# Day 11: Electric Power Grid & Smart Metering

**Domain:** Energy / Utilities / Grid Operations

**Difficulty:** Expert+ — time-series data, anomaly detection, network relations, outages, maintenance, billing, demand response.

**Goal:** Analyze consumption patterns, detect missing/erroneous meter data, correlate outages with maintenance and grid segments, compute peak loads, and produce billing-ready aggregates.

---

## Scenario description

A regional utility operates power generation and distribution across multiple substations and feeders. Customers have smart meters that report interval readings (15-minute granularity). The utility must:

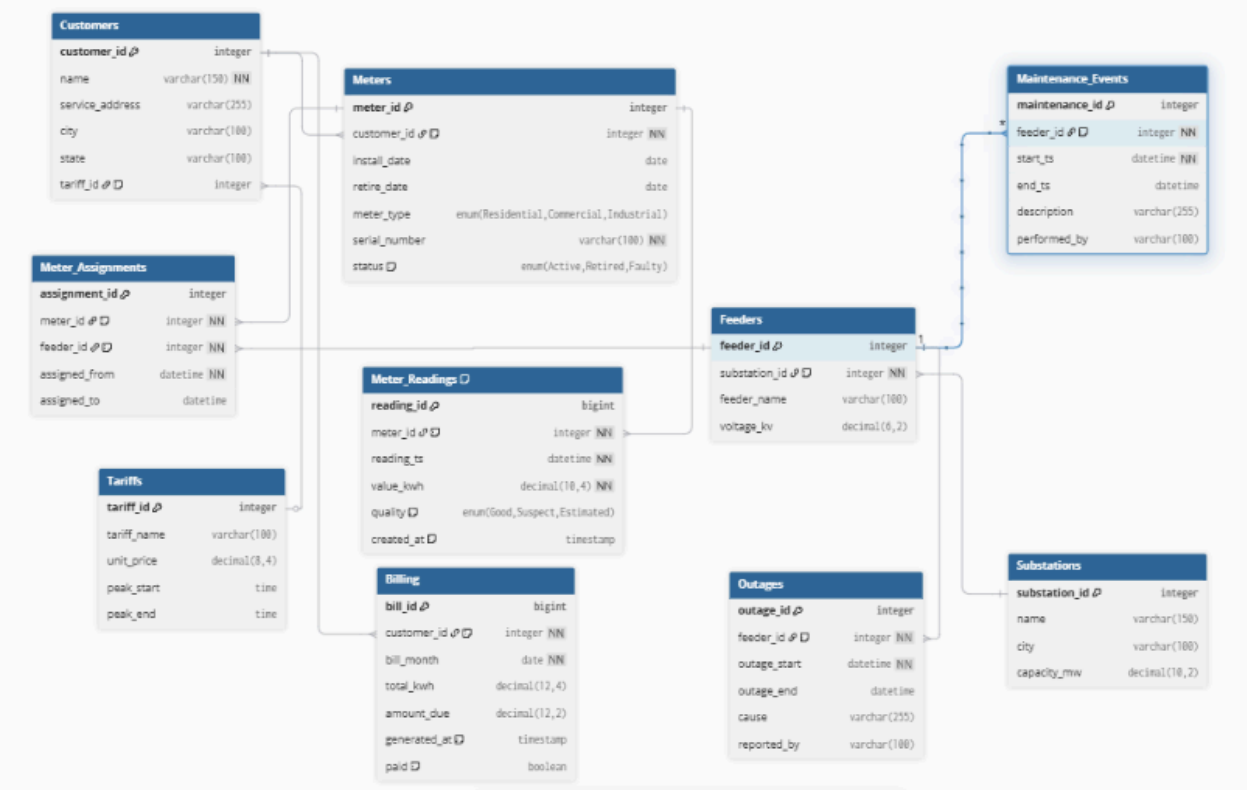
- Compute accurate consumption and billing.
- Detect missing or duplicate readings and meter tampering.
- Correlate outages and maintenance events with customer impact.
- Identify feeders/substations with chronic reliability problems (high outage minutes).
- Support demand response and market participation by computing peak loads and load factor.

