CPSC 304 Project Cover Page

Milestone #: 2

Date: July 21, 2024

Group Number: 32

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Aditya Goel	84873279	c3a6w	agoel25@student.ubc.ca
Ashmit Gupta	38002341	h6e5w	ashmitg229@gmail.com
Kaz Tahara-Edmonds	50257658	b1v9c	kazkte@gmail.com

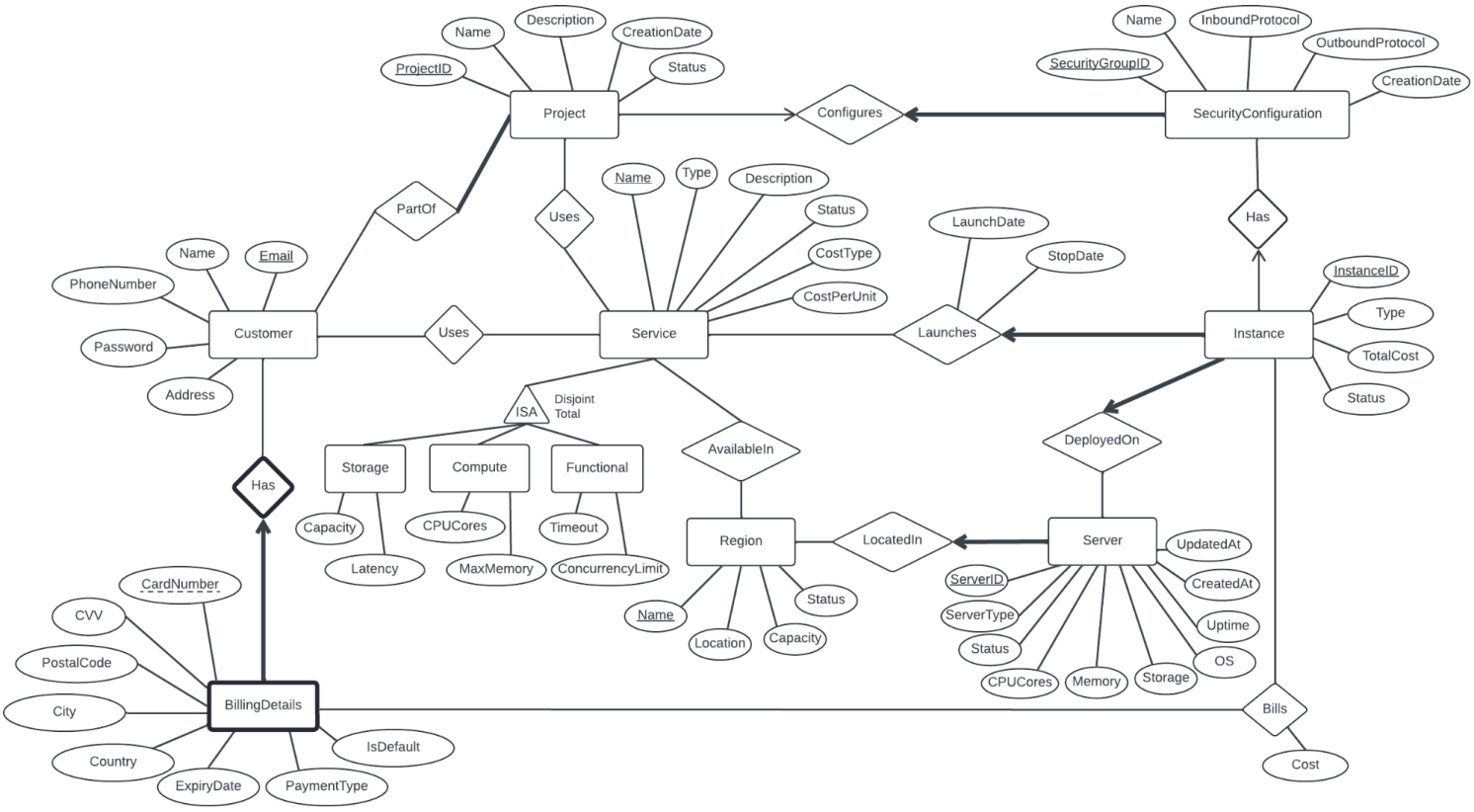
By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

2. A brief (~2-3 sentences) summary of your project

Our project models a cloud service provider. Customers will have the capability to create and manage projects. They will be able to utilize services and manage instances for the services they use. Furthermore, it will enable users to configure security settings, monitor server resources, and compute the costs associated with using cloud services.

3. The ER diagram you are basing your item #3 (below) on. This ER diagram may be the same as your milestone 1 submission or it might be different.



Changes to the ER diagram

Name changes:

1. Server's attribute "CPU" has been renamed to "CPUCores" to improve clarity
2. Launches' attribute "LaunchDateTime" has been renamed to "LaunchDate" and its data type was changed from DATETIME to DATE. This was done because we want our services to charge customers per date and not per hour.

