Energy Billing System - Jira Issue Tracker

Project Key: EBS
Project Type: Software Development
Workflow: Agile/Scrum

EPICS

Epic EBS-E1: Implement Residential Tiered Billing Logic

Epic Name: Residential Tiered Billing MVP
Epic Owner: James Patterson (Backend Tech Lead)
Reporter: Marcus Rodriguez (Product Owner)
Status: IN PROGRESS
Priority: Highest
Labels: `core-billing`, `mvp`, `residential`, `rating-service`

Epic Description:

This Epic encompasses all development, testing, and documentation required to implement the primary residential tiered rate plan (R1) in the new Rating Service microservice. This represents the minimum viable product for residential bill calculation and is a critical path dependency for the entire project.

The R1 rate plan uses a two-tier increasing block rate structure where consumption from 0-500 kWh is charged at a lower rate (baseline tier) and consumption above 500 kWh is charged at a higher rate. The system must also handle seasonal rate variations (summer vs. winter rates) and apply appropriate fixed monthly charges.

Business Value:

R1 is the default rate plan for approximately 1.8 million residential customers (78% of our customer base). Successful implementation of this Epic unblocks the entire residential billing workflow and represents approximately $95M in monthly billing volume.

Acceptance Criteria (Epic-Level):

✅ **AC-1:** Rating Service can successfully ingest valid Usage Records from the Meter Ingestion Service via Kafka message queue

✅ **AC-2:** Service correctly identifies R1 rate plan customers from Customer Account database

🔄 **AC-3:** System accurately applies two-tier consumption rate logic with 500 kWh threshold

🔄 **AC-4:** Service correctly selects summer vs. winter rates based on billing cycle end date

🔄 **AC-5:** Generated invoice line items accurately reflect tier 1 charges, tier 2 charges, and fixed charges as separate line items

🔄 **AC-6:** All calculations maintain 2 decimal precision and match financial audit requirements

🔄 **AC-7:** Complete audit trail logged for each calculation including rate plan version, inputs, and outputs

🔄 **AC-8:** System handles edge cases: zero consumption, exactly 500 kWh consumption, partial billing cycles

🔄 **AC-9:** Unit test coverage exceeds 90% for all R1 calculation logic

⏳ **AC-10:** Integration tests validate end-to-end billing flow from usage ingestion through invoice line item generation

⏳ **AC-11:** Performance testing confirms Rating Service can process 80,000 R1 accounts within 4-hour processing window

⏳ **AC-12:** UAT sign-off received from Finance team using production-like test data

Technical Architecture:

- Microservice: Rating Service (Spring Boot, Java 17)
- Database: PostgreSQL 15 with rate plan configuration tables
- Messaging: Kafka for async usage data consumption
- API: REST endpoints for synchronous rating requests (for customer portal usage estimates)

Associated Stories: 18 stories, 187 story points total

Dependencies:

- ⚠️ **Blocks:** EBS-E1 (Residential Billing), EBS-E3 (TOU Billing), EBS-E4 (Invoice Generation) - ALL CRITICAL PATH
- ⚠️ **Depends On:** Access to legacy mainframe system (secured), Database infrastructure (complete)

Current Status & Risk Assessment:

🔴 **CRITICAL RISK - ESCALATED:** Data quality issues worse than anticipated. Original estimate of 5% problematic records has increased to 15% based on deeper analysis. Current timeline projects 2-week delay to Data Verification Milestone.

Mitigation Actions In Progress:

1. Focused strategy: Migrate only active customers (95% of volume) in Phase 1
2. Secured approval for 2 additional data analysts (4-week contract)
3. Created "quarantine database" for problematic records requiring manual review
4. Automated data quality validation scripts to accelerate remediation
5. Weekly data quality review meetings with Finance team

Next Major Milestone: Phase 1 Migration Complete - Target: November 8, 2025 (REVISED from October 25)

Epic EBS-E3: Implement Time-of-Use Billing

Epic Name: Time-of-Use (TOU) Rate Plans
Epic Owner: James Patterson (Backend Tech Lead)
Reporter: Marcus Rodriguez (Product Owner)
Status: NOT STARTED
Priority: High
Labels: `tou-billing`, `residential`, `rating-service`, `smart-meter-required`

Epic Description:

Implement Time-of-Use (TOU) billing capability to support residential rate plan R2. TOU billing charges different rates based on when electricity is consumed (peak vs. off-peak hours), incentivizing customers to shift consumption to off-peak periods and reducing grid strain during high-demand periods.