---

## Database schema (MySQL)

> All DDL uses MySQL 8+ features (CTEs, window functions). Add suggested indexes after schema.

https://dbdiagram.io/d/69059d486735e11170be933c




## 1. Customers

```
CREATE TABLE Customers (
  customer_id INT PRIMARY KEY,
  name VARCHAR(150) NOT NULL,
  service_address VARCHAR(255),
  city VARCHAR(100),
  state VARCHAR(100),
  tariff_id INT
);
```

## 2. Meters

```
CREATE TABLE Meters (
  meter_id INT PRIMARY KEY,
  customer_id INT NOT NULL,
```

```
  install_date DATE,
  retire_date DATE NULL,
  meter_type ENUM('Residential','Commercial','Industrial'),
  serial_number VARCHAR(100) UNIQUE,
  status ENUM('Active','Retired','Faulty') DEFAULT 'Active',
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

### 3. Substations

```
CREATE TABLE Substations (
  substation_id INT PRIMARY KEY,
  name VARCHAR(150),
  city VARCHAR(100),
  capacity_mw DECIMAL(10,2)
);
```

### 4. Feeders

```
CREATE TABLE Feeders (
  feeder_id INT PRIMARY KEY,
  substation_id INT NOT NULL,
  feeder_name VARCHAR(100),
  voltage_kv DECIMAL(6,2),
  FOREIGN KEY (substation_id) REFERENCES Substations(substation_id)
);
```

### 5. Meter_Assignments

(Which meter is served by which feeder — can change over time.)

```
CREATE TABLE Meter_Assignments (
  assignment_id INT PRIMARY KEY,
  meter_id INT NOT NULL,
  feeder_id INT NOT NULL,
  assigned_from DATETIME NOT NULL,
  assigned_to DATETIME NULL,
  FOREIGN KEY (meter_id) REFERENCES Meters(meter_id),
  FOREIGN KEY (feeder_id) REFERENCES Feeders(feeder_id)
);
```

## 6. Meter_Readings

(Interval readings: `reading_ts` is the end of interval; `value_kwh` energy consumed during the interval.)

```
CREATE TABLE Meter_Readings (
  reading_id BIGINT PRIMARY KEY AUTO_INCREMENT,
  meter_id INT NOT NULL,
  reading_ts DATETIME NOT NULL,
  value_kwh DECIMAL(10,4) NOT NULL,    -- consumption in kWh for the interv
al
  quality ENUM('Good','Suspect','Estimated') DEFAULT 'Good',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (meter_id) REFERENCES Meters(meter_id)
);
-- IMPORTANT: add index on (meter_id, reading_ts) and partitioning by RANG
E on reading_ts for production
```

## 7. Outages

```
CREATE TABLE Outages (
  outage_id INT PRIMARY KEY,
  feeder_id INT NOT NULL,
```

```
  outage_start DATETIME NOT NULL,
  outage_end DATETIME NULL,  -- NULL for ongoing
  cause VARCHAR(255),
  reported_by VARCHAR(100),
  FOREIGN KEY (feeder_id) REFERENCES Feeders(feeder_id)
);
```

## 8. Maintenance_Events

```
CREATE TABLE Maintenance_Events (
  maintenance_id INT PRIMARY KEY,
  feeder_id INT NOT NULL,
  start_ts DATETIME NOT NULL,
  end_ts DATETIME NULL,
  description VARCHAR(255),
  performed_by VARCHAR(100),
  FOREIGN KEY (feeder_id) REFERENCES Feeders(feeder_id)
);
```

## 9. Tariffs

```
CREATE TABLE Tariffs (
  tariff_id INT PRIMARY KEY,
  tariff_name VARCHAR(100),
  unit_price DECIMAL(8,4),   -- price per kWh
  peak_start TIME NULL,      -- time-of-day peak start (optional)
  peak_end TIME NULL
);
```

## 10. Billing

(Generated monthly from meter reads)

```
CREATE TABLE Billing (
  bill_id BIGINT PRIMARY KEY AUTO_INCREMENT,
  customer_id INT NOT NULL,
  bill_month DATE NOT NULL,
  total_kwh DECIMAL(12,4),
  amount_due DECIMAL(12,2),
  generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  paid BOOLEAN DEFAULT FALSE,
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

