Database Evaluation Essay

Sarah Gillard

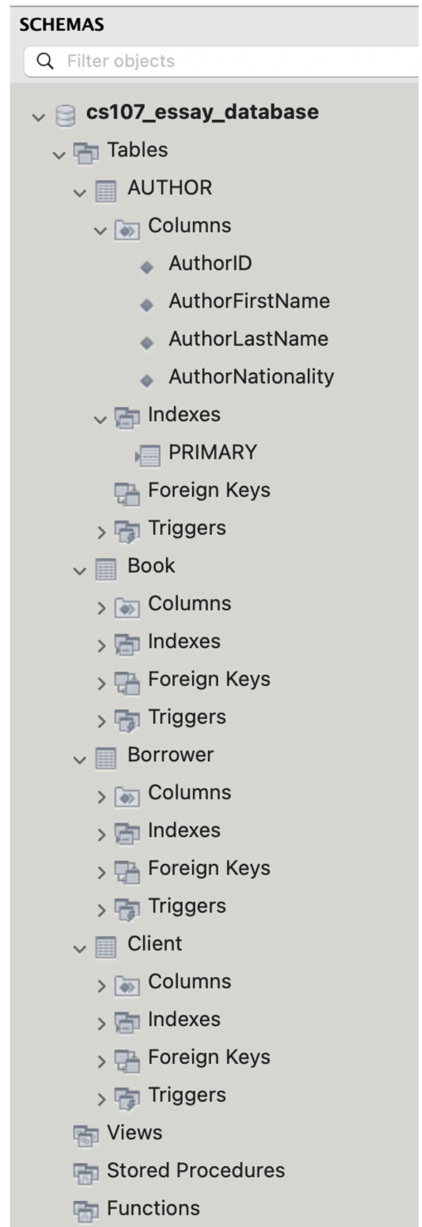
CS107: Database Fundamentals

February 6, 2024

Part 1 - Retrieve Data From a Database

To begin my assignment, I used MySQL to create the database provided in the ERD Diagram of the assignment instructions. I used the following code to create the database:

```
`cs107_essay_database` ;
USE cs107_essay_database;
CREATE TABLE Borrower (
BorrowID int(10),
ClientID int(10),
BookID int(10),
BorrowDate date DEFAULT NULL,
PRIMARY KEY (BorrowID)
);
CREATE TABLE Book (
BookIDBook int(10),
BookTitle char(10),
AuthorID int(10),
Genre nchar(15),
PRIMARY KEY (BookIDBook)
);
CREATE TABLE Client (
ClientID int(10),
ClientFirstName nchar(20),
```



ClientLastName nchar(20),

ClientDOB date DEFAULT NULL,

Occupation nchar(15),

PRIMARY KEY (ClientID)

);

CREATE TABLE AUTHOR (

AuthorID int(10),

AuthorFirstName nchar(20),

AuthorLastName nchar(20),

AuthorNationality nchar(15),

PRIMARY KEY (AuthorID)

);

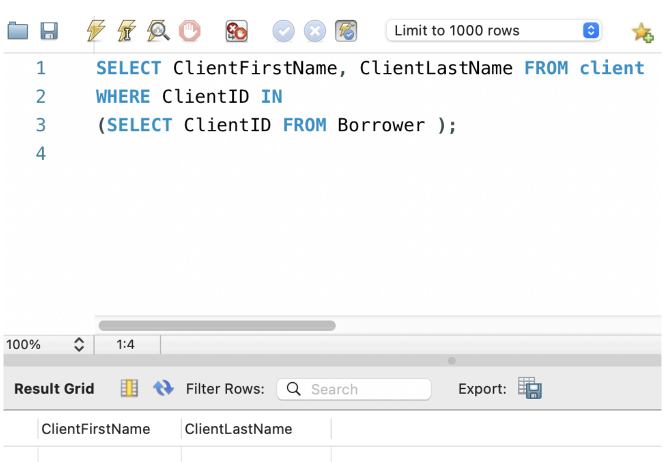
I wrote the following SQL statements to retrieve data:

- To select all clients who borrowed books:

SELECT ClientFirstName, ClientLastName FROM client

WHERE ClientID IN

(SELECT ClientID FROM Borrower);



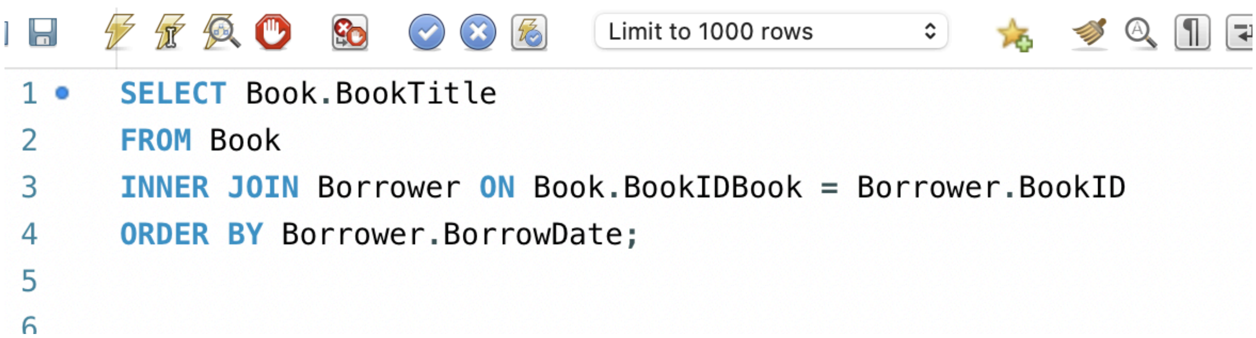
- To select all books borrowed by borrowers, ordering by borrow date:

SELECT Book.BookTitle

FROM Book

INNER JOIN Borrower ON Book.BookIDBook = Borrower.BookID

ORDER BY Borrower.BorrowDate;

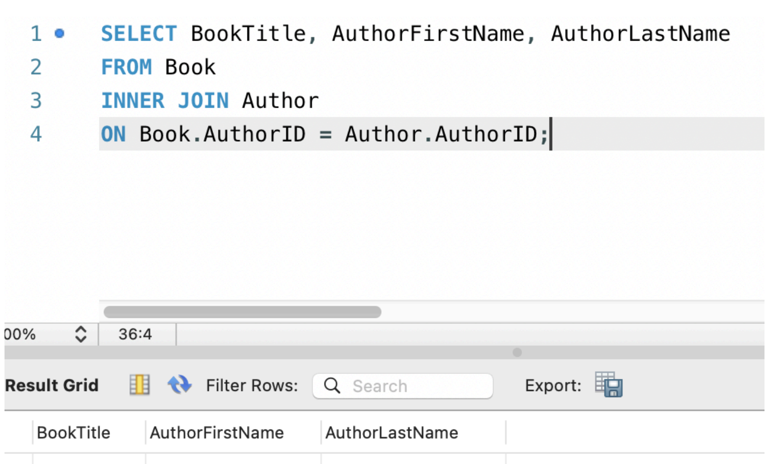


- To select all books and include the author first and last name:

SELECT BookTitle, AuthorFirstName, AuthorLastName

FROM Book

INNER JOIN Author ON Book.AuthorID = Author.AuthorID;



- To insert a new client with an occupation of pilot:

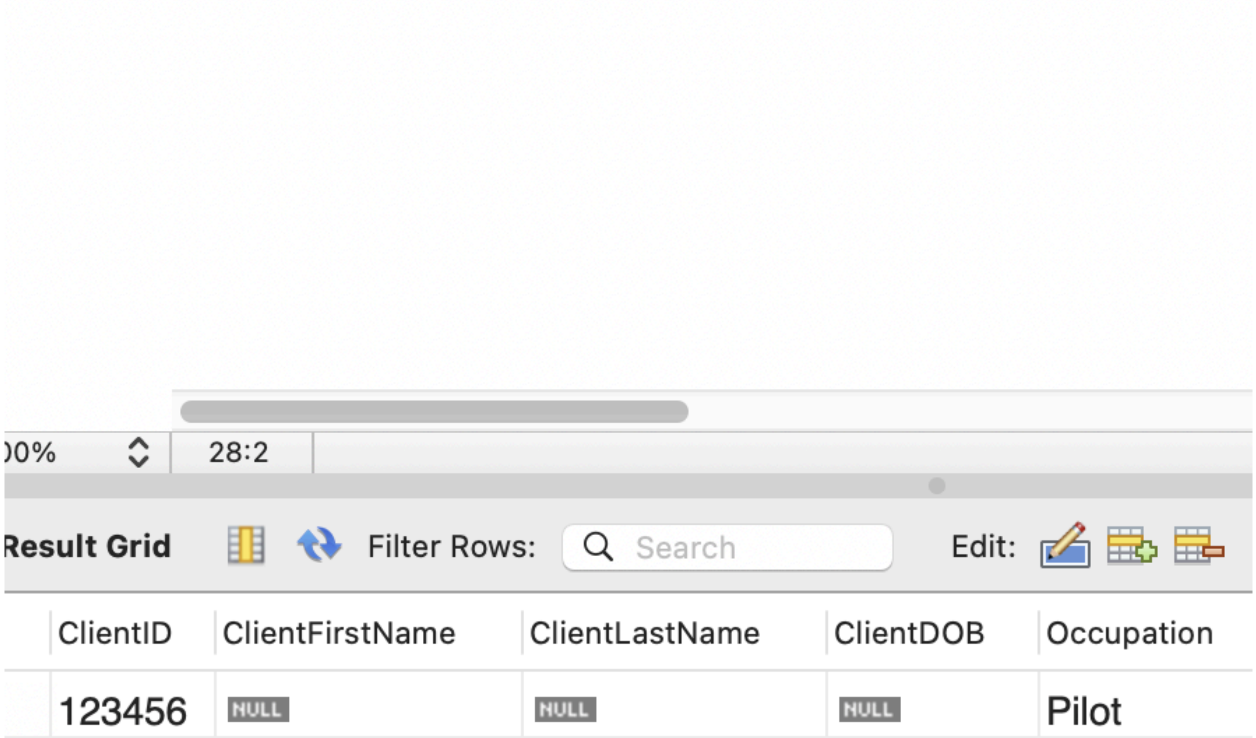
INSERT INTO Client(ClientID, Occupation)

VALUES(123456, 'Pilot');

```
INSERT INTO Client(ClientID, Occupation)
VALUES(123456, 'Pilot');
```

There is now a record in the Client table for a Pilot with a ClientID of 123456.

```
SELECT * FROM Client
WHERE Occupation = 'Pilot';
```



Part 2 - Database Normalization

The normalization process in a database involves restructuring to eliminate redundancy (Gloag, D., 2016). The current database consists of five tables: Produce, Animal Products, Grains, Suppliers, and Purchases. The identification of duplicate fields across tables indicates the need for streamlining. The following duplicate fields are identified across the tables in this database:

- ITEMID in the Produce, Animal Products, Grains, and Purchases tables.
- SUPPLIERID in the Produce, Animal Products, Grains, and Suppliers tables.
- TYPE in the Produce, Animal Products, and Grains tables.
- STOCKQTY in the Produce, Animal Products, and Grains tables.
- NXTDELIVERY in the Produce, Animal Products, and Grains tables.

I've highlighted the duplicate columns in the produced tables for easter identification.

Table Structure for Produce:

COLUMN_NAME	DATA_TYPE	NULLABLE
ITEMID	CHAR(5)	NO
SUPPLIERID	CHAR(10)	NO
PLUCODE	CHAR(4,2)	NO
PRODUCENAME	CHAR(15)	NO
TYPE	CHAR(10)	NO
STOCKQTY	NUMBER(4,2)	NO
NXTDELIVERY	DATE	NO

Table Structure for Animal Products:

COLUMN_NAME	DATA_TYPE	NULLABLE
ITEMID	CHAR(5)	NO
SUPPLIERID	CHAR(10)	NO
ANPRDNAME	CHAR(15)	NO
TYPE	CHAR(10)	NO
STOCKQTY	NUMBER(4,2)	NO
NXTDELIVERY	DATE	YES

