Annales Mines-Ponts SQL

2015–2026

Livret d'entraînement — extraits verbatim des sujets

MP*1 MP*2 MP

Ce livret appartient à :

Comment travailler avec ce livret

Chaque sujet est un extrait **mot pour mot** d'une épreuve d'ITC du concours Mines-Ponts : le contexte est sur la page de gauche, les questions en face.

Travaillez sur chaque annale **au stylo, sans document, en une seule fois** pour vous entraîner le plus possible aux conditions réelles du concours.

Il est conseillé d'inclure un travail régulier de ces annales dans votre planning (par exemple une par semaine à une date fixée). Chaque annale est courte (≈ 15 min) et il est essentiel de travailler régulièrement le SQL.

Pour un corrigé, cf. le livret de corrigés. Vous pouvez aussi tester vos requêtes sur le site de la classe. Un mémo de la syntaxe SQL figure en fin de livret.

À la page suivante, la liste des sujets, avec sa case « Fait », permet de suivre ceux que vous avez déjà traités.

Liste des sujets

Suivi personnel

Vous pouvez ici noter ou planifier les sujets que vous souhaitez travailler.

Date	Sujet

CCMP 2015 — Tests de validation d'une imprimante

Extrait verbatim de : CCMP 2015 — III. Base de données

Contexte et schéma de la base

Une représentation simplifiée de deux tables de la base de données qu'on souhaite utiliser est donnée ci-dessous :

Table `testfin` :

nSerie	dateTest	...	Imoy	Iec	...	fichierMes
230-588ZX2547	2012-04-22 14-25-45	…	0.45	0.11	…	mesure31025.csv
230-588ZX2548	2012-04-22 14-26-57	…	0.43	0.12	…	mesure41026.csv
…	…	…	…	…	…	…

Table `production` :

Num	nSerie	dateProd	type
20	230-588ZX2547	2012-04-22 15-52-12	JETDESK-1050
21	230-588ZX2549	2012-04-22 15-53-24	JETDESK-3050
…	…	…	…

Après son assemblage et avant les différents tests de validation, un numéro de série unique est attribué à chaque imprimante. A la fin des tests de chaque imprimante, les résultats d'analyse ainsi que le fichier contenant l'ensemble des mesures réalisées sur l'imprimante sont rangés dans la table `testfin`. Lorsqu'une imprimante satisfait les critères de validation, elle est enregistrée dans la table `production` avec son numero de série, la date et l'heure de sortie de production ainsi que son type.

Questions

Q8. Rédiger une requête SQL permettant d'obtenir les numéros de série des imprimantes ayant une valeur de Imoy comprise strictement entre deux bornes Imin et Imax.

Q9. Rédiger une requête SQL permettant d'obtenir les numéros de série, la valeur de l'écart type et le fichier de mesures des imprimantes ayant une valeur de Iec strictement inférieure à la valeur moyenne de la colonne Iec.

Q10. Rédiger une requête SQL qui permettra d'extraire à partir de la table `testfin` le numéro de série et le fichier de mesures correspondant aux imprimantes qui n'ont pas été validées en sortie de production.

CCMP 2016 — Modélisation de la propagation d'une épidémie

Extrait verbatim de : CCMP 2016 — Partie I. Tri et bases de données

Contexte et schéma de la base

Pour suivre la propagation des épidémies, de nombreuses données sont recueillies par les institutions internationales comme l'O.M.S. Par exemple, pour le paludisme, on dispose de deux tables :

- la table `palu` recense le nombre de nouveaux cas confirmés et le nombre de décès liés au paludisme ; certaines lignes de cette table sont données en exemple (on précise que `iso` est un identifiant unique pour chaque pays) :

nom	iso	annee	cas	deces
Bresil	BR	2009	309 316	85
Bresil	BR	2010	334 667	76
Kenya	KE	2010	898 531	26 017
Mali	ML	2011	307 035	2 128
Ouganda	UG	2010	1 581 160	8 431
...	...	...	...	...

- la table `demographie` recense la population totale de chaque pays ; certaines lignes de cette table sont données en exemple :

pays	periode	pop
BR	2009	193 020 000
BR	2010	194 946 000
KE	2010	40 909 000
ML	2011	14 417 000
UG	2010	33 987 000
...	...	...

Questions

Q5. Au vu des données présentées dans la table `palu`, parmi les attributs `nom`, `iso` et `annee`, quels attributs peuvent servir de clé primaire ? Un couple d'attributs pourrait-il servir de clé primaire ? (on considère qu'une clé primaire peut posséder plusieurs attributs). Si oui, en préciser un.