3. Launches' attribute "StopDateTime" has been renamed to "StopDate" and its data type was changed from DATETIME to DATE. This was done because we want our services to charge customers per date and not per hour.

Structure changes:

1. Removed "BillingAddress" attribute from BillingDetails entity and added "City" and "Country" attributes in BillingDetails entity. This was done to better mimic what actual businesses use to charge customers.

4. The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:

a. List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.

1. Customer

Customer(Email: VARCHAR(30), Name: VARCHAR(30), PhoneNumber: VARCHAR(15), Password: VARCHAR(30), Address: VARCHAR(50))

- PK: Email
- CK: PhoneNumber
- Unique: PhoneNumber
- Not Null: PhoneNumber, Name, Password, Address

2. PartOf

PartOf(**Email**: VARCHAR(30), **ProjectID**: INTEGER)

- PK: Email, ProjectID
- FK:
  - Email references Customer(Email)
  - ProjectID references Project(ProjectID)

3. Project

Project(ProjectID: INTEGER, Name: VARCHAR(30), Description: VARCHAR(100), CreationDate: DATE, Status: VARCHAR(20))

- PK: ProjectID
- Not Null: CreationDate

4. BillingDetails

BillingDetails(CardNumber: INTEGER, **Email**: VARCHAR(30), CVV: INTEGER, PostalCode: VARCHAR(15), City: VARCHAR(20), Country: VARCHAR(20), ExpiryDate: DATE, PaymentType: VARCHAR(6), IsDefault: VARCHAR(5))

- PK: CardNumber, Email
- FK: Email references Customer(Email)
- Not Null: CVV, PostalCode, ExpiryDate, City, Country, PaymentType

5. CustomerUsesService

CustomerUsesService(**Email**: VARCHAR(30), **Name**: VARCHAR(30))

- PK: Email, Name
- FK:
  - Email references Customer(Email)
  - Name references Service(Name)

6. ProjectUsesService

ProjectUsesService(**ProjectID**: INTEGER, **Name**: VARCHAR(30))

- PK: ProjectID, Name
- FK:
  - ProjectID references Project(ProjectID)
  - Name references Service(Name)

7. Service

Service(Name: VARCHAR(30), Type: VARCHAR(20), Description: VARCHAR(100), Status: VARCHAR(20), CostType: VARCHAR(20), CostPerUnit: INTEGER)

- PK: Name
- Not Null: Type, Description, CostType, CostPerUnit

8. Storage

Storage(**Name**: VARCHAR(30), Capacity: INTEGER, Latency: INTEGER)

- PK: Name
- FK: Name references Service(Name)

9. Compute

Compute(**Name**: VARCHAR(30), CPUCores: INTEGER, MaxMemory: INTEGER)

- PK: Name
- FK: Name references Service(Name)

10. Functional

Functional(**Name**: VARCHAR(30), Timeout: INTEGER, ConcurrencyLimit: INTEGER)

- PK: Name
- FK: Name references Service(Name)

11. Region

Region(Name: VARCHAR(30), Location: VARCHAR(20), Capacity: INTEGER, Status: VARCHAR(20))

- PK: Name
- Not Null: Location

12. AvailableIn

AvailableIn(**ServiceName**: VARCHAR(30), **RegionName**: VARCHAR(30))

- PK: ServiceName, RegionName
- FK:
  - ServiceName references Service(Name)
  - RegionName references Region(Name)

13. SecurityConfiguration

SecurityConfiguration(SecurityGroupID: INTEGER, **ProjectID**: INTEGER, Name: VARCHAR(30), InboundProtocol: VARCHAR(20), OutboundProtocol: VARCHAR(20), CreationDate: DATE)

- PK: SecurityGroupID
- CK: ProjectID
- FK:
  - ProjectID references Project(ProjectID)
- Not Null: ProjectID
- Unique: ProjectID

14. Server

Server(ServerID: INTEGER, **Name**: VARCHAR(30), ServerType: VARCHAR(30), Status: VARCHAR(20), CPUCores: INTEGER, Memory: INTEGER, Storage: INTEGER, OS: VARCHAR(20), Uptime: INTEGER, CreatedAt: DATE, UpdatedAt: DATE)

- PK: ServerID
- FK:
  - Name references Region(Name)
- Unique: ServerType
- Not Null: Name, ServerType, Memory, Storage

15. Instance

Instance(InstanceID: INTEGER, **Name**: VARCHAR(30), **ServerID**: INTEGER, **SecurityGroupID**: INTEGER, Type: VARCHAR(20), TotalCost: INTEGER, Status: VARCHAR(20), LaunchDate: DATE, StopDate: DATE)

- PK: InstanceID
- FK:
  - Name references Service(Name)
  - ServerID references Server(ServerID)
  - SecurityGroupID references SecurityConfiguration(SecurityGroupID)
- Not Null: Name, ServerID

16. Bills

Bills(**InstanceID**: INTEGER, **Email**: VARCHAR(30), **CardNumber**: INTEGER, Cost: INTEGER)

- PK: InstanceID, Email, CardNumber
- FK:
  - InstanceID references Instance(InstanceID)
  - Email references BillingDetails(Email)
  - CardNumber references BillingDetails(CardNumber)

5. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key). PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A → A.

