Bases de données relationnelles

-

SQL

2023-2024

NEGRE Elsa

Sources : G. Gardarin, R. Grin, M. Manouvrier, T-L. Nguyen

SGBD

- Principaux composants :
  - Système de gestion de fichiers
  - Gestionnaire de requêtes
  - Gestionnaire de transactions

- Principales fonctionnalités :
  - Contrôle de la redondance d'information
  - Partage des données (car plusieurs utilisateurs en même temps)
  - Gestion des autorisations d'accès
  - Vérifications des contraintes d'intégrité
  - Sécurité et reprise sur panne

Abstraction des données (3 niveaux)

- **Niveau interne ou physique** :
  - plus bas niveau
  - indique **comment** (avec quelles structures de données) sont stockées physiquement les données
- **Niveau logique ou conceptuel** :
  - décrit par un **schéma conceptuel** *ou logique*
  - indique quelles sont les données stockées et quelles sont leurs relations **indépendamment de l'implantation physique**
- **Niveau externe ou vue** :
  - **propre à chaque utilisateur**
  - décrit par un ou plusieurs **schémas externes**

SQL

Structured Query Language (normalisé en 1986)

- **SQL2/SQL92** : standard adopté en 1992
- **SQL3/SQL99** : extension de SQL2 avec "gestion" d'objets, déclencheurs …
- **SQL2003**: auto-incrémentation des clés, colonne calculée, prise en compte de XML, …
- **SQL2008**: correction de certains défauts et manques (fonctions, types, curseurs…)

SQL :

- **Langage de Manipulation de Données (DML) :** interroger et modifier les données de la base
- **Langage de Définition de Données (DDL) :** définir le schéma de la base de données
- **Langage de contrôle d'accès aux données (DCL)** : pour définir les privilèges d'accès des utilisateurs
- **Langage de contrôle des transactions (TCL)** : pour gérer les transactions.

Bibliographie

- *SQL2 - Application à Oracle, Access et RDB*
  Pierre DELMAL, 2ème Edition, De Boeck Université, 1998
  BU: 005.74 SQL
- *SQL Pour Oracle (avec exercices corrigés)*
  Christian Soutou, Eyrolles, 2005 – BU: 005.72 SOU
- *Initiation à SQL (cours et exercices corrigés)*
  Philip J. Pratt, Eyrolles,2001 – BU : 005.72 SQL
- *SQL (cours et exercices corrigés)*
  Frédéric Brouad et Christian Soutou, Coll. Synthex, Pearson Education, 2012 – BU : 005.72 SQL
- *Oracle PL/SQL - Précis & concis*
  Steven Feuerstein, Bill Pribyl et Chip Dawes, O 'Reilly, 2000

DML

Select : forme générale

- **SELECT** <liste de projection>
- **FROM** <liste de tables>
- [**WHERE** <critère de jointure> AND <critère de restriction>]
- [**GROUP BY** <attributs de partitionnement>]
- [**HAVING** <critère de restriction>]

- Restrictions
  - arithmétique (=, <, >, ≠, ≥, ≤)
  - textuelle (**LIKE**)
  - sur intervalle (**BETWEEN**) ou sur liste (**IN**)
- Possibilité de blocs imbriqués par :
  - **IN**, **EXISTS**, **NOT EXISTS**, **ALL**, **SOME**, **ANY**

Projection

Syntaxe SQL :

SELECT [UNIQUE[1]] *liste_attributs*[2] FROM *Table* ;

Équivalent AR :

$\Pi_{liste\_attributs}$ R(*Table*)

[1] Permet d'éliminer les doublons (on trouvera aussi DISTINCT)

[2] On peut mettre une étoile * pour demander tous les attributs
On peut renommer un attribut en ajoutant *AS NomAttribut*

Projection - Exemples

Soit la relation *Étudiants*(*#num*, *nom*, *prénom*, *âge*, *ville*, *CodePostal*)

Afficher toute la relation Étudiant.

SELECT * FROM *Étudiants*;

Donner les noms, les prénoms et les âges de tous les étudiants.

SELECT *nom*, *prénom*, *age* FROM *Étudiants*;

Donner les numéros des étudiants dans une colonne nommée Numéro.

