Bases de données

Quelques éléments du langage SQL

1. Architecture

1. Architecture
2. Représentation mathématique

1. Architecture
2. Représentation mathématique
3. Syntaxe SQL

Une **base de données relationnelle** est un ensemble fini de tables, contenant des informations relatives à un même thème, et dont certaines apparaissent sur plusieurs tables.

Une **base de données relationnelle** est un ensemble fini de tables, contenant des informations relatives à un même thème, et dont certaines apparaissent sur plusieurs tables.

Exemple

IHMC : informations relatives à l'alpinisme sur les plus hauts sommets.
Elle comporte 7 tables, dont seules 3 vont nous intéresser ici :

1. Summits
2. Climbers
3. FirstAscent

Summits

N	Alt	UName	Range	C1	C2	Coord
1	8850	Everest	Himalaya	NP	CN	N.27.59.17 E.86.55.31
2	8612	K2	Karakoram	PK	CN	N.35.52.57 E.76.30.48
3	8586	Kangch	Himalaya	NP	IN	N.27.42.09 E.88.08.49
4	8516	Lhotse	Himalaya	NP	CN	N.27.57.42 E.86.55.59
5	8505	Yalung	Himalaya	NP	{null}	N.27.42-19 E.88.08.09
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Climbers

Id	firstN	lastN	nation	birth	death
1	Noel	O'Dell	GB	1890-12-25	1987-02-21
2	Bill	Tilman	GB	1898-02-14	1977-02-18
3	Franck	Smythe	GB	1900-07-06	1949-06-27
4	Eric	Shipton	GB	1907-08-01	1977-03-28
5	Herbert	Tichy	AT	1912-06-01	1987-09-26
6	Josepp	Jochler	AT	1914-08-18	1991-11-11
7	Tenzing	Norgay	NP	1914-05-29	1986-05-09
8	Edmund	Hillary	NZ	1919-07-20	2008-01-11
9	Ernst	Reiss	CH	1920-02-24	2010-08-03
⋮	⋮	⋮	⋮	⋮	⋮

FirstAscent

N	Date	C1	C2	side	route	grade
1	1953-05-29	7	8	W Khumbu	SE ridge	F1
2	1954-07-31	12	15	S Godwin Austen	SE spur	F4
3	1955-05-25	14	16	W Kangchen	NE ridge	F2X
4	1956-05-18	11	13	W Khumbu	NE spur	F3
5	1973-05-14	256	257	W Yalung	N ridge	F3
6	1955-05-15	17	22	NE Kangchung	N ridge	E3+
7	1970-05-27	221	222	SW Shar	W ridge	E3
8	1954-10-09	18	21	W Nangpa	W ridge	E1
9	1960-05-13	23	28	E Churen	NE ridge	E3
⋮	⋮	⋮	⋮	⋮	⋮	⋮

- La création, le remplissage, la mise à jour, la sauvegarde etc. constituent l'**administration** de la base de donnée.

- La création, le remplissage, la mise à jour, la sauvegarde etc. constituent l'**administration** de la base de donnée.
- Les mécanismes d'interrogation de cette base constituent les **requêtes** ; *c'est la seule partie officiellement au programme en CPGE.*

- La création, le remplissage, la mise à jour, la sauvegarde etc. constituent l'**administration** de la base de donnée.
- Les mécanismes d'interrogation de cette base constituent les **requêtes** ; *c'est la seule partie officiellement au programme en CPGE.*
- Administration et requêtes se font par un logiciel dit **SGBD**, utilisant le langage SQL (Submit Query Language).

- La création, le remplissage, la mise à jour, la sauvegarde etc. constituent l'**administration** de la base de donnée.
- Les mécanismes d'interrogation de cette base constituent les **requêtes** ; *c'est la seule partie officiellement au programme en CPGE.*
- Administration et requêtes se font par un logiciel dit **SGBD**, utilisant le langage SQL (Submit Query Language).
- Pour une utilisation plus conviviale, le SGBD est complété par un autre logiciel dit **client graphique**, qui permet de contrôler la syntaxe et fait apparaître les tables à l'écran. C'est un environnement de développement intégré (IDE), comme Pyzo l'est pour Python.

- Une table est une partie R d'un ensemble produit $D_1 \times \ldots \times D_n$. Les D_i sont les **domaines** : ensembles d'entiers ou flottants ou chaines de caractères ou dates etc.

