Session 4 Database

Mohamed Emary

2025-01-13

1 Session Code

```sql
-- Insert
SELECT
  first_name,
  last_name,
  salary
FROM
  employees
WHERE
  salary > 80000;


SELECT
  *
FROM
  employees
WHERE
  first_name = "Mohamed";


-- Main Query and sub query
SELECT
  first_name,
  emp_no,
  bonus
FROM
  employees
WHERE
  bonus > (
    SELECT
      AVG(bonus)
    FROM
      employees
  );


SELECT
```

```sql
  first_name,
  last_name,
  salary,
  bonus
FROM
  employees
WHERE
  salary > (
    -- Can't do that since the sub query returns more than one result so the > operator don't know which value to compare with
    -- It will return 3 employees with the name Ahmed
    SELECT
      salary
    FROM
      employees
    WHERE
      First_name = 'Ahmed'
  );


-- To solve the problem above we can use `any` which will return any employee with salary greater than the salary of any one with the name Ahmed
-- It will even return the other two employees with the name Ahmed
-- employees with salary lower than lowest one with the name ahmed
SELECT
  first_name,
  last_name,
  salary,
  bonus
FROM
  employees
WHERE
  salary > ANY (
    SELECT
      salary
    FROM
      employees
    WHERE
      First_name = 'Ahmed'
  );


-- To ignore the two employees with the name Ahmed:
SELECT
  first_name,
  last_name,
  salary,
  bonus
FROM
```

```sql
  employees
WHERE
  salary > ANY (
    SELECT
      salary
    FROM
      employees
    WHERE
      First_name = 'Ahmed'
  )
  AND first_name <> 'Ahmed';


-- To get employees with salary more than all employees with the name
 ↪  'Ahmed'
SELECT
  first_name,
  last_name,
  salary,
  bonus
FROM
  employees
WHERE
  salary > ALL (
    SELECT
      salary
    FROM
      employees
    WHERE
      First_name = 'Ahmed'
  );


-- To get employees with salary equal to ahmed we can also use `IN`
-- We can also use `= ANY` instead of `IN`
SELECT
  first_name,
  last_name,
  salary,
  bonus
FROM
  employees
WHERE
  salary IN (
    SELECT
      salary
    FROM
      employees
    WHERE
      First_name = 'Ahmed'
```

```sql
  );


-- To get departments with no employees
-- This will return zero rows even though there are departments with no employees
-- This is because there is NULL values in the dept_no column in the employees table
-- so to solve this we must use `WHERE dept_no IS NOT NULL` in the subquery
SELECT
  *
FROM
  departments
WHERE
  dept_no NOT IN (
    SELECT DISTINCT
      dept_no
    FROM
      employees
    WHERE
      -- We have to add this line below
      dept_no IS NOT NULL
  );


-- To get the 2 employees with the highest salary
SELECT
  first_name,
  last_name,
  salary
FROM
  employees
WHERE
  salary IS NOT NULL
ORDER BY
  salary DESC limit 2;


-- To get the employees with highest two different salary
SELECT
  -- first_name,
  -- last_name,
  DISTINCT salary
FROM
  employees
WHERE
  salary IS NOT NULL
ORDER BY
  salary DESC limit 2;
```

```sql


-- To get employees who take the highest two different salaries
SELECT
  first_name,
  last_name,
  salary
FROM
  employees
WHERE
  salary IN (
    SELECT DISTINCT
      salary
    FROM
      employees
    WHERE
      salary IS NOT NULL
    ORDER BY
      salary DESC limit 2
  );


-- to delete a specific record we use `DELETE FROM` and `WHERE`
--
-- Before delete the statement below returns 32
SELECT
  COUNT(*)
FROM
  students_course;


-- The delete statement below deletes the record with student_no = 6 and
↪  course_no = 6
DELETE FROM students_course
WHERE
  student_no = 6
  AND course_no = 6;


-- After delete the statement below returns 32
SELECT
  COUNT(*)
FROM
  students_course;


/*
Truncate Vs Delete
Delete can delete specific records based on a condition `WHERE`