## Sample data (minimal but covers edge cases)

```
-- Customers
INSERT INTO Customers VALUES
(1,'Alice Gupta','12 Park Lane','Mumbai','Maharashtra',10),
(2,'Bala Rao','34 MG Road','Bangalore','Karnataka',10),
(3,'Celine Wong','78 Orchard','Singapore','Singapore',20);

-- Tariffs
INSERT INTO Tariffs VALUES
(10,'Residential Standard',0.1200,'18:00:00','22:00:00'),
(20,'Commercial Peak',0.1500,'10:00:00','16:00:00');

-- Substations & Feeders
INSERT INTO Substations VALUES
(1,'Mumbai South', 'Mumbai',500.00),
(2,'Bangalore Central', 'Bangalore',300.00);

INSERT INTO Feeders VALUES
(11,1,'Feeder-MUM-11',11.0),
(12,1,'Feeder-MUM-12',11.0),
(21,2,'Feeder-BLR-21',11.0);
```

```
-- Meters
INSERT INTO Meters VALUES
(1001,1,'2023-01-01',NULL,'Residential','SN1001','Active'),
(1002,2,'2023-03-01',NULL,'Residential','SN1002','Active'),
(1003,3,'2023-05-01',NULL,'Commercial','SN1003','Active');

-- Meter assignments (meter 1001 moved feeders on 2024-10-01)
INSERT INTO Meter_Assignments VALUES
(1,1001,11,'2023-01-01 00:00:00','2024-09-30 23:59:59'),
(2,1001,12,'2024-10-01 00:00:00',NULL),
(3,1002,21,'2023-03-01 00:00:00',NULL),
(4,1003,21,'2023-05-01 00:00:00',NULL);

-- Meter_Readings (15-min intervals). Include missing interval, duplicate, esti
mated
-- For brevity: a short span for meter 1001 on 2024-12-01
INSERT INTO Meter_Readings (meter_id, reading_ts, value_kwh, quality) VALU
ES
(1001,'2024-12-01 00:15:00',0.1500,'Good'),
(1001,'2024-12-01 00:30:00',0.1600,'Good'),
(1001,'2024-12-01 00:45:00',0.0000,'Suspect'),  -- suspicious zero
(1001,'2024-12-01 01:00:00',0.1700,'Good'),
(1001,'2024-12-01 01:00:00',0.1700,'Good'),     -- duplicate reading
(1001,'2024-12-01 01:15:00',0.1800,'Estimated'), -- estimated reading
-- Meter 1002 with a missing interval (gap)
(1002,'2024-12-01 00:15:00',0.2000,'Good'),
(1002,'2024-12-01 00:45:00',0.2100,'Good'),
-- Meter 1003 commercial heavy loads
(1003,'2024-12-01 00:15:00',1.5000,'Good'),
(1003,'2024-12-01 00:30:00',1.6500,'Good'),
(1003,'2024-12-01 00:45:00',1.7000,'Good');

-- Outages and maintenance (overlapping events)
INSERT INTO Outages VALUES
(900,'Feeder-11 Outage',11,'2024-11-30 23:50:00','2024-12-01 00:20:00','Stor
```

```
m','OpsTeam');  -- short outage affecting 00:15 interval
-- Maintenance overlapping next day
INSERT INTO Maintenance_Events VALUES
(700,11,'2024-12-01 01:00:00','2024-12-01 02:00:00','Transformer check','Mai
ntenanceCrew');
```

> Note: sample data intentionally includes: duplicate reading, zero/suspect reading, estimated reads, missing intervals, meter moved between feeders, outage overlapping an interval, and maintenance overlapping readings.

---

# ERD (textual)

```
Customers (1) ──< Meters (1) ──< Meter_Readings (M)
         \              \
          \             Meter_Assignments (M)
           \
            >── Customers.tariff_id ── Tariffs (1)

Substations (1) ──< Feeders (M)
Feeders (1) ──< Meter_Assignments (M)
Feeders (1) ──< Outages (M)
Feeders (1) ──< Maintenance_Events (M)
```

---

# 5 SQL Questions (Easy → Expert)

## Easy

**Q1 — Latest reading per customer (most recent timestamp and value):**

Return `customer_id, name, meter_id, latest_reading_ts, latest_value_kwh`. If a customer has multiple meters active, return the latest across all meters.

