# Grammaire BNF Complète de GLSimpleSQL

## Qu'est-ce que la Grammaire BNF ?

### Définition

**BNF = Backus-Naur Form** (Forme de Backus-Naur)

C'est une **notation formelle** pour décrire la syntaxe d'un langage de programmation.

### Pourquoi c'est Important ?

1. **Communication** : Permet de décrire sans ambiguïté la syntaxe du langage
2. **Documentation** : Base pour créer le manuel du langage
3. **Implémentation** : Guide pour créer le parser
4. **Validation** : Permet de vérifier si une phrase est syntaxiquement correcte

### Notations Utilisées

| Notation | Signification | Exemple |
|---|---|---|
| `::=` | "est défini comme" | `<nombre> ::= <chiffre>` |
| `\|` | "ou" (alternatives) | `` `::= '+' '-'` `` |
| `< >` | Symbole non-terminal (règle) | `<requête>`, `<table>` |
| `' '` ou `" "` | Symbole terminal (littéral) | `'SELECT'`, `";"` |
| `[ ]` | Optionnel (peut être absent) | `[WHERE <condition>]` |
| `{ }` | Répétition (0 ou plusieurs fois) | `{<statement>}` |
| `( )` | Groupement | `(',' <champ>)*` |

### Différence BNF vs EBNF

**BNF (Backus-Naur Form)** : Notation basique `bnf <liste> ::= <element> <liste> ::= <liste> ',' <element>`

**EBNF (Extended BNF)** : Notation étendue avec `[]`, `{}`, etc. `ebnf <liste> ::= <element> {',' <element>}`

EBNF est plus lisible et concise. Nous utiliserons les **deux** ci-dessous.

---

# GRAMMAIRE BNF COMPLÈTE DE GLSimpleSQL

## Version BNF Pure (Format Standard)

```bnf
/*
================================================================
GRAMMAIRE BNF DE GLSimpleSQL
Langage SQL Simplifié pour l'Enseignement
================================================================
*/

/*
================================================================
1. STRUCTURE GÉNÉRALE
================================================================
*/

::=

::= |

                ';' | ';' | ';' /*                              ˅
                ================================================================
                CREATE TABLE
                ================================================================
                */ ::= 'CREATE' 'TABLE' '(' ')' ::= |
                ',' ::= ::= 'INT' | 'FLOAT' | 'BOOL' |
                'VARCHAR' '(' ')' /*
                ================================================================
                INSERT INTO
                ================================================================
                */ ::= 'INSERT' 'INTO' 'VALUES' '(' ')' |
                'INSERT' 'INTO' '(' ')' 'VALUES' '('
                ')' ::= | ',' ::= | ',' ::= | | | ::= 'TRUE' |
                'FALSE' /*
                ================================================================
                SELECT
                ================================================================
::= ';' | ';' | */                                  ::= 'SELECT' 'FROM' |
'SELECT' 'FROM' 'WHERE'

::= '*' |

/*
================================================================
1. UPDATE
================================================================
*/

::= 'UPDATE' 'SET' 'WHERE'
```

::= | ','

::= '='

/*
==========

1. DELETE
   ==========
   */

::= 'DELETE' 'FROM' | 'DELETE' 'FROM' 'WHERE'

/*
==========

1. DROP TABLE
   ==========
   */

::= 'DROP' 'TABLE'

/*
==========

1. CONDITIONS (Clauses WHERE)
   ==========
   */

::= | 'AND' | 'OR' | 'NOT' | '(' ')'

::=

::= '=' | '!=' | '<' | '>' | '<=' | '>='

/*
==========

1. ÉLÉMENTS LEXICAUX (Tokens)
   ==========
   */

::=

::=

::= { | | '_' }

::= [ '+' | '-' ] { }

::= [ '+' | '-' ] { } '.' { }

::= "'" { } "'" | '"' { } '"'

