Chapitre B

Premier noyau de SQL

The limits of my language mean the limits of my world.
Ludwig Wittgenstein

B - Premier noyau de SQL

- B.1 – Historique et objectifs
- B.2 – Démarches de présentation
- B.3 – Introduction aux requêtes
- B.4 – Introduction à la manipulation
- B.5 – Conclusion

B.1 Historique et objectifs

- Edgard CODD 1970 -> SEQUEL
- SEQUEL -> SQL
- Structured Query Language
- 1979 première implémentation de SQL
- Référence normes
  - ANSI X3.135-1999, "Database Language SQL",
  - ISO/IEC 9075:1999, "Database Language SQL",

Fonctionnement de SQL

- Travail sur des groupes de données, et non pas sur des données individuelles
- Balayage automatique des tables
- Programmation déclarative (initialement)
- Existence d'un optimiseur de requête

Grands types d'instructions

- Interrogation
- Insertion, MAJ et suppression de lignes
- Création, remplacement, altération et suppression des composants (tables, etc.)
- Contrôle d'accès
- Cohérence et intégrité

Sous-ensembles de SQL

- Embedded SQL
  - instructions interactives compilées au vol
- Dynamic SQL
  - extensions procédurales
- Interfaces avec langages (pré-compilateurs)
  - PRO*C/C++
  - PRO*COBOL
  - etc.

Conventions d'écritures

- Alignement indifférent
- Majuscules/minuscules indifférentes dans les ordres et les noms de variables
- Majuscules/minuscules importantes dans les chaînes de caractères
  - 'Toto' ≠ 'toTo'

B.2 Démarches de présentation

- Top-down
  - grandes fonctionnalités
  - rédaction des instructions
  - notions structurantes du langages
- Bottom-up
  - compilation du langage
  - construction progressive du langage SQL

Top-down

- Avantages
  - objectif clair
  - permet d'avoir un premier aperçu
  - démarche intuitive
- Inconvénients
  - beaucoup de choses restent cachées
  - trop de pointeurs « en avant »
  - opérationalité peu évidente

Bottom-up

- Avantages
  - construction progressive
  - tout est parfaitement connu
- Inconvénients
  - on ne voit pas où l'on va
  - pas de hiérarchie entre l'important et le non-important

Structuration du cours

- Objectifs
  - Bien connaître le langage
  - Savoir programmer en SQL
- Structuration en deux passes
  - 1ère passe rapide top-down
    - présentation par des exemples
  - 2ème passe bottom-up plus approfondie
    - présentation de la grammaire de SQL

B.3 - Introduction aux requêtes

```
SELECT ....
FROM ......
[WHERE ….]
```

Projection

- `SELECT [DISTINCT] liste_resultats | * FROM nom_de_table`
- `liste_resultats`
  - `::= resultat [, resultat]*`
- `resultat`
  - `::= attribut | fonction (attribut)`

Rôle de **DISTINCT**

- Élimine les lignes résultats identiques

Projection avec tri

- Données jamais ordonnées
- Si tri alors
  - `ORDER BY nom_col1|numer_col1|expression1 [DESC][,nom_col1|numer_col1|expression1 [DESC]... etc..`

Restriction

- `SELECT ... FROM.... WHERE prédicat`
  - `prédicats simples`
  - `=, != ou <>, >, >=, < , <=`
  - `BETWEEN expr1 AND expr2`
  - `IN (expr1, expr2, ..)`
  - `LIKE chaîne`
    - substitution _ (un seul caractère à la fois)
    - longueur quelconque %

Expression

- Nom de colonne
- Constante

NULL

- Valeur inconnue, non définie ou absente
- `IS [NOT] NULL`

Prédicats composés

- `OR`
- `AND` (prioritaire)
- Souvent besoin de parenthèses pour préciser les priorités

