

**Катедра „Компютърни системи “**

**КУРСОВ ПРОЕКТ**

**ПО БАЗИ ОТ ДАННИ**

**Студент: Янко Янков Янков**

**Фак. № 121221167 Група: 43**

|  |
| --- |
| **Тема No13** |
| Да се разработи база данни за “bug tracker” система. Тази система пази информация за бъговете в даден IT екип или отдел. Всеки записан бъг има уникален номер, по който ще се проследява; описание на проблема (стъпки за репродуциране, операционна система, логове);проекта, в който е открит; час и дата, в които е логнат; приоритет; статус; QA, който е логнал проблема; developer, който трябва да го оправи. Проектите са обекти, които имат номер, име на мениджъра, който ръководи проекта, програмист, който е owner на проекта, програмист(и), които ще го имплементират, QA, който отговаря за тестването, както и кратко описание. Приоритетът на всеки един бъг може да бъде един от следните: minor, medium, major, critical. Статусът може да бъде: new, addressed, failed, closed.Допълнете таблиците с необходима информация по ваш избор. |

1.Да се проектира база от данни и да се представи ER диаграма със съответни CREATE TABLE заявки за средата MySQL.

2.Напишете заявка, в която демонстрирате SELECT с ограничаващо условие по избор.

3.Напишете заявка, в която използвате агрегатна функция и GROUP BY по ваш избор.

4.Напишете заявка, в която демонстрирате INNER JOIN по ваш избор.

5.Напишете заявка, в която демонстрирате OUTER JOIN по ваш избор.

6.Напишете заявка, в която демонстрирате вложен SELECT по ваш избор.

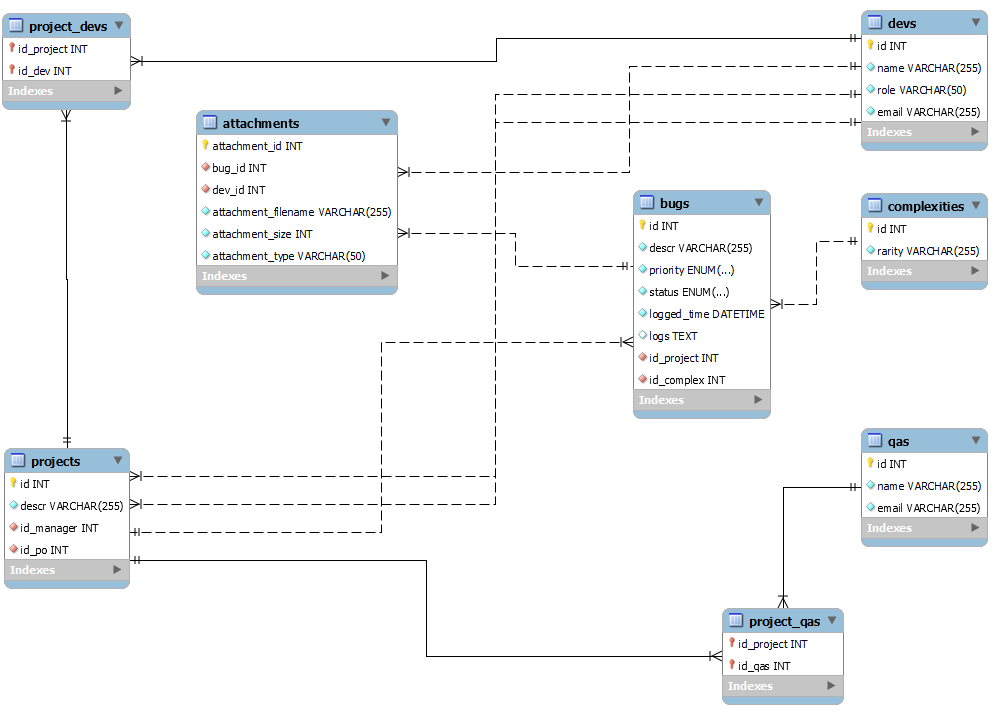
7.Напишете заявка, в която демонстрирате едновременно JOIN и агрегатна функция.

