Création de base de données en SQL - exercices dans le cadre du cours à l'IBIS.
Sébastien Clément, avril 2011

Interface Web PhpPgAdmin:

- permet de faire des requêtes SQL (≥1 commandes)
- permet de visualiser la structure des tables et leur contenu
- navigation facile avec les onglets.

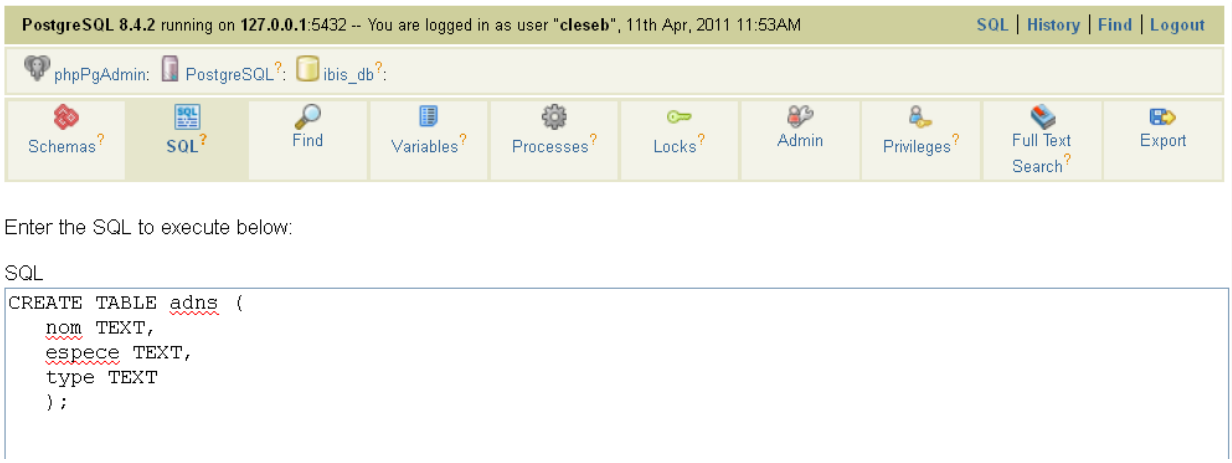
Paramètres de connexion au serveur de BD PostgreSQL:

http://dbs.arborea.ulaval.ca/phppgadmin/

Entrer votre nom d'utilisateur et mot de passe:

Choisir:
A) SQL - pour effectuer les commandes de création
B) Schéma: ibis01, ibis02, ... (selon votre utilisateur) - pour visualiser la structure et les données
Note: schéma = partition à l'intérieur d'une base de données

Terminal SQL:



Note: pour constater les restrictions d'accès spécifiques aux utilisateurs (privilèges), tentez de voir ou de manipuler les données dans les schémas des autres utilisateurs.

COMMANDES SQL - Création des tables, contraintes et enregistrements (DDL et DML):

Création de table 1 - ADNs:

```
CREATE TABLE adns (
        nom TEXT,
        espece TEXT,
        type TEXT
        );
```

Insertion d'enregistrements:

```
        INSERT INTO adns (nom,espece,type) VALUES ('at_312','A.
        thaliana','Génomique');
        INSERT INTO adns (nom,espece,type) VALUES ('at_312','A.
        thaliana','Génomique');
```

Note: on a entré 2 fois le même enr.!

On efface tout:

```
        DELETE FROM adns;
```

Pour empêcher la duplication d'enr.:

```
        ALTER TABLE adns ADD CONSTRAINT pk_adns PRIMARY KEY (nom);
```

Réinsertion 1:

```
        INSERT INTO adns (nom,espece,type) VALUES ('at_312','A.
        thaliana','Génomique');
```

Réinsertion 2:

```
        INSERT INTO adns (nom,espece,type) VALUES ('at_312','A.
        thaliana','Génomique');
```

Note: ici, la contrainte d'intégrité empêche les valeurs dupliquées dans "nom"

Création de table 2 - Amorces:

```
CREATE TABLE amorces (
	nom TEXT,
	sequence TEXT,
	sens TEXT,
	adn_gabarit TEXT,
	pos_vs_gabarit INTEGER,
	date_creation date
	);
```

On crée la clé primaire:

```
	ALTER TABLE amorces ADD CONSTRAINT pk_amorces PRIMARY KEY (nom);
```
Note: aurait-on pu choisir une autre colonne comme clé primaire ?

