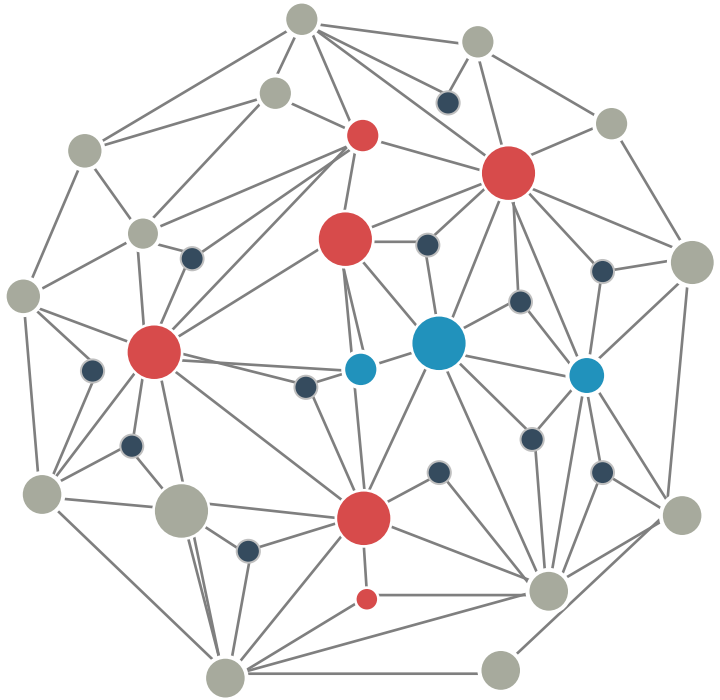
DST2 – Week 10

Advanced SQL

Zhaoyuan Fang

zhaoyuanfang@intl.zju.edu.cn



2021/11/30

Week 10 Learning Objectives

- MySQL Data types
- MySQL Built-in Functions
- Create table and insert data
- Integrity Constraints
- Functions and Procedures
- IF/ELSE and LOOP

Data types in SQL

- **Character and text:**
  - CHAR, VARCHAR, CLOB, etc
- **Numeric:**
  - Integers, decimal numbers, floating-point numbers, etc
- **Date and time:**
  - Date, datetime (timestamp), time, etc
- **Less common datatypes:**
  - Boolean: logical value (true/false)
  - Money: Money/Currency, often a decimal or floating-point number

Data types in MySQL: character and text

- CHAR(fixed_len): len<=255, fixed-length character strings
  - character strings that are always the same length
  - strings must be quoted (“”, ‘’ are the same)
  - e.g. postal codes (‘310000’, ‘325000’, …), province abbreviations (‘ZJ’, ‘SH’, ‘JS’, …)
  - Strings shorter than expected will be padded with spaces to reach the fixed length

Data types in MySQL: character and text

- VARCHAR(max_len): Variable-length character strings
  - character strings that vary in length from row to row, up to some maximum length
  - names of people (‘Tylor’, ‘Eva’, ‘Jacobson’) and companies, addresses, descriptions
  - Strings shorter than expected will not be padded with spaces

Data types in MySQL: character and text

- TEXT:
  - Character large object (other DMBSs “CLOB”)
  - Large amounts of character strings
  - e.g. entire text/XML/JSON documents
- BLOB:
  - Binary large object
  - can store images, sounds, videos, PDF files, Word files

Data types in MySQL: numeric

- Integers:
  - INT/INTEGER: no decimals, + and - allowed (-2,147,483,648 to 2, 147,483,647)
  - SIGNED [INT]: same as INT
  - UNSIGNED [INT]: only + (0 to 4,294,967,295)
  - BOOL/BOOLEAN: special short INT, 0 (false) or 1 (true)

Data types in MySQL: numeric

- Real numbers:
  - DECIMAL (m,d): fixed-point, fixed number of digits; m is total digits (1~65), d is digits right of the decimal (0~30)

    e.g. DECIMAL(8, 2) can store 10.32, 1000.10, 10000.56, 999 999.99, …
  - FLOAT: floating-point, up to 7 significant digits, less precise than DECIMAL but can store larger/smaller values
  - DOUBLE: floating-point, up to 15 significant digits, more precise than FLOAT

Data types in MySQL: date and time

- DATE:
  - Simply store dates “yyyy-mm-dd”, e.g. 2006-02-14, 2021-11-25
  - from 1000-1-1 through 9999-12-31
- TIME:
  - Store time in format of "hh:mm:ss“
  - from -838:59:59 through 838:59:59

