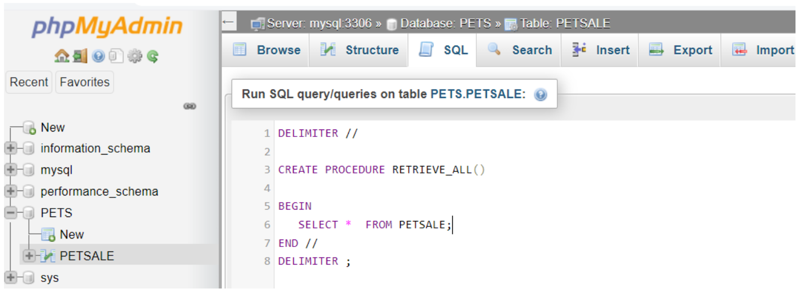
Data Engineering Day 10:

The credit for this course goes to Coursera. [Click More]

Another link : [Azure data Engineer]

Stored Procedure | Advanced Data Engineering

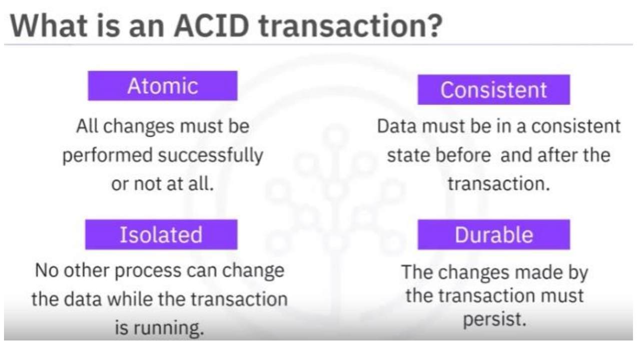


```
DELIMITER //

CREATE PROCEDURE RETRIEVE_ALL()

BEGIN
   SELECT *  FROM PETSALE;
END //
DELIMITER ;
```

The figure above shows that I have created the stored Procedure in SQL. The code mentioned below is also

```
DELIMITER @
CREATE PROCEDURE UPDATE_SALEPRICE (IN Animal_ID INTEGER, IN
Animal_Health VARCHAR(5))
BEGIN
    IF Animal_Health = 'BAD' THEN
        UPDATE PETSALE
        SET SALEPRICE = SALEPRICE - (SALEPRICE * 0.25)
        WHERE ID = Animal_ID;
    ELSEIF Animal_Health = 'WORSE' THEN
        UPDATE PETSALE
        SET SALEPRICE = SALEPRICE - (SALEPRICE * 0.5)
        WHERE ID = Animal_ID;
    ELSE
        UPDATE PETSALE
        SET SALEPRICE = SALEPRICE
        WHERE ID = Animal_ID;
    END IF;
END @
DELIMITER;
```

# ACID transactions:


What is an ACID transaction?

Atomic
All changes must be performed successfully or not at all.

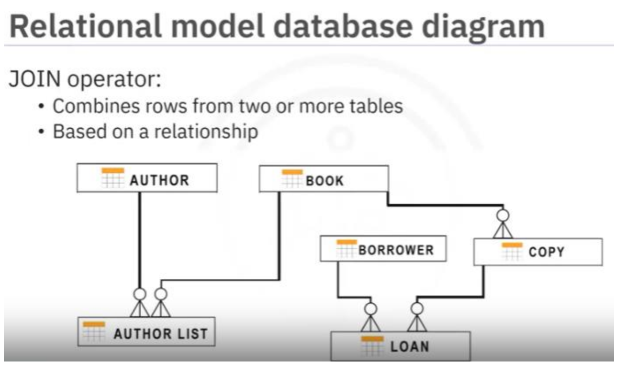
Consistent
Data must be in a consistent state before and after the transaction.

Isolated
No other process can change the data while the transaction is running.

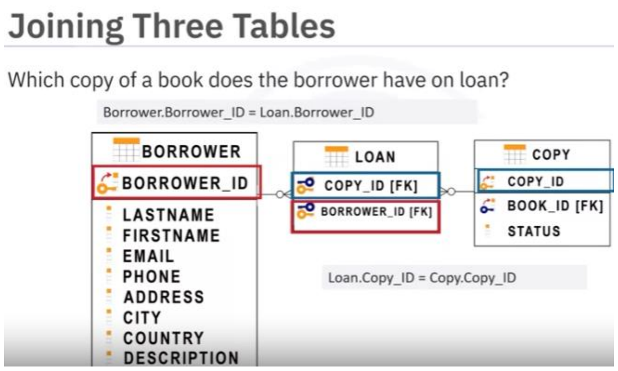
Durable
The changes made by the transaction must persist.

# Join Overviews (Primary Key and Foreign Keys):

- Used to combine data from two tables(it basically combines the rows from two or more tables).


Relational model database diagram

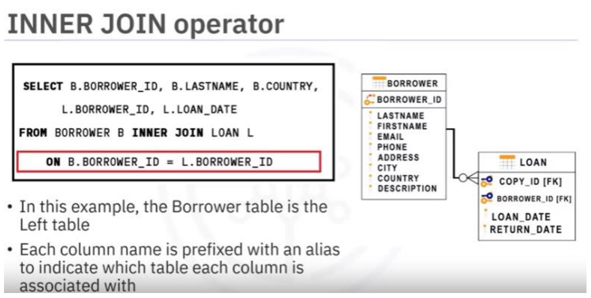
JOIN operator:
- Combines rows from two or more tables
- Based on a relationship


Joining Three Tables

Which copy of a book does the borrower have on loan?

# Inner Joins:

- Displays rows from two tables which consists of matching values.
- The primary key of the first table exists as a foreign key of second table.

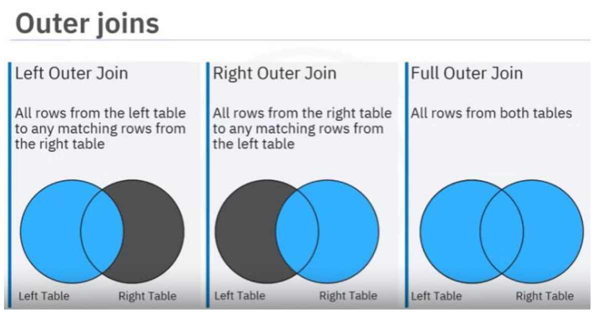


INNER JOIN operator

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
      L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

- In this example, the Borrower table is the Left table
- Each column name is prefixed with an alias to indicate which table each column is associated with

# Outer Joins:

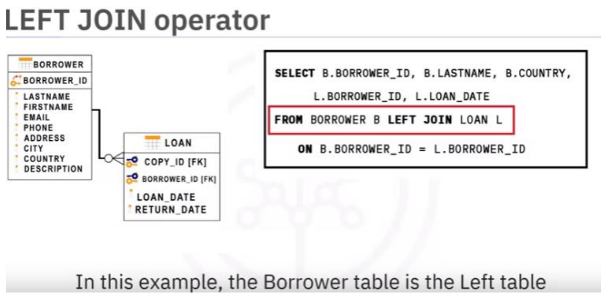


Outer joins

Left Outer Join	Right Outer Join	Full Outer Join
All rows from the left table to any matching rows from the right table	All rows from the right table to any matching rows from the left table	All rows from both tables

1.Left Joins:

- Returns all the rows from the left table and matching rows from the right table.



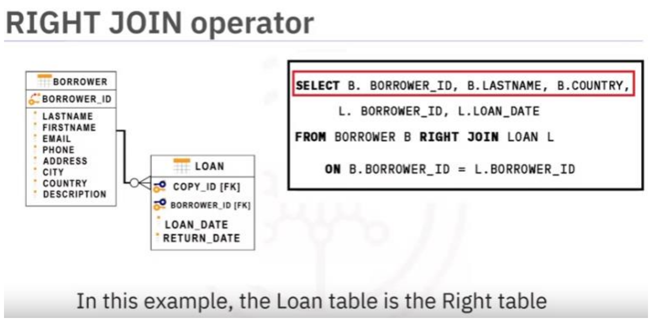
LEFT JOIN operator

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
      L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B LEFT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

In this example, the Borrower table is the Left table

2.Right Joins:

- Returns all the rows from the right table and matching rows from the left table.



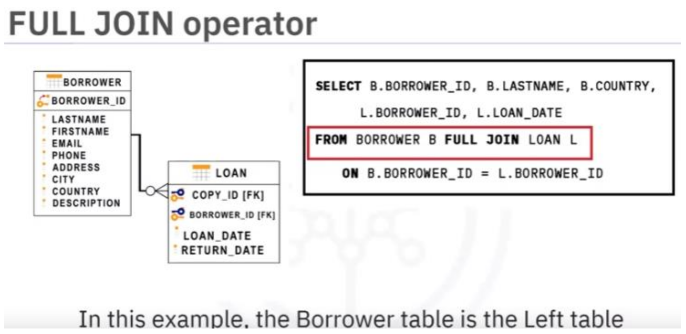
RIGHT JOIN operator

```
SELECT B. BORROWER_ID, B.LASTNAME, B.COUNTRY,
      L. BORROWER_ID, L.LOAN_DATE
FROM BORROWER B RIGHT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

In this example, the Loan table is the Right table

# Full Joins:

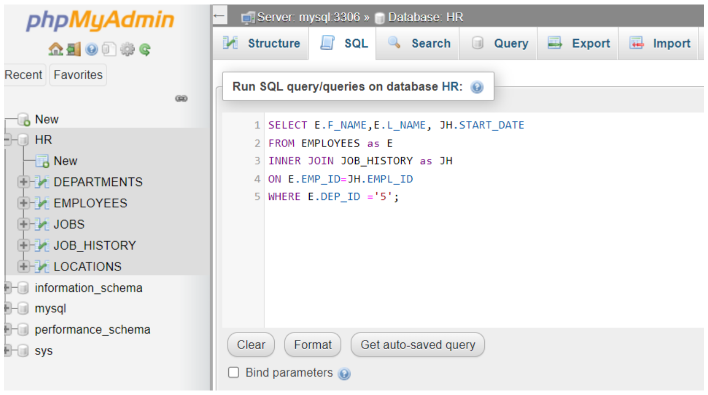
- Returns all the rows when a match in the left or right table.



```sql
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
    L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B FULL JOIN LOAN L
  ON B.BORROWER_ID = L.BORROWER_ID
```

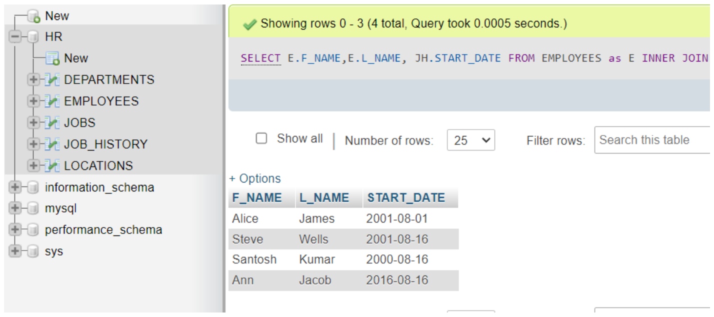
In this example, the Borrower table is the Left table

# Examples of Join function implementations:


phpMyAdmin
Recent Favorites
New
HR
New
DEPARTMENTS
EMPLOYEES
JOBS
JOB_HISTORY
LOCATIONS
information_schema
mysql
performance_schema
sys
Server: mysql:3306 » Database: HR
Structure SQL Search Query Export Import
Run SQL query/queries on database HR:
Clear Format Get auto-saved query
Bind parameters

```sql
SELECT E.F_NAME,E.L_NAME, JH.START_DATE
FROM EMPLOYEES as E
INNER JOIN JOB_HISTORY as JH
ON E.EMP_ID=JH.EMPL_ID
WHERE E.DEP_ID ='5';
```

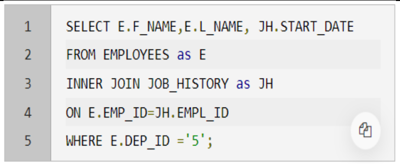
# Results of Join mentioned above



Showing rows 0 - 3 (4 total, Query took 0.0005 seconds.)

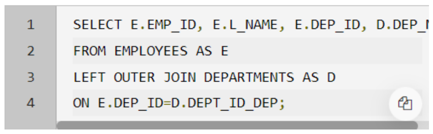
```sql
SELECT E.F_NAME,E.L_NAME, JH.START_DATE FROM EMPLOYEES as E INNER JOIN
```

F_NAME	L_NAME	START_DATE
Alice	James	2001-08-01
Steve	Wells	2001-08-16
Santosh	Kumar	2000-08-16
Ann	Jacob	2016-08-16



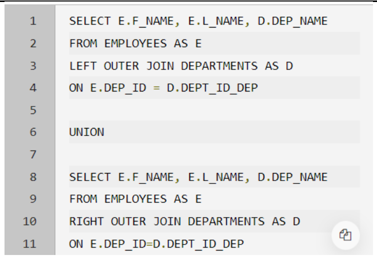
```sql
SELECT E.F_NAME,E.L_NAME, JH.START_DATE
FROM EMPLOYEES as E
INNER JOIN JOB_HISTORY as JH
ON E.EMP_ID=JH.EMPL_ID
WHERE E.DEP_ID ='5';
```

Inner Join



```sql
SELECT E.EMP_ID, E.L_NAME, E.DEP_ID, D.DEP_I
FROM EMPLOYEES AS E
LEFT OUTER JOIN DEPARTMENTS AS D
ON E.DEP_ID=D.DEPT_ID_DEP;
```

Outer Join



```sql
SELECT E.F_NAME, E.L_NAME, D.DEP_NAME
FROM EMPLOYEES AS E
LEFT OUTER JOIN DEPARTMENTS AS D
ON E.DEP_ID = D.DEPT_ID_DEP

UNION

SELECT E.F_NAME, E.L_NAME, D.DEP_NAME
FROM EMPLOYEES AS E
RIGHT OUTER JOIN DEPARTMENTS AS D
ON E.DEP_ID=D.DEPT_ID_DEP
```

Full Join

Summary notes:

Topic	Syntax	Description	Example
Cross Join	`SELECT column_name(s) FROM table1 CROSS JOIN table2;`	The `CROSS JOIN` is used to generate a paired combination of each row of the first table with each row of the second table.	`SELECT DEPT_ID_DEP, LOCT_ID FROM DEPARTMENTS CROSS JOIN LOCATIONS;`
Inner Join	`SELECT column_name(s) FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name; WHERE condition;`	You can use an `inner join` in a SELECT statement to retrieve only the rows that satisfy the join conditions on every specified table.	`select E.F_NAME,E.L_NAME, JH.START_DATE from EMPLOYEES as E INNER JOIN JOB_HISTORY as JH on E.EMP_ID=JH.EMPL_ID where E.DEP_ID ='5';`
Left Outer Join	`SELECT column_name(s) FROM table1 LEFT OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;`	The `LEFT OUTER JOIN` will return all records from the left side table and the matching records from the right table.	`select E.EMP_ID,E.L_NAME,E.DEP_ID,D.DEP_NAME from EMPLOYEES AS E LEFT OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP;`
Right Outer Join	`SELECT column_name(s) FROM table1 RIGHT OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;`	The `RIGHT OUTER JOIN` returns all records from the right table, and the matching records from the left table.	`select E.EMP_ID,E.L_NAME,E.DEP_ID,D.DEP_NAME from EMPLOYEES AS E RIGHT OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP;`
Full Outer Join	`SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON table1.column_name = table2.column_name WHERE condition;`	The `FULL OUTER JOIN` clause results in the inclusion of rows from two tables. If a value is missing when rows are joined, that value is null in the result table.	`select E.F_NAME,E.L_NAME,D.DEP_NAME from EMPLOYEES AS E FULL OUTER JOIN DEPARTMENTS AS D ON E.DEP_ID=D.DEPT_ID_DEP;`
Self Join	`SELECT column_name(s) FROM table1 T1, table1 T2 WHERE condition;`	A `self join` is regular join but it can be used to joined with itself.	`SELECT B.* FROM EMPLOYEES A JOIN EMPLOYEES B ON A.MANAGER_ID = B.MANAGER_ID WHERE A.EMP_ID = 'E1001';`

SQL Cheat Sheet: Views, Stored Procedures and Transactions

Skills Network

Views

Topic	Syntax	Description	Example
Create View	`CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;`	A `CREATE VIEW` is an alternative way of representing data that exists in one or more tables.	`CREATE VIEW EMPSALARY AS SELECT EMP_ID, F_NAME, L_NAME, B_DATE, SEX, SALARY FROM EMPLOYEES;`
Update a View	`CREATE OR REPLACE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;`	The `CREATE OR REPLACE` VIEW command updates a view.	`CREATE OR REPLACE VIEW EMPSALARY AS SELECT EMP_ID, F_NAME, L_NAME, B_DATE, SEX, JOB_TITLE, MIN_SALARY, MAX_SALARY FROM EMPLOYEES, JOBS WHERE EMPLOYEES.JOB_ID = JOBS.JOB_IDENT;`
Drop a View	`DROP VIEW view_name;`	Use the `DROP VIEW` statement to remove a view from the database.	`DROP VIEW EMPSALARY;`

Stored Procedures in IBM Db2 using SQL

Stored Procedures	`--#SET TERMINATOR @ CREATE PROCEDURE PROCEDURE_NAME`
	
	`LANGUAGE`
	
	`BEGIN`
	
	`END`
	`@`
		A `stored procedure` is a prepared SQL code that you can save, so the code can be reused over and over again.
		
		The default terminator for a stored procedure is semicolon(;). To set a different terminator we use `SET TERMINATOR` clause followed by the terminator such as ‘@’.
			`--#SET TERMINATOR @ CREATE PROCEDURE RETRIEVE_ALL`
			
			`LANGUAGE SQL`
			`READS SQL DATA`
			
			`DYNAMIC RESULT SETS 1`
			`BEGIN`
			
			`DECLARE C1 CURSOR`
			`WITH RETURN FOR`
			
			`SELECT * FROM PETSALE;`
			
			`OPEN C1;`
			
			`END`
			`@`

Stored Procedures in MySQL using phpMyAdmin

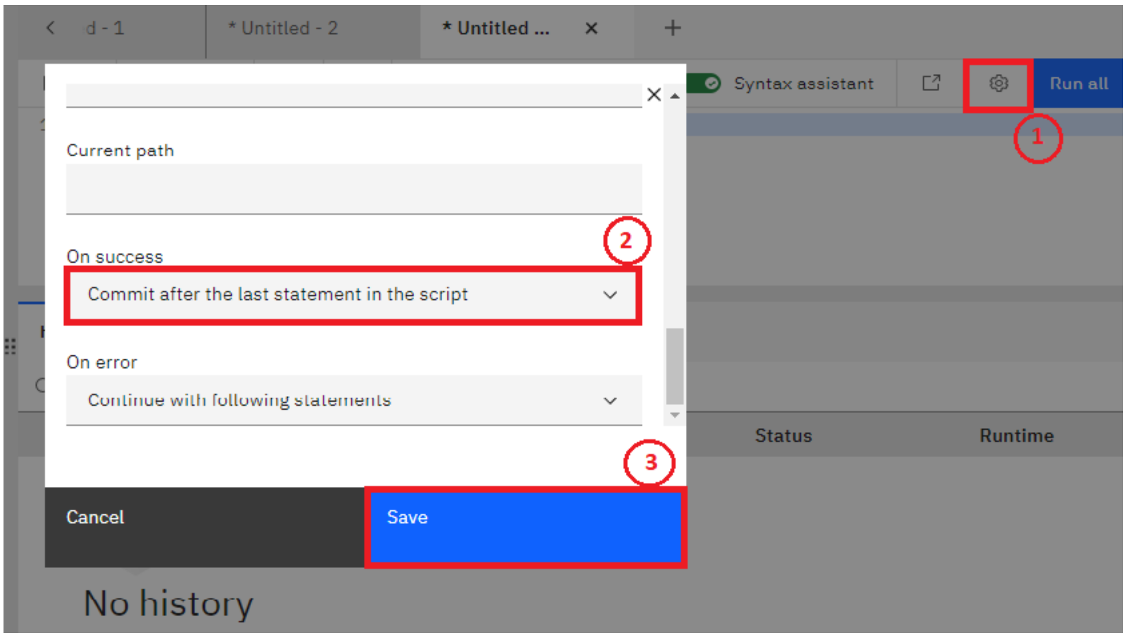
Stored Procedures	`DELIMITER //`
	
	`CREATE PROCEDURE PROCEDURE_NAME`
	
	`BEGIN`
	
	`END //`
	
	`DELIMITER ;`
		A `stored procedure` is a prepared SQL code that you can save, so the code can be reused over and over again.
		
		The default terminator for a stored procedure is semicolon (;). To set a different terminator we use `DELIMITER` clause followed by the terminator such as $$ or //.
			`DELIMITER //`
			
			`CREATE PROCEDURE RETRIEVE_ALL()`
			
			`BEGIN`
			
			`SELECT * FROM PETSALE;`
			
			`END //`
			
			`DELIMITER ;`

Transactions with Db2

Commit command	`COMMIT;`	A `COMMIT` command is used to persist the changes in the database.
		
		The default terminator for a COMMIT command is semicolon (;).
			`CREATE TABLE employee(ID INT, Name VARCHAR(20), City VARCHAR(20), Salary INT, Age INT);`
			
			`INSERT INTO employee( ID, Name, City, Salary, Age) VALUES( 1, ‘Priyanka pal’, ‘Nasik’, 36000, 21), (2, ‘Riya chowdary’`
			`82000, 29);`
			
			`SELECT *FROM employee;`
			`COMMIT;`

Rollback command	`ROLLBACK;`	A `ROLLBACK` command is used to rollback the transactions which are not saved in the database.
		
		The default terminator for a ROLLBACK command is
			`As auto-commit is enabled by default, all transactions will be committed. We need to disable this option to see rollback works.`
			
			`For db2, we have to disable auto-commit manually. Click the gear icon located on the right side of the SQL Assis window. Next, select the “On Success” drop-down and choose “commit after the last statement in the script” Remem save your changes!`

	semicolon (;).		



Current path

On success

Commit after the last statement in the script

On error

Continue with following statements

Cancel

Save

```
INSERT INTO employee VALUES (3, ‘Swetha Tiwari’, ‘Kanpur’, 38000, 38);

SELECT *FROM employee;
ROLLBACK;
SELECT *FROM employee;
```

Transactions with MySQL

			`CREATE TABLE employee(ID INT, Name VARCHAR(20), City VARCHAR(20), Salary INT, Age INT);`

Commit command	`COMMIT;`	A `COMMIT command` is used to persist the changes in the database.
		
		The default terminator for a COMMIT command is semicolon (;).
			`START TRANSACTION;`
			
			`INSERT INTO employee( ID, Name, City, Salary, Age) VALUES( 1, ‘Priyanka pal’, ‘Nasik’, 36000, 21), (2, ‘Riya chowdary’, ‘Bangalor’, 82000, 29);`
			
			`SELECT *FROM employee;`
			`COMMIT;`

Rollback command	`ROLLBACK;`	A `ROLLBACK command` is used to rollback the transactions which are not saved in the database.
		
		The default terminator for a ROLLBACK command is semicolon (;).
			`As auto-commit is enabled by default, all transactions will be committed. We need to disable this option to see how rollback works. For MySQL use the command “SET autocommit = 0;”`
			
			`INSERT INTO employee VALUES (3, ‘Swetha Tiwari’, ‘Kanpur’, 38000, 38);`
			
			`SELECT *FROM employee;`
			`ROLLBACK;`
			`SELECT *FROM employee;`

Db2 Transactions using Stored Procedure

Commit command	`-#SET TERMINATOR @`
	
	`CREATE PROCEDURE PROCEDURE_NAME`
	
	`BEGIN`
	
	`COMMIT;`
	
	`END`
	`@`
		A `COMMIT command` is used to persist the changes in the database.
		
		The default terminator for a COMMIT command is semicolon (;).
			`--#SET TERMINATOR @ CREATE PROCEDURE TRANSACTION_ROSE LANGUAGE SQL MODIFIES SQL DATA`
			
			`BEGIN`
			
			`DECLARE SQLCODE INTEGER DEFAULT 0;`
			`DECLARE retcode INTEGER DEFAULT 0;`
			`DECLARE CONTINUE HANDLER FOR SQLEXCEPTION`
			`SET retcode = SQLCODE;`
			
			`UPDATE BankAccounts`
			`SET Balance = Balance-200`
			`WHERE AccountName = ‘Rose’;`
			
			`UPDATE BankAccounts`
			`SET Balance = Balance-300`
			`WHERE AccountName = ‘Rose’;`
			
			`IF retcode < 0 THEN`
			`ROLLBACK WORK;`
			
			`ELSE`
			`COMMIT WORK;`
			
			`END IF;`
			
			`END`
			`@`

Rollback command	`--#SET TERMINATOR @`
	
	`CREATE PROCEDURE PROCEDURE_NAME`
	
	`BEGIN`
		A `ROLLBACK command` is used to rollback the transactions which are not saved in the database.	`--#SET TERMINATOR @ CREATE PROCEDURE TRANSACTION_ROSE LANGUAGE SQL MODIFIES SQL`

	`ROLLBACK;`
	
	`COMMIT;`
	
	`END`
	`@`
		The default terminator for a ROLLBACK command is semicolon (;).	`DATA`
			
			`BEGIN`
			
			`DECLARE SQLCODE INTEGER DEFAULT 0;`
			`DECLARE retcode INTEGER DEFAULT 0;`
			`DECLARE CONTINUE HANDLER FOR SQLEXCEPTION`
			`SET retcode = SQLCODE;`
			
			`UPDATE BankAccounts`
			`SET Balance = Balance-200`
			`WHERE AccountName = ‘Rose’;`
			
			`UPDATE BankAccounts`
			`SET Balance = Balance-300`
			`WHERE AccountName = ‘Rose’;`
			
			`IF retcode < 0 THEN`
			`ROLLBACK WORK;`
			
			`ELSE`
			`COMMIT WORK;`
			
			`END IF;`
			
			`END`
			`@`

MySQL Transactions using Stored Procedure

Commit command	`DELIMITER //`
	
	`CREATE PROCEDURE PROCEDURE_NAME`
	
	`BEGIN`
	
	`COMMIT;`
	
	`END //`
	
	`DELIMITER ;`
		A `COMMIT command` is used to persist the changes in the database.
		
		The default terminator for a COMMIT command is semicolon (;).
			`DELIMITER //`
			
			`CREATE PROCEDURE TRANSACTION_ROSE()`
			
			`BEGIN`
			
			`DECLARE EXIT HANDLER FOR SQLEXCEPTION`
			`BEGIN`
			`ROLLBACK;`
			`RESIGNAL;`
			`END;`
			
			`START TRANSACTION;`
			`UPDATE BankAccounts`
			`SET Balance = Balance-200`
			`WHERE AccountName = ‘Rose’;`
			
			`UPDATE BankAccounts`
			`SET Balance = Balance-300`
			`WHERE AccountName = ‘Rose’;`
			
			`COMMIT;`
			
			`END //`
			
			`DELIMITER ;`

Rollback command	`DELIMITER //`
	
	`CREATE PROCEDURE PROCEDURE_NAME`
	
	`BEGIN`
	
	`ROLLBACK;`
	
	`COMMIT;`
	
	`END //`
	
	`DELIMITER ;`
		A `ROLLBACK command` is used to rollback the transactions which are not saved in the database.
		
		The default terminator for a ROLLBACK command is semicolon (;).
			`DELIMITER //`
			
			`CREATE PROCEDURE TRANSACTION_ROSE()`
			
			`BEGIN`
			
			`DECLARE EXIT HANDLER FOR SQLEXCEPTION`
			`BEGIN`
			`ROLLBACK;`
			`RESIGNAL;`
			`END;`
			
			`START TRANSACTION;`
			`UPDATE BankAccounts`
			`SET Balance = Balance-200`
			`WHERE AccountName = ‘Rose’;`
			
			`UPDATE BankAccounts`
			`SET Balance = Balance-300`
			`WHERE AccountName = ‘Rose’;`
			
			`COMMIT;`
			
			`END //`
			
			`DELIMITER ;`

Author(s)

D.M Naidu

Changelog

Date	Version	Changed by	Change Description
2022-10-04	1.0	D.M.Naidu	Initial Version