::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

::=

/*
==========

1. COMMENTAIRES (Ignorés par l'analyseur)
   ==========
   */

::= '--' { }

::= '/' { } '/' ```

---

# GRAMMAIRE EBNF (Format Étendu - Plus Lisible)

```ebnf /*
==========
GRAMMAIRE EBNF DE GLSimpleSQL (Notation Étendue)
==========
*/

/*
==========

1. STRUCTURE GÉNÉRALE
   ==========
   */

programme ::= instruction+

instruction ::= ( create*table* | *insert*into | select | update | delete | drop_table ) ';'

/*
==========

1. CREATE TABLE
   ==========
   */

create*table ::= '*CREATE*' '*TABLE*' *identificateur* '*(*' *definition*champ (',' definition_champ)* ')'

definition*champ ::= *identificateur type*donnees
```

type*donnees ::= '*INT*' | '*FLOAT*' | '*BOOL*' | '*VARCHAR*' '*(*' *nombre*entier ')'

/*
==========

1. INSERT INTO
   ==========
   */

insert_into ::= 'INSERT' 'INTO' identificateur [ '(' identificateur (',' identificateur)* ')' ] 'VALUES' '(' valeur (',' valeur)* ')'

/*
==========

1. SELECT
   ==========
   */

select ::= 'SELECT' liste_selection 'FROM' identificateur [ 'WHERE' condition ]

liste_selection ::= '' | *identificateur (',' identificateur)*

/*
==========

1. UPDATE
   ==========
   */

update ::= 'UPDATE' identificateur 'SET' affectation (',' affectation)* 'WHERE' condition

affectation ::= identificateur '=' valeur

/*
==========

1. DELETE
   ==========
   */

delete ::= 'DELETE' 'FROM' identificateur [ 'WHERE' condition ]

/*
==========

1. DROP TABLE
   ==========
   */

drop_table ::= 'DROP' 'TABLE' identificateur

/*
==========

1. CONDITIONS
   ==========
   */

condition ::= condition_or

condition*or ::= condition*and ( 'OR' condition_and )*

condition*and ::= condition*not ( 'AND' condition_not )*

condition*not ::= [ 'NOT' ] condition*primaire

condition*primaire ::= identificateur operateur*comparaison valeur | '('
condition ')'

operateur_comparaison ::= '=' | '!=' | '<' | '>' | '<=' | '>='

/*
==========

1. VALEURS ET TYPES
   ==========
   */

valeur ::= nombre*entier* | *nombre*reel | chaine_caracteres | 'TRUE' | 'FALSE'

nombre_entier ::= ['+' | '-'] chiffre+

nombre_reel ::= ['+' | '-'] chiffre+ '.' chiffre+

chaine_caracteres ::= "'" caractere* "'" | '"' caractere* '"'

identificateur ::= lettre (lettre | chiffre | '_')*

lettre ::= 'a'..'z' | 'A'..'Z'

chiffre ::= '0'..'9' ```

---

# EXPLICATION PÉDAGOGIQUE LIGNE PAR LIGNE

## Exemple 1 : CREATE TABLE

**Grammaire :** `bnf <create_table> ::= 'CREATE' 'TABLE' <nom_table> '(' <liste_definitions_champs> ')' <liste_definitions_champs> ::= <definition_champ> | <liste_definitions_champs> ',' <definition_champ> <definition_champ> ::= <nom_champ> <type_donnees>`

**Lecture :**

1. <create_table> est composé de :
    - Le mot-clé 'CREATE'
    - Le mot-clé 'TABLE'
    - Un <nom_table> (identifiant)
    - Une parenthèse ouvrante '('
    - Une <liste_definitions_champs>
    - Une parenthèse fermante ')'

1. <liste_definitions_champs> peut être :

    - **Soit** un seul <definition_champ>
    - **Soit** une <liste_definitions_champs> existante suivie de ',' et d'un autre <definition_champ>
    - Cette règle **récursive** permet d'avoir 1, 2, 3, ... N champs

2. <definition_champ> est composé de :

    - Un <nom_champ> (identifiant)
    - Un <type_donnees> (INT, FLOAT, etc.)