1. Customer
    - Email → Name, PhoneNumber, Password, Address
    - PhoneNumber → Email, Name, Password, Address
2. PartOf
    - No non-trivial functional dependencies
3. Project
    - ProjectID → Name, Description, CreationDate, Status
4. BillingDetails
    - CardNumber, Email → CVV, PostalCode, City, Country, ExpiryDate, PaymentType, IsDefault
    - PostalCode → City, Country
    - CardNumber → PaymentType
5. CustomerUsesService
    - No non-trivial functional dependencies
6. ProjectUsesService
    - No non-trivial functional dependencies
7. Service
    - Name → Type, Description, Status, CostType, CostPerUnit
    - Type → CostType
8. Storage
    - Name → Capacity, Latency
9. Compute
    - Name → CPUCores, MaxMemory
10. Functional
    - Name → Timeout, ConcurrencyLimit
11. Region
    - Name → Location, Capacity, Status
12. AvailableIn
    - No non-trivial functional dependencies
13. SecurityConfiguration
    - SecurityGroupID → ProjectID, Name, InboundProtocol, OutboundProtocol, CreationDate
    - ProjectID → SecurityGroupID
    - InboundProtocol → OutboundProtocol
14. Server
    - ServerID → Name, ServerType, Status, CPUCores, Memory, Storage, OS, Uptime, CreatedAt, UpdatedAt
    - ServerType → Memory, Storage, CPUCores, OS
    - Memory, Storage → ServerType
15. Instance

    - InstanceID → Name, ServerID, SecurityGroupID, Type, TotalCost, Status, LaunchDate, StopDate
16. Bills
    - InstanceID, Email, CardNumber → Cost

6. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization.

You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, …). ALL Tables must be listed, not only the ones post normalization.

1. Customer(Email: VARCHAR(30), Name: VARCHAR(30), PhoneNumber: VARCHAR(15), Password: VARCHAR(30), Address: VARCHAR(50))

- PK: Email
- CK: PhoneNumber
- Unique: PhoneNumber
- Not Null: PhoneNumber, Name, Password, Address
- FD:
    - Email → Name, PhoneNumber, Password, Address
    - PhoneNumber → Email, Name, Password, Address

LHS of both functional dependencies are keys, so already in BCNF

2. PartOf(**Email**: VARCHAR(30), **ProjectID**: INTEGER)

- PK: Email, ProjectID
- FK:
    - Email references Customer(Email)
    - ProjectID references Project(ProjectID)
- FD: No non-trivial FDs

Already in BCNF

3. Project(ProjectID: INTEGER, Name: VARCHAR(30), Description: VARCHAR(100), CreationDate: DATE, Status: VARCHAR(20))

- PK: ProjectID
- Not Null: CreationDate
- FD:

    - ProjectID → Name, Description, CreationDate, Status

LHS of the functional dependency is a key so already in BCNF

4. BillingDetails(CardNumber: INTEGER, **Email**: VARCHAR(30), CVV: INTEGER, PostalCode: VARCHAR(15), City: VARCHAR(20), Country: VARCHAR(20), ExpiryDate: DATE, PaymentType: VARCHAR(6), IsDefault: VARCHAR(5))

- PK: CardNumber, Email
- FK: Email references Customer(Email)
- Not Null: CVV, PostalCode, ExpiryDate, City, Country, PaymentType
- FD:
    - CardNumber, Email → CVV, PostalCode, City, Country, ExpiryDate, PaymentType, IsDefault
    - PostalCode → City, Country
    - CardNumber → PaymentType

Normalized in BCNF:

1. PaymentInfo(**CardNumber**: INTEGER, **Email**: VARCHAR(30), CVV: INTEGER, **PostalCode**: VARCHAR(15), ExpiryDate: DATE, IsDefault: VARCHAR(5))
    a. PK: CardNumber, Email
    b. FK:
        i. Email references Customer(Email)
        ii. CardNumber references BillingInfo(CardNumber)
        iii. PostalCode references AddressInfo(PostalCode)
    c. Not Null: CVV, PostalCode, ExpiryDate, City, Country
2. AddressInfo(PostalCode: VARCHAR(15), City: VARCHAR(20), Country: VARCHAR(20))
    a. PK: PostalCode
    b. Not Null: City, Country
3. BillingInfo(CardNumber: INTEGER, PaymentType: VARCHAR(6))
    a. PK: CardNumber
    b. Not Null: PaymentType

5. CustomerUsesService(**Email**: VARCHAR(30), **Name**: VARCHAR(30))

- PK: Email, Name
- FK:
    - Email references Customer(Email)
    - Name references ServiceDetails(Name)

