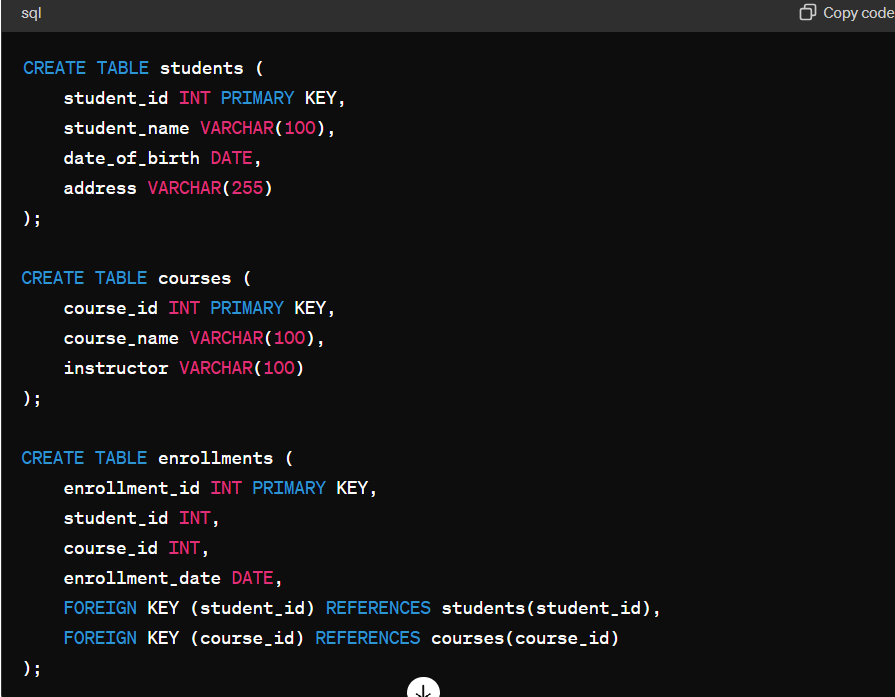
**Denormalization :**

Denormalization involves deliberately introducing redundancy into a database schema to optimize query performance or simplify data retrieval. Here's an example of denormalization:

Let's consider a scenario where we have two normalized tables: **students** and **courses**. The **students** table stores information about students, and the **courses** table stores information about courses they are enrolled in.

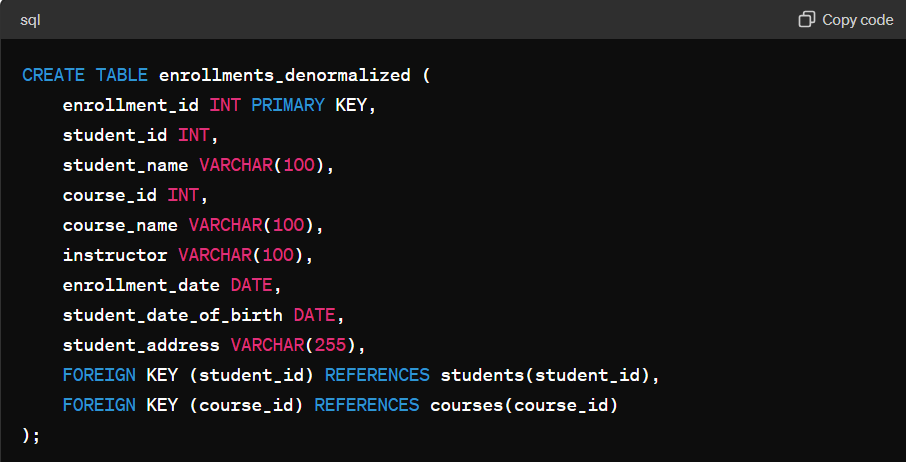
Normalized Structure:



In this normalized structure, we have separate tables for students and courses, and the **enrollments** table serves as a junction table to represent the many-to-many relationship between students and courses.

Now, let's denormalize this structure by including some redundant information:

Denormalized Structure:



In this denormalized structure, we've duplicated some fields from the **students** and **courses** tables into the **enrollments\_denormalized** table. By doing so, we eliminate the need for joins when querying enrollment information along with student and course details. This can lead to improved query performance, especially in scenarios where such queries are frequent.

However, denormalization also comes with trade-offs, including increased storage requirements and the risk of data inconsistency if redundant information is not properly maintained. Therefore, it's essential to carefully consider the specific requirements and trade-offs before opting for denormalization.