Asphalt Construction Co.

IND ENG 115 | Team 4

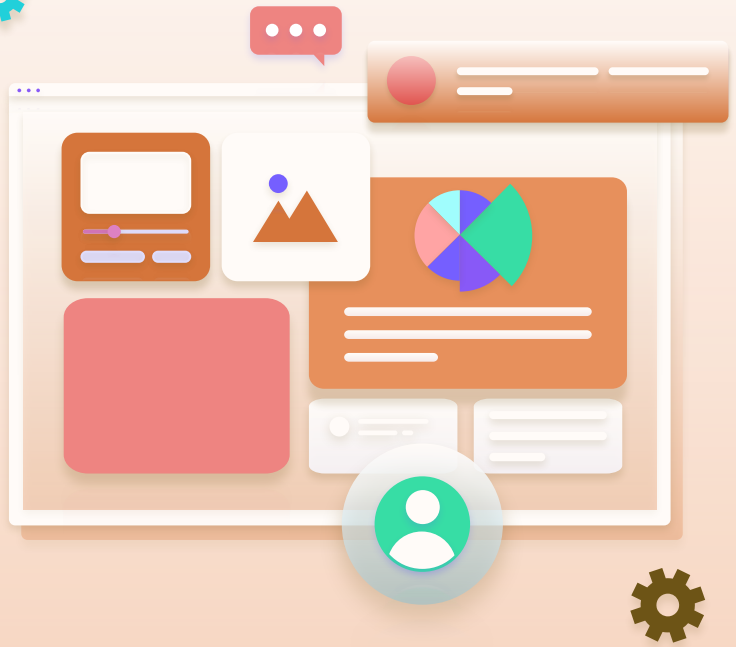
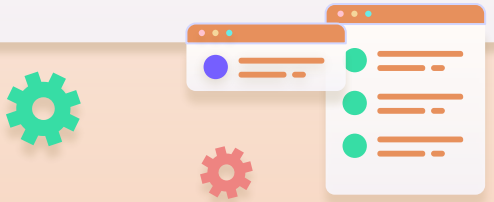
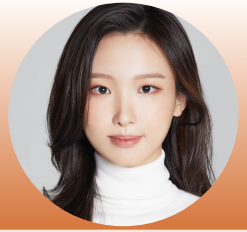