Data types in MySQL: date and time

- DATETIME:
  - Combination of DATE + TIME, "yyyy-mm-dd hh:mm:ss“
  - From 1970-1-1 to 9999-12-31
- TIMESTAMP:
  - Similar to DATETIME, but from 1970-1-1 to 2037-12-31
- YEAR[(4)]:
  - e.g. “2021”, “2000”

TIMESTAMP vs DATETIME

- TIMESTAMP has a limit on 2037-12-31 (“year 2038 problem”), need use DATETIME if you do not want this
- TIMESTAMP can automatically change date by user time zone:
  - If a user is in the UTC+8 time zone and stores a TIMESTAMP as '2018-04-11 09:00:00' , others from UTC time zone will see this data as '2018-04-11 01:00:00‘ (-8h)

TIMESTAMP vs DATETIME

- TIMESTAMP (also DATETIME in MySQL>=5.6) can keep track of when a row was inserted or last updated
  - default current_timestamp
  - on update current_timestamp
  - default current_timestamp on update current_timestamp

  Example:

```
create table Test(id int, time datetime default current_timestamp);
insert into Test(id) values(1);
select * from Test;
drop table Test;
```

Data type conversion in MySQL: implicit/automatic

- String to number:
  - mysql> select "3.14159";
  - mysql> select "3.14159" * 1 ;
- Date to number:
  - mysql> select DATE "2021-1-1";
  - mysql> select DATE "2021-1-1" - 0;
  - mysql> select customer_id, create_date, create_date + 0 from customer limit 5;

Data type conversion in MySQL: CAST()

- Syntax:
  - **CAST**(expression **AS** cast_ type)
  - The cast_type can be one of:
    - Strings: **CHAR** [(N)]. N or max_len is optional
    - Date and time: **DATE**, **TIME**, **DATETIME**
    - Integer: **SIGNED** [INTEGER], **UNSIGNED** [INTEGER]
    - Real number: **DECIMAL** [(M [, D])], where precision and digit are optional

- Examples:
  - DECIMAL->Integer

    mysql> select payment_id, amount, CAST(amount AS UNSIGNED INT) from payment limit 5;
  - DATE->CHAR

    mysql> select customer_id, create_date, CAST(create_date AS CHAR) from customer limit 5;
  - DATE->INT

    mysql> select customer_id, create_date, CAST(create_date AS UNSIGNED INT) from customer limit 5;

Built-in Functions in MySQL: what we have already seen

- We have already learned several functions in *SELECT*:
  - **Aggregate functions:** COUNT(), SUM(), AVG(), MAX(), MIN()
  - **Comparison functions**: is null, is not null, in, like
  - **CAST**(expression AS cast_ type)
- Many more built-in functions for *SELECT*:
  - Functions for **strings**: CONCAT(), SPACE(), LENGTH(), UPPER(), LOWER(), REVERSE()
  - Functions for **numeric**: ROUND(), CEILING(), FLOOR(), RAND(), …
  - Functions for **date/time**: CURRENT_DATE(), CURRENT_TIMESTAMP(), NOW(), YEAR(), DATE_ADD(), DATE_SUB(), EXTRACT(), …
  - Functions for **flow control**: IF()

Built-in Functions: for strings

- CONCAT
  - mysql> select CONCAT('very', ' ', 'good');
  - mysql> select concat('a',space(10),'b') as result; /* a   b */
- LENGTH
  - mysql> select LENGTH('very good');
- UPPER
  - mysql> select UPPER('very good');
- LOWER
  - mysql> select LOWER('VERY GOOD');
- REVERSE
  - mysql> select REVERSE('very good');

Built-in Functions: for numeric

- **ROUND**: take the integer part; **CEILING**: upper integer; **FLOOR**: lower integer
  - select ROUND(3.14), CEILING(3.14), FLOOR(3.14);
- **ABS**: get the absolute value; **SIGN**: get the sign (1 or -1)
  - select ABS(-3.14), SIGN(-3.14), SIGN(3.14);
- **RAND([seed])**: generate a random number
  - select RAND(), RAND();
  - select RAND(123), RAND(123); /* seed */
- **POWER(a, b)**: a^b; **SQRT**: square root
  - select POWER(-3.14, 2); select(9, 0.5); select SQRT(9);

