

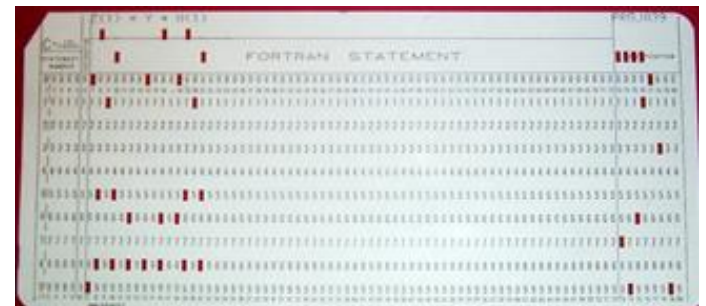
# SQL (Structured Query Language)

Amos Azaria

# SQL (Structured Query Language)

- SQL (in SQL server pronounced Seek Well).
- A language for executing commands on a database.
- Commands are based on Relational Algebra.
- Case *insensitive* language. Convention is to use upper case for all SQL keywords.

One of the very few case insensitive languages still used today. Old programming languages (Fortran, Cobol, Lisp, Basic) were case insensitive since they were designed for punched cards, which did not differentiate between lower case and upper case.



# Our tables

students

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

courses

id	name	lecturer	year	semester
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Program...	David Gol	2022	2

grades

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

# SELECT Command

- SELECT id FROM students
- SELECT id\*2 FROM students
- SELECT year, semester FROM courses
- SELECT \* FROM grades

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

id	name	lecturer	year	semester
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Program...	David Gol	2022	2

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

id	id*2	year	semester
111	222	2020	1
444	444	2021	1
222	666	2022	1
333	888	2022	1
		2022	2

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

# DISTINCT (unique)

- `SELECT DISTINCT year, semester FROM courses;`

year	semester
2020	1
2021	1
2022	1
2022	2

# WHERE Keyword

- SELECT id FROM students WHERE degree < 2
- SELECT age FROM students WHERE firstName LIKE '%i'

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

id	name	lecturer	year	semst
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Program...	David Gol	2022	2

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

id
111
333
444

age
23
24

# conditions

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

- >, <, =, <=, >=, <>
- AND, OR, NOT
- BETWEEN

- SELECT \* FROM grades WHERE grade BETWEEN 80 and 90

	courseId	studentId	grade	passed
	30	111	90	1
	20	222	85	1

- IN

- SELECT \* FROM grades WHERE studentId IN (111,444)

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

courseId	studentId	grade	passed
20	111	43	0
30	111	90	1
30	444	95	1

- LIKE

– SELECT \* FROM students WHERE firstName LIKE 'Chay%'

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass

# Nested Queries

- SQL is Compositional: The result of a select query is a relation!
- SELECT lecturer FROM courses WHERE id NOT IN (SELECT courseId FROM grades)

List the lecturers that did not feed any grades yet.

lecturer
Knows Nothing
Adel Smith

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

id	name	lecturer	year	sem
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Program...	David Gol	2022	2

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

Write an SQL query that lists all last names of students that failed:

- SELECT lastName FROM student WHERE id IN (SELECT studentId FROM grades WHERE passed = 0)

lastName
Glass
Golan



# Basic set operations

- UNION

Number of attributes in both queries must match

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

- (SELECT id,lastName FROM students WHERE id < 200)

UNION (SELECT id,lastName FROM students WHERE id BETWEEN 300 AND 400)

DISTINCT

id	lastName
111	Glass
333	Golan

- The following are not supported in MySQL but can be easily emulated with equivalent queries:

- INTERSECT

Can be emulated using "IN"

- EXCEPT

Can be emulated using "NOT IN"

# NULL

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan
444	23	0	1	Moti	Cohen
555	24	0	NULL	NULL	NULL
666	27	1	NULL	Tamar	NULL

- NULL denotes a missing value.
  - SELECT \* FROM students WHERE lastName IS NULL

id	age	gender	degree	firstName	lastName
555	24	0	NULL	NULL	NULL
666	27	1	NULL	Tamar	NULL

- SELECT \* FROM students WHERE degree = 1 OR degree <> 1

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan
444	23	0	1	Moti	Cohen

# COALESCE

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan
444	23	0	1	Moti	Cohen
555	24	0	NULL	NULL	NULL
666	27	1	NULL	Tamar	NULL