SELECT *#num* AS *Numéro* FROM *Étudiants*;

Sélection

Syntaxe SQL :
SELECT * FROM *table* WHERE *condition;*

Équivalent AR :
$\sigma_{condition}$ R(*Table*)

La condition peut formée sur des noms d'attributs ou des constantes avec

- des opérateurs de comparaison : =, >, <, <=, >=, <>[1]
- des opérateurs logiques : AND, OR, NOT
- des opérateurs : IN, BETWEEN+AND, LIKE, EXISTS, IS
- _ qui remplace un caractère et % qui remplace une chaîne de caractères

[1] La différence est parfois notée !=

Sélection – Exemples

Sur la relation *Étudiants*(*#Num*, *Nom*, *Prénom*, *Age*, *Ville*, *CodePostal*)

Quels sont tous les étudiants âgés de 20 ans ou plus ?

SELECT * FROM *Étudiants* WHERE (*Age* >= 20);

Quels sont tous les étudiants âgés de 19 à 23 ans ?

SELECT * FROM *Étudiants* WHERE *Age* IN (19, 20, 21, 22, 23);
SELECT * FROM *Étudiants* WHERE *Age* BETWEEN 19 AND 23;

Quels sont tous les étudiants habitant dans les Vosges ?

SELECT * FROM *Étudiant* WHERE *CodePostal* LIKE '88%' ;

Quels sont tous les étudiants dont la ville est inconnue/connue ?

SELECT * FROM *Étudiants* WHERE *Ville* IS NULL ;
SELECT * FROM *Étudiants* WHERE *Ville* IS NOT NULL ;

Prédicats du WHERE

exp1 = exp2
exp1 != exp2
exp1 > exp2
exp1 < exp2
exp1 <= exp2
exp1 >= exp2
exp1 BETWEEN exp2 AND exp3
exp1 LIKE exp2
exp1 IN (exp2, exp3, …)
exp1 NOT IN (exp2, exp3, …)
exp1 IS NULL
exp1 IS NOT NULL

exp *op* ANY/SOME (SELECT …)
exp *op* ALL (SELECT …)
avec *op* tel que =, !=, <, > ...

exp IN (SELECT …)
exp NOT IN (SELECT …)

Exemples de résultats

Relation

L..	enseignant_id (...	departement_id ...	nom (varchar)	prenom ...	grade (...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
3	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr
4	4	1	LIMAM	Medhi	ATER			
5	5	5	MyTaylor	IsRich	Vacataire			
6	6	1	RIGAUX	Philippe	PROF			
7	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
8	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

Résultat de la sélection *SELECT * FROM Enseignant WHERE Grade='MCF'* :

L..	enseignant_id...	departement_id...	nom (varchar)	prenom...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr

Résultat de la projection *SELECT Nom, Prenom FROM Enseignant* :

L..	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	NAIJA	Yosr
3	BAHRI	Afef
4	LIMAM	Medhi
5	MyTaylor	IsRich
6	RIGAUX	Philippe
7	CHAKHAR	Salem
8	MURAT	Cécile

Résultat de la requête *SELECT Nom, Prenom FROM Enseignant WHERE Grade='MCF'* :

Ligne	nom (varchar)	prenom...
1	MANOUVRIER	Maude
2	MURAT	Cécile

Prédicats du WHERE (2)

Clause EXISTS :

- Retourne VRAI si au moins un nuplet est renvoyé par la requête
- FAUX si aucun nuplet n'est retourné.
- La valeur NULL n'a aucun effet sur le booléen résultat

```
SELECT Nom, Prénom
 FROM Enseignant E
  WHERE NOT EXISTS
     ( SELECT *
        FROM Reservation_Salle S
          WHERE  S.Enseignant_ID = E.Enseignant_ID
      );
```

Produit Cartésien

Syntaxe SQL :

SELECT *

FROM $table_1$ [$Alias_1$], ..., $table_n$ [$Alias_n$],

Équivalent AR :

$Table_1 \times ... \times Table_n$

θ-Jointure

Syntaxe SQL :

SELECT *

FROM $table_1$ [$Alias_1$], ..., $table_n$ [$Alias_n$],

WHERE *condition*;

Possibilité de Renommage des tables

Équivalent AR :

