Project Report

CS3041: Information Management II

1. Overview

1.1 Concept

For this project I have designed a database for a customer relationship management (CRM) system to be used by energy providers. This project is based on work I previously did for my current employer who tasked me with developing a prototype system for a new customer. The completed system is now in use.

1.2 Design

The structure of the CRM can essentially be broken down into four parts: users, customers, meters and invoices.

1.2.1 Users

The energy provider has a number of users who can all access the CRM. Users can be of different types: *administrator*, *general worker*, *tester*, etc. Each user has an email, username and password. Users also have a status (*active*, *inactive* or *deleted*).

1.2.2 Customers

The energy provider has a number of customers whom it provides energy to. Each customer can have one or more contacts. A customer can also have a primary contact (flagged by the `IsPrimaryContact` column).

Each contact has an email, first and last name, phone number and address and each address has two address lines (the latter of which is optional), a county and an Eircode.

Since meters also have addresses (see *§1.2.3*) each address also has a `IsMeter` flag. Like users, customers also have a status (*active*, *inactive* or *deleted*).

1.2.3 Meters

Each customer can have many energy meters. Each meter has an external identifier (used by the *Electricity Supply Board* to identify it) and an address. Meters also have a type (*electricity*, *gas*, etc.) and a status (*active*, *inactive*, *deleted* or *awaiting approval*).

Each type of meter has a unit of measurement (e.g., cubic metres for gas and kilowatt hours for electricity) and these units have a conversion factor to the kilowatt hour standard.

Each meter will have many readings which contain a reading timestamp and an energy usage value.

1.2.4 Invoices

The energy provider can bill its customers for their energy usage over a specific time period - this is referred to as a *billing run*. Each billing run has a name, start and end date (to define the time period) and a status (*completed*, *in progress*, *not started*, *cancelled*, *deleted*).

Each billing run will generate a number of invoices and each invoice is for a specific customer and has an amount-to-charge and a status (*sent*, *approved*, *rejected*, *awaiting approval*, *cancelled*, *deleted*). Each invoice involves a number of the customer's meters so these are stored. The latest market prices (i.e. from kWh to Euros) are also stored.

When an invoice is approved an email is generated to send to the customer - the contents of this email, as well as the recipient's email address are stored in a table.

2. Tables

2.1 Important Notes

- Every table contains an `Id` column which acts as its primary key. This column is not related to the data in any way (phone number, social security number, etc.) for obvious reasons.
- Every table contains a `DataCreated` and `DateModified` column and a `TR_{table-name}_DC`, `TR_{table-name}_DM` trigger which handle the creation and updating of these columns. In my experience working with complex databases I've found these columns to be very important in maintaining data integrity and investigating bugs/bad data.
- Reference tables are used throughout the project as they are far more effective and tracking static properties like types or statuses as integers instead of strings. They also make the name-changing of these properties quicker, easier and safer.
- Dummy/test data have been added to each of the tables.

2.2 Users

The `User` table contains a `UserTypeId`, `UserStatusId`, `Email`, `Username`, `PasswordSalt` and `Password` (which is prepended to and hashed with `PasswordSalt`). The table also contains two foreign key constraints (to `UserType` and `UserStatus`) and a semantic constraint on the `Email` column to ensure it is formatted correctly.

The `UserType` and `UserStatus` tables are simply reference tables so both just contain the name of the type and status, respectively.

2.3 Customers

The `Customer` table contains a `CustomerStatusId` and `Name` as well as a foreign key constraint to `CustomerStatus`. The `CustomerStatus` table is a reference table.

The `Contact` table contains a `CustomerId`, an `IsPrimaryContact` flag, an `AddressId` and some basic contact information (name, email, etc.). It also contains two foreign key constraints (to `Customer` and `Address`) and semantic constraints on the `Email` and `Phone` columns, respectively.