TABLE OF CONTENTS



Team

Davita Verma
CEO



Semin Yoo
Data Analyst

Cayden Sewell
Primary Client Liaison

Chandni Jain
Database Designer

Rupali Sarathy
Database Designer

Joelle Siong Sin
CCO

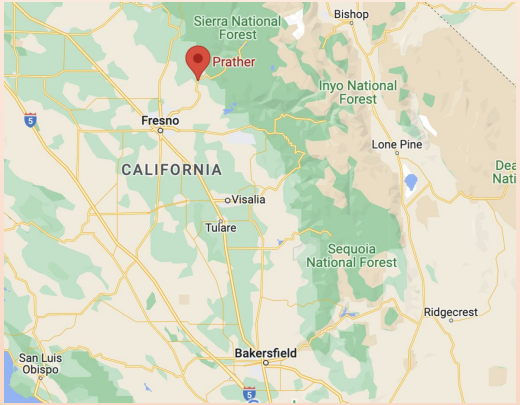
Lea Huynh
Data Analyst

Marco Morales
Data Analyst

Khoa Nguyen
Data Analyst

Jelo Francisco
Database Designer

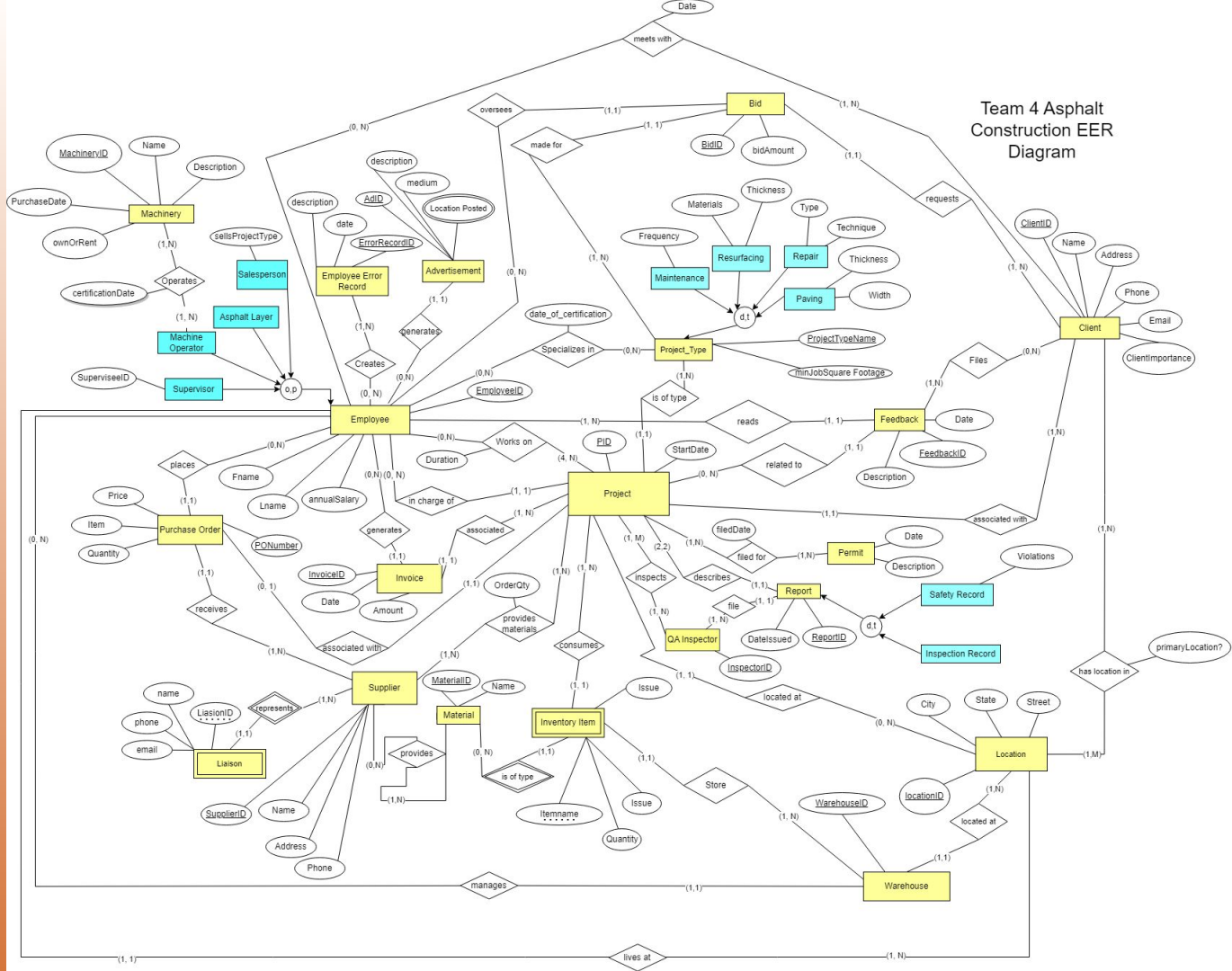
About Our Client



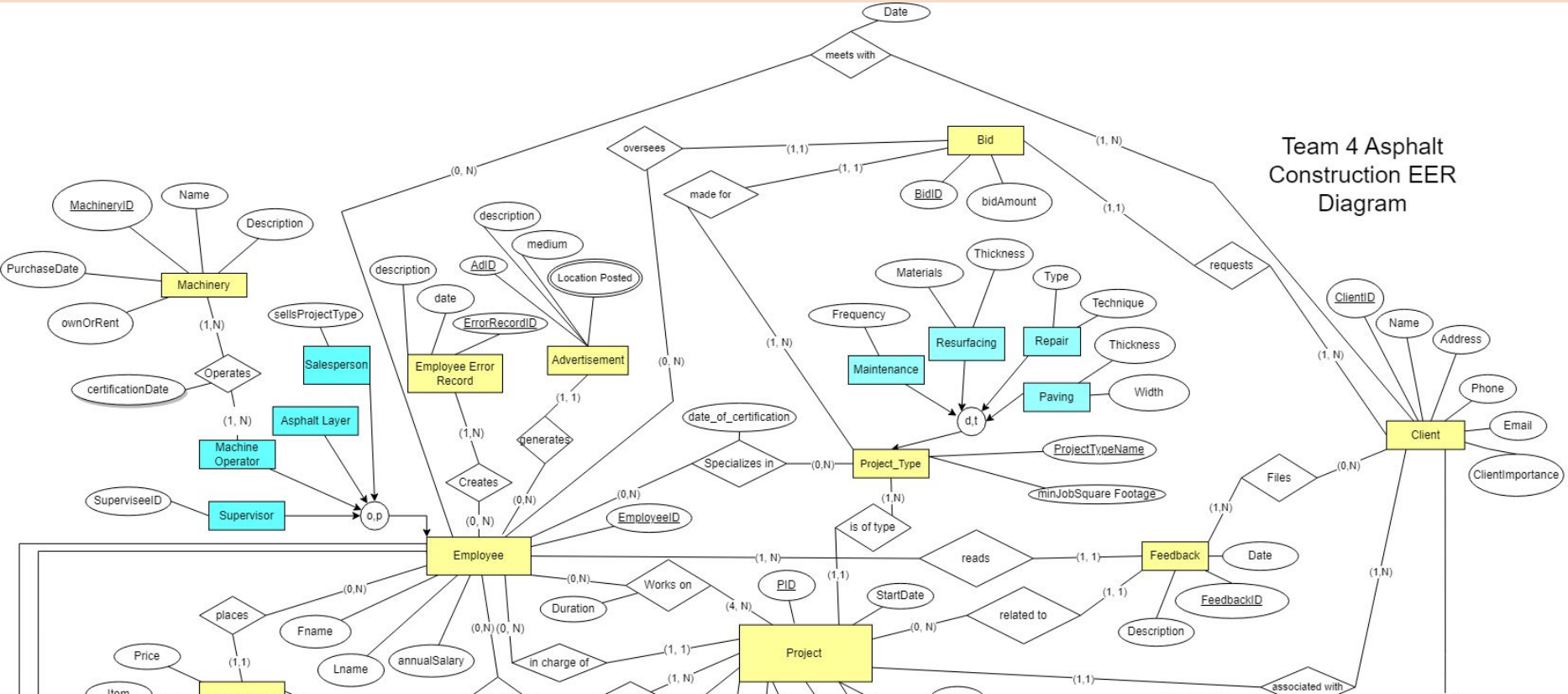
- **Asphalt Construction**- roads, driveways, parking lots, etc.
- Based out of **Prather, California**, but completes jobs throughout the **Central Valley**
- **Family-owned** and operated, providing service to around **fifty clients a year**
- Provide **customized services** to each customer by getting to know them personally, which contributes to their solid client base and continued success

