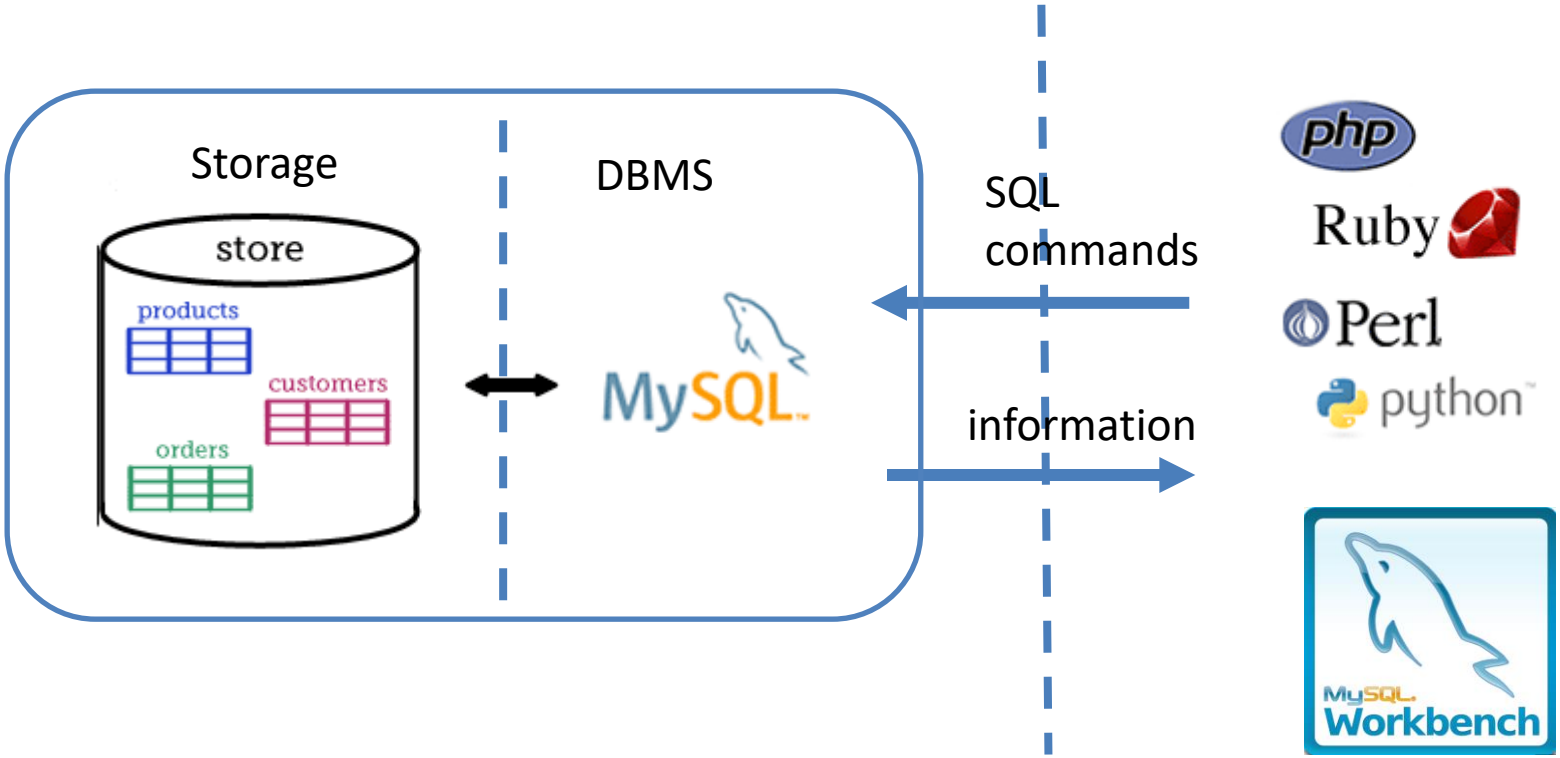
SQL (Structured Query Language)