The `Address` table contains an `IsMeter` flag, `AddressLine1`, `AddressLine2` (optional), `Eircode` and a `CountyId`. It contains a foreign key constraint to `County` and a semantic constraint on the `Eircode` column to check if it is either non-existent (null) or matches a particular regular expression.

The `County` table is a reference table containing the names of the Irish counties.

2.4 Meters

The `Meter` table contains a `MeterTypeId`, `MeterStatusId`, `CustomerId`, `AddressId` and `External Identifier`. As well as foreign key constraints to the following tables: `MeterTypeId`, `MeterStatusId`, `CustomerId` and `AddressId`. It also contains a semantic constraint on `External Identifier` which ensures that both gas and electricity meters match the *Electricity Supply Board*'s standard format. `MeterStatus` is a reference table.

The `MeterType` table contains a `Name` and `UnitId` as well as a foreign key constraint to the `Unit` table.

The `Unit` table contains a `Symbol`, `Name` and `ConversionFactor`. There is a unique constraint on the `Symbol` and `Name` columns and a check that ensures that `ConversionFactor` is non-negative. The `Symbol` column is simply the SI unit representation of the unit's name, e.g., *kilowatt hours* becomes *kWh*. The `ConversionFactor` column is the value one would need to multiply an energy amount of the current unit by to get the equivalent energy amount in kilowatt hours.

The `MeterReading` table contains a `MeterId`, `Timestamp` and `Value` as well as a foreign key constraint to the `Meter` table and a check that ensures that `Value` is non-negative.

2.5 Invoices

The `BillingRun` table contains a `BillingRunStatusId`, `Name`, `StartDate` and `EndDate` as well as a foreign key constraint to the `BillingRunStatus` table. `BillingRunStatus` is a reference table.

The `Invoice` table contains an `InvoiceStatusId`, `BillingRunId`, `CustomerId` and `Amount`. It contains foreign key constraints to the `InvoiceStatus`, `BillingRun` and `Customer` tables as well a check to ensure that `Amount` is non-negative. `InvoiceStatus` is a reference table.

The `InvoiceMeter` table contains an `InvoiceId` and `MeterId` as well as foreing key constraints to `Invoice` and `Meter`.

The `EmailQueue` table contains an `IsSent` flag, a `Recipient`, `Subject` and `Body` as well as a semantic constraint on `Email` to ensure it is formatted correctly.

The `MarketPrice` table contains a `MarketPriceTypeId`, `Timestamp` and `Value` as well as foreign key constraint to `MarketPriceType` and a check that ensures `Value` is non-negative. `MarketPriceType` is a reference table.

3. Views

3.1 ConvertedMeterReadingValue

This view simply converts meter reading values of all units to kilowatt hours. It returns the reading ID, timestamp and converted value as columns.

3.2 MeterUsageOverview

This view returns readable values for each meter reading including the original/converted values, the customer and meter unit names as well as the name of the meter type. It only uses active customers and meters.

3.3 MeterInvoiceIdPairs

This view selects distinct sets of meters for a particular invoice in a particular billing run. It only includes invoices that are awaiting approval and only includes active customers and meters. It also only selects meters that have readings between the start and end dates of the billing run.

3.4 InvoiceEmailContent

This view selects data that will be needed to generate the invoice email. It includes aggregated meter data from the `MeterUsageOverview` view as well as invoice data.

4. Stored Procedures

4.1 GetBillingRunDates

This procedure returns the start and end dates for a particular billing run.

4.2 GenerateBillingRunInvoices

This procedure calculates the amount-to-charge for each customer for a particular billing run using the `MeterUsageOverview` view and inserts these rows into the `Invoice` table. It also inserts relevant rows into the `InvoiceMeter` table using the `MeterInvoiceIdPairs` view. It makes use of variables to store the market price.

4.3 SendInvoiceEmail

This procedure generates a row in the `EmailQueue` table for a particular invoice. It makes heavy use of built-in string and date functions as well as advanced MySQL features such as cursors, loops and variables.

5. Functions & Triggers

5.1 Functions

5.1.1 GetMarketPrice(MarketPriceTypeId)

This function returns the latest market price value for a particular market price type.