Q6. Écrire une requête en langage SQL qui récupère depuis la table `palu` toutes les données de l'année 2010 qui correspondent à des pays où le nombre de décès dus au paludisme est supérieur ou égal à 1 000.

On appelle *taux d'incidence d'une épidémie* le rapport du nombre de nouveaux cas pendant une période donnée sur la taille de la population-cible pendant la même période. Il s'exprime généralement en « nombre de nouveaux cas pour 100 000 personnes par année ». Il s'agit d'un des critères les plus importants pour évaluer la fréquence et la vitesse d'apparition d'une épidémie.

Q7. Écrire une requête en langage SQL qui détermine le taux d'incidence du paludisme en 2011 pour les différents pays de la table `palu`.

Q8. Écrire une requête en langage SQL permettant de déterminer le nom du pays ayant eu le deuxième plus grand nombre de nouveaux cas de paludisme en 2010 (on pourra supposer qu'il n'y a pas de pays *ex æquo* pour les nombres de cas).

CCMP 2017 — Étude de trafic routier

Extrait verbatim de : CCMP 2017 — Partie VI. Base de données

Contexte et schéma de la base

On modélise ici un réseau routier par un ensemble de croisements et de voies reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés.

La base de données du réseau routier est constituée des relations suivantes :

- `Croisement(id, longitude, latitude)`
- `Voie(id, longueur, id_croisement_debut, id_croisement_fin)`

Dans la suite on considère `c` l'identifiant (`id`) d'un croisement donné.

Questions

Q26. Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant `c`.

Q27. Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement `c`.

Q28. Que renvoie la requête SQL suivante ?

```sql
SELECT V2.id_croisement_fin
FROM   Voie as V1
JOIN   Voie as V2
ON     V1.id_croisement_fin = V2.id_croisement_debut
WHERE  V1.id_croisement_debut = c
```

CCMP 2018 — Mesures de houle

Extrait verbatim de : CCMP 2018 — Partie IV. Base de données relationnelle

Contexte et schéma de la base

On dispose d'une base de données relationnelle `Vagues`.

La première table est `Bouee`. On se limite aux attributs suivants : le numéro d'identification `idBouee`, le nom du site `nomSite`, le nom de la mer ou de l'océan `localisation`, le type du capteur `typeCapteur` et la fréquence d'échantillonnage `frequence`.

Table `Bouee` :

idBouee	nomSite	localisation	typeCapteur	frequence
831	Porquerolles	Mediterranee	Datawell non directionnelle	2.00
291	Les pierres noires	Mer d'iroise	Datawell directionnelle	1.28
…	…	…	…	…

La seconde table est `Campagne`. On se limite aux attributs suivants : le numéro d'identification `idCampagne`, le numéro d'identification de la bouée `idBouee`, la date de début `debutCampagne` et la date de fin `finCampagne`.

Table `Campagne` :

idCampagne	idBouee	debutCampagne	finCampagne
08301	831	01/01/2010 00h00	15/01/2010 00h00
02911	291	15/10/2005 18h30	18/10/2005 08h00
…	…	…	…

La troisième table est `Tempete`. Les informations fournies relatives à un événement « tempête » sont les suivantes :

- date de début et fin de tempête,
- évolution des paramètres $H_{1/3}$ et $H_{\max}$ en fonction du temps,
- le détail de certains paramètres non définis ici, obtenus au pic de tempête.

On se limite aux attributs suivants : le numéro d'identification de la tempête `idTempete`, le numéro d'identification de la bouée `idBouee`, la date de début `debutTempete`, la date de fin `finTempete`, la valeur maximale de hauteur de vague `Hmax`.

Table `Tempete` :

idTempete	idBouee	debutTempete	finTempete	Hmax
083010	831	07/01/2010 20h00	09/01/2010 15h30	5.3
029012	291	16/10/2005 08h30	18/10/2005 09h00	8.5
…	…	…	…	…

Le schéma de la base de données est donc : `Vagues` $= \{$`Bouee`, `Campagne`, `Tempete`$\}$.

Questions

Q19. Formuler les requêtes SQL permettant de répondre aux questions suivantes :

- « Quels sont le numéro d'identification et le nom de site des bouées localisées en Méditerranée ? »
- « Quel est le numéro d'identification des bouées où il n'y a pas eu de tempêtes ? »
- « Pour chaque site, quelle est la hauteur maximale enregistrée lors d'une tempête ? »

(a)

(b)

(c)

CCMP 2019 — Autour des nombres premiers

Extrait verbatim de : CCMP 2019 — Partie IV. Analyse de performance de code

Contexte et schéma de la base

Au cours du développement des fonctions nécessaires à la manipulation des nombres premiers on s'aperçoit que le choix des algorithmes pour évaluer chaque fonction est primordial pour garantir des performances acceptables. On souhaite donc mener des tests à grande échelle pour évaluer les performances réelles du code qui a été développé. Pour ce faire on effectue un grand nombre de tests sur une multitude d'ordinateurs. Les données sont ensuite centralisées dans une base de données composée de deux tables.