Amos Azaria, Netanel Chkroun

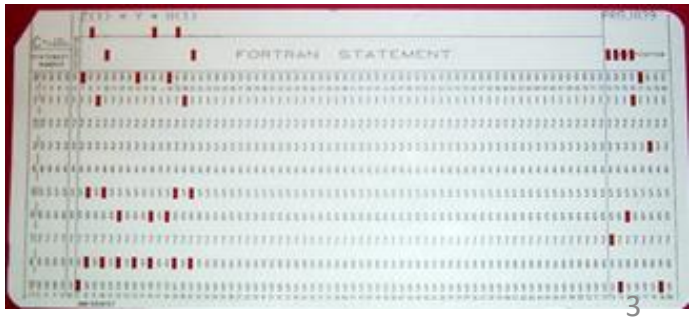
DBMS Architecture



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 programing languages (Fortran, Cobol, Lisp, Basic, Pascal) 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

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

courses

id	name	lecturer	year	semster
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

SELECT Command

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	semster
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

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

id
111
444
222
333

id*2
222
444
666
888

year	semster
2020	1
2021	1
2022	1
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)

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

- LIKE

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 * FROM students WHERE firstName LIKE 'Chay%'

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

Nested Queries

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	sems
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

- 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

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

DEMO -union

students

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

employees

id	employee_first_name	employees_last_name	employees_city
222	tal	negev	holon
444	moti	cohen	holon
777	lea	yosef	ariel
888	rachel	meir	holon
999	eli	yaron	bat yam

Basic set operations

Number of attributes in both queries must match

- UNION

SELECT id, firstName
FROM students
WHERE city = 'holon'
UNION SELECT id, employee_first_name FROM employees
WHERE employee_city = 'holon'

distinct

id	firstName
222	Tal
444	Moti
888	rachel

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

Can be emulated using "IN"

Can be emulated using "NOT IN"

INTERSECT & EXCEPT

```
SELECT * FROM employees
WHERE id IN (SELECT id FROM
students)
```

```
SELECT * FROM employees
WHERE id NOT IN (SELECT id FROM
students)
```

NULL

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
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

NULL Evaluation

First Name	Last Name	Salary	Bonus
John	Smith	1000	500
Mary	Smith	1000	1500
Peter	White	1800	NULL
Nick	Perry	1000	NULL

SELECT * **FROM** employee **WHERE** bonus + salary > 1300

1800 + NULL > 1300 ??

NULL > -500 ??

First Name	Last Name	Salary	Bonus
John	Smith	1000	500
Mary	Smith	1000	1500

NULL (cont): 3 Value Logic

- SQL expression can evaluate to each of the following truth values:
  - True
  - False
  - Unknown
- Any comparison with NULL is evaluated as 'Unknown'.
  - This is because a NULL value is assumed to be missing, and unknown.
  - Even (NULL=NULL) is evaluated as unknown, because each of the values may be different
- WHERE (and HAVING) clauses require 'True' so 'Unknown' is treated as 'False'.

3 Value Logic Truth Tables

AND	True	False	Unknown
True			
False			
Unknown			

OR	True	False	Unknown
True			
False			
Unknown			

NOT	
True	
False	
Unknown	

3 Value Logic Truth Tables (filled)

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

NOT	
True	False
False	True
Unknown	Unknown

COALESCE

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

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

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

INSERT INTO

- Inserting new rows to a table:

- INSERT INTO courses (id,name,lecturer,year,semester) VALUES (66, 'databases', null, 2025, 1);

- SELECT * from 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 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

LIMIT

- SELECT * FROM students LIMIT 2;