On ne peut avoir qu'une clé primaire par table.
Cependant, on peut imposer l'unicité des valeurs dans un autre champ, comme suit

```
	ALTER TABLE amorces ADD CONSTRAINT ct_check_sequence_uniqueness UNIQUE
	(sequence);
```

Insertion d'enregistrements:

```
	INSERT INTO amorces
	(nom,sequence,sens,adn_gabarit,pos_vs_gabarit,date_creation) VALUES
	('at39234F','ACACACAACAACACAATTCG','F','at_312',43,'2011-02-12');
	INSERT INTO amorces
	(nom,sequence,sens,adn_gabarit,pos_vs_gabarit,date_creation) VALUES
	('at39234R','CATACATATCATACTATCAACTA','R','at_312',512,'2011-02-12');
```

Note: remarquez la présence ou l'absence d'apostrophes dans les valeurs insérées

On veut que l'ADN gabarit ayant servi à créer les amorces provienne de notre table ADNs:
Actuellement, pas le cas:

```
	INSERT INTO amorces
	(nom,sequence,sens,adn_gabarit,pos_vs_gabarit,date_creation) VALUES
	('amorce1234','ACACACACATCATCATACAT','F','ADN du frigo 1223B',512,'2011-
	02-12');
```

On efface cette amorce:

```
	DELETE FROM amorces WHERE nom='amorce1234';
```
voir contenu amorces

On crée une clé étrangère dans 'amorces', pointant vers 'adns':

```
	ALTER TABLE amorces ADD constraint fk_amorces_adn_gabarit FOREIGN KEY
	(adn_gabarit) REFERENCES adns(nom) ON UPDATE CASCADE;
```

Note: ON UPDATE CASCADE...
Voir structure table

Nouvelle tentative:

	INSERT INTO amorces (nom,sequence,sens,adn_gabarit,pos_vs_gabarit,date_creation)
	VALUES ('amorce1234','ACACACACATCATCATACAT','F','ADN du frigo 1223B',512,'2011-02-
	12');

Note: interdit l'ajout de toute amorce non basée sur un ADN existant dans la table adns (intégrité référentielle)

Autre exemple de la puissance de cette contrainte:

```
SELECT * FROM adns;
SELECT * FROM amorces;

UPDATE adns SET nom='at_312_NOUVEAU' where nom='at_312';
SELECT * FROM adns;
SELECT * FROM amorces;
UPDATE adns SET nom='at_312' where nom='at_312_NOUVEAU';
```

Créer 3e table: PCRs

```
CREATE TABLE pcrs (
      id SERIAL,
      adn_nom TEXT,
      amorce_f_nom TEXT,
      amorce_r_nom TEXT,
      programme TEXT,
      melange TEXT,
      utilisateur TEXT,
      date_execution DATE,
      taille_amplicon INTEGER,
      details TEXT
      );
```

Note: SERIAL: numéro auto incrémenté, assure l'unicité de la clé.

On crée les clés primaires et secondaires:

```
ALTER TABLE pcrs ADD CONSTRAINT pk_pcrs PRIMARY KEY (id);

ALTER TABLE pcrs ADD constraint fk_pcrs_adn_nom FOREIGN KEY (adn_nom)
      REFERENCES adns(nom) ON UPDATE CASCADE;

ALTER TABLE pcrs ADD constraint fk_pcrs_amorce_f_nom FOREIGN KEY
      (amorce_f_nom) REFERENCES amorces(nom) ON UPDATE CASCADE;

ALTER TABLE pcrs ADD constraint fk_pcrs_amorce_r_nom FOREIGN KEY
      (amorce_r_nom) REFERENCES amorces(nom) ON UPDATE CASCADE;
```

Note: encore une fois, de cette façon, on ne peut créer que des PCRs avec des amorces et ADNs existants.
Ça implique une rigueur dans l'ordre d'entrée des enr. (adns et amorces avant), mais ça assure une cohérence des données impossible à atteindre sans programmation en fichiers à plat ou en Excel .

