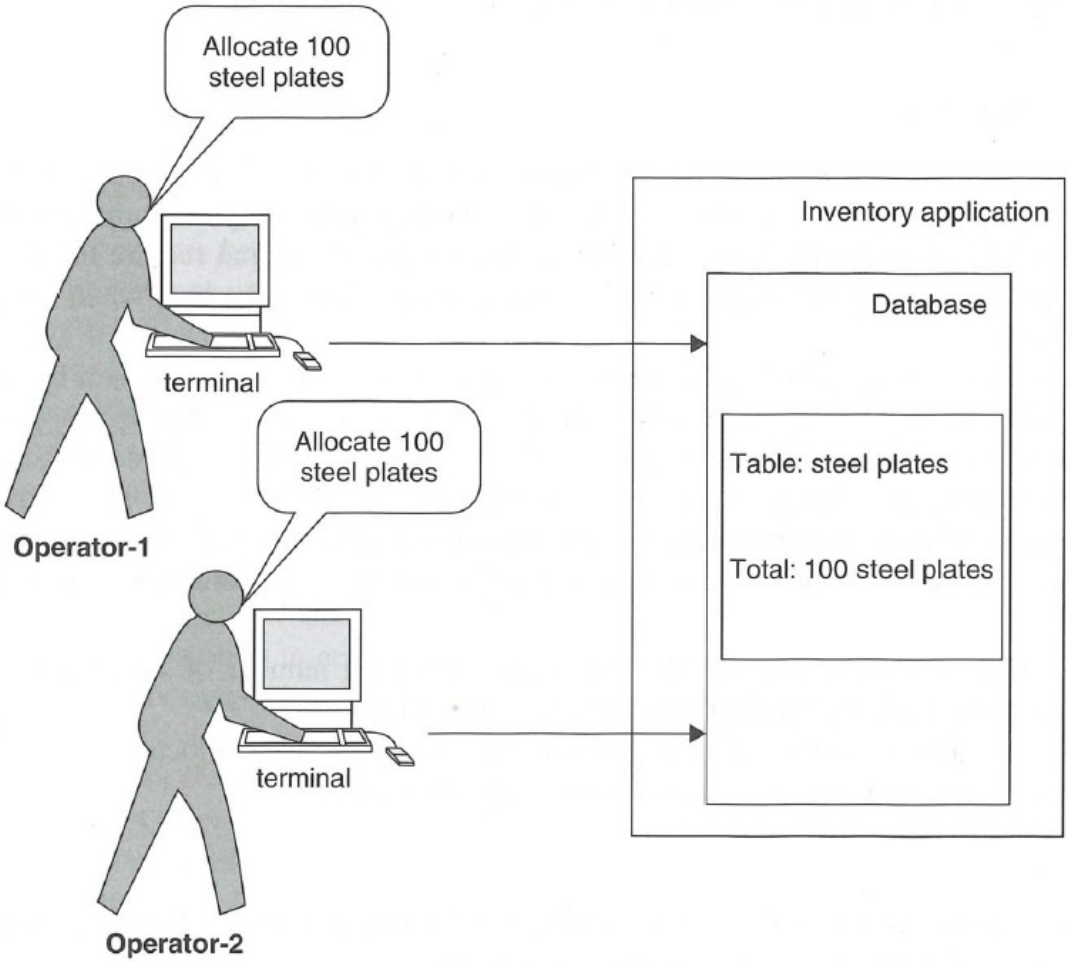
En informatique, et particulièrement dans les bases de données, une **transaction** telle qu'une réservation, un achat ou un paiement est mise en œuvre via une suite d'opérations qui font passer la base de données d'un état A – antérieur à la transaction – à un état B postérieur et des mécanismes permettent d'obtenir que cette suite soit à la fois atomique, cohérente, isolée et durable (ACID).

Source: Wikipédia

Transaction

- **Atomique** : La suite d'opérations est indivisible, en cas d'échec en cours d'une des opérations, la suite d'opérations doit être complètement annulée (rollback) quel que soit le nombre d'opérations déjà réussies.
- **Cohérente** : Le contenu de la base de données à la fin de la transaction doit être cohérent sans pour autant que chaque opération durant la transaction donne un contenu cohérent. Un contenu final incohérent doit entraîner l'échec et l'annulation de toutes opérations de la transaction.
- **Isolée** : Lorsque deux transactions A et B sont exécutées en même temps, les modifications effectuées par A ne sont ni visibles par B, ni modifiables par B tant que la transaction A n'est pas terminée et validée (commit).
- **Durable** : Une fois validé, l'état de la base de données doit être permanent, et aucun incident technique (exemple: crash) ne doit pouvoir engendrer une annulation des opérations effectuées durant la transaction.