$Table_1 \bowtie_\theta \ldots \bowtie_\theta Table_n$

Autre Syntaxe :

SELECT * FROM $table_1$ INNER JOIN $table_2$ ON *condition*;

Exemples de Jointure

La relation *Enseignant* :

Ligne	enseignant_id ...	departement_id ...	nom (varchar)	prenom ...	grade ...	telephone ...	fax ...	email (varchar)
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvrier@lmasade.dauphine.fr
2	4	1	LIMAM	Medhi	ATER			
3	5	5	MyTaylor	IsRich	Vacataire			
4	6	1	RIGAUX	Philippe	PROF			
5	8	1	MURAT	Cécile	MCF			murat@lamsade.dauphine.fr
6	7	1	CHAKHAR	Salem	ATER			chakhar@lamsade.dauphine.fr
7	2	1	NAIJA	Yosr	Moniteur			naija@lmasade.dauphine.fr
8	3	1	BAHRI	Afef	Moniteur			bahri@lmasade.dauphine.fr

La relation *Departement* :

Ligne	departement_id ...	nom_departement...
1	1	INFO
2	2	MATHS
3	3	GESTION
4	4	ECO
5	5	LANGUES
6	7	COMM
7	8	OPTION

*SELECT * FROM Enseignant e, Departement d WHERE e.Departement_ID=d.Departement_ID* :

L...	enseignant_id...	departement_id ...	nom (varchar)	prenom...	grade ...	telephone ...	fax ...	email (va...	departement_id...	nom_departement ...
1	1	1	MANOUVRIER	Maude	MCF	4185	4091	manouvri...	1	INFO
2	4	1	LIMAM	Medhi	ATER				1	INFO
3	5	5	MyTaylor	IsRich	Vacata...				5	LANGUES
4	6	1	RIGAUX	Philippe	PROF				1	INFO
5	8	1	MURAT	Cécile	MCF			murat@la...	1	INFO
6	7	1	CHAKHAR	Salem	ATER			chakhar...	1	INFO
7	2	1	NAIJA	Yosr	Moniteur			naija@lm...	1	INFO
8	3	1	BAHRI	Afef	Moniteur			bahri@lm...	1	INFO

Personnel

Nom_Employé	Ville
Tom	Marseille
Jerry	Paris
Alex	Limoges
Marthe	Perpignan

Employé

Nom_Employé	Filiale	Salaire
Tom	SUD_EST	10000
Jerry	IDF	25000
Sophie	IDF	15000
Marthe	SUD_OUEST	12000

Personnel]∞ *Employé*

Nom_Employé	Ville	Filiale	Salaire
Tom	Marseille	SUD_EST	10000
Jerry	Paris	IDF	25000
Alex	**Limoges**	**NULL**	**NULL**
Marthe	Perpignan	SUD_OUEST	12000

Personnel ∞[*Employé*

Nom_Employé	Ville	Filiale	Salaire
Tom	Marseille	SUD_EST	10000
Jerry	Paris	IDF	25000
Sophie	**NULL**	**IDF**	**15000**
Marthe	Perpignan	SUD_OUEST	12000

Jointures par requêtes imbriquées

Une jointure peut aussi être effectuée à l'aide d'une sous-requête.

SELECT *

FROM *Stock*

WHERE *#prod* IN (SELECT #prod
FROM *Produit*)

Sous-requête imbriquée

Principe : Le mot-clef "IN" permet ici de sélectionner les tuples *#prod* appartenant à la sous-requête.

- La sous-requête ne doit retourner qu'une colonne !
- Les tables de sous-requêtes ne sont pas visibles depuis l'extérieur

Union, Intersection et Différence

$Table_1 \cup Table_2$:

SELECT *liste_attributs* FROM $table_1$
UNION
SELECT *liste_attributs* FROM $table_2$;

$Table_1 \cap Table_2$:

SELECT *liste_attributs* FROM $table_1$
INTERSECT
SELECT *liste_attributs* FROM $table_2$;

$Table_1 - Table_2$:

SELECT *liste_attributs* FROM $table_1$
EXCEPT
SELECT *liste_attributs* FROM $table_2$;