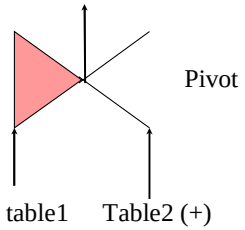
Produit cartésien

- `SELECT liste_attributs FROM liste_table`

Equijointure

- `Select liste_resultat from nom_table1, nom_table_2 where pivot`
- `pivot ::=[table1.]colonne1= [tableB.]colonne2`

Jointure externe



+ correspond à une table subordonnée (dans laquelle il manque des éléments)

- pivot :
  - [table1.]colonne1= [table2.]colonne2(+)
  - [table2.]colonne2(+)= [table1.]colonne1

Théta-jointure

- Jointure avec comparaison différente de = soit `!= ou <>, >=, < , <=`

Autojointure

- Jointure avec la même table

Notion d'alias

- Permet de renommer une table
- Ex :

```
SELECT a.toto, b.titi
FROM table_toto a, table_titi b
 WHERE etc...
```

- Commode dans certaines écritures de jointures

Jointures et restrictions simultanées

```
SELECT ...
FROM .....
WHERE ....
AND ...
AND ...
AND...
```

Exemples de requêtes

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

SELECT store_name **FROM** Store_Information

store_name
Los Angeles
San Diego
Los Angeles
Boston

SELECT DISTINCT store_name FROM Store_Information

store_name
Los Angeles
San Diego
Boston

```
SELECT store_name
FROM Store_Information
WHERE Sales > 1000
```

store_name
Los Angeles

```
SELECT store_name
FROM Store_Information
WHERE Sales > 1000
OR (Sales < 500 AND Sales > 275)
```

store_name
Los Angeles
San Francisco

```
SELECT *
FROM Store_Information
WHERE store_name IN ('Los Angeles', 'San Diego')
```

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
SELECT *
FROM Store_Information
WHERE Store_Date BETWEEN 'Jan-06-1999' AND 'Jan-10-1999'
```

Store_name	Sales	Store_Date
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
SELECT store_name, Sales, Store_Date
FROM Store_Information
ORDER BY Sales DESC
```

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
Boston	$700	Jan-08-1999
San Francisco	$300	Jan-08-1999
San Diego	$250	Jan-07-1999

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Select region-name
From Geography, Store_Information
Where Store_Information.store_name=Geography.store_name
And Sales > 1000

Fonctions agrégatives

- **AVG**
- **COUNT**
- **MAX**
- **MIN**
- **SUM**

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
SELECT "function type"("column_name")
FROM "table_name"
```

SELECT SUM(Sales) FROM Store_Information

SUM(Sales)
$2750

```
SELECT COUNT("column_name")
FROM "table_name"
```

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
SELECT COUNT(store_name)
FROM Store_Information
```

Count(store_name)
4

Regroupement

```
SELECT "column_name1", SUM("column_name2")
FROM "table_name"
GROUP BY "column_name1"
```

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
SELECT store_name, SUM(Sales)
FROM Store_Information
GROUP BY store_name
```

store_name	SUM(Sales)
Los Angeles	$1800
San Diego	$250
Boston	$700

Condition sur groupement

```
SELECT "column_name1", SUM("column_name2")
FROM "table_name"
GROUP BY "column_name1"
HAVING (arithmetic function condition)
```

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
SELECT store_name, SUM(sales)
FROM Store_Information
GROUP BY store_name
HAVING SUM(sales) > 1500
```

store_name	SUM(Sales)
Los Angeles	$1800

Alias

```
SELECT "table_alias"."column_name1" "column_alias"
FROM "table_name" "table_alias"
```

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
SELECT A1.store_name Store, SUM(A1.Sales) "Total Sales"
FROM Store_Information A1
GROUP BY A1.store_name
```

Store	Total Sales
Los Angeles	$1800
San Diego	$250
Boston	$700

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

```
SELECT A1.region_name REGION, SUM(A2.Sales) SALES
FROM Geography A1, Store_Information A2
WHERE A1.store_name = A2.store_name
GROUP BY A1.region_name
```

REGION	SALES
East	$700
West	$2050

Selects imbriqués

