miniSQL Documentation

Release 1.0.0

DevOpSec

May 21, 2019

CHAPTER 1

Project Requirements

This project was written as an exercise for my Database Systems class. Here were the requirements for the project.

1.1 Summary

For this assignment you will write a fully self-describing program to perform file operations using dynamic hashing.

1.2 The records

Each record will consist of a name field, an eighteen-character string, which is the key field and four non-negative integers; having field names a1, a2, a3, and a4.

1.3 The file

You will use block I/O to perform all operations on the file. A block size of 40 will be used. Within a block, you should read and write whole records. The file must be created before use, but not removed after uses. It should expand and contract as needed.

1.4 The Input

Input to your program will consist of commands, one per line, requesting operations to be performed on the file. Input will come from standard input. Legal commands are:

I record – Insert record into the file, unless one having the same key is already there.

R name – Retrieve the record having key name, if it is there.

D name – Delete the record having key name from the file, if it exists.

U name a-field name new-value. Update the given record by finding it (if it exists) and replacing the given information.

In order to make the input easy to parse:

1. The first character will begin in column one;

2. **The elements of each type of input will be separated by single tab**
   characters;
3. The name field will contain no tabs;
4. A new line will immediately follow the last element of each command.
5. **All input is guaranteed to be correct, so no data validation need be**
   performed.

1.5 The Output

This program will be fully self-describing, which means that a person can read only the output file and duplicate by hand the exact actions of the program on a given input sequence. Hand duplication should be possible even without availability of the input file.

1.6 Deliverables

1. A Database using MySQL and an corresponding ER Diagram describing the database.
2. A new DBMS system to fully realize your database.
3. An interface to your DBMS system in PHP.
4. Complete, professionally written documentation.
5. Test input files and commands.

1.7 Notes

1. Be sure to include identification information in all of your files and outputs.

2. The program will be tested by redirected standard I/O to and from files. Be sure that your input echo still works when I/O is redirected.

3. In general, your code should use low-level file I/O. If you are programming in C, you should use fread, fwrite and fseek for file I/O. If you are using Java, you probably want to use Buffered Streams.

