|  |  |  |
| --- | --- | --- |
| **Academic Year:** 2024-25 | **Year:** Second Year | **Semester:** II |
| **PRN No.:** | **Name:** | |
| **Subject:** Database Management System | | |
| **Assignment No.**: 9 |  | |
| **Date:** |  | |

**Lab Assignment: 09**

**Title:** Cursor:

**Cursor for Row-by-Row Processing**:

Use a cursor to process each row of a table and update another table based on the results.

**Write and Execute Cursors to perform following kind of operations:**

**Theory:**

What is Cursor?

A **Cursor** in SQL is a database object used to retrieve, manipulate, and navigate through a result set row-by-row. It is particularly useful when we need to perform operations on each row of the result set individually rather than applying a single operation to the entire set.

Explain Cursor Syntax

1. DECLARE cursor\_name CURSOR FOR

2. SELECT column1, column2 FROM table\_name WHERE condition;

3.

4. OPEN cursor\_name;

5.

6. FETCH NEXT FROM cursor\_name INTO variable1, variable2;

7.

8. MOVE FORWARD 1 FROM cursor\_name;

9. MOVE BACKWARD 1 FROM cursor\_name;

10.

11. CLOSE cursor\_name;

12.

Cursor Life Cycle

1. **Declaration:** The cursor is declared and associated with a specific SQL query.
2. **Opening:** The cursor is opened, and the SQL query is executed. The result set is now available for traversal.
3. **Fetching:** The cursor retrieves one or multiple rows at a time.
4. **Processing:** The fetched data is processed within a loop.
5. **Closing:** Once processing is complete, the cursor is closed to free up resources.

**Execution:**

1. **Create a stored procedure that uses a cursor to print all employee names from the employees table.**
2. **Design a cursor-based stored procedure to find and update products with quantity < 10 to 'Out of Stock' in the products table.**
3. **Write a cursor-based procedure that sends alerts for overdue invoices by updating the invoices table status.**
4. **Create a procedure that uses a cursor to list all products in the products table along with their prices.**
5. **Write a procedure that uses a cursor to read from the employees table and increase the salary by 10% for employees with experience > 5 years.**
6. **Develop a cursor-based procedure to check each student’s marks and insert 'Pass' or 'Fail' into a separate results table.**
7. **Write a procedure using a cursor that monitors the attendance table and marks students as 'Irregular' if attendance < 75%.**

1. dbms=# -- 1. Procedure to Print All Employee Names Using a Cursor

2. dbms=# CREATE OR REPLACE PROCEDURE print\_employee\_names()

3. dbms-# LANGUAGE plpgsql AS $$

4. dbms$# DECLARE

5. dbms$# emp\_name TEXT;

6. dbms$# emp\_cursor CURSOR FOR SELECT name FROM employees;

7. dbms$# BEGIN

8. dbms$# OPEN emp\_cursor;

9. dbms$# LOOP

10. dbms$# FETCH emp\_cursor INTO emp\_name;

11. dbms$# EXIT WHEN NOT FOUND;

12. dbms$# RAISE NOTICE 'Employee Name: %', emp\_name;

13. dbms$# END LOOP;

14. dbms$# CLOSE emp\_cursor;

15. dbms$# END;

16. dbms$# $$;

17. CREATE PROCEDURE

18. dbms=#

19. dbms=# -- Run procedure 1

20. dbms=# CALL print\_employee\_names();

21. NOTICE: Employee Name: Jane Smith

22. NOTICE: Employee Name: Michael Johnson

23. NOTICE: Employee Name: John Doe

24. NOTICE: Employee Name: Emily Davis

25. CALL

26. dbms=#

27. dbms=# -- 2. Procedure to Update Products with Quantity < 10 to 'Out of Stock'

28. dbms=# CREATE OR REPLACE PROCEDURE update\_product\_status()

29. dbms-# LANGUAGE plpgsql AS $$

30. dbms$# DECLARE

31. dbms$# prod\_id INT;

32. dbms$# prod\_cursor CURSOR FOR SELECT product\_id FROM products WHERE quantity < 10;

