MINISTÈRE
DE L'ÉDUCATION
NATIONALE
Liberté
Égalité
Fraternité

Concours externe de l'agrégation du second degré

Section informatique

Programme de la session 2027

Le présent document définit le programme des épreuves de la section informatique du concours de l'agrégation pour la session 2027.

Ce référentiel est constitué des programmes d'enseignement de la spécialité « numérique et sciences informatiques » (NSI) du cycle terminal de la voie générale du lycée, ainsi que de ceux des classes préparatoires scientifiques aux grandes écoles « mathématiques, physique, ingénierie et informatique » (MP2I) et « mathématiques, physique, informatique » (MPI), complétés et enrichis par les éléments exposés dans la suite de ce texte. Certains concepts fondamentaux de ces parcours d'enseignement sont rappelés dans la présentation qui suit afin d'assurer au document une cohérence globale. Pour l'ensemble des notions, il est attendu des candidats un recul scientifique correspondant à un niveau master.

La connaissance des langages OCaml, C et Python est attendue, ainsi que la maîtrise du langage de requêtes SQL. L'annexe A du présent document fixe les éléments exigibles pour l'ensemble de ces langages.

Architecture

- Circuits combinatoires/séquentiels, machine de Mealy, machine de Moore.
- Description et fonctionnement d'une machine de von Neumann (introduction à la programmation assembleur : instructions de calcul, instructions de branchement, accès à la mémoire et aux registres, instructions système).
- Exécution d'un appel de fonction, concept de pile.
- Hiérarchie mémoire.
- Typologie des machines parallèles (classification de Flynn, classification de Raina, machines multi-coeurs, supercalculateurs).
- Représentation des nombres à virgule flottante. Problèmes de précision des calculs flottants et de dépassement de capacité. Notion de mode d'arrondi.

Bases de données

- Création, suppression, modification de tables au travers du langage de requêtes SQL.
- Opérateurs de l'algèbre relationnelle et leurs propriétés : application à l'optimisation de requêtes.
- Calcul relationnel et théorème de Codd.

Calculabilité, complexité

- Modèle de calcul. Machines de Turing : définition, principales variantes (ruban biinfini vs infini, machine à plusieurs rubans). La machine de Turing est le modèle de calcul retenu pour l'étude des notions qui suivent.
- Calculabilité : universalité, décidabilité, indécidabilité. Problème de l'arrêt.
- Complexité : complexité en temps et en espace, classe P. Acceptation par certificat, classe NP. Réduction polynomiale. NP-complétude. Théorème de Cook.

La notion de machine de Turing non déterministe n'est pas exigible.

Logique

— Logique du premier ordre (aspects syntaxiques) : langages, termes, formules, variables libres et variables liées, substitutions, capture de variables.
— Logique du premier ordre (aspects sémantiques) : interprétation d'une formule dans un modèle, validité, satisfiabilité.

Fondements de la programmation

— Preuve de programmes : correction, terminaison, logique de Hoare, induction structurelle.

Chaîne de compilation

— Analyse lexicale, analyse syntaxique (principes de l'analyse descendante).
— Analyse sémantique élémentaire (arbre de syntaxe abstraite, environnement, analyse de portée, typage).
— Génération de code vers une machine à pile.

Génie logiciel

— Analyse : phases de développement ; diagrammes associés (diagrammes de cas d'utilisation, d'état, de séquence, de classes, etc.).
— Assertions, jeux de tests, tests en boîte blanche/noire.
— Notion de test fonctionnel appliqué aux tests unitaires, comportementaux et d'intégration.
— Notion de test non-fonctionnel : tests de performances, tests d'intrusions.
— Tests avancés : bouchons de test (*mock*), notion de couvertures de test.
— Gestion de projet : intégration & livraison continue, gestionnaire de code source (*e.g.*, Git).

Informatique et société

— Points clés du RGPD.
— Impact environnemental du numérique. Notion d'écoconception.
— Enjeux d'éthique du numérique et valeurs sous-jacentes.
— Cybersécurité : notions de vulnérabilité, menace, attaque ; chiffrement symétrique et asymétrique.

Intelligence artificielle

— Mesures de similarité pour l'apprentissage machine.
— Données d'entraînement et données de test, choix des descripteurs.
— Enjeux d'éthique (biais d'apprentissage, transparence).

