Université Mohammed V
Faculté des Sciences
Rabat

Cours de Bases de données

SQL

Master : Traitement intelligent des systèmes

Préparé par: Khawla Elansari

Année Universitaire: 2021/ 2022

1. Introduction : SQL

Bases de données

- Une base de données est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données). Ces informations sont en rapport avec une activité donnée et peuvent être utilisées par des programmes ou des utilisateurs communs, d'où la nécessité de leur mise en commun. Lumière sur la question.

- L'intérêt d'une base de données est de permettre la consultation d'un grand nombre de données à des utilisateurs qui s'en sont vu accordé le droit. Une base de données peut être locale (utilisable par une machine, un utilisateur) ou bien répartie (informations stockées sur des machines distantes et accessible par réseau.

Gestion des Bases de données

- Pour permettre une utilisation optimale d'une base de données, il faut mettre en place un système de gestion, d'où l'intérêt des SGBD (en anglais DBMS : Database management system). Le SGBD est donc un ensemble de services permettant :

  - *L'accès aux données de façon simple*
  - *D'autoriser l'accès aux informations à de multiples utilisateurs*
  - *La manipulation des données présentes dans la base (insertion, suppression, modification)*

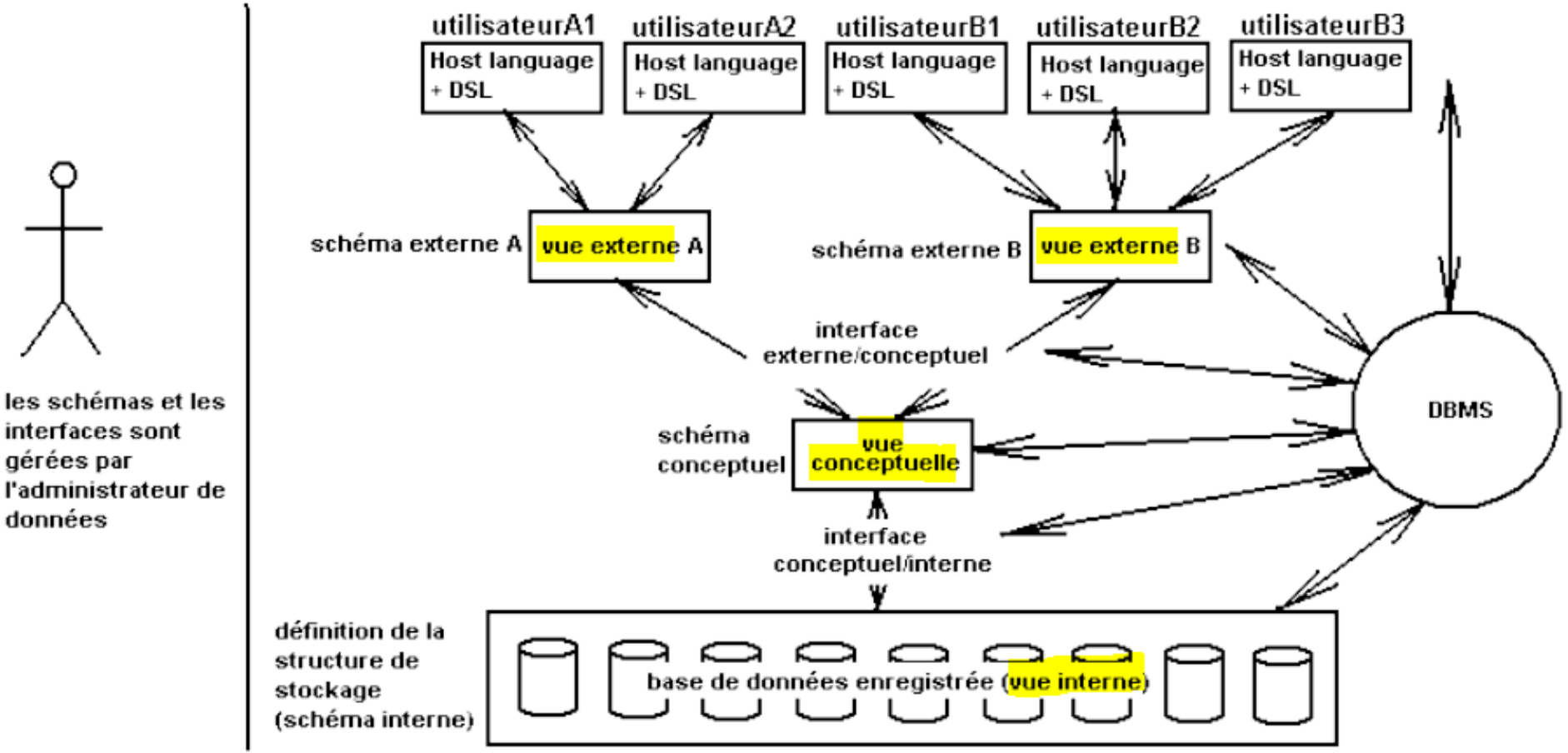
- SGBD: Outil informatique permettant aux utilisateurs de structurer, d'insérer, de modifier, de rechercher de manière efficace des donnée spécifiques, au sein d'une grande quantité d'informations, stockées sur mémoires secondaires partagée de manière transparente par plusieurs utilisateurs.