- Une table est une partie R d'un ensemble produit $D_1 \times \ldots \times D_n$. Les D_i sont les **domaines** : ensembles d'entiers ou flottants ou chaines de caractères ou dates etc.
- Les lignes, à partir de la deuxième, sont les éléments de R, dits **tuples**. Ils doivent être tous distincts.

- Une table est une partie R d'un ensemble produit $D_1 \times \ldots \times D_n$. Les D_i sont les **domaines** : ensembles d'entiers ou flottants ou chaines de caractères ou dates etc.
- Les lignes, à partir de la deuxième, sont les éléments de R, dits **tuples**. Ils doivent être tous distincts.
- Les colonnes de la table sont les projections de R sur les domaines D_i . Leurs noms s'appellent **attributs** de la relation, et sont notés A_i .

- Une table est une partie R d'un ensemble produit $D_1 \times \ldots \times D_n$. Les D_i sont les **domaines** : ensembles d'entiers ou flottants ou chaines de caractères ou dates etc.
- Les lignes, à partir de la deuxième, sont les éléments de R, dits **tuples**. Ils doivent être tous distincts.
- Les colonnes de la table sont les projections de R sur les domaines D_i . Leurs noms s'appellent **attributs** de la relation, et sont notés A_i .
- La première ligne de la table est le **schéma** de R. Il se note $R(A_1 : D_1, \ldots, A_n : D_n)$.

- Une table est une partie R d'un ensemble produit $D_1 \times \ldots \times D_n$. Les D_i sont les **domaines** : ensembles d'entiers ou flottants ou chaines de caractères ou dates etc.
- Les lignes, à partir de la deuxième, sont les éléments de R, dits **tuples**. Ils doivent être tous distincts.
- Les colonnes de la table sont les projections de R sur les domaines D_i . Leurs noms s'appellent **attributs** de la relation, et sont notés A_i .
- La première ligne de la table est le **schéma** de R. Il se note $R(A_1 : D_1, \ldots, A_n : D_n)$.
- La base complète est un ensemble de tables spécifiées par leur schéma.

Exemple : table Summits de IHMC

Summits (N : int, Alt : int, UName : varchar(20), Range : varchar(30), C1 : varchar(2), C2 : varchar(2), Coord : varchar(21))

Exemple : table Summits de IHMC

Summits (N : int, Alt : int, UName : varchar(20), Range : varchar(30), C1 : varchar(2), C2 : varchar(2), Coord : varchar(21))

N	Alt	UName	Range	C1	C2	Coord
1	8850	Everest	Himalaya	NP	CN	N.27.59.17 E.86.55.31
2	8612	K2	Karakoram	PK	CN	N.35.52.57 E.76.30.48
3	8586	Kangch	Himalaya	NP	IN	N.27.42.09 E.88.08.49
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Exemple : table Summits de IHMC

Summits (N : int, Alt : int, UName : varchar(20), Range : varchar(30), C1 : varchar(2), C2 : varchar(2), Coord : varchar(21))

N	Alt	UName	Range	C1	C2	Coord
1	8850	Everest	Himalaya	NP	CN	N.27.59.17 E.86.55.31
2	8612	K2	Karakoram	PK	CN	N.35.52.57 E.76.30.48
3	8586	Kangch	Himalaya	NP	IN	N.27.42.09 E.88.08.49
⋮	⋮	⋮	⋮	⋮	⋮	⋮

En bleu : le schéma. En vert : un tuple. En rose : un attribut.

1. Une **clé** de la table R est un sous-ensemble minimal d'attributs permettant de caractériser chaque tuple de R. Parmi les différentes clés possibles, on en choisit une dite **clé primaire**.

1. Une **clé** de la table R est un sous-ensemble minimal d'attributs permettant de caractériser chaque tuple de R. Parmi les différentes clés possibles, on en choisit une dite **clé primaire**.

1. N, Coord sont des clés possibles pour Summits. UName n'en est pas une car il peut y avoir des sommets homonymes.

1. Une **clé** de la table R est un sous-ensemble minimal d'attributs permettant de caractériser chaque tuple de R. Parmi les différentes clés possibles, on en choisit une dite **clé primaire**.
2. Un **identifiant** est un entier qu'on introduit artificiellement pour servir de clé primaire.