8.Създайте тригер по ваш избор.

9.Създайте процедура, в която демонстрирате използване на курсор.

Вашата работа трябва да включва: задание, ER-диаграма, CREATE TABLE заявки, всички останали заявки, решения на задачите от 2 до 9 и резултатите от тях.

1. **Да се проектира база от данни и да се представи ER диаграма със съответни CREATE TABLE заявки за средата MySQL**

Основните обекти, за които трябва да съхраняваме информация, според заданието са: „Bugs“, “QA”, “Projects” и“Devs”. Допълнително ще създадем още няколко таблици. Първата ще отразява сложността на грешката като те могат да бъдат: „Low“, “Medium”, “High” или “Critical”. Във втората допълнителна таблица „Attachments“, ще се отразяват качетните решения на различните „бъгове“. Като тя ще запазват данните за кой е качил определеният файл, за кой „bug“ се отнася, името на файла, неговата големина и тип. За проектиране на базата ще използваме модела ER-диаграма (Entity Relationship DIagram):

Заявките, с които създаваме базата данни и таблиците са:

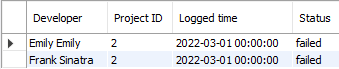
|  |
| --- |
| CREATE SCHEMA IF NOT EXISTS `bugtracker`;  USE `bugtracker`;  CREATE TABLE IF NOT EXISTS `qas` (  `id` INT AUTO\_INCREMENT PRIMARY KEY,  `name` VARCHAR(255) NOT NULL,  `email` VARCHAR(255) NOT NULL  );  CREATE TABLE IF NOT EXISTS `complexities` (  `id` INT AUTO\_INCREMENT PRIMARY KEY,  `rarity` VARCHAR(255) NOT NULL  );  CREATE TABLE IF NOT EXISTS `devs` (  `id` INT AUTO\_INCREMENT PRIMARY KEY,  `name` VARCHAR(255) NOT NULL,  `role` VARCHAR(50) NOT NULL,  `email` VARCHAR(255) NOT NULL  );  CREATE TABLE IF NOT EXISTS `projects` (  `id` INT AUTO\_INCREMENT PRIMARY KEY,  `descr` VARCHAR(255) NOT NULL,  `id\_manager` INT NOT NULL,  `id\_po` INT NOT NULL,  CONSTRAINT FOREIGN KEY (`id\_manager`)  REFERENCES `devs` (`id`),  CONSTRAINT FOREIGN KEY (`id\_po`)  REFERENCES `devs` (`id`)  );  CREATE TABLE IF NOT EXISTS `project\_devs` (  `id\_project` INT NOT NULL,  `id\_dev` INT NOT NULL,  PRIMARY KEY (`id\_project` , `id\_dev`),  CONSTRAINT FOREIGN KEY (`id\_project`)  REFERENCES `projects` (`id`),  CONSTRAINT FOREIGN KEY (`id\_dev`)  REFERENCES `devs` (`id`)  );  CREATE TABLE IF NOT EXISTS `project\_qas` (  `id\_project` INT NOT NULL,  `id\_qas` INT NOT NULL,  PRIMARY KEY (`id\_project` , `id\_qas`),  CONSTRAINT FOREIGN KEY (`id\_project`)  REFERENCES `projects` (`id`),  CONSTRAINT FOREIGN KEY (`id\_qas`)  REFERENCES `qas` (`id`)  );  CREATE TABLE IF NOT EXISTS `bugs` (  `id` INT AUTO\_INCREMENT PRIMARY KEY,  `descr` VARCHAR(255) NOT NULL,  `priority` ENUM('minor', 'medium', 'major', 'critical') NOT NULL,  `status` ENUM('new', 'addressed', 'failed', 'closed') NOT NULL,  `logged\_time` DATETIME NOT NULL,  logs TEXT,  `id\_project` INT NOT NULL,  `id\_complex` INT NOT NULL,  CONSTRAINT FOREIGN KEY (`id\_project`)  REFERENCES `projects` (`id`),  CONSTRAINT FOREIGN KEY (`id\_complex`)  REFERENCES `complexities` (`id`)  );  CREATE TABLE `attachments` (  `attachment\_id` INT PRIMARY KEY,  `bug\_id` INT NOT NULL,  `dev\_id` INT NOT NULL,  `attachment\_filename` VARCHAR(255) NOT NULL,  `attachment\_size` INT NOT NULL,  `attachment\_type` VARCHAR(50) NOT NULL,  FOREIGN KEY (`bug\_id`)  REFERENCES bugs (`id`),  FOREIGN KEY (`dev\_id`)  REFERENCES devs (`id`)  ); |