Pourquoi les SGBD ?

- Intégrité des données : Respect de contraintes qui doivent être programmées (ex: contrôles sur date de naissance, sur code postal, numéro de tél., ...),
- Amélioration de la sécurité des données: utilisateurs de différents niveaux d'expérience et avec différents droits d'accès
- Amélioration du partage des données au sein de l'organisation: accès en lecture / écriture.
- Gestion des concurrences et remises à jour: Le SGBD joue le rôle d'un programme superviseur pour gérer les transactions
- Gain du temps

Architecture d'une base de données

- Le **niveau interne** est défini par le schéma physique qui indique comment l'information est enregistrée sur les mémoires auxiliaires => Ce schéma utilise donc les termes de fichiers, d'index, d'adressages, etc...
- **Le niveau conceptuel** est défini par un schéma conceptuel dont le rôle est de définir les règles de description des données et des relations entre ces données.
- **Le niveau externe** contrairement aux précédents niveaux correspond à plusieurs schémas externes qui ne sont autres que les vues (partielles) qu'ont les différents utilisateurs de la base de données. Les schémas externes ne sont que des sous-schémas du schéma conceptuel.


utilisateurA1
utilisateurA2
utilisateurB1
utilisateurB2
utilisateurB3
Host language + DSL
Host language + DSL
Host language + DSL
Host language + DSL
Host language + DSL
schéma externe A
vue externe A
schéma externe B
vue externe B
interface externe/conceptuel
DBMS
schéma conceptuel
vue conceptuelle
interface conceptuel/interne
définition de la structure de stockage (schéma interne)
base de données enregistrée (vue interne)
les schémas et les interfaces sont gérées par l'administrateur de données

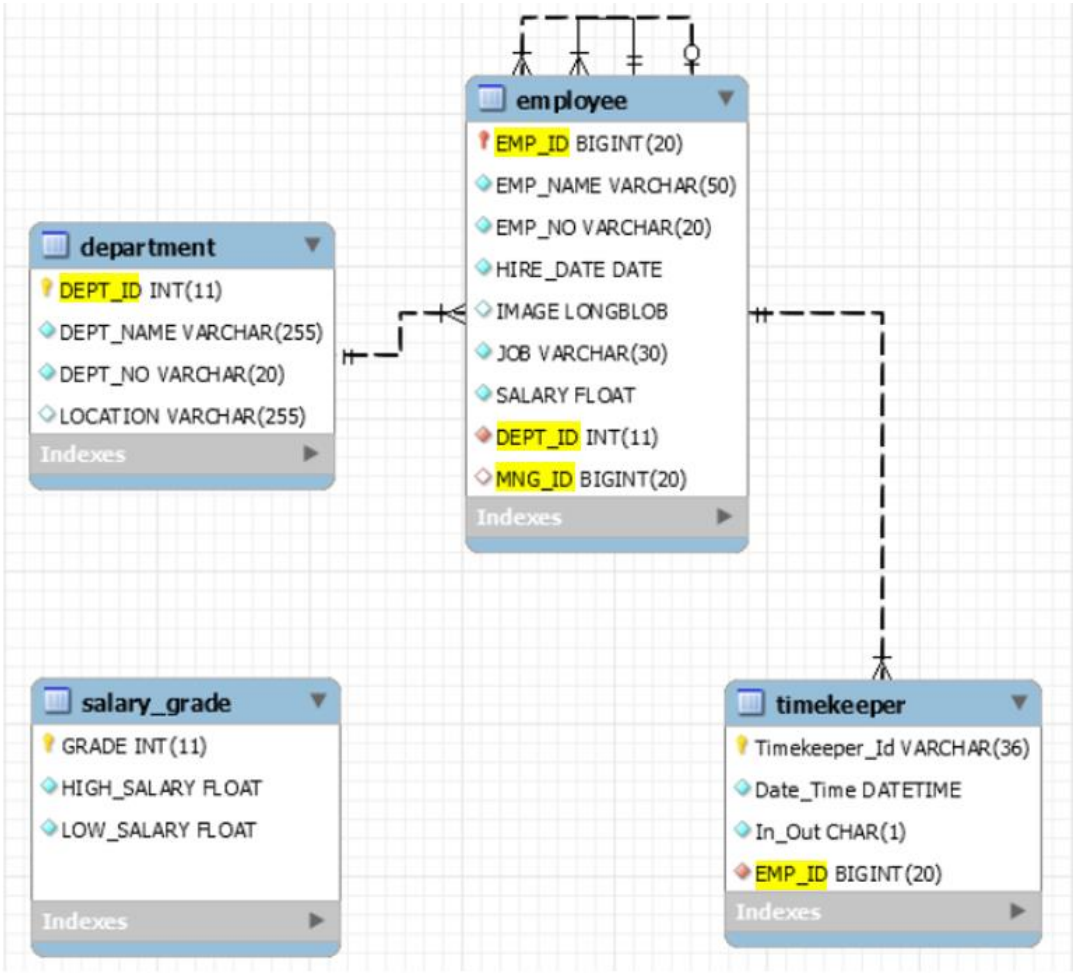
- SQL signifie « Structured Query Language » c'est-à-dire « Langage d'interrogation Structuré ». C'est un langage complet de gestion de bases de données relationnelles.
- Il a été conçu par IBM dans les années 70. Il est devenu le langage standard des systèmes de gestion de bases de données (SGBD) relationnelles (SGBDR).
- Les instructions SQL couvrent 4 domaines :
  - Langage de définition de données (LDD): ***CREATE***, ***ALTER***, ***DROP***, ***RENAME***, ***COMMENT*** ou ***TRUNCATE*** - ***TABLE***, ***VIEW***, ***INDEX…***
  - Langage de manipulation de données (LMD): ***SELECT***, ***UPDATE***, ***INSERT***, ***DELETE***, ***JOIN*** et ***GROUP BY***
  - Langage de contrôle de données (LCD): ***GRANT*** et ***REVOKE***
  - Langage de contrôle des transactions: ***COMMIT*** et ***ROLLBACK***