---

## Medium

### Q2 — Daily consumption per customer for 2024-12-01:

Compute total kWh per customer for 2024-12-01 by summing `value_kwh`. Exclude readings with `quality = 'Suspect'`. Show `customer_id, name, total_kwh`.

---

## Hard

### Q3 — Detect missing intervals and duplicates per meter for 2024-12-01:

Assume proper interval is every 15 minutes. For each `meter_id`, produce:

- `meter_id`, `missing_intervals_count`, `duplicate_readings_count`, `suspect_zero_count` (value_kwh=0 and quality='Suspect') for that date.

---

## Difficult

### Q4 — Feeders with highest outage minutes in Nov 2024 and affected customers:

For each feeder compute total outage minutes during November 2024 (`outage_end` - `outage_start` in minutes, clipped to November). Then list top 3 feeders with total outage minutes and a subcount of unique customers assigned to that feeder during November (based on `Meter_Assignments` active intervals overlapping November). Output: `feeder_id, feeder_name, total_outage_minutes, unique_customers`.

---

## Expert

### Q5 — Peak hour detection & load factor per substation for Dec 1, 2024:

For each substation compute:

- `substation_id, city`
- `peak_hour_start` (hour starting timestamp with highest aggregated kW demand across all feeders for 2024-12-01; convert interval kWh to kW by `value_kwh * 4` because 15-min intervals → kW = kWh / 0.25)
- `peak_hour_demand_kw` (sum kW during that hour)
- `daily_energy_kwh` (sum kWh for the day)
- `load_factor` = `daily_energy_kwh / (peak_hour_demand_kw * 24)` (note: approximate)

Return one row per substation. Use window functions to find hour with max demand.

---

## Solutions (MySQL queries + explanations & optimization tips)

> All queries use MySQL 8+ features: window functions, CTEs, TIMESTAMPDIFF, GENERATE_SERIES emulation for missing intervals (we emulate using a derived numbers table). For production, prebuilt calendar table/interval table is recommended.

---

### Q1 — Latest reading per customer

```
-- Q1: Latest reading per customer
WITH latest_meter_reading AS (
  SELECT
    mr.meter_id,
    mr.reading_ts,
    mr.value_kwh,
    ROW_NUMBER() OVER (PARTITION BY mr.meter_id ORDER BY mr.reading_t
s DESC) AS rn
  FROM Meter_Readings mr
)
SELECT
  c.customer_id,
  c.name,
  lm.meter_id,
  lm.reading_ts AS latest_reading_ts,
  lm.value_kwh AS latest_value_kwh
FROM Customers c
JOIN Meters m ON c.customer_id = m.customer_id
LEFT JOIN (
  SELECT meter_id, reading_ts, value_kwh
  FROM latest_meter_reading
```

```
  WHERE rn = 1
) lm ON m.meter_id = lm.meter_id
WHERE m.status = 'Active'
ORDER BY c.customer_id;
```

**Explanation:**

- Compute latest per `meter_id` using `ROW_NUMBER()` ordered by `reading_ts` descending.
- Join to `Customers` and `Meters` to return latest per customer-meter pair.
- If a customer has multiple meters, this returns one row per meter. To show a single latest across all meters per customer, wrap with another CTE and pick max `reading_ts` per customer.

**Variant (single row per customer — latest across all meters):**

```
WITH latest_per_meter AS (
  SELECT mr.meter_id, mr.reading_ts, mr.value_kwh, m.customer_id,
         ROW_NUMBER() OVER (PARTITION BY mr.meter_id ORDER BY mr.readin
g_ts DESC) rn
  FROM Meter_Readings mr
  JOIN Meters m ON mr.meter_id = m.meter_id
),
latest_per_customer AS (
  SELECT customer_id, reading_ts, value_kwh,
         ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY reading_t
s DESC) rn
  FROM (
    SELECT meter_id, customer_id, reading_ts, value_kwh FROM latest_per_met
er WHERE rn = 1
  ) t
)
SELECT c.customer_id, c.name, lpc.reading_ts, lpc.value_kwh
FROM Customers c
LEFT JOIN latest_per_customer lpc ON c.customer_id = lpc.customer_id AND l
```

```
pc.rn = 1;
```