Programmation

— Programmation objet : objets, classes, héritage, polymorphisme de sous-typage.
— Programmation fonctionnelle : ordre supérieur, structures immuables, polymorphisme paramétrique.

Réseaux

— Caractéristiques des réseaux et performances associées :
  — réseaux d'accès, réseaux de coeur ;
  — topologies de réseaux : point à point, à diffusion ;
  — performances : débit de transmission, délai, taux de perte.
— Modélisation en couches : OSI, TCP/IP, encapsulation.
— Transmission :
  — Adressage : adresses MAC, IPv4, IPv6 ;
  — Routage : principes ; système de nom de domaine (DNS) ; routage à vecteur de distance (algorithme de Bellman-Ford), routage par état de liens (algorithme de Dijkstra) ;
  — Solutions de transport : principes ; TCP, UDP.
— Programmation réseau : API Socket en Python et en C, à l'aide d'un aide-mémoire fourni.

Systèmes d'exploitation

— Séquence de démarrage : de l'initialisation aux processus utilisateur.
— Abstractions fournies par le système : accès au matériel, répartition équitable des ressources, accès aux fichiers (dont `open`, `read`, `write` et `close`).
— Liens entre système d'exploitation et applications : adressage physique et virtuel, notion de pagination, MMU, interruptions, appels systèmes, gestion des processus (dont `fork`, `wait`, `waitpid`), gestion du temps (ticks et implémentation du temps partagé).
— Isolation et interaction entre les processus : espace mémoire d'une application, communication entre applications (`pipe`, `mmap`).
— Concurrence : modèles de cohérence (forte, faible, PRAM et au relâchement) et d'équité. Construction des mutex et sémaphores à partir des instructions atomiques *test and set*. Schéma lecteurs rédacteurs.
— Émulation et virtualisation : types d'hyperviseurs (natif, hébergé) et conteneur.
— Virtualisation matérielle et para-virtualisation.

MINISTÈRE DE L'ÉDUCATION NATIONALE
Liberté
Égalité
Fraternité

Concours externe de l'agrégation du second degré

Section informatique

Programme de la session 2027

A Langages

A.1 Langage Python

Cette section liste limitativement les éléments du langage Python (version 3) dont la connaissance, est exigible des candidats à l'agrégation d'informatique. Elle contient en particulier le programme de l'Informatique Tronc Commun des CPGE scientifiques, mais ne s'y limite pas. Les éléments ci-dessous s'inscrivent dans la perspective de lire et d'écrire des programmes en Python ; aucun concept sous-jacent n'est exigible au titre de la présente annexe. Aucune connaissance sur un module particulier n'est exigible des candidats.

A.1.1 Traits et éléments techniques à connaître

Les éléments et notations suivants du langage Python doivent pouvoir être compris et utilisés par les candidats sans faire l'objet d'un rappel, y compris lorsqu'ils n'ont pas accès à un ordinateur.

Traits généraux

- Typage dynamique : l'interprète détermine le type à la volée lors de l'exécution du code.
- Principe d'indentation.
- Portée lexicale : lorsqu'une expression fait référence à une variable à l'intérieur d'une fonction, Python cherche la valeur définie à l'intérieur de la fonction et à défaut la valeur dans l'espace global du module.
- Appel de fonction par valeur : l'exécution de $f(e)$ évalue d'abord l'expression e puis exécute f avec la valeur obtenue. Certaines valeurs sont des scalaires (booléens, entiers, etc.) d'autres des adresses (listes, objets, etc.).
- Appel de fonction par référence d'objet : l'exécution de $f(e)$ évalue d'abord l'expression e et lie l'objet résultant au paramètre de la fonction f. Distinction entre objets immuables (entiers, flottants, booléens, chaînes, n-uplets) et objets mutables (listes, dictionnaires, ensembles) dans la sémantique d'appel.

Types de base

- Opérations sur les entiers (`int`) : `+`, `-`, `*`, `//`, `**`, `%` avec des opérandes positifs.
- Opérations sur les flottants (`float`) : `+`, `-`, `*`, `/`, `**`.
- Opérations sur les booléens (`bool`) : `not`, `or`, `and` (et leur caractère paresseux).
- Comparaisons `==`, `!=`, `<`, `>`, `<=`, `>=`.