Добавяме и тестови данни в таблиците:

|  |
| --- |
| INSERT INTO qas (name, email) VALUES  ('Ivan Ivanov', 'ivanov@example.com'),  ('Yanko Yankov', 'yankov@example.com'),  ('Ivan Petkov', 'petkov@example.com');  INSERT INTO complexities (rarity) VALUES  ('Low'),  ('Medium'),  ('High'),  ('Critical');  INSERT INTO devs (name, role, email) VALUES  ('Georgi Georgiev', 'Manager', 'georgiev@example.com'),  ('Hristo Petrov', 'PO', 'petrov@example.com'),  ('Emily Emily', 'Developer', 'emily@example.com'),  ('Frank Sinatra', 'Developer', 'frank@example.com');  INSERT INTO projects (descr, id\_manager, id\_po) VALUES  ('Project A', 1, 2),  ('Project B', 1, 2);  INSERT INTO project\_devs (id\_project, id\_dev) VALUES  (1, 3),  (1, 4),  (2, 3),  (2, 4);  INSERT INTO project\_qas (id\_project, id\_qas) VALUES  (1, 1),  (2, 2);  INSERT INTO bugs (descr, priority, status, logged\_time, logs, id\_project, id\_complex) VALUES  ('Bug 1 in Project A', 'minor', 'new', '2022-01-01 00:00:00', 'Log of bug 1', 1, 1),  ('Bug 2 in Project A', 'major', 'addressed', '2022-02-01 00:00:00', 'Log of bug 2', 1, 2),  ('Bug 3 in Project B', 'critical', 'failed', '2022-03-01 00:00:00', 'Log of bug 3', 2, 3),  ('Bug 4 in Project B', 'medium', 'closed', '2022-04-01 00:00:00', 'Log of bug 4', 2, 4);  INSERT INTO attachments (attachment\_id, bug\_id, dev\_id, attachment\_filename, attachment\_size, attachment\_type) VALUES  (1, 1, 3, 'attachment1.txt', 100, 'text/plain'),  (2, 1, 4, 'attachment2.jpg', 200, 'image/jpeg'),  (3, 2, 3, 'attachment3.png', 300, 'image/png'); |

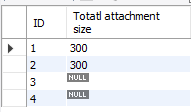
**2. Напишете заявка, в която демонстрирате SELECT с ограничаващо условие по избор =** ще изведем информация за всички „бъгове”, чиято сложност е „High“.

|  |
| --- |
| SELECT  `devs`.`name` AS `Developer`,  `projects`.`id` AS `Project ID`,  `bugs`.`logged\_time` AS `Logged time`,  `bugs`.`status` AS `Status`  FROM  `devs`  JOIN  `project\_devs` ON `devs`.`id` = `project\_devs`.`id\_dev`  JOIN  `projects` ON `project\_devs`.`id\_project` = `projects`.`id`  JOIN  `bugs` ON `projects`.`id` = `bugs`.`id\_project`  JOIN  `complexities` ON `bugs`.`id\_complex` = `complexities`.`id`  WHERE  `complexities`.`rarity` = 'high'; |



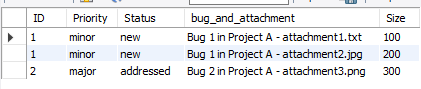
**3.Напишете заявка, в която използвате агрегатна функция и GROUP BY по ваш избор –** тази заявка изчислява общия размер на прикачените файлове за всяко ниво на сложност на грешките. Използваме LEFT JOIN за да ни получим резултати за всички сложности на грешките, дори и за тези, за които нямаме информация.