33. dbms$# BEGIN

34. dbms$# OPEN prod\_cursor;

35. dbms$# LOOP

36. dbms$# FETCH prod\_cursor INTO prod\_id;

37. dbms$# EXIT WHEN NOT FOUND;

38. dbms$# UPDATE products SET status = 'Out of Stock' WHERE product\_id = prod\_id;

39. dbms$# END LOOP;

40. dbms$# CLOSE prod\_cursor;

41. dbms$# END;

42. dbms$# $$;

43. CREATE PROCEDURE

44. dbms=#

45. dbms=# -- Run procedure 2

46. dbms=# CALL update\_product\_status();

47. CALL

48. dbms=# SELECT product\_id, name, quantity, status FROM products WHERE quantity < 10;

49. product\_id | name | quantity | status

50. ------------+---------+----------+--------------

51. 1 | Laptop | 5 | Out of Stock

52. 3 | Mouse | 8 | Out of Stock

53. 4 | Monitor | 3 | Out of Stock

54. (3 rows)

55.

56.

57. dbms=#

58. dbms=# -- 3. Procedure to Send Alerts for Overdue Invoices

59. dbms=# CREATE OR REPLACE PROCEDURE update\_invoice\_status()

60. dbms-# LANGUAGE plpgsql AS $$

61. dbms$# DECLARE

62. dbms$# inv\_id INT;

63. dbms$# inv\_cursor CURSOR FOR SELECT invoice\_id FROM invoices WHERE due\_date < CURRENT\_DATE AND status <> 'Overdue';

64. dbms$# BEGIN

65. dbms$# OPEN inv\_cursor;

66. dbms$# LOOP

67. dbms$# FETCH inv\_cursor INTO inv\_id;

68. dbms$# EXIT WHEN NOT FOUND;

69. dbms$# UPDATE invoices SET status = 'Overdue' WHERE invoice\_id = inv\_id;

70. dbms$# END LOOP;

71. dbms$# CLOSE inv\_cursor;

72. dbms$# END;

73. dbms$# $$;

74. CREATE PROCEDURE

75. dbms=#

76. dbms=# -- Run procedure 3

77. dbms=# CALL update\_invoice\_status();

78. CALL

79. dbms=# SELECT invoice\_id, due\_date, status FROM invoices;

80. invoice\_id | due\_date | status

81. ------------+------------+---------

82. 1 | 2025-05-01 | Overdue

83. 2 | 2025-04-25 | Overdue

84. 3 | 2025-05-08 | Overdue

85. (3 rows)

86.

87.

88. dbms=#

89. dbms=# -- 4. Procedure to List All Products and Their Prices

90. dbms=# CREATE OR REPLACE PROCEDURE list\_products()

91. dbms-# LANGUAGE plpgsql AS $$

92. dbms$# DECLARE

93. dbms$# prod\_name TEXT;

94. dbms$# prod\_price NUMERIC;

95. dbms$# prod\_cursor CURSOR FOR SELECT name, price FROM products;

96. dbms$# BEGIN

97. dbms$# OPEN prod\_cursor;

98. dbms$# LOOP

99. dbms$# FETCH prod\_cursor INTO prod\_name, prod\_price;

100. dbms$# EXIT WHEN NOT FOUND;

101. dbms$# RAISE NOTICE 'Product: %, Price: %', prod\_name, prod\_price;

102. dbms$# END LOOP;

103. dbms$# CLOSE prod\_cursor;

104. dbms$# END;

105. dbms$# $$;

106. CREATE PROCEDURE

107. dbms=#

108. dbms=# -- Run procedure 4

109. dbms=# CALL list\_products();

110. NOTICE: Product: Keyboard, Price: 20.00

111. NOTICE: Product: Laptop, Price: 800.00

112. NOTICE: Product: Mouse, Price: 15.00

113. NOTICE: Product: Monitor, Price: 150.00

114. CALL

115. dbms=#

116. dbms=# -- 5. Procedure to Increase Salary by 10% for Employees with Experience > 5 Years