1. N, Coord sont des clés possibles pour Summits. UName n'en est pas une car il peut y avoir des sommets homonymes.

1. Une **clé** de la table R est un sous-ensemble minimal d'attributs permettant de caractériser chaque tuple de R. Parmi les différentes clés possibles, on en choisit une dite **clé primaire**.
2. Un **identifiant** est un entier qu'on introduit artificiellement pour servir de clé primaire.

1. N, Coord sont des clés possibles pour Summits. UName n'en est pas une car il peut y avoir des sommets homonymes.
2. Id est un identifiant pour Climbers.

1. Une **clé** de la table R est un sous-ensemble minimal d'attributs permettant de caractériser chaque tuple de R. Parmi les différentes clés possibles, on en choisit une dite **clé primaire**.
2. Un **identifiant** est un entier qu'on introduit artificiellement pour servir de clé primaire.
3. Lorsqu'un attribut est une clé primaire d'une autre table, il prend le nom de **clé étrangère**.

1. N, Coord sont des clés possibles pour Summits. UName n'en est pas une car il peut y avoir des sommets homonymes.
2. Id est un identifiant pour Climbers.

1. Une **clé** de la table R est un sous-ensemble minimal d'attributs permettant de caractériser chaque tuple de R. Parmi les différentes clés possibles, on en choisit une dite **clé primaire**.
2. Un **identifiant** est un entier qu'on introduit artificiellement pour servir de clé primaire.
3. Lorsqu'un attribut est une clé primaire d'une autre table, il prend le nom de **clé étrangère**.

1. N, Coord sont des clés possibles pour Summits. UName n'en est pas une car il peut y avoir des sommets homonymes.
2. Id est un identifiant pour Climbers.
3. N est une clé étrangère de FirstAscent vers Summits.

Création d'une table

```
CREATE TABLE Ascents (
        date DATE,
        C1 INT,
        C2 INT,
        N INT,
        alt INT,
        side VARCHAR(2),
        route VARCHAR(20),
        grade VARCHAR(4),
PRIMARY KEY(date,C1) ) ;
```

Insertion ou suppression d'un tuple

```
INSERT INTO Ascents
VALUES ( '1996-07-23',
         4912,
         4913,
         37,
         7708,
         'N Tirich',
         'NW spur',
         'D4+' );
```

Insertion ou suppression d'un tuple

```
INSERT INTO Ascents
VALUES ( '1996-07-23',
         4912,
         4913,
         37,
         7708,
         'N Tirich',
         'NW spur',
         'D4+' );
```

```
DELETE FROM Ascents
WHERE C1=5785 AND C2 IS NULL;
```

Modification d'un tuple, destruction d'une table

```
UPDATE Climbers
SET death='2018-01-20'
WHERE Id=2312 ;
```

Modification d'un tuple, destruction d'une table

```
UPDATE Climbers
SET death='2018-01-20'
WHERE Id=2312 ;
```

```
DROP TABLE Ascents ;
```

La première question à se poser quand on aborde un problème de requête est : dans quelle(s) table(s) se trouvent les données ?

La première question à se poser quand on aborde un problème de requête est : dans quelle(s) table(s) se trouvent les données ?

Dans cette section on commence par le cas simple : toutes les données se trouvent dans une seule table T. La réponse à la requête, si elle est syntaxiquement correcte, sera alors une sous-table de T.

La première question à se poser quand on aborde un problème de requête est : dans quelle(s) table(s) se trouvent les données ?

Dans cette section on commence par le cas simple : toutes les données se trouvent dans une seule table T. La réponse à la requête, si elle est syntaxiquement correcte, sera alors une sous-table de T.

La syntaxe de base est :

```
SELECT attributs - - obligatoire - -
FROM table - - obligatoire - -
(WHERE condition)
(ORDER BY attribut) ; - - le point virgule final est obligatoire - -
```

Projection

On ne garde que certaines colonnes de la table, précisées par leur nom d'attribut.
Les tuples sont distincts, mais leurs projections ne le sont pas forcément. Pour qu'elles le soient, on doit utiliser DISTINCT.