1. Include test output as well.

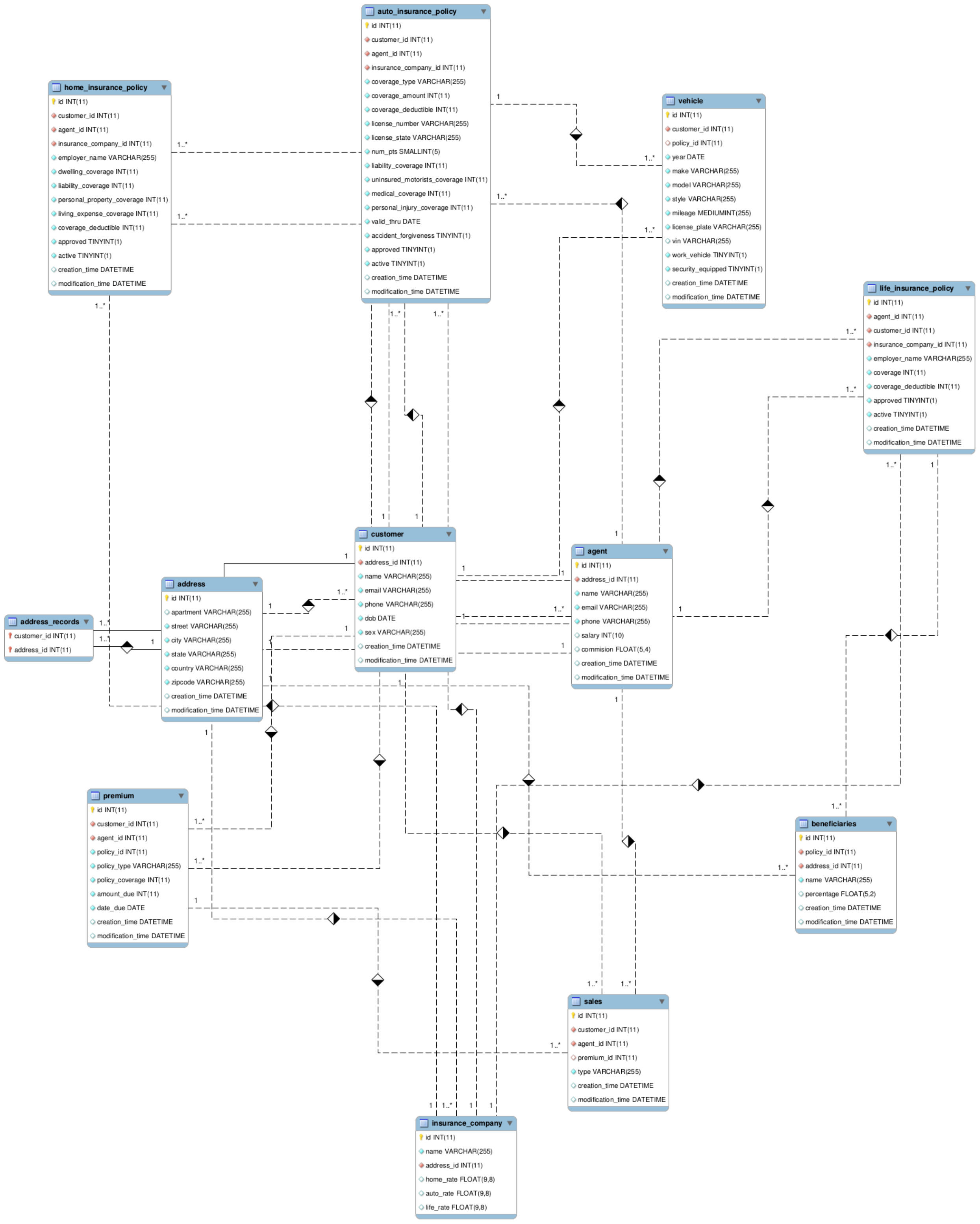
CHAPTER 2

Database

2.1 Description

description.pdf

2.2 Diagrams


auto_insurance_policy
id INT(11)
customer_id INT(11)
agent_id INT(11)
insurance_company_id INT(11)
coverage_type VARCHAR(255)
coverage_amount INT(11)
coverage_deductible INT(11)
license_number VARCHAR(255)
license_state VARCHAR(255)
num_pts SMALLINT(5)
liability_coverage INT(11)
uninsured_motorists_coverage INT(11)
medical_coverage INT(11)
personal_injury_coverage INT(11)
valid_thru DATE
accident_forgiveness TINYINT(1)
approved TINYINT(1)
active TINYINT(1)
creation_time DATETIME
modification_time DATETIME
home_insurance_policy
id INT(11)
customer_id INT(11)
agent_id INT(11)
insurance_company_id INT(11)
employer_name VARCHAR(255)
dwelling_coverage INT(11)
liability_coverage INT(11)
personal_property_coverage INT(11)
living_expense_coverage INT(11)
coverage_deductible INT(11)
approved TINYINT(1)
active TINYINT(1)
creation_time DATETIME
modification_time DATETIME
vehicle
id INT(11)
customer_id INT(11)
policy_id INT(11)
year DATE
make VARCHAR(255)
model VARCHAR(255)
style VARCHAR(255)
mileage MEDIUMINT(255)
license_plate VARCHAR(255)
vin VARCHAR(255)
work_vehicle TINYINT(1)
security_equipped TINYINT(1)
creation_time DATETIME
modification_time DATETIME
life_insurance_policy
id INT(11)
agent_id INT(11)
customer_id INT(11)
insurance_company_id INT(11)
employer_name VARCHAR(255)
coverage INT(11)
coverage_deductible INT(11)
approved TINYINT(1)
active TINYINT(1)
creation_time DATETIME
modification_time DATETIME
customer
id INT(11)
address_id INT(11)
name VARCHAR(255)
email VARCHAR(255)
phone VARCHAR(255)
dob DATE
sex VARCHAR(255)
creation_time DATETIME
modification_time DATETIME
agent
id INT(11)
address_id INT(11)
name VARCHAR(255)
email VARCHAR(255)
phone VARCHAR(255)
salary INT(10)
commision FLOAT(5,4)
creation_time DATETIME
modification_time DATETIME
address
id INT(11)
apartment VARCHAR(255)
street VARCHAR(255)
city VARCHAR(255)
state VARCHAR(255)
country VARCHAR(255)
zipcode VARCHAR(255)
creation_time DATETIME
modification_time DATETIME
address_records
customer_id INT(11)
address_id INT(11)
premium
id INT(11)
customer_id INT(11)
agent_id INT(11)
policy_id INT(11)
policy_type VARCHAR(255)
policy_coverage INT(11)
amount_due INT(11)
date_due DATE
creation_time DATETIME
modification_time DATETIME
beneficiaries
id INT(11)
policy_id INT(11)
address_id INT(11)
name VARCHAR(255)
percentage FLOAT(5,2)
creation_time DATETIME
modification_time DATETIME
sales
id INT(11)
customer_id INT(11)
agent_id INT(11)
premium_id INT(11)
type VARCHAR(255)
creation_time DATETIME
modification_time DATETIME
insurance_company
id INT(11)
name VARCHAR(255)
address_id INT(11)
home_rate FLOAT(9,8)
auto_rate FLOAT(9,8)
life_rate FLOAT(9,8)
1
1..*