5.2 Triggers

5.2.1 DateCreated/DateModified

See *§2.1*.

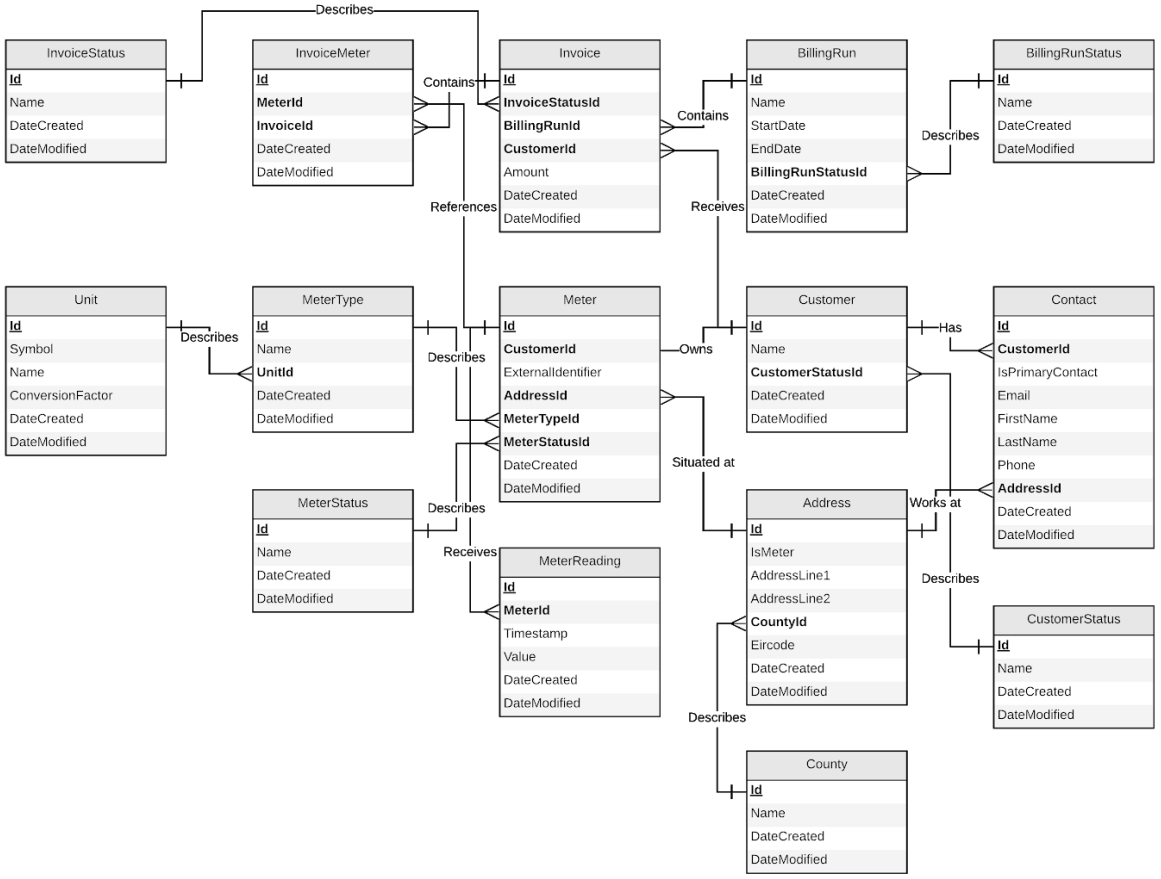
5.2.2 InvoiceApproved

This is triggered when an `Invoice` has its `InvoiceStatusId` changed to `2` (*approved*). It calls the `SendInvoiceEmail` stored procedure.