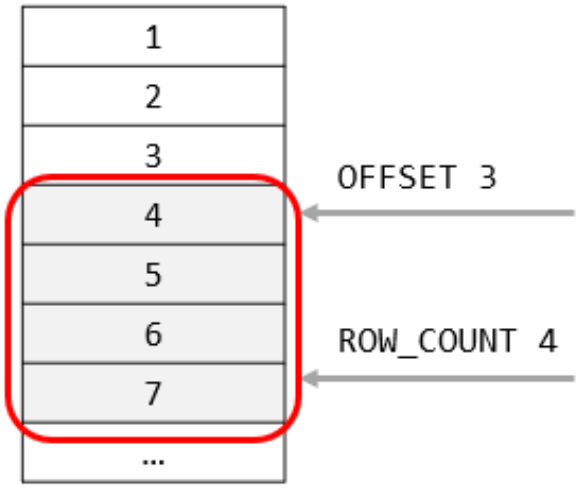
id	age	gender	degree	firstName	lastName
111	21	1	1	Chava	Glass
222	28	1	3	Tal	Negev
NULL	NULL	NULL	NULL	NULL	NULL

- SELECT * FROM students ORDER BY firstName LIMIT 2;

id	age	gender	degree	firstName	lastName
111	21	1	1	Chava	Glass
333	24	0	1	Gadi	Golan
NULL	NULL	NULL	NULL	NULL	NULL

Limit

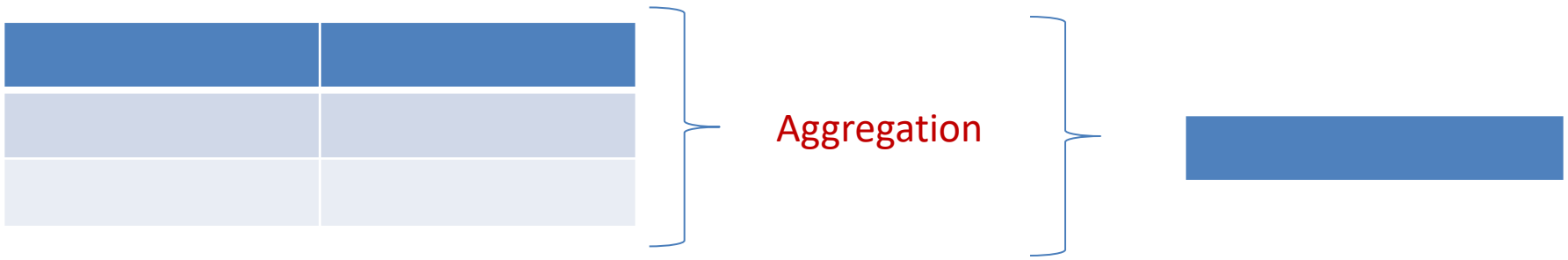
```
SELECT n FROM t
ORDER BY n
LIMIT 3, 4;
```

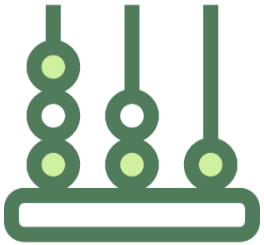


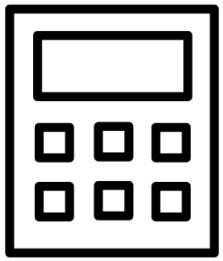
- SELECT * FROM students ORDER BY firstName LIMIT 2, 2

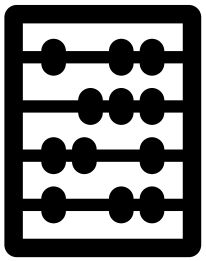
id	age	gender	degree	firstName	lastName
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
NULL	NULL	NULL	NULL	NULL	NULL

