

# Session 4 Database

Mohamed Emary

2025-01-13

## 1 Session Code

```
1  -- Insert
2  SELECT
3      first_name,
4      last_name,
5      salary
6  FROM
7      employees
8  WHERE
9      salary > 80000;
10
11
12  SELECT
13      *
14  FROM
15      employees
16  WHERE
17      first_name = "Mohamed";
18
19
20  -- Main Query and sub query
21  SELECT
22      first_name,
23      emp_no,
24      bonus
25  FROM
26      employees
27  WHERE
28      bonus > (
29          SELECT
30              AVG(bonus)
31          FROM
32              employees
33      );
34
35
36  SELECT
```

```
37     first_name,
38     last_name,
39     salary,
40     bonus
41 FROM
42     employees
43 WHERE
44     salary > (
45         -- Can't do that since the sub query returns more than one result so
46         -- the > operator don't know which value to compare with
47         -- It will return 3 employees with the name Ahmed
48         SELECT
49             salary
50         FROM
51             employees
52         WHERE
53             First_name = 'Ahmed'
54     );
55
56 -- To solve the problem above we can use `any` which will return any
57 -- employee with salary greater than the salary of any one with the name
58 -- Ahmed
59 -- It will even return the other two employees with the name Ahmed
60 -- employees with salary lower than lowest one with the name ahmed
61 SELECT
62     first_name,
63     last_name,
64     salary,
65     bonus
66 FROM
67     employees
68 WHERE
69     salary > ANY (
70         SELECT
71             salary
72         FROM
73             employees
74         WHERE
75             First_name = 'Ahmed'
76     );
77
78 -- To ignore the two employees with the name Ahmed:
79 SELECT
80     first_name,
81     last_name,
82     salary,
83     bonus
84 FROM
```

```
84     employees
85 WHERE
86     salary > ANY (
87         SELECT
88             salary
89         FROM
90             employees
91         WHERE
92             First_name = 'Ahmed'
93     )
94 AND first_name <> 'Ahmed';
95
96
97 -- To get employees with salary more than all employees with the name
98 --   'Ahmed'
99 SELECT
100     first_name,
101     last_name,
102     salary,
103     bonus
104 FROM
105     employees
106 WHERE
107     salary > ALL (
108         SELECT
109             salary
110         FROM
111             employees
112         WHERE
113             First_name = 'Ahmed'
114     );
115
116 -- To get employees with salary equal to ahmed we can also use `IN`
117 -- We can also use `= ANY` instead of `IN`
118 SELECT
119     first_name,
120     last_name,
121     salary,
122     bonus
123 FROM
124     employees
125 WHERE
126     salary IN (
127         SELECT
128             salary
129         FROM
130             employees
131         WHERE
132             First_name = 'Ahmed'
```

```
133     );
134
135
136 -- To get departments with no employees
137 -- This will return zero rows even though there are departments with no
    ↳ employees
138 -- This is because there is NULL values in the dept_no column in the
    ↳ employees table
139 -- so to solve this we must use `WHERE dept_no IS NOT NULL` in the
    ↳ subquery
140 SELECT
141     *
142 FROM
143     departments
144 WHERE
145     dept_no NOT IN (
146         SELECT DISTINCT
147             dept_no
148         FROM
149             employees
150         WHERE
151             -- We have to add this line below
152             dept_no IS NOT NULL
153     );
154
155
156 -- To get the 2 employees with the highest salary
157 SELECT
158     first_name,
159     last_name,
160     salary
161 FROM
162     employees
163 WHERE
164     salary IS NOT NULL
165 ORDER BY
166     salary DESC limit 2;
167
168
169 -- To get the employees with highest two different salary
170 SELECT
171     -- first_name,
172     -- last_name,
173     DISTINCT salary
174 FROM
175     employees
176 WHERE
177     salary IS NOT NULL
178 ORDER BY
179     salary DESC limit 2;
```

```
180
181
182 -- To get employees who take the highest two different salaries
183 SELECT
184     first_name,
185     last_name,
186     salary
187 FROM
188     employees
189 WHERE
190     salary IN (
191         SELECT DISTINCT
192             salary
193         FROM
194             employees
195         WHERE
196             salary IS NOT NULL
197         ORDER BY
198             salary DESC limit 2
199     );
200
201
202 -- to delete a specific record we use `DELETE FROM` and `WHERE`
203 --
204 -- Before delete the statement below returns 32
205 SELECT
206     COUNT(*)
207 FROM
208     students_course;
209
210
211 -- The delete statement below deletes the record with student_no = 6 and
212   ↳ course_no = 6
213 DELETE FROM students_course
214 WHERE
215     student_no = 6
216     AND course_no = 6;
217
218 -- After delete the statement below returns 32
219 SELECT
220     COUNT(*)
221 FROM
222     students_course;
223
224
225 /*
226 Truncate Vs Delete
227 Delete can delete specific records based on a condition `WHERE`
228
```