SQL LDD

Le Langage de Définition des Données est la partie de SQL qui permet de décrire les tables et autres objets manipulés par les SGBD.

Types syntaxiques:

- **VARCHAR2(n)** Chaîne de caractères de longueur variable (maximum n)
- **CHAR(n)** Chaîne de caractères de longueur fixe (n caractères)
- **NUMBER** Nombre entier (40 chiffres maximum)
- **NUMBER(n, m)** Nombre de longueur totale n avec m décimales
- **DATE** Date (DD-MON-YY est le format par défaut)
- **LONG** Flot de caractères




```
create table DEPARTMENT (
        DEPT_ID number(10,0) not null,
        DEPT_NAME varchar2(255 char) not null,
        DEPT_NO varchar2(20 char) not null unique,
        LOCATION varchar2(255 char),
        primary key (DEPT_ID)
    );

create table EMPLOYEE (
        EMP_ID number(19,0) not null,
        EMP_NAME varchar2(50 char) not null,
        EMP_NO varchar2(20 char) not null unique,
        HIRE_DATE date not null,
        IMAGE blob,
        JOB varchar2(30 char) not null,
        SALARY float not null,
        DEPT_ID number(10,0) not null,
        MNG_ID number(19,0),
        primary key (EMP_ID)
    );

create table SALARY_GRADE (
        GRADE number(10,0) not null,
        HIGH_SALARY float not null,
        LOW_SALARY float not null,
        primary key (GRADE)
    );

create table TIMEKEEPER (
        Timekeeper_Id varchar2(36 char) not null,
        EMP_ID number(19,0) not null,
        primary key (Timekeeper_Id)
    );
```

Création de table:

CREATE TABLE <nom de la table> (<nom de colonne> <type> [NOT NULL] [, <nom de colonne> <type>]…, [<contrainte>]…);

<contrainte>: CONSTRAINT <nom de contrainte> <sorte de contrainte>

<sorte de contrainte> est :

- PRIMARY KEY (attribut1, [attribut2…])
- FOREIGN KEY (attribut1, [attribut2…]) REFERENCES <nom de table associée>(attribut1, [attribut2…])
- CHECK (attribut <condition>) (ex: age>25, age IN(24, 25, 26), age BETWEEN 24 and 26)

Modification d'une table existante:

ALTER TABLE <nom de la table> **ADD** (<nom de colonne> <type> [, <contrainte>]…);

ALTER TABLE <nom de la table> **ADD** (<contrainte> [, <contrainte>]…);

ALTER TABLE <nom de la table> **MODIFY** ([<nom de colonne> <nouveau type>] [,<nom de colonne>

<nouveau type>]…);

Description de la structure d'une table:

DESCRIBE <nom de la table>;

Destruction de table:

DROP TABLE <nom de la table>;

SQL LMD

Le langage de manipulation de données est un langage de programmation et un sous-ensemble de SQL pour manipuler les données d'une base de données.