TOU billing requires 15-minute interval meter data from smart meters, complex period classification logic (peak/off-peak/super-off-peak), holiday calendar management, and sophisticated usage aggregation across multiple time periods.

Business Value:

TOU rates are critical for grid modernization and demand response programs. Approximately 145,000 customers have opted into TOU billing (6% penetration), with aggressive targets to grow to 25% penetration (575,000 customers) over next 18 months. TOU participants typically reduce peak consumption by 8-12%, providing significant grid capacity benefits.

Technical Complexity:

This Epic is significantly more complex than tiered billing (EBS-E1) due to:

- Requires interval-level data processing (2,880 intervals per 30-day cycle per customer)
- Complex time period classification with timezone handling
- Holiday calendar management and maintenance
- Dynamic peak period definitions that vary by season and utility service territory
- Performance optimization critical due to high data volumes

Acceptance Criteria (Epic-Level):

⏳ **AC-1:** Rating Service can ingest and process 15-minute interval meter data

⏳ **AC-2:** System correctly classifies each interval as PEAK, OFF_PEAK, or SUPER_OFF_PEAK based on date, time, and holiday calendar

⏳ **AC-3:** Holiday calendar is configurable and maintainable by business users without code deployment

⏳ **AC-4:** System accurately aggregates consumption across time periods for multi-thousand interval datasets

⏳ **AC-5:** Invoice line items clearly show consumption and charges broken down by time period

⏳ **AC-6:** Performance testing validates processing of 145,000 TOU accounts within 4-hour window

⏳ **AC-7:** Timezone handling correctly processes intervals across daylight saving time transitions

Associated Stories: 14 stories, 176 story points (estimated)

Dependencies:

- ⚠️ **Depends On:** EBS-E1 (Tiered Billing) must be complete and stable
- ⚠️ **Depends On:** Smart meter interval data integration (infrastructure team)

Planned Start: Sprint 17 (November 2025)
Estimated Completion: Sprint 21 (January 2026)

Epic EBS-E4: Invoice Generation and Delivery Service

Epic Name: Invoice Generation Microservice
Epic Owner: Kevin Park (Frontend Tech Lead)
Reporter: Marcus Rodriguez (Product Owner)
Status: IN PROGRESS

Priority: Highest
Labels: `invoice`, `pdf-generation`, `email-delivery`, `core-service`

Epic Description:

Develop the Invoice Generation Service responsible for creating customer-facing invoice documents (PDF format), invoice summary records, and coordinating delivery through multiple channels (email, postal mail, customer portal). The service must generate professional, compliant, accessible invoice documents that meet regulatory requirements and customer usability standards.

This service consumes billing calculation results from the Rating Service and produces final invoice documents with detailed line items, usage comparisons, payment instructions, and regulatory disclosures.

Key Responsibilities:

1. **Invoice Document Generation:**
   - Generate PDF invoices using customer-friendly templates
   - Include detailed line-item breakdown of charges
   - Show usage comparison graphs (current vs. historical)
   - Display payment options and due date prominently
   - Include required regulatory disclosures and notices
2. **Invoice Data Management:**
   - Store invoice records in Invoice Repository database
   - Maintain invoice lifecycle status (generated, delivered, paid, overdue)
   - Support invoice retrieval for customer service and self-service portal
   - Archive historical invoices per retention policy (7 years)
3. **Delivery Coordination:**
   - Queue invoices for email delivery with PDF attachment
   - Generate batch files for postal mail vendor (customers without email)
   - Publish invoice availability notifications to customer portal
   - Handle delivery failures and retry logic
4. **Compliance & Accessibility:**
   - PDF/A format for long-term archival
   - Section 508 accessibility compliance for screen readers
   - Multi-language support (English, Spanish, Hebrew)
   - Include all PUC-required disclosures and consumer rights information

Business Value:

Invoice generation is the final critical step in the billing process - the customer-facing output that represents all upstream calculations. Quality, clarity, and timeliness of invoice delivery directly impacts customer satisfaction, payment rates, and call center volume. Poor invoice quality or delivery failures result in immediate customer complaints and payment delays.