## 1 Session Code

---

```
229 Truncate deletes all records in a table so you can't use `WHERE` with
    ↳ `TRUNCATE`
230
231 TRUNCATE is way faster than DELETE because it doesn't go through all the
    ↳ records to delete them
232
233 Truncate is DDL while Delete is DML
234 */
235 -- get the first 5 employees ordered by emp_no
236 SELECT
237     *
238 FROM
239     employees
240 ORDER BY
241     emp_no limit 5;
242
243
244 -- Update employee with emp_no 10003
245 UPDATE employees
246 SET
247     salary = 85008,
248     bonus = 7000,
249     dept_no = 2,
250     title = 'Engineer',
251     birth_date = '8-8-1975',
252     last_name = 'Adel'
253 WHERE
254     emp_no = 10003;
255
256
257 -- =====
258 SELECT
259     *
260 FROM
261     employees
262 WHERE
263     dept_no = 3;
264
265
266 -- To increase the salary of employees in department 3 by 20%
267 UPDATE employees
268 SET
269     salary = salary * 1.2
270 WHERE
271     dept_no = 3;
272
273
274 -- Return the employees in department 3 to their original salary
275 UPDATE employees
276 SET
```

```
277     salary = salary / 1.2
278 WHERE
279     dept_no = 3;
280
281
282 -- To increase the salary of employees in finance department by 20%
283 -- Since employees department doesn't have a dept_no column, we have to
284   ↳ use a subquery to get the dept_no of the finance department
285 UPDATE employees
286 SET
287     salary = salary * 1.2
288 WHERE
289     dept_no = (
290         SELECT
291             dept_no
292         FROM
293             departments
294         WHERE
295             dept_name = 'Finance'
296     );
297
298 SELECT
299     *
300 FROM
301     departments;
302
303
304 INSERT INTO
305     departments
306 VALUES
307     (12, 'Follow Up');
308
309
310 DELETE FROM departments
311 WHERE
312     dept_no = 12;
313
314
315 -- The statement below will not work because the number of columns in the
316   ↳ table is not equal to the number of columns in VALUES of the insert
317   ↳ statement
318 --
319 -- When inserting data into a table:
320 -- 1. the number of columns in the table must be equal to the number of
321   ↳ columns in the VALUES clause
322 -- 2. the ordered of inserted values in VALUES must match the order of
323   ↳ columns in the table
324 --
```

```
321 -- In Postgers sql strings must be enclosed in single quotes not double
    ↳ quotes
322 INSERT INTO
323     employees
324 VALUES
325     (
326         30,
327         '5-5-2000',
328         'Sayed',
329         'Mahmoud',
330         'M',
331         '5-5-2022',
332         60000,
333         4000,
334         NULL,
335         NULL,
336         NULL
337     );
338
339
340 -- if you want to insert only a subset of columns you must specify the
    ↳ columns you want to insert into
341 -- But we can only ignore columns that allow NULLs
342 INSERT INTO
343     employees (
344         emp_no,
345         birth_date,
346         first_name,
347         last_name,
348         gender,
349         hire_date,
350         salary,
351         bonus
352     )
353 VALUES
354     (
355         31,
356         '5-5-2000',
357         'Sayed',
358         'Mahmoud',
359         'M',
360         '5-5-2022',
361         60000,
362         4000
363     );
364
365
366 -- To save the resulting table of a select statement into a table, we have
    ↳ to use the INTO keyword
367 --
```



```
368 -- insert values vs select into
369 -- insert values is used to insert static values into a table
370 -- select into is used to insert the result of a select statement into a
    ↳ table
371 SELECT
372     emp_no,
373     first_name,
374     last_name,
375     salary INTO emp1
376 FROM
377     employees
378 WHERE
379     salary IS NOT NULL
380 ORDER BY
381     salary DESC LIMIT 10;
382
383
384 SELECT
385     *
386 FROM
387     emp1;
388
389
390 -- to create a new database use the CREATE DATABASE statement
391 CREATE
392 DATABASE task1;
393
394
395 -- To remove the database again use DROP DATABASE
396 DROP
397 DATABASE task1;
398
399
400 -- To create a table use the CREATE TABLE statement
401 -- To modify a table structure use the ALTER TABLE statement
402 -- To remove a table use the DROP TABLE statement
403 -- DROP removes the whole table from the database while DELETE removes the
    ↳ rows from the table
404 --
405 -- ALTER allows you to:
406 -- Add Columns
407 -- Delete Columns
408 -- Modify Columns
409 -- Add Constraints
410 -- Delete Constraints
411 DROP TABLE std;
412
413
414 CREATE TABLE
415     std (student_no INT, student_name VARCHAR(30));
```

```
416
417
418 -- To add two new columns to the table we use the ALTER TABLE statement
419 ALTER TABLE std
420 ADD student_address VARCHAR(50),
421 ADD student_bd DATE;
422
423
424 ALTER TABLE std
425 ADD test INT NOT NULL;
426
427
428 -- To drop one of the existing columns
429 ALTER TABLE std
430 DROP COLUMN student_bd;
431
432
433 -- To make changes on existing column use `ALTER TABLE table_name ALTER
434   ↳ COLUMN column_name`
435 ALTER TABLE std
436 ALTER COLUMN student_name
437 SET
438   NOT NULL;
439
440 SELECT
441   *
442 FROM
443   std;
444
445
446 -- Constraints types:
447 -- NOT NULL
448 -- UNIQUE
449 -- PRIMARY KEY
450 -- FOREIGN KEY
451 --
452 -- You can add Constraints in either CREATE TABLE or ALTER TABLE
453 -- But since we have already created the table, we can only use ALTER
454   ↳ TABLE
455 ALTER TABLE std
456 ADD CONSTRAINT s_pk PRIMARY KEY (student_no);
457
458 -- It's better to give your constraint a name but you still can ignore the
459   ↳ name of the constraint and let the system give it a name
460 -- But if you do that and tried to make modifications to the constraint
461   ↳ later, you will need to know the name the system gave to the
462   ↳ constraint
463 ALTER TABLE std DROP CONSTRAINT s_pk;
```

```
461
462
463 -- Since you can only have one primary key in a table, the statement below
464   ↳ will give an error
465 -- multiple primary keys for table "courses" are not allowed
466 CREATE TABLE
467   courses (
468     course_no INT PRIMARY KEY,
469     course_name VARCHAR(30) PRIMARY KEY,
470     course_duration INT
471   );
472
473 -- It should be like this
474 CREATE TABLE
475   courses (
476     course_no INT PRIMARY KEY,
477     course_name VARCHAR(30),
478     course_duration INT
479   );
480
481
482 -- And if you want a value to be similar to PK (UNIQUE and NOT NULL) you
483   ↳ can use UNIQUE constraint
484 -- Since we already created the table we will use ALTER
485 ALTER TABLE courses
486 ADD CONSTRAINT c_un UNIQUE (course_name);
487
488 -- we can apply a constraint on multiple columns at once
489 -- The statement below will accept similar values in course number if the
490   ↳ course name is different
491 -- OR similar values in course name if the course number is different
492 -- But it will not accept similar values in both course number and course
493   ↳ name
494 ALTER TABLE courses
495 ADD CONSTRAINT c_pk UNIQUE (course_no, course_name);
496
497 -- we can also force values of a column to be within a range
498 -- The statement below will cause course_duration to be between 26 and 80.
499   ↳ Any other value will not be accepted
500 ALTER TABLE courses
501 ADD CONSTRAINT c_chk CHECK (course_duration BETWEEN 26 AND 80);
502
503 -- you can also apply check on multiple columns and use `AND` or `OR` to
504   ↳ combine the conditions
505 -- The statement below will cause course_duration to be between 26 and 80
506   ↳ OR course_no to be between 100 and 200
```

```
504 ALTER TABLE coursess
505 ADD CONSTRAINT c_chk CHECK (
506     course_duration BETWEEN 26 AND 80
507     OR course_no BETWEEN 100 AND 200
508 );
509
510
511 -- We can also set a default value to a table column
512 ALTER TABLE coursess
513 ALTER COLUMN course_duration
514 SET
515     DEFAULT 30;
516
517
518 -- To create a primary key from two columns
519 CREATE TABLE
520     teach (
521         student_no INT,
522         course_no INT,
523         grade INT,
524         CONSTRAINT t_pk PRIMARY KEY (student_no, course_no)
525     );
526
527
528 ALTER TABLE teach
529 ADD CONSTRAINT s_fk FOREIGN KEY (student_no) REFERENCES students
530     ↪ (student_no),
531 ADD CONSTRAINT c_fk FOREIGN KEY (course_no) REFERENCES coursess
532     ↪ (course_no);
533
534 -- FOREIGN KEY constraint:
535 -- 1. FOREIGN key makes you not able to delete something in the parent
536     ↪ table that doesn't exist in the child table
537 -- 2. And you can't insert something in the child table that doesn't exist
538     ↪ in the parent table
539
540 --
541 --
542 -- insert, update, delete
543 -- Update
544 -- Delete
545 -- Truncate
```