Interrogation des donnees:

SELECT [DISTINCT] <nom de colonne>[, <nom de colonne>]…
FROM <nom de table>[, <nom de table>]…
[WHERE <condition>]
[GROUP BY <nom de colonne>[, <nom de colonne>]…
[HAVING <condition avec calcul verticaux>]]
[ORDER BY <nom de colonne>[, <nom de colonne>]…]

Insertion de données

INSERT INTO <nom de table> [(colonne, …)] VALUES (valeur, …)

INSERT INTO <nom de table> [(colonne, …)] SELECT (Another Select clause)

Modification de données

UPDATE <nom de table> **SET** colonne = valeur, … [**WHERE** condition]

UPDATE <nom de table> **SET** colonne = **SELECT**… (Another Select Clause)

Suppression de données

DELETE FROM <nom de table> [**WHERE** condition]

Les clauses de Select

SELECT * FROM table -- Tous les attributs de la relation font partie du résultat

SELECT colonne1 [alias1], colonne2 [alias2],… **FROM** table -- Si alias est composé de plusieurs mots (séparés par des blancs), il faut utiliser des guillemets.

SELECT DISTINCT colonne1 [alias1], colonne2 [alias2],… **FROM** table -- Suppression des duplicates

WHERE <condition>

select * **from** EMPLOYEE **where** Job **IN** ('PRESIDENT', 'MANAGER');

Autres opérateurs spécifiques :

- **IS NULL** : teste si la valeur d'une colonne est une valeur nulle (inconnue).
- **IN (**liste**)** : teste si la valeur d'une colonne coïncide avec l'une des valeurs de la liste.
- **BETWEEN** v1 **AND** v2 : teste si la valeur d'une colonne est comprise entre les valeurs v1 et v2 (v1 <= valeur <= v2).

Les opérateurs

Opérateurs de comparaison logiques

Opérateur	Description
AND	Effectue une comparaison «Et logique»
BETWEEN a **AND** b	Est situé dans l'intervalle
IS NOT NULL	Effectue une comparaison pour vérifier si la valeur n'est pas NULL.
IS NULL	Effectue une comparaison pour vérifier si la valeur est NULL.
OR	Effectue une comparaison «Ou logique»
NOT	Effectue une «négation logique»
NOT BETWEEN *a* **AND** *b*	Effectue une comparaison n'étant pas dans l'intervalle spécifié.

Opérateurs de chaine de caractères

Opérateur	Description
chaine1 **||** *chaine2*	Effectue la concaténation de chaine de caractères
chaine1 = *chaine2*	Comparaison d'une égalité
chaine1 **<>** *chaine2*	Comparaison d'une différence
chaine1 **>** *chaine2*	Comparaison de plus grand que
chaine1 **>=** *chaine2*	Comparaison de plus grand ou égal que
chaine1 **<** *chaine2*	Comparaison de plus petit que
chaine1 **<=** *chaine2*	Comparaison de plus petit ou égal que
chaine1 **LIKE** *chaine2*	Comprend la chaine de caractères spécifiés. Le caractère générique est «%».
chaine1 **NOT LIKE** *chaine2*	Ne comprend pas la chaine de caractères spécifiés. Le caractère générique est «%».
IN (a,b,...)	Est un membre de

Opérateurs arithmétiques, binaires et logiques

Opérateur	Description
=	Comparaison d'une égalité
<>	Comparaison d'une différence
>	Comparaison de plus grand que
>=	Comparaison de plus grand ou égal que
<	Comparaison de plus petit que
<=	Comparaison de plus petit ou égal que
+	Effectue une addition
-	Effectue une soustraction
*	Effectue une multiplication
/	Effectue une division

Calculs horizontaux

Des expressions arithmétiques permettant d'appliquer des opérations sur plusieurs champs de la même ligne. Elles peuvent être utilisées dans les clauses **WHERE** et **SELECT**.

Exemple: Faire la somme de deux champs sur la même ligne

select REPLACE(EMP_NO, 'E', 'EMP') **from** EMPLOYEE ;

Fonctions Prédéfinies

ABS(n)	Valeur absolue
CEIL(n)	Plus petit entier relatif égal ou supérieur
FLOOR(n)	Plus grand entier relatif inférieur ou égal
MOD(m,n)	Reste de la division de m par n
POWER(m,n)	m puissance n
SIGN(n)	Indique le signe de n
SQRT(n)	Racine carrée de n
ROUND(n)	Arrondi de n à 100 (Partie entière)
ROUND(n,m)	Arrondi de n à 10-m
TRUNC(n)	n tronqué à 100 (Partie entière)
NVL(n1, n2)	Permet de substituer la valeur n2 à n1, au cas où cette dernière est une valeur nulle
INITCAP(char)	La première lettre de chaque mot de la chaîne de caractères est mise en majuscule, toutes les autres lettre sont mises en minuscules.
LOWER(char)	Toutes les lettres sont mises en minuscules
UPPER(char)	Toutes les lettres sont mises en majuscules
REPLACE(char, char1, char2)	Remplace dans la chaîne de caractères char la chaîne de caractères char1 par la chaîne de caractères char2.
SUBSTR(char1,n[,m])	Extrait de la chaîne de caractères char1, les caractères situés à partir du rang n jusqu'à la longueur m, ou jusqu'à la fin si m non spécifié.