**Index suggestions:**

- `Meter_Readings(meter_id, reading_ts)` (composite)
- `Meters(customer_id)`

---

## Q2 — Daily consumption per customer for 2024-12-01

```
-- Q2: Daily consumption per customer for 2024-12-01
SELECT
  c.customer_id,
  c.name,
  ROUND(SUM(mr.value_kwh), 4) AS total_kwh
FROM Meter_Readings mr
JOIN Meters m ON mr.meter_id = m.meter_id
JOIN Customers c ON m.customer_id = c.customer_id
WHERE DATE(mr.reading_ts) = '2024-12-01'
  AND mr.quality <> 'Suspect'
GROUP BY c.customer_id, c.name
ORDER BY total_kwh DESC;
```

**Explanation:**

- Straightforward sum filtered by date and excluding `Suspect` quality.
- Use `DATE(reading_ts)` for clarity; for large tables prefer range filter: `reading_ts >= '2024-12-01' AND reading_ts < '2024-12-02'`.

**Performance tip:**

- Avoid `DATE()` on the column (prevents index use). Use:

  ```
  WHERE mr.reading_ts >= '2024-12-01 00:00:00' AND mr.reading_ts < '2024-12-02 00:00:00'
  ```

## Q3 — Detect missing intervals and duplicates per meter for 2024-12-01

This is tricky — we must know expected intervals (every 15 minutes). We emulate intervals using a numbers table. For small datasets we build it on the fly.

```
-- Q3: Missing intervals, duplicates, and suspect zeros per meter
WITH params AS (
  SELECT '2024-12-01 00:00:00' AS day_start, '2024-12-02 00:00:00' AS day_
end, 15 AS interval_min
),
nums AS (
  -- generate integers 0..95 for 24h * 4 intervals
  SELECT 0 AS n UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SEL
ECT 3 UNION ALL SELECT 4
  UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION A
LL SELECT 8 UNION ALL SELECT 9
  UNION ALL SELECT 10 UNION ALL SELECT 11 UNION ALL SELECT 12 UNION
ALL SELECT 13 UNION ALL SELECT 14
  UNION ALL SELECT 15 UNION ALL SELECT 16 UNION ALL SELECT 17 UNION
ALL SELECT 18 UNION ALL SELECT 19
  UNION ALL SELECT 20 UNION ALL SELECT 21 UNION ALL SELECT 22 UNIO
N ALL SELECT 23 UNION ALL SELECT 24
  UNION ALL SELECT 25 UNION ALL SELECT 26 UNION ALL SELECT 27 UNIO
N ALL SELECT 28 UNION ALL SELECT 29
  UNION ALL SELECT 30 UNION ALL SELECT 31 UNION ALL SELECT 32 UNIO
N ALL SELECT 33 UNION ALL SELECT 34
  UNION ALL SELECT 35 UNION ALL SELECT 36 UNION ALL SELECT 37 UNIO
N ALL SELECT 38 UNION ALL SELECT 39
  UNION ALL SELECT 40 UNION ALL SELECT 41 UNION ALL SELECT 42 UNIO
N ALL SELECT 43 UNION ALL SELECT 44
  UNION ALL SELECT 45 UNION ALL SELECT 46 UNION ALL SELECT 47 UNIO
N ALL SELECT 48 UNION ALL SELECT 49
```

```
  UNION ALL SELECT 50 UNION ALL SELECT 51 UNION ALL SELECT 52 UNIO
N ALL SELECT 53 UNION ALL SELECT 54
  UNION ALL SELECT 55 UNION ALL SELECT 56 UNION ALL SELECT 57 UNIO
N ALL SELECT 58 UNION ALL SELECT 59
  UNION ALL SELECT 60 UNION ALL SELECT 61 UNION ALL SELECT 62 UNIO
N ALL SELECT 63 UNION ALL SELECT 64
  UNION ALL SELECT 65 UNION ALL SELECT 66 UNION ALL SELECT 67 UNIO
N ALL SELECT 68 UNION ALL SELECT 69
  UNION ALL SELECT 70 UNION ALL SELECT 71 UNION ALL SELECT 72 UNIO
N ALL SELECT 73 UNION ALL SELECT 74
  UNION ALL SELECT 75 UNION ALL SELECT 76 UNION ALL SELECT 77 UNIO
N ALL SELECT 78 UNION ALL SELECT 79
  UNION ALL SELECT 80 UNION ALL SELECT 81 UNION ALL SELECT 82 UNIO
N ALL SELECT 83 UNION ALL SELECT 84
  UNION ALL SELECT 85 UNION ALL SELECT 86 UNION ALL SELECT 87 UNIO
N ALL SELECT 88 UNION ALL SELECT 89
  UNION ALL SELECT 90 UNION ALL SELECT 91 UNION ALL SELECT 92 UNIO
N ALL SELECT 93 UNION ALL SELECT 94
  UNION ALL SELECT 95
),
expected_intervals AS (
  SELECT
    TIMESTAMPADD(MINUTE, n.interval_min * nums.n, params.day_start) AS ex
pected_ts
  FROM params
  CROSS JOIN nums
  CROSS JOIN (SELECT 1 AS interval_min) AS n
),
reads AS (
  SELECT mr.meter_id, mr.reading_ts, mr.value_kwh, mr.quality
  FROM Meter_Readings mr
  WHERE mr.reading_ts >= '2024-12-01 00:00:00' AND mr.reading_ts < '2024-
12-02 00:00:00'
)
SELECT
  m.meter_id,
```

```
  COALESCE(miss.missing_count, 0) AS missing_intervals_count,
  COALESCE(dup.duplicate_count, 0) AS duplicate_readings_count,
  COALESCE(sz.suspect_zero_count, 0) AS suspect_zero_count
FROM Meters m
LEFT JOIN (
  -- missing intervals per meter: expected timestamps not present in reads
  SELECT r.meter_id, COUNT(*) AS missing_count
  FROM (
    SELECT mtr.meter_id, ei.expected_ts
    FROM Meters mtr
    CROSS JOIN expected_intervals ei
  ) r
  LEFT JOIN reads rr ON rr.meter_id = r.meter_id AND rr.reading_ts = r.expecte
d_ts
  WHERE rr.reading_ts IS NULL
  GROUP BY r.meter_id
) miss ON miss.meter_id = m.meter_id
LEFT JOIN (
  -- duplicates: count readings per (meter, ts) > 1
  SELECT meter_id, SUM(cnt - 1) AS duplicate_count
  FROM (
    SELECT meter_id, reading_ts, COUNT(*) AS cnt
    FROM reads
    GROUP BY meter_id, reading_ts
    HAVING COUNT(*) > 1
  ) x
  GROUP BY meter_id
) dup ON dup.meter_id = m.meter_id
LEFT JOIN (
  -- suspect zeros
  SELECT meter_id, COUNT(*) AS suspect_zero_count
  FROM reads
  WHERE value_kwh = 0 AND quality = 'Suspect'
  GROUP BY meter_id
) sz ON sz.meter_id = m.meter_id
```

```
ORDER BY m.meter_id;
```