Aggregate Functions









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

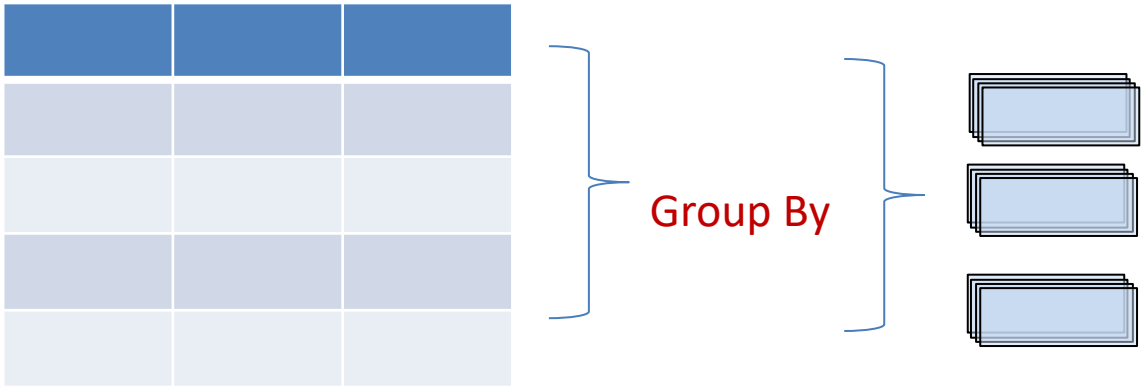
Note: AVG() function does not consider the NULL values during its calculation.

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

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

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

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

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.

36 17:28:41 SELECT name, COUNT(lecturer) FROM courses GROUP BY semester LIMIT 0, 1000 Error Code: 1055. Expression #1 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'test.courses... 0.000 sec

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

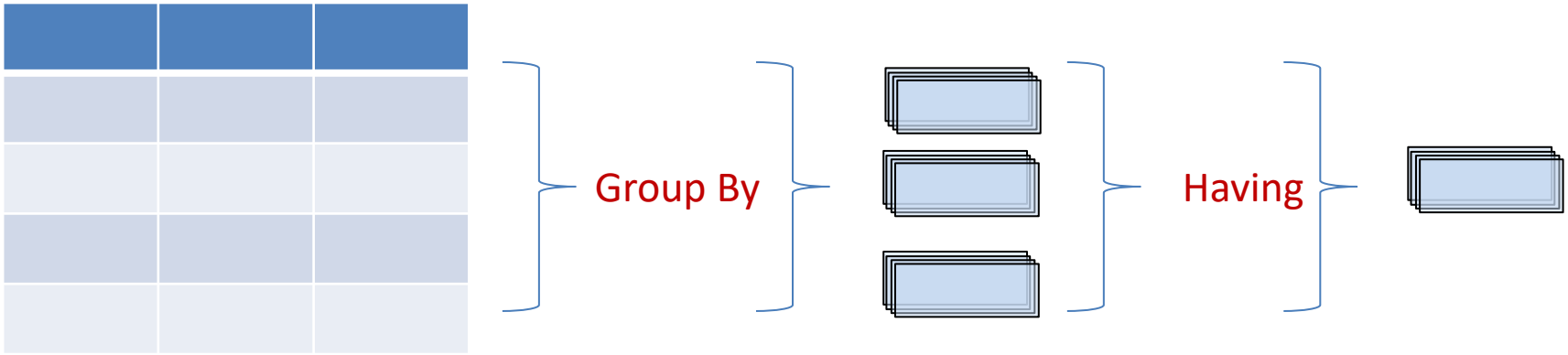
- Write a query that returns the average grade for every course: 
  - SELECT courseId, AVG(grade) FROM grades GROUP BY courseId
- Write a query that returns the maximum grade for every student: 
  - SELECT studentId, MAX(grade) FROM grades GROUP BY studentId

	courseId	AVG(grade)
	20	64.0000
	30	92.5000
	40	53.5000

	studentId	MAX(grade)
	111	90
	222	85
	333	40
	444	95

Having

Condition on the group



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

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

- 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 DISTINCT courseId, AVG(grade) FROM grades WHERE passed > 0 GROUP BY courseId HAVING AVG(grade) < 70 ORDER BY courseId, LIMIT 2;
- We first 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.
- Then the DISTINCT removes identical rows
- The ORDER BY sorts the results
- The LIMIT presents only the requested rows

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

Retrieving data from two tables

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 * 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 students.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

Multiple consecutive joins

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

1

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	72	1
333	24	0	1	Gadi	Golan	40	333	45	0
444	23	0	1	Moti	Cohen	30	444	95	1

2

id	name	lecturer	year	semester
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
20	Calculus	Tamar Ezra	2021	1
40	Advanced Programming	David Gol	2022	2
40	Advanced Programming	David Gol	2022	2
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).