Types structurés

- Structures indicées immuables (chaînes, n-uplets) : `len`, accès par indice positif valide, concaténation `+`, répétition `*`, extraction de tranche.
- Listes : création par compréhension `[`e `for` x `in` s`]`, par `[`e`]` `*` n, par `append` successifs ; `len`, accès par indice positif valide ; concaténation `+`, extraction de tranche, copie (y compris son caractère superficiel) ; `pop` en dernière position. Utilisation des listes pour représenter des tableaux.
- Dictionnaires : création `{`c_1 `:` v_1`,` `...,` c_n `:` v_n`}`, accès, insertion (d`[`k`]` `=` v), présence d'une clé (k `in` d), `len`, `copy`.

Structures de contrôle

- Instruction d'affectation avec `=`. Dépaquetage de n-uplets.
- Instruction conditionnelle : `if`, `elif`, `else`.
- Boucle `while` (sans `else`). `break`, `continue`, `return` dans un corps de boucle.
- Boucle `for` (sans `else`) et itération sur `range(`a`, `b`)`, une chaîne, un n-uplet, une liste, un dictionnaire au travers des méthodes `keys` et `items`.
- Définition d'une fonction `def` f`(`x_1`, ..., `x_n`)`, `return`.
- Exceptions : lever une exception avec `raise`, la rattraper avec `try/except`.

Programmation objet

- Classe, constructeur (`__init__` avec un premier argument `self`), attributs.
- Méthodes (avec un premier argument `self`).
- Encapsulation de données dans un objet.
- Application : construction de structures chaînées (cellules de listes) et arborescentes (nœuds d'arbres). Utilisation de `None` comme pointeur nul, comparaison avec `is None`. Encapsulation de la structure dans une autre classe.
- Héritage. On se limite à de l'héritage simple.

Divers

- Introduction d'un commentaire avec `#`.
- Utilisation simple de `print`, sans paramètre facultatif.
- Importation de modules avec `import` *module* ou `import` *module* `as` *alias* ou `from` *module* `import` $f, g, \ldots$
- Assertion : `assert` (sans message d'erreur).

A.1.2 Éléments techniques devant être reconnus et utilisables après rappel

Les éléments suivants du langage Python doivent pouvoir être utilisés par les candidats pour écrire des programmes dès lors qu'ils ont fait l'objet d'un rappel et que la documentation correspondante est fournie.

Types structurés

- Ensembles : création `set()`, insertion (`add`), présence d'un élément (e `in` s), `len`.

Divers

- Annotations de types. Vérification statique avec un outil tel que `mypy`.
- Manipulation de fichiers texte (la documentation utile de ces fonctions doit être rappelée ; tout problème relatif aux encodages est éludé) : `open`, `read`, `readline`, `readlines`, `split`, `write`, `close`. Construction `with`.
- Tirage pseudo-aléatoire avec la fonction `randint` du module `random`.

A.2 Langage C

La présente section liste limitativement les éléments du langage C (norme C99 ou plus récente) dont la connaissance, selon les modalités de chaque sous-section, est exigible des étudiants à la fin de la première année. Ces éléments s'inscrivent dans la perspective de lire et d'écrire des programmes en C ; aucun concept sous-jacent n'est exigible au titre de la présente annexe.

À l'écrit, on travaille toujours sous l'hypothèse que les entêtes suivants ont tous été inclus : `<assert.h>`, `<stdbool.h>`, `<stddef.h>`, `<stdint.h>`, `<stdio.h>`, `<stdlib.h>`. Mais ces fichiers ne font pas en soi l'objet d'une étude et aucune connaissance particulière des fonctionnalités qu'ils apportent n'est exigible.

A.2.1 Traits et éléments techniques à connaître

Les éléments et notations suivants du langage C doivent pouvoir être compris et utilisés par les étudiants sans faire l'objet d'un rappel, y compris lorsqu'ils n'ont pas accès à un ordinateur.

Traits généraux

- Typage statique. Types indiqués par le programme lors de la déclaration ou définition.
- Passage par valeur.
- Délimitation des portées par les accolades. Les retours à la ligne et l'indentation ne sont pas signifiants mais sont nécessaires pour la lisibilité du code.
- Déclaration et définition de fonctions, uniquement dans le cas d'un nombre fixé de paramètres.
- Gestion de la mémoire : pile et tas, allocation statique et dynamique, durée de vie des objets.