Already in BCNF

6. ProjectUsesService(**ProjectID**: INTEGER, **Name**: VARCHAR(30))

- PK: ProjectID, Name
- FK:
    - ProjectID references Project(ProjectID)
    - Name references ServiceDetails(Name)
- FD: No non-trivial functional dependencies

Already in BCNF

7. Service(Name: VARCHAR(30), Type: VARCHAR(20), Description: VARCHAR(100), Status: VARCHAR(20), CostType: VARCHAR(20), CostPerUnit: INTEGER)

- PK: Name
- Not Null: Type, Description, CostType, CostPerUnit
- FD:
    - Name → Type, Description, Status, CostType, CostPerUnit
    - Type → CostType

Normalized in BCNF:

1. ServiceType(Type: VARCHAR(20), CostType: VARCHAR(20))
    a. PK: Type
    b. Not Null: CostType
2. ServiceDetails(Name: VARCHAR(30), **Type**: VARCHAR(20), Description: VARCHAR(100), Status: VARCHAR(20), CostPerUnit: INTEGER)
    a. PK: Name
    b. FK:
        i. Type references ServiceType(Type)
    c. Not Null: Type, Description, CostPerUnit

8. Storage(**Name**: VARCHAR(30), Capacity: INTEGER, Latency: INTEGER)

- PK: Name
- FK: Name references ServiceDetails(Name)
- FD:
    - Name → Capacity, Latency

LHS of the functional dependency is a key so already in BCNF

9. Compute(**Name**: VARCHAR(30), CPUCores: INTEGER, MaxMemory: INTEGER)

- PK: Name
- FK: Name references ServiceDetails(Name)
- FD:
    - Name → CPUCores, MaxMemory

LHS of the functional dependency is a key so already in BCNF

10. Functional(**Name**: VARCHAR(30), Timeout: INTEGER, ConcurrencyLimit: INTEGER)

- PK: Name
- FK: Name references ServiceDetails(Name)
- FD:
    - Name → Timeout, ConcurrencyLimit

LHS of the functional dependency is a key so already in BCNF

11. Region(Name: VARCHAR(30), Location: VARCHAR(20), Capacity: INTEGER, Status: VARCHAR(20))

- PK: Name
- Not Null: Location
- FD
    - Name → Location, Capacity, Status

LHS of the functional dependency is a key so already in BCNF

12. AvailableIn(**ServiceName**: VARCHAR(30), **RegionName**: VARCHAR(30))

- PK: ServiceName, RegionName
- FK:
    - ServiceName references ServiceDetails(Name)
    - RegionName references Region(Name)
- FD: No non-trivial FD's

Already in BCNF

13. SecurityConfiguration(SecurityGroupID: INTEGER, **ProjectID**: INTEGER, Name: VARCHAR(30), InboundProtocol: VARCHAR(20), OutboundProtocol: VARCHAR(20), CreationDate: DATE)

- PK: SecurityGroupID
- CK: ProjectID

- FK:
    - ProjectID references Project(ProjectID)
- Not Null: ProjectID
- Unique: ProjectID
- FD
    - SecurityGroupID → ProjectID, Name, InboundProtocol, OutboundProtocol, CreationDate
    - ProjectID → SecurityGroupID
    - InboundProtocol → OutboundProtocol

Normalized in BCNF:

1. ProjectSecurity(**ProjectID**: INTEGER, SecurityGroupID: INTEGER)
    a. PK: ProjectID
    b. CK: SecurityGroupID
    c. FK: ProjectID references Project(ProjectID)
    d. Not Null: SecurityGroupID
    e. Unique: SecurityGroupID
2. SecurityProtocol(InboundProtocol: VARCHAR(20), OutboundProtocol: VARCHAR(20))
    a. PK: InboundProtocol
3. SecurityInfo(**ProjectID**: INTEGER, Name: VARCHAR(30), **InboundProtocol**: VARCHAR(20), CreationDate: DATE)
    a. PK: ProjectID
    b. FK:
        i. ProjectID references ProjectSecurity(ProjectID)
        ii. InboundProtocol references SecurityProtocol(InboundProtocol)

14. Server(ServerID: INTEGER, **Name**: VARCHAR(30), ServerType: VARCHAR(30), Status: VARCHAR(20), CPUCores: INTEGER, Memory: INTEGER, Storage: INTEGER, OS: VARCHAR(20), Uptime: INTEGER, CreatedAt: DATE, UpdatedAt: DATE)

- PK: ServerID
- FK:
    - Name references Region(Name)
- Unique: ServerType
- FD:
    - ServerID → Name, ServerType, Status, CPUCores, Memory, Storage, OS, Uptime, CreatedAt, UpdatedAt
    - ServerType → Memory, Storage, CPUCores, OS
    - Memory, Storage → ServerType

Normalized in BCNF:

1. ServerTypeInfo(ServerType: VARCHAR(30), Memory: INTEGER, Storage: INTEGER, OS: VARCHAR(20), CPUCores: INTEGER)
    a. PK: ServerType
2. ServerInfo(ServerID: INTEGER, **Name**: VARCHAR(30), **ServerType**: VARCHAR(30), Status: VARCHAR(20), Uptime: INTEGER, CreatedAt: DATE, UpdatedAt: DATE)
    a. PK: ServerID
    b. FK:
        i. Name references Region(Name)
        ii. ServerType references ServerTypeInfo(ServerType)

15. Instance(InstanceID: INTEGER, **Name**: VARCHAR(30), **ServerID**: INTEGER, **ProjectID**: INTEGER, Type: VARCHAR(20), TotalCost: INTEGER, Status: VARCHAR(20), LaunchDate: DATE, StopDate: DATE)

- PK: InstanceID
- FK:
    - Name references ServiceDetails(Name)
    - ServerID references ServerInfo(ServerID)
    - ProjectID references ProjectSecurity(ProjectID)
- Not Null: Name, ServerID
- FD:
    - InstanceID → Name, ServerID, ProjectID, Type, TotalCost, Status, LaunchDate, StopDate

LHS of the functional dependency is a key so already in BCNF

16. Bills(**InstanceID**: INTEGER, **Email**: VARCHAR(30), **CardNumber**: INTEGER, Cost: INTEGER)

- PK: InstanceID, Email, CardNumber
- FK:
    - InstanceID references Instance(InstanceID)
    - Email references PaymentInfo(Email)
    - CardNumber references BillingInfo(CardNumber)
- FD:
    - InstanceID, Email, CardNumber → Cost

LHS of the functional dependency is a key so already in BCNF

7. The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc. Unless you know that you will always have exactly x characters for a given character, it is better to use the VARCHAR data type as opposed to a VARCHAR(Y). For example, UBC courses always use four characters to represent which department offers a course. In that case, you will want to use VARCHAR(4) for the department attribute in your SQL DDL statement. If you are trying to represent the name of a UBC course, you will want to use VARCHAR as the number of characters in a course name can vary greatly.

1.

```
CREATE TABLE Customer (
        Email VARCHAR(30) PRIMARY KEY,
        Name VARCHAR(30) NOT NULL,
        PhoneNumber VARCHAR(15) NOT NULL UNIQUE,
        Password VARCHAR(30) NOT NULL,
        Address VARCHAR(50) NOT NULL
);
```

2.

```
CREATE TABLE PartOf (
        Email VARCHAR(30),
        ProjectID INTEGER,
        PRIMARY KEY (Email, ProjectID),
        FOREIGN KEY (Email) REFERENCES Customer(Email)
                ON DELETE NO ACTION,
        FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
                ON DELETE CASCADE,
);
```

3.

```
CREATE TABLE Project (
        ProjectID INTEGER PRIMARY KEY,
        Name VARCHAR(30),
        Description VARCHAR(100),
        CreationDate DATE NOT NULL,
        Status VARCHAR(20)
);
```

4.

```
CREATE TABLE PaymentInfo (
        CardNumber INTEGER,
        Email VARCHAR(30),
        CVV INTEGER NOT NULL,
        PostalCode VARCHAR(15) NOT NULL,
        ExpiryDate DATE NOT NULL,
        IsDefault VARCHAR(5),
        PRIMARY KEY (CardNumber, Email),
        FOREIGN KEY (Email) REFERENCES Customer(Email)
                ON DELETE CASCADE,
        FOREIGN KEY (CardNumber) REFERENCES BillingInfo(CardNumber)
                ON DELETE CASCADE,
        FOREIGN KEY (PostalCode) REFERENCES AddressInfo(PostalCode)
                ON DELETE CASCADE,
);
```

5.

```
CREATE TABLE AddressInfo (
        PostalCode VARCHAR(15) PRIMARY KEY,
        City VARCHAR(20) NOT NULL,
        Country VARCHAR(20) NOT NULL
);
```

6.

```
CREATE TABLE BillingInfo (
        CardNumber INTEGER PRIMARY KEY,
        PaymentType VARCHAR(6) NOT NULL
);
```

7.

```
CREATE TABLE CustomerUsesService (
        Email VARCHAR(30),
        Name VARCHAR(30),
        PRIMARY KEY (Email, Name),
        FOREIGN KEY (Email) REFERENCES Customer(Email)
                ON DELETE CASCADE,
        FOREIGN KEY (Name) REFERENCES ServiceDetails(Name)
                ON DELETE CASCADE
);
```

8.

```
CREATE TABLE ProjectUsesService (
        ProjectID INTEGER,
        Name VARCHAR(30),
        PRIMARY KEY (ProjectID, Name),
        FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
                ON DELETE CASCADE,
        FOREIGN KEY (Name) REFERENCES ServiceDetails(Name)
                ON DELETE CASCADE
);
```

9.

```
CREATE TABLE ServiceType (
        Type VARCHAR(20) PRIMARY KEY,
        CostType VARCHAR(20) NOT NULL
);
```