2.3 Mysql Schema

```
DROP DATABASE IF EXISTS global_insurance;
CREATE DATABASE global_insurance;

USE global_insurance;

DROP TABLE IF EXISTS address;
CREATE TABLE address (
  id INT(11) NOT NULL AUTO_INCREMENT,
  apartment VARCHAR(255) DEFAULT '',
  street VARCHAR(255) NOT NULL,
  city VARCHAR(255) NOT NULL,
  state VARCHAR(255) NOT NULL,
  country VARCHAR(255) NOT NULL,
  zipcode VARCHAR(255) NOT NULL,
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
);

DROP TABLE IF EXISTS agent;
CREATE TABLE agent (
  id INT(11) NOT NULL AUTO_INCREMENT,
  address_id INT(11) NOT NULL,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  phone VARCHAR(255) NOT NULL,
  salary INT(10) DEFAULT 0,
  commision float(5,4) DEFAULT 0.00,
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (address_id) REFERENCES address (id)
);

DROP TABLE IF EXISTS customer;
CREATE TABLE customer (
  id INT(11) NOT NULL AUTO_INCREMENT,
  address_id INT(11) NOT NULL,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  phone VARCHAR(255) NOT NULL,
  dob DATE NOT NULL,
  sex VARCHAR(255) NOT NULL, -- male | female | other
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (address_id) REFERENCES address (id)
);

DROP TABLE IF EXISTS address_records;
CREATE TABLE address_records (
  customer_id INT(11) NOT NULL,
  address_id INT(11) NOT NULL,
  PRIMARY KEY (customer_id, address_id),
  CONSTRAINT FOREIGN KEY (customer_id) REFERENCES customer (id),
  CONSTRAINT FOREIGN KEY (address_id) REFERENCES address (id)
);

-- create address_record when customer address created
DROP TRIGGER IF EXISTS insert_address_record;
DELIMITER //
CREATE TRIGGER insert_address_record AFTER INSERT ON customer
  FOR EACH ROW
  BEGIN
    INSERT IGNORE INTO address_records VALUES (NEW.id, NEW.address_id);
  END;//
DELIMITER ;

-- create address_record when customer address updated
DROP TRIGGER IF EXISTS update_address_record;
DELIMITER //
CREATE TRIGGER update_address_record AFTER UPDATE ON customer
  FOR EACH ROW
  BEGIN
    INSERT IGNORE INTO address_records VALUES (NEW.id, NEW.address_id);
  END;//
DELIMITER ;

DROP TABLE IF EXISTS premium;
CREATE TABLE premium (
  id INT(11) NOT NULL AUTO_INCREMENT,
  customer_id INT(11) NOT NULL,
  agent_id INT(11) NOT NULL,
  policy_id INT(11) NOT NULL, -- constrained by application
  policy_type VARCHAR(255) NOT NULL, -- home | auto | life
  policy_coverage INT(11) NOT NULL,
  amount_due INT(11) NOT NULL, -- total coverage * (rate - (deductible / total
coverage * 0.01))
  date_due DATE NOT NULL,
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (customer_id) REFERENCES customer (id),
  CONSTRAINT FOREIGN KEY (agent_id) REFERENCES agent (id)
);

DROP TABLE IF EXISTS insurance_company;
CREATE TABLE insurance_company (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  address_id INT(11) NOT NULL,
  home_rate float(9,8) DEFAULT 0.000025,
  auto_rate float(9,8) DEFAULT 0.00028,
  life_rate float(9,8) DEFAULT 0.000021,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (address_id) REFERENCES address (id)
);

-- update premiums when insurance_company rates change
DROP TRIGGER IF EXISTS update_insurance_company_premium;
DELIMITER //
CREATE TRIGGER update_insurance_company_premium AFTER UPDATE ON insurance_company
  FOR EACH ROW
  BEGIN
    UPDATE premium p
    LEFT JOIN (
        select id as policy_id,'home' as policy_type,NEW.home_rate as
rate,coverage_deductible as deductible from home_insurance_policy t1 where
t1.insurance_company_id=NEW.id
        union select id as policy_id,'auto' as policy_type,NEW.auto_rate as
rate,coverage_deductible as deductible from auto_insurance_policy t2 where
t2.insurance_company_id=NEW.id
        union select id as policy_id,'life' as policy_type,NEW.life_rate as
rate,coverage_deductible as deductible from life_insurance_policy t3 where
t3.insurance_company_id=NEW.id
      ) as temp USING (policy_id,policy_type)
    SET amount_due =
(p.policy_coverage*((temp.rate)-((temp.deductible)/p.policy_coverage*0.01)));
  END;//
DELIMITER ;

DROP TABLE IF EXISTS sales;
CREATE TABLE sales (
  id INT(11) NOT NULL AUTO_INCREMENT,
  customer_id INT(11) NOT NULL,
  agent_id INT(11) NOT NULL,
  premium_id INT(11) DEFAULT NULL,
  type VARCHAR(255) NOT NULL, -- appt | auto | home | life
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (customer_id) REFERENCES customer (id),
  CONSTRAINT FOREIGN KEY (agent_id) REFERENCES agent (id),
  CONSTRAINT FOREIGN KEY (premium_id) REFERENCES premium (id)
);

DROP TABLE IF EXISTS home_insurance_policy;
CREATE TABLE home_insurance_policy (
  id INT(11) NOT NULL AUTO_INCREMENT,
  customer_id INT(11) NOT NULL,
  agent_id INT(11) NOT NULL,
  insurance_company_id INT(11) NOT NULL,
  employer_name VARCHAR(255) NOT NULL,
  dwelling_coverage INT(11) NOT NULL DEFAULT 0,
  liability_coverage INT(11) NOT NULL DEFAULT 200000,
  personal_property_coverage INT(11) NOT NULL DEFAULT 0,
  living_expense_coverage INT(11) NOT NULL DEFAULT 0,
  coverage_deductible INT(11) NOT NULL DEFAULT 500,
  approved TINYINT(1) NOT NULL DEFAULT 0,
  active TINYINT(1) NOT NULL DEFAULT 0,
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (customer_id) REFERENCES customer (id),
  CONSTRAINT FOREIGN KEY (agent_id) REFERENCES agent (id),
  CONSTRAINT FOREIGN KEY (insurance_company_id) REFERENCES insurance_company (id)
);

-- update premium when home insurance is sold
DROP TRIGGER IF EXISTS insert_home_insurance_premium;
DELIMITER //
CREATE TRIGGER insert_home_insurance_premium AFTER INSERT ON home_insurance_policy
```

```
  FOR EACH ROW
  BEGIN
    DECLARE total_coverage INT;
    SET total_coverage :=
(NEW.dwelling_coverage+NEW.liability_coverage+NEW.personal_property_coverage+NEW.living_expense_cov
    INSERT INTO premium VALUES (
      DEFAULT,NEW.customer_id,NEW.agent_id,NEW.id,'home',total_coverage,
      (total_coverage*((SELECT home_rate FROM insurance_company WHERE id =
NEW.insurance_company_id)-(NEW.coverage_deductible/total_coverage*0.01))),
      (CURRENT_DATE+INTERVAL 1 YEAR),DEFAULT,DEFAULT
    );
  END;//
DELIMITER ;

-- update premium when home insurance is updated
DROP TRIGGER IF EXISTS update_home_insurance_premium;
DELIMITER //
CREATE TRIGGER update_home_insurance_premium AFTER UPDATE ON home_insurance_policy
  FOR EACH ROW
  BEGIN
    DECLARE total_coverage INT;
    SET total_coverage :=
(NEW.dwelling_coverage+NEW.liability_coverage+NEW.personal_property_coverage+NEW.living_expense_cov
    UPDATE premium SET policy_coverage = total_coverage,
      amount_due = (total_coverage*((SELECT home_rate FROM insurance_company WHERE id
 = NEW.insurance_company_id)-(NEW.coverage_deductible/total_coverage*0.01)))
    WHERE policy_id = NEW.id and policy_type = 'home';
  END;//
DELIMITER ;

-- create sales record when home insurance is sold
DROP TRIGGER IF EXISTS insert_home_insurance_sales;
DELIMITER //