Définitions et types de base

- Types entiers signés `int8_t`, `int32_t` et `int64_t`, types entiers non signés `uint8_t`, `uint32_t` et `uint64_t`. Lorsque la spécification d'une taille précise pour le type n'apporte rien à l'exercice, on utilise les types signé `int` et non signé `unsigned int`. Opérations arithmétiques `+`, `-`, `/`, `*`. Opération `%` entre opérandes positifs. Ces opérations sont sujettes à dépassement de capacité. À l'écrit, on élude les difficultés liées à la sémantique des constantes syntaxiques. On ne présente pas les opérateurs d'incrémentation.
- Le type `char` sert exclusivement à représenter des caractères codés sur un octet. Notation `'\0'` pour le caractère nul.
- Type `double` (on considère qu'il est sur 64 bits). Opérations `+`, `-`, `*`, `/`.
- Type `bool` et les constantes `true` et `false`. Opérateurs `!`, `&&`, `||` (y compris évaluation paresseuse). Les entiers ne doivent pas être utilisés comme booléens, ni l'inverse.
- Opérateurs de comparaison `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Les constantes du programme sont définies par `const` *type* c `=` v. On n'utilise pas la directive du préprocesseur `#define` à cette fin.

Types structurés

- Tableaux statiques : déclaration par *type* T`[`s`]` où s est une constante littérale entière. Lecture et écriture d'un terme de tableau par son indice T`[`i`]` ; le langage ne vérifie pas la licéité des accès. Tableaux statiques multidimensionnels.

— Définition d'un type structuré par **struct** nom_s {$type_1$ $champ_1$; ... $type_n$ $champ_n$;} et ensuite **typedef struct** nom_s nom (la syntaxe doit cependant être rappelée si les étudiants sont amenés à écrire de telles définitions). Lecture et écriture d'un champ d'une valeur de type structure par $v.champ$ ainsi que v->$champ$. L'organisation en mémoire des structures n'est pas à connaître.
— Chaînes de caractères vues comme des tableaux de caractères avec sentinelle nulle. Fonctions `strlen`, `strcpy`, `strcat`.

Structures de contrôle
— Conditionnelle **if** (c) s_T, **if** (c) s_T **else** s_F.
— Boucle **while** (c) s ; boucle **for** ($init$; fin; $incr$) s, possibilité de définir une variable dans $init$; **break**.
— Définition et déclaration de fonction, passage des paramètres par valeur, y compris des pointeurs. Cas particuliers : passage de paramètre de type tableau, simulation de valeurs de retour multiples.

Pointeurs et gestion de la mémoire
— Pointeur vers une valeur en mémoire, notation $type$* p = &v. On considère que les pointeurs sont sur 64 bits.
— Déréférencement d'un pointeur valide, notation *p. On ne fait pas d'arithmétique des pointeurs.
— Pointeurs comme moyen de réaliser une structure récursive. Pointeur `NULL`.
— Création d'un objet sur le tas avec `malloc` et **sizeof** (on peut présenter `size_t` pour cet usage mais sa connaissance n'est pas exigible). Libération avec `free`.
— Transtypage de données depuis et vers le type **void*** dans l'optique stricte de l'utilisation de fonctions comme `malloc`.
— En particulier : gestion de tableaux de taille non statiquement connue ; linéarisation de tels tableaux quand ils sont multidimensionnels.

Divers
— Utilisation de **assert** lors d'opérations sur les pointeurs, les tableaux, les chaînes.
— Flux standard.
— Utilisation élémentaire de `printf` et de `scanf`. La syntaxe des chaînes de format n'est pas exigible.
— Notion de fichier d'en-tête. Directive **#include** "$fichier.h$".
— Commentaires /* ... */ et commentaires ligne //

A.2.2 Éléments techniques devant être reconnus et utilisables après rappel

Les éléments suivants du langage C doivent pouvoir être utilisés par les étudiants pour écrire des programmes dès lors qu'ils ont fait l'objet d'un rappel et que la documentation correspondante est fournie.

Traits généraux et divers
— Utilisation de **#define**, **#ifndef** et **#endif** lors de l'écriture d'un fichier d'en-tête pour rendre son inclusion idempotente.
— Rôle des arguments de la fonction **int main(int argc, char* argv[])** ; utilisation des arguments à partir de la ligne de commande.

— Fonctions de conversion de chaînes de caractères vers un type de base comme `atoi`.
— Définition d'un tableau par un initialisateur {t_0, t_1, ..., t_{N-1}}.
— Définition d'une valeur de type structure par un initialisateur {.c_1 = v_1, .c_2 = v_2, ...}.
— Compilation séparée.