**Explanation:**

- Generate expected 15-min intervals for the day (0..95 → 96 intervals).
- For each meter, check which expected timestamps are missing in `Meter_Readings` → `missing_intervals_count`.
- Count duplicate readings per `(meter_id, reading_ts)` where count >1.
- Count suspect zero readings (value_kwh = 0 & quality='Suspect').

**Performance tips:**

- Replace the ad-hoc `nums` with a permanent `calendar_intervals` table to avoid massive cross joins.
- Use index `Meter_Readings(meter_id, reading_ts)` so lookups are fast.

---

## Q4 — Feeders with highest outage minutes in Nov 2024 and affected customers

```
-- Q4: Outage minutes in Nov 2024 by feeder, and unique customers assigne
d during November

WITH outages_nov AS (
  SELECT
    o.feeder_id,
    GREATEST(TIMESTAMPDIFF(MINUTE,
          GREATEST(o.outage_start, '2024-11-01 00:00:00'),
          LEAST(COALESCE(o.outage_end, NOW()), '2024-11-30 23:59:59')
        ), 0) AS outage_minutes_clipped
  FROM Outages o
  WHERE o.outage_start < '2024-12-01' AND (o.outage_end IS NULL OR o.outa
ge_end >= '2024-11-01')
),
assignments_nov AS (
```

```
  -- assignments active at any time during November
  SELECT DISTINCT ma.meter_id, ma.feeder_id
  FROM Meter_Assignments ma
  WHERE ma.assigned_from < '2024-12-01 00:00:00'
    AND (ma.assigned_to IS NULL OR ma.assigned_to >= '2024-11-01 00:00:0
0')
)
SELECT
  f.feeder_id,
  f.feeder_name,
  COALESCE(SUM(o.outage_minutes_clipped),0) AS total_outage_minutes,
  COALESCE(COUNT(DISTINCT ma_meter.customer_id),0) AS unique_custom
ers
FROM Feeders f
LEFT JOIN outages_nov o ON f.feeder_id = o.feeder_id
LEFT JOIN (
  -- link meters to customers for assignments
  SELECT ma.feeder_id, m.customer_id
  FROM Meter_Assignments ma
  JOIN Meters m ON ma.meter_id = m.meter_id
  WHERE ma.assigned_from < '2024-12-01 00:00:00'
    AND (ma.assigned_to IS NULL OR ma.assigned_to >= '2024-11-01 00:00:0
0')
) ma_meter ON ma_meter.feeder_id = f.feeder_id
WHERE f.substation_id IS NOT NULL
GROUP BY f.feeder_id, f.feeder_name
ORDER BY total_outage_minutes DESC
LIMIT 3;
```

**Explanation:**

- `outages_nov` computes clipped outage minutes to November (handles outages crossing month boundaries and ongoing outages).
- `assignments_nov` finds meter assignments that overlap November.

- We join feeders to outages and unique customers (via meter assignments→meters→customers).
- `COUNT(DISTINCT customer_id)` yields unique customers impacted.

**Edge cases handled:**

- Ongoing outages (`outage_end IS NULL`) are clipped to Nov 30 using `COALESCE(..., NOW())`, but we then `LEAST` with end of month; ensures not overcounting.
- Assignments that start before November and end after or NULL (still active) are included.

**Index suggestions:**

- `Outages(feeder_id, outage_start, outage_end)`
- `Meter_Assignments(feeder_id, assigned_from, assigned_to)`