Projection

On ne garde que certaines colonnes de la table, précisées par leur nom d'attribut.
Les tuples sont distincts, mais leurs projections ne le sont pas forcément. Pour qu'elles le soient, on doit utiliser DISTINCT.

```
SELECT DISTINCT firstN, secondN FROM Climbers;
```

Cette requête renvoie la table de tous les alpinistes répertoriés dans IHMC, sans homonymes : il y a 18 Joe Brown, un seul apparaîtra.

Projection

On ne garde que certaines colonnes de la table, précisées par leur nom d'attribut.
Les tuples sont distincts, mais leurs projections ne le sont pas forcément. Pour qu'elles le soient, on doit utiliser DISTINCT.

```
SELECT DISTINCT firstN, secondN FROM Climbers;
```

Cette requête renvoie la table de tous les alpinistes répertoriés dans IHMC, sans homonymes : il y a 18 Joe Brown, un seul apparaîtra.

SELECT n'est pas une sélection mais une projection.

Sélection

On ne garde que certaines lignes de la table, celles qui vérifient un booléen à préciser.
On peut ensuite projeter pour ne garder que les attributs intéressants.

Sélection

On ne garde que certaines lignes de la table, celles qui vérifient un booléen à préciser.
On peut ensuite projeter pour ne garder que les attributs intéressants.

```
SELECT * - - on garde tous les attributs - -
FROM Summits
WHERE Alt>=8000;
```

Sélection

On ne garde que certaines lignes de la table, celles qui vérifient un booléen à préciser.
On peut ensuite projeter pour ne garder que les attributs intéressants.

```
SELECT * - - on garde tous les attributs - -
FROM Summits
WHERE Alt>=8000 ;
```

```
SELECT UName, Range - - projection - -
FROM Summits
WHERE Alt>=8000 ; - - sélection - -
```

Tri

Par défaut, les données sont triées par ordre croissant.
Pour les chaines de caractères, il s'agit de l'ordre lexicographique, les nombres précédent les lettres.

Tri

Par défaut, les données sont triées par ordre croissant.
Pour les chaines de caractères, il s'agit de l'ordre lexicographique, les nombres précédent les lettres.

```
SELECT N, UName, Range
FROM Summits
ORDER BY Coord ;
```

Tri

Par défaut, les données sont triées par ordre croissant.
Pour les chaines de caractères, il s'agit de l'ordre lexicographique, les nombres précédent les lettres.

```
SELECT N, UName, Range
FROM Summits
ORDER BY Coord ;
```

Cette requête va ordonner les sommets de IHMC en commençant par les plus proches de l'équateur dans l'hémisphère Nord (N<S) ; en cas d'égalité de latitude, le plus proche du méridien de Greenwich dans l'hémisphère Est apparaîtra en premier (E<W).

Pour trier par ordre décroissant, on doit ajouter DESC.

Pour trier par ordre décroissant, on doit ajouter DESC.

On peut ensuite sélectionner une tranche de résultats par :

- LIMIT a : résultats de rang $< a$, en commençant à 0 ; il y aura donc a lignes dans la table retournée ;
- LIMIT a,b : résultats de $a \leq$ rang $< b$: $b - a$ lignes.

Pour trier par ordre décroissant, on doit ajouter DESC.

On peut ensuite sélectionner une tranche de résultats par :

- LIMIT a : résultats de rang $< a$, en commençant à 0 ; il y aura donc a lignes dans la table retournée ;
- LIMIT a,b : résultats de $a \leq$ rang $< b$: $b - a$ lignes.

LIMIT 1 donne donc le min, DESC LIMIT 1 donne le max.

Pour trier par ordre décroissant, on doit ajouter DESC.

On peut ensuite sélectionner une tranche de résultats par :

- LIMIT a : résultats de rang $< a$, en commençant à 0 ; il y aura donc a lignes dans la table retournée ;
- LIMIT a,b : résultats de $a \leq$ rang $< b$: $b - a$ lignes.

LIMIT 1 donne donc le min, DESC LIMIT 1 donne le max.

ORDER BY ... (DESC) (LIMIT ...)
est obligatoirement la dernière instruction

Pour trier par ordre décroissant, on doit ajouter DESC.

On peut ensuite sélectionner une tranche de résultats par :

- LIMIT a : résultats de rang $< a$, en commençant à 0 ; il y aura donc a lignes dans la table retournée ;
- LIMIT a,b : résultats de $a \leq$ rang $< b$: $b - a$ lignes.

