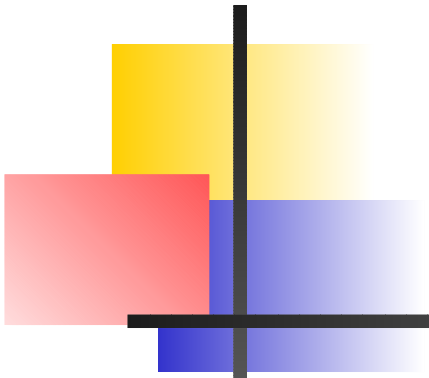

Modèle relationnel Création et modification des relations en SQL

ENT - Clé sql2009

Insertion dans une relation

Pour insérer un tuple dans une relation:

```
insert into Sailors (sid, sname, rating, age)
   values (12, 'Emmanuel', 5,21);
```

Nous pouvons ne pas mettre la liste des attributs si les valeurs sont dans le bon ordre.
C'est mieux (bon style) de mettre les noms des attributs

Nous pouvons aussi insérer le résultat d'une requête. Supposons une relation STUDENTS contenant des informations sur des étudiants ...

```
insert into Sailors (sid, sname,  age)
select S.id, S.name, S.age
from Students S
where S.age >=18;
```

- La table d'étudiant ne contient pas des informations concernant le *rating*. La valeur sera donc NULL.
- Nous pouvons utiliser une valeur par DÉFAUT (pendant la création) pour rating.

Suppression d'un tuple

- Il est possible de supprimer un tuple qui satisfait une *condition*. La condition peut être spécifiée en utilisant tout le pouvoir de SQL.

Supprimer les marins qui n'ont pas de *rating*

```
delete from  Sailors S
where S.rating is NULL;
```

Supprimer les marins qui n'ont pas une réservation pour *Clipper*

```
delete from  Sailors S
where S.name NOT IN (
select R.name
from Reserves R, Boats B
where S.sid = R.sum and R.bid = B.bid
and B.bname = 'Clipper');
```

Modification d'un tuple

- Nous pouvons modifier des valeurs dans une relation.
- Dans la mise à jour, la clause WHERE est appliquée AVANT pour déterminer les tuples qui doivent être changés.
- Si le tuple qui est modifié est aussi utilisé pour déterminer la nouvelle valeur, alors la valeur à droite de l'expression correspond à la valeur ancienne (avant la modification).

Augmenter l'age et diminuer le rating du marin numéro 62.

```
update Sailors  S
set S.age = S.age +1, S.rating = S.rating -1
where S.sid = 62;

update Sailors  S
set S.rating = S.rating -1
where S.rating >= 8;
```

Comment créer les schéma de relation en SQL?

Pour créer un schéma de relation

```
CREATE TABLE  Sailors (
  sid  Number(4),
  sname Varchar2(40),
  rating Number(2),
  age Number(3));
```

Pour ajouter un attribut dans une table existante.

```
ALTER TABLE Sailors ADD COLUMN address varchar2(70);
```

Pour supprimer une table

```
DROP TABLE Sailors;
```

Contraintes d'intégrité

- Le but des contraintes d'intégrité est:
  - Assurer l'implémentation d'une certaine sémantique.
    Par exemple, *un marin peut réserver un seul bateau par jour*
  - Assurer que les mises à jour ne violent pas la cohérence (l'intégrité) de la base
    Par exemple, si le numéro d'identification d'un bateau est unique nous ne pouvons pas insérer un nouveau bateau avec le numéro d'un bateau déjà existant.

Les contraintes de clés primaire

- Une **clé primaire** est un sous ensemble minimal d'attributs pour lequel deux tuples de la base **ne peuvent pas avoir la même valeur**.
- C'est un identificateur unique.
- Pour déterminer les clés nous avons besoin des **dépendances fonctionnelles**
- Une relation peut avoir **plusieurs clés**.
- En SQL nous pouvons déclarer qu'un ensemble d'attributs est une clé en utilisant
  - la contrainte UNIQUE: plusieurs sous ensembles d'attributs peuvent être déclarés comme unique
  - la contrainte PRIMARY KEY: une seule contrainte de ce type par relation.

Les contraintes de clés primaire

Une seule personne peut réserver un bateau dans une journée.

```
create table Reservation(
snum  Number(4),
bnum Number(5),
day Date,
primary key (bnum,day));
```

Une seule personne peut réserver un bateau dans une journée. De plus, une personne peut avoir une seule réservation "en route".

```
create table Reservation(
snum  Number(4),
bnum Number(5),
day Date,
primary key (bnum,day);
unique (snum));
```

Les contraintes de clés primaire

Une autre syntaxe .. directement à coté de l'attribut

```
create table Boats(
bid Number(5) primary key,
bname Varchar2(30),
color Varchar2(10));
```

En donnant un nom à la contrainte:

```
create table Boats(
bid Number(5),
bname Varchar2(30),
color Varchar2(10)
CONSTRAINT BoatKey PRIMARY KEY (bid));
```

