

Piazza Resources

Relevant: Chapter 4, sections 1-6 of "Database Systems – The Complete Book"

Sample question 1

CREATE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
```

create_definition:

```
col_name column_definition
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (
    ↪ index_col_name,...)
[index_option] ...
| {INDEX|KEY} [index_name] [index_type] (index_col_name
    ↪ ,...)
[index_option] ...
| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
[index_name] [index_type] (index_col_name,...)
[index_option] ...
| {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (
    ↪ index_col_name,...)
[index_option] ...
| [CONSTRAINT [symbol]] FOREIGN KEY
[index_name] (index_col_name,...) reference_definition
| CHECK (expr)
```

column_definition:

```
data_type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string']
[COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
[STORAGE {DISK|MEMORY|DEFAULT}]
[reference_definition]
```

Create Table Using Another Table

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

Select Syntax

```
SELECT col1, col2, col3, ..., FROM table1
WHERE col4 = 1 AND col5 = 2
GROUP BY # aggregate the data
HAVING count(*) > 1 # limit aggregated data
ORDER BY col2
```

Insert Syntax

```
# insert values manually
INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
VALUES (1, Rebel, Labs);
# or by using the results of a query
INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
SELECT id, last_name, first_name FROM table2
```

Alter table Syntax

```
# add a column
alter table tablename
add column_name datatype
# drop a column
alter table tablename
drop col1, drop col2
# alter a column's datatype
alter table tablename
alter column column_name datatype / modify column
    ↪ column_name datatype auto_increment
# add a constraint
alter table tablename
add constraint fk_foreign_key (col) references tblname2
    ↪ (col2)
```

Update Syntax

```
UPDATE table1
SET table1.col1 = 1
FROM table2
WHERE col2 = 2
```

Create index Syntax

```
create index idx_name
on table_name(col)
```

Constraint Syntax

Sample Solution 1

```
2. create table addresses (
    id int auto_increment unique not null,
    street_name varchar(100) not null,
    city varchar(20) not null,
    region varchar(20),
    postal_code int(5),
    country varchar(20) not null,
    constraint uc_address unique(street_name, city,
    ↪ region, postal_code, country),
    constraint pk_addresses primary key(id)
)
```

```
3. insert addresses(addresses, city, region,
    ↪ postal_code, country)
    select addresses, city, region,
    ↪ postal_code, country
    from customers
    group by addresses, city, region, postal_code
    ↪ , country
OR
insert into addresses(...)
    select distinct address, city, region,
    ↪ postal_code, country from customers
```

```
4. # Clone a table
create table new_customers like customers
# Load the data
insert into new_customers
select * from customers
```

```
5. alter table new_customers
add column address_id int
```

```
6. update table new_customers
set new_customers.address_id = addresses.id
from new_customers join addresses on
new_customers.address = addresses.street_name
and new_customers.city = addresses.city
and new_customers.region = addresses.region
and new_customers.postal_code = addresses.
    ↪ postal_code
and new_customers.country = addresses.country
OR
'update 'Q1E4RTYU7IOP[-\te new_customers
set address_id=(select id from address where
(...AND (... OR (addresses.region is null or
    ↪ new_customers.region is null)))
```

```
7. alter table addresses
drop column street_name,
drop city,
drop region,
drop postal_code,
drop country
```

```
8. alter table new_customers
add constraint fk_new_customers_addresses foreign
    ↪ key (address_id)
references addresses(id);
create index idx_new_customer_id on
new_customers(id)
```

Sample question 2

Define a datamodel using the notation we used in class. You should define either a logical or physical model. From your model, create the SQL DDL. The datamodel is the following.

- Companies: These are businesses and have properties Name, ID. The ID should be unique and derived from the company name. Companies also have an address.
- Persons: A Person has a last_name, first_name and middle initial. A Person also has an email. A Person also has an address.
- A Company may be related to one or more Persons. A Person may be related to one or more Companies. A Person-Company relationship has a type, e.g. Employee, Contractor, Consultant. The data model should ONLY allow creation of relationships that conform to one of the types. It must be possible to add new, named types.

Trigger Syntax

Example:

```
create definer = 'root' @ 'localhost' trigger '
    ↳ University'. 'enrollment_before_insert' before
    ↳ insert on 'enrollment' for each row
begin
    declare new_day varchar(1);
    declare new_start int;
    declare new_end int;

    select sections.
```

Function Syntax

```
create function function_name(param1, param2 ...)
    returns datatype
    [not] deterministic
statements
```

Example:

```
drop function if exists customerlevel;
DELIMITER $$

CREATE FUNCTION CustomerLevel(p_creditLimit double)
    ↳ RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);

    IF p_creditLimit > 50000 THEN
        SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >=
        ↳ 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;
```

```
RETURN (lvl);
END

create definer='root'@'localhost' trigger 'University
    ↳ '..'person1_BEFORE_INSERT' before insert on '
    ↳ person1' for each row
begin
    set new.uni = generate_uni_1(new.last_name, new
    ↳ .first_name)
end

create definer='root'@'localhost' function '
    ↳ generate_uni'(last_name varchar(32), first_name
    ↳ varchar(32)) returns varchar(8) charset utf8
begin
    declare c1 char(2);
    declare c2 char(2);
    declare prefix char(5);
    declare uniCount int;
    declare newUni varchar(6);

    set c1 = upper(substr(last_name, 1, 2));
    set c2 = upper(substr(first_name, 1, 2));
    set prefix = concat(c1, c2, '%');

    select count(uni) into uniCount from person1 where
    ↳ uni like prefix;
    set newUni = concat(c1, c2, uniCount);
    return newUni;
end
```

Sample Solution 2

```
# Create companies
create table companies(
    id varchar(6) not null,
    name varchar(100) not null,
    address varchar(200),
    constraint pk_companies primary key(id)
)

delimiter $$
create definer='root'@'localhost' trigger 'companies'.
    ↳ companies_before_insert' before insert on
    ↳ companies for each row
begin
    set new.id = generate_id(new.name);
end$$

create function generate_id (name varchar(100))
    returns varchar(6)
BEGIN
    declare prefix varchar(3);
    declare idcount int(3);
    declare newId varchar(6);
    set prefix = upper(substr(name, 1, 3));
    select count(id) into idcount from companies where
    ↳ id like prefix;
```

```
set newId = concat(prefix, idcount);
return newId;
END $$
delimiter ;

# Use the procedure to create person
create table persons(
    id int not null auto_increment
    last_name varchar(20) not null,
    first_name varchar(20) not null,
    initial varchar(2),
    email varchar(100),
    address varchar(100),
    typeid int,
    constraint pk_persons primary key (id),
    constriant fk_persons_type foreign key(typeid)
);
delimiter $$
create trigger persons_before_insert before insert on
    ↳ persons for each row
begin
    call procedure(new.last_name, new.first_name,
    ↳ @initial);
    set new.initial = @initial;
end$$

create procedure generate_initial(in last_name varchar
    ↳ (20) first_name varchar(20) ,out initial
    ↳ varchar(2))
begin
    set initial = concat(upper(substr(last_name, 1,
    ↳ 1)), upper(substr(first_name, 1, 1)))
end$$
delimiter ;
```

Sample Question 3

Sample Solution 3

```
TO COPY DON'S CODE
create view valid1 as select *
from
(select dish as dish1 from appetizers) as a1 join
(select dish as dish2 from appetizers) as a

create view validorder as
select * from valid1
union
select * from valid2
union
select * from valid3

create procedure

create function # function can update data like insert

create trigger
```