- `Meters(meter_id, customer_id)`

---

## Q5 — Peak hour detection & load factor per substation for Dec 1, 2024 (Expert)

**Notes/Assumptions:**

- Convert 15-min `value_kwh` to instantaneous kW for that interval: `kW = value_kwh / 0.25 = value_kwh * 4`.
- Peak hour = contiguous 1-hour window (4 intervals). We aggregate kW across intervals within each hour starting at HH:00 (i.e., group by `DATE_FORMAT(reading_ts, '%Y-%m-%d %H:00:00')`).
- Map meters → current feeder → substation using `Meter_Assignments` valid at reading_ts. We'll join on assignment where `assigned_from <= reading_ts` and `(assigned_to IS NULL OR assigned_to >= reading_ts)` (this maps per-interval assignment).

```
-- Q5: Peak hour and load factor per substation for 2024-12-01

WITH day_reads AS (
  SELECT
    mr.meter_id,
```

```
    mr.reading_ts,
    mr.value_kwh,
    -- compute kW for the interval (kWh per 15 min → kW)
    (mr.value_kwh * 4.0) AS interval_kw
  FROM Meter_Readings mr
  WHERE mr.reading_ts >= '2024-12-01 00:00:00' AND mr.reading_ts < '2024-
12-02 00:00:00'
    AND mr.quality <> 'Suspect'
),
-- map each reading to the feeder assignment active at that timestamp
reads_with_feeder AS (
  SELECT dr.*, ma.feeder_id
  FROM day_reads dr
  LEFT JOIN Meter_Assignments ma
    ON dr.meter_id = ma.meter_id
   AND ma.assigned_from <= dr.reading_ts
   AND (ma.assigned_to IS NULL OR ma.assigned_to >= dr.reading_ts)
),
-- join to substation
reads_with_substation AS (
  SELECT rwf.*, f.substation_id
  FROM reads_with_feeder rwf
  LEFT JOIN Feeders f ON rwf.feeder_id = f.feeder_id
),
-- aggregate kW by substation and hour bucket
hourly_substation AS (
  SELECT
    substation_id,
    DATE_FORMAT(reading_ts, '%Y-%m-%d %H:00:00') AS hour_start,
    SUM(interval_kw) AS total_kw_in_hour,
    SUM(value_kwh) AS total_kwh_in_hour
  FROM reads_with_substation
  GROUP BY substation_id, hour_start
),
-- find peak hour per substation using window functions
ranked_hours AS (
```

```
  SELECT
    hs.*,
    RANK() OVER (PARTITION BY hs.substation_id ORDER BY hs.total_kw_in_ho
ur DESC) AS rk
  FROM hourly_substation hs
),
-- daily aggregates per substation
daily_substation AS (
  SELECT
    rs.substation_id,
    SUM(rs.value_kwh) AS daily_energy_kwh
  FROM reads_with_substation rs
  GROUP BY rs.substation_id
)
SELECT
  s.substation_id,
  s.city,
  rh.hour_start AS peak_hour_start,
  rh.total_kw_in_hour AS peak_hour_demand_kw,
  COALESCE(ds.daily_energy_kwh,0) AS daily_energy_kwh,
  CASE
    WHEN rh.total_kw_in_hour IS NULL OR rh.total_kw_in_hour = 0 THEN NULL
    ELSE ROUND( COALESCE(ds.daily_energy_kwh,0) / (rh.total_kw_in_hour * 2
4.0), 4)
  END AS load_factor
FROM Substations s
LEFT JOIN (
  SELECT * FROM ranked_hours WHERE rk = 1
) rh ON s.substation_id = rh.substation_id
LEFT JOIN daily_substation ds ON s.substation_id = ds.substation_id
ORDER BY s.substation_id;
```

