DATABASES

Lecture 3. DDL. Creating, modifying, deleting objects
DML. Inserting data

WHY POSTGRESQL?

PostgreSQL is an advanced object-relational database management system that supports an extended subset of the SQL standard, including transactions, foreign keys, subqueries, triggers, user-defined types and functions.


Instagram

Netflix


Instacart


Spotify

Uber Technologies


reddit

RELATIONAL DATABASES

- Data is stored in tables
- Tables are related to each other

What is SQL?

Structured query language (SQL) is a standard language for database creation and manipulation. SQL is the standard language for database management.

All the RDBMS systems use SQL as their standard database language. SQL programming language uses various commands for different operations.

History

- 1970 - Dr. E. F. "Ted" of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974 - Structured Query Language appeared.
- 1978 - IBM worked to develop Codd's ideas and released a product named System/R.
- 1986 - IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

SQL features

- **Case insensitive - it doesn't matter whether capital or lowercase letters are used to write code**

```
SELECT name FROM employees;
seleCT name From employees;
```

- **You can use as many spaces as you like**

```
SELECT name,     age
        FROM employees;
```

- **Your code can be moved to new lines (and even necessary, for more convenient code readability)**

SQL features

- Comments:
  Single-line comment: Starts with --
  Multi-line comment: Enclosed between /* ... */

```
SELECT * FROM employees; -- This is a single-line comment

/* This is a

      multi-line comment */

SELECT * FROM departments;
```

- NULL handling for managing missing data.

```
SELECT name FROM employees WHERE salary IS NULL;
```

SQL language statements

DDL – Data Definition Language (CREATE, ALTER, DROP, TRUNCATE, USE). Is a standard for commands that define the different structures in a database.

DCL – Data Control Language (GRANT, REVOKE). Used for controlling access to the data in the database; user & permission management.

TCL – Transaction Control Language (BEGIN TRANSACTION, COMMIT, ROLLBACK)

DML – Data Manipulation Language (SELECT, INSERT, UPDATE, DELETE, MERGE). Used to query the database for information by getting information that is already stored there.

DDL - Data Definition Language

• CREATE is used to create new database objects like tables, indexes, views, or schemas.

• ALTER is used to modify the structure of an existing database object, such as adding, modifying, or deleting columns in a table.

•DROP is used to delete database objects like tables, indexes, or databases completely. Once dropped, the data and structure are lost, and recovery may not be possible.

• TRUNCATE removes all rows from a table but keeps the table structure intact. It is faster than DELETE as it does not log individual row deletions and typically resets any auto-increment counters.

Database creation

To create a database, you must be a superuser or have the special CREATEDB privilege.

CREATE DATABASE database_name;

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ LC_COLLATE [=] lc_collate ]
           [ LC_CTYPE [=] lc_ctype ]
           [ TABLESPACE [=] tablespace_name ]
           [ ALLOW_CONNECTIONS [=] allowconn ]
           [ CONNECTION LIMIT [=] connlimit ]
           [ IS_TEMPLATE [=] istemplate ] ]
```

Database creation parameters

- name - The name of a database to create.
- user_name - The role name of the user who will own the new database, or DEFAULT to use the default
- template - The name of the template from which to create the new database, or DEFAULT to use the default

Database creation parameters

- encoding - Character set encoding to use in the new database
- lc_collate - Collation order to use in the new database
- lc_ctype - Character classification to use in the new database

Database creation parameters

- tablespace_name - The name of the tablespace that will be associated with the new database
- allow_connections - If false then no one can connect to this database
- conn_limit - How many concurrent connections can be made to this database. -1 (the default) means no limit.
- istemplate - If true, then this database can be cloned by any user with CREATEDB privileges; if false (the default), then only superusers or the owner of the database can clone it.

Database creation examples

NOTES:

•**Default Values**: If any of these parameters are not explicitly defined during database creation, PostgreSQL will use system defaults. For instance, template1 is used by default, and the encoding will be the default encoding of the template database.

•**Tablespaces**: The choice of a tablespace allows storing the database on specific disks or storage volumes for performance, organizational, or management reasons.

By leveraging these parameters, a database can be created with fine-grained control over its properties, performance, and behavior.