LIMIT 1 donne donc le min, DESC LIMIT 1 donne le max.

ORDER BY ... (DESC) (LIMIT ...)
est obligatoirement la dernière instruction

```
SELECT N, UName, Alt, Range
FROM Summits
ORDER BY Alt DESC LIMIT 10 ;
```

N	UName	Alt	Range
1	Everest	8850	Himalaya
2	K2	8612	Karakoram
3	Kangchenjunga	8586	Himalaya
4	Lhotse	8516	Himalaya
5	Yalung Kang	8505	Himalaya
6	Makalu	8485	Himalaya
7	Lotse Shar	8401	Himalaya
8	Cho Oyu	8188	Himalaya
9	Dhaulaghiri	8167	Himalaya
10	Manaslu	8163	Himalaya

Renommage d'un attribut

SELECT attribut AS nom FROM ... ;

Utile pour que la table retournée soit plus parlante.

Renommage d'un attribut

```
SELECT attribut AS nom FROM ... ;
```

Utile pour que la table retournée soit plus parlante.

```
SELECT firstN AS prénom, lastN AS nom, birth, death
FROM Climbers
WHERE prénom='Joe' AND nom='Brown' ;
```

Renommage d'une table

Utile quand on fait une jointure faisant intervenir 2 tables ayant des attributs de même nom. Il faut alors préfixer cet attribut pour éviter l'ambigüité : préfixer par une lettre est plus efficace.

Renommage d'une table

Utile quand on fait une jointure faisant intervenir 2 tables ayant des attributs de même nom. Il faut alors préfixer cet attribut pour éviter l'ambigüité : préfixer par une lettre est plus efficace.

Pas de AS : il suffit de faire suivre le nom de la table de son alias.

Renommage d'une table

Utile quand on fait une jointure faisant intervenir 2 tables ayant des attributs de même nom. Il faut alors préfixer cet attribut pour éviter l'ambigüité : préfixer par une lettre est plus efficace.

Pas de AS : il suffit de faire suivre le nom de la table de son alias.

```
SELECT S.C1 AS country
FROM Summits S JOIN FirstAscents ON ...
WHERE S.C1 IN {'PK','AF'};
```

Ici le renommage est nécessaire car C1 est présent dans les 2 tables, mais ne correspond pas du tout à la même chose : Country dans Summits, Climber dans FirstAscents.

Lorsque les informations dont on a besoin se trouvent dans 2 tables, il est nécessaire d'effectuer une **jointure**.

Lorsque les informations dont on a besoin se trouvent dans 2 tables, il est nécessaire d'effectuer une **jointure**.

- La jointure sans condition est le **produit cartésien** des 2 tables R et S, c'est à dire l'ensemble de tous les longs tuple du type (r, s) où r est un tuple de R et s un tuple de S.

Lorsque les informations dont on a besoin se trouvent dans 2 tables, il est nécessaire d'effectuer une **jointure**.

- La jointure sans condition est le **produit cartésien** des 2 tables R et S, c'est à dire l'ensemble de tous les longs tuple du type (r, s) où r est un tuple de R et s un tuple de S.
- La **jointure naturelle** consiste à identifier dans le produit cartésien les attributs ayant le même nom dans les 2 tables.

Lorsque les informations dont on a besoin se trouvent dans 2 tables, il est nécessaire d'effectuer une **jointure**.

- La jointure sans condition est le **produit cartésien** des 2 tables R et S, c'est à dire l'ensemble de tous les longs tuple du type (r, s) où r est un tuple de R et s un tuple de S.
- La **jointure naturelle** consiste à identifier dans le produit cartésien les attributs ayant le même nom dans les 2 tables.
- La **jointure conditionnelle** identifie dans le produit cartésien un (ou plusieurs) attribut(s) ayant des noms différents mais représentant la même chose.

Lorsque les informations dont on a besoin se trouvent dans 2 tables, il est nécessaire d'effectuer une **jointure**.

- La jointure sans condition est le **produit cartésien** des 2 tables R et S, c'est à dire l'ensemble de tous les longs tuple du type (r, s) où r est un tuple de R et s un tuple de S.
- La **jointure naturelle** consiste à identifier dans le produit cartésien les attributs ayant le même nom dans les 2 tables.
- La **jointure conditionnelle** identifie dans le produit cartésien un (ou plusieurs) attribut(s) ayant des noms différents mais représentant la même chose.