Gestions des ressources de la machine
— Gestion de fichiers : `fopen` (dans les modes `r` ou `w`), `fclose`, `fscanf`, `fprintf` avec rappel de la syntaxe de formatage.
— Fils d'exécution : inclusion de l'entête `pthread.h`, type `pthread_t`, commandes `pthread_create` avec attributs par défaut, `pthread_join` sans récupération des valeurs de retour.
— Mutex : inclusion de l'entête `pthread.h`, type `pthread_mutex_t`, commandes `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_mutex_destroy`.
— Sémaphore : inclusion de l'entête `semaphore.h`, type `sem_t`, commandes `sem_init`, `sem_destroy`, `sem_wait`, `sem_post`.

A.3 Langage OCaml

La présente section liste limitativement les éléments du langage OCaml (version 4 ou supérieure) dont la connaissance, selon les modalités de chaque sous-section, est exigible des étudiants. Aucun concept sous-jacent n'est exigible au titre de la présente annexe.

A.3.1 Traits et éléments techniques à connaître

Les éléments et notations suivants du langage OCaml doivent pouvoir être compris et utilisés par les étudiants sans faire l'objet d'un rappel, y compris lorsqu'ils n'ont pas accès à un ordinateur.

Traits généraux
— Typage statique, inférence des types par le compilateur. Idée naïve du polymorphisme.
— Passage par valeur.
— Portée lexicale : lorsqu'une définition utilise une variable globale, c'est la valeur de cette variable au moment de la définition qui est prise en compte.
— Curryfication des fonctions. Fonction d'ordre supérieur.
— Gestion automatique de la mémoire.
— Les retours à la ligne et l'indentation ne sont pas signifiants mais sont nécessaires pour la lisibilité du code.

Définitions et types de base
— `let`, `let rec` (pour des fonctions), `let rec` ... `and` ..., `fun` x y `->` e.
— `let` v `=` e `in` e', `let rec` f x `=` e `in` e'.
— Expression conditionnelle `if` e `then` e_V `else` e_F.
— Types de base : `int` et les opérateurs `+`, `-`, `*`, `/`, l'opérateur `mod` quand toutes les grandeurs sont positives ; exception `Division_by_zero` ; `float` et les opérateurs `+.`, `-.`, `*.`, `/.` ; `bool`, les constantes `true` et `false` et les opérateurs `not`, `&&`, `||` (y compris évaluation paresseuse). Entiers et flottants sont sujets aux dépassements de capacité.

— Comparaisons sur les types de base : `=`, `<>`, `<`, `>`, `<=`, `>=`.
— Types `char` et `string` ; 'x' quand x est un caractère imprimable, "x" quand x est constituée de caractères imprimables, `String.length`, s`.[`i`]`, opérateur `^`. Existence d'une relation d'ordre total sur `char`. Immuabilité des chaînes.

Types structurés
— n-uplets ; non-nécessité d'un `match` pour récupérer les valeurs d'un n-uplet.
— Listes : type `'a list`, constructeurs `[]` et `::`, notation `[`x`;` y`;` z`]` ; opérateur `@` (y compris sa complexité) ; `List.length`. Motifs de filtrage associés.
— Tableaux : type `'a array`, notations `[|...|]`, t`.(`i`)`, t`.(`i`) <-` v ; fonctions `length`, `make`, et `copy` (y compris le caractère superficiel de cette copie) du module `Array`.
— Type `'a option`.
— Déclaration de type, y compris polymorphe.
— Types énumérés (ou sommes, ou unions), récursifs ou non ; les constructeurs commencent par une majuscule, contrairement aux identifiants. Motifs de filtrage associés.
— Filtrage : `match` e `with` p_0 `->` v_0 `|` p_1 `->` v_1 ... ; les motifs ne doivent pas comporter de variable utilisée antérieurement ni deux fois la même variable ; motifs plus ou moins généraux, notation `_`, importance de l'ordre des motifs quand ils ont des instances communes.

Programmation impérative
— Absence d'instruction ; la programmation impérative est mise en œuvre par des expressions impures ; `unit`, `()`.
— Références : type `'a ref`, notations `ref`, `!`, `:=`. Les références doivent être utilisées à bon escient.
— Séquence `;`. La séquence intervient entre deux expressions.
— Boucle `while` c `do` b `done` ; boucle `for` v `=` d `to` f `do` b `done`.