10.

```
CREATE TABLE ServiceDetails (
        Name VARCHAR(30) PRIMARY KEY,
        Type VARCHAR(20) NOT NULL DEFAULT “Service”,
        Description VARCHAR(100) NOT NULL,
        Status VARCHAR(20),
        CostPerUnit INTEGER NOT NULL,
        FOREIGN KEY (Type) REFERENCES ServiceType(Type)
                ON DELETE SET DEFAULT
);
```

11.

```
CREATE TABLE Storage (
        Name VARCHAR(30) PRIMARY KEY,
        Capacity INTEGER,
        Latency INTEGER,
        FOREIGN KEY (Name) REFERENCES ServiceDetails(Name)
                ON DELETE CASCADE
);
```

12.

```
CREATE TABLE Compute (
        Name VARCHAR(30) PRIMARY KEY,
```

```
        CPUCores INTEGER,
        MaxMemory INTEGER,
        FOREIGN KEY (Name) REFERENCES ServiceDetails(Name)
                ON DELETE CASCADE
);
```

13.

```
CREATE TABLE Functional (
        Name VARCHAR(30) PRIMARY KEY,
        Timeout INTEGER,
        ConcurrencyLimit INTEGER,
        FOREIGN KEY (Name) REFERENCES ServiceDetails(Name)
                ON DELETE CASCADE
);
```

14.

```
CREATE TABLE Region (
        Name VARCHAR(30) PRIMARY KEY,
        Location VARCHAR(20) NOT NULL,
        Capacity INTEGER,
        Status VARCHAR(20)
);
```

15.

```
CREATE TABLE AvailableIn (
        ServiceName VARCHAR(30),
        RegionName VARCHAR(30),
        PRIMARY KEY (ServiceName, RegionName),
        FOREIGN KEY (ServiceName) REFERENCES ServiceDetails(Name)
                ON DELETE CASCADE,
        FOREIGN KEY (RegionName) REFERENCES Region(Name)
                ON DELETE CASCADE
);
```

16.

```
CREATE TABLE ProjectSecurity (
        ProjectID INTEGER PRIMARY KEY,
        SecurityGroupID INTEGER UNIQUE NOT NULL,
        FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
                ON DELETE CASCADE
```

```
);

17.
CREATE TABLE SecurityProtocol (
        InboundProtocol VARCHAR(20) PRIMARY KEY,
        OutboundProtocol VARCHAR(20)
);

18.
CREATE TABLE SecurityInfo (
        ProjectID INTEGER PRIMARY KEY,
        Name VARCHAR(30),
        InboundProtocol VARCHAR(20) DEFAULT “HTTPS”,
        CreationDate DATE,
        FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
                ON DELETE CASCADE,
        FOREIGN KEY (InboundProtocol) REFERENCES SecurityProtocol(InboundProtocol)
                ON DELETE SET DEFAULT
);

19.
CREATE TABLE ServerTypeInfo (
        ServerType VARCHAR(30) PRIMARY KEY,
        Memory INTEGER,
        Storage INTEGER,
        OS VARCHAR(20),
        CPUCores INTEGER
);

20.
CREATE TABLE ServerInfo (
        ServerID INTEGER PRIMARY KEY,
        Name VARCHAR(30),
        ServerType VARCHAR(30),
        Status VARCHAR(20),
        Uptime INTEGER,
        CreatedAt DATE,
        UpdatedAt DATE,
        FOREIGN KEY (Name) REFERENCES Region(Name)
                ON DELETE CASCADE,
```

```
        FOREIGN KEY (ServerType) REFERENCES ServerTypeInfo(ServerType)
                ON DELETE CASCADE
);
```

21.

```
CREATE TABLE Instance (
        InstanceID INTEGER PRIMARY KEY,
        Name VARCHAR(30) NOT NULL,
        ServerID INTEGER NOT NULL,
        ProjectID INTEGER,
        Type VARCHAR(20),
        TotalCost INTEGER,
        Status VARCHAR(20),
        LaunchDate DATE,
        StopDate DATE,
        FOREIGN KEY (Name) REFERENCES ServiceDetails(Name)
                ON DELETE CASCADE,
        FOREIGN KEY (ServerID) REFERENCES ServerInfo(ServerID)
                ON DELETE CASCADE,
        FOREIGN KEY (ProjectID) REFERENCES ProjectSecurity(ProjectID)
                ON DELETE SET NULL
);
```

22.

```
CREATE TABLE Bills (
        InstanceID INTEGER DEFAULT “DELETED_INSTANCE”,
        Email VARCHAR(30) DEFAULT “DELETED_USER”,
        CardNumber INTEGER DEFAULT “DELETED_CARD”,
        Cost INTEGER,
        PRIMARY KEY (InstanceID, Email, CardNumber),
        FOREIGN KEY (InstanceID) REFERENCES Instance(InstanceID)
                ON DELETE SET DEFAULT,
        FOREIGN KEY (Email) REFERENCES PaymentInfo(Email)
                ON DELETE SET DEFAULT,
        FOREIGN KEY (CardNumber) REFERENCES BillingInfo(CardNumber)
                ON DELETE SET DEFAULT
);
```