Client Work Pictures

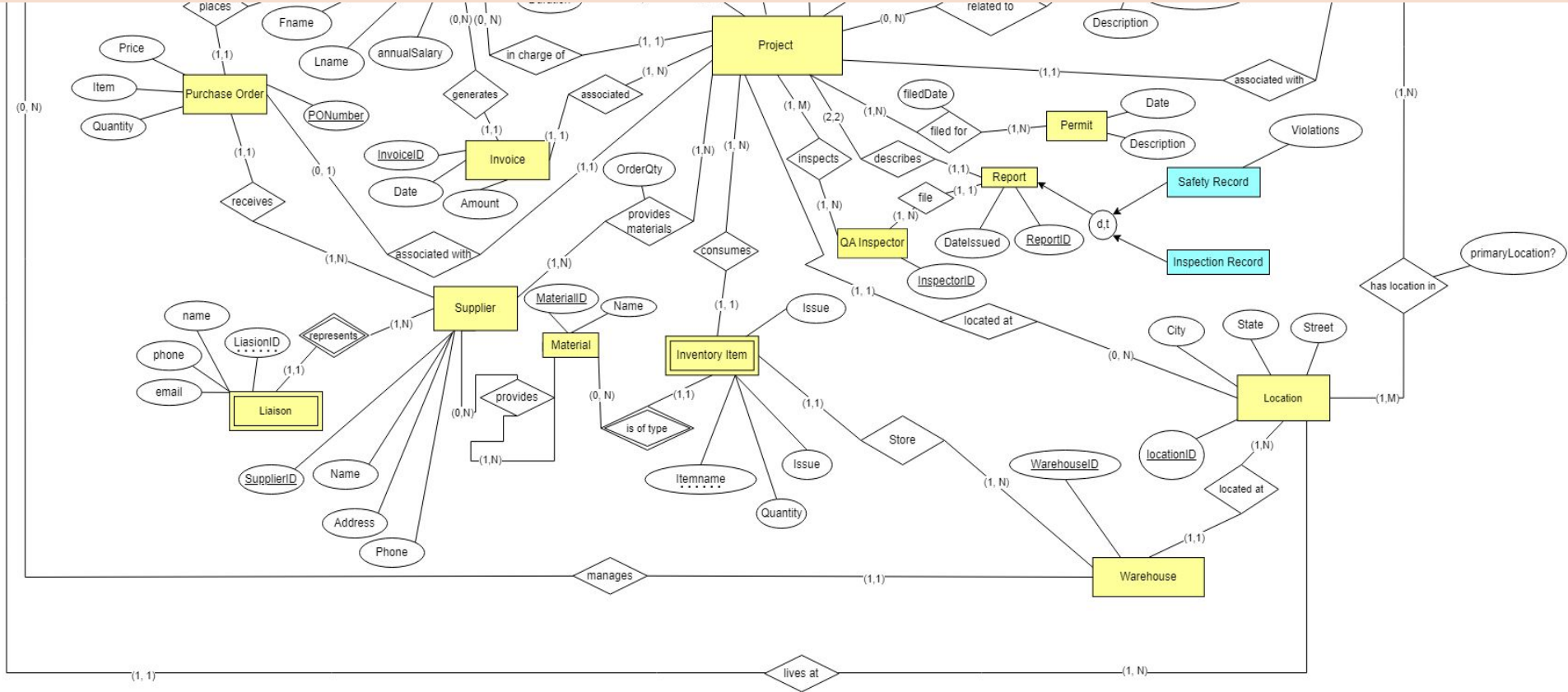
Final EER Diagram



Top Half of Diagram



Bottom Half of Diagram



Relational Schema

1. **Project**(PID, ProjectTypeName[4] , ClientID[2] , locationID[10], PONumber[17] , EmployeeID[3], StartDate)
2. **Client**(ClientID, Name, Address, Phone, Email, ClientImportance)
3. **Employee**(EmployeeID, locationID[10], FirstName, LastName, annualSalary)
   a. **Supervisor**(EmployeeID[3], SuperviseeID[3])
   b. **Machine_Operator**(EmployeeID[3])
   c. **Asphalt_Layer**(EmployeeID[3])
   d. **Salesperson**(EmployeeID[3], sellsProjectType)
4. **Project_Type**(ProjectTypeName, minJobSquareFootage)
   a. **Maintenance**(ProjectTypeName[4], MaintenanceFrequency)
   b. **Resurfacing**(ProjectTypeName[4], ResurfacingMaterials, Thickness)
   c. **Repair**(ProjectTypeName[4], Type, Technique)
   d. **Paving**(ProjectTypeName[4], Thickness, Width)

Relational Schema

5. **Feedback**(FeedbackID, EmployeeID[3], PID[1], Date, Description)
6. **Permit**(PermitID, Date, Description)
7. **Report**(ReportID, DateIssued, PID[1],inspectorID[31])
    a. **Safety_Record**(ReportID[7], Violations)
    b. **Inspection_Record**(ReportID[7])
8. **Material**(MaterialID, Name)
9. **Inventory_Item**(ItemName, isofTypeMaterialID[8], PID[1], storeAtWarehouseID[11], Quantity, Issue)
10. **Location**(locationID, Street, City, State)
11. **Warehouse**(WarehouseID, locationID[10], managedByEmpID[3])
12. **Supplier**(SupplierID, name, address, phone)
13. **Invoice**(InvoiceID, generatedByEmpID[3], associatedWithPID[1], Date, Amount)

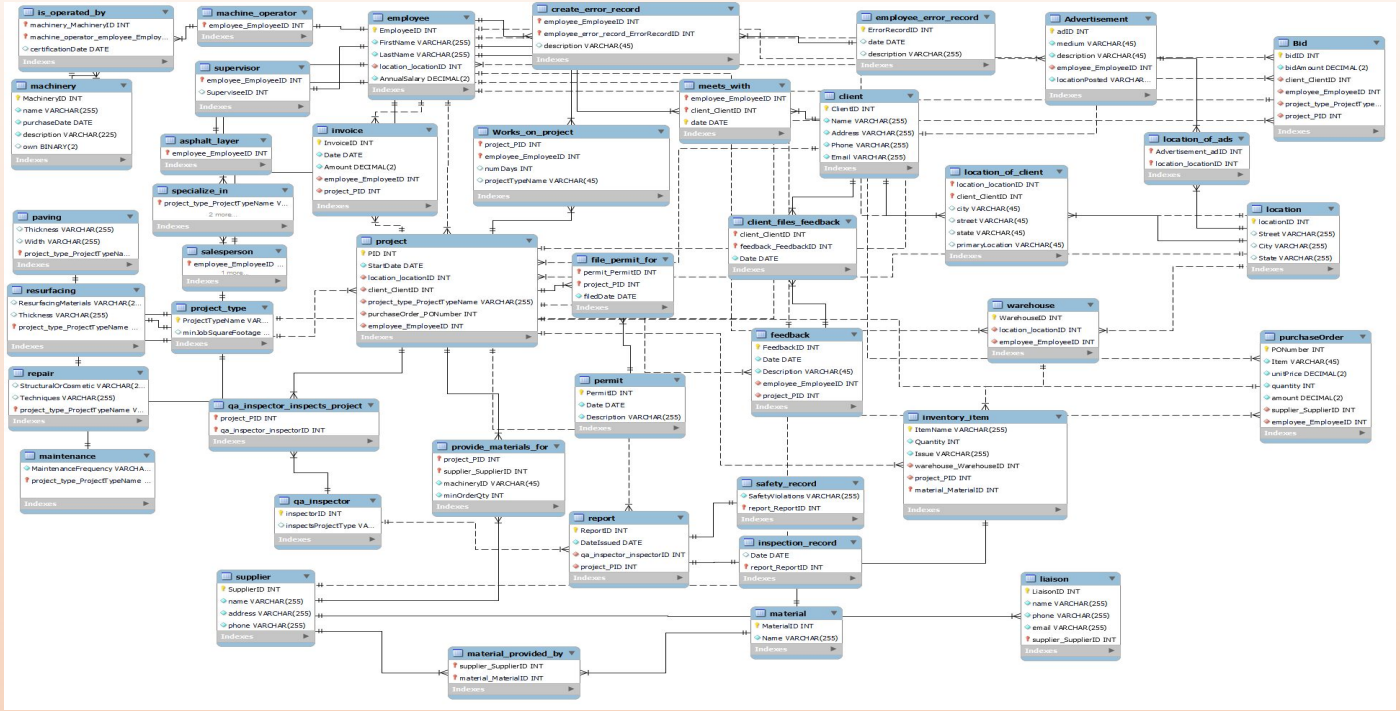
Relational Schema

14. **Bid**(BidID, bidAmount, overseeByEmpID[3], madeForProjTypeName[4], requestedByClientID[2], associatedWithPID[1])
15. **Advertisement**(AdID, generatedByEmpID[3], medium, description,locationPosted)
16. **Employee_Error_Record**(ErrorRecordID, date, description)
17. **PurchaseOrder**(PONumber, placedByEmpID[3], receivedBySupplierID[12], Item, Quantity, unitPrice)
18. **Liaison**(LiaisonID, representsSupplierID[12], name, phone, email)
19. **Machinery**(MachineryID, name, purchaseDate, description, ownOrRent)
20. **Is_Operated_By**(MachineryID[19], EmployeeID[3b], certificationDate)
21. **File_permit_for**(PID[1], PermitID[6], filedDate)
22. **Provide_materials_for** (PID[1], SupplierID[12], OrderQty)