La première table est `ordinateurs` et permet de stocker des informations sur les ordinateurs utilisés pour les tests. Ses attributs sont :

- `nom` TEXT, clé primaire, le nom de l'ordinateur.
- `gflops` INTEGER la puissance de l'ordinateur en milliards d'opérations flottantes par seconde.
- `ram` INTEGER la quantité de mémoire vive de l'ordinateur en Go.

Exemple du contenu de cette table :

nom	gflops	ram
nyarlathotep114	69	32
nyarlathotep119	137	32
…	…	…
shubniggurath42	133	16
azathoth137	85	8

La seconde table est `fonctions` et stocke les informations sur les tests effectués pour différentes fonctions en cours de développement. Ses attributs sont :

- `id` INTEGER l'identifiant du test effectué.
- `nom` TEXT le nom de la fonction testée (par exemple li, Ei, etc).
- `algorithme` TEXT le nom de l'algorithme qui permet le calcul de la fonction testée (par exemple BBS si on teste une fonction de génération de nombres aléatoires).
- `teste_sur` TEXT le nom du PC sur lequel le test a été effectué.
- `temps_exec` INTEGER le temps d'exécution du test en millisecondes.

Exemple du contenu de cette table :

id	nom	algorithme	teste_sur	temps_exec
1	li	rectangles	nyarlathotep165	2638
2	li	rectangles	shubniggurath28	736
3	li	trapezes	nyarlathotep165	4842
…	…	…	…	…
2154	Ei	puiseux	nyarlathotep145	2766
2155	aleatoire	BBS	azathoth145	524

Questions

Q25. Expliquer pourquoi il n'est pas possible d'utiliser l'attribut `nom` comme clé primaire de la table `fonctions`.

Q26. Écrire des requêtes SQL permettant de :

1. Connaître le nombre d'ordinateurs disponibles et leur quantité moyenne de mémoire vive.
2. Extraire les noms des PC sur lesquels l'algorithme `rectangles` n'a pas été testé pour la fonction nommée `li`.
3. Pour la fonction nommée `Ei`, trier les résultats des tests du plus lent au plus rapide. Pour chaque test retenir le nom de l'algorithme utilisé, le nom du pc sur lequel il a été effectué et la puissance du PC.

(1)

(2)

(3)

CCMP 2020 — Images de vagues et de structures

Extrait verbatim de : CCMP 2020 — Partie I, § I.a Chargement d'un modèle 3D à partir d'une base de données

Contexte et schéma de la base

On souhaite importer, dans Python, le modèle d'un bateau à moteur, élaboré préalablement. Ce modèle définit plusieurs maillages élémentaires (coque, bouée, échelle, moteur, etc.). Le fichier contenant ce modèle est une base de données relationnelle. Son schéma détaillé ci-dessous est illustré en figure 3 :

- relation `maillages_bateau` : ensemble des maillages. Un maillage possède un identifiant (entier) et un nom (chaîne de caractères) :

`maillages_bateau(id,nom)`

- relation `faces` : ensemble des facettes du modèle. Chaque facette est définie par un numéro unique, l'identifiant du maillage auquel elle appartient ainsi que les identifiants des sommets qui la composent. Tous sont des entiers.

`faces(numero,maillage,s1,s2,s3)`

- relation `sommets` : liste des sommets du modèle. Chaque sommet est défini par un identifiant (entier) et ses coordonnées dans l'espace par rapport au repère principal de la scène (flottant) :

`sommets(id,x,y,z)`

Table `maillages_bateau` :

id	nom
1	coque
2	bouée
3	échelle
4	moteur
…	…

Table `faces` :

numero	maillage	s1	s2	s3
1	3	1	2	3
2	3	2	4	3
3	2	3	12	5
…	…	…	…	…

Table `sommets` :

id	x	y	z
1	0.0	0.0	0.0
2	1.0	0.0	0.0
3	0.0	1.0	0.0
…	…	…	…

Figure 3 – Exemples des relations de la base de données.

Questions

Q1. Proposez une requête SQL permettant de compter le nombre de maillages que contient le modèle du bateau.