A new student has just joined!

- 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

id	age	gender	degree	firstName	lastName	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	20	111	43	0

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

1
1
1
0
1

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

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 veterinarian DBMS:
  - CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20), sex CHAR(1), birth DATE);

column name

column type

Table name

CHAR(X) – Will always consume X bytes.
VARCHAR(x) – Will consume as many bytes as the input (+1) upto x
TEXT – up-to 65K chars
LONGTEXT – over 4GB

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

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.

Passenger on flights

What is/are the Unique key/keys ? 

What is/are the primary key/keys ? 

idNumber	passportNumber	firstName	lastName	flightNmuber
1324561	a4651625	david	cohen	Ly214
5467815	65sf44515	yosi	levi	Ly254
5615124	65f5Ef165	jhon	smith	AL456
NULL	45157552	adi	bar	AA451

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

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

courses

id	name	lecturer	year	semster
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

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 myIndex (ownerId));

Note: NOT NULL

INDEX (or KEY) must be defined after a comma

Query 1 | grades - Table | grades | example - Table

Table Name: example

Schema: course_tables

Collation: Schema Def

Engine: InnoDB

Comments:

Not Null

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	☐	☑	☑	☐	☐	☐	☐	☐	
passport_num	INT	☐	☐	☑	☐	☐	☐	☐	☐	
number	INT	☑	☑	☐	☐	☐	☐	☑	☐	
		☐	☐	☐	☐	☐	☐	☐	☐	

PRIMARY KEY

UNIQUE KEY

Auto Increment

Column Name: number

Data Type: INT

Collation: Table Default

Default:

Comments:

Storage: Virtual Stored

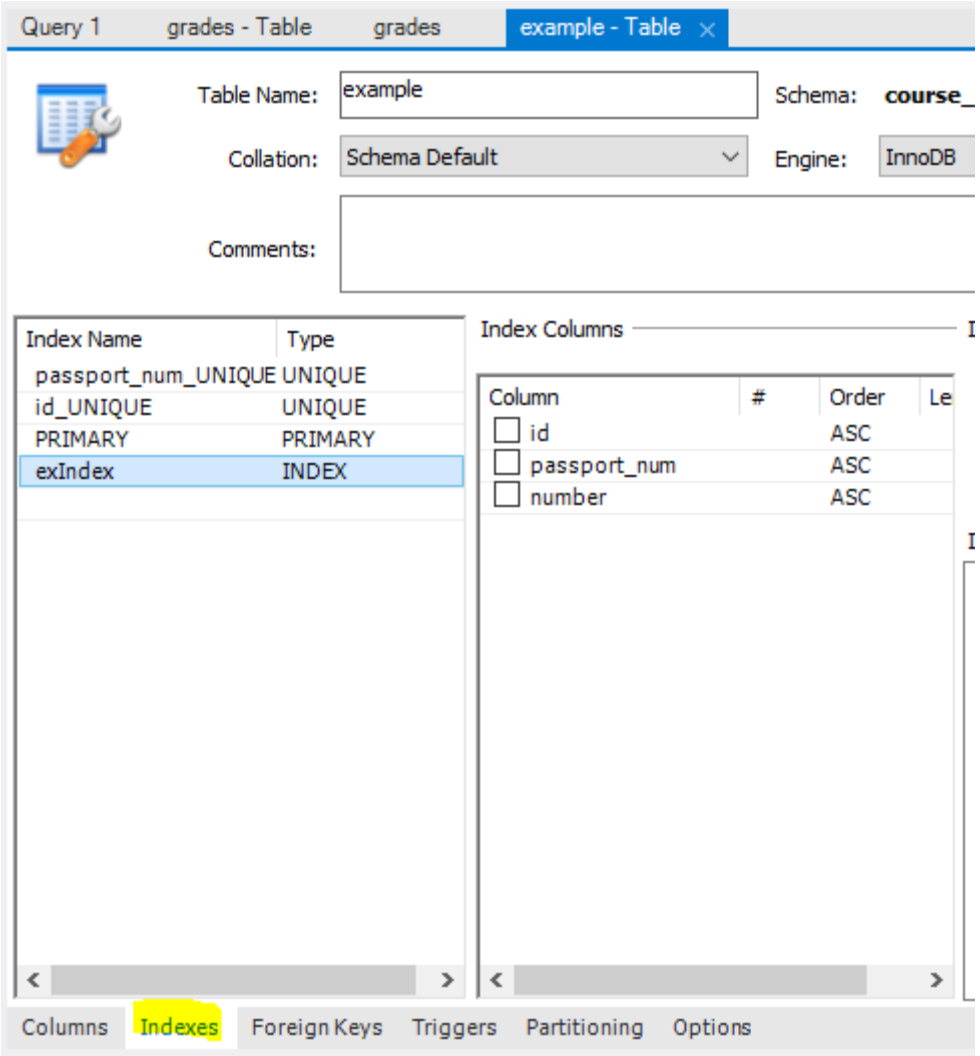
☑ Primary Key ☐ Not Null ☐ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☑ Auto Increment ☐ Generated