Acceptance Criteria (Epic-Level):

🔄 **AC-1:** Service generates PDF invoices meeting brand standards and regulatory requirements

🔄 **AC-2:** Invoice templates support all rate plan types (tiered, TOU, demand-based)

🔄 **AC-3:** Usage comparison graphs accurately display current vs. historical consumption

⏳ **AC-4:** System processes 80,000 invoices per day within 2-hour processing window

⏳ **AC-5:** Email delivery achieves >98% success rate with retry handling for failures

⏳ **AC-6:** PDFs meet PDF/A archival standards and Section 508 accessibility requirements

⏳ **AC-7:** Invoice data retrievable through API for customer portal integration

⏳ **AC-8:** Multi-language invoice generation working for English and Spanish

Associated Stories: 16 stories, 142 story points

Dependencies:

- ⚠️ **Depends On:** EBS-E1 (Rating Service) must be producing invoice line items
- ⚠️ **Integrates With:** Email delivery service (SendGrid), Print vendor API, Customer Portal

Progress: 4/16 stories complete (25%)
Estimated Completion: Sprint 18 (November 2025)

Epic EBS-E5: Customer Self-Service Portal

Epic Name: Customer Web Portal
Epic Owner: Kevin Park (Frontend Tech Lead)
Reporter: Marcus Rodriguez (Product Owner)
Status: NOT STARTED
Priority: Medium
Labels: `customer-portal`, `self-service`, `frontend`, `post-mvp`

Epic Description:

Develop a modern, responsive web application enabling customers to view billing history, make payments, manage account preferences, analyze usage patterns, and simulate billing under different rate plans. The portal empowers customers with self-service capabilities, reducing call center volume and improving customer satisfaction.

Target Launch: September 2026 (Post-cutover enhancement)

Key Features:

- View current balance and payment due date
- Access last 36 months of invoice history (view and download PDFs)
- Make one-time payments via credit card, debit card, or bank account
- Enroll in auto-pay and paperless billing
- View daily/monthly usage data with interactive charts
- Compare usage to similar homes in neighborhood (anonymized)
- Simulate estimated bills under different rate plan options
- Set usage alerts and budget notifications
- Update contact information and communication preferences
- Report power outages and service issues
- Schedule service appointments

Associated Stories: 28 stories, 245 story points (estimated)

USER STORIES

Story EBS-24: Create RateTier Database Table

Story Type: Technical Task
Reporter: James Patterson
Assignee: Developer-1 (Sarah Kim)
Status: DONE ✅
Sprint: Sprint 11
Story Points: 5
Labels: `database`, `schema`, `backend`, `rating-service`

Story Description:

As a Developer, I want to create the `rate_tier` database table so that tier-based rate information can be persisted and retrieved efficiently by the Rating Service.

Acceptance Criteria:

✅ **AC-1:** Table `rate_tier` created in PostgreSQL with proper schema definition

✅ **AC-2:** Foreign key relationship established to `rate_plan` table

✅ **AC-3:** Indexes created on frequently queried columns (rate_plan_id, tier_number)

✅ **AC-4:** Database migration script created using Flyway versioning

✅ **AC-5:** Entity class created in Java code with JPA annotations

✅ **AC-6:** Repository interface created extending JpaRepository

Technical Details:

```
CREATE TABLE rate_tier (
  rate_tier_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  rate_plan_id UUID NOT NULL,
  tier_number INTEGER NOT NULL,
  threshold_kwh NUMERIC(10,2) NOT NULL,
  rate_per_kwh_summer NUMERIC(8,4) NOT NULL,
  rate_per_kwh_winter NUMERIC(8,4) NOT NULL,
  effective_date DATE NOT NULL,
  expiration_date DATE,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_rate_plan FOREIGN KEY (rate_plan_id)
    REFERENCES rate_plan(rate_plan_id),
  CONSTRAINT unique_tier_per_plan
    UNIQUE (rate_plan_id, tier_number, effective_date)
);

CREATE INDEX idx_rate_tier_plan ON rate_tier(rate_plan_id);
CREATE INDEX idx_rate_tier_effective ON rate_tier(effective_date, expiration_date);
```

Definition of Done:

- Code reviewed and approved
- Migration script tested in dev environment
- Unit tests written for repository methods
- Documentation updated in Confluence

Completed: September 18, 2025

Story EBS-25: Define Two-Tier Rate Structure for R1

Story Type: User Story
Reporter: Marcus Rodriguez
Assignee: Developer-2 (Michael Chen)
Status: DONE ✅
Sprint: Sprint 11
Story Points: 3
Labels: `rate-configuration`, `r1-rate-plan`, `backend`

Story Description:

As a Billing Analyst, I want to define the two-tier consumption structure for Rate Plan R1 so that I can apply the correct rates to customer usage during billing calculations.