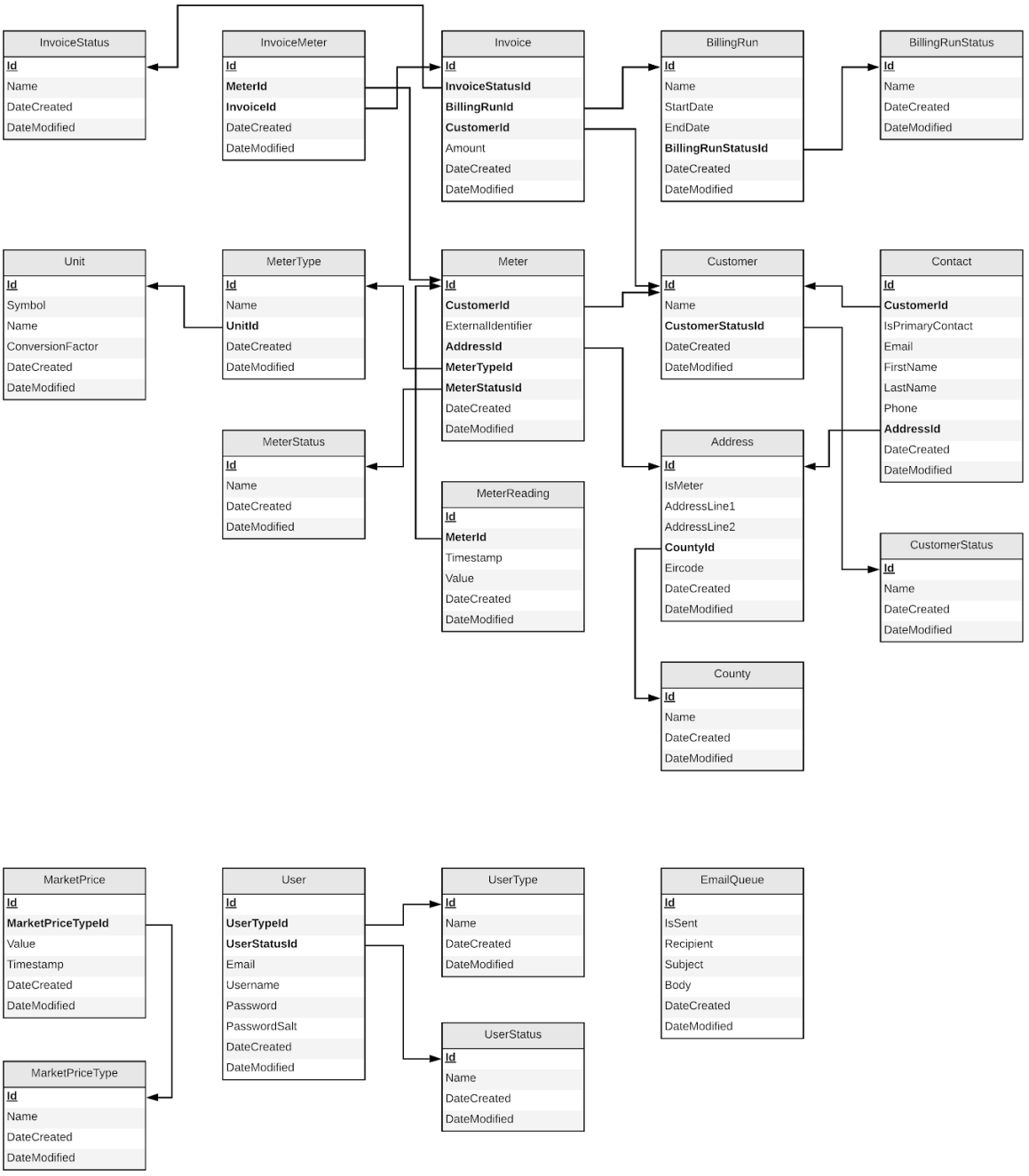
6. Diagrams

Keys: **primary**, **foreign**.