8. INSERT statements to populate each table with at least 5 tuples.

```
INSERT
INTO        Customer (Email, Name, PhoneNumber, Password, Address)
VALUES      (‘john.doe@dummy.com’, ‘John Doe’, ‘1234567890’, ‘password’,
            ‘9090 main st, Vancouver’),
            (‘james.w@dummy.com’, ‘James Welch’, ‘2345678901’, ‘security1’,
            ‘212 oak st, Tokyo’),
            (‘arjund122@dummy.com’, ‘Arjun Div’, ‘3456789012’, ‘12345pass’,
            ‘1009 nh 29, New Delhi’),
            (‘jin.park@dummy.com’, ‘Jin Park’, ‘4567890123’, ‘password’,
            ‘Blue springs, Seoul’),
            (‘jane.doe@dummy.com’, ‘Jane Doe’, ‘5678901234’, ‘num123’,
            ‘2100 Student Union, Vancouver’);

INSERT
INTO        PartOf(Email, ProjectID)
VALUES      (‘john.doe@dummy.com’, 1),
            (‘james.w@dummy.com’, 2),
            (‘arjund122@dummy.com’, 3),
            (‘jin.park@dummy.com’, 4),
            (‘jane.doe@dummy.com’, 5);

INSERT
INTO        Project(ProjectID, Name, Description, CreationDate, Status)
VALUES      (1, ‘Migrate’, ‘Migrating app to cloud’, ‘2024-07-20’, ‘Active’),
            (2, ‘Analytics’, ‘Cloud analytics’, ‘2024-07-20’, ‘Active’),
            (3, ‘OpsPipeline’, ‘Operations Pipeline’, ‘2024-07-20’, ‘Active’),
            (4, ‘ModelTraining’, ‘AI models on cloud GPU’, ‘2024-07-20’, ‘Active’),
            (5, ‘Store’, ‘Cloud storage’, ‘2024-07-20’, ‘Active’);

INSERT
INTO        PaymentInfo(CardNumber, Email, CVV, PostalCode, ExpiryDate, IsDefault)
VALUES      (4536123465366009, ‘john.doe@dummy.com’, 001, ‘V6T1Z2’,
            ‘2024-12-01’, ‘TRUE’),
            (7878932100108375, ‘james.w@dummy.com’, 900, ‘135-0016’,
            ‘2025-12-01’, ‘TRUE’),
            (2233600070708181, ‘arjund122@dummy.com’, 500, ‘110001’,
            ‘2024-12-31’, ‘TRUE’),
            (2378001083456789, ‘jin.park@dummy.com’, 400, ‘01014’, ‘2026-06-30’,
            ‘TRUE’),
```

```
        (0101999963531010, 'jane.doe@dummy.com', 249, '89116', '2025-05-25',
        'TRUE');

INSERT
INTO    AddressInfo(PostalCode, City, Country)
VALUES  ('V6T1Z2', 'Vancouver', 'Canada'),
        ('135-0016', 'Tokyo', 'Japan'),
        ('110001', 'New Delhi', 'India'),
        ('01014', 'Seoul', 'South Korea'),
        ('89116', 'Las Vegas', 'United States');

INSERT
INTO    BillingInfo(CardNumber, PaymentType)
VALUES  (4536123465366009, 'Credit Card'),
        (7878932100108375, 'Credit Card'),
        (2233600070708181, 'Credit Card'),
        (2378001083456789, 'Credit Card'),
        (0101999963531010, 'Credit Card');

INSERT
INTO    CustomerUsesService(Email, Name)
VALUES  ('john.doe@dummy.com', 'gamma'),
        ('james.w@dummy.com', 'authIt'),
        ('arjund122@dummy.com', 'GPUb'),
        ('jin.park@dummy.com', 'RapidX'),
        ('jane.doe@dummy.com', 'cSQL');

INSERT
INTO    ProjectUsesService(ProjectID, Name)
VALUES  (1, 'gamma'),
        (2, 'authIt'),
        (3, 'GPUb'),
        (4, 'RapidX'),
        (5, 'cSQL');

INSERT
INTO    ServiceType(Type, CostType)
VALUES  ('Serverless', 'PerJob'),
        ('Authentication', 'PerUser'),
        ('GPUCompute', 'PerDay'),
```

```
        ('HPC', 'PerDay'),
        ('Database', 'PerQuery');

INSERT
INTO    ServiceDetails(Name, Type, Description, Status, CostPerUnit)
VALUES  ('gamma', 'Serverless', 'Handling serverless functions', 'Active', 1),
        ('authIt', 'Authentication', 'Authentication with large capacity', 'Active', 2),
        ('GPUb', 'GPUCompute', 'GPU-based rendering service', 'Active', 3),
        ('RapidX', 'HPC', 'High performance computing service', 'Active', 4),
        ('cSQL', 'Database', 'Cloud SQL database service', 'Active', 5);