Acceptance Criteria:

✅ **AC-1:** Rate Plan R1 configuration created in database with tier definitions

✅ **AC-2:** Tier 1: 0-500 kWh with summer rate $0.1247/kWh and winter rate $0.1198/kWh

✅ **AC-3:** Tier 2: >500 kWh with summer rate $0.1584/kWh and winter rate $0.1498/kWh

✅ **AC-4:** Fixed charges configured: Service Charge $15.00, Infrastructure Fee $3.50

✅ **AC-5:** Rate configuration retrievable through RatePlanService API

Business Rules:

GIVEN a customer is on Rate Plan R1
WHEN their usage for the billing cycle is 750 kWh
AND the billing cycle is in winter season (Oct-May)
THEN the system must calculate:
  Tier 1 charges: 500 kWh × $0.1198 = $59.90
  Tier 2 charges: 250 kWh × $0.1498 = $37.45
  Energy subtotal: $97.35
  Fixed charges: $15.00 + $3.50 = $18.50
  Subtotal before tax: $115.85

Test Data Created:

- Sample R1 rate plan record with tier configurations
- Test cases for 0 kWh, 250 kWh, 500 kWh, 750 kWh, 1500 kWh consumption scenarios

Definition of Done:

- Rate configuration inserted into database

- Configuration data validated by Finance team
- Test data available in all environments (dev, test, staging)
- API integration test validates rate retrieval

Completed: September 20, 2025

Story EBS-51: Map Legacy Customer IDs to New UUIDs

Story Type: User Story
Reporter: Lisa Thompson
Assignee: Data-Engineer-1 (Jennifer Wong)
Status: IN PROGRESS 🔄
Sprint: Sprint 14
Story Points: 8
Labels: `data-migration`, `etl`, `customer-data`

Story Description:

As a Data Engineer, I need an ETL script to map legacy CustomerIDs (numeric format) to new UUID-based identifiers so that customer history is preserved and all references maintain data integrity during the migration.

Background:

Legacy system uses 7-digit numeric Customer IDs (example: 2847563). New EBS uses UUID format for all entity identifiers. We must create a deterministic mapping that:

- Ensures one-to-one correspondence between old and new IDs
- Maintains referential integrity across all related tables
- Enables lookup in both directions (old->new, new->old)
- Preserves audit trail of ID transformation

Acceptance Criteria:

✅ **AC-1:** Python ETL script created using Apache NiFi integration

🔄 **AC-2:** Script generates deterministic UUIDs from legacy Customer IDs (using UUID v5 with consistent namespace)

🔄 **AC-3:** ID mapping table created storing old_customer_id, new_customer_uuid, migration_timestamp

🔄 **AC-4:** Validation confirms no duplicate UUIDs generated (100% uniqueness)

⏳ **AC-5:** Performance testing confirms script processes 2.35M records within 2 hours

⏳ **AC-6:** Rollback procedure tested to revert ID mappings if needed

⏳ **AC-7:** Integration test validates all foreign key references updated correctly

Technical Approach:

```
import uuid
import hashlib

NAMESPACE_UUID = uuid.UUID('6ba7b810-9dad-11d1-80b4-00c04fd430c8')

def generate_deterministic_uuid(legacy_customer_id):
    """
    Generate UUID v5 based on legacy customer ID
    Ensures same input always produces same UUID
    """
    id_string = f"CUSTOMER-{legacy_customer_id}"
    return uuid.uuid5(NAMESPACE_UUID, id_string)

# Example
legacy_id = 2847563
new_uuid = generate_deterministic_uuid(legacy_id)
# Result: 'a7f3c2e1-5b4d-5a9f-8c7e-1d2f3e4a5b6c'
```

Dependencies:

- Access to legacy customer database (secured)
- New EBS customer table schema finalized

Current Progress:

- Script 70% complete
- Successfully tested with 50,000 test records
- Performance optimization needed for full dataset

Blockers:

- None currently

Definition of Done:

- Code reviewed and approved by Tech Lead
- Successfully processes full 2.35M customer dataset

- Validation report confirms 100% success rate
- Documentation added to migration runbook

Target Completion: October 11, 2025

Story EBS-52: Create Reconciliation Report Tool

Story Type: User Story
Reporter: Marcus Rodriguez
Assignee: QA-Engineer-1 (David Park)
Status: NOT STARTED ⏳
Sprint: Sprint 15 (Planned)
Story Points: 13
Labels: `testing`, `data-validation`, `reconciliation`, `quality-assurance`

Story Description:

As a QA Analyst, I want an automated reconciliation report tool so I can compare total billed amounts between the old legacy system and new EBS during parallel testing to ensure billing accuracy before production cutover.