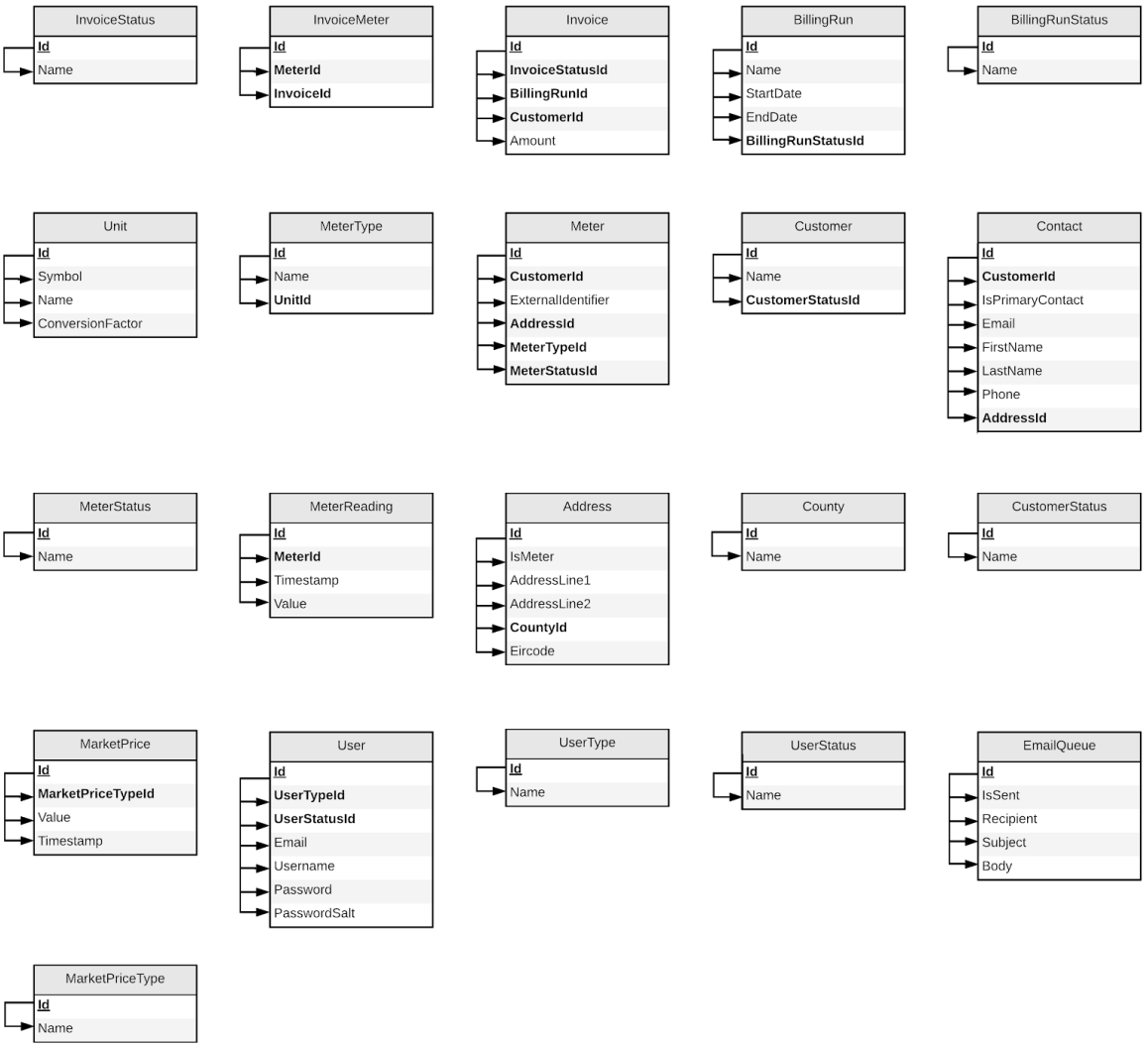
6.1 Entity Relationship Diagram



6.2 Relational Schema



6.3 Functional Dependency Diagram



7. Normalisation

The database is evidently normalised (according to the standard definitions) for each of the following forms:

- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce-Codd Normal Form