CREATE TRIGGER insert_home_insurance_sales AFTER INSERT ON home_insurance_policy
  FOR EACH ROW FOLLOWS insert_home_insurance_premium
  BEGIN
    DECLARE premium_id INT;
    SELECT id INTO premium_id FROM premium WHERE policy_id = NEW.id and policy_type =
 'home';
    INSERT INTO sales VALUES
(DEFAULT,NEW.customer_id,NEW.agent_id,premium_id,'home',DEFAULT,DEFAULT);
  END;//
DELIMITER ;

DROP TABLE IF EXISTS auto_insurance_policy;
CREATE TABLE auto_insurance_policy (
  id INT(11) NOT NULL AUTO_INCREMENT,
  customer_id INT(11) NOT NULL,
  agent_id INT(11) NOT NULL,
  insurance_company_id INT(11) NOT NULL,
  coverage_type VARCHAR(255) NOT NULL DEFAULT 'standard', -- standard | collision |
comprehensive
  coverage_amount INT(11) NOT NULL DEFAULT 0,
  coverage_deductible INT(11) NOT NULL DEFAULT 500,
  license_number VARCHAR(255) NOT NULL,
  license_state VARCHAR(255) NOT NULL,
  num_pts SMALLINT(5) NOT NULL,
  liability_coverage INT(11) NOT NULL DEFAULT 200000,
  uninsured_motorists_coverage INT(11) NOT NULL DEFAULT 0,
  medical_coverage INT(11) NOT NULL DEFAULT 50000,
  personal_injury_coverage INT(11) NOT NULL DEFAULT 1000000000,
  valid_thru DATE NOT NULL,
  accident_forgiveness TINYINT(1) NOT NULL DEFAULT 0,
  approved TINYINT(1) NOT NULL DEFAULT 0,
  active TINYINT(1) NOT NULL DEFAULT 0,
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (customer_id) REFERENCES customer (id),
  CONSTRAINT FOREIGN KEY (agent_id) REFERENCES agent (id),
  CONSTRAINT FOREIGN KEY (insurance_company_id) REFERENCES insurance_company (id)
);

-- update premium when auto insurance is sold
DROP TRIGGER IF EXISTS insert_auto_insurance_premium;
DELIMITER //
CREATE TRIGGER insert_auto_insurance_premium AFTER INSERT ON auto_insurance_policy
  FOR EACH ROW
  BEGIN
    DECLARE total_coverage INT;
    SET total_coverage :=
(NEW.coverage_amount+NEW.liability_coverage+NEW.uninsured_motorists_coverage+NEW.medical_coverage+N
    INSERT INTO premium VALUES (
      DEFAULT,NEW.customer_id,NEW.agent_id,NEW.id,'auto',total_coverage,
      (total_coverage*((SELECT auto_rate FROM insurance_company WHERE id =
NEW.insurance_company_id)-(NEW.coverage_deductible/total_coverage*0.01))),
      (CURRENT_DATE+INTERVAL 1 YEAR),DEFAULT,DEFAULT
    );
  END;//
DELIMITER ;

-- update premium when auto insurance is updated
DROP TRIGGER IF EXISTS update_auto_insurance_premium;
DELIMITER //
CREATE TRIGGER update_auto_insurance_premium AFTER UPDATE ON auto_insurance_policy
  FOR EACH ROW
  BEGIN
    DECLARE total_coverage INT;
    SET total_coverage :=
(NEW.coverage_amount+NEW.liability_coverage+NEW.uninsured_motorists_coverage+NEW.medical_coverage+N
    UPDATE premium SET policy_coverage = total_coverage,
      amount_due = (total_coverage*((SELECT auto_rate FROM insurance_company WHERE id
 = NEW.insurance_company_id)-(NEW.coverage_deductible/total_coverage*0.01)))
    WHERE policy_id = NEW.id and policy_type = 'auto';
  END;//
DELIMITER ;

-- create sales record when auto insurance is sold
DROP TRIGGER IF EXISTS insert_auto_insurance_sales;
DELIMITER //
CREATE TRIGGER insert_auto_insurance_sales AFTER INSERT ON auto_insurance_policy
  FOR EACH ROW FOLLOWS insert_auto_insurance_premium
  BEGIN
    DECLARE premium_id INT;
    SELECT id INTO premium_id FROM premium WHERE policy_id = NEW.id and policy_type =
 'auto';
    INSERT INTO sales VALUES
(DEFAULT,NEW.customer_id,NEW.agent_id,premium_id,'auto',DEFAULT,DEFAULT);
```

```
  END;//
DELIMITER ;

DROP TABLE IF EXISTS life_insurance_policy;
CREATE TABLE life_insurance_policy (
  id INT(11) NOT NULL AUTO_INCREMENT,
  agent_id INT(11) NOT NULL,
  customer_id INT(11) NOT NULL,
  insurance_company_id INT(11) NOT NULL,
  employer_name VARCHAR(255) NOT NULL,
  coverage INT(11) NOT NULL DEFAULT 500000,
  coverage_deductible INT(11) NOT NULL DEFAULT 500,
  approved TINYINT(1) NOT NULL DEFAULT 0,
  active TINYINT(1) NOT NULL DEFAULT 0,
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (customer_id) REFERENCES customer (id),
  CONSTRAINT FOREIGN KEY (agent_id) REFERENCES agent (id),
  CONSTRAINT FOREIGN KEY (insurance_company_id) REFERENCES insurance_company (id)
);

-- update premium when life insurance is sold
DROP TRIGGER IF EXISTS insert_life_insurance_premium;
DELIMITER //
CREATE TRIGGER insert_life_insurance_premium AFTER INSERT ON life_insurance_policy
  FOR EACH ROW
  BEGIN
    INSERT INTO premium VALUES (
      DEFAULT,NEW.customer_id,NEW.agent_id,NEW.id,'life',NEW.coverage,
      (NEW.coverage*((SELECT life_rate FROM insurance_company WHERE id =
NEW.insurance_company_id)-(NEW.coverage_deductible/NEW.coverage*0.01))),
      (CURRENT_DATE+INTERVAL 1 YEAR),DEFAULT,DEFAULT
    );
  END;//
DELIMITER ;

-- update premium when life insurance is updated
DROP TRIGGER IF EXISTS update_life_insurance_premium;
DELIMITER //
CREATE TRIGGER update_life_insurance_premium AFTER UPDATE ON life_insurance_policy
  FOR EACH ROW
  BEGIN
    UPDATE premium SET policy_coverage = NEW.coverage,
      amount_due = (NEW.coverage*((SELECT life_rate FROM insurance_company WHERE id =
 NEW.insurance_company_id)-(NEW.coverage_deductible/NEW.coverage*0.01)))
    WHERE policy_id = NEW.id and policy_type = 'life';
  END;//
DELIMITER ;

-- create sales record when life insurance is sold
DROP TRIGGER IF EXISTS insert_life_insurance_sales;
DELIMITER //
CREATE TRIGGER insert_life_insurance_sales AFTER INSERT ON life_insurance_policy
  FOR EACH ROW FOLLOWS insert_life_insurance_premium
  BEGIN
    DECLARE premium_id INT;
    SELECT id INTO premium_id FROM premium WHERE policy_id = NEW.id and policy_type =
 'life';
```

```
    INSERT INTO sales VALUES
(DEFAULT,NEW.customer_id,NEW.agent_id,premium_id,'life',DEFAULT,DEFAULT);
  END;//
DELIMITER ;

DROP TABLE IF EXISTS beneficiaries;
CREATE TABLE beneficiaries (
  id INT(11) NOT NULL AUTO_INCREMENT,
  policy_id INT(11) NOT NULL,
  address_id INT(11) NOT NULL,
  name VARCHAR(255) NOT NULL,
  percentage float(5,2) DEFAULT 0.00, -- constrained by application
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (policy_id) REFERENCES life_insurance_policy (id),
  CONSTRAINT FOREIGN KEY (address_id) REFERENCES address (id) ON DELETE CASCADE
);

DROP TABLE IF EXISTS vehicle;
CREATE TABLE vehicle (
  id INT(11) NOT NULL AUTO_INCREMENT,
  customer_id INT(11) NOT NULL,
  policy_id INT(11) DEFAULT NULL,
  year DATE NOT NULL,
  make VARCHAR(255) NOT NULL,
  model VARCHAR(255) NOT NULL,
  style VARCHAR(255) NOT NULL,
  mileage MEDIUMINT(255) NOT NULL,
  license_plate VARCHAR(255) NOT NULL,
  vin VARCHAR(255) DEFAULT '',
  work_vehicle TINYINT(1) NOT NULL,
  security_equipped TINYINT(1) NOT NULL DEFAULT 0,
  creation_time DATETIME DEFAULT CURRENT_TIMESTAMP,
  modification_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  CONSTRAINT FOREIGN KEY (customer_id) REFERENCES customer (id),
  CONSTRAINT FOREIGN KEY (policy_id) REFERENCES auto_insurance_policy (id)
);
```