Table Structure for Grains:

COLUMN_NAME	DATA_TYPE	NULLABLE
ITEMID	CHAR(5)	NO
SUPPLIERID	CHAR(10)	NO
GRAINNAME	CHAR(15)	NO
TYPE	CHAR(10)	NO
STOCKQTY	NUMBER(4,2)	NO
NXTDELIVERY	DATE	YES

Table Structure for Suppliers:

COLUMN_NAME	DATA_TYPE	NULLABLE
SUPPLIERID	CHAR(10)	NO
LASTDELIVERY	DATE	YES
SPECIALTY	CHAR(15)	YES
ACTIVE	CHAR(11)	NO

Table Structure for Purchases:

COLUMN_NAME	DATA_TYPE	NULLABLE
ITEMID	CHAR(5)	NO
TOTALBOUGHT	NUMBER(8,2)	YES
TOTALSOLD	NUMBER(8,2)	YES
TOTALREV	NUMBER(10,2)	YES
MARGIN	NUMBER(10,2)	YES

The database must conform to 1NF, ensuring unique field names, distinct values, consistent data types, and the presence of primary keys (Gibbs, M., 2020). The lack of identified primary keys can be addressed by assigning ITEMID as the primary key for the Purchases table and SUPPLIERID as the primary key for the Suppliers