Business Context:

During parallel testing phase (January-March 2026), both legacy and new systems will process the same customer accounts. We must validate that billing amounts match within acceptable tolerance to gain confidence in cutover readiness. Any discrepancies must be investigated and resolved.

Acceptance Criteria:

⏳ **AC-1:** Tool extracts billing data from both legacy system and new EBS for same billing cycle

⏳ **AC-2:** Report compares billing amounts at account level with tolerance threshold (0.1% or $0.50, whichever is greater)

⏳ **AC-3:** Dashboard displays summary statistics: total accounts, matched accounts, discrepant accounts, total variance

⏳ **AC-4:** Detailed variance report lists all discrepant accounts with old amount, new amount, and variance percentage

⏳ **AC-5:** Report exportable to Excel format for Finance team analysis

⏳ **AC-6:** Tool identifies common patterns in discrepancies (e.g., all tax calculation variances)

⏳ **AC-7:** Automated email notifications sent when variance exceeds acceptable threshold

Functional Requirements:

- Process 80,000 accounts per daily billing run
- Complete reconciliation within 1 hour of both systems finishing
- Store reconciliation history for trend analysis
- Support drill-down to invoice line-item level comparison

Acceptance Tolerance Thresholds:

- Account-level variance: ≤0.1% or $0.50 (whichever is greater)
- Total billing cycle variance: ≤0.01% of total billed amount
- Number of discrepant accounts: <1% of total accounts processed

Technical Stack:

- Python pandas for data processing
- PostgreSQL for data storage
- Tableau/Power BI for dashboard visualization
- Email integration via SendGrid API

Dependencies:

- Access to legacy system billing output files
- EBS invoice data available in database
- Business rules documentation for tolerance thresholds

Estimated Start: Sprint 15 (October 27, 2025)

Story EBS-74: Tokenize Payment Card Data

Story Type: Security Story
Reporter: Priya Sharma (Security Lead)
Assignee: Developer-3 (Alex Martinez)
Status: NOT STARTED ⏳
Sprint: Backlog
Story Points: 8
Labels: `security`, `pci-compliance`, `payment-processing`, `high-priority`

Story Description:

As a Security Officer, I want all payment card data to be tokenized by the payment gateway (Stripe) so that the EBS application never stores sensitive cardholder data and remains out of PCI-DSS scope.

Security Requirement:

PCI-DSS (Payment Card Industry Data Security Standard) imposes significant security requirements on any system that stores, processes, or transmits credit card data. By tokenizing card data at the payment gateway and only storing tokens in EBS, we dramatically reduce compliance scope and security risk.

Acceptance Criteria:

⏳ **AC-1:** Customer payment card information collected via Stripe.js (client-side) - EBS backend never receives raw card data

⏳ **AC-2:** Stripe API returns secure token representing card - token stored in EBS database instead of actual card number

⏳ **AC-3:** Payment processing uses token to charge card through Stripe API

⏳ **AC-4:** Card data display shows only last 4 digits and card brand (e.g., "Visa ending in 1234")

⏳ **AC-5:** Token invalidation process implemented when customer removes payment method

⏳ **AC-6:** Security audit confirms no plaintext card data stored anywhere in EBS infrastructure

⏳ **AC-7:** PCI-DSS Self-Assessment Questionnaire (SAQ-A) eligibility confirmed with QSA consultant

Integration Flow:

1. Customer Portal (Frontend)
   ↓
2. Stripe.js captures card data (client-side, never touches EBS backend)
   ↓
3. Stripe API returns secure token (tok_xxxxxxxxx)
   ↓
4. EBS backend receives only token + metadata
   ↓
5. EBS stores token in encrypted payment_methods table
   ↓
6. For payments: EBS sends token to Stripe API with charge amount
   ↓
7. Stripe processes payment and returns transaction ID