2.4 Mysql Data

```
use global_insurance;

-- create example data
insert into address values (DEFAULT,DEFAULT,'1234 fake st.','detroit','mi','united
states','00000',DEFAULT,DEFAULT);
insert into address values (DEFAULT,DEFAULT,'1234 fake rd.','flint','mi','united
states','00000',DEFAULT,DEFAULT);
insert into address values (DEFAULT,DEFAULT,'1234 fake blvd.','grand
rapids','mi','united states','00000',DEFAULT,DEFAULT);
insert into address values (DEFAULT,DEFAULT,'1234 fake
circle.','lansing','mi','united states','00000',DEFAULT,DEFAULT);
insert into address values (DEFAULT,DEFAULT,'1234 faker st.','detroit','mi','united
states','00000',DEFAULT,DEFAULT);
insert into address values (DEFAULT,DEFAULT,'1234 fakest st.','detroit','mi','united
states','00000',DEFAULT,DEFAULT);

insert into agent values (DEFAULT,3,'arya
stark','astark@gmail.com','012-345-6789',100000,0.02,DEFAULT,DEFAULT);

insert into customer values (DEFAULT,1,'julian
assange','rjames@gmail.com','012-345-6789',DATE('1993-01-01'),'male',DEFAULT,DEFAULT);
insert into customer values (DEFAULT,2,'edward
snowden','esnowden@gmail.com','012-345-6789',DATE('1993-01-01'),'male',DEFAULT,DEFAULT);
update customer set address_id = 5 where id = 1;
update customer set address_id = 6 where id = 1;

insert into insurance_company values (DEFAULT,'pwning
inc',4,DEFAULT,DEFAULT,DEFAULT);

insert into home_insurance_policy values (DEFAULT,1,1,1,'The Rock
Johnson',250000,DEFAULT ,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
insert into auto_insurance_policy values
(DEFAULT,1,1,1,DEFAULT,28000,DEFAULT,'0987654321','mi',12,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DATE('202
insert into life_insurance_policy values (DEFAULT,1,1,1,'The Rock
Johnson',400000,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
```