Built-in Functions: for date/time

- **CURRENT_DATE**, **CURRENT_TIME**, **CURRENT_TIMESTAMP**
  - select CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP(); /*the date/time in your own time zone */
- **UTC_DATE**, **UTC_TIME**
  - select UTC_DATE(), UTC_TIME(); /*Universal Time Coordinate (UTC) date is Greenwich Mean Time GMT*/
- **YEAR**, **MONTH**, **DAYOFMONTH**, **DAYOFWEEK**: get parts from a **date**
  - SET @t="2021-11-28 20:23:51"; /* we can SET a variable */
  - select YEAR(@t), MONTH(@t), DAYOFMONTH(@t), DAYOFWEEK(@t);
- Similar functions for **time**: HOUR(…), MINUTE(…), SECOND(…)
  - SET @t ="2021-11-28 20:23:51"; select HOUR(@t), MINUTE(@t), SECOND(@t);

Built-in Functions for date/time: EXTRACT

- **EXTRACT**: get parts from a **date**
  - Syntax: EXTRACT(unit FROM date)
- unit could be:
  - **YEAR**: year; **MONTH**: month; **YEAR_MONTH**: year and month
  - **DAY**: days; **HOUR**: hours; **MINUTE**: minutes; **SECOND**: seconds;
  - **DAY_HOUR**: day and hours; **DAY_MINUTE**: day, hours, and minutes; **DAY_SECOND**: day, hours, minutes, and seconds
  - **HOUR_MINUTE**: hour and minutes; **MINUTE_SECOND**: minutes and seconds; **HOUR_SECOND**: hours, minutes, and seconds
- **Examples**
  - SET @t="2021-11-28 20:23:51"; select EXTRACT(YEAR FROM @t), EXTRACT(DAY_MINUTE FROM @t);

Built-in Functions for date/time: calculations

- **DATE_ADD**(date, INTERVAL expression unit): returns a DATE or DATETIME value equal to the specified date plus the specified interval
  - select DATE_ADD(@t, INTERVAL 1 DAY);
  - select DATE_ADD(@t, INTERVAL 1 MONTH);
- **DATE_SUB**(date, INTERVAL expression unit): returns a DATE or DATETIME value equal to the date minus the specified interval
  - select DATE_SUB(@t, INTERVAL 1 DAY);

Built-in Functions for date/time: calculations

- **DATEDIFF**(date1, date2): returns the number of days (date1 - date2)
  - select DATEDIFF("2021-11-21", "2021-11-1");
- **TO_DAYS**(date): returns the number of days since the year 0. Not reliable for dates <1582.
  - select TO_DAYS("2021-11-21"); /* very roughly 2021 x 365 */
- **TIME_TO_SEC**(time): returns the number of seconds since midnight 00:00, useful for calculating elapsed time.
  - select TIME_TO_SEC("0:10"); /* 10 min x 60 */

Other Built-in Functions: IF

- **Syntax**
  - **IF**(test_ expression, if_true_expression, else_ expression)
- **Examples**
  - use dvdrental;
  - select title, rating, **IF**(rating!="R", "good film", “x") AS good_movie from film limit 10;

```
+-------------------+--------+------------+
| title             | rating | good_movie |
+-------------------+--------+------------+
| Academy Dinosaur  | PG     | good film  |
| Ace Goldfinger    | G      | good film  |
| Adaptation Holes  | NC-17  | good film  |
| Affair Prejudice  | G      | good film  |
| African Egg       | G      | good film  |
| Agent Truman      | PG     | good film  |
| Airplane Sierra   | PG-13  | good film  |
| Airport Pollock   | R      | x          |
| Alabama Devil     | PG-13  | good film  |
| Aladdin Calendar  | NC-17  | good film  |
+-------------------+--------+------------+
```

Create databases and tables

- Create databases
- Create table and insert data
- Integrity Constraints

Create database in SQL

- **Syntax**

  CREATE DATABASE database_name;

- **Demo**
  - mysql> CREATE DATABASE coursedb;
  - mysql> use coursedb; /* use it */
  - mysql> show tables; /* empty set*/
- **How to delete a database**
  - mysql> DROP DATABASE coursedb;
  - Do not forget re-create the database for further use

Create table in SQL

- **Syntax**

```
CREATE TABLE table_name (
        column_name_1   TYPE   column_constraints,
        column_name_2    …
);
```

- **Examples**
  - mysql> use coursedb;
  - mysql> CREATE TABLE Stu (id INT, name VARCHAR(30));

  - How to delete a table?
  - mysql> DROP TABLE Stu;

Create table in SQL: column constraints

- **Column constraints:**
  - **NOT NULL:** NULL values not allowed
  - **UNIQUE**: no duplicates
  - **AUTO INCREMENT**: e.g. for an integer column, each new insertion would add 1 to it
  - **DEFAULT default_value**: convenient to have a default value
- **Examples**
  - coursedb; drop table Stu; /* remove the previous table first */
  - CREATE TABLE Stu (id INT **NOT NULL**, name VARCHAR(30) **DEFAULT** "Not available");

