

COP 5725
Database Management Systems
Project Deliverable 3

Group 21

Chen, Zhaoyang <zhaoyang.chen@ufl.edu>

Feng, Zhongxi <zhongxi.feng@ufl.edu>

Niu, Yicong <yicong.niu@ufl.edu>

Qi, Lin <linqi@ufl.edu>

Interactive City Weather Report
Project Phase III
User Interface Design & Database Design

Contents

Modified ER Diagram	3
Transformation of the ER Diagram into Relation Schemas	3
Transformation of the Relation Schemas into SQL Table Schemas	5

Modified ER Diagram

We canceled the login function and users can view various weather data without logging in.

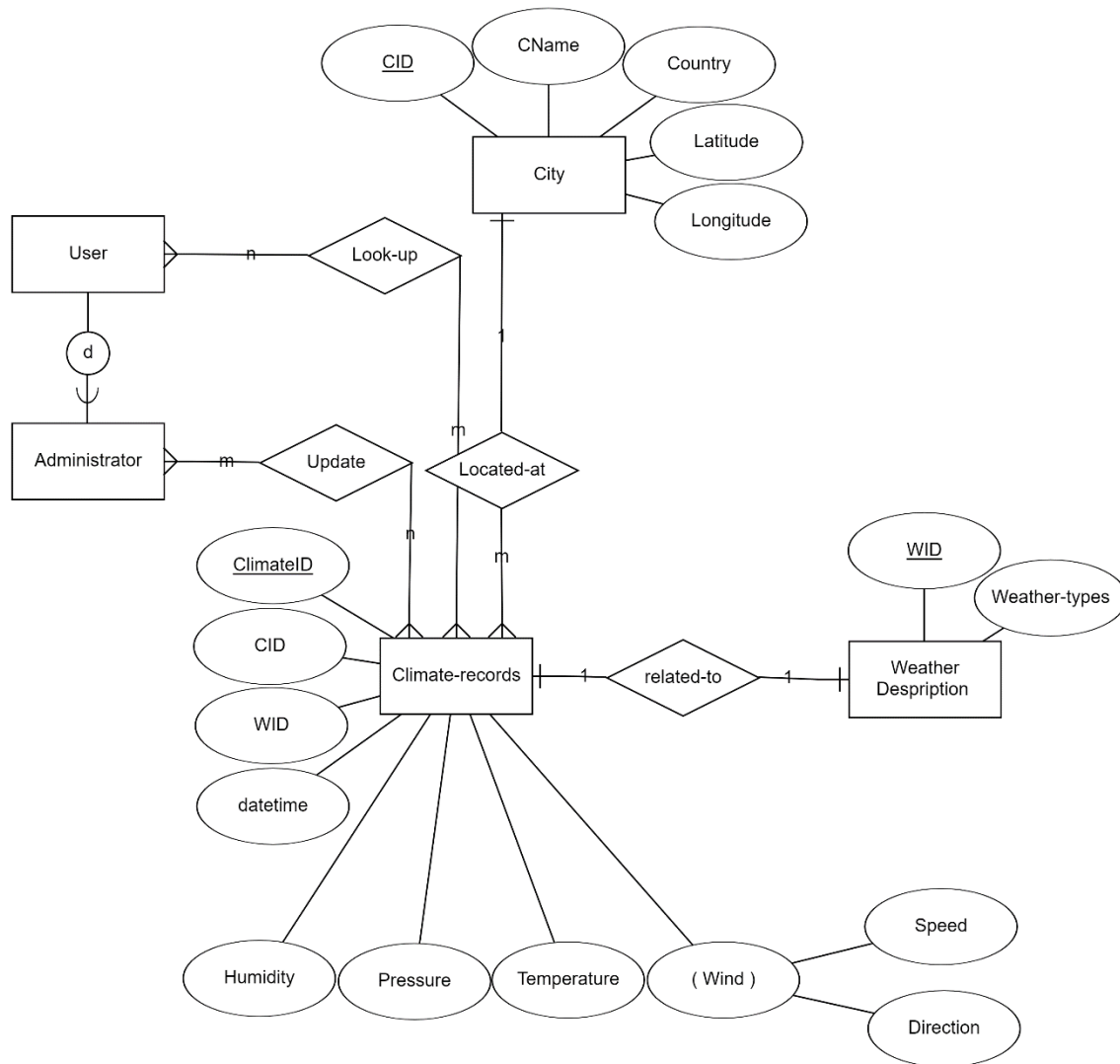


Figure 1: Entity-Relationship Diagram

Transformation of the ER Diagram into Relation Schemas

User (void entity)

Administrator (void entity)

WeatherDescription (WID: Varchar, WeatherTypes: Varchar)

City (CID: Varchar, CName: Varchar, Country: Varchar, Latitude: Number, Longitude: Number)

ClimateRecords (ClimateID: Varchar, CID: Varchar, WID: Varchar, datetime: Date, Humidity: Number, Pressure: Number, Temperature: Number, WindSpeed: Number, WindDirection: Number)

In our design, we assign an ID to each entity to denote a unique record. For clarity and efficiency, the ID field is an ascending integer field.