Columns | Indexes | Foreign Keys | Triggers | Partitioning | Options

Apply | Revert


Query 1
grades - Table
grades
example - Table
Table Name:
example
Schema:
course_
Collation:
Schema Default
Engine:
InnoDB
Comments:
Index Name
Type
passport_num_UNIQUE UNIQUE
id_UNIQUE UNIQUE
PRIMARY PRIMARY
exIndex INDEX
Index Columns
Column
#
Order
id ASC
passport_num ASC
number ASC
Columns
Indexes
Foreign Keys
Triggers
Partitioning
Options

Integrity Constraints

CHECK keyword:

- CHECK (country IN ('USA’, 'UK’, 'Israel’, 'India’))
- CREATE TABLE people (ID int PRIMARY KEY, lastName varchar(100) NOT NULL, firstName varchar(100), Age int, CHECK (Age>=18));
  - INSERT INTO people (id, lastName, firstName, age) VALUES (4324, 'Bow', 'Gil', 15);

6	10:33:09	INSERT INTO people (id, lastName, firstName, age) VALUES (4324, 'Bow', 'Gil', 15)	Error Code: 3819. Check constraint 'people_chk_1' is violated.

- CREATE TABLE WorkingDay (work_day DATE, income REAL, expenses REAL, revenue REAL, CHECK(revenue=income-expenses));
  - INSERT INTO WorkingDay (work_day, income, expenses, revenue) VALUES (SYSDATE(), 200, 100, 50);

10	10:36:15	INSERT INTO WorkingDay (work_day, income, expenses, revenue) VALUES (...	Error Code: 3819. Check constraint 'workingday_chk_1' is violated.

Foreign Key

- A foreign key is used to link two relations
- A foreign key is one field (or more) in one table that is a Primary key in another table.
- E.g. The field ‘studentId’ in “grades” table is a foreign key for the field ‘id’ in “students” table

