Tables

Students

student_id	name	major
1	Alice Smith	Biology
2	Bob Johnson	Computer Science
3	Clara White	Math

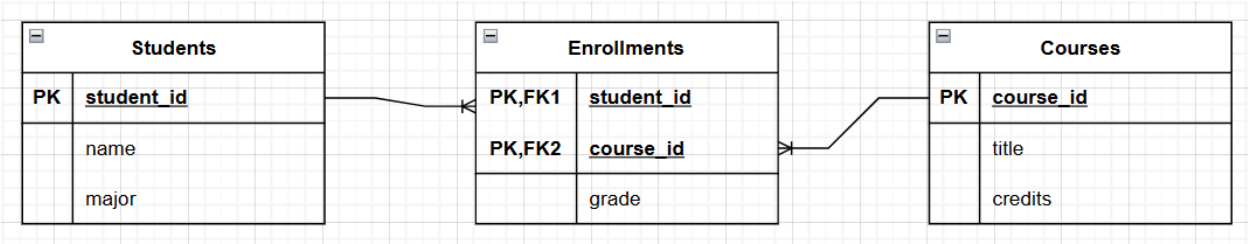
Courses

course_id	title	credits
101	Intro to Biology	4
102	Data Structures	3
103	Calculus I	4

Enrollments

student_id	**course_id**	grade
1	101	89
2	102	92
3	103	85
2	103	76

Entity-relationship diagram



Basic terminology	
Table	A collection of related data held in a structured format with rows and columns.
Field	A column in a table, representing a data attribute.
Record	A row in a table, representing one entry.
Entity	A real-world object represented in a database (e.g., Student, Course).
Attribute	A property of an entity (e.g., name, major).
One-to-many	One record in Table A relates to many in Table B (e.g., one course has many enrollments).
Many-to-many	Records in Table A relate to many in Table B and vice versa (e.g., many students take many courses).
Entity-relationship diagram	A visual representation of entities and their relationships.
Primary key	A field uniquely identifying a record.
Composite key	A primary key made up of two or more fields that together uniquely identify a record.
Foreign key	A field in one table that links to the primary key in another table.

We want to design tables where the attributes depend on the primary key, on the whole key, and nothing but the key.
We can have a table for each entity, with the tables' fields being the entity's attributes, and a table for each relationship between entities.

Primary commands SQL keywords do not need to be uppercase, but making them uppercase makes queries more readable. All commands should end with **;**		
CREATE TABLE	Defines a new table in the database along with its columns, data types, and constraints (like primary keys or foreign keys).	**CREATE TABLE Students (** ** student_id INT PRIMARY KEY,** ** name VARCHAR(100),** ** major VARCHAR(50)** **);**
SHOW TABLES	List all tables in the database. This is a MySQL command.	**SHOW TABLES;**
DESCRIBE	List all of the columns in a table and information about those columns. This is a MySQL command.	**DESCRIBE Students;**
DROP TABLE	Permanently deletes a table and all its data and structure from the database.	**DROP TABLE Students;**
INSERT INTO ... VALUES	Adds new rows of data into a table.	**INSERT INTO Students (student_id, name, major)** **VALUES (4, 'Daniel Grey', 'Physics');**
DELETE FROM ... WHERE	Removes rows from a table that meet a specified condition.	**DELETE FROM Students WHERE student_id = 4;**
UPDATE ... SET ... WHERE	Modifies existing data in a table for rows that meet a given condition.	**UPDATE Students SET major = 'Mathematics' WHERE student_id = 3;**
SELECT ... FROM	Retrieves data from one or more tables. Can be used with filters, sorting, aggregation, and joins.	**SELECT * FROM Students;**

Query modifiers		
TOP	Limits the result to a specified number of rows	**SELECT TOP 2 * FROM Students;**
DISTINCT	Eliminates duplicate values in the result for the selected column(s).	**SELECT DISTINCT major FROM Students;**
ORDER BY	Sorts the result set by one or more columns, either ascending (**ASC**) or descending (**DESC**).	**SELECT * FROM Enrollments ORDER BY grade DESC;**
UNION	Combines the result sets of two **SELECT** statements and removes duplicates. Both queries must have the same number of columns.	**SELECT name FROM Students** **UNION** **SELECT title FROM Courses;**
WHERE	Filters rows based on a specified condition.	**SELECT * FROM Enrollments WHERE grade <> 85;**
AND	Combines multiple conditions; all must be true.	**SELECT * FROM Enrollments WHERE grade > 80 AND course_id = 103;**
OR	Combines multiple conditions; at least one must be true.	**SELECT * FROM Enrollments WHERE grade > 80 OR course_id = 103;**
BETWEEN	Tests whether a value is in between two values, inclusive.	**SELECT * FROM Courses WHERE course_id BETWEEN 101 AND 103;**
IN	Tests whether a value matches any value in a list.	**SELECT * FROM Courses WHERE course_id IN (101, 103);**
LIKE	Matches a pattern in a string. ● **%** matches zero or more characters ● **_** matches a single character ● **[ax]** matches one character, either **a** or **x** ● **[a-k]** matches one character in the range **a**-**k**	**SELECT * FROM Students WHERE name LIKE 'A%';**

NOT	Negates a condition.	`SELECT * FROM Courses WHERE NOT course_id IN (101, 103);`
MIN	Finds the smallest value in a column.	`SELECT MIN(grade) FROM Enrollments;`
MAX	Finds the largest value in a column.	`SELECT MAX(grade) FROM Enrollments;`
SUM	Calculates the total sum of a numeric column.	`SELECT SUM(credits) FROM Courses;`
AVG	Calculates the average of a numeric column.	`SELECT AVG(grade) FROM Enrollments;`
STDEV	Returns the standard deviation of a numeric column	`SELECT STDEV(grade) FROM Enrollments;`
COUNT	Counts the number of rows (or non-null values in a column).	`SELECT COUNT(*) FROM Students;`
GROUP BY	Groups rows that have the same values in some columns into summary rows (often used with aggregate functions like **AVG**, **SUM**, **COUNT**).	`SELECT student_id, AVG(grade) FROM Enrollments GROUP BY student_id;`

HAVING	Filters groups created by **GROUP BY**, similar to **WHERE** but for aggregates.	`SELECT student_id, AVG(grade) AS avg_grade`
		`FROM Enrollments`
		`GROUP BY student_id`
		`HAVING AVG(grade) > 80;`

INNER JOIN	Returns only rows with matching values in both tables.	`SELECT Students.name, Courses.title, Enrollments.grade`
		`FROM Enrollments`
		`INNER JOIN Students ON Enrollments.student_id = Students.student_id;`

LEFT JOIN	Returns all rows from the left table, and matched rows from the right. If no match, **NULL**.	`SELECT Students.name, Enrollments.grade`
		`FROM Students`
		`LEFT JOIN Enrollments ON Students.student_id =`
		`Enrollments.student_id;`

RIGHT JOIN	Returns all rows from the right table, and matched rows from the left. If no match, **NULL**.	`SELECT Courses.title, Enrollments.grade`
		`FROM Courses`
		`RIGHT JOIN Enrollments ON Courses.course_id = Enrollments.course_id;`

FULL OUTER JOIN	Returns all rows when there's a match in either table. If no match, **NULL**.	**SELECT Students.name, Courses.title** **FROM Students** **FULL OUTER JOIN Enrollments ON Students.student_id = Enrollments.student_id;**