DATABASE MANAGEMENT SYSTEMS
CENG 3005

Final Project Report

TEAM MEMBERS
Ahsen Pehlivan- 230709039
Görkem Özden- 230709078
Ayşe Kıllı- 230709039

Project Description

This project focuses on designing and implementing a relational database system for analyzing oil pipeline accident records. The database organizes real-world accident data to enable efficient querying of financial losses, environmental impacts, and accident causes. Using a normalized schema in MySQL, the system supports meaningful analytical queries to better understand pipeline safety and risk factors.

Changes Since Phase I Submission

Since the Phase I submission, several important improvements have been made to both the project description and the database design to increase normalization, flexibility, and maintainability.
The most critical change is the decomposition of the large central table used in the initial design. Attributes related to accident outcomes were separated into a new table named Accident_Result, which is now linked to Accident_Report in a one-to-one relationship. Furthermore, the attributes within Accident_Result were reorganized into three logical groups—Impacts, Costs, and Spill Volumes—to improve usability and to allow easier extension of the schema if new result types are added in the future.

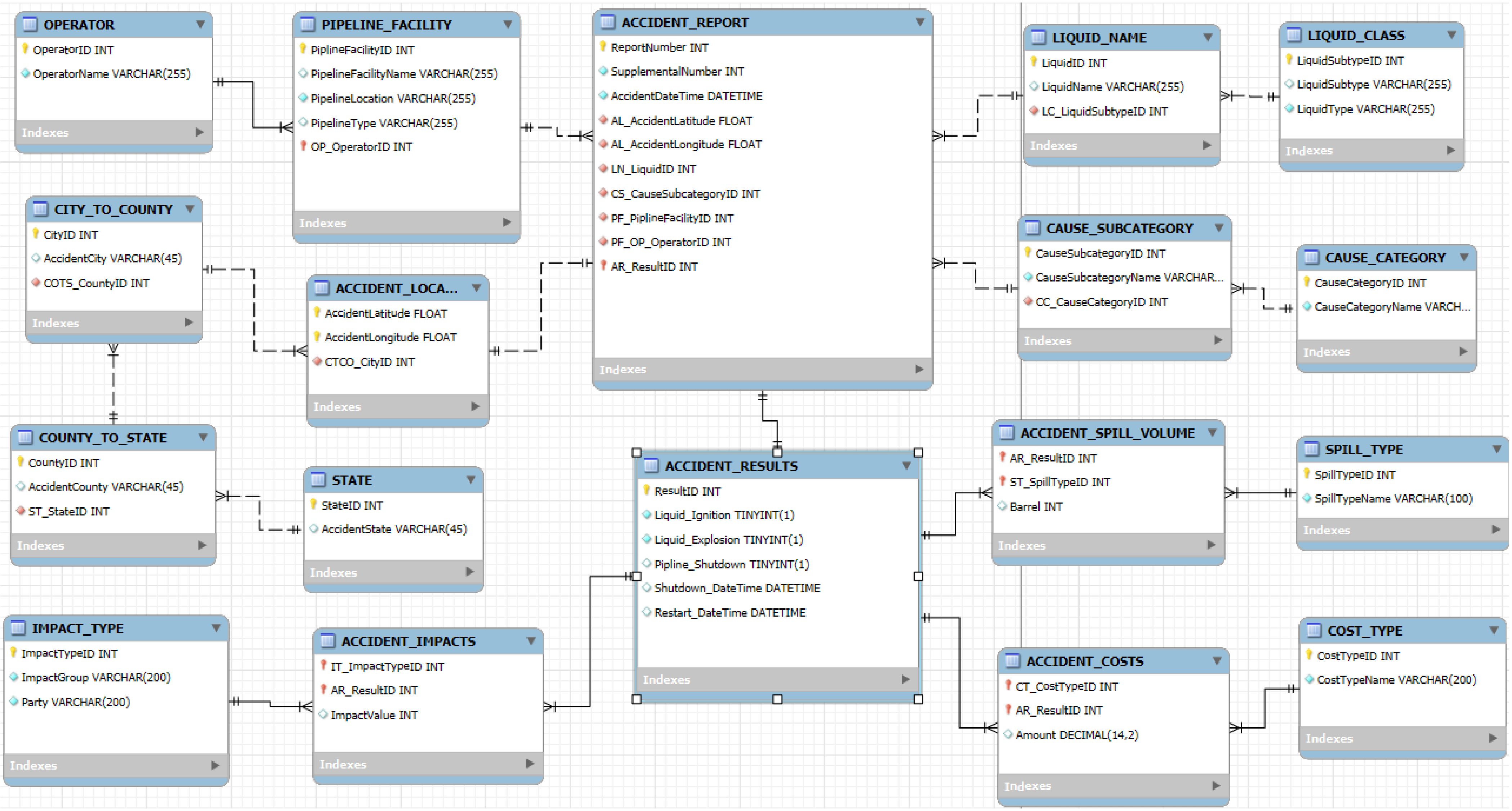
In addition, the Accident_Location table was normalized by splitting it into separate State, City, and County tables, each with its own auto-increment primary key and appropriate relationships. This change reduces data redundancy and improves location-based querying.

Several tables were also updated to replace natural keys with surrogate keys. The Pipeline_Facility table previously used the facility name as the primary key; in the new design, an auto-increment ID was introduced instead. Similarly, auto-increment primary keys were added to the Liquid_Name and Liquid_Class tables, replacing name-based primary keys.

Finally, the Cause structure was refined by separating it into Cause_Category and Cause_Subcategory tables, each with its own auto-increment ID, allowing clearer classification and better support for future expansion.

After Phase I, we normalized the schema by splitting the central accident table into multiple related tables (e.g., results/cost/location). Therefore, we updated our view and stored procedure definitions to match the new schema and verified them by executing sample queries.

Overall, these changes significantly improved the normalization level of the database, reduced redundancy, and made the schema more scalable and easier to maintain compared to the Phase I design.



OPERATOR
OperatorID INT
OperatorName VARCHAR(255)
Indexes
PIPELINE_FACILITY
PiplineFacilityID INT
PipelineFacilityName VARCHAR(255)
PipelineLocation VARCHAR(255)
PipelineType VARCHAR(255)
OP_OperatorID INT
Indexes
ACCIDENT_REPORT
ReportNumber INT
SupplementalNumber INT
AccidentDateTime DATETIME
AL_AccidentLatitude FLOAT
AL_AccidentLongitude FLOAT
LN_LiquidID INT
CS_CauseSubcategoryID INT
PF_PiplineFacilityID INT
PF_OP_OperatorID INT
AR_ResultID INT
Indexes
LIQUID_NAME
LiquidID INT
LiquidName VARCHAR(255)
LC_LiquidSubtypeID INT
Indexes
LIQUID_CLASS
LiquidSubtypeID INT
LiquidSubtype VARCHAR(255)
LiquidType VARCHAR(255)
Indexes
CITY_TO_COUNTY
CityID INT
AccidentCity VARCHAR(45)
COTS_CountyID INT
Indexes
ACCIDENT_LOCA...
AccidentLatitude FLOAT
AccidentLongitude FLOAT
CTCO_CityID INT
Indexes
CAUSE_SUBCATEGORY
CauseSubcategoryID INT
CauseSubcategoryName VARCHAR...
CC_CauseCategoryID INT
Indexes
CAUSE_CATEGORY
CauseCategoryID INT
CauseCategoryName VARCH...
Indexes
COUNTY_TO_STATE
CountyID INT
AccidentCounty VARCHAR(45)
ST_StateID INT
Indexes
STATE
StateID INT
AccidentState VARCHAR(45)
Indexes
ACCIDENT_RESULTS
ResultID INT
Liquid_Ignition TINYINT(1)
Liquid_Explosion TINYINT(1)
Pipline_Shutdown TINYINT(1)
Shutdown_DateTime DATETIME
Restart_DateTime DATETIME
Indexes
ACCIDENT_SPILL_VOLUME
AR_ResultID INT
ST_SpillTypeID INT
Barrel INT
Indexes
SPILL_TYPE
SpillTypeID INT
SpillTypeName VARCHAR(100)
Indexes
IMPACT_TYPE
ImpactTypeID INT
ImpactGroup VARCHAR(200)
Party VARCHAR(200)
Indexes
ACCIDENT_IMPACTS
IT_ImpactTypeID INT
AR_ResultID INT
ImpactValue INT
Indexes
ACCIDENT_COSTS
CT_CostTypeID INT
AR_ResultID INT
Amount DECIMAL(14,2)
Indexes
COST_TYPE
CostTypeID INT
CostTypeName VARCHAR(200)
Indexes

Loading Database

Data Source and Tools:
For this project, a pipeline accident dataset obtained from Kaggle was utilized (Source URL: https://www.kaggle.com/datasets/usdot/pipeline-accidents). Due to the dataset's size (approximately 2,795 records), a Python script was developed using pandas and mysql-connector libraries to automate the loading process, rather than relying on manual entry.

Pre-processing:
Prior to database insertion, data inconsistencies were resolved programmatically.
- Date Formats: Non-standard dates were converted into the MySQL-compatible YYYY-MM-DD format.
- Boolean Mapping: Text-based "YES/NO" values were mapped to binary 1/0 to align with the database schema.
- Null Handling: Missing values (NaN) were identified and converted to NULL or 0 depending on the data type.

Loading Strategy:
To accommodate the relational Star Schema design, a normalized loading approach was used. First, unique information (like Operators) was saved into separate tables. Then, the main accident reports were linked to these tables using IDs.

Data Completeness:
All available records from the source CSV were successfully uploaded to the database; no data was excluded. However, the data structure was transformed for normalization purposes. Specifically, multiple cost-related columns in the source file were pivoted and stored as individual rows within the ACCIDENT_COSTS table.

Brief User's Guide (Views + Stored Procedures)

Views
v_state_accident_summary:
Summarizes pipeline accidents by state, including accident count, total spill amounts, net loss in barrels, and total accident-related costs. It is used to compare accident impact across states.

v_operator_netloss_cost_summary:
Provides a per-operator summary of total net spill loss (barrels) and total accident cost. It helps identify operators with the highest environmental and financial impact.

v_causecategory_accident_count:
Counts the number of accidents for each cause category to identify the most common causes of pipeline accidents.

Stored Procedures

sp_operator_annual_summary (IN, OUT, INOUT):
For a given operator and year, returns the number of accidents (OUT) and updates a running total accident cost using an INOUT parameter.

sp_state_annual_summary (IN, OUT, INOUT):
For a given state and year, returns the number of accidents (OUT) and updates a running total accident cost using an INOUT parameter.

Outputs of Selected Views and Stored Procedures

Stored Procedure Outputs
Operator annual summary output:

OperatorYear_AccidentCount	OperatorYear_RunningTotalCost
14	894516851.00

CALL sp_operator_annual_summary(11169, 2010, @acc_count, @running_total);
Demonstrates parameterized execution using IN, OUT, and INOUT variables.

State annual summary output:

StateYear_AccidentCount	StateYear_RunningTotalCost
114	6682194.00

CALL sp_state_annual_summary('TX', 2010, @acc_count, @running_total);
Demonstrates accident count and cost aggregation by state and year using stored procedures.

View Outputs
State accident summary output:

AccidentState	AccidentCount	TotalUnintentionalRelease_Barrels	TotalIntentionalRelease_Barrels	TotalRecovered_Barrels	TotalNetLoss_Barrels	TotalCost_USD
TX	1007	716052	1005781	221685	1500148	460201946.00
OK	238	146592	177337	64729	259200	113540563.00
LA	169	175701	513689	39004	650386	295685453.00
CA	153	47832	47832	33032	62632	550123724.00
KS	150	55886	61695	30952	86629	48717328.00
IL	109	200417	200437	64142	336712	269700386.00
WY	99	24451	29518	11356	42613	18019655.00
NJ	82	6123	6123	4239	8007	26007344.00
MN	59	21163	21586	6279	36470	36542991.00
IN	57	17675	17675	14144	21206	85220746.00

Shows the states with the highest number of accidents and their associated spill losses and costs.

Operator net loss and cost summary output:

OperatorID	OperatorName	TotalNetLoss_Barrels	TotalCost_USD
31618	ENTERPRISE PRODUCTS OPERATING LLC	967342	91188207.00
31627	DENBURY ONSHORE, LLC	280766	616939.00
11169	ENBRIDGE ENERGY, LIMITED PARTNERSHIP	241881	2690255206.00
32109	ONEOK NGL PIPELINE LP	208900	12874532.00
2731	CHEVRON PIPE LINE CO	190728	279299443.00
30829	TEPPCO CRUDE PIPELINE, LLC	184384	75788961.00
31570	TESORO HIGH PLAINS PIPELINE COMPAN...	177502	51257380.00
32543	DENBURY GREEN PIPELINE-TEXAS, LLC	172736	481929.00
18718	SUNOCO PIPELINE L.P.	159631	120532054.00
31556	CHEVRON MIDSTREAM PIPELINES LLC	145429	31953644.00

Highlights operators with the largest net spill losses and total costs.

Cause category accident count output:

CauseCategoryID	CauseCategory	AccidentCount
1	MATERIAL/WELD/EQUIP FAILURE	1435
4	CORROSION	592
5	INCORRECT OPERATION	378
2	NATURAL FORCE DAMAGE	118
6	ALL OTHER CAUSES	118
3	EXCAVATION DAMAGE	97
7	OUTSIDE FORCE DAMAGE	57

Displays the most frequent cause categories of pipeline accidents.

System Limitations and Suggested Improvements

Limitations

- The dataset was obtained from a single public source, which may contain missing or inconsistent values across some attributes.
- Some categorical fields (such as spill type and cost type names) include naming variations, requiring pattern-based matching in queries.
- Accident locations are linked using latitude and longitude values, which may introduce precision and matching limitations.
- Complex analytical queries require multiple joins across normalized tables, which can impact query performance on large datasets.

Suggested Improvements

- Introduce additional data cleaning and standardization during the data loading (ETL) phase to enforce consistent naming conventions.
- Replace latitude/longitude-based joins with a unique location identifier to improve reliability and performance.
- Add indexes on frequently queried attributes such as accident date, operator ID, and state ID to optimize query execution.
- Extend the system with a simple application interface or dashboard to allow non-technical users to interact with the database more easily.

Full DDL Specification

The complete SQL DDL specification of our database is provided in Appendix A (pipline_db_ddl.sql). The script includes each table definition with attribute data types, NOT NULL constraints where applicable, primary keys, and foreign key (referential) constraints. In addition, we added a sample row as SQL comments under each table to illustrate the expected format of the data.
We upload the SQL file in DYS.

SQL Code and Supporting Programs

All SQL code used in this project is included in Appendix B. This includes table creation scripts (DDL), views, stored procedures, and analytical SQL queries. In addition, a Python script was developed and used to preprocess and load the dataset into the MySQL database. Due to the length of the SQL and data-loading scripts, representative samples are shown in the report, while the complete code is submitted digitally via DYS.

Example Table Definition (SQL – DDL)

```
CREATE TABLE `accident_costs` (
  `CT_CostTypeID` int NOT NULL,
  `AR_ResultID` int NOT NULL,
  `Amount` decimal(14,2) DEFAULT NULL,
  PRIMARY KEY (`CT_CostTypeID`,`AR_ResultID`),
  KEY `AR_ResultID` (`AR_ResultID`),
  CONSTRAINT `accident_costs_ibfk_1` FOREIGN
KEY (`CT_CostTypeID`) REFERENCES
`cost_type` (`CostTypeID`),
  CONSTRAINT `accident_costs_ibfk_2` FOREIGN
KEY (`AR_ResultID`) REFERENCES
```

```
 KEY (`AR_ResultID`) REFERENCES
`accident_results` (`ResultID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client =
@saved_cs_client */;

SELECT * FROM accident_costs LIMIT 1;
# CT_CostTypeID   AR_ResultID          Amount
--    1            1           26000.00
```

Example View

```
CREATE OR REPLACE VIEW v_causecategory_accident_count AS
SELECT
   cc.CauseCategoryID,
   cc.CauseCategoryName AS CauseCategory,
   COUNT(*) AS AccidentCount
FROM ACCIDENT_REPORT ar
JOIN CAUSE_SUBCATEGORY cs
  ON ar.CS_CauseSubcategoryID = cs.CauseSubcategoryID
JOIN CAUSE_CATEGORY cc
  ON cs.CC_CauseCategoryID = cc.CauseCategoryID
GROUP BY cc.CauseCategoryID, cc.CauseCategoryName;
```

Example Stored Procedure

```
CREATE PROCEDURE sp_state_annual_summary(
   IN p_state VARCHAR(45),
   IN p_year INT,
   OUT p_accident_count INT,
   INOUT p_total_cost DECIMAL(14,2)
)
BEGIN
   DECLARE v_year_cost DECIMAL(14,2);

   -- OUT: Accident count for state/year
   SELECT COUNT(*)
    INTO p_accident_count
   FROM ACCIDENT_REPORT ar
   JOIN ACCIDENT_LOCATION al
    ON ar.AL_AccidentLatitude  = al.AccidentLatitude
   AND ar.AL_AccidentLongitude = al.AccidentLongitude
   JOIN CITY_TO_COUNTY ctc
    ON al.CTCO_CityID = ctc.CityID
   JOIN COUNTY_TO_STATE cts
    ON ctc.COTS_CountyID = cts.CountyID
   JOIN STATE s
    ON cts.ST_StateID = s.StateID
   WHERE s.AccidentState = p_state
    AND YEAR(ar.AccidentDateTime) = p_year;
```

```
   -- Total cost for state/year
   SELECT IFNULL(SUM(ac.Amount), 0)
    INTO v_year_cost
   FROM ACCIDENT_REPORT ar
   JOIN ACCIDENT_LOCATION al
    ON ar.AL_AccidentLatitude  = al.AccidentLatitude
   AND ar.AL_AccidentLongitude = al.AccidentLongitude
   JOIN CITY_TO_COUNTY ctc
    ON al.CTCO_CityID = ctc.CityID
   JOIN COUNTY_TO_STATE cts
    ON ctc.COTS_CountyID = cts.CountyID
   JOIN STATE s
    ON cts.ST_StateID = s.StateID
   JOIN ACCIDENT_COSTS ac
    ON ar.AR_ResultID = ac.AR_ResultID
   WHERE s.AccidentState = p_state
    AND YEAR(ar.AccidentDateTime) = p_year;

   -- INOUT: running total update
   SET p_total_cost = IFNULL(p_total_cost, 0) + IFNULL(v_year_cost, 0);
END $$

DELIMITER ;
```

Example Analytical Query

```
SELECT
   cc.CauseCategoryName AS CauseCategory,
   COUNT(*) AS AccidentCount
FROM ACCIDENT_REPORT ar
JOIN CAUSE_SUBCATEGORY cs
  ON ar.CS_CauseSubcategoryID =
cs.CauseSubcategoryID
JOIN CAUSE_CATEGORY cc
  ON cs.CC_CauseCategoryID = cc.CauseCategoryID
GROUP BY cc.CauseCategoryName
ORDER BY AccidentCount DESC;
```

Example Python Data Loading Script

https://github.com/ahsenpehlivan/CENG-3005-DatabaseManagementSystem/blob/main/data_loader.py

This script reads the raw dataset, performs basic preprocessing, and inserts the cleaned data into the MySQL database.