INSERT
INTO    Storage(Name, Capacity, Latency)
VALUES  ('B3Vault', 500, 5),
        ('Box', 100, 10),
        ('GigaCache', 2000, 2),
        ('Submin2', 1500, 3),
        ('cSQL', 300, 7);

INSERT
INTO    Compute(Name, CPUCores, MaxMemory)
VALUES  ('GPUb', 32, 128),
        ('RapidX', 16, 64),
        ('EC3', 64, 256),
        ('EngineCl', 8, 32),
        ('Titanium', 128, 512);

INSERT
INTO    Functional(Name, Timeout, ConcurrencyLimit)
VALUES  ('gamma', 300, 10),
        ('authIt', 200, 20),
        ('Apex', 500, 5),
        ('Actions', 400, 15),
        ('C2Chain', 100, 25);

INSERT
INTO    Region(Name, Location, Capacity, Status)
VALUES  ('us-east-1', 'Virginia', 500, 'Active'),
        ('us-west-2', 'Oregon', 500, 'Active'),
        ('eu-central-1', 'Frankfurt', 300, 'Active'),
```

```
        ('eu-west-1', 'Ireland', 300, 'Active'),
        ('sa-east-1', 'Caracas', 200, 'Active');

INSERT
INTO    AvailableIn(ServiceName, RegionName)
VALUES  ('gamma', 'us-east-1'),
        ('authIt', 'us-west-2'),
        ('GPUb', 'eu-central-1'),
        ('RapidX', 'eu-west-1'),
        ('cSQL', 'sa-east-1');

INSERT
INTO    ProjectSecurity(SecurityGroupID, ProjectID)
VALUES  (1, 1),
        (2, 2),
        (3, 3),
        (4, 4),
        (5, 5);

INSERT
INTO    SecurityProtocol(InboundProtocol, OutboundProtocol)
VALUES  ('HTTP', 'TCP'),
        ('HTTPS', 'UDP'),
        ('FTP', 'SNMP'),
        ('SMTP', 'DNS'),
        ('IMAP', 'TLS');

INSERT
INTO    SecurityInfo(ProjectID, Name, InboundProtocol, CreationDate)
VALUES  (1, 'HyperText Transfer Protocol', 'HTTP', '2024-07-21'),
        (2, 'HyperText Transfer Protocol Secure', 'HTTPS', '2024-07-21'),
        (3, 'File Transfer Protocol', 'FTP', '2024-07-21'),
        (4, 'Simple Mail Transfer Protocol', 'SMTP', '2024-07-21'),
        (5, 'Internet Message Access Protocol', 'IMAP', '2024-07-21');

INSERT
INTO    ServerTypeInfo(ServerType, Memory, Storage, OS, CPUCores)
VALUES  ('functional.micro', 1, 8, 'Linux', 1),
        ('functional.large', 8, 32, 'Linux', 4),
        ('compute.large', 8, 64, 'Linux', 16),
```

```
('compute.xlarge', 16, 128, 'Linux', 32),
('storage.large', 32, 1024, 'Windows', 8);

INSERT
INTO        ServerInfo(ServerID, Name, ServerType, Status, Uptime, CreatedAt,
            UpdatedAt)
VALUES      (101, 'us-east-1', 'functional.micro', 'Active', 1000, '2024-01-04', '2024-07-15'),
            (102, 'us-west-2', 'functional.large', 'Active', 800, '2024-01-05', '2024-07-16'),
            (103, 'eu-central-1', 'compute.large', 'Active', 600, '2024-02-01', '2024-07-16'),
            (104, 'eu-west-1', 'compute.xlarge', 'Active', 1200, '2024-07-11', '2024-07-16'),
            (105, 'sa-east-1', 'storage.large', 'Inactive', 300, '2024-07-16', '2024-07-16');

INSERT
INTO        Instance(InstanceID, Name, ServerID, ProjectID, Type, TotalCost,
            Status, LaunchDate, StopDate)
VALUES      (1, 'gamma', 101, 1, 'micro', 0, 'Running', '2024-07-20', NULL),
            (2, 'authIt', 102, 2, 'large', 9, 'Stopped', '2024-07-20', '2024-07-21'),
            (3, 'GPUb', 103, 3, 'large', 0, 'Running', '2024-07-20', NULL),
            (4, 'RapidX', 104, 4, 'xlarge', 0, 'Running', '2024-07-20', NULL),
            (5, 'cSQL', 105, 5, 'large', 5, 'Stopped', '2024-07-16', '2024-07-16');

INSERT
INTO        Bills(InstanceID, Email, CardNumber, Cost)
VALUES      (1, 'john.doe@dummy.com', 4536123465366009, 500),
            (2, 'james.w@dummy.com', 7878932100108375, 244),
            (3, 'arjund122@dummy.com', 2233600070708181, 306),
            (4, 'jin.park@dummy.com', 2378001083456789, 123),
            (5, 'jane.doe@dummy.com', 0101999963531010, 890);
```