Union, Intersection et Différence - Exemples

```
SELECT Nom, Prenom FROM Enseignant
UNION
SELECT Nom, Prenom FROM Etudiant ;
```

Ligne	nom (varc...	prenom...
1	BAHRI	Afef
2	CHAKHAR	Salem
3	Debécé	Aude
4	GAMOTTE	Albert
5	HIBULAIRE	Pat
6	LIMAM	Medhi
7	MANOUVRIER	Maude
8	MURAT	Cécile
9	MyTaylor	IsRich
10	NAIJA	Yosr
11	ODENT	Jamal
12	RASLATABLE	Deborah
13	RIGAUX	Philippe

```
SELECT Nom, Prenom FROM Enseignant
EXCEPT
SELECT Nom, Prenom FROM Etudiant ;
```

Ligne	nom (varchar)	prenom ...
1	MANOUVRIER	Maude
2	MURAT	Cécile
3	MyTaylor	IsRich
4	RIGAUX	Philippe

```
SELECT Nom, Prenom FROM Enseignant
INTERSECT
SELECT Nom, Prenom FROM Etudiant;
```

Ligne	nom (varchar)	prenom...
1	BAHRI	Afef
2	CHAKHAR	Salem
3	LIMAM	Medhi
4	NAIJA	Yosr

Division

Livre(ISBN, Titre, Editeur)

Emprunt(EmpruntID,ISBN, DateEmprunt,EtudiantID)

Etudiant(EtudiantID,Nom, Prenom)

« Quels livres ont été empruntés par tous les étudiants? »

{t.Titre / Livre(t) ∧ ¬ [∃ u Etudiant (u) ∧
¬ (∃ v Emprunt(v) ∧
(v.Etudiant_ID=u.Etudiant_ID) ∧
(v.ISBN=t.ISBN)
)
]
}

Il n'y a pas de mot-clé "quel que soit " en SQL2

```
SELECT t.Titre FROM Livre t WHERE NOT EXISTS
    ( SELECT * FROM Etudiant u WHERE NOT EXISTS
        ( SELECT * FROM Emprunt v
            WHERE u.EtudiantID=v.EtudiantID AND v.ISBN=t.ISBN
        )
    );
```

Division - Exemple

La relation *Enseignement* :

L..	enseignement_id ...	departement_id ...	intitule (varchar)	description (varchar)
1	1	1	Bases de Données	Niveau Licence : Modélisation E/A et UML, Modèle relationnel, Algèbre Relation...
2	2	1	Mise à Niveau Informatique	Pour les étudiants de GMI entrant directement en IUP2: Architecture, Algorithm...
3	3	1	Mise à Niveau Bases de Données	Pour les étudiants de DESS ID ou DEA127 - Programme Licence et Maîtrise en ...
4	4	5	Anglais	

La relation *Inscription* :

Ligne	etudiant_id (int4)	enseignement_id (int4)	departement_id (int4)	date_inscription (date)
1	1	1	1	2004-02-25
2	1	2	1	2004-07-22
3	3	2	1	2004-07-22
4	5	2	1	2004-07-22
5	4	2	1	2004-07-22
6	1	3	1	2004-07-22
7	1	4	5	2004-07-22
8	2	4	5	2004-07-22

$\Pi_{Etudiant\_ID,\ Enseignement\_ID,\ Departement\_ID}$ (*Inscription*) $\div$ $\Pi_{Enseignement\_ID,\ Departement\_ID}$ (*Enseignement*) :

Ligne	etudiant_id (int4)
1	1

Tri de résultats

Syntaxe :

Cette clause se place derrière la clause WHERE

ORDER BY attribut [ordre] [, attribut [ordre] ...]

On peut préciser un ordre croissant ASC ou décroissant DESC.

Exemple

Trier Stock par numéro de produit croissant et par quantité décroissante

```
SELECT *
FROM Stock
WHERE qte > 0
ORDER BY #prod ASC, qte DESC
```

Agrégation des résultats

Il est possible d'utiliser des fonctions *f* d'agrégation dans le résultat d'une sélection.