```

```
Truncate deletes all records in a table so you can't use `WHERE` with
↪  `TRUNCATE`

TRUNCATE is way faster than DELETE because it doesn't go through all the
↪  records to delete them

Truncate is DDL while Delete is DML
 */
-- get the first 5 employees ordered by emp_no
SELECT
  *
FROM
  employees
ORDER BY
  emp_no limit 5;


-- Update employee with emp_no 10003
UPDATE employees
SET
  salary = 85008,
  bonus = 7000,
  dept_no = 2,
  title = 'Engineer',
  birth_date = '8-8-1975',
  last_name = 'Adel'
WHERE
  emp_no = 10003;


-- ======================
SELECT
  *
FROM
  employees
WHERE
  dept_no = 3;


-- To increase the salary of employees in department 3 by 20%
UPDATE employees
SET
  salary = salary * 1.2
WHERE
  dept_no = 3;


-- Return the employees in department 3 to their original salary
UPDATE employees
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
    ↪  use a subquery to get the dept_no of the finance department
284 UPDATE employees
285 SET
286   salary = salary * 1.2
287 WHERE
288   dept_no = (
289     SELECT
290       dept_no
291     FROM
292       departments
293     WHERE
294       dept_name = 'Finance'
295   );
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
    ↪  table is not equal to the number of columns in VALUES of the insert
    ↪  statement
316 --
317 -- When inserting data into a table:
318 -- 1. the number of columns in the table must be equal to the number of
    ↪  columns in the VALUES clause
319 -- 2. the ordered of inserted values in VALUES must match the order of
    ↪  columns in the table
320 --
```

```sql
-- In Postgers sql strings must be enclosed in single quotes not double quotes
INSERT INTO
  employees
VALUES
  (
    30,
    '5-5-2000',
    'Sayed',
    'Mahmoud',
    'M',
    '5-5-2022',
    60000,
    4000,
    NULL,
    NULL,
    NULL
  );


-- if you want to insert only a subset of columns you must specify the columns you want to insert into
-- But we can only ignore columns that allow NULLs
INSERT INTO
  employees (
    emp_no,
    birth_date,
    first_name,
    last_name,
    gender,
    hire_date,
    salary,
    bonus
  )
VALUES
  (
    31,
    '5-5-2000',
    'Sayed',
    'Mahmoud',
    'M',
    '5-5-2022',
    60000,
    4000
  );


-- To save the resulting table of a select statement into a table, we have to use the INTO keyword
--
```

```sql
-- insert values vs select into
-- insert values is used to insert static values into a table
-- select into is used to insert the result of a select statement into a table
SELECT
  emp_no,
  first_name,
  last_name,
  salary INTO emp1
FROM
  employees
WHERE
  salary IS NOT NULL
ORDER BY
  salary DESC LIMIT 10;


SELECT
  *
FROM
  emp1;


-- to create a new database use the CREATE DATABASE statement
CREATE
DATABASE task1;


-- To remove the database again use DROP DATABASE
DROP
DATABASE task1;


-- To create a table use the CREATE TABLE statement
-- To modify a table structure use the ALTER TABLE statement
-- To remove a table use the DROP TABLE statement
-- DROP removes the whole table from the database while DELETE removes the rows from the table
--
-- ALTER allows you to:
-- Add Columns
-- Delete Columns
-- Modify Columns
-- Add Constraints
-- Delete Constraints
DROP TABLE std;


CREATE TABLE
  std (student_no INT, student_name VARCHAR(30));
```

```sql


-- To add two new columns to the table we use the ALTER TABLE statement
ALTER TABLE std
ADD student_address VARCHAR(50),
ADD student_bd DATE;


ALTER TABLE std
ADD test INT NOT NULL;


-- To drop one of the existing columns
ALTER TABLE std
DROP COLUMN student_bd;


-- To make changes on existing column use `ALTER TABLE table_name ALTER
↪  COLUMN column_name`
ALTER TABLE std
ALTER COLUMN student_name
SET
  NOT NULL;


SELECT
  *
FROM
  std;


-- Constraints types:
-- NOT NULL
-- UNIQUE
-- PRIMARY KEY
-- FOREIGN KEY
--
-- You can add Constraints in either CREATE TABLE or ALTER TABLE
-- But since we have already created the table, we can only use ALTER
↪  TABLE
ALTER TABLE std
ADD CONSTRAINT s_pk PRIMARY KEY (student_no);


-- It's better to give your constraint a name but you still can ignore the
↪  name of the constraint and let the system give it a name
-- But if you do that and tried to make modifications to the constraint
↪  later, you will need to know the name the system gave to the
↪  constraint
ALTER TABLE std DROP CONSTRAINT s_pk;
```

```sql


-- Since you can only have one primary key in a table, the statement below will give an error
-- multiple primary keys for table "coursess" are not allowed
CREATE TABLE
  coursess (
    course_no INT PRIMARY KEY,
    course_name VARCHAR(30) PRIMARY KEY,
    course_duration INT
  );


-- It should be like this
CREATE TABLE
  coursess (
    course_no INT PRIMARY KEY,
    course_name VARCHAR(30),
    course_duration INT
  );


-- And if you want a value to be similar to PK (UNIQUE and NOT NULL) you can use UNIQUE constraint
-- Since we already created the table we will use ALTER
ALTER TABLE coursess
ADD CONSTRAINT c_un UNIQUE (course_name);


-- we can apply a constraint on multiple columns at once
-- The statement below will accept similar values in course number if the course name is different
-- OR similar values in course name if the course number is different
-- But it will not accept similar values in both course number and course name
ALTER TABLE coursess
ADD CONSTRAINT c_pk UNIQUE (course_no, course_name);


-- we can also force values of a column to be within a range
-- The statement below will cause course_duration to be between 26 and 80. Any other value will not be accepted
ALTER TABLE coursess
ADD CONSTRAINT c_chk CHECK (course_duration BETWEEN 26 AND 80);


-- you can also apply check on multiple columns and use `AND` or `OR` to combine the conditions
-- The statement below will cause course_duration to be between 26 and 80 OR course_no to be between 100 and 200
```

```sql
ALTER TABLE coursess
ADD CONSTRAINT c_chk CHECK (
  course_duration BETWEEN 26 AND 80
  OR course_no BETWEEN 100 AND 200
);


-- We can also set a default value to a table column
ALTER TABLE coursess
ALTER COLUMN course_duration
SET
  DEFAULT 30;


-- To create a primary key from two columns
CREATE TABLE
  teach (
    student_no INT,
    course_no INT,
    grade INT,
    CONSTRAINT t_pk PRIMARY KEY (student_no, course_no)
  );


ALTER TABLE teach
ADD CONSTRAINT s_fk FOREIGN KEY (student_no) REFERENCES students
↪  (student_no),
ADD CONSTRAINT c_fk FOREIGN KEY (course_no) REFERENCES coursess
↪  (course_no);


-- FOREIGN KEY constraint:
-- 1. FOREIGN key makes you not able to delete something in the parent
↪  table that doesn't exist in the child table
-- 2. And you can't insert something in the child table that doesn't exist
↪  in the parent table
--
--
-- insert, update, delete
-- Update
-- Delete
-- Truncate
```