Transaction



Source:
Raphael, B., Smith, Ian F. C.:
Engineering Informatics –
Fundamentals of Computer-Aided
Engineering, 2nd edition, Wiley

Pratique

- Lier une base de données avec un logiciel BIM (Doc. Partie 1.)

SQL : généralités

- **SQL** (sigle de *Structured Query Language*, en français **langage de requête structurée**) est un langage informatique normalisé servant à exploiter des bases de données relationnelles.
- Créé en 1974, normalisé depuis 1986, le langage est reconnu par la grande majorité des systèmes de gestion de bases de données relationnelles (abrégé SGBDR) du marché.

Source: Wikipedia

Les commandes proposées par SQL peuvent être divisé en

- Opérations prévues pour définir la structure de la donnée (p. ex.: création des tableaux)
- Opérations pour manipuler la donnée (p. ex. d'insérer d'effacer des lignes dans un tableau)

Source: Raphael, B., Smith, Ian F. C.: Engineering Informatics – Fundamentals of Computer-Aided Engineering, 2nd edition, Wiley

CREATE/DROP DATABASE

Pour créer une base de données qui sera appelée « ma_base » il suffit d'utiliser la requête suivante :

```
CREATE DATABASE ma_base
```

Pour supprimer la base de données « ma_base », la requête est la suivante :

```
DROP DATABASE ma_base
```

Attention : cela va supprimer toutes les tables et toutes les données de cette base.

Source: Archambeau, Cours SQL

CREATE TABLE

La commande CREATE TABLE permet de créer une table en SQL. Un tableau est une entité qui est contenu dans une base de données pour stocker des données ordonnées dans des colonnes. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacun des colonne (entier, chaine de caractères, date, valeur binaire …).

Syntaxe

La syntaxe générale pour créer une table est la suivante :

```
CREATE TABLE nom_de_la_table
(
    colonne1 type_donnees,
    colonne2 type_donnees,
    colonne3 type_donnees,
    colonne4 type_donnees
)
```

Dans cette requête, 4 colonnes ont été définies. Le mot-clé ≪ type_données ≫ sera à remplacer par un mot-clé pour définir le type de données (INT, DATE, TEXT …). Pour chaque colonne, il est également possible de définir des options telles que (liste non-exhaustive) :

- **NOT NULL :** empêche d'enregistrer une valeur nulle pour une colonne.
- **DEFAULT :** attribuer une valeur par défaut si aucune donnée n'est indiquée pour cette colonne lors de l'ajout d'une ligne dans la table.
- **PRIMARY KEY :** indiquer si cette colonne est considérée comme clé primaire pour un index.

Source: Archambeau, Cours SQL

CREATE TABLE

Exemple

Imaginons que l'on souhaite créer une table utilisateur, dans laquelle chaque ligne correspond à un utilisateur inscrit sur un site web. La requête pour créer cette table peut ressembler à ceci :

```
CREATE TABLE utilisateur
(
    id INT PRIMARY KEY NOT NULL,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    email VARCHAR(255),
    date_naissance DATE,
    pays VARCHAR(255),
    ville VARCHAR(255),
    code_postal VARCHAR(5),
    nombre_achat INT
)
```

Voici des explications sur les colonnes créées :

- **id :** identifiant unique qui est utilise comme clé primaire et qui n'est pas nulle
- **nom :** nom de l'utilisateur dans une colonne de type VARCHAR avec un maximum de 100 caractères au maximum
- **prenom :** idem mais pour le prénom
- **email :** adresse email enregistre sous 255 caractères au maximum
- **date_naissance :** date de naissance enregistre au format AAAA-MM-JJ (exemple : 1973-11-17)
- **pays :** nom du pays de l'utilisateur sous 255 caractères au maximum
- **ville :** idem pour la ville
- **code_postal :** 5 caractères du code postal
- **nombre_achat :** nombre d'achat de cet utilisateur sur le site

Source: Archambeau, Cours SQL

SELECT

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce a la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

Commande basique

L'utilisation basique de cette commande s'effectue de la manière suivante :

```
SELECT nom_du_champ
FROM nom_du_tableau
```

Cette requête va sélectionner (SELECT) le champ ≪ nom_du_champ ≫ provenant (FROM) du tableau appelé ≪ nom_du_tableau ≫.

Exemple

Imaginons une base de données appelée ≪ client ≫ qui contient des informations sur les clients d'une entreprise.

Table « client » :

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

Si l'ont veut avoir la liste de toutes les villes des clients, il suffit d'effectuer la requête suivante :

```
SELECT ville
FROM client
```

Résultat :

ville
Paris
Nantes
Lyon
Marseille
Grenoble

Source: Archambeau, Cours SQL

Ordre avec SELECT

Fonctionnalités disponibles avec la commande SELECT:

- joindre un autre tableau aux résultats
- filtrer
- classer les résultats d'une requête
- grouper les résultats

```
SELECT *
FROM table
WHERE condition
GROUP BY expression
HAVING condition
{ UNION | INTERSECT | EXCEPT }
ORDER BY expression
LIMIT count
OFFSET start
```

A noter : cette requête imaginaire sert principale d'aide-mémoire pour savoir dans quel ordre sont utilisé chacun des commandes au sein d'une requête SELECT.