table. For tables with composite primary keys (Produce, Animal Products, and Grains), a unique primary key can be created. We can use GRAINID for the Grain table primary key, PRODUCEID for the Produce table primary key, and ANDPRID for the Animal Products table primary key.

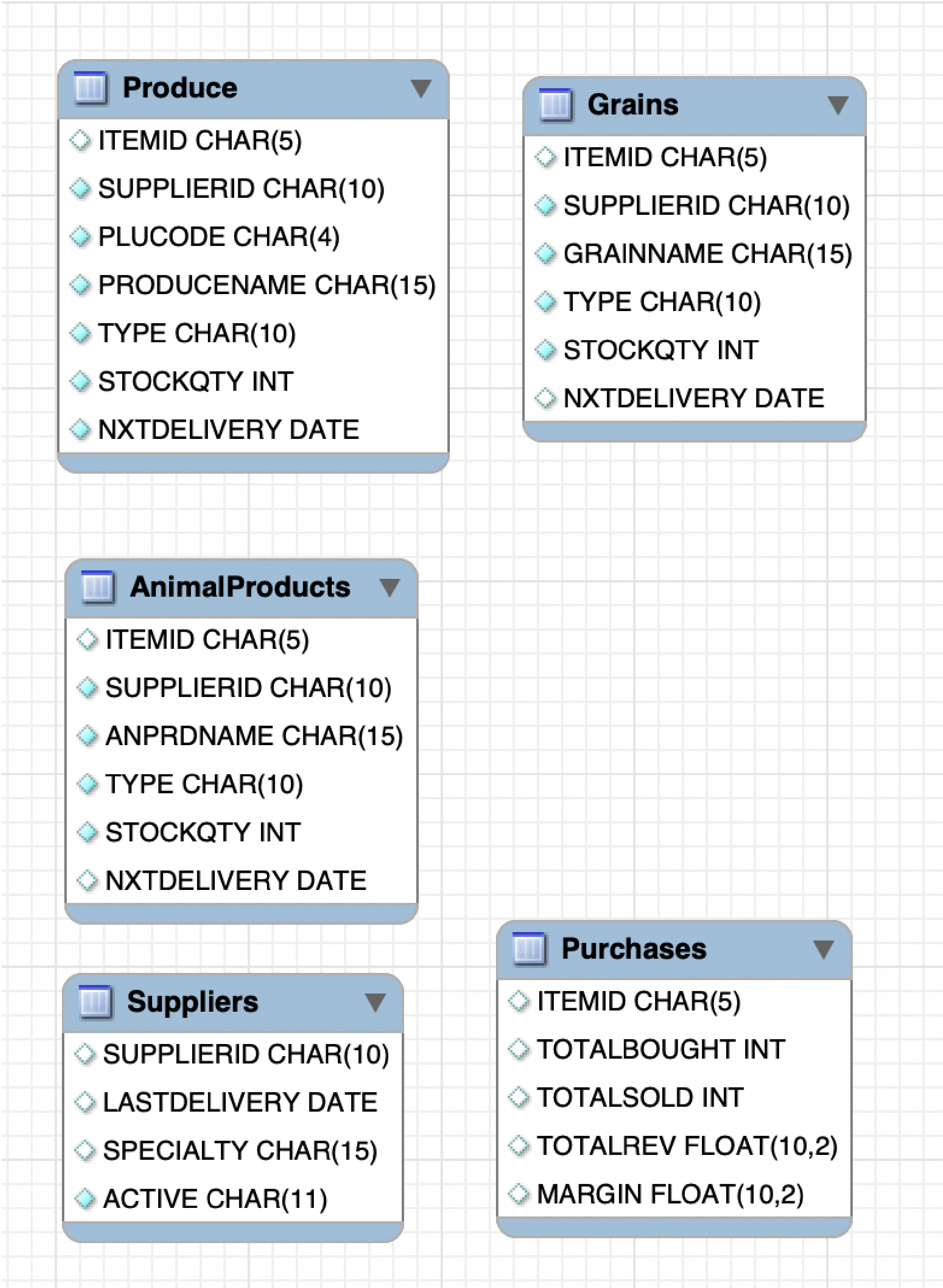
To achieve 2NF, the database must adhere to the first normal form and eliminate partial dependencies on the primary key. We want each attribute to be functionally dependent on a primary key. To achieve this, we can create additional tables to store information dependent on a subset of the primary key. For example, we can create a new table called SupplierInfo with the following fields: SUPPLIERID, LASTDELIVERY, SPECIALTY, and ACTIVE. This table will help eliminate partial dependencies on the SUPPLIERID in the Suppliers table.