Jointure = sélection sur le produit cartésien

Produit cartésien

Table A :

UName	Alt	Range
Everest	8850	Himalaya
K2	8612	Karakoram

Table B :

Alt	Date	Route
8850	1953-05-29	SE ridge
8586	1955-05-25	NE ridge

Produit cartésien

Table A :

UName	Alt	Range
Everest	8850	Himalaya
K2	8612	Karakoram

Table B :

Alt	Date	Route
8850	1953-05-29	SE ridge
8586	1955-05-25	NE ridge

```
SELECT * FROM A,B ; - - ou A JOIN B - -
```

Produit cartésien

Table A :

UName	Alt	Range
Everest	8850	Himalaya
K2	8612	Karakoram

Table B :

Alt	Date	Route
8850	1953-05-29	SE ridge
8586	1955-05-25	NE ridge

```
SELECT * FROM A,B ; - - ou A JOIN B - -
```

UName	Alt	Range	Alt	Date	Route
Everest	8850	Himalaya	8850	1953-05-29	SE ridge
Everest	8850	Himalaya	8586	1955-05-25	NE ridge
K2	8612	Karakoram	8850	1953-05-29	SE ridge
K2	8612	Karakoram	8586	1955-05-25	NE ridge

Jointure naturelle

```
SELECT * FROM A,B WHERE A.Alt=B.Alt ;
```

UName	Alt	Range	Alt	Date	Route
Everest	8850	Himalaya	8850	1953-05-29	SE ridge

Jointure naturelle

```
SELECT * FROM A,B WHERE A.Alt=B.Alt ;
```

UName	Alt	Range	Alt	Date	Route
Everest	8850	Himalaya	8850	1953-05-29	SE ridge

```
SELECT * FROM A NATURAL JOIN B ;
```

UName	Alt	Range	Date	Route
Everest	8850	Himalaya	1953-05-29	SE ridge

Jointure conditionnelle

Syntaxe générale :

```
SELECT ... FROM R JOIN S ON R.att1=S.att2 WHERE ... ;
```

Préfixer les attributs n'est nécessaire que s'il y a un risque d'ambigüité.

Jointure conditionnelle

Syntaxe générale :

```
SELECT ... FROM R JOIN S ON R.att1=S.att2 WHERE ... ;
```

Préfixer les attributs n'est nécessaire que s'il y a un risque d'ambigüité.

Comment obtenir, pour chacun des 3 plus hauts sommets, son numéro, la date de sa première ascension, et les prénoms et noms des ascensionnistes ?

Jointure conditionnelle

Syntaxe générale :

```
SELECT ... FROM R JOIN S ON R.att1=S.att2 WHERE ... ;
```

Préfixer les attributs n'est nécessaire que s'il y a un risque d'ambigüité.

Comment obtenir, pour chacun des 3 plus hauts sommets, son numéro, la date de sa première ascension, et les prénoms et noms des ascensionnistes ?

```
SELECT N, Date, firstN, lastN
FROM Climbers JOIN FirstAscent ON Id=C1 OR Id=C2
WHERE N<=3 ;
```

On peut faire la jointure de 2 exemplaires d'une même table, ce qui permet de voir si un même attribut apparait plusieurs fois.

Comment savoir quels alpinistes ont fait la première ascension de plusieurs sommets ?

On peut faire la jointure de 2 exemplaires d'une même table, ce qui permet de voir si un même attribut apparait plusieurs fois.

Comment savoir quels alpinistes ont fait la première ascension de plusieurs sommets ?

```
SELECT F.C1, F.C2, F.N
FROM FirstAscent F JOIN FirstAscent G
ON (F.C1=G.C1 OR F.C1=G.C2 OR F.C2=G.C1 OR F.C2=G.C2)
WHERE F.N !=G.N ;
```

Jointure de 3 tables

On fait les jointures successivement :

```
SELECT ...
FROM (R JOIN S ON R.att1=S.att2) T - - 1è jointure - -
JOIN U ON T.att3=U.att4 - - 2è jointure - -
WHERE ... ;
```

En rouge : on nomme la table intermédiaire, pour pouvoir y référer.