Q2. Proposez une requête SQL permettant de récupérer la liste des numéros des facettes (`numero`) du maillage nommé « gouvernail ».

Q3. Expliquez ce que renvoie la requête SQL suivante :

```
SELECT (MAX(x)-MIN(x))
FROM sommets AS s JOIN faces AS f JOIN  maillages_bateau AS m
ON (s.id=f.s1 OR s.id=f.s2 OR s.id=f.s3) AND f.maillage=m.id
WHERE m.nom="coque"
```

CCMP 2021 — Marchons, marchons, marchons...

Extrait verbatim de : CCMP 2021 — Partie I. Randonnée

Contexte et schéma de la base

Lors de la préparation d'une randonnée, une accompagnatrice doit prendre en compte les exigences des participants. Elle dispose d'informations rassemblées dans deux tables d'une base de données :

- la table `Rando` décrit les randonnées possibles – la clef primaire entière `rid`, son nom, le niveau de difficulté du parcours (entier entre 1 et 5), le dénivelé (en mètres), la durée moyenne (en minutes) :

rid	rnom	diff	deniv	duree
1	La belle des champs	1	20	30
2	Lac de Castellane	4	650	150
3	Le tour du mont	2	200	120
4	Les crêtes de la mort	5	1200	360
5	Yukon Ho !	3	700	210
…	…	…	…	…

- la table `Participant` décrit les randonneurs – la clef primaire entière `pid`, le nom du randonneur, son année de naissance, le niveau de difficulté maximum de ses randonnées :

pid	pnom	ne	diff_max
1	Calvin	2014	2
2	Hobbes	2015	2
3	Susie	2014	2
4	Rosalyn	2001	4
…	…	…	…

Questions

Donner une requête SQL sur cette base pour :

Q1. Compter le nombre de participants nés entre 1999 et 2003 inclus.

Q2. Calculer la durée moyenne des randonnées pour chaque niveau de difficulté.

Q3. Extraire le nom des participants pour lesquels la randonnée n°42 est trop difficile.

Q4. Extraire les clés primaires des randonnées qui ont un ou des homonymes (nom identique et clé primaire distincte), sans redondance.

CCMP 2022 — Modélisation numérique d'un matériau magnétique

Extrait verbatim de : CCMP 2022 — Partie II : Recherche dans une base de données de matériaux magnétiques

Contexte et schéma de la base

Il existe des bases de données contenant les propriétés de nombreux matériaux, dont des propriétés magnétiques. Dans cette partie, on donne un modèle simplifié d'une telle base, et on souhaite effectuer quelques requêtes sur celle-ci.

La base de données possède la structure suivante :

- La table `materiaux` contient un champ `id_materiau`, clé primaire de la table de valeur entière, un champ `nom` de type chaîne de caractères pour le nom du matériau et un champ `t_curie` de valeur entière pour la température de Curie du matériau en kelvin.

id_materiau	nom	t_curie
4534	cobalt	1 388
1254	dioxyde de chrome	386
8713	nickel	627
8284	YIG	560
…	…	…

- La table `fournisseurs`, contenant un champ `id_fournisseur`, clé primaire de type entier qui précise le code de chaque fournisseur, et un champ `nom_fournisseur` de type chaîne de caractères pour le nom du fournisseur.

id_fournisseur	nom_fournisseur
145	Worldwide Materials
13	Materials Company
…	…

- La table `prix` qui contient un champ `id_prix`, clef primaire de type entier, un champ `id_mat` dont les valeurs sont incluses dans l'ensemble des valeurs de la clé `id_materiau` de la table `materiaux`, un champ `id_four` dont les valeurs sont incluses dans l'ensemble des valeurs de la clé `id_fournisseur` de la table `fournisseurs`, et un champ `prix_kg` de type flottant qui précise le prix au kg que ce fournisseur propose pour ce matériau, en euros. Un fournisseur qui ne propose pas un matériau donné n'a pas d'entrée correspondante dans cette table.

id_prix	id_mat	id_four	prix_kg
1	4567	145	50.40
2	8671	13	1357.30
3	1763	145	52.75
…	…	…	…

Les requêtes demandées dans cette partie sont à écrire en langage SQL.

Questions

Q6. Écrire une requête permettant d'obtenir le nom de tous les matériaux qui ont une température de Curie strictement inférieure à 500 kelvins.

Un client potentiel souhaite acheter 4,5 kilogrammes de nickel (d'identifiant `8713`, que l'on pourra utiliser directement dans les requêtes) et sélectionner le fournisseur le moins cher.