```
SELECT "column_name1"
FROM "table_name1"
WHERE "column_name2" [Comparison Operator]
(SELECT "column_name3"
FROM "table_name2"
WHERE [Condition])
```

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
San Francisco	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

```
SELECT SUM(Sales) FROM Store_Information
WHERE Store_name IN
(SELECT store_name FROM Geography
WHERE region_name = 'West')
```

SUM(Sales)
2050

Union, intersection, difference

Table ***Store_Information***

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Internet_Sales***

Inter_Date	Sales
Jan-07-1999	$250
Jan-10-1999	$535
Jan-11-1999	$320
Jan-12-1999	$750

```
[SQL Statement 1]
UNION
[SQL Statement 2]
```

```
SELECT Store_Date FROM Store_Information
UNION
SELECT Inter_Date FROM Internet_Sales
```

```
Jan-05-1999
Jan-07-1999
Jan-08-1999
Jan-10-1999
Jan-11-1999
Jan-12-1999
```

UNION
INTERSECT
MINUS

B.4 - Introduction à la manipulation

- Création de table
- Insertion
- Effacement
- Modification de contenu
- Modification de strcture

Création et effacement de table

- Possible de préciser de multiples choses
- Présentation simplifiée
- Possibilité de modifier une table
- Ordres
  - **CREATE TABLE**
  - **DROP TABLE**
  - **ALTER TABLE**

Création simplifiée

```
CREATE TABLE "table_name"
("column 1" "data_type_for_column_1",
"column 2" "data_type_for_column_2",
... )
```

```
CREATE TABLE customer
(First_Name char(50),
Last_Name char(50),
Address char(50),
City char(50),
Country char(25),
Birth_Date date)
```

Suppression de table

```
DROP TABLE "table_name"
```

```
DROP TABLE customer.
```

Manipulation

- **DELETE**
  - **DELETE TABLE5 WHERE N0=25;**
- **INSERT**
  - **INSERT INTO TABLE6 VALUES (45, ' TITI ', 357);**
- **UPDATE**
  - **UPDATE TABLE8 SET N0=34 WHERE N0=56;**

Insertion standard d'une ligne

```
INSERT INTO "table_name" ("column1", "column2", ...)
VALUES ("value1", "value2", ...)
```

Table *Store_Information*

Column Name	Data Type
Store_name	char(50)
Sales	float
Store_Date	Date

```
INSERT INTO Store_Information (Store_name, Sales, Store_Date)
VALUES ('Los Angeles', 900, 'Jan-10-1999')
```

Insertion à partir d'une autre table

```
INSERT INTO "table1" ("column1", "column2", ...)
SELECT "column3", "column4", ...
FROM "table2"
```

```
INSERT INTO Store_Information (Store_name, Sales,Store_Date)
SELECT Store_name, Sales,Store_Date
FROM Sales_Information
WHERE Year(Store_Date) = 1998
```

Effacement d'une ligne

```
DELETE FROM "table_name"
WHERE {condition}
```

Table *Store_Information*

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
DELETE FROM Store_Information
WHERE store_name = "Los Angeles"
```

Table *Store_Information*

Store_name	Sales	Store_Date
San Diego	$250	Jan-07-1999
Boston	$700	Jan-08-1999

Mise à jour d'une ligne

```
UPDATE "table_name"
SET "column_1" = [new value]
WHERE {condition}
```

Table *Store_Information*

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

```
UPDATE Store_Information
SET Sales = 500
WHERE store_name = "Los Angeles"
AND Date = "Jan-08-1999"
```

Table *Store_Information*

Store_name	Sales	Store_Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$500	Jan-08-1999
Boston	$700	Jan-08-1999

Mise à jour de plusieurs lignes :

```
UPDATE "table_name"
SET column_1 = [value1], column_2 = [value2]
WHERE {condition}
```

B.5 - Conclusion

- On ne travaille que sur des tables

« Tout est table »

- Langage déclaratif (SQL)
- Langage procédural (PL/SQL - PROetc)