**Exemple concret :** `sql CREATE TABLE Client ( id INT, nom VARCHAR(50), age INT )`

**Arbre de dérivation :** <create_table> ├── 'CREATE' ├── 'TABLE' ├── <nom_table> → "Client" ├── '(' ├── <liste_definitions_champs> │ ├── <liste_definitions_champs> │ │ └── <definition_champ> │ │ ├── <nom_champ> → "id" │ │ └── <type_donnees> → 'INT' │ ├── ',' │ └── <definition_champ> │ ├── <nom_champ> → "nom" │ └── <type_donnees> → 'VARCHAR' '(' 50 ')' │ ├── ',' │ └── <definition_champ> │ ├── <nom_champ> → "age" │ └── <type_donnees> → 'INT' └── ')'

## Exemple 2 : SELECT avec WHERE

**Grammaire :** `bnf <select> ::= 'SELECT' <liste_selection> 'FROM' <nom_table> ['WHERE' <condition>] <condition> ::= <condition_simple> | <condition> 'AND' <condition> | <condition> 'OR' <condition>`

**Exemple concret :** `sql SELECT nom, age FROM Client WHERE age > 18 AND nom = 'Dupont'`

**Analyse :**

1. 'SELECT' : mot-clé
2. nom, age : correspond à <liste_selection>
3. 'FROM' : mot-clé
4. Client : correspond à <nom_table>
5. 'WHERE' : mot-clé optionnel (présent ici)
6. age > 18 AND nom = 'Dupont' : correspond à <condition>
    - age > 18 : <condition_simple>

- 'AND' : opérateur logique
- nom = 'Dupont' : <condition_simple>

**Arbre de condition :** <condition> (AND) ├── <condition_simple> │ ├── <nom_champ> → "age" │ ├── <operateur> → '>' │ └── <valeur> → 18 ├── 'AND' └── <condition_simple> ├── <nom_champ> → "nom" ├── <operateur> → '=' └── <valeur> → 'Dupont'

---

# DIAGRAMMES SYNTAXIQUES (Railroad Diagrams)

## CREATE TABLE

``` CREATE ──→ TABLE ──→ [identificateur] ──→ ( ──→ [definition_champ] ──┬→ ) ──→ ↑ │ └── , ←──────┘

[definition*champ] : [identificateur]* ──→ *[type*donnees]

[type_donnees] : ┬──→ INT ──────────────┬→ ├──→ FLOAT ──────────┤ ├──→ BOOL ──────────┤ └──→ VARCHAR ──→ ( ──→ [nombre] ──→ ) ──┘ ```

## SELECT

``` SELECT ──→ [liste_selection] ──→ FROM ──→ [table] ──┬→ ; ──→ │ └→ WHERE ──→ [condition] ──→

[liste_selection] : ┬──→ * ──────────────┬→ └──→ [champ] ──┬→──────┘ ↑ │ └── , ←┘ ```

## INSERT INTO

INSERT ──→ INTO ──→ [table] ──┬──────────────────────────────┬→ VALUES ──→ ( ──→ [valeur] ──┬→ ) ──→ │ │ ↑ │ └→ ( ──→ [champ] ──┬→ ) ──────┘ └── , ←──┘ ↑ │ └── , ←──┘

---

# TABLEAU RÉCAPITULATIF : BNF vs Code Bison

| Élément BNF | Code Bison | Exemple |
|---|---|---|
| <règle> ::= | règle: | select: |
| 'terminal' | TOKEN | SELECT |
| <non_terminal> | règle | field_list |
| `A | B` | A<br>\| B | INT_TYPE<br>\| FLOAT_TYPE |
| A B C | A B C | SELECT field_list FROM |
| [optionnel] | | opt_where_clause |

| Élément BNF | Code Bison | Exemple |
|---|---|---|
| | `/* empty */`<br>`\| élément` | |
| `{répétition}*` | Règle récursive | `field_list: field`<br>`\| field_list ',' field` |