Ajout du reste des données

ADNs:

```
INSERT INTO adns (nom,espece,type) VALUES ('pt_221s','P. taeda','cDNA');
INSERT INTO adns (nom,espece,type) VALUES ('pg_S352','P.
glauca','cDNA');
INSERT INTO adns (nom,espece,type) VALUES ('ADN_K33','P.
aeruginosa','Plasmidique');
INSERT INTO adns (nom,espece,type) VALUES ('ce_ttY14','C.
elegans','cDNA');
```

Amorces:

```
INSERT INTO amorces
(nom,sequence,adn_gabarit,sens,pos_vs_gabarit,date_creation) VALUES
('pt00324F','GCCTGCAAGAAGACCCTAGTT','pt_221s','F',52,'2011-03-05');
INSERT INTO amorces
(nom,sequence,adn_gabarit,sens,pos_vs_gabarit,date_creation) VALUES
('pt00324R','CACTTAGGCAACATCCACTTACC','pt_221s','R',744,'2011-03-05');
INSERT INTO amorces
(nom,sequence,adn_gabarit,sens,pos_vs_gabarit,date_creation) VALUES
('K33_001F','CCAGATCTACTACATACTTACTAT','ADN_K33','F',23,'2010-09-07');
INSERT INTO amorces
(nom,sequence,adn_gabarit,sens,pos_vs_gabarit,date_creation) VALUES
('K33_001R','CAACATCTATCTACTATCTAT','ADN_K33','R',455,'2010-09-07');
INSERT INTO amorces
(nom,sequence,adn_gabarit,sens,pos_vs_gabarit,date_creation) VALUES
('K33_002F','CACACACACAATACGACACAT','ADN_K33','F',433,'2010-09-07');
INSERT INTO amorces
(nom,sequence,adn_gabarit,sens,pos_vs_gabarit,date_creation) VALUES
('K33_002R','CGGCGATTATTCGATTACGATCT','ADN_K33','R',971,'2010-09-07');
```

PCRs:

```
INSERT INTO pcrs
(adn_nom,amorce_f_nom,amorce_r_nom,programme,melange,utilisateur,date_ex
ecution,taille_amplicon,details) VALUES
('at_312','at39234F','at39234R','Prog0077','Mix321','G. Turcotte','2011-
04-02',NULL,'Non-spécifique');
INSERT INTO pcrs
(adn_nom,amorce_f_nom,amorce_r_nom,programme,melange,utilisateur,date_ex
ecution,taille_amplicon,details) VALUES
('at_312','at39234F','at39234R','Prog0002','Mix321','G. Turcotte','2011-
04-08',1400,'Belle amplification');
INSERT INTO pcrs
(adn_nom,amorce_f_nom,amorce_r_nom,programme,melange,utilisateur,date_ex
ecution,taille_amplicon,details) VALUES
('ADN_K33','K33_001F','K33_001R','Prog0012','Mix217','M. Simard','2011-
02-09',NULL,'Aucune amplification');
INSERT INTO pcrs
(adn_nom,amorce_f_nom,amorce_r_nom,programme,melange,utilisateur,date_ex
ecution,taille_amplicon,details) VALUES
('ADN_K33','K33_001F','K33_001R','Prog0012','Mix218','M. Simard','2011-
02-10',450,'Belle amplification');
```

Note1: remarquer l'utilisation du NULL (absence de valeur) et non pas de '', qui n'est pas une valeur nulle
Note2: l'ajout de données en lot (ex: 1000 lignes et plus) plutôt que par commandes SQL individuelles, est non-seulement possible, mais préférable

Autres contraintes:

```
ALTER TABLE adns ADD CONSTRAINT ct_adns_type_domain CHECK (type IN
('cDNA','Plasmidique','Génomique'));

ALTER TABLE amorces ADD CONSTRAINT
ct_amorces_sequence_allow_nucleotide_letters_only CHECK (sequence !~
'[^atgcATGC]');
```