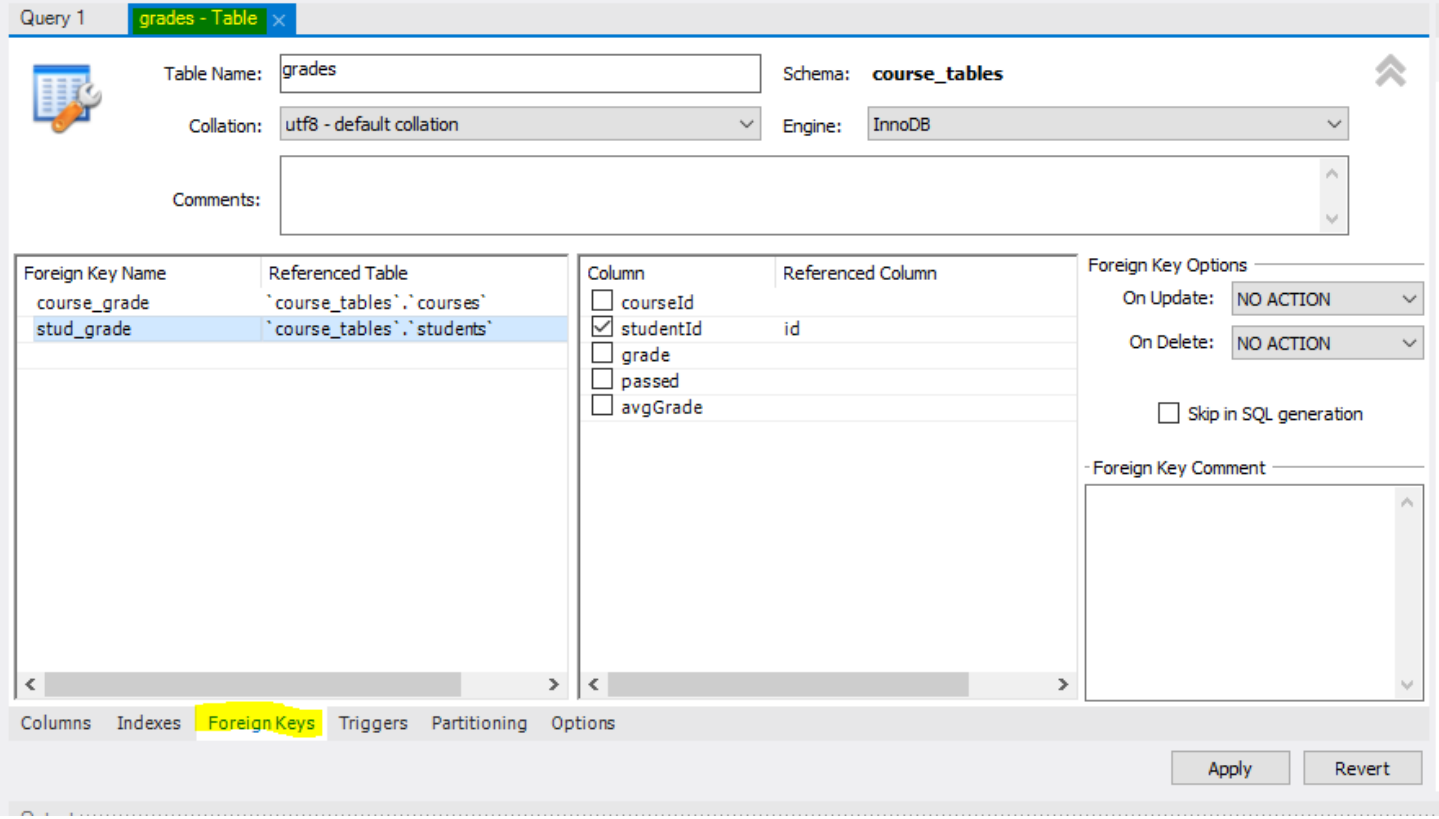
students

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

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

Foreign Key


Query 1
grades - Table
Table Name:
grades
Schema:
course_tables
Collation:
utf8 - default collation
Engine:
InnoDB
Comments:
Foreign Key Name
Referenced Table
course_grade
`course_tables`.`courses`
stud_grade
`course_tables`.`students`
Column
Referenced Column
courseId
studentId
id
grade
passed
avgGrade
Foreign Key Options
On Update:
NO ACTION
On Delete:
NO ACTION
Skip in SQL generation
Foreign Key Comment
Columns
Indexes
Foreign Keys
Triggers
Partitioning
Options
Apply
Revert

Foreign Key

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

- INSERT INTO grades (courseId, studentId, grade, passed) VALUES ('40', '123', '99', '1');

```
Operation failed: There was an error while applying the SQL script to the database.
ERROR 1452: 1452: Cannot add or update a child row: a foreign key constraint fails
(`course_tables`.`grades`, CONSTRAINT `stud_grade` FOREIGN KEY (`studentId`) REFERENCES
`students` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION)
```

- INSERT INTO grades (courseId, studentId, grade, passed) VALUES ('40’, ‘444', '99', '1’);

SQL script was successfully applied to the database.