Q7. Écrire une requête permettant d'obtenir les noms de tous les fournisseurs proposant du nickel et le prix proposé par chacun pour 4,5 kilogrammes de nickel.

Q8. Modifier ou compléter la requête précédente afin d'obtenir le nom du fournisseur de nickel le moins cher ainsi que le prix à payer chez ce fournisseur pour ces 4,5 kilogrammes de nickel. En cas d'égalité du prix optimal entre plusieurs fournisseurs, on obtiendra les noms de tous les fournisseurs possibles.

Q9. Écrire une requête permettant d'obtenir le nom de tous les matériaux et le prix moyen pour un kilogramme de chacun de ces matériaux (la moyenne étant calculée pour tous les fournisseurs proposant ce matériau), en se limitant aux prix moyens strictement inférieurs à 50 euros par kilogramme.

CCMP 2023 — La typographie informatisée

Extrait verbatim de : CCMP 2023 — Partie II – Gestion de polices de caractères vectorielles

Contexte et schéma de la base

Une base de données stocke les informations liées aux polices de caractères dans 4 tables ou relations.

Famille décrit les familles de polices, avec `fid` la clé primaire entière et `fnom` leur nom.

Police décrit les polices de caractères disponibles, avec `pid` la clé primaire entière, `pnom` le nom de la police et `fid` de numéro de sa famille.

Caractere décrit les caractères, avec `code` la clé primaire entière, `car` le caractère lui-même, `cnom` le nom du caractère.

Glyphe décrit les glyphes disponibles, avec `gid` la clé primaire entière, `code` le code du caractère correspondant au glyphe, `pid` le numéro de la police à laquelle le glyphe appartient, `groman` un booléen vrai pour du roman et faux pour de l'italique et `gdesc` la description vectorielle du glyphe.

Voici un extrait du contenu de ces tables.

Table `Famille` :

fid	fnom
1	Humane
2	Garalde
3	Réale
4	Didone
5	Mécane
6	Linéale
…	…

Table `Police` :

pid	pnom	fid
1	Centaur	1
2	Garamond	2
3	Times New Roman	3
4	Computer Modern	4
…	…	…
21	Triangle	6
…	…	…

Table `Caractere` :

code	car	cnom
65	A	lettre majuscule latine a
66	B	lettre majuscule latine b
…	…	…
97	a	lettre minuscule latine a
98	b	lettre minuscule latine b
99	c	lettre minuscule latine c
…	…	…

Table `Glyphe` :

gid	code	pid	groman	gdesc
1	65	20	True	`[ [ [0, 0], [1, 2], [2, 0] ], [ [0.5, 1], [1.5, 1] ] ]`
2	65	20	False	`[ [ [0, 0], [2, 2], [2, 0] ], [ [1, 1], [2, 1] ] ]`
…	…	…	…	…
501	97	21	True	`[ [ [0, 0], [0.5, 1], [1, 0], [0, 0] ] ]`
502	98	21	True	`[ [ [0, 2], [0, 0], [1, 0.5], [0, 1] ] ]`
503	99	21	True	`[ [ [1, 1], [0, 0.5], [1, 0] ] ]`
504	100	21	True	`[ [ [1, 2], [1, 0], [0, 0.5], [1, 1] ] ]`
…	…	…	…	…

Questions

Q3. Proposer une requête en SQL sur cette base de données pour compter le nombre de glyphes en roman (cf. description précédente).

Q4. Proposer une requête en SQL afin d'extraire la description vectorielle du caractère A dans la police nommée Helvetica en italique.

Q5. Proposer une requête en SQL pour extraire les noms des familles qui disposent de polices et leur nombre de polices, classés par ordre alphabétique.

CCMP 2024 — Introduction à deux problèmes en communication numérique

Extrait verbatim de : CCMP 2024 — Partie 1.2 Exploitation d'analyses existantes

Contexte et schéma de la base

On peut éviter de réaliser une analyse de la chaîne de caractères à coder en exploitant des données statistiques disponibles. La numérisation de larges corpus littéraires a permis la création de bases de données contenant des informations relatives au nombre d'occurrences des différents caractères alphanumériques dans diverses langues. Nous allons supposer disposer d'une base de données constituée de trois tables (les clefs primaires sont soulignées) :

- `caractere(idCar, symbole, typeCaractere, codeHTML)` ;
- `corpus(idLivre, titre, auteur, annee, nombreCaracteres, langue)` ;
- `occurrences(idCar, idLivre, nombreOccurrences)` ;

dont voici quelques extraits :

Table `caractere` :

idCar	symbole	typeCaractere	codeHTML
65	'A'	'lettre'	'A'
48	'0'	'nombre'	'0'

Table `corpus` :

idLivre	titre	auteur	annee	nombreCaracteres	langue
1	'Germinal'	'Zola'	1885	1152365	'Français'
2	'Les Misérables'	'Hugo'	1862	2245300	'Français'

Table `occurrences` :

idCar	idLivre	nombreOccurrences
62	31	155
37	21	1550

Questions

Q8. Écrire en langage SQL une requête permettant d'obtenir la liste sans doublon des auteurs présents dans la table `corpus`.