Create table in SQL: primary key constraint

- **Syntax 1**

```
CREATE TABLE table_name (
        column_name_1   TYPE   PRIMARY KEY,
          …
);
```

Column constraints:

- **NOT NULL**
- **UNIQUE**
- **AUTO INCREMENT**
- **DEFAULT default value**
- **PRIMARY KEY**

- **PRIMARY KEY constraint:**
  - A valid *relation* (table) should have a primary key
  - By default, PRIMARY KEY == NOT NULL + UNIQUE
- **Examples**
  - mysql> use coursedb;   drop table Stu;
  - Mysql> CREATE TABLE Stu (id INT **PRIMARY KEY**, name varchar(30) DEFAULT "Not available");

Create table in SQL: primary key constraint

- **Syntax 2**

```
CREATE TABLE table_name (
        column_name_1   TYPE,
          …,
        [CONSTRAINT name] PRIMARY KEY (column_name_1)
);
```

Column constraints:

- **NOT NULL**
- **UNIQUE**
- **AUTO INCREMENT**
- **DEFAULT default value**
- **PRIMARY KEY**

- **Table-level constraint**
  - This syntax can also be used for UNIQUE constraint:
  - e.g. CONSTRAINT uq UNIQUE(column_name)
- **Examples**
  - use coursedb; drop table Stu;
  - CREATE TABLE Stu (stu_id int NOT NULL, name varchar(30) DEFAULT "Not available", PRIMARY KEY (stu_id));
  - CREATE TABLE Stu (stu_id int NOT NULL, name varchar(30) DEFAULT "Not available", **CONSTRAINT** pk PRIMARY KEY (id));

Create table in SQL: foreign key constraint

- **Syntax**

```
CREATE TABLE table_name (
        column_name_1   TYPE,
          …,
         [CONSTRAINT name] FOREIGN KEY (column_name_1)
                                REFERENCES table_name_2 (column_name_1)
);
```

- **Examples**
  - mysql> use coursedb;
  - mysql> CREATE TABLE Class (course_id int PRIMARY KEY, stu_id int NOT NULL, FOREIGN KEY (stu_id) REFERENCES Stu(stu_id));

Create table in SQL: another example

DEMO:

```
CREATE TABLE dst2employee(
        id      INT             NOT NULL,
        name    VARCHAR (20)    NOT NULL,
        age     INT             NOT NULL,
        address     CHAR (25) ,
        salary DECIMAL (18, 2),
        PRIMARY KEY (id));

SELECT * FROM dst2employee;
```

Column constraints:

- **NOT NULL**
- **UNIQUE**
- **AUTO INCREMENT**
- **DEFAULT default value**
- **PRIMARY KEY**

Create table in SQL: task

Task 1 : Let's create a table called dst2studentaccount with:

student_id (serial, primary key),
first_name (variable character less than 50 in length, must be unique),
last_name (variable character less than 50 in length),
email variable character less than 355 in length and must be unique,
create_on (timestamp),
last_login (timestamp).

```
CREATE TABLE dst2studentaccount(
   student_id      serial  PRIMARY KEY,     /* serial: unsigned not null unique auto_increment */
   first_name      VARCHAR (50) UNIQUE NOT NULL,
   last_name       VARCHAR (50) NOT NULL,
   email           VARCHAR (355)          UNIQUE NOT NULL,
   created_on      TIMESTAMP      NOT NULL,
   last_login      TIMESTAMP
);
SELECT * FROM dst2studentaccount;
```

Integrity Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- Examples:
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $4.00 an hour
  - A customer must have a (non-null) phone number

Constraints on single relations

- **primary key**
- **not null**
- **unique**
- **check** (*P*), where *P* is a predicate

The check clause

- **check** (*P*), where *P* is a predicate

  CHECK for CREATE TABLE :