|  |
| --- |
| SELECT id\_complex as `ID`, SUM(attachment\_size) as `Total attachment size`  FROM bugs  LEFT JOIN attachments ON bugs.id = attachments.bug\_id  GROUP BY id\_complex; |



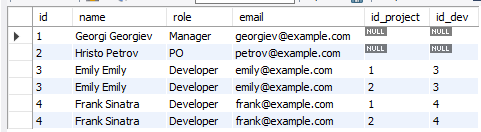
**4.Напишете заявка, в която демонстрирате INNER JOIN по ваш избор –** ще напишем заявка, с която да изведем цялата информация за грешки и техните качени решения.

|  |
| --- |
| SELECT  b.id as `ID`,  b.priority as `Priority`,  b.status as `Status`,  CONCAT(b.descr, ' - ', a.attachment\_filename) AS bug\_and\_attachment,  a.attachment\_size as `Size`  FROM  bugs b  INNER JOIN  attachments a ON b.id = a.bug\_id; |



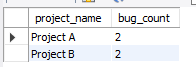
**5. Напишете заявка, в която демонстрирате OUTER JOIN по ваш избор -** тази заявка ще избере всички редове от таблицата „devs“ и ще включи всички съответстващи редове от таблицата „project\_devs". За свързването ще използваме LEFT OUTER JOIN и по този начин = ако има програмист, който все още не работи върху проект, би трябало да присъства в резултатите

|  |
| --- |
| SELECT \*  FROM `devs`  LEFT OUTER JOIN `project\_devs` ON `devs`.`id` = `project\_devs`.`id\_dev` |



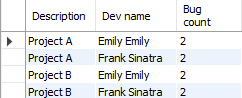
**6. Напишете заявка, в която демонстрирате вложен SELECT по ваш избор -** тази заявка ще върне името на всеки проект заедно с общия брой грешки, свързани с него. Вложеният SELECT го използваме да преброи “bugs” за всеки проект като ги хрупираме като проверяваме дали ‚id\_project‘ съвпада с ‚id‘ на текущия проект от външния SELECT.

|  |
| --- |
| SELECT  p.descr AS project\_name,  (SELECT COUNT(\*) FROM bugs WHERE bugs.id\_project = p.id) AS bug\_count  FROM  projects p; |

****

**7. Напишете заявка, в която демонстрирате едновременно JOIN и агрегатна функция –** тази заявка ще ни изведе броя на грешките, свързани с всеки проект. След това резултатите се групират по “id” и “description”, за да се покажат отделен брой за всеки проект.

|  |
| --- |
| SELECT  p.descr AS 'Description',  d.name AS 'Dev name',  COUNT(b.id) AS 'Bug count'  FROM  projects p  JOIN  project\_devs pd ON pd.id\_project = p.id  JOIN  devs d ON d.id = pd.id\_dev  JOIN  bugs b ON b.id\_project = p.id  GROUP BY p.id , d.id  HAVING COUNT(b.id) > 0  ORDER BY 'Description' , 'Dev name'; |



**8. Създайте тригер по ваш избор –** създаваме тригер, чрез който ще извикваме грешка ако към един програмист са зададени повече от 5 неразрешени грешки. Този тригер ще се задейства, след като нов ред бъде вмъкнат в таблицата „bugs“, и ще преброи броя на неразрешените грешки, присвоени на програмистa на ново вмъкнатия „бъг“.

|  |
| --- |
| DELIMITER //  CREATE TRIGGER `check\_unresolved\_bugs` AFTER INSERT ON `bugs`  FOR EACH ROW  BEGIN  DECLARE num\_unresolved\_bugs INT;  SELECT COUNT(\*) INTO num\_unresolved\_bugs  FROM bugs  WHERE id\_complex = NEW.id\_complex  AND status IN ('new', 'addressed');  IF num\_unresolved\_bugs > 5 THEN  SIGNAL SQLSTATE '45000'  SET MESSAGE\_TEXT = 'Developer cannot be assigned to more than 5 unresolved bugs at a time';  END IF;  END; //  DELIMITER ; |