- COALESCE(val1, val2, val3 ...): returns the first value that is not NULL
  - SELECT id, COALESCE(lastName, firstName, 'unknown') FROM students

id	COALESCE(lastName, firstName, 'unknown')
111	Glass
222	Negev
333	Golan
444	Cohen
555	unknown
666	Tamar

# INSERT INTO

- Inserting new rows to a table:
  - INSERT INTO courses  
(id,name,lecturer,year,semester) VALUES (66,  
'databases', null, 2025, 1);

id	name	lecturer	year	semester
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Programming	David Gol	2022	2
66	databases	NULL	2025	1

# ORDER BY

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

SELECT id,firstName FROM students ORDER BY  
lastName

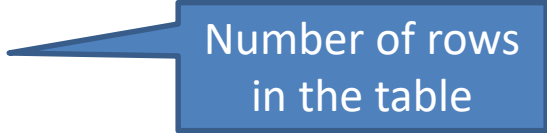
id	firstName
444	Moti
111	Chaya
333	Gadi
222	Tal

SELECT gender,age,lastName FROM students ORDER BY gender ASC, age  
DESC

gender	age	lastName
0	24	Golan
0	23	Cohen
1	28	Negev
1	21	Glass

# Aggregate Functions

- SELECT **COUNT(\*)** FROM students
- SELECT **AVG(grade)** FROM grades
- SELECT **SUM(passed)** FROM grades
- SELECT **MAX(grade)** FROM grades
- SELECT **MIN(grade)** FROM grades



Number of rows  
in the table

# GROUP BY

id	name	lecturer	year	semester
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Programming	David Gol	2022	2
66	databases	NULL	2025	1

- Suppose we want to know how many courses we have every year:
- `SELECT year, COUNT(year) FROM courses`

year	count(year)
2020	6

- `SELECT year, COUNT(year) FROM courses GROUP BY year`
- `SELECT name, COUNT(lecturer) FROM courses GROUP BY semester`

year	count(year)
2020	1
2021	1
2022	3
2025	1

Picks a (random) entry from each group

name	COUNT(lecturer)
Introduction to intro.	4
Advanced Programming	1

Only 4 in semester 1 because we count lecturer, and one of them is NULL.

# GROUP BY (cont)

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

- Write a query that returns the average grade for every course:
  - SELECT courseId, AVG(grades) FROM grades GROUP BY courseId
- Write a query that returns the maximum grade for every student:
  - SELECT studentId, MAX(grades) FROM grades GROUP BY studentId



courseId	studentId	grade	passed
20	111	43	0
30	111	90	1
20	222	85	1
40	222	67	1
40	333	40	0
30	444	95	1

# HAVING

- HAVING is a condition on the group.
- SELECT year, COUNT(year) FROM courses GROUP BY year HAVING COUNT(year) > 1

year	count(year)
2022	3

- WHERE vs. HAVING:
  - WHERE is done before the grouping and HAVING is done after it
- E.g. courses in which the average grade of students who *passed* is under 70:
  - SELECT courseId,AVG(grade) FROM grades WHERE passed > 0 GROUP BY courseId HAVING AVG(grade) < 70;

Exempli  
Gratia

courseId	avg(grade)
40	67.0000

# Query execution order

- `SELECT courseId, AVG(grade) FROM (SELECT * FROM grades) WHERE passed > 0 GROUP BY courseId HAVING AVG(grade) < 70;`
- Like in math, we first process the most inner query.
- Then we look at the FROM part to know which table we want (or joined tables – see next slides).
- Then the WHERE predicate to know which rows we are interested in.
- Then the GROUP BY.
- Then the HAVING.
- Then we look at the SELECT to know which columns to show.