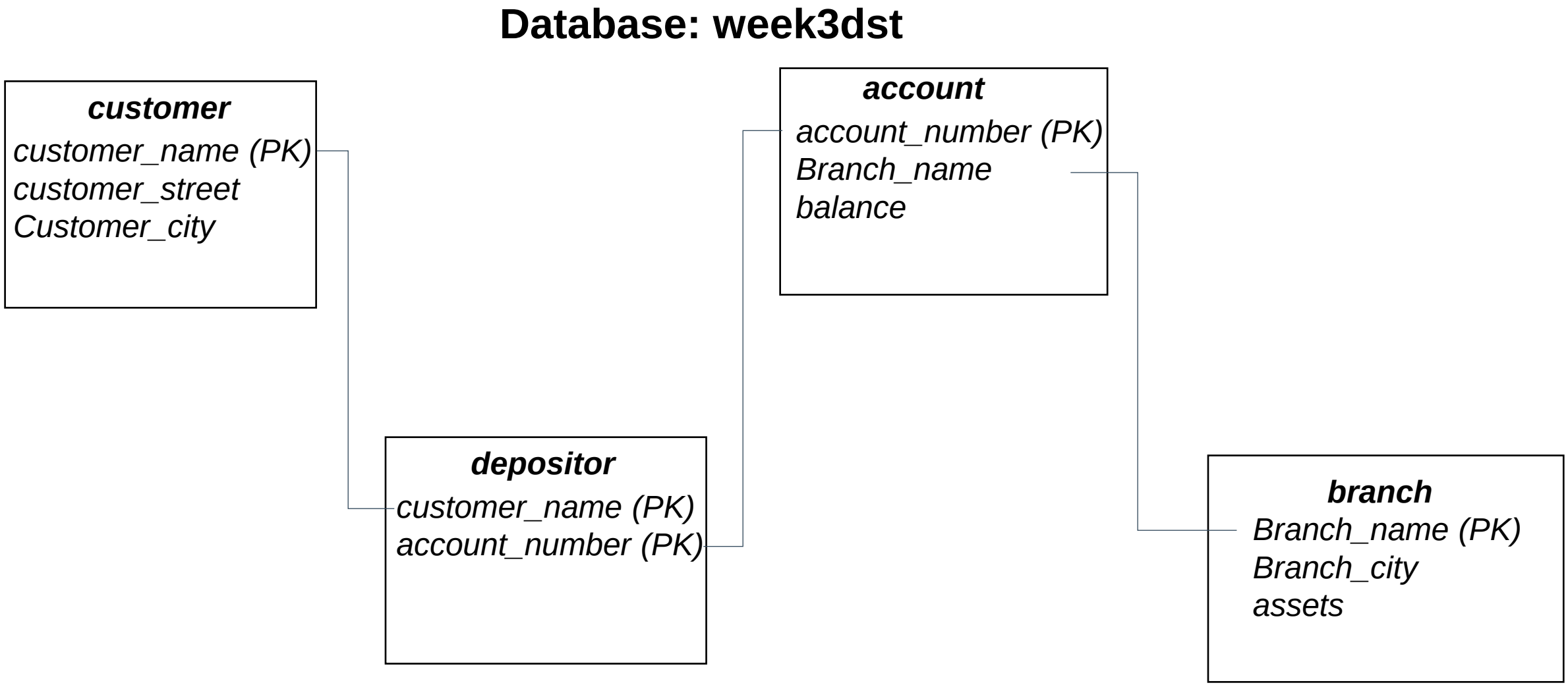
  e.g. Declare branch_name as the primary key for branch and ensure that the values of assets are non-negative.

```
create table branch
        (branch_name     char(15),
         branch_city     char(30),
         assets          int,
         primary key (branch_name),
         check (assets >= 0))
```

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If “Perryridge” is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch “Perryridge”.
- Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:
  - The primary key clause lists attributes that comprise the primary key.
  - The unique key clause lists attributes that comprise a candidate key.
  - The foreign key clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key. By default, a foreign key references the primary key attributes of the referenced table.

Create database - Example



Create database - Example

First, let's create a new database called '**week3dst**' : create database week3dst; use week3dst;

```
CREATE TABLE customer
(customer_name          char(20),
 customer_street        char(30),
 customer_city          char(30),
 PRIMARY KEY(customer_name));
```

```
CREATE TABLE branch
(branch_name            char(15),
 branch_city            char(30),
 assets                 numeric(12,2),
 PRIMARY KEY (branch_name));
```

Tip: see slide notes for SQL

```
CREATE TABLE account
(account_number         char(10),
 branch_name            char(15),
 balance                integer,
 PRIMARY KEY (account_number),
 FOREIGN KEY (branch_name) REFERENCES branch(branch_name));
```

```
CREATE TABLE depositor
(customer_name          char(20),
 account_number         char(10),
 PRIMARY KEY (customer_name, account_number),
 FOREIGN KEY (account_number) REFERENCES account(account_number),
 FOREIGN KEY (customer_name) REFERENCES customer(customer_name));
```

Then, use **SELECT** to check your tables:

```
SELECT * FROM account; SELECT * FROM branch; SELECT * FROM depositor; SELECT * FROM customer;
```

