1. Projektantrag

1.1 Allgemeine Projektdaten

- **Projekttitel:**
  CLI-SQL-Syntax-Validator in C (Parser + Fehlerdiagnose)
- **Ausbildungsberuf:**
  Fachinformatiker/in – Anwendungsentwicklung
- **Projektteam:**
  Florian Leutloff, Nick Hildebrandt, Ha Hong Ho
- **Projektleitung (Azubi):**
  Nick Hildebrandt
- **Stellvertretende Projektleitung:**
  Florian Leutloff
- **Projektzeitraum:**
  KW 02 / 25 bis KW 08 / 25 (bitte eintragen)
- **Gesamtstunden:**
  ☐ 48 Schulstunden *a 3 Personen*

1.2 Projektbeschreibung

Ausgangssituation / Problemstellung:
SQL wird in vielen Anwendungen eingesetzt, und gerade beim Lernen oder beim schnellen Prototyping passieren häufig Syntaxfehler (z. B. fehlende Kommas, falsche Reihenfolge von Keywords, ungeschlossene Strings/Klammern). Viele Tools erkennen Fehler erst zur Laufzeit in einer Datenbank – oder liefern unklare Fehlermeldungen. Für Übungsaufgaben und für die schnelle Qualitätskontrolle von SQL-Skripten fehlt ein leichtgewichtiges Werkzeug, das zur schnellen validieren verwendet werden kann.

Ziel des Projekts:
Ziel ist die Entwicklung eines **Kommandozeilen-Programms in C**, das **SQL-Statements einliest** (per Argument oder über stdin/Datei) und diese **syntaktisch validiert**. Das Tool analysiert den Text in mehreren Stufen (Tokenisierung → Parsing → Validierung) und gibt bei Fehlern **verständliche Meldungen** aus, idealerweise mit **Position (Zeile/Spalte)** und einer kurzen **Hinweisdiagnose** (z. B. „; erwartet“, „UNION benötigt SELECT“, „String nicht geschlossen“).

Erwarteter Nutzen:

- **Für Schule / Ausbildung:** schneller Check von SQL-Aufgaben, besseres Verständnis von Syntaxregeln durch konkrete Fehlermeldungen.
- **Für Anwender/Übungsbetrieb:** leichtgewichtiges Tool ohne DB-Server, nutzbar in CI/Pre-Commit-Hooks, auf Linux einsetzbar.
- **Technischer Lerneffekt:** Parserbau, formale Sprachen, saubere CLI-Tools, Testbarkeit (Unit-/Integrationstests) und Teamarbeit mit Git.

1.3 Projektziele

Muss-Ziele:

1. CLI-Programm in C, das SQL-Text annimmt (Argument und/oder stdin und/oder Datei).
2. Tokenizer, der SQL in Tokens zerlegt (Keywords, Identifiers, Literale, Operatoren, Klammern, Kommas, Semikolon).
3. Parser/Validator für eine definierte SQL-Teilmenge (z. B. SELECT, FROM, WHERE, JOIN, ORDER BY, GROUP BY; optional INSERT/UPDATE/DELETE).
4. Fehlerausgabe mit **klarer Meldung** und **Fehlerposition**.
5. Versionsverwaltung (Git), Team-Workflow (Branches/MRs).

Kann-Ziele (optional):

- Unterstützung mehrerer Statements in einer Eingabe (getrennt durch `;`)
- Unit-Tests zur Qualitätssicherung
- „Did-you-mean“-Hinweise für Tippfehler bei Keywords

Nicht-Ziele (Abgrenzung):

- Keine Ausführung von SQL gegen eine echte Datenbank (kein Query-Execution).
- Keine semantische Validierung wie „Tabelle existiert“, „Spaltenname korrekt“.
- Kein vollständiger SQL-Standard (ISO) – nur definierte Teilmenge.

1.4 Projektabgrenzun

- **Nicht im Projekt enthalten:** DB-Anbindung, Abfrageausführung, Optimizer, Vollständigkeit für alle SQL-Dialekte (PostgreSQL/MySQL/SQLite etc.).
- **Externe Systeme/Komponenten, die nicht entwickelt werden:** Datenbankserver, GUI, Editor-Plugins.