117. dbms=# CREATE OR REPLACE PROCEDURE increase\_salary()

118. dbms-# LANGUAGE plpgsql AS $$

119. dbms$# DECLARE

120. dbms$# emp\_id INT;

121. dbms$# sal\_cursor CURSOR FOR SELECT employee\_id FROM employees WHERE experience > 5;

122. dbms$# BEGIN

123. dbms$# OPEN sal\_cursor;

124. dbms$# LOOP

125. dbms$# FETCH sal\_cursor INTO emp\_id;

126. dbms$# EXIT WHEN NOT FOUND;

127. dbms$# UPDATE employees SET salary = salary \* 1.1 WHERE employee\_id = emp\_id;

128. dbms$# END LOOP;

129. dbms$# CLOSE sal\_cursor;

130. dbms$# END;

131. dbms$# $$;

132. CREATE PROCEDURE

133. dbms=#

134. dbms=# -- Run procedure 5

135. dbms=# SELECT employee\_id, name, experience, salary FROM employees WHERE experience > 5;

136. employee\_id | name | experience | salary

137. -------------+-------------+------------+----------

138. 1 | John Doe | 6 | 55000.00

139. 3 | Emily Davis | 8 | 66000.00

140. (2 rows)

141.

142.

143. dbms=# CALL increase\_salary();

144. CALL

145. dbms=# SELECT employee\_id, name, experience, salary FROM employees WHERE experience > 5;

146. employee\_id | name | experience | salary

147. -------------+-------------+------------+----------

148. 1 | John Doe | 6 | 60500.00

149. 3 | Emily Davis | 8 | 72600.00

150. (2 rows)

151.

152.

153. dbms=#

154. dbms=# -- 6. Fixed Procedure to Check Marks and Insert 'Pass' or 'Fail' into Results Table

155. dbms=# CREATE OR REPLACE PROCEDURE evaluate\_students()

156. dbms-# LANGUAGE plpgsql AS $$

157. dbms$# DECLARE

158. dbms$# s\_id INT;

159. dbms$# s\_marks INT;

160. dbms$# res\_cursor CURSOR FOR SELECT student\_id, marks FROM students;

161. dbms$# BEGIN

162. dbms$# OPEN res\_cursor;

163. dbms$# LOOP

164. dbms$# FETCH res\_cursor INTO s\_id, s\_marks;

165. dbms$# EXIT WHEN NOT FOUND;

166. dbms$# IF s\_marks >= 50 THEN

167. dbms$# INSERT INTO results (student\_id, status) VALUES (s\_id, 'Pass');

168. dbms$# ELSE

169. dbms$# INSERT INTO results (student\_id, status) VALUES (s\_id, 'Fail');

170. dbms$# END IF;

171. dbms$# END LOOP;

172. dbms$# CLOSE res\_cursor;

173. dbms$# END;

174. dbms$# $$;

175. CREATE PROCEDURE

176. dbms=#

177. dbms=# -- Run procedure 6

178. dbms=# -- First, clear any existing results to avoid duplicate entries

179. dbms=# DELETE FROM results;

180. DELETE 0

181. dbms=# CALL evaluate\_students();

182. CALL

183. dbms=# SELECT s.student\_id, s.marks, r.status

184. dbms-# FROM students s

185. dbms-# JOIN results r ON s.student\_id = r.student\_id;

186. student\_id | marks | status

187. ------------+-------+--------

188. 1 | 65 | Pass

189. 2 | 40 | Fail

190. 3 | 85 | Pass

191. 4 | 30 | Fail

192. 5 | 65 | Pass

193. 6 | 40 | Fail

194. 7 | 85 | Pass

195. 8 | 30 | Fail

196. 9 | 65 | Pass

197. 10 | 40 | Fail

198. 11 | 85 | Pass

199. 12 | 30 | Fail

200. (12 rows)

201.

202.