CHAPTER 3

Test Queries

3.1 MySQL Input

```
use global_insurance;

-- |========== get address records for a customer ==========|
select * from address t1, address_records t2
  where t2.customer_id = 1 and t1.id = t2.address_id;

-- |========== get agent managing each insurance policy for a customer ==========|
select (select t4.name where t4.id = t1.agent_id) as 'Home Insurance Agent',
    (select t4.name where t4.id = t2.agent_id) as 'Auto Insurance Agent',
    (select t4.name where t4.id = t3.agent_id) as 'Life Insurance Agent'
  from home_insurance_policy t1,
    auto_insurance_policy t2,
    life_insurance_policy t3,
    agent t4
  where t1.customer_id = 1
    or t2.customer_id = 1
    or t3.customer_id = 1;

-- | ========== get total coverage amount for an insurance company ==========|
select
(SUM(t1.dwelling_coverage)+SUM(t1.liability_coverage)+SUM(t1.personal_property_coverage)+SUM(t1.liv
 as 'Home Insurance Coverage',
(SUM(t2.coverage_amount)+SUM(t2.liability_coverage)+SUM(t2.uninsured_motorists_coverage)+SUM(t2.med
 as 'Auto Insurance Coverage',
   SUM(t3.coverage) as 'Life Insurance Coverage'
  from home_insurance_policy t1,
    auto_insurance_policy t2,
    life_insurance_policy t3,
    insurance_company t4
  where t1.insurance_company_id = 1
    or t2.insurance_company_id = 1
    or t3.insurance_company_id = 1;

-- |========== show sales for arya stark ==========|
select * from sales where agent_id = 1;
-- "

-- |========== show premiums for a customer ==========|
select policy_type as 'Insurance Type',amount_due as 'Amount Due' from premium where
customer_id = 1;

-- |========== update a policy and check premiums again ==========|
update auto_insurance_policy set coverage_amount = 34000, coverage_deductible = 200,
uninsured_motorists_coverage = 50000, accident_forgiveness = 1 where customer_id = 1;
  select policy_type as 'Insurance Type',amount_due as 'Amount Due' from premium
where customer_id = 1;

-- |========== update an insurance company and check premiums again ==========|
update insurance_company set home_rate = 0.0002559, auto_rate = 0.0028222, life_rate
= 0.0003299 where id=1;
  select name,home_rate,auto_rate,life_rate from insurance_company;
  select policy_type,amount_due from premium;
```