Jointure de 3 tables

On fait les jointures successivement :

```
SELECT ...
FROM (R JOIN S ON R.att1=S.att2) T - - 1è jointure - -
JOIN U ON T.att3=U.att4 - - 2è jointure - -
WHERE ... ;
```

En rouge : on nomme la table intermédiaire, pour pouvoir y référer.

Comment obtenir, pour chacun des 3 plus hauts sommets, son nom, la date de sa première ascension, et les prénoms et noms des ascensionnistes ?

Les informations se trouvent dans les 3 tables :

- nom du sommet dans Summits ;
- date de première ascension dans FirstAscent ;
- prénoms et noms dans Climbers.

Attention : il existe dans Summits et dans FirstAscent des attributs C1 et C2, qui n'ont rien à voir. On ne garde donc dans Summits que les attributs utiles.

Les informations se trouvent dans les 3 tables :

- nom du sommet dans Summits ;
- date de première ascension dans FirstAscent ;
- prénoms et noms dans Climbers.

Attention : il existe dans Summits et dans FirstAscent des attributs C1 et C2, qui n'ont rien à voir. On ne garde donc dans Summits que les attributs utiles.

```
SELECT UName, Date, firstN AS prénom, lastN AS nom
FROM ( SELECT N, UName FROM Summits)
NATURAL JOIN FirstAscent
JOIN Climbers ON C1=Id OR C2=Id
WHERE N<=3 ;
```

L'**agrégation** consiste à regrouper tous les tuples d'une table ayant même valeur pour un ou plusieurs attributs. Dans le résultat n'apparaîtra alors que le dernier de ces tuples, il faut imaginer que tous les autres sont cachés derrière.

L'**agrégation** consiste à regrouper tous les tuples d'une table ayant même valeur pour un ou plusieurs attributs. Dans le résultat n'apparaîtra alors que le dernier de ces tuples, il faut imaginer que tous les autres sont cachés derrière.

On a généralement besoin de sélectionner certains tuples de la table :

- si on sélectionne avant de regrouper, on utilise WHERE ;
- si on sélectionne après le regroupement, on utilise HAVING.

Syntaxe de l'agrégation

SELECT attributs
FROM table
(WHERE condition préliminaire)
GROUP BY att1,..,attn
(HAVING condition finale) ;

Syntaxe de l'agrégation

```
SELECT attributs
FROM table
(WHERE condition préliminaire)
GROUP BY att1,..,attn
(HAVING condition finale) ;
```

- Les 3 premières lignes créent une table ;

Syntaxe de l'agrégation

```
SELECT attributs
FROM table
(WHERE condition préliminaire)
GROUP BY att1,..,attn
(HAVING condition finale) ;
```

- Les 3 premières lignes créent une table ;
- la suivante effectue l'agrégation dans cette table : regroupement des tuples ayant même valeur sur *chacune* des colonnes att1,...,attn ;

Syntaxe de l'agrégation

```
SELECT attributs
FROM table
(WHERE condition préliminaire)
GROUP BY att1,..,attn
(HAVING condition finale) ;
```

- Les 3 premières lignes créent une table ;
- la suivante effectue l'agrégation dans cette table : regroupement des tuples ayant même valeur sur *chacune* des colonnes att1,...,attn ;
- la dernière ligne ne garde, dans cette table regroupée, que les lignes vérifiant la condition finale.

Exemple stupide

Comment obtenir la liste des ascensionnistes anglais de IHMC, sans homonymes ?

Exemple stupide

Comment obtenir la liste des ascensionnistes anglais de IHMC, sans homonymes ?

Bonne solution : utiliser DISTINCT.

Exemple stupide

Comment obtenir la liste des ascensionnistes anglais de IHMC, sans homonymes ?

Bonne solution : utiliser DISTINCT.
Autres :

```
SELECT firstN, lastN FROM Climbers
WHERE Nation='GB'
GROUP BY firstN, lastN ;
```

```
SELECT firstN, lastN FROM Climbers
GROUP BY firstN, lastN
HAVING Nation='GB' ;
```

Fonctions d'agrégation

En pratique, l'agrégation n'a d'intérêt que lorsqu'on a besoin d'appliquer une fonction à chaque groupe séparément. Les fonctions compatibles avec l'agrégation sont : COUNT, MIN, MAX, SUM, AVG (moyenne).