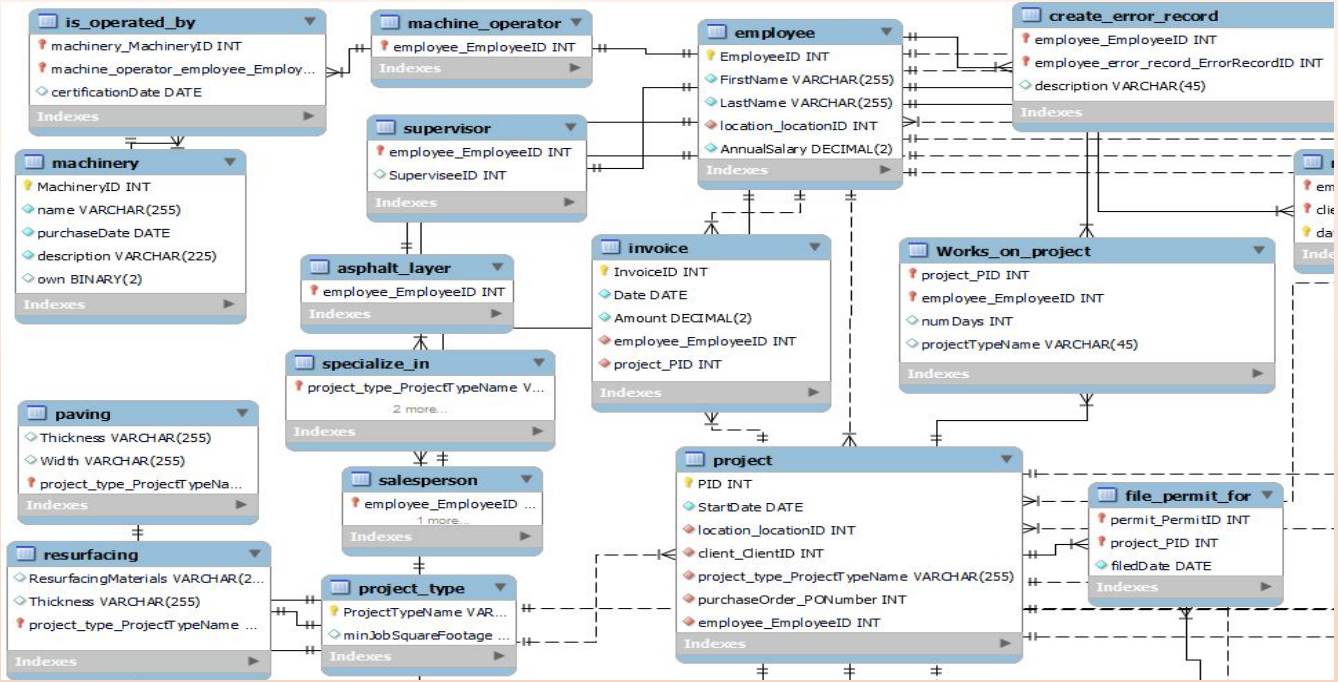
Relational Schema

23. **Work_on_project** (PID[1], EmployeeID[3], Duration, ProjectTypeName[4])
24. **Meets_with**(EmployeeID[3], ClientID[2], Date)
25. **Client_files_feedback**(ClientID[2], FeedbackID[5], Date)
26. **Create_error_record**(EmployeeID[3], ErrorRecordID[16], Description)
27. **Specialize_in**(EmployeeID[3], ProjectTypeName[4], date_of_certification)
28. **Material_provided_by**(MaterialID[8], SupplierID[12])
29. **Location_of_client**(ClientID[2], LocationID[10], City[11], Street[11], State[11], primaryLocation)
30. **Location_of_ads**(AdID[15], locationID[10])
31. **QA_Inspector**(InspectorID, inspectsProjectType[4])
32. **QA_Inspector_Inspects_Project**(InspectorID[32], PID[1])

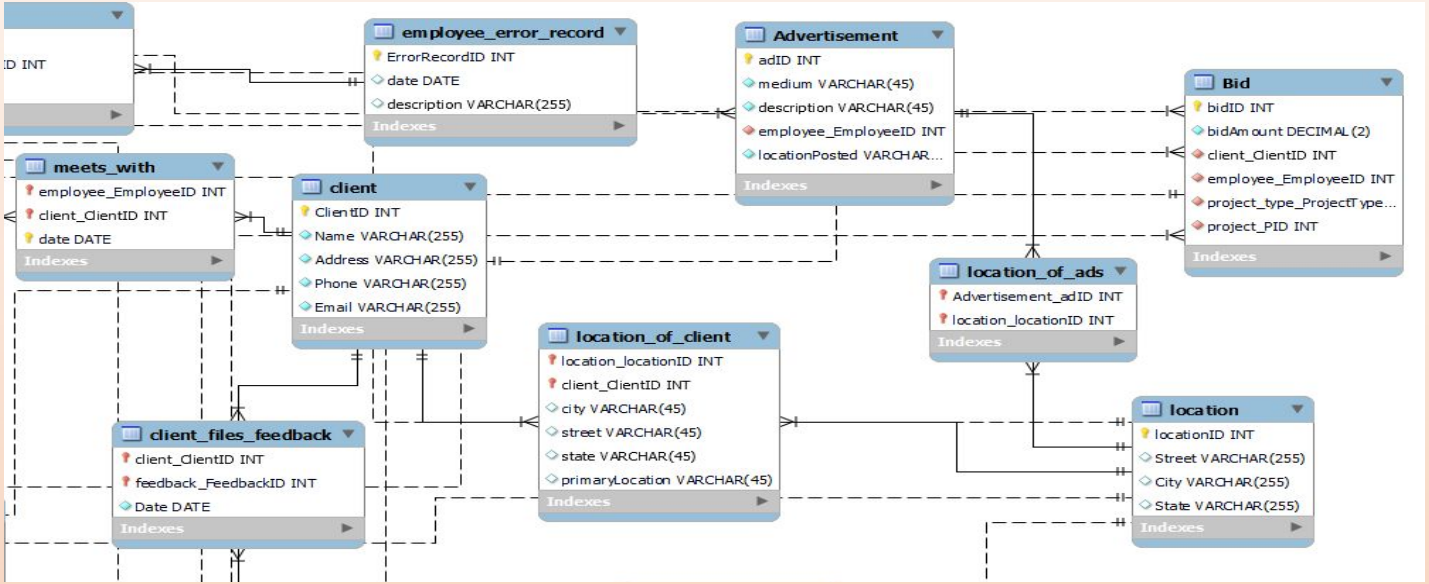
MySQL Implementation



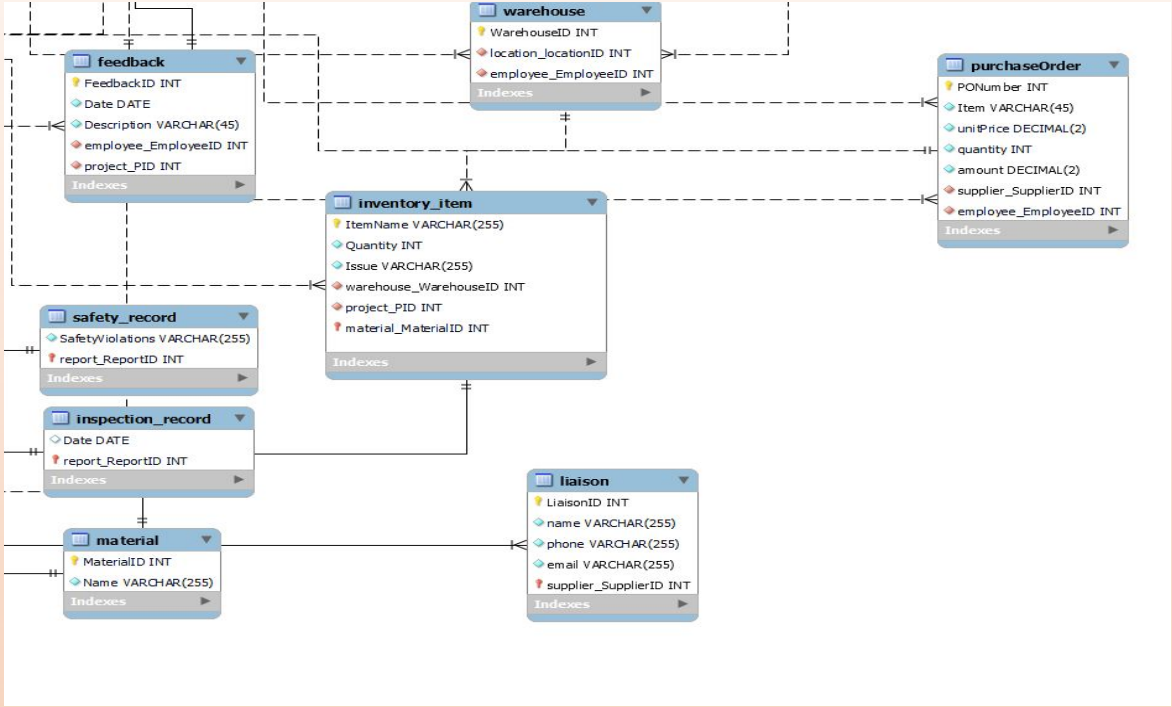
MySQL Implementation



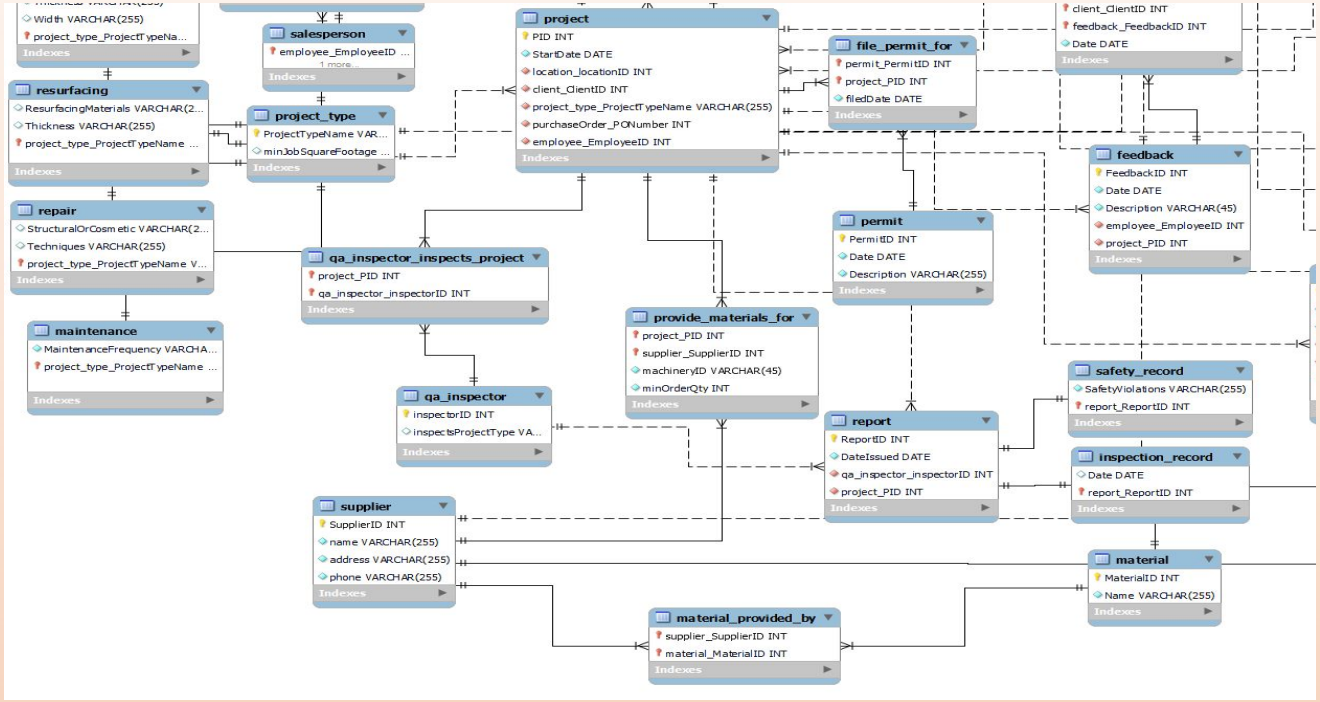
MySQL Implementation



MySQL Implementation



MySQL Implementation



Interesting Query 1 - Material Usage and Availability

Business justification:
Every MILP requires an objective function (maximizing or minimizing a function based off of your specific goal) subject to a set of constraints.

In reference to our client, our objective is to maximize the profit. This includes maximizing our revenue, while taking associated costs into account.

There are inventory constraints we can take into consideration regarding this objective function. We must ensure that the amount of material needed for each project doesn't exceed our current inventory.

Query 1 MySQL

Decision Variables for MILP:

Project Revenue

Project Cost

- MaterialCost
- EmployeeCost

Inventory Levels

- ItemName
- ProjectMaterialQuantity
- TotalInventory (total quantity of each item)

```sql
SELECT
    p.PID,
    # variables for project1 revenue
    INV.Amount,
    # variables for time constraints
    MAX(W.numDays),
    # variables for inventory contraints
    I.MaterialD,
    I.Quantity AS ProjectMaterialQuantity,
    (SELECT SUM(Quantity) FROM 115data.inventory_item WHERE MaterialD = I.MaterialD) AS TotalInventory
FROM
    115data.project1 P, 115data.invoice INV, 115data.inventory_item I,
    115data.employee E, 115data.client C, 115data.work_on_project W

WHERE
    P.PID = I.PID AND P.EmployeeID = E.EmployeeID
    AND P.ClientID = C.ClientID AND INV.associatedWithPID = P.PID AND W.PID = P.PID
GROUP BY
    P.PID, INV.Amount, I.MaterialD, ProjectMaterialQuantity;
```

Results of Query 1

PID	Amount	MAX(W.numDays)	MaterialID	ProjectMaterialQuantity	TotalInventory
2710	3729	36	10	60	342
8035	3219	25	91	22	254
1579	2344	39	67	52	133
2729	3115	35	69	11	161
7708	3770	38	91	43	254
7323	1898	31	18	47	105
9792	1405	35	43	76	138
8297	3443	30	24	12	117
8983	645	14	18	14	105
6267	3154	25	10	71	342
4763	2879	34	91	63	254
5999	2483	38	91	38	254
6279	2824	22	69	61	161
1346	754	32	91	33	254
5671	3952	18	10	34	342
6259	411	24	69	10	161
6167	935	31	10	73	342
5700	1709	37	67	71	133
3684	3651	31	91	42	254
6048	2452	17	67	10	133
9407	1763	37	24	30	117
4747	2742	25	10	66	342
1341	2743	38	43	62	138
4300	417	21	91	13	254
9506	349	23	18	44	105

We are trying to maximize the profit we can make producing asphalt paving project X and asphalt parking lot repair project Y. Project X makes an estimated profit of 300 dollars, while Project Y makes a profit of 500. Each Project X site requires 69 lbs of materials, and each Project Y site requires 36 lbs. We have a total material inventory of 342 lbs for X and 254 lbs for Y, and a max amount of 10 projects per year. To solve this problem, we use the formulas shown above to maximize profit. Let's consider that revenue and cost are not necessarily linear and cost varies with the volume of material used to build x and y. We can say that the cost of putting material on each parking lot is proportional to the square of the volume of materials used.

$$\text{Material Cost} = V_X^2 \cdot X + V_Y^2 \cdot Y$$

$$\text{Maximize } P_X \cdot X + P_Y \cdot Y - (\text{Material Cost})$$

$$X \cdot \text{material\_required\_X} \leq \text{material\_inventory\_X}$$
$$Y \cdot \text{material\_required\_Y} \leq \text{material\_inventory\_Y}$$
$$X + Y \leq \text{max\_projects\_per\_year}$$

*Note: numbers are averaged queried synthetic data.

MILP Integration

Using this objective function and the specified constraints, we can input this MLP into Gurobi to obtain the optimal solution for these decision variables. This will maximize our objective (Profit) and will satisfy all specified constraints.

Ran with Gurobi, replacing the quadratic terms in the objective function with piecewise-linear approximations. This allows us to use Gurobi to solve the MILP problem.

```python
import gurobipy as gp
from gurobipy import GRB

# Create a model
model = gp.Model("asphalt_projects_optimization")

# Constants
P_X = 300  # Profit for asphalt paving project X
P_Y = 500  # Profit for asphalt parking lot repair project Y
material_inventory_X = 342  # Total material inventory for X
material_inventory_Y = 254  # Total material inventory for Y
material_required_X = 69  # Material units required for one unit of project X
material_required_Y = 36  # Material units required for one unit of project Y
max_projects_per_year = 10  # Maximum number of projects (X and Y combined) per year

# Variables
X = model.addVar(vtype=GRB.INTEGER, name="X")
Y = model.addVar(vtype=GRB.INTEGER, name="Y")
V_X = model.addVar(lb=0.0, name="V_X")
V_Y = model.addVar(lb=0.0, name="V_Y")

# Objective function
profit = P_X * X + P_Y * Y
cost = V_X**2 + V_Y**2  # Removed coefficients a and b
model.setObjective(profit - cost, GRB.MAXIMIZE)

# Constraints
model.addConstr(X * material_required_X <= material_inventory_X, "material_constraint_X")
model.addConstr(Y * material_required_Y <= material_inventory_Y, "material_constraint_Y")
model.addConstr(X + Y <= max_projects_per_year, "max_projects_per_year")

# Solve the model
model.optimize()

# Print the results
print("Optimal value:", model.objVal)
print("Optimal X:", X.x)
print("Optimal Y:", Y.x)
print("Optimal V_X:", V_X.x)
print("Optimal V_Y:", V_Y.x)
```

Key Query Takeaways

- According to the results below, the optimal production plan to maximize profit involves producing 3 of Project X and 7 of Project Y over an entire year, with no need for additional materials. The total profit in this optimal scenario is $4700.00.
- Allows the business to prioritize businesses relatively objectively and minimizes negative impact of shortages
- Must be done regularly to ensure on-time project completion

```
Optimal solution found (tolerance 1.00e-04)
Best objective 4.400000000000e+03, best bound 4.400000000000e+03, gap 0.0000%
Optimal value: 4400.0
Optimal X: 3.0
Optimal Y: 7.0
Optimal V_X: 0.0
Optimal V_Y: 0.0
```

Interesting Query 2 - Predicting Cost Overrun

Problem

The projects taken on by Asphalt Construction Co., like other real-world projects, face the risk of going **over-budget** due to fluctuating material costs or unexpected weather conditions. If a project goes beyond a certain threshold that a client did not expect, it can lead to frustration and losing their confidence in our company.

Approaches

To prevent this from happening, our team decided to take 2 approaches: a) to **predict a project's final cost**, and b) to **predict whether a project will go 50% over** its original agreed-upon price or not. For the first subquery, we trained a **Random Forest** and a **Gradient Booster**. For the second, we trained a **Decision Tree** (for interpretability) and a **Random Forest Classifier**, with the main features being the bid amount (the original agreed-upon price) and the type of project.

Interesting Query 2 - Predicting Cost Overrun

```
SELECT bid.associatedWithPID AS PID,
    MAX(bid.bidAmount) AS bidAmount,
        bid.madeForProjTypeName,
        invoice_totals.TotalCost
FROM bid
JOIN (
    SELECT associatedWithPID, SUM(Amount) AS TotalCost
    FROM invoice
    GROUP BY associatedWithPID
) AS invoice_totals ON bid.associatedWithPID = invoice_totals.associatedWithPID
GROUP BY bid.associatedWithPID, bid.madeForProjTypeName;
```

PID	bidAmount	madeForProjTypeName	TotalCost
2710	3956	Maintanence	4354
8035	2856	Paving	3219
1579	2499	Repair	2892
2729	2249	Resurfacing	3115
7708	1640	Paving	3770
7323	1572	Resurfacing	1898
9792	2876	Paving	4102
8297	1429	Resurfacing	3443
8983	1140	Resurfacing	1632
6267	3018	Repair	3154
4763	1792	Repair	2879
5999	1619	Resurfacing	2483
6279	2153	Repair	2824
1346	3009	Paving	3271
5671	440	Resurfacing	3952
6259	1050	Paving	1321
6167	1212	Repair	1562
5700	3585	Maintanence	5371

Implementation of the Models

```
from sklearn.ensemble import GradientBoostingRegressor

grid_values = {'n_estimators': np.linspace(100, 5*1000, 5, dtype='int32'),
               'max_leaf_nodes': np.linspace(2, 10, 9, dtype='int32')
               }

gbr = (GridSearchCV(GradientBoostingRegressor(random_state=42), grid_values).
       fit(X_train, y_train).best_estimator_)
```

```
comparison_data = {'Gradient Boosting': [OSR2(gbr, X_test, y_test, y_train),
                                         mean_squared_error(y_test, gbr.predict(X_test)),
                                         mean_absolute_error(y_test, gbr.predict(X_test))]
                   }
comparison_table = pd.DataFrame(data=comparison_data, index=['OSR2', 'MSE', 'MAE']).transpose()
(comparison_table.style.set_properties(
    **{'font-size': '12pt',}).
 set_table_styles(
     [{'selector': 'th', 'props': [('font-size', '10pt')]}])
)
comparison_table.style.background_gradient(low=0, high=1, cmap="viridis", axis=1)
```

Gradient Boosting Regressor

We used sklearn's implementation. The number of estimators and the maximum number of leaf nodes were cross-validated with 5 folds.

```
grid_values = {'max_features': np.arange(1, 5),
               'min_samples_leaf': [5],
               'n_estimators': [500],
               'random_state': [42]}

rf_cv = GridSearchCV(RandomForestRegressor(), param_grid=grid_values, cv=5).fit(X_train, y_train)
rf_cv = rf_cv.best_estimator_
rf_y_pred = rf_cv.predict(X_test)
```

```
comparison_data = {'Gradient Boosting': [OSR2(gbr, X_test, y_test, y_train),
                                         mean_squared_error(y_test, gbr.predict(X_test)),
                                         mean_absolute_error(y_test, gbr.predict(X_test))],
                   'Random Forest': [OSR2(rf_cv, X_test, y_test, y_train),
                                         mean_squared_error(y_test, rf_cv.predict(X_test)),
                                         mean_absolute_error(y_test, rf_cv.predict(X_test))]
                   }
comparison_table = pd.DataFrame(data=comparison_data, index=['OSR2', 'MSE', 'MAE']).transpose()
(comparison_table.style.set_properties(
    **{'font-size': '12pt',}).
 set_table_styles(
     [{'selector': 'th', 'props': [('font-size', '10pt')]}])
)
comparison_table.style.background_gradient(low=0, high=1, cmap="viridis", axis=1)
```

Random Forest Regressor

We also used sklearn's implementation. Cross-validation of the the feature subset size was conducted.

Implementation of the Models

```
grid_values = {'ccp_alpha': np.linspace(0, 0.01, 11)}

dtc = DecisionTreeClassifier(random_state=42, min_samples_leaf=2)
dtc_cv = GridSearchCV(dtc, param_grid=grid_values, cv=5, error_score='raise').fit(X_train, y_train)
plt.figure(figsize=(10,10))
plot_tree(dtc_cv.best_estimator_,
          feature_names=X_train.columns,
          class_names=['0','1'],
          filled=True,
          impurity=False,
          fontsize=12)

dtc = dtc_cv.best_estimator_
dtc_y_pred = dtc.predict(X_test)
dtc_acc = np.mean(y_test == dtc_y_pred)
cm = confusion_matrix(y_test, dtc_y_pred)
tn, fp, fn, tp = cm.ravel()
dtc_TPR = tp / (tp + fn)
dtc_FPR = fp / (fp + tn)
dtc_PRE = tp / (tp + fp)
print ("Confusion Matrix: \n", cm)
print ("\nAccuracy:", dtc_acc)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = [False, True])

cm_display.plot()
plt.show()

acc, tpr, fpr, pre, f1 = basic_metrics(dtc_y_pred, y_test)
comparison_data = {'Decision Tree': [acc, tpr, fpr, pre, f1]
                   }

comparison_table = pd.DataFrame(data=comparison_data, index=['Accuracy', 'TPR', 'FPR','PRE', 'F1 Score']).transpose()
comparison_table.style.set_properties(**{'font-size': '12pt',}).set_table_styles([{'selector': 'th', 'props': [('font-size', '10pt')]}])
comparison_table.style.background_gradient(low=0, high=1, cmap="viridis", axis=1)
```

Decision Tree Classifier

We used sklearn's implementation of decision trees. Cross-validation of the pruning parameter was conducted.

```
grid_values = {'max_features': np.arange(1, 5),
               'min_samples_leaf': [5],
               'n_estimators': [500],
               'random_state': [42]}

rf_cv = GridSearchCV(RandomForestClassifier(), param_grid=grid_values, cv=5).fit(X_train, y_train)
rf_cv = rf_cv.best_estimator_
rf_y_pred = rf_cv.predict(X_test)
cm = confusion_matrix(y_test, np.array(rf_y_pred))
tn, fp, fn, tp = cm.ravel()
rf_cv_TPR = tp / (tp + fn) if (tp + fn) > 0 else 0
rf_cv_FPR = fp / (fp + tn) if (fp + tn) > 0 else 0
rf_cv_PRE = tp / (tp + fp) if (tp + fp) > 0 else 0
rf_cv_acc = np.mean(y_test == rf_y_pred)
print ("Confusion Matrix: \n", cm)
print ("\nAccuracy:", rf_cv_acc)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = [False, True])

cm_display.plot()
plt.show()

acc, tpr, fpr, pre, f1 = basic_metrics(rf_y_pred, y_test)
comparison_data = {'Random Forest': [acc, tpr, fpr, pre, f1]
                   }

comparison_table = pd.DataFrame(data=comparison_data, index=['Accuracy', 'TPR', 'FPR','PRE', 'F1 Score']).transpose()
comparison_table.style.set_properties(**{'font-size': '12pt',}).set_table_styles([{'selector': 'th', 'props': [('font-size', '10pt')]}])
comparison_table.style.background_gradient(low=0, high=1, cmap="viridis", axis=1)
```

Random Forest Classifier

We also used sklearn's implementation. Cluster-then-predict methodology was used. Cross-validation of the the feature subset size was conducted.

Results: Random Forest and Gradient Boosting Regressors

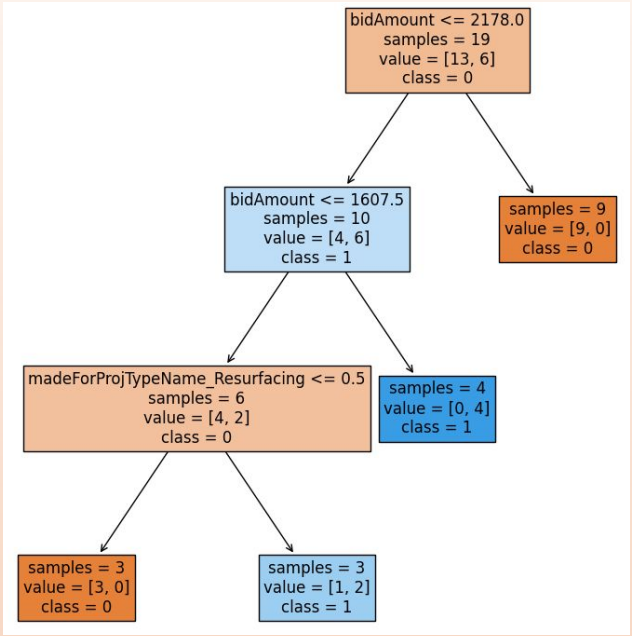
	Feature	Importance score (GBR)
0	bidAmount	92.7
1	madeForProjTypeName_Maintanence	0.3
2	madeForProjTypeName_Paving	0.2
3	madeForProjTypeName_Repair	0.8
4	madeForProjTypeName_Resurfacing	6.1

	Feature	Importance score (RFR)
0	bidAmount	90.5
1	madeForProjTypeName_Maintanence	0.0
2	madeForProjTypeName_Paving	7.8
3	madeForProjTypeName_Repair	0.0
4	madeForProjTypeName_Resurfacing	1.7

	OSR2	MSE	MAE
Gradient Boosting	0.638222	443663.988610	678.024231
Random Forest	0.618796	470741.276189	696.736328

Both the random forest (RFR) and gradient boosting (GBR) regressors agree that **bid amount**, **resurfacing** and **paving** is a significant feature. However, we stress that due to the **small dataset**, the models might **not have captured the intricate relationships among all the variables**, which resulted in **relatively low prediction quality**.

Results of the Decision Tree (Predicting Final Cost > 150%)



	Accuracy	TPR	FPR	PRE	F1 Score
Decision Tree	0.700000	0.000000	0.222222	0.000000	0.741176

The decision tree provides thresholds for original **bid amounts** to classify budget overruns. **Resurfacing** projects also appears to be a feature of importance. Interestingly, higher original bid amounts are predicted to stay within budget, likely due to its “50%” threshold being higher.

Results of the Random Forest (Predicting Final Cost > 150%)

The random forest identified important features and was able to rank them. It appears that **bid amount** is (naturally) a big factor, and the fact that a project is **paving** or **resurfacing** is also significant. Its **accuracy is also better** than the decision trees'.