Q9. Écrire une requête SQL permettant d'obtenir la fréquence d'occurrences (donc comprise entre 0 et 1) de chaque caractère en `'Français'`. Plus précisément, la requête devra renvoyer une table à deux attributs : une colonne contenant le symbole de chaque caractère, et une autre colonne contenant le rapport entre le nombre total d'occurrences du caractère et le nombre total de caractères des textes de tout le corpus.

CCMP 2025 — Autour du sac à dos

Extrait verbatim de : CCMP 2025 — 1 Base de données - Exemple de fret maritime de conteneurs

Contexte et schéma de la base

On donne les tables `NAVIRES` et `CONTENEURS` suivantes :

- `NAVIRES(idN, evp, portDepN, dateDep, portDestN)`
  - `idN` : clé primaire, identifiant du navire (type entier, non nul)
  - `evp` : capacité en équivalent conteneurs de longueur 20 pieds (type entier)
  - `portDepN` : port de départ (type chaîne de caractères)
  - `dateDep` : date de départ (type date)
  - `portDestN` : port de destination (type chaîne de caractères)

Table `NAVIRES` :

idN	evp	portDepN	dateDep	portDestN
12	1500	"Marseille"	2025-06-23	"Hambourg"
2	16000	"Valence"	2025-06-18	"Algésiras"
5	500	"Le Havre"	2025-10-06	"Rotterdam"
8	23000	"Hongkong"	2025-07-02	"Shanghai"
…	…	…	…	…

- `CONTENEURS(idC, idN, taille, portDepC, dateDisp, portDestC, val)`
  - `idC` : clé primaire, identifiant du conteneur (type entier)
  - `idN` : identifiant du navire attribué (type entier, valeur 0 si non encore attribué).
  - `taille` : longueur du conteneur, 20 ou 40 pieds (type entier). La hauteur et la profondeur sont identiques.
  - `portDepC` : port de départ (type chaîne de caractères)
  - `dateDisp` : date de mise à disposition (type date)
  - `portDestC` : port de destination (type chaîne de caractères)
  - `val` : tarification, valeur entière de 1 à 5 par ordre croissant de profit pour l'entreprise (type entier)

Table `CONTENEURS` :

idC	idN	taille	portDepC	dateDisp	portDestC	val
103	12	20	"Marseille"	2025-08-08	"Hambourg"	2
218	12	40	"Marseille"	2025-07-08	"Valence"	1
5	0	40	"Le Havre"	2025-10-01	"Shangai"	5
885	8	20	"Hongkong"	2025-07-01	"Barcelone"	1
…	…	…	…	…	…	…

Pour les deux questions qui suivent, on demande d'écrire les requêtes en langage SQL.

On notera que pour les valeurs d'attributs de type date (format AAAA-MM-JJ), les relations d'ordre $(<, \leq, >, \geq, =, \neq)$ sont autorisées. Par exemple, 2025-07-02 $<$ 2025-10-03.

Questions

Q1. Écrire une requête permettant de retourner la liste des identifiants de conteneurs et leur ratio tarification/longueur trié par ordre décroissant, partant de Marseille et devant aller à Barcelone, avec une mise à disposition avant le 01/01/2025.

Q2. Écrire une requête permettant de retourner les identifiants de navire et leur nombre respectif de conteneurs attribués.

CCMP 2026 — Jeu Quixo à deux joueurs

Extrait verbatim de : CCMP 2026 — Partie 4. Gestion d'une base de données

Contexte et schéma de la base

Pour réaliser des analyses sur les parties, on les stocke dans une base de données. Cette base de données est composée de 2 tables :

- **Partie** avec les attributs :
  - `id` l'identifiant d'une partie, clé primaire, de type entier ;
  - `id_joueur1` l'identifiant du joueur 1, clé étrangère, de type entier ;
  - `id_joueur2` l'identifiant du joueur 2, clé étrangère, de type entier ;
  - `id_gagnant` l'identifiant du gagnant, clé étrangère, de type entier ;
  - `date` la date de la partie, de type chaîne de caractères ;
  - `nbtours` le nombre de tours de la partie, de type entier.
- **Joueur** avec les attributs :
  - `id` l'identifiant du joueur, clé primaire, de type entier ;
  - `nom` le nom du joueur, de type chaîne de caractères ;
  - `prenom` le prénom du joueur, de type chaîne de caractères.