Fonctions d'agrégation

En pratique, l'agrégation n'a d'intérêt que lorsqu'on a besoin d'appliquer une fonction à chaque groupe séparément. Les fonctions compatibles avec l'agrégation sont : COUNT, MIN, MAX, SUM, AVG (moyenne).

Chaque fonction prend en argument un attribut, et retourne un nouvel attribut. Ces fonctions sont bien sûr utilisables sans agrégation, elles opèrent alors sur la totalité de la table.

Fonctions d'agrégation

En pratique, l'agrégation n'a d'intérêt que lorsqu'on a besoin d'appliquer une fonction à chaque groupe séparément. Les fonctions compatibles avec l'agrégation sont : COUNT, MIN, MAX, SUM, AVG (moyenne).

Chaque fonction prend en argument un attribut, et retourne un nouvel attribut. Ces fonctions sont bien sûr utilisables sans agrégation, elles opèrent alors sur la totalité de la table.

```
SELECT ..., f(att1) AS nom, ...
FROM table GROUP BY ...
HAVING condition sur g(att2) ;
```

Exemple moins stupide

Table T :

Alt	Date	firstN	lastN	age
8850	1953-05-29	Tenzing	Norgay	39
8850	1953-05-29	Edmund	Hillary	33
8612	1954-07-31	Achille	Compagnoni	34
8612	1954-07-31	Lino	Lacedelli	33
8612	1977-08-09	Ichiro	Yoshizawa	29
8612	1977-08-10	Ashraf	Aman	31

Exemple moins stupide

Table T :

Alt	Date	firstN	lastN	age
8850	1953-05-29	Tenzing	Norgay	39
8850	1953-05-29	Edmund	Hillary	33
8612	1954-07-31	Achille	Compagnoni	34
8612	1954-07-31	Lino	Lacedelli	33
8612	1977-08-09	Ichiro	Yoshizawa	29
8612	1977-08-10	Ashraf	Aman	31

```
SELECT * FROM T GROUP BY Alt ;
```

Alt	Date	firstN	lastN	age
8850	1953-05-29	Edmund	Hillary	33
8612	1954-07-31	Ashraf	Aman	31

```
SELECT Alt, MIN(Date), MIN(firstN), MIN(lastN), MIN(age)
FROM T GROUP BY Alt ;
```

```
SELECT Alt, MIN(Date), MIN(firstN), MIN(lastN), MIN(age)
FROM T GROUP BY Alt ;
```

Alt	min(Date)	min(firstN)	min(lastN)	min(age)
8850	1953-05-29	Edmund	Hillary	33
8612	1954-07-31	Achille	Aman	29

```
SELECT Alt, MIN(Date), MIN(firstN), MIN(lastN), MIN(age)
FROM T GROUP BY Alt ;
```

Alt	min(Date)	min(firstN)	min(lastN)	min(age)
8850	1953-05-29	Edmund	Hillary	33
8612	1954-07-31	Achille	Aman	29

```
SELECT Alt, AVG(age) AS AgeMoyen FROM T GROUP BY Alt
HAVING AVG(age)<35 ;
```

```
SELECT Alt, MIN(Date), MIN(firstN), MIN(lastN), MIN(age)
FROM T GROUP BY Alt ;
```

Alt	min(Date)	min(firstN)	min(lastN)	min(age)
8850	1953-05-29	Edmund	Hillary	33
8612	1954-07-31	Achille	Aman	29

```
SELECT Alt, AVG(age) AS AgeMoyen FROM T GROUP BY Alt
HAVING AVG(age)<35 ;
```

Alt	AgeMoyen
8612	34,25

Exemple pas du tout stupide

Comment obtenir, pour chacun des 10 plus hauts sommets, son nom, le nombre d'alpinistes français l'ayant gravi, l'âge du plus jeune de ceux-ci ?

Exemple pas du tout stupide

Comment obtenir, pour chacun des 10 plus hauts sommets, son nom, le nombre d'alpinistes français l'ayant gravi, l'âge du plus jeune de ceux-ci ?

```
SELECT UName, COUNT(N) as NFC, Now()-MAX(birth) AS
AgeMini
FROM ((SELECT N, UName FROM Summits WHERE N<=10)
NATURAL JOIN Ascents)
JOIN (SELECT * FROM Climbers WHERE nation='FR')
ON C1=Id OR C2=Id
GROUP BY N ;
```