# students x grades

- SELECT \* FROM students, grades

id	age	gender	degree	firstName	lastName	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	20	111	43	0
222	28	1	3	Tal	Negev	20	111	43	0
333	24	0	1	Gadi	Golan	20	111	43	0
444	23	0	1	Moti	Cohen	20	111	43	0
111	21	1	1	Chaya	Glass	30	111	90	1
222	28	1	3	Tal	Negev	30	111	90	1
333	24	0	1	Gadi	Golan	30	111	90	1
444	23	0	1	Moti	Cohen	30	111	90	1
111	21	1	1	Chaya	Glass	20	222	85	1
222	28	1	3	Tal	Negev	20	222	85	1
333	24	0	1	Gadi	Golan	20	222	85	1
444	23	0	1	Moti	Cohen	20	222	85	1
111	21	1	1	Chaya	Glass	40	222	67	1
222	28	1	3	Tal	Negev	40	222	67	1
333	24	0	1	Gadi	Golan	40	222	67	1
444	23	0	1	Moti	Cohen	40	222	67	1
111	21	1	1	Chaya	Glass	40	333	40	0
222	28	1	3	Tal	Negev	40	333	40	0
333	24	0	1	Gadi	Golan	40	333	40	0
444	23	0	1	Moti	Cohen	40	333	40	0
111	21	1	1	Chaya	Glass	30	444	95	1
222	28	1	3	Tal	Negev	30	444	95	1
333	24	0	1	Gadi	Golan	30	444	95	1
444	23	0	1	Moti	Cohen	30	444	95	1

# INNER JOIN

- Suppose we want to get students full information (not just ids) with their grades
- `SELECT * FROM students, grades WHERE student.id = grades.studentId`
- `SELECT * FROM students INNER JOIN grades ON students.id = grades.studentId`

id	age	gender	degree	firstName	lastName	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	20	111	43	0
111	21	1	1	Chaya	Glass	30	111	90	1
222	28	1	3	Tal	Negev	20	222	85	1
222	28	1	3	Tal	Negev	40	222	67	1
333	24	0	1	Gadi	Golan	40	333	40	0
444	23	0	1	Moti	Cohen	30	444	95	1

# Multiple consecutive joins

- SELECT \* FROM students INNER JOIN grades on students.id = grades.studentId INNER JOIN courses on grades.courseId = courses.id

id	age	gender	degree	firstName	lastName	courseId	studentId	grade	passed	id	name	lecturer	year	semester
111	21	1	1	Chaya	Glass	20	111	43	0	20	Calculus	Tamar Ezra	2021	1
111	21	1	1	Chaya	Glass	30	111	90	1	30	Algebra	Shay Mann	2022	1
222	28	1	3	Tal	Negev	20	222	85	1	20	Calculus	Tamar Ezra	2021	1
222	28	1	3	Tal	Negev	40	222	72	1	40	Advanced Programming	David Gol	2022	2
333	24	0	1	Gadi	Golan	40	333	45	0	40	Advanced Programming	David Gol	2022	2
444	23	0	1	Moti	Cohen	30	444	95	1	30	Algebra	Shay Mann	2022	1

# LEFT OUTER JOIN

- When using left outer join, *all* rows FROM left table appear in the result, even if they have no match (they match nulls).
- INSERT INTO students (id, age, gender, degree, firstName, lastName) VALUES (700, 26, 1, 2, 'Maya', 'Levi');
- SELECT \* FROM students **INNER JOIN** grades ON students.id = grades.studentId
- SELECT \* FROM students **LEFT JOIN** grades ON students.id = grades.studentId

A new student  
has just joined!

id	age	gender	degree	firstName	lastName	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	20	111	43	0
111	21	1	1	Chaya	Glass	20	111	43	0
111	21	1	1	Chaya	Glass	30	111	90	1
222	28	1	3	Tal	Negev	20	222	85	1
222	28	1	3	Tal	Negev	40	222	67	1
333	24	0	1	Gadi	Golan	40	333	40	0
444	23	0	1	Moti	Cohen	30	444	95	1
700	26	1	2	Maya	Levi	NULL	NULL	NULL	NULL

# RIGHT OUTER JOIN

We have a new grade  
for an unregistered  
student...