Source: Archambeau, Cours SQL

WHERE

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

Syntaxe

La commande WHERE s'utilise en complément a une requête utilisant SELECT. La façon la plus simple de l'utiliser est la suivante :

```
SELECT nom_colonnes
FROM nom_table
WHERE condition
```

Exemple

Imaginons une base de données appelée ≪ client ≫ qui contient le nom des clients, le nombre de commandes qu'ils ont effectues et leur ville :

id	nom	nbr_commande	ville
1	Paul	3	paris
2	Maurice	0	rennes
3	Joséphine	1	toulouse
4	Gérard	7	paris

Pour obtenir seulement la liste des clients qui habitent à Paris, il faut effectuer la requête suivante :

```
SELECT *
FROM client
WHERE ville = 'paris'
```

Résultat :

id	nom	nbr_commande	nbr_commande
1	Paul	3	paris
4	Gérard	7	paris

Attention : dans notre cas tout est en minuscule donc il n'y a pas eu de problème. Cependant, si un table est sensible à la casse, il faut faire attention aux majuscules et minuscules.

Source: Archambeau, Cours SQL

Opérateurs de comparaison

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Source: Archambeau, Cours SQL

AND et OR

L'opérateur AND permet de s'assurer que la condition1 ET la condition2 sont vrai :

```
SELECT nom_colonnes
FROM nom_table
WHERE condition1 AND condition2
```

L'opérateur OR vérifie quant à lui que la condition1 OU la condition2 est vrai :

```
SELECT nom_colonnes FROM nom_table
 WHERE condition1 OR condition2
```

Ces opérateurs peuvent être combinés à l'infini et mélangés. L'exemple ci-dessous filtre les résultats de la table « nom_table » si condition1 ET condition2 OU condition3 est vrai :

```
SELECT nom_colonnes FROM nom_table
 WHERE condition1 AND (condition2 OR condition3)
```

Attention : il faut penser à utiliser des parenthèses lorsque c'est nécessaire. Cela permet d'éviter les erreurs car et ça améliore la lecture d'une requête par un humain.

ORDER BY

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

Syntaxe

Une requête ou l'ont souhaite filtrer l'ordre des résultats utilise la commande ORDER BY de la sorte :

```
SELECT colonne1, colonne2
FROM table
ORDER BY colonne1
```

Par défaut les résultats sont classes par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus élaboré ressemblerais alors à cela :

```
SELECT colonne1, colonne2, colonne3
FROM table
ORDER BY colonne1 DESC, colonne2 ASC
```

A noter : il n'est pas obligé d'utiliser le suffixe « ASC » sachant que les résultats sont toujours classe par ordre ascendant par défaut. Toutefois, c'est plus pratique pour mieux s'y retrouver, surtout si on a oublié l'ordre par défaut.

Exemple

Pour l'ensemble de nos exemple, nous allons prendre un base « utilisateur » de test, qui contient les données suivantes :

id	nom	prenom	date_inscription	tarif_total
1	Durand	Maurice	2012-02-05	145
2	Dupond	Fabrice	2012-02-07	65
3	Durand	Fabienne	2012-02-13	90
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27

Pour récupérer la liste de ces utilisateurs par ordre alphabétique du nom de famille, il est possible d'utiliser la requête suivante :

```
SELECT *
FROM utilisateur
ORDER BY nom
```

Source: Archambeau, Cours SQL

ORDER BY

Résultat :

id	nom	prenom	date_inscription	tarif_total
4	Dubois	Chloé	2012-02-16	98
5	Dubois	Simon	2012-02-23	27
2	Dupond	Fabrice	2012-02-07	65
1	Durand	Maurice	2012-02-05	145
3	Durand	Fabienne	2012-02-13	90

Résultat :

id	nom	prenom	date_inscription	tarif_total
5	Dubois	Simon	2012-02-23	27
4	Dubois	Chloé	2012-02-16	98
2	Dupond	Fabrice	2012-02-07	65
3	Durand	Fabienne	2012-02-13	90
1	Durand	Maurice	2012-02-05	145

En utilisant deux méthodes de tri, il est possible de retourner les utilisateurs par ordre alphabétique ET pour ceux qui ont le même nom de famille, les trier par ordre décroissant d'inscription. La requête serait alors la suivante :

```
SELECT *
FROM utilisateur
ORDER BY nom, date_inscription DESC
```

Source: Archambeau, Cours SQL

JOIN

Jointure SQL

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

Exemple

En général, les jointures consistent à associer des lignes de 2 tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table. Imaginons qu'une base de 2 données possède une table ≪ utilisateur ≫ et une autre table ≪ adresse ≫ qui contient les adresses de ces utilisateurs. Avec une jointure, il est possible d'obtenir les données de l'utilisateur et de son adresse en une seule requête.