1.5 Geplante Technologien und Werkzeuge

- **Programmiersprache(n):** C (C23)
- **Frameworks / Bibliotheken:** glibc sonst keine zwingend;
- **Datenbank:** keine (nur Syntaxvalidierung)
- **Entwicklungsumgebung:** VS Code / CLion / Vim (je nach Team), GCC/Clang
- **Versionsverwaltung:** Git (GitHub), Branching + Merge Requests

1.6 Grobe Zeitplanung in Unterrichtsstunden (Übersicht)

Phase	h
Analyse und Informationsbeschaffung	16
Entwurf	10
Implementierung	17
Test	5
Summe	**48**

2. Projektplanung und Feinplanung

> **Hinweis:** KWs bitte passend zu eurem Zeitraum eintragen. Verantwortlichkeiten sind so gesetzt, dass jede Person klare Pakete hat, aber Pairing möglich ist.

Phase 1: Analyse und Informationsbeschaffung

Nr.	Arbeitspaket	Verantwortlich	h	KW
1.0	Schreiben des Projektantrags, Einarbeitung in Tokenizing und Lexing für SQL	Florian, Nick und Ha	3*2h=**6**	02
1.1	Einarbeiten in die C Programmiersprache / Tools	Florian, Nick und Ha	3*3h=**9**	02
1.1	Git-Repository aufsetzen, Meson-Projekt anlegen und *Hello World* bauen, um Compiler und Development-Tools zu validieren	Nick	1	02
1.2	Basis-CLI-Interface mit Validierung der Argumente sowie Implementierung eines einfachen `--help`	Nick	2	02
1.3	Implementierung einer einfachen Linked List zur Nutzung eines dynamischen String-Arrays mit Iteration in C	Nick	3	02
1.4	Ist-Analyse / Recherche (SQL-Grammatik-Ansätze, Tokenizer-Design, Vergleich einfacher Parser-Strategien)	Ha	2	02
1.5	Testdaten-Sammlung (gültige/ungültige Beispiele, Edge-Cases: Strings, Escapes, Kommentare)	Florian	2	02

Phase 2: Entwurf

Nr.	Arbeitspaket	Verantwortlich	h	KW
2.1	Architekturentwurf (Module: cli, lexer, parser, ast, diagnostics, utils)	Ha	3	02
2.2	Grammatik/Parser-Strategie festlegen (z. B. rekursiver Abstieg, Pratt für Ausdrücke in WHERE)	Ha	3	05
2.3	Datenstrukturen (Token, TokenStream, AST-Knoten, Error/Diag-Structs)	Florian	2	05
2.4	Test-Setup mit Meson	Nick	2	05

Phase 3: Implementierung

Nr.	Arbeitspaket	Verantwortlich	h	KW
3.1	Lexer/Tokenizer (Keywords, Identifier, Zahlen, Strings, Kommentare, Whitespace, Positionen)	Ha	5	05
3.2	Parser: Statement-Level (SELECT …; optional INSERT/UPDATE/DELETE)	Florian	5	05
3.3	Parser: Expressions (WHERE Bedingungen, Operatoren, Klammern, Präzedenz)	Florian	4	05
3.4	Diagnostics (Fehlerposition, erwartete Tokens, verständliche Meldungen)	Nick	3	05

Phase 4: Test

Nr.	Arbeitspaket	Verantwortlich	h	KW
4.1	Modultests Parser (gültige/ungültige Statements, Grenzfälle)	Florian	2	08
4.2	Integrationstests CLI (stdin/file, Exitcodes, Ausgabeformat)	Nick	2	08
4.3	Review + Bugfix	Nick	1	08

3. Projektorganisationsplan (mit Redundanzen)

3.1 Projektrollen

Rolle	Hauptverantwortlich	Stellvertretung
Projektleitung	**Nick Hildebrandt**	Florian Leutloff
Parser/Validator	**Ha Hong Ho**	Nick Hildebrandt
CLI Interface	**Nick Hildebrandt**	Ha Hong Ho
Qualitätssicherung / Test	**Florian Leutloff**	Nick Hildebrandt
Dokumentation	**Nick Hildebrandt**	Florian Leutloff

3.2 Kommunikations-, Abstimmungsstruktur und Bereitstellung von Dokumenten

- **Zentrale Ablage:** gemeinsames Git-Repository (GitHub), Ordnerstruktur:
  - `/src` (C-Code), `/tests`, `/doc`
- **Dokumentation:** im Repository als Markdown (README, sowie Projektantrag)
- **Abstimmung:**
  - Kurzer Weekly-/Turnus-Check (10–15 Minuten) im Unterricht oder via Chat/Teams
- **Qualität:**
  - Code-Reviews via Merge Request
  - einheitlicher C-Style (Format-Regeln + Compiler-Warnings als Fehler, z. B. `-Wall -Wextra -Werror`)

Genehmigung

- **Datum:** 06.01.2026
- **Genehmigt durch (Lehrkraft):** _____