- INSERT INTO grades (courseId, studentId, grade, passed) values (30, 600, 82, 1);
- SELECT \* FROM students **RIGHT JOIN** grades  
ON students.id = grades.studentId

id	age	gender	degree	firstName	lastName	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	20	111	43	0
111	21	1	1	Chaya	Glass	30	111	90	1
222	28	1	3	Tal	Negev	20	222	85	1
222	28	1	3	Tal	Negev	40	222	67	1
333	24	0	1	Gadi	Golan	40	333	40	0
444	23	0	1	Moti	Cohen	30	444	95	1
NULL	NULL	NULL	NULL	NULL	NULL	30	600	82	1

# FULL OUTER JOIN

- MySQL doesn't support full outer join, but this can be accomplished by uniting a LEFT JOIN with a RIGHT JOIN:
- (SELECT \* FROM students **LEFT JOIN** grades ON students.id = grades.studentId) **UNION** (SELECT \* FROM students **RIGHT JOIN** grades ON students.id = grades.studentId)

id	age	gender	degree	firstName	lastName	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	20	111	43	0
111	21	1	1	Chaya	Glass	30	111	90	1
222	28	1	3	Tal	Negev	20	222	85	1
222	28	1	3	Tal	Negev	40	222	67	1
333	24	0	1	Gadi	Golan	40	333	40	0
444	23	0	1	Moti	Cohen	30	444	95	1
700	26	1	2	Maya	Levi	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	30	600	82	1



# UPDATE

- **UPDATE** grades **SET** grade=78, passed=1  
WHERE studentId=111 AND courseId = 20

- **SELECT \* FROM** grades

courseId	studentId	grade	passed
20	111	78	1
30	111	90	1
20	222	85	1
40	222	67	1
40	333	40	0
30	444	95	1
30	600	82	1

- **UPDATE** grades **SET** grade=grade+5 **WHERE**  
courseId=40

courseId	studentId	grade	passed
20	111	78	1
30	111	90	1
20	222	85	1
40	222	72	1
40	333	45	0
30	444	95	1
30	600	82	1

# DELETE

- DELETE FROM grades WHERE studentId=600  
OR courseId=20
- (3 rows affected)
- SELECT \* FROM grades:

courseId	studentId	grade	passed
30	111	90	1
40	222	67	1
40	333	40	0
30	444	95	1

# CRUD

- Basic 4 operations on data:
  - Create: INSERT (CREATE)
  - Read: SELECT
  - Update: UPDATE (ALTER)
  - Delete: DELETE (DROP)

# CREATE TABLE

- Creating a table is usually done using the GUI, but can also be done using the command-line.
- A pet table in a veterinarianian DBMS:
  - CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20), sex CHAR(1), birth DATE);

Additional SQL types:

INT

REAL: float/double

BOOLEAN

XML

DATETIME – e.g. '2015-05-21 23:28:01'

See more at:

<https://www.techonthenet.com/mysql/datatypes.php>

CHAR(X) – Will always consume X bytes.

VARCHAR(x) – Will consume as many bytes as the input (+1) upto x

TEXT – upto 65K chars

LONGTEXT – over 4GB

# Keys

- **PRIMARY KEY:** a column or set of columns that identify the entry (may not be NULL). E.g. id in the students table, or the columns of studentId and courseId in the grades table.
- **UNIQUE KEY:** unique but may be NULL, e.g. a passport number column (not everyone has a passport, but no two people have the same passport number).
- **INDEX (or just KEY):** allows faster indexing. We will usually define as indexes attributes that are likely to be used in joins or appear in the WHERE clause (e.g. lastName). Indexes improve the DBMS's performance as it won't need to read all entries in order to gather those satisfying the condition (e.g. all students whose last name is 'Cohen').
- All Keys are stored in B-trees (or hash-indexes).
- We will discuss keys in detail when we talk about normalization.

# CREATE TABLE (with keys)

PRIMARY KEY and  
UNIQUE can appear  
right after type

- `CREATE TABLE pet2 (petId INT PRIMARY KEY,  
name VARCHAR(20), ownerId INT NOT NULL,  
species VARCHAR(20), sex CHAR(1), birth  
DATE, INDEX(ownerId));`

INDEX (or KEY) must be  
defined after a comma

Note: NOT  
NULL

# Integrity Constraints

- CHECK keyword E.g.
  - CHECK (country IN ('USA','UK','Israel','India'))
  - CREATE TABLE WorkingDay (work\_day DATE, income REAL, expenses REAL, revenue REAL, CHECK(revenue=income-expenses));
- Unfortunately, MySQL does not support Integrity Constraints.