Тестваме със следната INSERT заявка:

|  |
| --- |
| INSERT INTO bugs (descr, priority, status, logged\_time, id\_project, id\_complex) VALUES  ('UI button is not working', 'major', 'new', '2023-05-05 10:00:00', 1, 2),  ('UI button is not working2', 'major', 'new', '2023-05-05 10:00:00', 1, 2),  ('UI button is not working3', 'major', 'new', '2023-05-05 10:00:00', 1, 2),  ('UI button is not working4', 'major', 'new', '2023-05-05 10:00:00', 1, 2); |

И резултатът е:



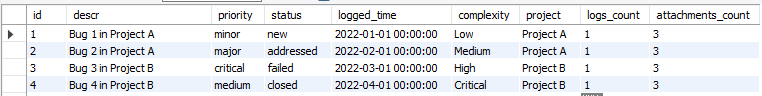
|  |
| --- |
| *Error Code: 1644. Developer cannot be assigned to more than 5 unresolved bugs at a time* |

**9.Създайте процедура, в която демонстрирате използване на курсор –** целта на тази процедура е да предостави обобщен доклад за „bugs“ таблицата заедно с броя на „log“ файлове и прикачените файлове, свързани с всяка грешка.

|  |
| --- |
| DELIMITER //  CREATE PROCEDURE `bug\_summary`()  BEGIN  DECLARE done INT DEFAULT false;  DECLARE bug\_id INT;  DECLARE bug\_descr VARCHAR(255);  DECLARE bug\_priority ENUM('minor', 'medium', 'major', 'critical');  DECLARE bug\_status ENUM('new', 'addressed', 'failed', 'closed');  DECLARE bug\_time DATETIME;  DECLARE bug\_complexity VARCHAR(255);  DECLARE bug\_project VARCHAR(255);  DECLARE bug\_logs TEXT;  DECLARE bug\_logs\_count INT DEFAULT 0;  DECLARE bug\_attachments\_count INT DEFAULT 0;    DECLARE cur CURSOR FOR SELECT b.id, b.descr, b.priority, b.status, b.logged\_time, c.rarity, p.descr, b.logs  FROM bugs b  JOIN complexities c ON b.id\_complex = c.id  JOIN projects p ON b.id\_project = p.id;    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;    DROP TEMPORARY TABLE IF EXISTS bug\_summary;    CREATE TEMPORARY TABLE bug\_summary (  id INT PRIMARY KEY,  descr VARCHAR(255),  priority ENUM('minor', 'medium', 'major', 'critical'),  status ENUM('new', 'addressed', 'failed', 'closed'),  logged\_time DATETIME,  complexity VARCHAR(255),  project VARCHAR(255),  logs\_count INT,  attachments\_count INT  );    OPEN cur;    read\_loop: LOOP  FETCH cur INTO bug\_id, bug\_descr, bug\_priority, bug\_status, bug\_time, bug\_complexity, bug\_project, bug\_logs;  IF done THEN  LEAVE read\_loop;  END IF;    SET bug\_logs\_count = LENGTH(bug\_logs) - LENGTH(REPLACE(bug\_logs, ',', '')) + 1;    SELECT COUNT(\*) INTO bug\_attachments\_count FROM attachments WHERE bug\_id = bug\_id;    INSERT INTO bug\_summary (id, descr, priority, status, logged\_time, complexity, project, logs\_count, attachments\_count)  VALUES (bug\_id, bug\_descr, bug\_priority, bug\_status, bug\_time, bug\_complexity, bug\_project, bug\_logs\_count, bug\_attachments\_count);  END LOOP;    CLOSE cur;    SELECT \* FROM bug\_summary;    END  //  DELIMITER ;;  SELECT @result AS result;  END  //  DELIMITER ; |

Тестваме процедурата:

|  |
| --- |
| call `bug\_summary`(); |

И резултатът е: