Databases Design. Introduction to SQL

LECTURE 5

SQL
Data Definition Language

Review of last lecture

- Normalization and Normal Forms
- 3NF relation is considered a «good» database design
- Improvement of a database design

Database Design stages

- Subject Area Analysis
- Conceptual Design
- Logical Design
- Physical Design

SQL

- **SQL (Structured Query Language)** is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS).
- Based upon relational algebra, SQL includes a **data definition language (DDL)** and a **data manipulation language (DML)**.

SQL DDL

Data Definition Language (DDL) defines constructs that structure the data in the database.

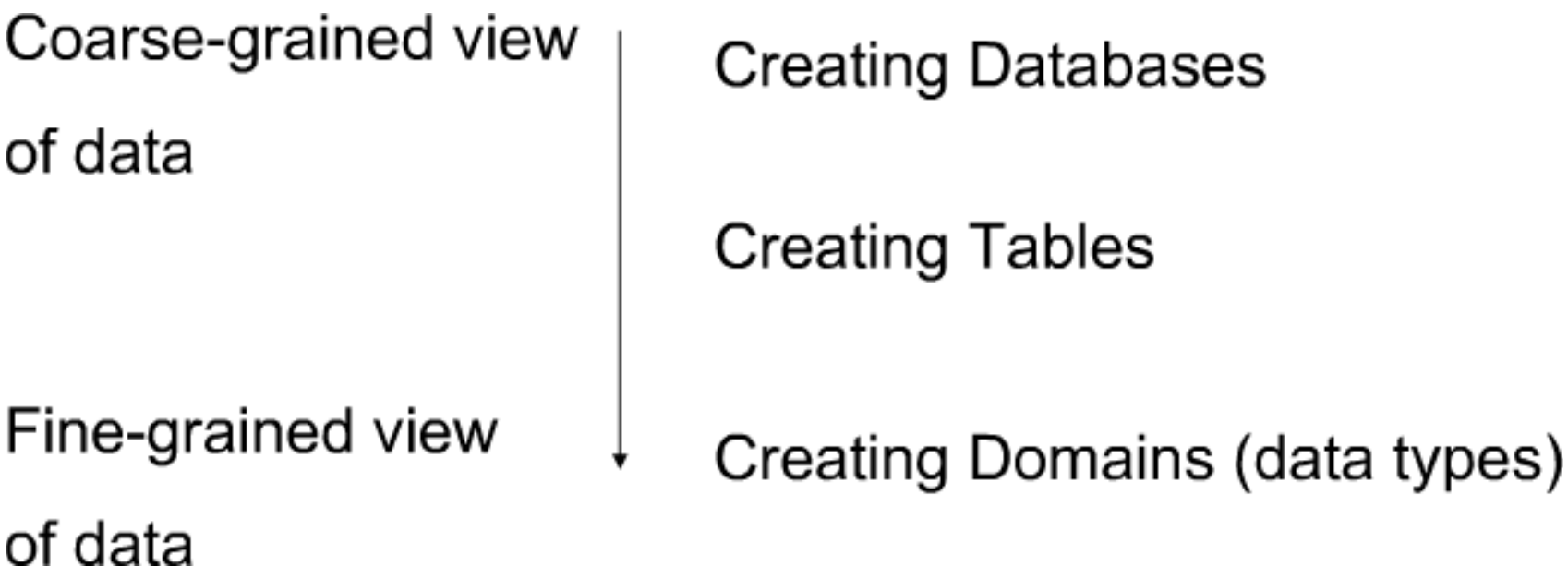
DDL statements are used to build and modify the structure of your tables and other objects in the database:

- CREATE DB
- CREATE TABLE
- ALTER TABLE
- DROP TABLE

- Note: the dialect of SQL supported by PostgreSQL will be used here.

Top-Down view of SQL DDL

- At the ‘top’ a database is created
- Further down the hierarchy, a set of tables are created
- At the bottom of the hierarchy data types are created



Creating a Database

PostgreSQL has the CREATEDB command that creates the database.

The create schema command takes two arguments

- database name
- owner of the database

Creating a Table

The CREATE TABLE statement allows to define

- name of the table
- name of each column
- domain of each column
- constraints on the columns (keys and other constraints)

Creating a Table

- Syntax:

```
CREATE TABLE table_name (
    column1name column1domain,
    column2name column2domain,…,
    columnNname columnNdomain,
    PRIMARY KEY (pkcolumn(s)),
    FOREIGN KEY (column) REFERENCES
table(column) );
```

CREATE TABLE: example

```
CREATE TABLE Groups(
group_id  int,
group_name varchar(15),
PRIMARY KEY (group_id));
```

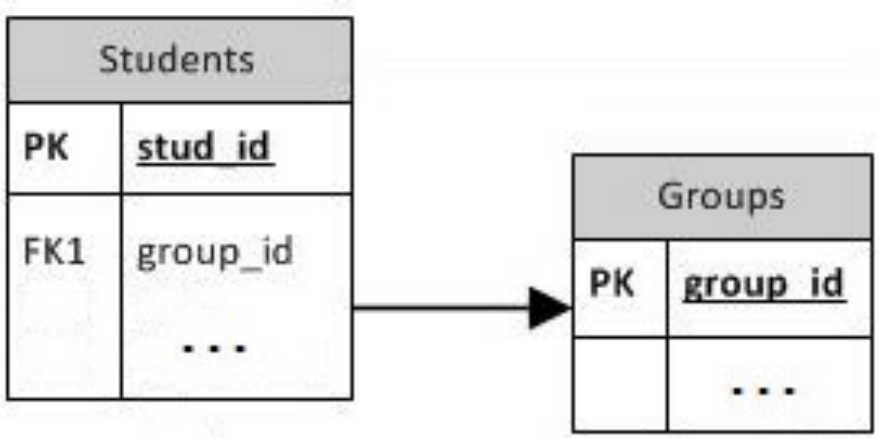
or

```
CREATE TABLE Groups(
group_id int PRIMARY KEY ,
group_name varchar(15));
```

CREATE TABLE: example with FK

```
CREATE TABLE Groups(
group_id  int,
group_name varchar(15),
PRIMARY KEY (group_id));

CREATE TABLE Students(
stud_id int,
first_name varchar(20),
last_name varchar(20),
group_id int,
PRIMARY KEY (stud_id),
FOREIGN KEY (group_id) REFERENCES Groups(group_id));
```



CREATE TABLE: example with FK

```
CREATE TABLE Students(
stud_id int PRIMARY KEY,
first_name varchar(20),
last_name varchar(20),
group_id int,
FOREIGN KEY (group_id) REFERENCES Groups(group_id));
```

or

```
CREATE TABLE Students(
stud_id int PRIMARY KEY,
first_name varchar(20),
last_name varchar(20),
group_id int REFERENCES Groups(group_id));
```

Constraints

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.
The available constraints in SQL are:

- **NOT NULL**: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
- **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY**: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
- **FOREIGN KEY**: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
- **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.

Defining Constraints

In addition to PK and FK constraints the following types of constraints can also be added:

- CHECK
- NOT NULL
- UNIQUE

CHECK

- **Check constraints** tell the DBMS the acceptable values for a column

- We can build this constraint using the CHECK keyword in a CREATE TABLE statement.

CHECK example

- Consider the bank account example. One integrity constrain could be that balances must be positive.

```
CREATE TABLE account(
    id integer,
    balance float CHECK (balance>0),
    PRIMARY KEY (id));
```

NOT NULL

- **NOT NULL constraints** ensures values exist in all rows for a given column.

- Suppose we define *balance* to be NOT NULL in the Account table.
- Anytime we insert an Account record, a *balance* must be defined. Otherwise, an error is thrown.

- PKs have an implicit NOT NULL constraint.

NOT NULL example

- Query the ACCOUNT table such that balances have a not-null constraint:

```
CREATE TABLE account (
    id integer,
    balance float NOT NULL,
    PRIMARY KEY (id));
```

NOT NULL with CHECK

- Query the ACCOUNT table such that balances have a not-null constraint:

```
CREATE TABLE account (
    id integer,
    balance float NOT NULL CHECK (balance>0),
    PRIMARY KEY (id));
```

UNIQUE

- **Unique constraints** ensure that values in columns are unique.
- Unique allows to model **alternate key** (secondary key).
- One or more columns may be defined as unique – so the combination of two columns may be unique, but the two columns themselves need not be unique.

UNIQUE example

- If the CUSTOMER table had unique names – then Name is an **alternate key**.
- We can create this CUSTOMER table as

```
CREATE TABLE Customer (
    id integer,
    name varchar(6),
    PRIMARY KEY (id),
    UNIQUE (name));
```

UNIQUE example

```
CREATE TABLE Customer (
id int,
name varchar(6),
PRIMARY KEY (id),
UNIQUE (name));
```

or

```
CREATE TABLE Customer (
id int PRIMARY KEY,
name varchar(6) UNIQUE);
```

Data types

SQL allows columns to be defined as one of five main classes of data:

- Numeric
- Character strings
- Bit strings
- Temporal Data
- Boolean Data

Numeric Data

Exact numbers may be INTEGER (or INT), SMALLINT, BIGINT

- Like the C programming language's short data type, SMALLINT ranges between -32768 to 32767 inclusive.
- INTEGER ranges between -2,147,483,648 and 2,147,483,647 inclusive.
- BIGINT ranges between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807 inclusive.