3.2 MySQL Output

```
+----+-----------+-----------------+---------+-------+---------------+---------+--------------------
| id | apartment | street          | city    | state | country       | zipcode |
creation_time        | modification_time   | customer_id | address_id |
+----+-----------+-----------------+---------+-------+---------------+---------+--------------------
|  1 |           | 1234 fake st.   | detroit | mi    | united states | 00000   |
2019-05-11 18:26:30 | 2019-05-11 18:26:30 |           1 |          1 |
|  5 |           | 1234 faker st.  | detroit | mi    | united states | 00000   |
2019-05-11 18:26:30 | 2019-05-11 18:26:30 |           1 |          5 |
|  6 |           | 1234 fakest st. | detroit | mi    | united states | 00000   |
2019-05-11 18:26:30 | 2019-05-11 18:26:30 |           1 |          6 |
+----+-----------+-----------------+---------+-------+---------------+---------+--------------------
+----------------------+----------------------+----------------------+
| Home Insurance Agent | Auto Insurance Agent | Life Insurance Agent |
+----------------------+----------------------+----------------------+
| arya stark           | arya stark           | arya stark           |
+----------------------+----------------------+----------------------+
+-------------------------+-------------------------+-------------------------+
| Home Insurance Coverage | Auto Insurance Coverage | Life Insurance Coverage |
+-------------------------+-------------------------+-------------------------+
|                  450000 |              1000334000 |                  400000 |
+-------------------------+-------------------------+-------------------------+
+----+-------------+----------+------------+------+---------------------+---------------------+
| id | customer_id | agent_id | premium_id | type | creation_time       |
modification_time   |
+----+-------------+----------+------------+------+---------------------+---------------------+
|  1 |           1 |        1 |          1 | home | 2019-05-11 18:26:30 | 2019-05-11
18:26:30 |
|  2 |           1 |        1 |          2 | auto | 2019-05-11 18:26:30 | 2019-05-11
18:26:30 |
|  3 |           1 |        1 |          3 | life | 2019-05-11 18:26:30 | 2019-05-11
18:26:30 |
+----+-------------+----------+------------+------+---------------------+---------------------+
+----------------+------------+
| Insurance Type | Amount Due |
+----------------+------------+
| home           |        110 |
| auto           |    2823141 |
| life           |        127 |
+----------------+------------+
+-----------------+------------+
| Insurance Type | Amount Due |
+-----------------+------------+
| home            |        110 |
| auto            |    2823141 |
| life            |        127 |
+-----------------+------------+
+------------+------------+------------+------------+
| name       | home_rate  | auto_rate  | life_rate  |
+------------+------------+------------+------------+
| pwning inc | 0.00025590 | 0.00282220 | 0.00032990 |
+------------+------------+------------+------------+
+-------------+------------+
| policy_type | amount_due |
+-------------+------------+
| home        |        110 |
| auto        |    2823141 |
| life        |        127 |
+-------------+------------+
```

3.3 miniSQL Input

```
insert test_scores 94 76 82 79;
retrieve test_scores;
update test_scores a1=90;
retrieve test_scores;
update test_scores a1=96 a2=78 a3=84 a4=81;
retrieve test_scores;
delete test_scores;

insert primes1 2 3 5 7;
insert primes2 11 13 17 19;
insert primes3 23 29 31 37;
insert primes4 41 43 47 53;
retrieve *;
delete primes1;
retrieve *;
delete *;
retrieve *;

retrieve not_here;
delete not_here;
insert bad_query;
update a bad query;
```

3.4 miniSQL Output

```
Query successful! Record created..
//=============||=============||=============||=============||=============\\
|| Record Name ||      a1     ||      a2     ||      a3     ||      a4     ||
|]=============[]=============[]=============[]=============[]=============[|
|| test_scores ||      94     ||      76     ||      82     ||      79     ||
\\=============||=============||=============||=============||=============//
```

```

Query successful! Record updated..
//=============||=============||=============||=============||=============\\
|| Record Name ||     a1      ||     a2      ||     a3      ||     a4      ||
|]=============[]=============[]=============[]=============[]=============[|
|| test_scores ||     90      ||     76      ||     82      ||     79      ||
\\=============||=============||=============||=============||=============//

Query successful! Record updated..
//=============||=============||=============||=============||=============\\
|| Record Name ||     a1      ||     a2      ||     a3      ||     a4      ||
|]=============[]=============[]=============[]=============[]=============[|
|| test_scores ||     96      ||     78      ||     82      ||     79      ||
\\=============||=============||=============||=============||=============//

Query successful! Record created..
Query successful! Record created..
Query successful! Record created..
Query successful! Record created..
//=============||=============||=============||=============||=============\\
|| Record Name ||     a1      ||     a2      ||     a3      ||     a4      ||
|]=============[]=============[]=============[]=============[]=============[|
||   primes1   ||      2      ||      3      ||      5      ||      7      ||
||   primes2   ||     11      ||     13      ||     17      ||     19      ||
||   primes3   ||     23      ||     29      ||     31      ||     37      ||
||   primes4   ||     41      ||     43      ||     47      ||     53      ||
\\=============||=============||=============||=============||=============//

//=============||=============||=============||=============||=============\\
|| Record Name ||     a1      ||     a2      ||     a3      ||     a4      ||
|]=============[]=============[]=============[]=============[]=============[|
||   primes2   ||     11      ||     13      ||     17      ||     19      ||
||   primes3   ||     23      ||     29      ||     31      ||     37      ||
||   primes4   ||     41      ||     43      ||     47      ||     53      ||
\\=============||=============||=============||=============||=============//

[WARNING]: Query returned 0 results..
[WARNING]: Query returned 0 results..
[ERROR]: record does not exist
[ERROR]: insert command requires 4 values
```

CHAPTER 7

storage specification

Records are stored in a database file, encoded as binary data.