Calculs verticaux

Contrairement aux calculs horizontaux, le résultat d'une fonction agrégative est évalué une seule fois pour tous les tuples du résultat.

Exemple: Calculer le salaire moyen

```
select AVG(Salary) from EMPLOYEE ;
```

Fonctions Prédéfinies

AVG(expr)	Moyenne de toutes les valeurs de expr
COUNT(*)	Nombre d'enregistrements retournés par la sélection.
COUNT(expr)	Nombre d'enregistrements retournés par la sélection, pour lesquels expr n'a pas une valeur NULL.
MAX(expr)	Valeur maximale de toutes les valeurs de expr
MIN(expr)	Valeur minimale de toutes les valeurs de expr
STDDEV(expr)	Ecart type de toutes les valeurs de expr
SUM(expr)	Somme de toutes les valeurs de expr
VARIANCE(expr)	Variance de toutes les valeurs de expr

Classification

La classification permet de regrouper les lignes d'une table dans des classes d'équivalence ou sous-tables ayant chacune la même valeur pour la colonne de la classification.

L'opérateur de partitionnement s'exprime par la clause GROUP BY : **GROUP BY** colonne1, [colonne2,…]

Exemple : Calculer le salaire moyen par Job Role

select Job, AVG(Salary) **from** EMPLOYEE **GROUP BY** Job;

Les colonnes indiquées dans **SELECT**, sauf les attributs arguments des fonctions agrégatives, doivent être mentionnées dans **GROUP BY**.

En présence de la clause **GROUP BY**, les fonctions agrégatives s'appliquent à l'ensemble des valeurs de chaque classe d'équivalence.

Recherche dans les sous-tables

Des conditions de sélection peuvent être appliquées aux sous-tables engendrées par la clause **GROUP BY**, comme c'est le cas avec la clause **WHERE** pour les tables.

Cette sélection s'effectue avec la clause **HAVING** qui doit suivre la clause **GROUP BY**. Sa syntaxe est : **HAVING** condition

Exemple : Garder uniquement les Job roles dont la moyenne des salaires est supérieure à 2000

select Job, **AVG(**Salary**) as** AvgSalary **from** EMPLOYEE **GROUP BY** Job **Having AVG(**Salary**) >**2000;

Gestion des transactions

- Toute commande SQL LMD (**INSERT**, **UPDATE** ou **DELETE**) démarre par défaut une transaction.

- Une transaction est un ensemble de modifications de la base qui forme un tout indivisible. Il faut effectuer ces modifications entièrement ou pas du tout, sous peine de laisser la base dans un état incohérent. Les Systèmes de Gestion de Bases de Données permettent aux utilisateurs de gérer leurs transactions. Ils peuvent à tout moment :

  - Valider la transaction en cours par la commande **COMMIT**. Les modifications deviennent définitives et visibles à tous les utilisateurs.
  - Annuler la transaction en cours par la commande **ROLLBACK**. Toutes les modifications depuis le début de la transaction sont alors défaites.

- En cours de transaction, seul l'utilisateur ayant effectué les modifications les voit. Ce mécanisme est utilisé par les systèmes de gestion de bases de données pour assurer l'intégrité de la base en cas de fin anormale d'une tâche utilisateur : il y a automatiquement ROLLBACK des transactions non terminées.

Gestion des transactions

- Une modification directe de la base (**INSERT**, **UPDATE** ou **DELETE**) ne sera validée que par la commande **COMMIT**, ou lors de la sortie normale de sqlPlus par la commande **EXIT**.

- Il est possible de se mettre en mode "validation automatique", dans ce cas, la validation est effectuée automatiquement après chaque commande SQL de mise à jour de données. Pour se mettre dans ce mode, il faut utiliser l'une des commandes suivantes :

SET AUTOCOMMIT IMMEDIATE ou **SET AUTOCOMMIT ON**

- L'annulation de ce mode se fait avec la commande : **SET AUTOCOMMIT OFF**

SQL LCD

Le langage de contrôle de données est un langage de programmation et un sous-ensemble de SQL pour contrôler l'accès aux données d'une base de données.