Numeric Data

- Approximate numbers are numbers that cannot be represented exactly, such as real numbers (pi).
- We represent such numbers as floating-point values of various precisions (numbers of decimal places).

Character Strings

- Character strings are sequences of printable characters

- All character strings in SQL are started and ended using single quotes. For example, ‘string’ is a valid SQL string.

- Character strings come in two flavors:
  - Fixed-length strings
  - Variable-length strings

Character Strings

- Fixed-length character strings are defined to be of a given length, say 10 characters.
  - All values in the column of this type have 10 characters.
  - If any rows have less than 10 characters, they are padded with spaces to fill up the space.

- Columns of this type are defined as char(n) where n is the length of the string. So, a ten-character string is defined as char(10).

- The default length is 1, so char defines a 1-character column.

Character Strings

- Variable-length character strings have a maximum length, like fixed-length character strings, but, unlike fixed-length strings, variable-length strings that are shorter than the maximum length are not padded with spaces.

- Variable-length strings are know as *varchars*. We define a variable length string with a maximum of 10 characters as varchar(10).

Temporal Data

- SQL provides support for storing date and time data.
- All SQL implementations support the DATE data type.
- PostgreSQL supports timestamp, interval, date, time and time with time zone types.

Temporal Data

Type	Description	Example	Earliest	Latest
timestamp	Stores both date and time.	1999-01-08 04:05:06	4713 BC	AD 1465001
interval	Stores time intervals.	'1 12:59:10' read as 1 day 12 hours, 59 minutes, 10 seconds	-178000000 years	178000000 years
date	Stores dates only.	1999-01-08	4713 BC	32767 AD
Time	Times of day	04:05:06	00:00:00.00	23:59:59.99
time with time zone	Times of day.	04:05:06-12	00:00:00.00+12	23:59:59.99 -12

Boolean Data Types

- PostgreSQL (and most other dialects of SQL) support the boolean data type.

- Valid forms of true are:

  TRUE, ‘t’, ‘true’, ‘y’, ‘yes’, ‘1’
- Valid forms of false are:

  FALSE, ‘f’, ‘false’, ‘n’, ‘no’, ‘0’

Altering a Table

- When you create a table and you realize that you made a mistake, or the requirements of the application change, you can drop the table and create it again. But this is not a convenient option if the table is already filled with data, or if the table is referenced by other database objects (for instance a foreign key constraint). Therefore PostgreSQL provides a family of commands to make modifications to existing tables.

- ALTER TABLE command is used to modify a structure of an existing table.

Altering a Table

The syntax is

```
ALTER TABLE table_name …;
```

Possible modifications:

- Add / remove columns
- Add / remove constraints
- Change column data types
- Rename columns / tables
- Etc.

Add column

- Suppose we wanted to add a column to bank database's account table that stored the data the account was opened. The original account table was created as

```
CREATE TABLE account (
        id integer,
        balance float,
        PRIMARY KEY (id));
```

Add column

- The syntax is

  ALTER TABLE table_name ADD COLUMN column_name datatype;

- So, to add the opening date of an account, we write the following query:

  ALTER TABLE account ADD COLUMN opendate date;

Add column with constraints

- The syntax is

  ALTER TABLE table_name ADD COLUMN column_name datatype constraint;

- So, to add the opening date of an account, we write the following query:

  ALTER TABLE account ADD COLUMN acc_value int CHECK (acc_value < 0);

Drop column

- Removing a column: the DROP COLUMN statement is used with ALTER command

- The syntax is:

  ALTER TABLE table_name DROP COLUMN column_name;

- So, to drop the opendate column of account, we write:

  ALTER TABLE account DROP COLUMN opendate;

Data type

- The basic syntax of ALTER TABLE to change the data TYPE of a column in a table is as follows:

```
ALTER TABLE table_name ALTER COLUMN column_name TYPE datatype;
```

- Example:

```
ALTER TABLE account ALTER COLUMN opendate TYPE varchar(15);
```

Rename column

- Rename a column: use RENAME COLUMN statement in the ALTER TABLE command.

- To rename the Account table's Balance column to AccountBalance we write:

```
ALTER TABLE account RENAME COLUMN
balance TO accountbalance;
```

Rename table

- Renaming a table: use the RENAME keyword in the ALTER TABLE command.

- To rename the Account table to Bankaccount, we write:

```
ALTER TABLE account RENAME TO
bankaccount;
```

Add foreign key

- SQL DDL also allows us to add constraints to tables using the ALTER TABLE command. We can add key, unique, not-null, and check constraints.