Questions

Q20. Écrire une requête SQL qui renvoie le nombre de parties et le nombre moyen de tours réalisés par le joueur nommé "John Doe" quand il a commencé la partie (il est le joueur 1).

Q21. Écrire une requête SQL qui renvoie le nombre de tours de la partie ayant nécessité le moins de tours, ainsi que les noms et prénoms des joueurs 1 et 2 ayant participé à cette partie. On supposera l'unicité de cette partie.

Mémo SQL

SQL est un langage **déclaratif** : une requête décrit le résultat recherché, sans imposer la manière de le calculer. Les mots-clés ne sont pas sensibles à la casse, mais on les écrit par convention en majuscules. Les parenthèses regroupent les expressions, délimitent les sous-requêtes et rendent explicite l'enchaînement des jointures. Les espaces et retours à la ligne n'affectent pas la requête : on les utilise pour l'indenter et la rendre lisible, condition essentielle pour l'écrire et la vérifier sans erreur.

Requêtes sans agrégat

Le résultat est une table : les lignes retenues, projetées sur les expressions du `SELECT`.

```
SELECT expression1 AS nom1, expression2, ...
FROM (table1 JOIN table2 ON condition_de_jointure)
WHERE condition_sur_les_lignes
ORDER BY expression
LIMIT n OFFSET k
```

Seuls `SELECT` et `FROM` sont obligatoires, les autres mots-clés sont facultatifs mais apparaissent toujours dans cet ordre.

- **Doublons** : `SELECT DISTINCT ...` élimine les doublons du résultat.
- **Opérations** : les expressions peuvent calculer (`+`, `-`, `*`, `/`), dans le `SELECT` comme dans le `WHERE`.
- **Comparaisons** : `=`, `<>`, `<`, `<=`, `>`, `>=`. La comparaison des chaînes de caractères se fait par ordre lexicographique.
- **Opérateurs logiques** : les conditions se combinent avec `AND`, `OR`, `NOT`.
- **Chaînes de caractères** : entre apostrophes, `'Marseille'`, les guillemets doubles (`"..."`) désignent en principe des noms de tables ou de colonnes, pas des chaînes. Un format `AAAA-MM-JJ` est triable par ordre chronologique.
- **Tri** : `ORDER BY expression` croissant par défaut (`ASC`), décroissant avec `DESC`.
- **Limites** : `LIMIT n` ne garde que les `n` premières lignes ; `OFFSET k` en saute d'abord `k`. Sans `ORDER BY`, ces lignes ne sont pas déterminées et `LIMIT` peut aussi écarter des ex æquo.

Par exemple, pour déterminer le deuxième élément le plus grand :

```
SELECT expression
FROM table
ORDER BY expression DESC
LIMIT 1 OFFSET 1
```

Requêtes avec agrégat

Le résultat est une ligne par **groupe** : on regroupe, puis on résume chaque groupe par des agrégats.

```
SELECT attribut_de_groupe, AGG1(expression), AGG2(expression), ...
FROM (table1 JOIN table2 ON condition_de_jointure)
WHERE condition_sur_les_lignes
GROUP BY attribut_de_groupe
HAVING condition_sur_les_groupes
ORDER BY expression
LIMIT n OFFSET k
```

- **Agrégats** avec `e` une expression (pouvant contenir des attributs et opérations)
  - `COUNT(*)` (nombre de lignes), `COUNT(attribut)` (nombre de valeurs), `COUNT(DISTINCT attribut)` (nombre de valeurs distinctes)
  - `MIN(e)` (minimum), `MAX(e)` (maximum)
  - `SUM(e)` (somme), `AVG(e)` (moyenne)
- Avec `GROUP BY`, le `SELECT` ne peut **dépendre** que des attributs de regroupement et des agrégats.
- Sans `GROUP BY` on agrège sur toutes les lignes, le résultat est donc une seule ligne.
- Deux clauses permettent de filtrer :
  - `WHERE` filtre les **lignes**, **avant** le regroupement,
  - `HAVING` filtre les **groupes**, **après** le regroupement.