Syntaxe :

SELECT *f* ([ALL | DISTINCT] expression)
FROM ...

où *f* peut être

COUNT	nombre de tuples
SUM	somme des valeurs d'une colonne
AVG	moyenne des valeurs d'une colonne
MAX	maximum des valeurs d'une colonne
MIN	minimum des valeurs d'une colonne

Pour COUNT, on peut aussi utiliser COUNT(*)
Seul COUNT prend en compte les valeurs à NULL.

Fonctions d'agrégation - Exemples

Résultats (de Pierre)

Matière	*Coef*	*Note*
Maths	4	15
Sc Nat	3	9
Sc Phy	3	12
Français	2	13
Sc Hum	2	11
Anglais	1	10
Sport	1	12

Quelle est la meilleure note ?

SELECT MAX(*Note*) FROM *Résultats*

→ 15

Quelle est la plus mauvaise note ?

SELECT MIN(*Note*) FROM *Résultats*

→ 9

Quelle la somme pondérée des notes ?

SELECT SUM(*Note*Coef*) FROM *Résultats* → 193

Quelle est la moyenne (pondérée) de Pierre ?

SELECT SUM(*Note*Coef*)/*Sum*(*Coef*) FROM *Résultats*

→ 12,06

Dans combien de matières Pierre a-t-il eu plus de 12 ?

SELECT COUNT(*) FROM *Résultats* WHERE *Note* > 12

→ 2

Partitionnement des résultats

Syntaxe

GROUP BY *liste_attributs*

HAVING *condition avec fonction*

Cette clause regroupe les résultats par valeur selon la condition

Dans l'ordre, on effectue

- la sélection SELECT
- le partitionnement GROUP BY
- on retient les partitions intéressantes HAVING
- on trie avec ORDER BY.

Partitionnement des résultats - Exemples

Résultats (de Pierre)

Matière	*Coef*	*Note*
Maths	4	15
Sc Nat	3	9
Sc Phy	3	12
Français	2	13
Sc Hum	2	11
Anglais	1	10
Sport	1	12

Quelle est la note moyenne pour chaque coefficient ?

```
SELECT coef, Avg(note) as Moyenne
FROM Résultats
GROUP BY coef;
```

Coef	*Moyenne*
1	11
2	12
3	10.5
4	15

Quels sont les coefficients auxquels participe une seule matière ?

```
SELECT coef
FROM Résultats
GROUP BY coef
HAVING count(*)=1;
```

Coef
4

Ajouts de tuples dans une relation

Syntaxe :

Pour insérer un tuple complètement spécifié :

INSERT INTO *Table* VALUES (val_1,..., val_n);

Pour insérer un tuple incomplètement spécifié :

INSERT INTO *Table* (*liste_attributs*)VALUES (val_1,..., val_n);

On peut insérer un tuple à partir d'une relation ayant le même schéma.

INSERT INTO *Table*

SELECT *

FROM ...

Exemples d'insertion

Sur les relations *Étudiants* (*#Num*, *Nom*, *Prénom*, *Age*, *Ville*, *CodePostal*)
ClubThéâtre(#Num, *Nom*, *Prénom*)

Ajouter l'étudiant Sylvain HEBON, 21 ans, habitant Nancy avec le numéro 634.

INSERT INTO *Étudiants*
VALUES (634, 'HEBON', 'Sylvain', 'Nancy', '54000', 21);

Ajouter tous les étudiants Vosgiens dans le Club de Théâtre

INSERT INTO *ClubThéâtre*
SELECT *#Num, Nom, Prénom*
FROM *Étudiants*
WHERE *CodePostal* LIKE '88%';

Modification de tuples

Syntaxe :

UPDATE *Table*

SET $attribut_1 = expr_1$, ..., $attribut_n = expr_n$

FROM ...

WHERE ...

Les expressions peuvent être

- une constante
- une valeur NULL
- une clause SELECT

Mise à jour - Exemple

Sur la relation *Étudiants* (*#Num*, *Nom*, *Prénom*, *Age*, *Ville*, *CodePostal*)

Augmenter d'un an l'age de tous les étudiants.

```
UPDATE Étudiants
SET Age = Age + 1;
```

On a appris que tous les étudiants de Bar-le-Duc ont déménagé à Nancy.

```
UPDATE Étudiants
SET Ville = 'Nancy', CodePostal = '54000'
WHERE Ville = 'Bar-Le-Duc';
```

Suppression de Tuples

Syntaxe :

DELETE FROM *Table*

[WHERE *condition*]

Remarque :

Si on supprime tous les tuples d'une relation, le schéma de relation existe toujours !