- In the bank example, suppose we had the Customer and Account tables as before, but we did not place foreign keys on the tables.
- Query to add foreign key:

```
ALTER TABLE customer ADD FOREIGN KEY
(accountId) REFERENCES account (id);
```

Add and drop NOT NULL

- The basic syntax of ALTER TABLE to add a NOT NULL constraint to a column in a table is as follows:

```
ALTER TABLE table_name ALTER COLUMN column_name SET NOT NULL;

ALTER TABLE table_name ALTER COLUMN column_name DROP NOT NULL;
```

DROP TABLE

- DROP TABLE statement is used to remove a table definition and all associated data and constraints for that table.
- Delete a table from the database using the DROP TABLE command (suppose we want to delete the Account table):

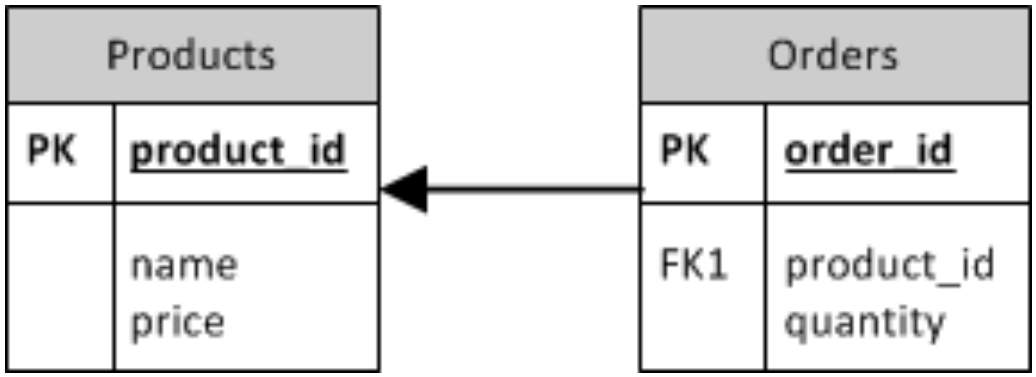
```
DROP TABLE account;
```

Note: once we drop a table, it deletes all data in the table and removes the table from the database.

To empty a table of rows without destroying the table, use DELETE statement.

DROP TABLE with CASCADE

Tables: Products, Orders (references Products)



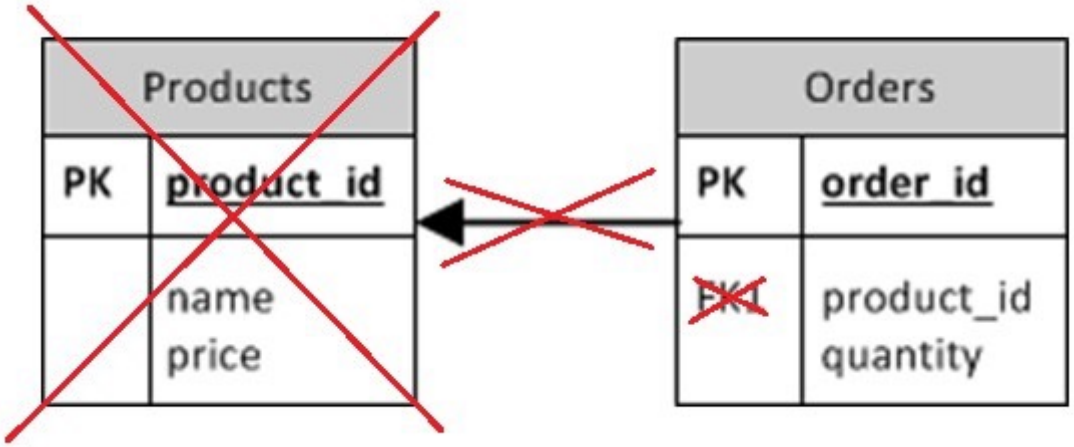
DROP TABLE products;

NOTICE: constraint orders_product_id_fkey on table orders depends on table products
ERROR: cannot drop table products because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.

DROP TABLE with CASCADE

DROP TABLE products CASCADE;

- In this case the command doesn't delete the Orders table, only Foreign Key constraint.



- RESTRICT keyword instead of CASCADE determines the default behavior: prevents removal of objects from which other objects depend on.

DROP TABLE full syntax

- Full syntax of DROP TABLE command:

DROP TABLE [IF EXISTS] table_name [, …]
[CASCADE | RESTRICT]

- IF EXISTS Do not throw an error if the table does not exist. A notice is issued in this case.

Books

- **Connolly, Thomas M. Database Systems**: A Practical Approach to Design, Implementation, and Management / Thomas M. Connolly, Carolyn E. Begg.- United States of America: Pearson Education

- **Garcia-Molina, H. Database system**: The Complete Book / Hector Garcia-Molina.- United States of America: Pearson Prentice Hall

- **Sharma, N. Database Fundamentals**: A book for the community by the community / Neeraj Sharma, Liviu Perniu.- Canada

- www.postgresql.org/docs/manuals/