Database Schema:

```
CREATE TABLE customer_payment_method (
  payment_method_id UUID PRIMARY KEY,
  customer_account_id UUID NOT NULL,
  stripe_payment_method_token VARCHAR(255) NOT NULL, -- tokenized
  card_brand VARCHAR(20), -- Visa, Mastercard, etc.
  last_four_digits CHAR(4), -- display purposes only
  expiration_month INTEGER,
  expiration_year INTEGER,
  is_default BOOLEAN DEFAULT false,
  created_at TIMESTAMP,
  -- NO STORAGE OF: full card number, CVV, PIN
  CONSTRAINT fk_customer FOREIGN KEY (customer_account_id)
    REFERENCES customer_account(account_id)
);
```

Security Controls:

- All API calls to Stripe use TLS 1.3
- Stripe tokens encrypted at rest in database (AES-256)
- Access to payment_methods table restricted to payment service only
- Audit logging for all payment method access
- Token rotation every 12 months

Dependencies:

- Stripe account setup complete with production API keys
- Frontend integration with Stripe.js library
- Security audit scheduled with QSA consultant

Definition of Done:

- Code passes security review by Security Lead
- Integration tested in staging environment
- QSA consultant confirms PCI-DSS SAQ-A eligibility
- Security documentation updated

Story EBS-76: Implement Late Fee Calculation

Story Type: User Story
Reporter: Marcus Rodriguez
Assignee: Developer-2 (Michael Chen)

Status: IN PROGRESS 🔄
Sprint: Sprint 14
Story Points: 5
Labels: `dunning`, `late-fees`, `billing`, `collections`

Story Description:

As a Billing Analyst, I want the system to automatically generate and apply a $5.00 Late Fee line item if a bill remains unpaid 15 days past the due date, so that late payment policies are enforced consistently and collection costs are recovered.

Business Rules:

- Late fee triggers exactly 15 days after payment due date
- Late fee amount: $5.00 (flat fee, not percentage-based)
- Late fee appears as separate line item on next invoice
- Maximum one late fee per billing cycle (no compounding late fees)
- Late fee waived if payment received within 24 hours of fee assessment
- Late fee subject to taxes like other charges

Acceptance Criteria:

🔄 **AC-1:** Automated job runs daily checking for invoices 15+ days past due date

🔄 **AC-2:** Late fee line item automatically created for qualifying invoices

🔄 **AC-3:** Late fee amount set to $5.00 with line item type "LATE_FEE"

⏳ **AC-4:** Late fee waiver logic implemented for payments received within 24-hour grace period

⏳ **AC-5:** Late fee appears on customer's next invoice with clear description

⏳ **AC-6:** Customer notification sent when late fee assessed

⏳ **AC-7:** Late fee revenue tracked separately for financial reporting

Calculation Logic:

```
Daily Job (runs at 02:00 AM):
  FOR EACH invoice WHERE:
    - invoice_status = 'UNPAID'
    - payment_due_date < (CURRENT_DATE - 15 days)
    - NOT EXISTS late_fee for this invoice

  DO:
    1. Create invoice_line_item:
```

```
   - line_item_type = 'LATE_FEE'
   - description = 'Late Payment Fee'
   - amount = $5.00
   - related_invoice_id = original_invoice_id

2. Update account balance:
   current_balance += $5.00

3. Create customer notification:
   - template = 'LATE_FEE_ASSESSED'
   - delivery_method = EMAIL + SMS

4. Log audit trail:
   - action = 'LATE_FEE_APPLIED'
   - reason = 'Payment overdue 15 days'
```

Edge Cases to Handle:

1. **Payment Plan Customers:** Late fees waived for customers on approved payment plans making scheduled payments
2. **Disputed Bills:** Late fees not assessed while billing dispute is under investigation
3. **Bankruptcy Protection:** Late fees not assessed for customers with active bankruptcy filing
4. **Regulatory Maximum:** Verify $5.00 fee complies with state regulations (some states cap late fees)

Test Scenarios:

- Invoice exactly 15 days overdue → late fee applies
- Invoice 14 days overdue → no late fee
- Invoice 30 days overdue → only one $5.00 late fee (not multiple)
- Payment received on day 15 before late fee job runs → no late fee
- Payment received day 16 within 24-hour grace period → late fee waived