203. dbms=#

204. dbms=# -- 7. Fixed Procedure to Monitor Attendance and Mark 'Irregular'

205. dbms=# CREATE OR REPLACE PROCEDURE check\_attendance()

206. dbms-# LANGUAGE plpgsql AS $$

207. dbms$# DECLARE

208. dbms$# s\_id INT;

209. dbms$# att\_value INT;

210. dbms$# att\_cursor CURSOR FOR SELECT student\_id, attendance FROM attendance;

211. dbms$# BEGIN

212. dbms$# OPEN att\_cursor;

213. dbms$# LOOP

214. dbms$# FETCH att\_cursor INTO s\_id, att\_value;

215. dbms$# EXIT WHEN NOT FOUND;

216. dbms$# IF att\_value < 75 THEN

217. dbms$# UPDATE attendance SET status = 'Irregular' WHERE student\_id = s\_id;

218. dbms$# END IF;

219. dbms$# END LOOP;

220. dbms$# CLOSE att\_cursor;

221. dbms$# END;

222. dbms$# $$;

223. CREATE PROCEDURE

224. dbms=#

225. dbms=# -- Run procedure 7

226. dbms=# -- First, reset all attendance statuses to default

227. dbms=# UPDATE attendance SET status = 'Regular';

228. UPDATE 4

229. dbms=# SELECT \* FROM attendance;

230. student\_id | attendance | status

231. ------------+------------+---------

232. 1 | 80 | Regular

233. 2 | 60 | Regular

234. 3 | 90 | Regular

235. 4 | 50 | Regular

236. (4 rows)

237.

238.

239. dbms=# CALL check\_attendance();

240. CALL

241. dbms=# SELECT \* FROM attendance;

242. student\_id | attendance | status

243. ------------+------------+-----------

244. 1 | 80 | Regular

245. 3 | 90 | Regular

246. 2 | 60 | Irregular

247. 4 | 50 | Irregular

248. (4 rows)

249.

**Conclusion:**

**FAQs:**

1. What are the main steps to use a cursor in MySQL?

To use a cursor in MySQL stored procedures, you need to follow these main steps:

1. Declare the cursor: Define the cursor and associate it with a SELECT statement
2. Declare variables: Create variables to store the fetched data
3. Declare handlers: Set up handlers for conditions like NOT FOUND
4. Open the cursor: Initialize the cursor to start using it
5. Loop through the result set: Create a loop structure to process each row
6. Fetch data: Retrieve data from cursor into variables
7. Process the data: Perform operations using the fetched data
8. Close the cursor: Release the cursor resources when done
9. How do you open and close a cursor in MySQL?’

In MySQL, cursors are opened and closed using the OPEN and CLOSE statements:

**Opening a cursor**:

OPEN cursor\_name;

When you open a cursor:

* The SELECT statement associated with the cursor is executed
* A result set is created and positioned before the first row
* The cursor is ready to fetch rows

**Closing a cursor**:

CLOSE cursor\_name;

When you close a cursor:

* All resources associated with the cursor are released
* The cursor can no longer be used (unless reopened)
* It's important to always close cursors to prevent memory leaks

1. Explain the use of FETCH with a MySQL cursor.

The FETCH statement retrieves the current row from a cursor and advances the cursor position to the next row. It's used to move through the result set one row at a time.

Syntax:

FETCH cursor\_name INTO variable1, variable2, ..., variableN;

1. How do you handle the NOT FOUND condition while using cursors in MySQL? Explain with an example.

In MySQL, the NOT FOUND condition is raised when a FETCH statement attempts to retrieve a row, but no more rows exist in the result set. This condition needs to be handled to prevent errors and properly exit cursor loops.

There are two main ways to handle the NOT FOUND condition:

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

1. What is the difference between an explicit cursor and an implicit cursor in MySQL?

Explicit Cursors:

* Definition: Cursors that are explicitly declared, opened, fetched, and closed by the programmer
* Control: Provide complete control over how result sets are processed
* Usage: Declared with DECLARE CURSOR statement
* Processing: Allow row-by-row processing of result sets
* Memory: Consume more memory as they hold the entire result set
* Use case: Best for situations where you need to process each row individually or in a specific order