COMMANDES SQL - Interroger une base de données (DML):

Base

```
SELECT * from amorces;
```

Sélection des colonnes

```
SELECT nom,sequence FROM amorces;
```

Sélection des lignes:

```
SELECT * FROM amorces WHERE sens='F';
```

Calculs et constantes:

```
SELECT (pos_vs_gabarit+1000) AS pos_corrigee FROM amorces;

SELECT nom,'Georges' AS createur, date_creation FROM amorces;
```

Condition et concaténation:

```
SELECT nom, (CASE WHEN sens='F' THEN 'Forward' ELSE 'Reverse' END) AS
sens_long,pos_vs_gabarit||sens AS pos_sens FROM amorces;
```

Clauses DISTINCT, ORDER BY, NULL, IN:

```
SELECT DISTINCT sens FROM amorces;

SELECT * FROM amorces ORDER BY date_creation;

SELECT * FROM pcrs WHERE taille_amplicon IS NOT NULL;

SELECT adn_nom,amorce_f_nom,amorce_r_nom FROM pcrs WHERE id >1 AND
programme IN ('Prog0002','Prog0012');
```

Multi-tables

Jointure normale:

```
SELECT adns.nom,adns.espece,adns.type, amorces.nom,
amorces.sequence
FROM adns JOIN amorces ON adns.nom=amorces.adn_gabarit;
```

Note1: Dans les requêtes multi-tables, il y a risque d'avoir un nom de champ commun à plusieurs tables, donc on doit spécifier la table devant le nom du champ, séparé par un point.
Note2: Pour chaque ADN, il peut y avoir plus d'une amorce, donc les infos sur l'ADN sont répliquées.
Note3: la jointure limite les enregistrements à ceux communs entre 'adns' et 'amorces', c'est à dire que les ADN pour lesquels il n'y a pas d'amorces n'apparaissent pas. Pour faire apparaître aussi ces ADN, on utilise la jointure à gauche (ci-dessous).

Jointure à gauche:

```
SELECT adns.nom,adns.espece,adns.type, amorces.nom,
amorces.sequence
FROM adns LEFT JOIN amorces ON adns.nom=amorces.adn_gabarit;
```

Sous-requêtes/requêtes imbriquées:

```
SELECT adn_nom,amorce_f_nom, (SELECT sequence FROM amorces WHERE
nom=amorce_f_nom) AS am_f_seq FROM pcrs;
```

Note: Alternative à une jointure quand on veut aller chercher un enregistrement précis

d'une autre table avec une clé de la table actuelle, à condition que la sous-requête ne retourne qu'un enregistrement et une seule colonne.

Requêtes d'agrégation

```
SELECT sens, AVG(pos_vs_gabarit) AS avg_pos, COUNT(*) AS nb_amorces FROM
amorces GROUP BY sens;
```

Recherche par expressions régulières:

LIKE

```
SELECT * FROM amorces WHERE sequence LIKE '%GGC%';
```

POSIX

```
SELECT * FROM amorces WHERE nom ~ '[^0-9]3';

SELECT nom, sequence, length(sequence) FROM amorces WHERE nom ~
'[^0-9]3';

SELECT nom, sequence,
round((length(regexp_replace(sequence,'[^GC]','','g'))::numeric/le
ngth(sequence)::numeric*100),1) AS pct_gc, length(sequence) FROM
amorces WHERE nom ~ '[^0-9]3';
```

Sauvegarder une requête (création de vue):

```
CREATE VIEW v_calculs_amorces AS
    SELECT nom, sequence,
    round((length(regexp_replace(sequence,'[^GC]','','g'))::numeric/le
    ngth(sequence)::numeric*100),1) AS pct_gc, length(sequence) FROM
    amorces;
```

Maintenant, la définition de la vue est sauvegardée dans 'Views'
Pratique pour les vues utilisées couramment.
Dès que les données sont mises-à-jour, les changements sont visibles dans cette vue.