Populating a Table With Rows

The INSERT statement is used to populate a table with rows:

INSERT INTO table_name (column_1, column_2, …) VALUES (value_column_1, value_column_2, …);

DEMO:

Add one tuple for each table:

```
INSERT INTO customer (customer_name, customer_street, customer_city)
        VALUES ('Tylor', 'Main', 'Haining');

INSERT INTO branch (branch_name, branch_city, assets)
        VALUES ('CCBHN', 'Haining', 200000.00);

INSERT INTO account (account_number, branch_name, balance)
        VALUES ('1', 'CCBHN', 200.00);

INSERT INTO depositor (customer_name, account_number)
        VALUES ('Tylor', '1');
```

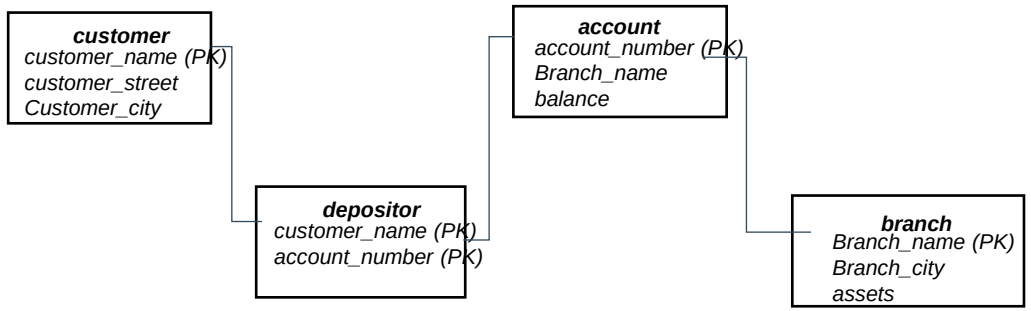
Populating a Table With Rows

Task 2 : Insert records into the **week3dst** database with the following information.

Customer_name = ‘Smith’, customer_street = ‘First’, cusomter_city =‘Hangzhou’,

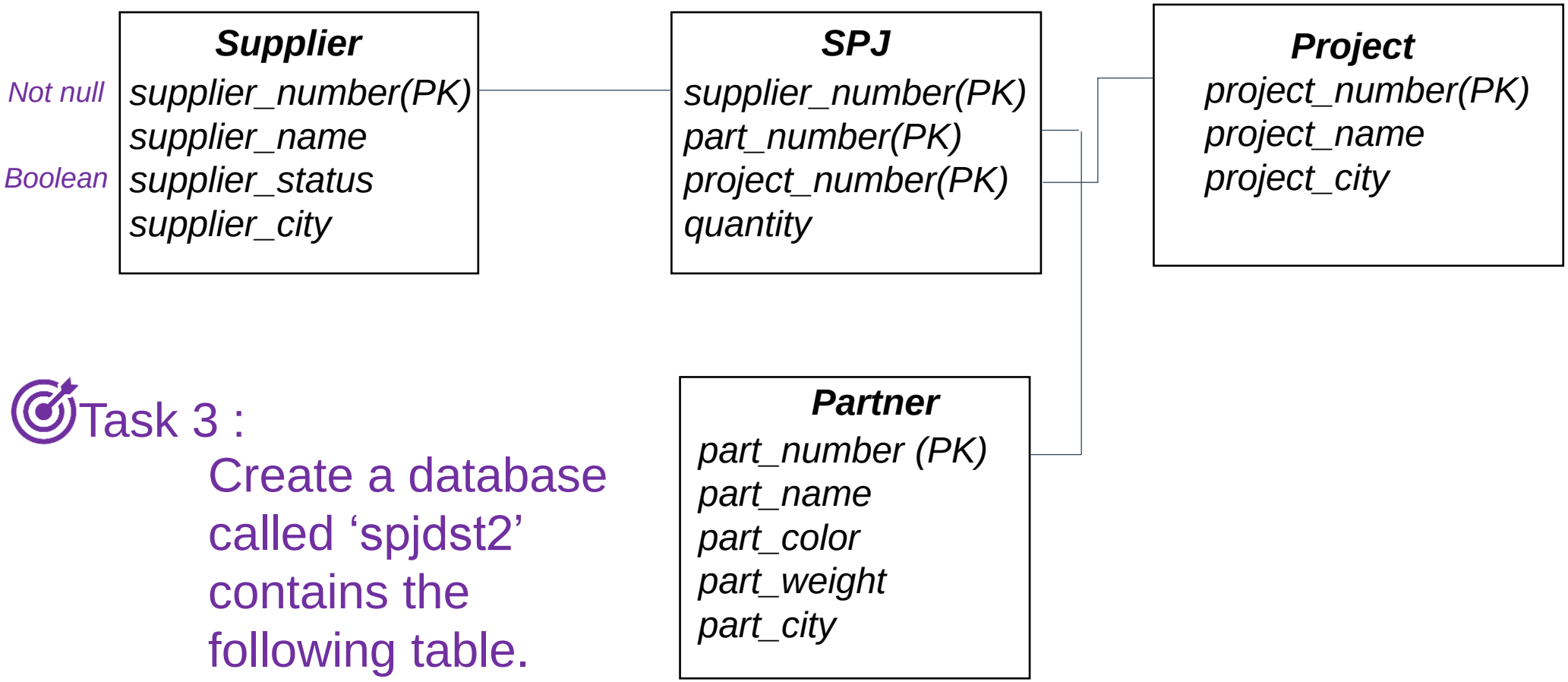
Branch_name = ‘CITIHZ’, branch_city= ‘Hangzhou’, assests = 1000000.00