Primary Key × Unique

- Primary Key: les valeurs nulles ne sont pas possibles
- Unique: il est possible d'avoir des valeurs nulles
- Nous pouvons ajouter la contrainte NOT NULL pour indiquer qu'un attribut ne peut pas avoir des valeurs nulls.

```
create table Sailors (
sid  Number(4) primary key,
sname Varchar2(40) not null,
rating Number(2),
age Number(3));
```

Les contraintes de clés étrangères

- Les contraintes de clés étrangères mettent en rapport plusieurs tables.
- Soit deux relations R et S. Soit une **clé étrangère** qui implémente la **dépendance d'inclusion**: $R.X \subseteq S.Y$
- Cela veut dire:
  1. Les tuples (de valeurs) associés aux attribut X dans la relation R doivent apparaître comme tuples (de valeurs) associés aux attributs Y d'une relation S.
  2. Y est associé à une contrainte UNIQUE ou PRIMARY KEY

```
create table Reservation(
  snum  Number(4),
  bnum Number(5),
  day Date,
  primary key (snum,day),
  foreign key (snum) references Sailors(sid),
  foreign key (bnum) references Boats(bid) );
```

Les contraintes de clés étrangères

Une autre syntaxe:

```
 create table Reservation(
   snum  Number(4),
   bnum Number(5),
   day Date,
   primary key (snum,day),
   CONSTRAINT FKNumSailor FOREIGN KEY (snum)
         references Sailors(sid),
   CONSTRAINT FKNumBoat FOREIGN KEY (bnum) references Boats(bid)
);
```

Le renforcement des contraintes

```
create table Reservation(
  snum  Number(4),
  bnum Number(5),
  day Date,
  primary key (snum,day),
  CONSTRAINT FKNumSailor FOREIGN KEY (snum)
        references Sailors(sid)
        ON DELETE CASCADE);
```

Autres contraintes sémantiques

- Nous pouvons spécifier des contraintes diverses avec CHECK *condition*
- Exemples:

L'attribut *rating* doit être associé à des entiers de 1 à 10.

```
create table Sailors (
sid  Number(4), sname Varchar2(40) not null,
rating Number(2), age Number(3)
primary key (sid),
CHECK (rating >= 1 and rating <=10));
```

Valeurs par default

- Il est possible de déclarer une valeur par défaut associée à un attribut.
- En général, cela se fait dans n'importe quelle instruction où nous devons déclarer un attribut et son type.
- Mot clé: DEFAULT suivi de la valeur (ex: NULL ou une constante)
- Certaines valeur sont fournies par le système comme la date.
- Exemple:

```
gender CHAR(1) DEFAULT '?',
birthday DATE DEFAULT DATE '0000-00-00'
```

- Example:

```
ALTER TABLE Sailors ADD phone CHAR(16) DEFAULT 'unlisted';
```

Les vues

- Une vue est une relation correspondant au résultat d'une requête sur la base. **Cette relation n'est pas stockée dans la base**, mais calculée à chaque fois que nous avons besoin.
- Nous pouvons faire des requêtes et des mises à jour sur une vue (des limites pour les mises à jour sont imposées par les SGBD).
- Exemple

```
create view ActiveSailors (name, age, day)
as select S.sname, S.age, R.day
   from Sailors S, Reservation R
   where S.sid = R.snum and S.rating > 6;
```

Une requête:

```
select S.sname, S.age,
from ActiveSailors
where S.age > 50 ;
```

Quelques remarques coté administration

- Une base de données Oracle est constituée de plusieurs éléments :
  - Des processus chargés en mémoire sur le serveur
  - Des fichiers physiques stockés sur le serveur
  - D'un espace mémoire sur le serveur appelé SGA (System Global Area)
- On appelle instance Oracle les processus et la SGA d'une base de données Oracle.
- Ne pas confondre l'instance Oracle (le SGBD) et la base de données!!
- Rappel: un schéma de base de données est l'ensemble de schéma des relations d'une base.
- **Dans Oracle un schéma de base de données est associé à un utilisateur.** Ainsi toutes les tables déclarées sur votre compte Oracle sont gérées via une instance Oracle et vues comme un schéma de base de données (vous avez peut être des tables sans aucun rapport dans votre compte, mais le SGBD les voit comme une base de données).
  Pour avoir **un autre schéma de BD il faut avoir un autre compte utilisateur**.
- Un utilisateur - par défaut - ne peut pas voir la base d'un autre.

Contrôle d'accès

- En général, les bases de données (d'une entreprise, par exemple) contiennent une grande quantité d'information.
- La plupart des utilisateurs a besoin d'accéder une petite partie de ces informations. SGBD doit fournir des mécanismes pour contrôler l'accès aux données.
- **Les privilèges ou *Discretionary acess control***:
  - Une approche pour le contrôle d'accès offerts par un SGBD.
  - Permet à un utilisateur d'accéder à un objet d'une certaine manière (lecture ou modification, par exemple).
  - L'utilisateur qui crée un objet (par exemple une table ou une vue) a tous les privilèges applicables sur cet objet.
  - Le SGBD ensuite va surveiller comment ces privilèges sont donnés ou enlever des autres utilisateurs.
  - SQL fournit les commandes GRANT pour donner des privilèges à des utilisateurs et REVOKE pour les supprimer.