	Feature	Importance score
0	bidAmount	70.8
1	madeForProjTypeName_Maintanence	0.0
2	madeForProjTypeName_Paving	22.6
3	madeForProjTypeName_Repair	0.0
4	madeForProjTypeName_Resurfacing	6.6

	Accuracy	TPR	FPR	PRE	F1 Score
Random Forest	0.900000	0.000000	0.000000	0.000000	0.852632

Normalization

VIOLATES 1NF:

Client(ClientID, ClientLName, Address, Phone, Email)

If a client has more than 1 address (perhaps they own multiple houses), 1NF is violated. Another way this relation violates 1NF is when multiple phone numbers are given for the same client, so if there are backup forms of contact given for the project in case someone is not reachable (e.g., a home and an office phone) then this violates the 1NF rule of all attributes being single-valued.

Client(ClientID, ClientLName, Email)
Addresses(ClientID, Address)
PhoneNumbers(ClientID, Phone)

VIOLATES 2NF:

If Employee and Project_Type relations were composed in the following ways:

Employee(EmployeeID, TeamID, FirstName, LastName, annualSalary, TeamLocation)
Project_Type(ProjectTypeName, ProjectCategory, minJobSquareFootage, CategorySupervisor)

Since non-prime attributes of each relation are functionally dependent on a subset of candidate key, (EmployeeID, TeamID) and (ProjectTypeName, ProjectCategory), it would result in partial dependencies, thus violating 2NF.

VIOLATES 2NF (Continued)

Employee(EmployeeID, FirstName, LastName, annualSalary)

a. **Supervisor**(EmployeeID3, SuperviseeID3)
b. **Machine_Operator**(EmployeeID3)
c. **Asphalt_Layer**(EmployeeID3)
d. **Salesperson**(EmployeeID3, sellsProjectType4)

Project_Type(ProjectTypeName, minJobSquareFootage)

a. **Maintenance**(ProjectTypeName4, MaintenanceFrequency)
b. **Resurfacing**(ProjectTypeName4, ResurfacingMaterials, Thickness)
c. **Repair**(ProjectTypeName4, Type, Technique)
d. **Paving**(ProjectTypeName4, Thickness, Width)

The relations, Employee and Project_Type, ensure adherence to 2NF by linking each sub-attribute set to a specific employee / project type. This makes their non-prime attributes fully dependent on a single, unique identifier, avoiding any partial dependencies that would violate 2NF.

VIOLATES 3NF:

Project(PID, ProjectTypeName[4] , ClientID[2] , ClientLName, locationID[10], PONumber[17] , EmployeeID[3], EmployeeLName, StartDate)

Currently, this does violate 3NF and we created the normalization for the relation above. Multiple non-prime keys have a dependency on each other. However, we already have the relations of ProjectType as number 4 on the list of relations (Client is number 2, Employee is 3, and Location is 10). This normalization is shown in our finalized schema, but looking at this initial Project relation shows our normalization process following the 3NF rules.

Project(PID, StartDate)
ProjectHasPO(PID, PONumber)
ProjectIsType(PID, ProjectType)
Client (ClientID, ClientLName)
Employee(EmployeeID, EmployeeLName)
Location(locationID, City, Street, State)

VIOLATES BCNF:

EmployeeSalary(EmployeeID, annualSalary, SSN)

To violate BCNF, we modified a relation to capture the salary of employees (for social security contribution purposes). Since every person has a unique SSN, it follows that an SSN functionally determines EmployeeID. To follow BCNF, the SSN is broken off into its own relation, which removes all non-prime attributes that determine the candidate key (EmployeeID).

EmployeeSalary(EmployeeID, annualSalary)
EmployeeSSN(EmployeeID, SSN)

Reflection

Key Successes:

- 1 - on - 1 communication with client
- Building out visual diagrams and code for project needs
- In-depth analysis of customer business model and needs
- Customer approval of helpful queries that would expand on their business propositions and inventory analysis

Key Improvements for Future Work:

- Greater understanding of customer work through an in-person visit
- Meet customer and other workers to watch their usability of our database to create further changes
- Implementing our curated queries during an in-person visit in order to compare simulated vs. actual results

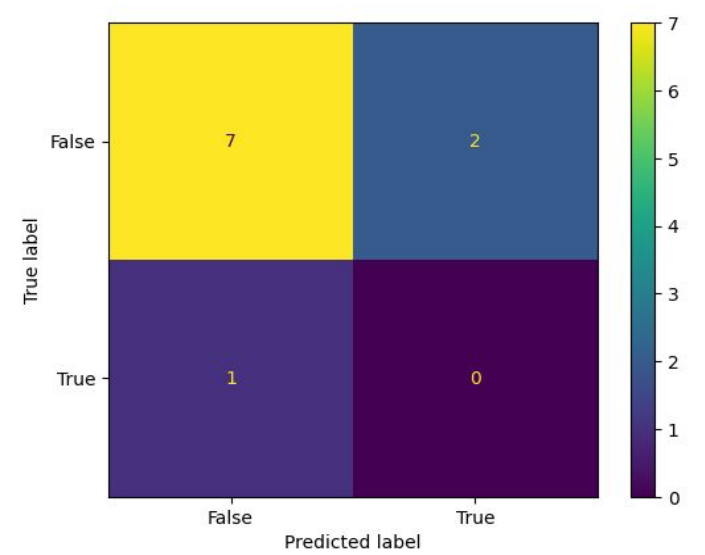
Next Steps

- Further development of beneficial interesting queries for the client
  - Safety: Conduct train-test split and use feature engineering to determine the weights of safety features present in the records
- The ability to log any challenges faced during the construction process to learn and evaluate for the next project
  - Implementing a feedback system to understand the drawbacks of our queries
- Focus on collecting real data from the Asphalt Construction Company to better build our models and utilize their data
- Adding more information about client-specific characteristics
  - This company is family-oriented and customer-based, take these personalizations into account

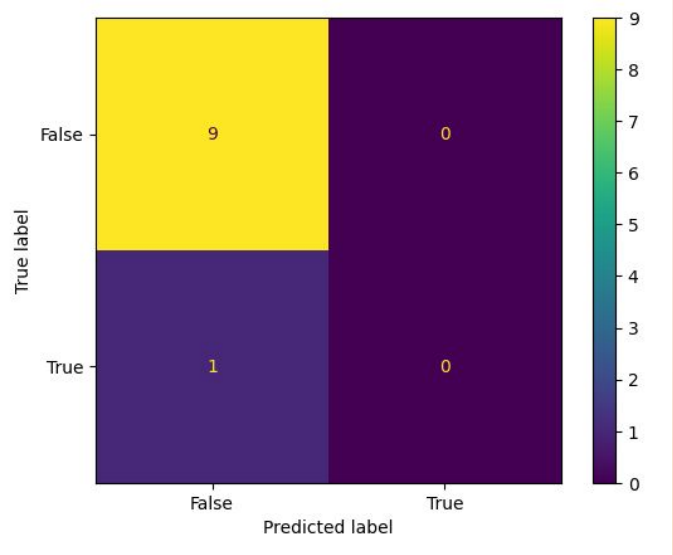
Thank You!

Any Questions?

Appendix A: Confusion Matrices



Confusion Matrix for the Decision Tree



Confusion Matrix for the Random Forest