# DROP TABLE

- DROP TABLE deletes a table:
  - DROP TABLE pet2
  - [DELETE TABLE pet2 – returns an error, since DELETE is used to remove entries (rows), with the following syntax: DELETE ... FROM ...]

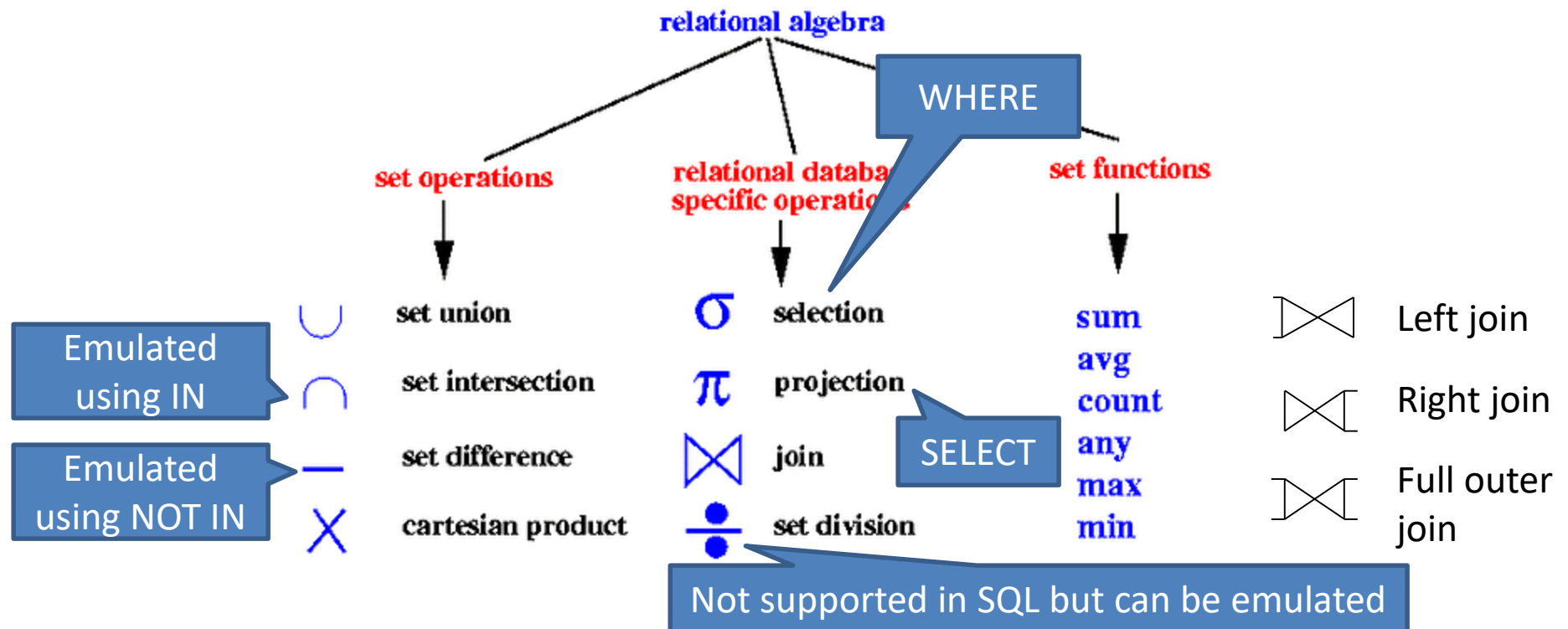


# ALTER

- The command ALTER is used to modify a table:
  - ALTER TABLE pet ADD death DATE
  - ALTER TABLE pet DROP death

# Relational Algebra

- Relational algebra defines commands similar to those found in SQL (unfortunately with different names). Each command has a symbol.



# Relational Algebra Examples

- SELECT firstName, id FROM students WHERE degree > 1

$$\Pi_{firstName, id}(\sigma_{(degree > 1)}(students))$$

- SELECT students.lastName, students.id, grades.grade FROM students RIGHT JOIN grades ON students.Id=grades.studentId WHERE grades.courseId=20;

$$\Pi_{(lastName, id, grade)}(\sigma_{(courseId=20)}(students \bowtie_{id=studentId} grades))$$