Commande GRANT

- `GRANT privileges ON object TO user [WITH GRANT OPTION]`
- Dans notre cadre, `object` est une table ou une vue

Commande GRANT: privilèges

Certains privilèges peuvent être spécifiés:

- SELECT: Le **droit d'accéder (ou lire) tous les attributs** d'une relation spécifiée comme `object`, y compris les attributs ajoutés à posteriori avec ALTER TABLE.
- INSERT (nomColonne): Le **droit d'insérer des tuples** avec des valeurs (autres que NULL ou défaut) correspondant aux colonnes (attributs) d'une relation spécifiée comme `object`.

  Si le droit d'insertion concerne toutes les colonnes (y compris celles ajouter à posteriori) nous devons utiliser seulement INSERT.
- UPDATE or UPDATE (nomColonne): similaire à INSERT
- DELETE: Le **droit de supprimer des colonnes d'une relation spécifiés comme `object`**.
- REFERENCES (nomColonne): Le **droit de définir des clés étrangères** (dans d'autres tables) qui font référence à une colonne spécifiée dans la relation `object`. REFERENCES donne les droits à toutes les colonnes.

Commande GRANT

```
GRANT privileges ON object TO user [WITH GRANT OPTION]
```

- Si un utilisateur a un privilège avec GRANT OPTION **il peut passer ce privilège à d'autres utilisateurs** (avec ou sans GRANT OPTION).
- Un utilisateur qui **crée sa table** a **tous les privilèges sur elle y compris le droit de passer ses privilèges à d'autres**.
- Un utilisateur qui **crée sa vue** a, sur la vue, les **droits qui correspondent aux privilèges sur les tables** servant à la construction de la vue.
- L'utilisateur qui crée une vue doit avoir le **droit SELECT sur les tables** servant à la construction de la vue.
- L'utilisateur qui crée une vue **a le droit SELECT avec GRANT OPTION seulement s'il a ce droit pour la table** servant à la construction de la vue.
- Si l'utilisateur qui crée une vue a les droits de mises à jour sur la table servant à la construction de la vue, alors il a aussi le droit de mise à jour sur la vue.
- Seulement le propriétaire d'un schéma peut exécuter les déclarations CREATE, ALTER, DROP de ce schéma. **Le droit d'exécuter ces déclarations ne peut pas être transmis ou supprimé.**

La sécurité et les vues

- Ensemble avec les commandes GRANT et REVOKE, les vues sont importantes pour assurer la sécurité. La vue nous permet de cacher des informations de la base à des utilisateurs.
- Exemple

```
create view ActiveSailors (name, age, day)
as select S.sname, S.age, R.day
   from Sailors S, Reservation R
   where S.sid = R.snum and S.rating > 6;
```

- Un utilisateur qui peut accéder ActiveSailors mais qui ne peut pas accéder Sailors ou Reservation, peut connaître les noms des marins ayant réservé un bateaux, mais pas leur ID ou les noms des bateaux qu'ils ont réservés.

Exemples

- Privilèges sont attribués à des ID (un utilisateur, un groupe...)
- Supposons que *Joe* est le propriétaire des tables Sailors, Boats et Reservation. Les prenons ci-dessous indiquent des ID.

```
GRANT INSERT, DELETE ON Reservation TO Yuppy WITH GRANT OPTION
GRANT SELECT ON Reservation TO Mike
GRANT SELECT ON Sailors TO Mike WITH GRANT OPTION
GRANT UPDATE (rating) ON Sailors TO Lea
GRANT REFERENCES (bid) ON Boats TO Bill
```

- Avec ses privilèges Mike peut créer la vue ActiveSailors, mais il ne peut pas passer ses privilèges SELECT à d'autres.

Commande REVOKE

- Supprimer un droit ou l'option GRANT

```
REVOKE [GRANT OPTION FOR] privileges ON object
       FROM users {RESTRICT | CASCADE}
```

- CASCADE: Un utilisateur Toto AVAIT un droit (avec GRANT OPTION). Il a passé son droit à d'autres. Quand cet utilisateur perd son droit **tous** les utilisateurs ayant reçu les privilèges en question seulement via Toto, perdent leur privilèges aussi.
- RESTRICT: Si l'utilisateur n'a pas passé ses droits, seulement le privilège directement supprimé est supprimé. Si l'utilsateur a passé ses droits, la commande REVOKE n'est pas acceptée.

Comment définir les contraintes?

- Pour savoir quelles contraintes implémenter il faut bien étudier la base de données que nous voulons implémenter.
- Il faut analyser quelles sont les contraintes que nous voulons mettre en place. Ces contraintes sont, pour la plupart, exprimées via de dépendances fonctionnelles...