Dependencies:

- Invoice data model supports line items with type LATE_FEE
- Customer notification service can send late fee notices
- Payment processing service can recognize late fee waivers

Current Progress:

- Core calculation logic implemented (85% complete)
- Unit tests written and passing
- Integration with notification service pending

Definition of Done:

- All acceptance criteria met
- Unit test coverage >90%
- Integration tests passing
- UAT sign-off from Finance team
- Customer-facing late fee notice template approved

Target Completion: October 10, 2025

Story EBS-SP1: Investigate Kafka Connectors for Data Replication

Story Type: Spike (Research/Investigation)
Reporter: Dr. Robert Kumar (Principal Architect)
Assignee: DevOps-Engineer-1 (Carlos Martinez)
Status: IN PROGRESS 🔄
Sprint: Sprint 14
Story Points: 3
Time Box: 3 days
Labels: `spike`, `research`, `kafka`, `data-replication`, `analytics`

Spike Description:

Research and proof-of-concept required to determine the best method for near-real-time replication of production billing data from PostgreSQL to S3 for business intelligence and analytics consumption. Need to evaluate Kafka Connect options and provide architectural recommendation.

Context:

Business Intelligence team requires read-only access to billing data for reporting and analytics. Direct queries against production database risk performance impact. Need streaming replication solution that:

- Minimizes impact on production database
- Provides near-real-time data availability (<5 minute latency)
- Supports schema evolution
- Cost-effective at scale (2.4M accounts, ~500GB data)

Investigation Tasks:

✅ **Task 1:** Research Kafka Connect PostgreSQL source connectors (Debezium vs. Confluent JDBC connector)

🔄 **Task 2:** Evaluate S3 sink connector options and data format (Parquet vs. Avro vs. JSON)

🔄 **Task 3:** Build minimal proof-of-concept with sample billing data

⏳ **Task 4:** Performance testing - measure replication lag and resource utilization

⏳ **Task 5:** Cost analysis - Kafka cluster sizing and AWS data transfer costs

Deliverables:

1. **Technical Recommendation Document:**
    - Recommended connector architecture
    - Configuration templates
    - Performance characteristics
    - Cost estimates
2. **Proof-of-Concept:**
    - Working connector configuration
    - Sample data successfully replicated to S3
    - Basic performance metrics
3. **Implementation Estimate:**
    - Story point estimate for full implementation
    - Risks and dependencies identified

Evaluation Criteria:

- Replication latency <5 minutes
- Production database CPU impact <5%
- S3 storage costs reasonable (<$200/month)
- Schema evolution supported without pipeline breakage
- Operational complexity manageable by DevOps team

Current Findings (In Progress):

- Debezium connector appears superior for change data capture (CDC)
- Parquet format best for analytics query performance in Athena
- Initial PoC shows <2 minute replication latency

Definition of Done:

- Recommendation document complete and reviewed
- PoC demonstrated to architecture review board
- Implementation stories created in backlog with estimates

Target Completion: October 8, 2025

This comprehensive Jira structure provides detailed tracking for all aspects of the EBS project. Would you like me to continue with the email communications and Slack conversation artifacts?Blocks:** EBS-E3 (Time-of-Use Billing), EBS-E5 (Invoice Generation Service)

- ⚠️ **Depends On:** EBS-E0 (Meter Data Ingestion Service) - COMPLETE

Story Breakdown:

- Foundation & Data Model: 4 stories (EBS-24 through EBS-27)
- Calculation Logic: 6 stories (EBS-28 through EBS-33)
- Integration & API: 4 stories (EBS-34 through EBS-37)
- Testing & Quality: 4 stories (EBS-38 through EBS-41)

Progress Metrics:

- Stories Complete: 8/18 (44%)
- Story Points Complete: 84/187 (45%)
- Estimated Completion: Sprint 16 (October 28, 2025)

Risks:

🔴 **HIGH RISK:** Performance testing has not yet validated 80K account processing capability. Current performance tests only reach 15K accounts. Need load testing infrastructure improvements.

🟡 **MEDIUM RISK:** Finance UAT resources not yet fully committed. Need formal sign-off from Finance Director by October 10 to maintain schedule.