The third normal form (3NF), further reduces dependencies. To achieve 3NF, we can create additional tables to store related information. A new table named

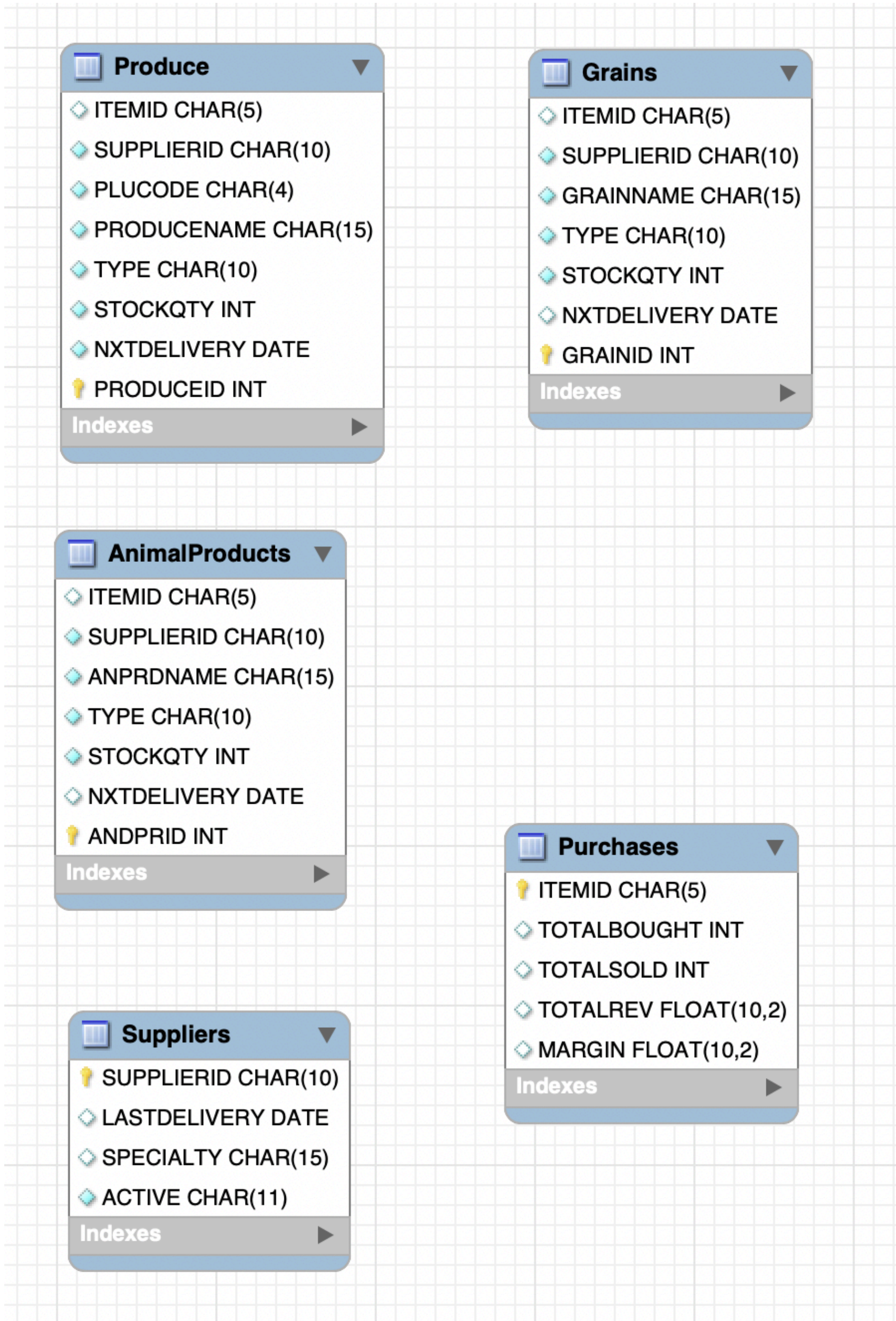
Product_Supply can be created with the fields ITEMID, SUPPLIERID, STOCKQTY, and NXTDELIVERY, eliminating transitive dependencies in the Produce, Animal Products, and Grains tables.

Additionally, a table named Purchases can be created with the fields ITEMID, TOTALBOUGHT, TOTALSOLD, TOTALREV, and MARGINID. To further normalize the database structure, we can also create a Profit_Margin table with the fields MARGINID and MARGIN. New tables can also be made for PLU_Codes, Produce_Products, Animal_Products, Product_Type, and Grain_Products to store information related to PLUCODE, PRODUCENAME, ANPRDNAME, TYPE, and GRAINNAME. These changes aim to bring the database into 3NF by minimizing dependences and streamlining structure.

The tables before normalization:

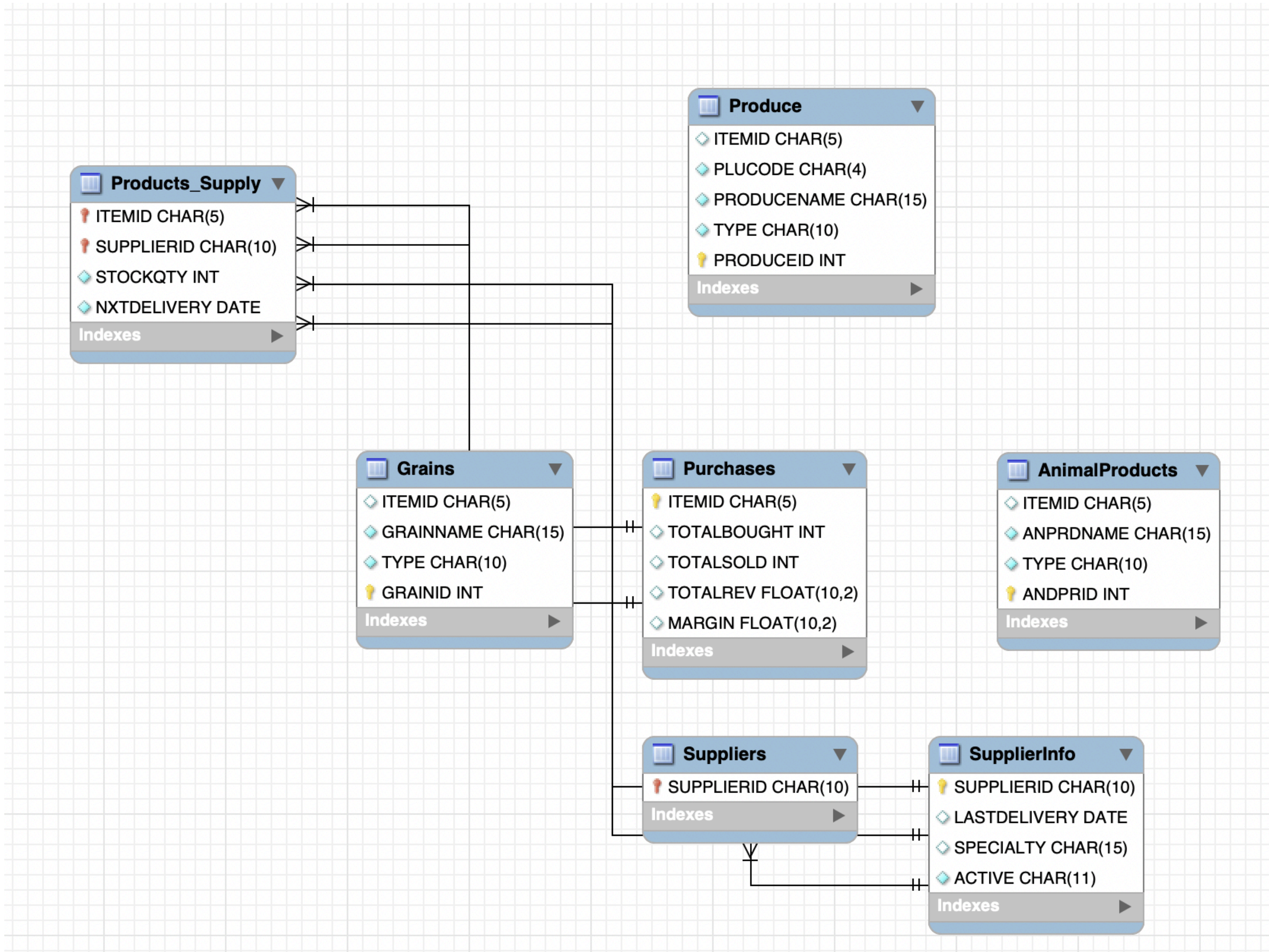


The tables in First Normal Form (1NF):



Each table now has a unique primary key.