DROP TABLE

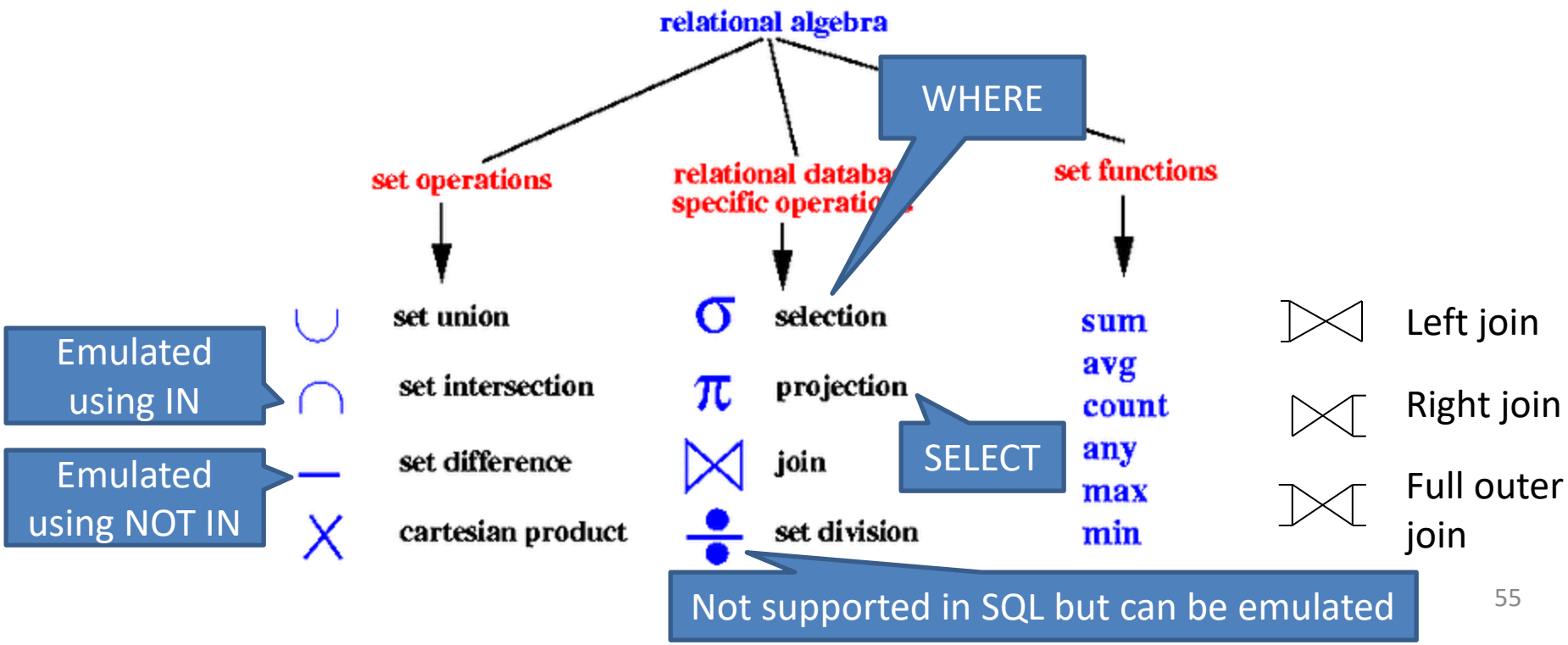
- DROP TABLE deletes a table:
  - DROP TABLE pet2
  - [DELETE TABLE pet2 – returns an error, since DELETE is used to remove entries (rows), with the following syntax: DELETE … FROM …]

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;

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