record storage format:

```
<record_name>\t<a1>\t<a2>\t<a3>\t<a4>\r\n
```

miniSQL data constraints:

storage type	**data type**	**size**
field name	string	18 B
field data	unsigned integer	4 B * 4
tab char	char	1 B * 4
CRLF string	char	2 B
total record size		40 B

CHAPTER 5

miniSQL module

class `minisql.`**`MiniDB`**

Access to DB, lower level storage operations, and higher level api (querying)

`close` ( )

Close database connection and cleanup

`createRecord` (*name*, *a1*, *a2*, *a3*, *a4*)

Create a record in the database

Parameters
- **`name`** (*`str`*) – record name
- **`a1`** (*`int`*) – field data
- **`a2`** (*`int`*) – field data
- **`a3`** (*`int`*) – field data
- **`a4`** (*`int`*) – field data

Returns True | False

`deleteRecord` (*name*)

Delete a record from the database

Parameters **`name`** (*`str`*) – record to delete

Returns True | False

`deleteRecords` ( )

Delete all records from the database

Returns True | False

`getRecord` (*name*)

Get record from database

Parameters **`name`** (*`str`*) – record name

Returns tuple(<name>, <a1>, <a2>, <a3>, <a4>) | None

`getRecords` ( )

Get aal records from database

Parameters **`name`** (*`str`*) – record name

Returns list(tuple(<name>, <a1>, <a2>, <a3>, <a4>)) | [()]

interpretQueries (*queries*)
Interpret queries as a list of commands to execute in miniSQL language

Parameters **queries** – list of queries to process

Returns False to exit, True to continue (applicable to miniSQL shell)

parseQueries (*raw_queries*)
Parse input into interpretable queries based on a SQL-like syntax
Queries are delimited by a semicolon and query commands are delimited by whitespace

Parameters **raw_queries** – command string input

Returns list of queries to execute

query (*query_string*)
Interface to query the database

Parameters **query_string** – queries to run

Returns True if they succeed, false if they fail

recordExists (*name*)
Check if database record exists

Parameters **name** (*str*) – record name

Returns True | False

updateRecord (*name*, *a1=None*, *a2=None*, *a3=None*, *a4=None*)
Update a record in the database

Parameters
- **name** (*str*) – record name
- **a1** (*int*) – field data
- **a2** (*int*) – field data
- **a3** (*int*) – field data
- **a4** (*int*) – field data

Returns True | False

class minisql.**MiniShell**
Simple shell-like interpreter for miniSQL queries

close ( )
Close database connection and cleanup

complete (*text*, *state*)
Find all possible commands to complete text

Parameters
- **text** –
- **state** –

Returns

start ( )
Start the shell command interpreter
Accept commands until user quits program

`minisql.`**`main`**`( )`

Parse command line arguments and decide how data is processed

Returns 0 on success, 1 on failure

`minisql.`**`printTable`**`(` *headers, data, num_rows=None, fmt='block'* `)`

Print in table format

Parameters

- **`headers`** – list of strings
- **`data`** – list of lists (rows)
- **`num_rows`** – int
- **`fmt`** – str format type

`minisql.`**`supportsColor`**`(` *stream* `)`

Check if terminal supports ASCII color codes
Modified method from Python cookbook, #475186

Parameters **`stream`** – file stream to check

Returns True == supported, False == not supported

`minisql.`**`validateStdinReadable`**`( )`

Check if stdin has been redirected by parent shell
See the following table for supported input types.
Note that as of python v3.3 symlinks are followed by default.

Stdin Type	Supported	Program Context
directory	no	
keyboard	yes	miniSQL shell
storage	no	
file	yes	miniSQL cmd
symlink	no	
pipe	yes	miniSQL cmd
socket	no	

Returns not readable == -1, 0 == readable in cmd, 1 == readable in shell

settings module

`settings.`**`MAJOR_RELEASE_NUMBER`**
miniSQL major release version

`settings.`**`MINOR_RELEASE_NUMBER`**
miniSQL minor release version

`settings.`**`PATCH_RELEASE_NUMBER`**
miniSQL patch release version

`settings.`**`LOGO`**
miniSQL ascii logo

`settings.`**`DB_STORAGE_FILE`**
where database records stored

`settings.`**`DB_ALLOWED_FIELDS`**
allowed field names for a database record

`settings.`**`DB_RECORD_SIZE`**
number of bytes per record

`settings.`**`DB_RECORD_NAME_SIZE`**
size of field name in bytes

`settings.`**`DB_FIELD_DATA_SIZE`**
size of field data in bytes

`settings.`**`DB_ENCODING`**
encoding used for strings

`settings.`**`SHELL_INTRO`**
intro message displayed in minisql shell

`settings.`**`SHELL_PROMPT`**
user prompt displayed in minisql shell

`settings.`**`SHELL_HISTORY_FILE`**
stores minisql shell commands for command history lookup

`settings.`**`SHELL_HISTORY_LENGTH`**

number of lines of history to store

CHAPTER 7

storage specification

record storage format:

```
<record_name>\t<a1>\t<a2>\t<a3>\t<a4>\r\n
```

miniSQL data constraints:

storage type	data type	size
field name	string	18 B
field data	unsigned integer	4 B * 4
tab char	char	1 B * 4
CRLF string	char	2 B
total record size		40 B

CHAPTER 8

Indices and tables

- *Index*
- *Module Index*
- *Search Page*

Python Module Index

Index