Les Index

- L'index est utile pour accélérer l'exécution d'une requête SQL qui lit des données et ainsi améliorer les performances d'une application utilisant une base de données.

la création d'index peut être réalisée pour un ou plusieurs attributs fréquemment consultés. La commande à utiliser est la suivante :

CREATE [UNIQUE] [NOCOMPRESS] INDEX <nom_index>
ON <nom_table> (<nom_colonne>, [nom_colonne]…)

- Un index unique permet de spécifier qu'une ou plusieurs colonnes doivent contenir des valeurs uniques à chaque enregistrement.

N.B: Le SGBDR crée systématiquement un index chaque fois que l'on pose une clef primaire (PRIMARY KEY) ou une contrainte d'unicité (UNIQUE) sur une table.

- Plus les données d'un index sont petites, plus l'index est efficace. Plus ces mêmes données sont longues ou composées d'un grand nombre de colonnes et moins l'index est efficace

- Un index ralentit les rafraîchissements de la base (conséquence de la mise à jour de l'arbre ou des bitmaps). En revanche il accélère les accès

- Il est conseillé de créer des index sur des colonnes (majoritairement des clés étrangères) utilisées dans les clauses de jointures

- Les index sont pénalisants lorsqu'ils sont définis sur des colonnes très souvent modifiées ou si la table contient peu de lignes.

Clé primaire vs. Clé unique

- La clé primaire n'acceptera pas les valeurs NULL alors que la clé unique peut accepter les valeurs NULL.
- Une table ne peut avoir qu'une clé primaire alors qu'il peut y avoir plusieurs clés uniques sur une table

Les Vues:

- Une vue est une représentation logique d'une requête utilisateur réalisée sur une ou plusieurs tables. C'est une table virtuelle basée sur les données des tables de la base. Elle se comporte ensuite comme si elle était une table du moins pour ce qui est de son interrogation.

- Une vue a un rôle de sécurité -- restreindre l'accès à certaines colonnes ou certaines lignes d'une table pour certains utilisateurs (confidentialité)

- Elle permet certains contrôles d'intégrité lorsqu'elles sont utilisées pour les mises à jour (la clause **With check option**)

- Elle constitue un élément essentiel d'optimisation de performance.

- Lorsque le système évalue une requête formulée sur une vue, il combine la requête de l'utilisateur et la requête de définition de la vue pour obtenir le résultat.

CREATE VIEW *nom_vue [liste_attribut]*
AS *(**SELECT** ...) [**WITH CHECK OPTION**];*

De manière similaire à une relation de la base, une vue peut être:

- consultée via une requête d'interrogation ;

- décrite structurellement grâce à la commande **DESCRIBE** ou en interrogeant la table système **ALL_VIEWS** ;

- détruite par l'ordre **DROP VIEW** <nom_vue>.

Lorsqu'une vue est utilisée pour effectuer des opérations de mise à jour, elle est soumise à des contraintes fortes. En effet pour que les mises à jour, à travers une vue, soient automatiquement répercutées sur la relation de base associée, il faut impérativement que :

- la vue ne comprenne pas de clause **GROUP BY**.

- la vue n'ait qu'une seule relation dans la clause **FROM**. Ceci implique que dans une vue multi-relation, les jointures soient exprimées de manière imbriquée.

La clause **WITH CHECK OPTION** permet de spécifier que les tuples insérés ou mis à jour via la vue doivent satisfaire aux conditions de la requête SELECT définissant la vue. Ceci garantit que les tuples insérés ou modifiés via la vue lui appartiennent bien.

Ex:

1 - CREATE VIEW Sal_1500_3000 AS (select * from EMPLOYEE where Salary BETWEEN 1500 AND 3000);

insert into sal_1500_3000(EMP_ID, EMP_NAME, EMP_NO, HIRE_DATE, JOB, SALARY, DEPT_ID, MNG_ID)

values(7799, 'Sebastien','E7799','09-JUN-81','MANAGER','3500','30','7839');

=> Cet enregistrement sera inséré dans la table EMPLOYEE mais pas dans la vue vu qu'il ne satisfait pas la condition.