As we don't need login in our web application. We create two void entities, User and Administrator. The **User** schema denotes a normal user that could use our application and look up all the data. Single user could use the application to find several records. A record could be looked up by several users. Thus, the cardinality is set to be m:n. User schema was not in the original dataset but is created and maintained by us.

The relational schema **Administrator** is a subclass of the schema User, so the attributes of User also apply to Administrator. One administrator could manipulate many data records. A record could be updated by several administrators. Thus, the cardinality is set to be m:n.

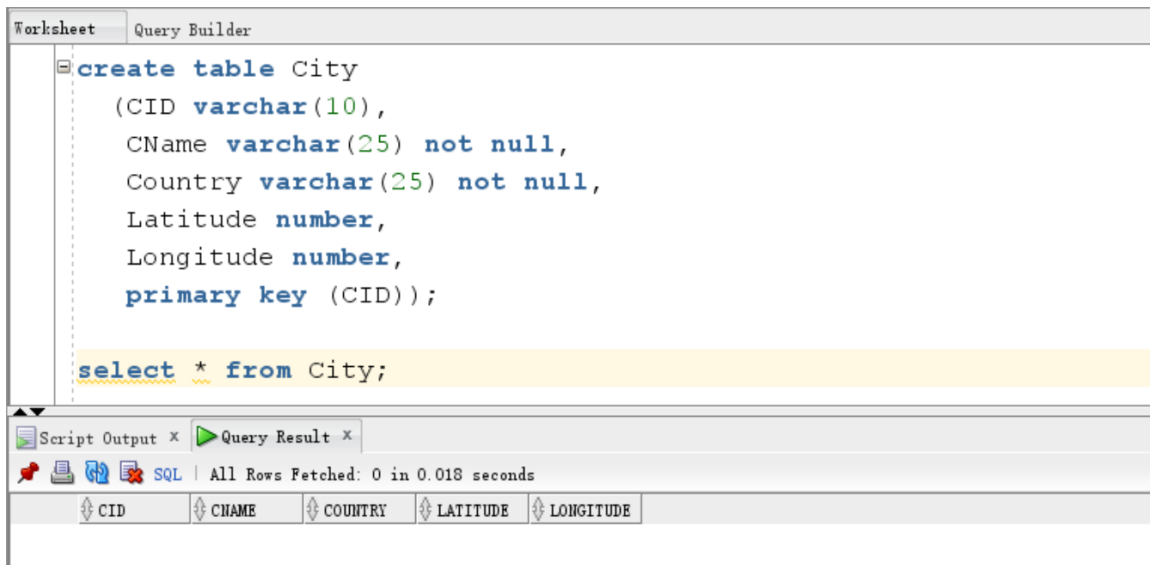
For the relational schema **City**, we add an integer primary key CID that denotes each unique city. The name of the city and the country the city belongs to are stored as string. Due to the precision, latitude and longitude are stored as float type. A city corresponds to many records, while a record could only belong to one specific city. Thus, the cardinality is set to be 1:m.

For the relational schema **WeatherDescription**, the weather is described from a casual perspective, so we assign the field to be a string field. We add an integer primary key WID to denote each record. A record in WeatherDescription only belongs to one record in ClimateRecords and vice versa. So the cardinality is set to be 1:1.

Our main relational schema, **ClimateRecords**, consists of all major features mentioned in the weather-related table in our database. It includes Humidity, Pressure, Temperature, WindSpeed, and WindDirection. To apply the aggregation function on each attribute and keep the precision, we assign each domain to float data type. The integer primary key is ClimateID. According to the cardinality between City and ClimateRecords, which is 1:m, we add CID as a foreign key to ClimateRecords. Also, the cardinality between WeatherDescription and ClimateRecords is 1:1, so we add WID as a foreign key to ClimateRecords. Thus, by WID and CID, one could find the corresponding city and weather description in the ClimateRecords table.

Transformation of the Relation Schemas into SQL Table Schemas

After the Entity-Relationship diagram was transformed into relation schema, a collection of SQL table schema is required to be transformed. As the screenshot showing below, some entities or attributes were created by varchar (like CID, CName and country). And attributes like latitude and longitude were obviously set by number. In the end, CID was set as the primary key. At the rest of this screenshot, we created an empty table starting by CID.