Sous-requête

On peut utiliser un agrégat dans une **sous-requête scalaire**, par exemple pour déterminer tous les éléments plus petits que la moyenne :

```
SELECT attribut
FROM table1
WHERE attribut < (SELECT AVG(attribut) FROM table1)
```

Un filtre sur une clé peut aussi garantir une unique valeur :

```
SELECT expression FROM table1
WHERE attribut < (SELECT attribut FROM table2 WHERE cle = valeur)
```

Alias

`AS` donne un nom à une table ou à une colonne calculée. Le préfixe `alias.attribut` indique sans ambiguïté la table d'origine d'un attribut.

```
SELECT a.attribut * b.attribut AS produit
FROM (table1 AS a JOIN table2 AS b ON a.cle = b.cle)
```

Attention à l'ordre effectif des opérations :

$$\text{FROM} \to \text{WHERE} \to \text{GROUP BY} \to \text{HAVING} \to \text{SELECT} \to \text{ORDER BY} \to \text{LIMIT/OFFSET}$$

un alias déclaré dans un `SELECT` n'est pas disponible dans le `WHERE` mais l'est dans le `ORDER BY`.

Opérations ensemblistes

Il est possible d'effectuer des opérations ensemblistes sur des requêtes qui renvoient le **même nombre de colonnes**, dans le même ordre, avec des types compatibles.

- `UNION` renvoie l'union des lignes des deux requêtes,
- `INTERSECT` renvoie l'intersection des lignes des deux requêtes,
- `EXCEPT` renvoie les lignes de la première requête, privées de celles de la seconde.

On peut enchaîner plusieurs opérations. Un éventuel `ORDER BY` s'écrit une seule fois, à la fin, et trie le résultat complet. Les trois opérations suppriment les doublons.

```
SELECT expression FROM table1
UNION
SELECT expression FROM table2
ORDER BY expression ASC
```

Anti-jointure

`EXCEPT` permet notamment d'écrire une **anti-jointure** : on cherche les valeurs présentes dans `table1` mais absentes de `table2`.

```
SELECT expression FROM table1
EXCEPT
SELECT expression FROM table2
```

Hors programme : une anti-jointure peut aussi s'écrire avec une sous-requête et `NOT IN`.

Cela peut être combiné avec un `JOIN` pour filtrer les lignes d'une table selon l'absence de correspondance dans une autre table.

```
SELECT expression
FROM table1
EXCEPT
SELECT tab1.expression
FROM (table1 AS tab1 JOIN table2 AS tab2
  ON tab1.cle = tab2.cle)
```

Produit cartésien et jointure

`FROM tab1, tab2` forme le **produit cartésien** des deux tables : si la table `tab1` a n_1 lignes et c_1 colonnes, et la table `tab2` a n_2 lignes et c_2 colonnes, le produit cartésien a $n_1 \times n_2$ lignes et $c_1 + c_2$ colonnes.

En pratique on utilise plutôt des **jointures**, qui ne conservent que les lignes ayant des correspondances entre les tables. La syntaxe `FROM (tab1 JOIN tab2 ON condition)` ne conserve que les lignes du produit cartésien qui satisfont `condition`.

table1

cle	expr1
1	a
2	b

table2

cle1	expr2
1	x
1	y
3	z

$\times \rightarrow$

Produit cartésien

cle	expr1	cle1	expr2
1	a	1	x
1	a	1	y
1	a	3	z
2	b	1	x
2	b	1	y
2	b	3	z

cle = cle1 $\rightarrow$

Jointure

cle	expr1	cle1	expr2
1	a	1	x
1	a	1	y

La jointure associe ici la ligne de clé `1` de `table1` aux deux lignes de `table2` portant cette clé. Les clés `2` et `3`, sans correspondance, ne figurent pas dans le résultat.

```
SELECT expression
FROM (table1 JOIN table2 ON condition_de_jointure)
```

revient à

```
SELECT expression
FROM table1, table2
WHERE condition_de_jointure
```

Jointures de plusieurs tables

On peut enchaîner les `JOIN` ; chaque clause `ON` relie la nouvelle table aux précédentes. Plusieurs égalités se combinent avec `AND`.

```
SELECT expression
FROM ((table1 AS a JOIN table2 AS b ON a.cle = b.cle1)
  JOIN table3 AS c ON b.cle3 = c.cle)
```

Une **auto-jointure** utilise deux fois la même table, sous deux alias distincts :

```
SELECT expression
FROM (table1 AS a JOIN table1 AS b
  ON a.attribut = b.attribut)
```

Vocabulaire des clés

Une **clé primaire** identifie chaque ligne de façon unique ; elle peut être constituée de plusieurs attributs. Une **clé étrangère** référence la clé primaire d'une autre table : l'égalité entre ces deux clés fournit alors une condition de jointure entre les tables.

Notes

Notes