2- CREATE VIEW Sal_3000_5000 AS (select * from EMPLOYEE where Salary BETWEEN 3000 AND 5000) WITH CHECK OPTION ;

insert into sal_3000_5000(EMP_ID, EMP_NAME, EMP_NO, HIRE_DATE, JOB, SALARY, DEPT_ID, MNG_ID)

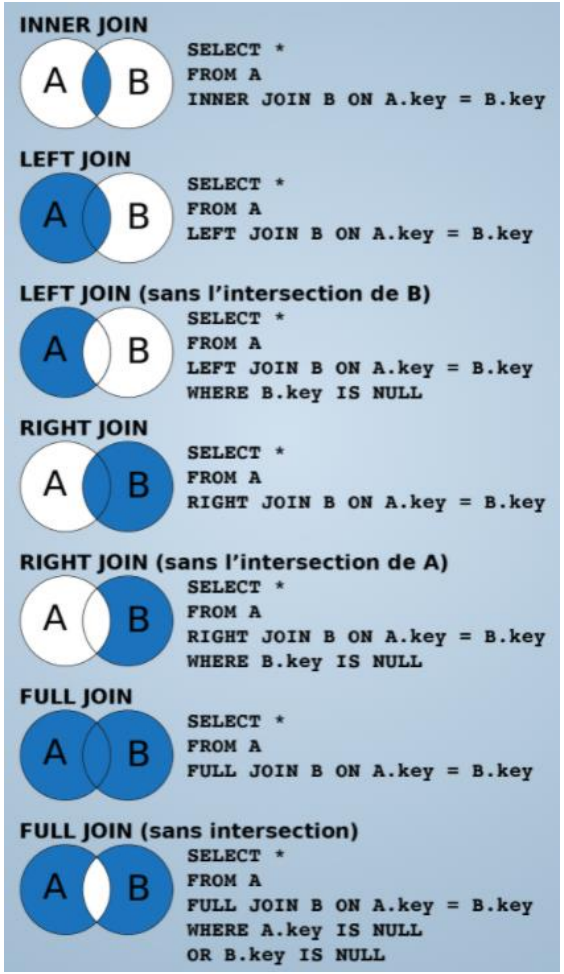
Values (7800, 'Sebastien','E7800','09-JUN-81','MANAGER','2500','30','7839');

=> Cet enregistrement ne sera pas inséré dans la table EMPLOYEE ni dans la vue vu qu'il ne satisfait pas la condition (Retourne une err)

Les Jointures:

- Les jointures en SQL permettent d'associer plusieurs tables dans une même requête.

- 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



```
Select Emp.*, Dept.Dept_ID, Dept.Dept_Name from
employee_join Emp left outer join department_join Dept
on emp.dept_id = dept.dept_id;
```

Employee_join

	EMP_ID	EMP_NAME	EMP_NO	HIRE_DATE	IMAGE	JOB	SALARY	DEPT_ID	MNG_ID
1	7839	KING	E7839	17-NOV-81	(null)	PRESIDENT	5000	70	(null)
2	7566	JONES	E7566	02-APR-81	(null)	MANAGER	2975	20	7839
3	7902	FORD	E7902	03-DEC-81	(null)	ANALYST	3000	20	7566
4	7369	SMITH	E7369	17-DEC-80	(null)	CLERK	800	20	7902
5	7698	BLAKE	E7698	01-MAY-81	(null)	MANAGER	2850	30	7839
6	7499	ALLEN	E7499	20-FEB-81	(null)	SALESMAN	1600	30	7698

Department_join

	DEPT_ID	DEPT_NAME	DEPT_NO	LOCATION
1	10	ACCOUNTING	D10	NEW YORK
2	20	RESEARCH	D20	DALLAS
3	30	SALES	D30	CHICAGO
4	40	OPERATIONS	D40	BOSTON
5	50	IT	D50	BOSTON

	EMP_ID	EMP_NAME	EMP_NO	HIRE_DATE	IMAGE	JOB	SALARY	DEPT_ID	MNG_ID	DEPT_ID_1	DEPT_NAME
1	7839	KING	E7839	17-NOV-81	(null)	PRESIDENT	5000	70	(null)	(null)	(null)
2	7566	JONES	E7566	02-APR-81	(null)	MANAGER	2975	20	7839	20	RESEARCH
3	7902	FORD	E7902	03-DEC-81	(null)	ANALYST	3000	20	7566	20	RESEARCH
4	7369	SMITH	E7369	17-DEC-80	(null)	CLERK	800	20	7902	20	RESEARCH
5	7698	BLAKE	E7698	01-MAY-81	(null)	MANAGER	2850	30	7839	30	SALES
6	7499	ALLEN	E7499	20-FEB-81	(null)	SALESMAN	1600	30	7698	30	SALES

Select Emp.*, Dept.Dept_ID, Dept.Dept_Name **from**

employee_join Emp **Right outer join** department_join Dept

on emp.dept_id = dept.dept_id;