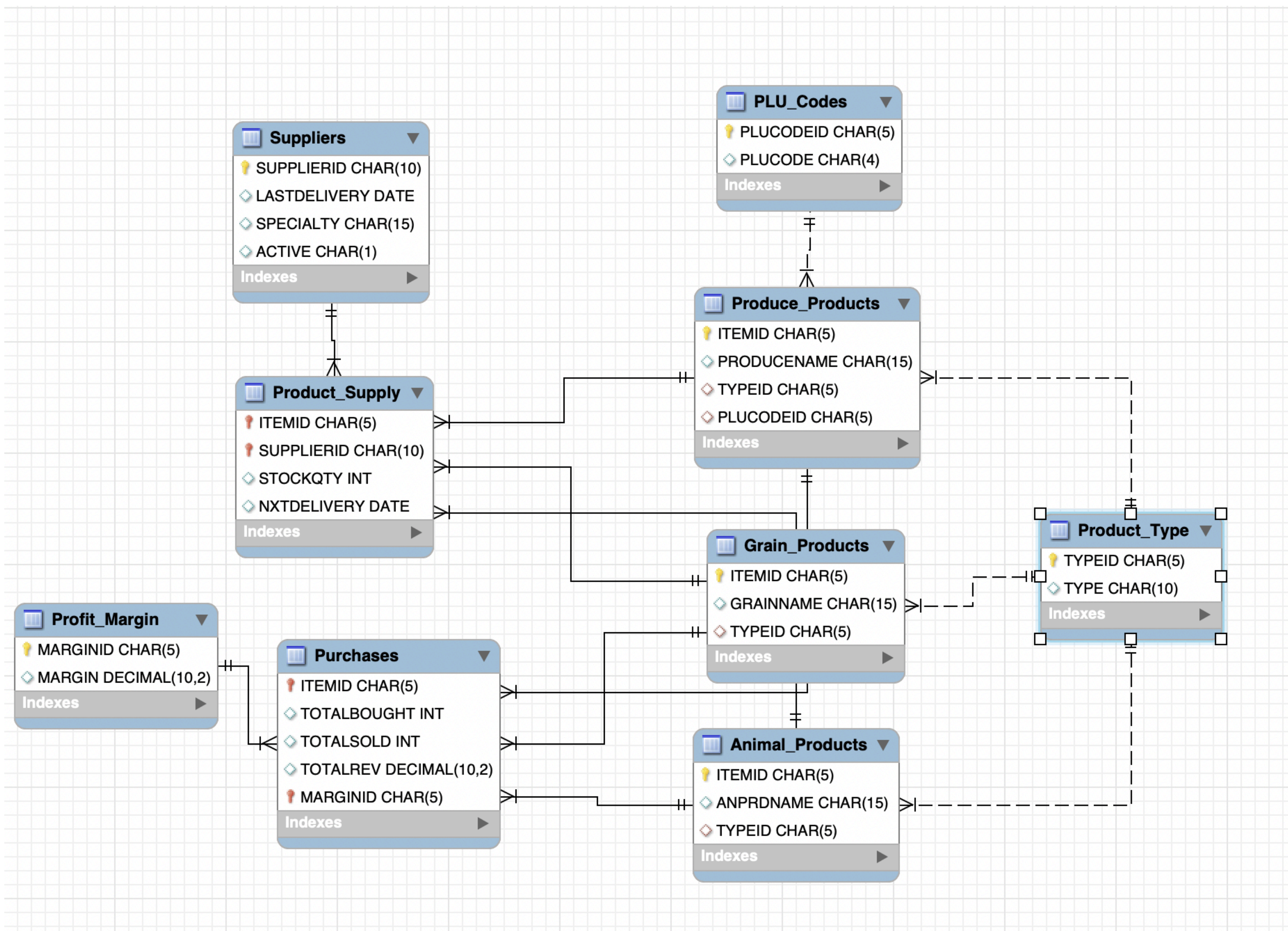
The tables in Second Normal Form (2NF):



Partial dependencies on a primary key are now eliminated. Each attribute is functionally dependent on a primary key.

Part 3 - Entity Relationship Diagram in Third Normal Form (3NF)

I created the following ER diagram using MySQL Workbench to model the miniature database in the third normal form (3NF), eliminating the redundancies to streamline the database as stated in the previous page of this report:



Dependencies are further reduced.

References

Gibbs, M. (2016, October 29). *SQL: inner joins*. Study. https://study.com/academy/lesson/sql-inner-joins.html

Gibbs, M. (2020, March 14). *Third normal form in DBMS with examples*. Study. https://study.com/academy/lesson/third-normal-form-in-dbms-with-examples.html

Gibbs, M. (2020, March 16). *First normal form in DBMS with examples*. Study. https://study.com/academy/lesson/first-normal-form-in-dbms-with-examples.html

Gibbs, M. (2022, May 6). *Second normal form in DBMS with examples.* Study. https://study.com/academy/lesson/second-normal-form-in-dbms-with-examples.html

Gibbs, M. (2023, June 19). *Cardinality in database design: examples and crow's foot notation*. Study. https://study.com/academy/lesson/cardinality-in-database-design-examples-and-crows-foot-notation.html

Gloag, D. (2016, September 7). *SQL normalization: example & rules*. Study. https://study.com/academy/lesson/sql-normalization-example-rules.html

McGrath, Mike. *SQL in Easy Steps.* 4th ed., In Easy Steps, 2020.