Stage IV - Elaboration: Database Design
Team 01-01

Demonstrate that all the relations in the relational schema are normalized to Boyce–Codd normal form (BCNF).
• For each table, specify whether it is in BCNF or not, and explain why.
• For each table that is not in BCNF, show the complete process that normalizes it to BCNF.

Our tables:

METER_INFO: propertyName, portfolioManagerId, portfolioManagerMeterId, meterName, meterType, units, measurementMethod, includedInPropertyMetricsYn, dateActive, dateInactive, dateFirstEntry, dateLastEntry, aggregateYn, rateCode, rateCodeDescription

METER_INFO is in BCNF because it has the primary key meterName and every other attribute is functionally dependent on it.

CUSTOM_METER_INFO: meterName, customIdName, customIdValue, priority

CUSTOM_METER_INFO is in BCNF because it has the primary key meterName and every other attribute is functionally dependent on it.

METER_ENTRY: meterName, meterConsumptionId, month, year, usage, ~~usageUnits~~, cost, estimationYn, lastModifiedDate, lastModifiedBy

METER_ENTRY was not originally in BCNF because one of our attributes, usageUnits, is functionally dependent on meterName while every other attribute is functionally dependent on the primary key meterConsumptionId. To fix this, we removed usageUnits from the relation, since it already exists in METER_INFO.

GREEN_POWER_INFO: meterConsumptionId, quantity, biogasPercent, geothermalPercent, hydropowerPercent, solarPercent, windPercent, unknownPercent, ~~generationFacilityLocation, generationFacilityId, generationFacilityName, egridSubRegion,~~ ***generationLocationId***

GREEN_POWER_INFO was not originally in BCNF because certain values were dependent on generationFacilityId. After removing those attributes and moving

them to their own entity, it is in BCNF because it has the primary key meterConsumptionId and every other attribute is functionally dependent on it.

RENEWABLE_METER_INFO: meterConsumptionId, usedOnSiteYn, usedOnSiteUnits, exportedOffSiteYn, exportedOffSiteUnits, cost, recOwnership, ~~generationLocation, egridSubregion, generationFacilityLocation, generationFacilityId~~, ***generationLocationId***

RENEWABLE_METER_INFO was not originally in BCNF because certain values were dependent on generationFacilityId. After removing those attributes and moving them to their own entity, it is in BCNF because it has the primary key meterConsumptionId and every other attribute is functionally dependent on it.

GENERATION_FACILITY: generationFacilityId, generationLocation, egridSubregion, generationFacilityLocation

GENERATION_FACILITY is a new strong entity. It is in BCNF because it has the primary key generationFacilityId and every other attribute is functionally dependent on it.

DISPOSED_WASTE_METER_INFO: meterConsumptionId, landfillPercent, incinerationPercent, incinerationPercent, unknownPercent

DISPOSED_WASTE_METER_INFO is in BCNF because it has the primary key meterConsumptionId and every other attribute is functionally dependent on it.

Define the different views (virtual tables) required. For each view list the data and transaction requirements. Give a few examples of queries, in English, to illustrate.

1) This view gets the average cost/usage of every month for each individual meter. It can be used by queries to obtain information about which meters cost the most/least per month.

```
CREATE VIEW MeterMonthlyCostAvg
SELECT METER_INFO.meterName, METER_INFO.meterType,
METER_INFO.meterUnits, METER_ENTRY.month, AVG(METER_ENTRY.usage) as
averageUsage, AVG(METER_ENTRY.cost) as averageCost
FROM METER_ENTRY LEFT JOIN METER_INFO ON METER_ENTRY.meterName =
METER_INFO.meterName
GROUP BY METER_INFO.meterName, METER_ENTRY.month;
```

2) This view gets the average cost/usage of every month for each type of energy. It can be used by queries to obtain information about which energy sources cost the most/least per month.

```
CREATE VIEW TypeMonthlyCostAvg
SELECT METER_INFO.meterType, METER_INFO.meterUnits, METER_ENTRY.month,
AVG(METER_ENTRY.usage) as averageUsage, AVG(METER_ENTRY.cost) as
averageCost
FROM METER_ENTRY LEFT JOIN METER_INFO ON METER_ENTRY.meterName =
METER_INFO.meterName
GROUP BY METER_INFO.meterType, METER_ENTRY.month;
```

3) *This view gets the average total cost of every month. It can be used by queries to obtain information about which months are the most expensive.*

```
CREATE VIEW OverallMonthlyCostAvg
SELECT month, AVG(cost) as averageCost
FROM METER_ENTRY
GROUP BY month;
```

Design a complete set of SQL queries to satisfy the transaction requirements identified in the previous stages, using the relational schema and views defined in earlier tasks.

1) Get average total cost of every month

```
SELECT *
FROM OverallMonthlyCostAvg;
```

2) Get month with the highest average total cost

```
SELECT month, MAX(averageCost)
FROM OverallMonthlyCostAvg;
```

3) Get month with the lowest average total cost

```
SELECT month, MIN(averageCost)
FROM OverallMonthlyCostAvg;
```

4) Get average monthly cost of every meter for every month

```
SELECT *
FROM MeterMonthlyCostAvg
ORDER BY meterName, month;
```

5) Get the month that has the highest monthly average cost for each meter

```
SELECT meterName, meterType, month, MAX(averageCost)
FROM MeterMonthlyCostAvg
GROUP BY meterName
ORDER BY meterName;
```

6) Get the month that has the lowest monthly average cost for each meter

```
SELECT meterName, meterType, month, MIN(averageCost) FROM
MeterMonthlyCostAvg GROUP BY meterName ORDER BY meterName;
```

7) Get average monthly cost of every meter type for every month

```
SELECT *
FROM TypeMonthlyCostAvg
ORDER BY meterType, month;
```

8) Get the month that has the highest monthly average cost for each meter type

```
SELECT meterType, month, MAX(averageCost)
FROM TypeMonthlyCostAvg
GROUP BY meterType
ORDER BY meterType;
```

9) Get the month that has the lowest monthly average cost for each meter

```
SELECT meterType, month, MIN(averageCost)
FROM TypeMonthlyCostAvg
GROUP BY meterType
ORDER BY meterType;
```