Account_number=2, balance = 1000.00




```
INSERT INTO customer (customer_name, customer_street, customer_city)
        VALUES (‘Smith’, ‘First’, ‘Hangzhou’);
INSERT INTO branch (branch_name, branch_city, assets)
        VALUES (‘CITIHZ’, Hangzhou’, 1000000.00);
INSERT INTO account (account_number, branch_name, balance)
        VALUES (‘2’, ‘CITIHZ’, 1000.00);
INSERT INTO depositor (customer_number, account_number)
        VALUES (‘Smith’, ‘2’);
```

Create database – Task3



Task 3 :

Create a database called ‘spjdst2’ contains the following table.

```
create database spjdst2;    use spjdst2;

CREATE TABLE Supplier
(supplier_number        int,
 supplier_name          varchar(30),
 supplier_status        boolean,
 supplier_city          varchar(30),
 PRIMARY KEY(supplier_number));

CREATE TABLE Partner
(part_number            int,
 part_name              varchar(20),
 part_color             varchar(30),
 part_weight            decimal(20,2),
 part_city              varchar(10),
 PRIMARY KEY (part_number));

CREATE TABLE Project
 (project_number        int,
  project_name          varchar(15),
  project_cit           int,
  PRIMARY KEY (project_number));

CREATE TABLE SPJ
 (supplier_number       int,
  part_number           int,
  project_number        int,
  quantity              int,
  PRIMARY KEY (supplier_number,part_number,project_number),
  FOREIGN KEY (supplier_number) REFERENCES
                        Supplier(supplier_number),
 FOREIGN KEY (part_number) REFERENCES Part(part_number),
 FOREIGN KEY (project_number) REFERENCES
                        Project(project_number)));
```

Tip: you need type it yourself in MySQL (to remove the formatting that can raise errors)

Customized Functions and Procedures

- We have seen many built-in functions
- MySQL also permits user-defined routines
  - Stored procedures, Functions
  - with if-then-else statements, loops, etc.
- e.g. Stored Procedures
  - Can store procedures in the database
  - then execute them using the **call** statement
  - permit external applications to operate on the database without knowing about internal details

Functions and Procedures

- Functions and Procedures are both stored routines
- Functions return values and can be run like the built-in functions
- Procedures do not return values (IN/OUT/INOUT parameters) and can be run using CALL keyword

- Besides, SQL also supports a rich set of imperative constructs, including: loops, if-then-else, assignment

MySQL Functions

DEMO:

- Define a function that adds two integers.

```
delimiter //      /*   change default delimiter from ";" to "//",   */
                  /*   so that we can use ";" within the function body   */
CREATE FUNCTION   my_add(x integer, y integer)
RETURNS integer
DETERMINISTIC    /* state same result on same input, not essential */
   BEGIN
      RETURN  x+y;      /* now we can use ";" inside a function */
   END //               /* the new delimiter "//" */
delimiter ;            /* change the delimiter from "//" back to default ";" */

/* Now we can use the customized function my_add()  */
SELECT my_add(1,2);
```

MySQL Functions

Task 4 :

Write a function to convert RMB to USD. Let's assume current currency is 1USD=7RMB. Then calculate how much is 100RMB in USD?

```
delimiter //
CREATE FUNCTION RMB2USD(x decimal(20,2))
RETURNS decimal(20,2)
DETERMINISTIC
BEGIN
    RETURN x/7;
END //
delimiter ;

SELECT RMB2USD(100);
```