Exemple :

Retirer de la liste tous les étudiants de plus de 22 ans.

DELETE FROM *Étudiants*

WHERE *Age* > 22;

DDL

Types SQL

Type	*Taille (Octets)*	*Signification*
Int	4	Valeur Entière
SmallInt	2	Valeur Entière
TinyInt	1	Valeur Entière
float	4/8	Valeur Décimale
Char (longueur)	Fixe (max 255)	Chaîne de caractères
VarChar (longueur)	Var (max 255)	Chaîne de caractères
Text	Var (max 2^{31}-1)	Chaîne de caractères
Image	Var (max 2^{31}-1)	Chaîne binaire

Type	*Taille (Octets)*	*Signification*
Bit	1	Valeur Binaire
Binary	Fixe (max 255)	Chaîne binaire
Varbinary	Var (max 255)	Chaîne binaire
Money	8	Valeur en $
DateTime	2×4 octets	Nb Jours depuis 1/1/1900 + Heure

Création de table

Syntaxe :

CREATE TABLE *nomTable* (

Attribut *Domaine* [*Contraintes* ...],

...

Attribut *Domaine* [*Contraintes* ...],

[*Contraintes* ...]

)

Création de table - Exemple

Créer la table *Stock1 (Pièce, NbP, Fournisseur)*

```
CREATE TABLE Stock1 (
        Pièce           VARCHAR(20) NOT NULL,
        NbP             INT,
        Fournisseur     CHAR(20) NOT NULL,
        PRIMARY KEY (Pièce, Fournisseur)
)
```

Création de table – Exemple 2

```
CREATE TABLE Enseignant
(
 Enseignant_ID                  integer,
 Departement_ID      integer NOT NULL,
 Nom                 varchar(25) NOT NULL,
 Prenom                         varchar(25) NOT NULL,
 Grade                          varchar(25)
 CONSTRAINT CK_Enseignant_Grade
 CHECK (Grade IN ('Vacataire', 'Moniteur','ATER', 'MCF', 'PROF')),
 Telephone                      varchar(10) DEFAULT NULL,
 Fax                            varchar(10) DEFAULT NULL,
 Email                          varchar(100) DEFAULT NULL,
 CONSTRAINT PK_Enseignant PRIMARY KEY (Enseignant_ID),
 CONSTRAINT "FK_Enseignant_Departement_ID"
   FOREIGN KEY (Departement_ID)
    REFERENCES Departement (Departement_ID)
     ON UPDATE RESTRICT ON DELETE RESTRICT
);
```

Contrainte de domaine

Définition de la clé primaire

Définition d'une clé étrangère

Création de table – Exemple 3

```
CREATE TABLE Reservation
(
 Reservation_ID     integer,
 Batiment           varchar(1) NOT NULL,
 Numero_Salle       varchar(10) NOT NULL,
 Enseignement_ID    integer NOT NULL,
 Departement_ID     integer NOT NULL,
 Enseignant_ID      integer NOT NULL,
 Date_Resa          date NOT NULL DEFAULT CURRENT_DATE,
 Heure_Debut                time NOT NULL DEFAULT CURRENT_TIME,
 Heure_Fin          time NOT NULL DEFAULT '23:00:00',
 Nombre_Heures              integer NOT NULL,
 CONSTRAINT PK_Reservation PRIMARY KEY (Reservation_ID),
 CONSTRAINT "FK_Reservation_Salle"  FOREIGN KEY (Batiment,Numero_Salle) REFERENCES Salle
     (Batiment,Numero_Salle)  ON UPDATE RESTRICT ON DELETE RESTRICT,
 CONSTRAINT "FK_Reservation_Enseignement" FOREIGN KEY (Enseignement_ID,Departement_ID)
     REFERENCES Enseignement (Enseignement_ID,Departement_ID) ON UPDATE RESTRICT ON
     DELETE RESTRICT,
 CONSTRAINT "FK_Reservation_Enseignant" FOREIGN KEY (Enseignant_ID) REFERENCES Enseignant
     (Enseignant_ID) ON UPDATE RESTRICT ON DELETE RESTRICT,
 CONSTRAINT CK_Reservation_Nombre_Heures CHECK (Nombre_Heures >=1),
 CONSTRAINT CK_Reservation_HeureDebFin
   CHECK (Heure_Debut < Heure_Fin)
);
```

Modification et Suppression de Relation

Modification de Schéma de relation (Syntaxe variable !)