---

# EXERCICE : Vérifier une Phrase avec la Grammaire

**Question :** La phrase suivante est-elle valide selon notre grammaire ? `sql`
`SELECT nom, prenom FROM Client WHERE age > 18 AND actif = TRUE;`

**Réponse :** OUI ✓

**Dérivation complète :**

```
<instruction> → <select> ';' → 'SELECT' <liste_selection> 'FROM'
<nom_table> 'WHERE' <condition> ';' → 'SELECT' <liste_champs>
'FROM' <nom_table> 'WHERE' <condition> ';' → 'SELECT' <nom_champ>
',' <nom_champ> 'FROM' <nom_table> 'WHERE' <condition> ';' →
'SELECT' "nom" ',' "prenom" 'FROM' "Client" 'WHERE' <condition>
';' → 'SELECT' "nom" ',' "prenom" 'FROM' "Client" 'WHERE'
<condition> 'AND' <condition> ';' → 'SELECT' "nom" ',' "prenom"
'FROM' "Client" 'WHERE' <condition_simple> 'AND'
<condition_simple> ';' → 'SELECT' "nom" ',' "prenom" 'FROM'
"Client" 'WHERE' "age" '>' 18 'AND' "actif" '=' 'TRUE' ';'
```

---

# POURQUOI C'EST IMPORTANT POUR VOTRE RAPPORT

## 1. Montre votre Compréhension Théorique

- Vous comprenez la théorie des langages formels
- Vous savez formaliser une syntaxe

## 2. Documente Complètement le Langage

- Quelqu'un peut implémenter GLSimpleSQL juste avec la grammaire
- Pas d'ambiguïté sur ce qui est accepté ou non

## 3. Base pour Justifier vos Choix

- "J'ai implémenté la règle X parce que la grammaire spécifie Y"

## 4. Facilite les Extensions Futures

- Quelqu'un veut ajouter une fonctionnalité ? Facile avec la grammaire

---

## TEMPLATE POUR VOTRE FICHIER Grammaire.pdf

Créez un document avec cette structure :

```markdown

# Grammaire Formelle de GLSimpleSQL

## 1. Introduction

GLSimpleSQL est un langage SQL simplifié...

## 2. Notation Utilisée

- ::= signifie "est défini comme"
- | signifie "ou"
- ...

## 3. Grammaire BNF Complète

[Copiez la grammaire BNF d'ici]

## 4. Grammaire EBNF (Notation Étendue)

[Copiez la grammaire EBNF d'ici]

## 5. Exemples de Dérivation

[Ajoutez 2-3 exemples comme ci-dessus]

## 6. Diagrammes Syntaxiques

[Ajoutez quelques diagrammes pour les règles principales]

## 7. Priorité des Opérateurs

| Priorité | Opérateurs | Associativité |
|---|---|---|
| 1 (haute) | NOT | Droite |
| 2 | =, !=, <, >, <=, >= | Gauche |
| 3 | AND | Gauche |
```

| Priorité | Opérateurs | Associativité |
|---|---|---|
| 4 (basse) | OR | Gauche |

```

---

# CONCLUSION

Cette grammaire BNF formelle définit complètement la syntaxe du langage GLSimpleSQL. Elle sert de référence pour :

1. **L'implémentation du parser** dans `sql_parser.y`
2. **La validation syntaxique** des requêtes SQL
3. **La documentation** du langage
4. **La vérification de conformité** au cahier des charges

---

**Auteur :** El-yass Hasnaoui
**Module :** THL et Compilation (I513)
**Filière :** LST GL S5
**Année :** 2025-2026
```