On peut aussi imaginer qu'un site web possède une table pour les articles (titre

, contenu, date de publication …) et une autre pour les rédacteurs (nom, date d'inscription, date de naissance …). Avec une jointure il est possible d'effectuer une seule recherche pour afficher un article et le nom du rédacteur. Cela évite d'avoir à afficher le nom du rédacteur dans la table ≪ article ≫.

Il y a d'autres cas de jointures, incluant des jointures sur la même table ou des jointures d'inégalité. Ces cas étant assez particulier et pas si simple à comprendre, ils ne seront pas élaborés sur cette page.

INNER JOIN

Jointure interne pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est l'une des jointures les plus communes.

```
SELECT id, A1, B1
FROM Table A
INNER JOIN Table B on Table A.id = Table B.id
```

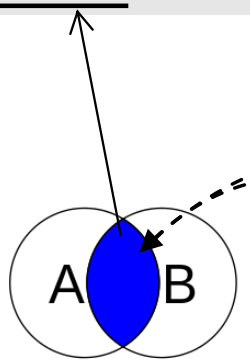


Table A

id	A1
1	…
2	…
3	…
4	…

Table B

id	B1
1	…
2	…
4	…
5	…

Résultat

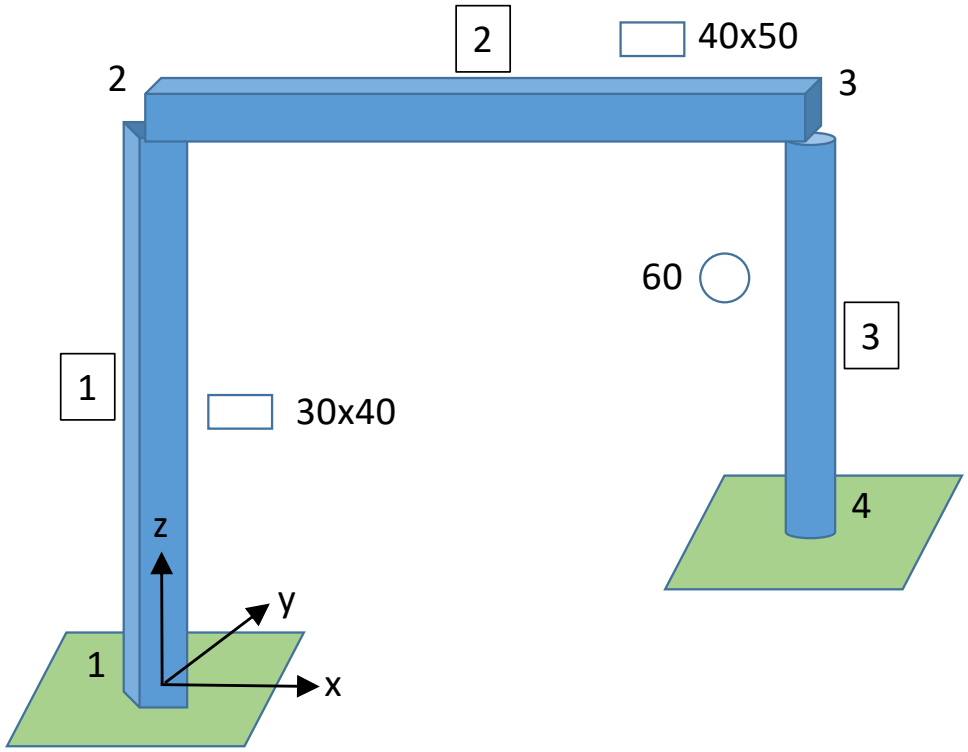
id	A1	B1
1	…	…
2	…	…
4	…	…

Pratique

- Lier la base de données avec un logiciel BIM (Doc Partie 1).

Pratique

Développez une base de données relationnel pour un portique



Pratique/Rendue «Database design»

Développez une base de données relationnelle pour un portique
Il faut que les tables puissent composer les informations suivantes:

- Positionnement de chaque élément dans l'espace (points de départ et d'arrivée).
- Les caractéristiques d'éléments: géométrie des sections (rectangulaire, circulaire), matériaux (acier / bois / béton)

Déposez votre «Database design» (les tables) sur moodle dans le répertoire prévu à cet effet avant le 06.01.2025, 22h00.

De préférence, utilisez un tableur (p. ex. Excel).

Pratique

- Exercices, partie 2.3

Merci pour votre attention !

Si vous avez des questions, n’hésitez pas à les partager