Exemple pour Oracle v6 :

ALTER TABLE *Table*

[ADD (définition_attribut | Contrainte), [définition_attribut | Contrainte] ...)]

[MODIFY (définition_attribut [, définition_attribut]...)]

[DROP CONSTRAINT contrainte]

Suppression complète d'une relation (et de son schéma) :

DROP TABLE *Table*;

Attention, toutes les données de la table sont perdues !

Vues et Relations temporaires

Une vue est une **relation non stockée** dans la base de données mais **recalculée** à chaque utilisation.

Syntaxe :
CREATE VIEW *NomVue* AS
Requête_de_définition[1]

Exemple :

```
CREATE VIEW Personnes_Âgées AS
SELECT *
FROM Personnes
WHERE Age > 70;
```

La suppression s'effectue avec DROP VIEW *NomVue*;

[1] La requête ne doit pas contenir de tris (ORDER BY).

Administration de Base de Données

Création d'une base de données

Syntaxe :
CREATE DATABASE *NomBdd;*

Destruction totale d'une base de données

Syntaxe :
DROP DATABASE *NomBdd;*

Triggers

- Les triggers (déclencheurs en français) sont compilés et enregistrés dans le dictionnaire des données de la base et ils sont le plus souvent écrits dans le même langage.

- Leur exécution est déclenchée automatiquement par des événements liés à des actions sur la base.

- Les événements déclencheurs peuvent être les commandes LMD insert, update, delete ou les commandes LDD create, alter, drop.

Triggers (2)

- La syntaxe :

```
CREATE [OR REPLACE] TRIGGER nom-trigger
{BEFORE | AFTER } {INSERT | DELETE | UPDATE [OF col1, col2,... ]}
ON {nom-table | nom-vue }
[REFERENCING ...]
[FOR EACH ROW]
[WHEN condition ]
bloc PL/SQL
```

- Pour supprimer un trigger :

```
drop trigger nomTrigger;
```

Trigger - Exemple

- Une table cumul sert à enregistrer le cumul des augmentations dont ont bénécié les employés d'une entreprise.

- Mise à jour automatique d' une table cumul qui totalise les augmentations de salaire de chaque employé.

```
CREATE OR REPLACE TRIGGER totalAugmentation
AFTER UPDATE OF sal ON emp
FOR EACH ROW
begin
update cumul
set augmentation = augmentation + :NEW.sal - :OLD.sal
where matricule = :OLD.matr;
end;
```

- Il faudra aussi créer un autre trigger qui ajoute une ligne dans la table cumul quand un employé est créé :

```
CREATE OR REPLACE TRIGGER creetotal
AFTER INSERT ON emp
for each row
begin
insert into cumul (matricule, augmentation)
values (:NEW.matr, 0);
end;
```

Trigger – Exemple – PostgreSQL (1)

```
CREATE OR REPLACE FUNCTION FunctionTriggerReservation()
RETURNS trigger AS
' DECLARE
  resa Reservation.Reservation_ID%TYPE;
  BEGIN
   SELECT INTO resa Reservation_ID
    FROM Reservation
     WHERE  …
  IF FOUND THEN RAISE EXCEPTION "Réservation impossible, salle occupée à
     la date et aux horaires demandés";
ELSE RETURN NEW;
  END IF;
END;'
LANGUAGE 'plpgsql';

CREATE TRIGGER InsertionReservation
   BEFORE INSERT ON Reservation
     FOR EACH ROW
       EXECUTE PROCEDURE
          FunctionTriggerReservation();
```

Trigger – Exemple – PostgreSQL (2)

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);

CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Check that empname and salary are given
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.empname;
        END IF;
        -- Who works for us when she must pay for it?
        IF NEW.salary < 0 THEN
            RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
        END IF;
        -- Remember who changed the payroll when
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```