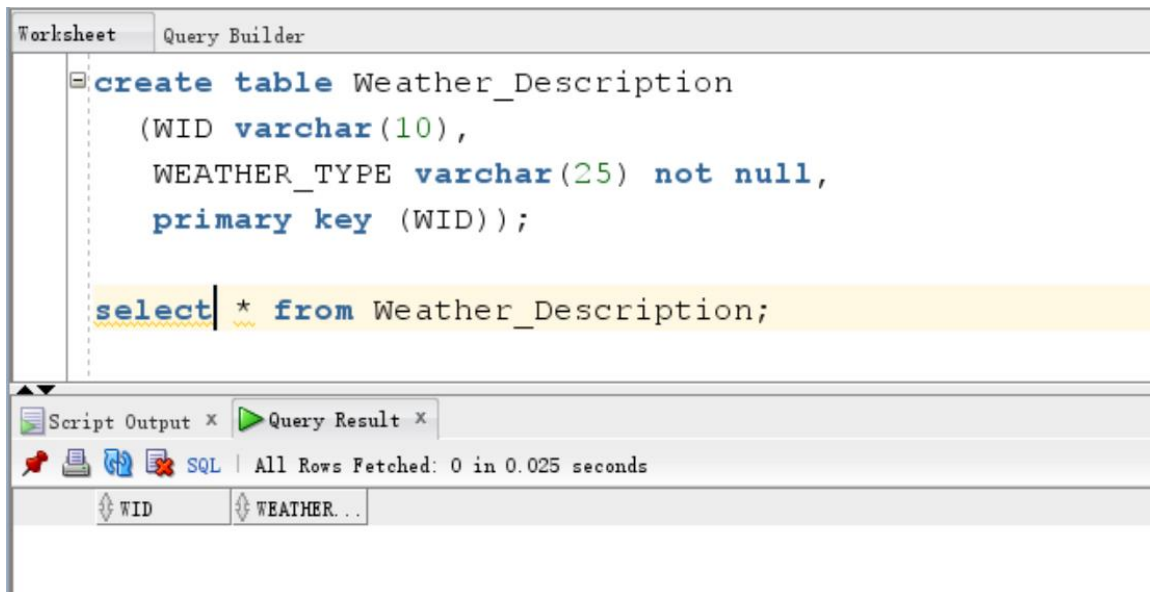
The screenshot shows a SQL Query Builder window with a 'Worksheet' tab. The SQL editor contains the following code:

```
create table City
(
  CID varchar(10),
  CName varchar(25) not null,
  Country varchar(25) not null,
  Latitude number,
  Longitude number,
  primary key (CID));

select * from City;
```

Below the editor, the 'Script Output' and 'Query Result' tabs are visible. The 'Query Result' tab shows the execution status: 'All Rows Fetched: 0 in 0.018 seconds'. Below this, a table structure is displayed with columns: CID, CNAME, COUNTRY, LAITITUDE, and LONGITUDE.

For another main entity, weather description, another table was created with the primary key, WID by varchar, and weather-type by varchar. Similarly, the empty table was displayed in the screenshot.



The screenshot shows a SQL Query Builder window with a 'Worksheet' tab. The SQL editor contains the following code:

```
create table Weather_Description
(
  WID varchar(10),
  WEATHER_TYPE varchar(25) not null,
  primary key (WID));

select * from Weather_Description;
```

Below the editor, the 'Script Output' and 'Query Result' tabs are visible. The 'Query Result' tab shows the execution status: 'All Rows Fetched: 0 in 0.025 seconds'. Below this, a table structure is displayed with columns: WID and WEATHER...

In the next main entity, Climate_Records, a series of attributes was created. Identities like CID and WID were set by varchar. Since time is a critical attribute for the complex queries in our project, Datetime was set by date. Attributes like humidity, pressure, temperature, wind speed and wind direction were set by number. In addition, two foreign keys were created by CID and WID to make sure the tables are connected and the SQL schema table can be improved by the constraints of these two foreign keys. In the end, an empty table was shown in the screenshot.

The screenshot displays a SQL Query Builder window with a 'Worksheet' tab. The main area contains the following SQL code:

```
create table Climate_Records
(
    ClimateID varchar(10),
    CID varchar(10),
    WID varchar(10),
    datetime date,
    Humidity number,
    Pressure number,
    Temperature number,
    WindSpeed number,
    WindDirection number,
    primary key (ClimateID),
    CONSTRAINT FK_CID FOREIGN KEY (CID)
    REFERENCES City(CID),
    CONSTRAINT FK_WID FOREIGN KEY (WID)
    REFERENCES Weather_Description(WID)
);

select * from Climate_records;
```

Below the code editor, there is a 'Script Output' and 'Query Result' section. The status bar indicates 'All Rows Fetched: 0 in 0.022 seconds'. At the bottom, a table structure is shown with columns: CLIMATEID, CID, WID, DATETIME, HUMIDITY, PRESSURE, TEMPERA..., WINDSPEED, and WINDDIR...