Epic EBS-E2: Migrate and Verify Customer Account Data

Epic Name: Legacy Data Migration & ETL
Epic Owner: Lisa Thompson (Data Engineering Tech Lead)
Reporter: Marcus Rodriguez (Product Owner)
Status: IN PROGRESS
Priority: Highest
Labels: `data-migration`, `etl`, `critical-path`, `legacy-integration`

Epic Description:

This Epic tracks the extraction, transformation, loading (ETL), and comprehensive verification of all customer account data, billing history, payment history, and service address information from the legacy mainframe system (BillCalc v3.1 on IBM AS/400) to the new cloud-based EBS platform.

The migration must handle approximately 2.48 million customer accounts, 36 million historical invoice records (3 years of history), 89 million payment transaction records, and associated master data. Data quality validation discovered that approximately 15% of legacy records contain incomplete, inconsistent, or corrupted data requiring remediation before migration.

Migration Scope:

Phase 1 - Active Customer Master Data (CURRENT SPRINT):

- 2.35 million active customer accounts (95% of total)
- Customer demographics: name, contact information, account status
- Service address information and geo-coding
- Rate plan assignments and enrollment dates
- Program enrollments (CARE, medical baseline, etc.)
- Payment method preferences and auto-pay configurations

Phase 2 - Historical Billing Data (Sprint 15-16):

- 36 months of invoice history per regulatory requirements
- Invoice header records: dates, amounts, payment status
- Invoice line item detail for audit trail
- Billing cycle history and rate plan changes

Phase 3 - Payment Transaction History (Sprint 16-17):

- All payment transactions for active accounts (past 36 months)
- Payment method details (anonymized card numbers, bank account info)
- Payment application records (how payments applied to invoices)
- Adjustment and refund transaction history

Phase 4 - Inactive/Archived Accounts (Post-Cutover):

- 130,000 inactive accounts (5% of total)
- Lower priority, will migrate to separate archive database
- No immediate cutover dependency

Business Value:

Successful data migration is an absolute prerequisite for production cutover. Without accurate historical data, customers cannot access billing history, payment plans cannot continue, and regulatory reporting requirements cannot be met. This Epic directly impacts project go-live timeline.

Data Quality Challenges Discovered:

During initial ETL development, significant data quality issues were discovered in the legacy system:

1. **Incomplete Service Addresses (12% of records):**
    - Missing postal codes or apartment numbers
    - Inconsistent address formatting
    - Geographic coordinates missing for 8% of addresses
2. **Orphaned Records (3% of data):**
    - Customer accounts with no linked service address
    - Payment records with invalid customer account references
    - Invoice records with missing line item details
3. **Data Type Inconsistencies (5% of fields):**
    - Numeric fields stored as text with embedded formatting
    - Date fields with inconsistent formats (YYMMDD vs MM/DD/YYYY)
    - Currency values with missing or inconsistent decimal places
4. **Duplicate Records (2% of accounts):**
    - Same customer with multiple account numbers
    - Requires merge logic and manual review for conflict resolution

Acceptance Criteria (Epic-Level):

✅ **AC-1:** ETL scripts successfully extract all data from legacy AS/400 mainframe system using JDBC connectivity

✅ **AC-2:** Data validation framework identifies and reports all data quality issues with detailed error codes

🔄 **AC-3:** Transformation logic successfully maps all legacy data structures to new EBS schema

🔄 **AC-4:** Data cleansing procedures remediate or quarantine all problematic records

🔄 **AC-5:** Reconciliation reports confirm 100% of critical account data migrated successfully (allowing for planned exclusion of inactive accounts)

⏳ **AC-6:** Financial reconciliation validates that sum of all migrated billing amounts matches legacy system totals within 0.01% tolerance

⏳ **AC-7:** Spot-check validation of 1,000 randomly selected customer accounts confirms data accuracy at >99.5% field-level accuracy

⏳ **AC-8:** Performance testing confirms full migration completes within 48-hour cutover window

⏳ **AC-9:** Rollback procedures tested and documented in case migration must be reversed

⏳ **AC-10:** Data migration runbook completed and approved by PMO

Technical Architecture:

- **ETL Tool:** Apache NiFi for orchestration, custom Python scripts for complex transformations
- **Staging Database:** PostgreSQL instance for data quality validation
- **Data Quality Framework:** Great Expectations for validation rules
- **Source System:** IBM AS/400 mainframe, COBOL data files via JDBC bridge
- **Target System:** AWS RDS PostgreSQL 15 (production EBS database)

Associated Stories: 12 stories, 154 story points total