Employee_join

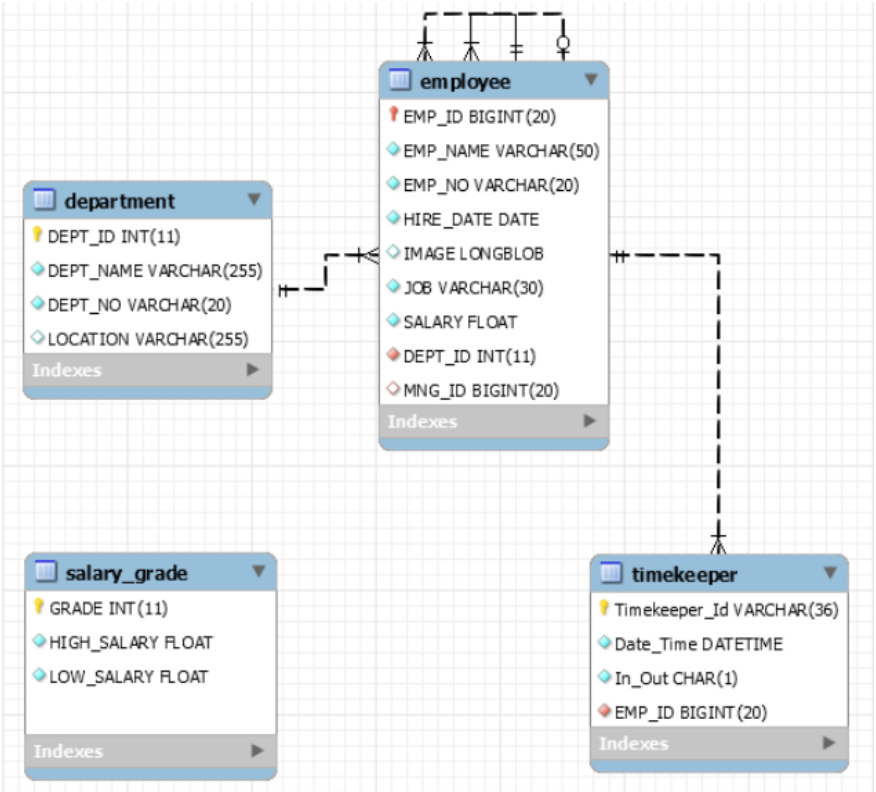
	EMP_ID	EMP_NAME	EMP_NO	HIRE_DATE	IMAGE	JOB	SALARY	DEPT_ID	MNG_ID
1	7839	KING	E7839	17-NOV-81	(null)	PRESIDENT	5000	70	(null)
2	7566	JONES	E7566	02-APR-81	(null)	MANAGER	2975	20	7839
3	7902	FORD	E7902	03-DEC-81	(null)	ANALYST	3000	20	7566
4	7369	SMITH	E7369	17-DEC-80	(null)	CLERK	800	20	7902
5	7698	BLAKE	E7698	01-MAY-81	(null)	MANAGER	2850	30	7839
6	7499	ALLEN	E7499	20-FEB-81	(null)	SALESMAN	1600	30	7698

Department_join

	DEPT_ID	DEPT_NAME	DEPT_NO	LOCATION
1	10	ACCOUNTING	D10	NEW YORK
2	20	RESEARCH	D20	DALLAS
3	30	SALES	D30	CHICAGO
4	40	OPERATIONS	D40	BOSTON
5	50	IT	D50	BOSTON

	EMP_ID	EMP_NAME	EMP_NO	HIRE_DATE	IMAGE	JOB	SALARY	DEPT_ID	MNG_ID	DEPT_ID_1	DEPT_NAME
1	7566	JONES	E7566	02-APR-81	(null)	MANAGER	2975	20	7839	20	RESEARCH
2	7902	FORD	E7902	03-DEC-81	(null)	ANALYST	3000	20	7566	20	RESEARCH
3	7369	SMITH	E7369	17-DEC-80	(null)	CLERK	800	20	7902	20	RESEARCH
4	7698	BLAKE	E7698	01-MAY-81	(null)	MANAGER	2850	30	7839	30	SALES
5	7499	ALLEN	E7499	20-FEB-81	(null)	SALESMAN	1600	30	7698	30	SALES
6	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	50	IT
7	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	40	OPERATIONS
8	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	10	ACCOUNTING

Exercice:


employee
EMP_ID BIGINT(20)
EMP_NAME VARCHAR(50)
EMP_NO VARCHAR(20)
HIRE_DATE DATE
IMAGE LONGBLOB
JOB VARCHAR(30)
SALARY FLOAT
DEPT_ID INT(11)
MNG_ID BIGINT(20)
Indexes
department
DEPT_ID INT(11)
DEPT_NAME VARCHAR(255)
DEPT_NO VARCHAR(20)
LOCATION VARCHAR(255)
Indexes
salary_grade
GRADE INT(11)
HIGH_SALARY FLOAT
LOW_SALARY FLOAT
Indexes
timekeeper
Timekeeper_Id VARCHAR(36)
Date_Time DATETIME
In_Out CHAR(1)
EMP_ID BIGINT(20)
Indexes

Thank you!