Divers
— Usage de `begin` … `end`.
— `print_int`, `print_float`, `print_string`, `read_int`, `read_float`, `read_line`.
— Exceptions : levée et filtrage d'exceptions existantes avec `raise`, `try` ... `with` ... ; dans les cas irrattrapables, on peut utiliser `failwith`.
— Utilisation d'un module : notation $M.f$. Les noms des modules commencent par une majuscule.
— Syntaxe des commentaires, à l'exclusion de la nécessité d'équilibrer les délimiteurs dans un commentaire.

A.3.2 Éléments techniques devant être reconnus et utilisables après rappel

Les éléments suivants du langage OCaml doivent pouvoir être utilisés par les étudiants pour écrire des programmes dès lors qu'ils ont fait l'objet d'un rappel et que la documentation correspondante est fournie.

Traits divers
— Types de base : opérateur `mod` avec opérandes de signes quelconques, opérateur `**`.

— Types enregistrements mutables ou non, notation `{`c_0 `:` t_0`;` c_1 `:` t_1`; ...}`, `{`c_0 `:` t_0`;` **`mutable`** c_1 `:` t_1 `; ...}` ; leurs valeurs, notations `{`c_0 `=` v_0`;` c_1 `=` v_1`; ...}`, $e.c$, $e.c$ `<-` v.
— Fonctions de conversion entre types de base.
— Listes : fonctions `mem`, `exists`, `for_all`, `filter`, `map`, `iter` du module `List`.
— Tableaux : fonctions `make_matrix`, `init`, `mem`, `exists`, `for_all`, `map` et `iter` du module `Array`.
— Types mutuellement récursifs.
— Filtrage : plusieurs motifs peuvent être rassemblés s'ils comportent exactement les mêmes variables. Notation **`function`** p_0 `->` v_0 `|` p_1 `->` v_1 ....
— Boucle **`for`** v `=` f **`downto`** d **`do`** b **`done`**.
— Piles et files mutables : fonctions `create`, `is_empty`, `push` et `pop` des modules `Queue` et `Stack` ainsi que l'exception `Empty`.
— Dictionnaires mutables réalisés par tables de hachage sans liaison multiple ni randomisation par le module `Hashtbl` : fonctions `create`, `add`, `remove`, `mem`, `find` (y compris levée de `Not_found`), `find_opt`, `iter`.
— `Sys.argv`.
— Utilisation de `ocamlc` ou `ocamlopt` pour compiler un fichier dépendant uniquement de la bibliothèque standard.

Gestions des ressources de la machine
— Gestion de fichiers : fonctions `open_in`, `open_out`, `close_in`, `close_out`, `input_line`, `output_string`.
— Fils d'exécution : recours au module `Thread`, fonctions `Thread.create`, `Thread.join`.
— Mutex : recours au module `Mutex`, fonctions `Mutex.create`, `Mutex.lock`, `Mutex.unlock`.

A.4 Langage SQL

Requêtes **`SELECT`** avec simple clause **`WHERE`** (sélection), projection, renommage **`AS`**. Sous-requêtes.

Requêtes **`CREATE`**, **`INSERT`**, **`UPDATE`** et **`DELETE`**.

Utilisation des mots-clés **`DISTINCT`**, **`LIMIT`**, **`OFFSET`**, **`ORDER BY`**. Les opérateurs au programme sont `+`, `-`, `*`, `/` (on passe outre les subtilités liées à la division entière ou flottante), `=`, `<>`, `<`, `<=`, `>`, `>=`, **`AND`**, **`OR`**, **`NOT`**, **`IS NULL`**, **`IS NOT NULL`**.

Opérateurs ensemblistes **`UNION`**, **`INTERSECT`** et **`EXCEPT`**, produit cartésien.

Jointures internes T_1 **`JOIN`** T_2 ... **`JOIN`** T_n **`ON`** ϕ, externes à gauche T_1 **`LEFT JOIN`** T_2 **`ON`** ϕ.

Agrégation avec les fonctions **`MIN`**, **`MAX`**, **`SUM`**, **`AVG`** et **`COUNT`**, y compris avec **`GROUP BY`**. Pour la mise en œuvre des agrégats, on s'en tient à la norme SQL99.

Filtrage des agrégats avec **`HAVING`**.