**Explanation:**

- `day_reads` selects readings for the day and computes `interval_kw`.

- `reads_with_feeder` joins each reading to its feeder assignment active at reading time (important because meters can move).
- `reads_with_substation` resolves `substation_id` from feeder.
- `hourly_substation` aggregates kW per substation per hour (HH:00).
- `ranked_hours` uses `RANK()` to find the hour with the highest demand per substation.
- `daily_substation` computes total kWh per substation for the day.
- Final SELECT returns peak hour and `load_factor` = daily_energy_kwh / (peak_hour_kw * 24) — an approximate load factor.

**Notes on correctness & assumptions:**

- Load factor formula approximated: Usually `load_factor = (total energy over period) / (peak demand × period length)`. Here period length is 24 hours.
- If multiple hours tie for peak, `RANK()` returns them; this picks the first in ordered set (or may produce multiple rows if multiple rk=1 — the sample selects all rk=1 but joined back to Substations gives one row per substation with arbitrary tie resolution).

**Index suggestions:**

- `Meter_Readings(meter_id, reading_ts, value_kwh)` (composite)
- `Meter_Assignments(meter_id, assigned_from, assigned_to)` (composite)
- `Feeders(feeder_id, substation_id)`
- Partition `Meter_Readings` by RANGE on `reading_ts` (monthly)

**Performance tips:**

- For massive data, pre-aggregate hourly summaries per meter in daily ETL (materialized) to avoid scanning raw interval data repeatedly.
- Avoid joining large `Meter_Readings` raw tables for ad-hoc queries unless filtered by date ranges and using partition pruning.
- Use approximate methods (sketches) for quick dashboards, and exact for billing.

---

## Final notes & production recommendations

1. **Time-series storage**: `Meter_Readings` should be partitioned by date (monthly or weekly). Consider clickhouse/influx/bigtable for massive scale OLAP.
2. **Calendar table**: Maintain a `time_intervals` table for efficient interval generation rather than on-the-fly numbers tables.
3. **Data quality pipeline**: Run daily checks to mark `quality` = 'Estimated' / 'Suspect' and flag meters with repeated gaps/zeros.
4. **Meter assignment history**: Always join readings to the assignment time window — meters can be moved.
5. **ETL & materialized summaries**: Keep `meter_hourly_agg(meter_id, hour_start, kwh_sum)` for repeated analytics queries.
6. **Indexing**: primary keys, and composite indexes on join/time columns (e.g., `(meter_id, reading_ts)` and `(feeder_id, outage_start)`).
7. **Anomaly detection**: Use rolling z-scores over historical intervals to detect spikes/tampering; this can be implemented with SQL window functions for small datasets or ML pipelines for larger ones.