DEMO:

```
delimiter //
CREATE FUNCTION
my_add(x integer, y integer)
RETURNS integer
DETERMINISTIC
BEGIN
        RETURN x+y;
END //
delimiter ;
```

MySQL Procedures: simple

DEMO:

- Define a procedure that select two columns (film id, film title) from table film.

```
delimiter //          /*    change default delimiter from ";" to "//",   */
                      /*    so that we can use ";" within the function body   */
CREATE PROCEDURE  get_film()
DETERMINISTIC    /* state same result on same input, not essential */
   BEGIN
      select film_id, title from film;   /* now we can use ";" inside function */
   END //                 /* the new delimiter "//" */
delimiter ;               /* change delimiter from "//" back to default ";" */

/* Now we can use the customized procedure get_film ()  */
CALL get_film();
```

MySQL Procedures: with one argument

DEMO:

- Define a procedure that select retrieve all films that matches a given film_id.

```
delimiter //

CREATE PROCEDURE  get_film_by_id(x int)   /* parameter */
DETERMINISTIC    /* state same result on same input, not essential */
    BEGIN
        select * from film where film_id = x;
    END //
delimiter ;

/* Now we can use the customized procedure get_film_by_id ()  */
CALL get_film_by_id(1);
CALL get_film_by_id(2);
```

MySQL Procedures: task

Task 5: Write a procedure for the film table, returns all films whose titles match a particular pattern using LIKE operator.

```
delimiter //

CREATE PROCEDURE  get_film_by_pattern(x varchar(50))
    BEGIN
        select * from film where title like x;
    END //

delimiter ;

CALL get_film_by_pattern('%Garden');
```

IF THEN ELSE Statement

The IF-THEN-ELSE statement: executes a command when the condition is true, or an alternative command when the condition is false.

```
IF condition THEN
  statements;
[ELSEIF condition THEN
  statements;]
ELSE
  alternative-statements;
END IF;
```

DEMO:

```
delimiter //
CREATE PROCEDURE my_compare(a int, b int)
DETERMINISTIC
BEGIN
IF a>b THEN select "a is larger than b" as result;
ELSEIF a=b THEN select "a equals b" as result;
ELSE select "a is smaller than b" as result;  END
IF;
END//
delimiter ;

CALL my_compare(10,20);
CALL my_compare(20,10);
```

IF THEN ELSE Statement

Task 6: Write a if-else statement to compare two dates. (whether date1 is earlier, or the same date or later than date2.)

```
delimiter //
CREATE PROCEDURE my_compare_date(a date, b date)
BEGIN
DETERMINISTIC
IF a>b THEN select "date1 is later than date2" as result;
ELSEIF a=b THEN select "date1 equals date2" as result;
ELSE select "date1 is earlier than date2" as result;
END IF;
END//
delimiter ;
```

```
mysql> CALL  my_compare_date("2021-9-20","2021-10-20");
mysql> CALL  my_compare_date("2021-11-20","2021-10-20");
```

Simple Loop Statement

Loop is common in many programming languages. It will execute a sequence of steps until a certain condition is reached.

MySQL supports LOOP within a function or procedure. (Other loop keywords are WHILE & REPEAT).

DEMO:

A function to calculate the sum from 1 to N

```
delimiter //
CREATE PROCEDURE cumsum(N int)
BEGIN
DETERMINISTIC
DECLARE s int default 0;   /*local variables*/
DECLARE i int default 1;
my_loop : LOOP
   SET s=s+i;
   select i as “added”, s as “result”;
   SET i=i+1;
   IF i>N THEN LEAVE my_loop; END IF;
END LOOP;
END //
delimiter ;

CALL cumsum(100);
```

Simple Loop Statement

Task 6: A function to calculate the cumulative product of n number (using simple LOOP).

```
delimiter //
CREATE PROCEDURE cumprod (n int)
BEGIN
DECLARE s int default 1;
DECLARE i int default 1;
my_loop : LOOP
   SET s=s*i;
   select i as “added”, s as “result”;
   SET i=i+1;
   IF i>n THEN  LEAVE my_loop;  END IF;
END LOOP;
END //
delimiter ;
```

```
CALL cumprod(100);
```

Summary

- MySQL Data Types and Built-in Functions
  - char/varchar, numeric/int/decimal/float/double, date/time/datetime/timestamp
  - Conversion between data types
  - Built-in Functions
- Create table and insert data in SQL
  - How to create table and insert data into SQL
- Integrity Constraints
  - Not null, primary key, unique, check
- Customized Functions and Procedures
  - Create functions and procedures in SQL
  - IF/ELSE, simple LOOP