```
CREATE DATABASE mydb OWNER myuser
TEMPLATE template1
ENCODING 'UTF8’
LC_COLLATE 'en_US.UTF-8’
LC_CTYPE 'en_US.UTF-8’
TABLESPACE mytablespace
ALLOW_CONNECTIONS true
CONNECTION LIMIT 50
IS_TEMPLATE false;
```

```
CREATE DATABASE university
 WITH OWNER = postgres
 ENCODING = 'UTF8'
 CONNECTION LIMIT = -1;
```

Table creation

- CREATE TABLE — define a new table
- CREATE TABLE will create a new, initially empty table in the current database.
- The table will be owned by the user issuing the command.

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ TABLESPACE tablespace_name ]
```

Table creation parameters

- **IF NOT EXISTS** - Do not throw an error if a relation with the same name already exists.
- **table_name** - The name (optionally schema-qualified) of the table to be created.
- **column_name** - The name of a column to be created in the new table.
- **data_type** - The data type of the column. This can include array specifiers.
- **COLLATE collation** - The COLLATE clause assigns a collation to the column (which must be of a collatable data type).
- **INHERITS (parent_table [, ...])** - The optional INHERITS clause specifies a list of tables from which the new table automatically inherits all columns. Parent tables can be plain tables or foreign tables.
- **LIKE source_table [like_option ...]** - The LIKE clause specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

Table creation parameters

- **CONSTRAINT constraint_name** - An optional name for a column or table constraint.
- **TABLESPACE tablespace_name** -The tablespace_name is the name of the tablespace in which the new table is to be created.

CONSTRAINTS

- **NOT NULL:** a constraint that indicates that this column cannot be empty (NULL), and must have a value
- **PRIMARY KEY:** used to uniquely identify a row in the table. It is good practice to specify in each table the column that contains the primary key.
- **FOREIGN KEY:** Used to ensure referential integrity of the data.
- **UNIQUE:** Ensures that all values in a column are different.
- **CHECK:** Makes sure that all values in a column satisfy certain criteria.
- **DEFAULT:** you can set a default column value when no other value has been specified

Examples

```
CREATE TABLE IF NOT EXISTS students (
   student_id     SERIAL PRIMARY KEY,
   first_name     VARCHAR(50)  NOT NULL,
   last_name      VARCHAR(50)  NOT NULL,
   birth_date     DATE,
   enrollment_year INT DEFAULT 2025);
```

```
CREATE TABLE enrollments (
 student_id INT NOT NULL
   REFERENCES students(student_id) ON DELETE
CASCADE,
 course_id  INT NOT NULL
   REFERENCES courses(course_id)  ON DELETE
CASCADE,
 grade CHAR(2) CHECK (grade IN ('A','B','C','D','F')),
 PRIMARY KEY (student_id, course_id))  --composite
key);
```

```
CREATE TABLE IF NOT EXISTS
archived_students (LIKE students
INCLUDING ALL)
TABLESPACE fast_disk;
```

```
CREATE TABLE IF NOT EXISTS courses (
   course_id   SERIAL PRIMARY KEY,
   course_name VARCHAR(100) NOT
NULL,
   credits    INT        NOT NULL CHECK
(credits > 0)
);
```

Modifying tables (Alter)

ALTER TABLE statement is used to change the definition or structure of an existing table.

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
```

Following actions can be performed

- Columns – Add, Delete (Drop), Modify or Rename
- Constraints – Add, Drop
- Index – Add, Drop

ALTER TABLE Parameters

- **ADD COLUMN** - adds a new column to the table, using the same syntax as CREATE TABLE
- **DROP COLUMN** - drops a column from a table. Indexes and table constraints involving the column will be automatically dropped as well.
- **SET DATA TYPE/TYPE** - changes the type of a column of a table
- **SET/DROP DEFAULT** - These forms set or remove the default value for a column.
- **SET/DROP NOT NULL** - These forms change whether a column is marked to allow null values or to reject null values.
- **ADD table_constraint** - This form adds a new constraint to a table using the same syntax as CREATE TABLE.

ALTER

The basic syntax of an ALTER TABLE command to add/drop a Column in an existing table is as follows.

Syntax

```
ALTER TABLE table_name
ADD COLUMN column_name Data Type;
```

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

```
ALTER TABLE students ADD COLUMN gpa NUMERIC(3,2);
```

ALTER

The basic syntax of an ALTER TABLE command to Modify/Rename a Column in an existing table is as follows.

Syntax

```
ALTER TABLE table_name
ALTER COLUMN column_name TYPE New Data
Type;
```

```
ALTER TABLE table_name
RENAME COLUMN column_1 TO column_2
```

```
ALTER TABLE students RENAME COLUMN gpa TO
grade_point_avg;
```

ALTER

The basic syntax of an ALTER TABLE command to add/drop a Constraint on an existing table is as follows.

Syntax

1. ALTER TABLE table_name ALTER COLUMN column_name SET NOT NULL;
2. ALTER TABLE table_name ALTER COLUMN column_name DROP NOT NULL;
3. ALTER TABLE table_name ADD CONSTRAINT column_name CHECK (column_name >= 100);
4. ALTER TABLE table_name ADD PRIMARY KEY (column_name);

ALTER TABLE courses ADD CONSTRAINT uq_course_name UNIQUE (course_name);

General Notes on ALTER

- Permissions
- Concurrency
- Complex Operations

DROP

DROP TABLE — removes tables from the database.

Only the table owner, the schema owner, and superuser can drop a table.

```
DROP TABLE [ IF EXISTS ] name [, ...]
    [ CASCADE | RESTRICT ]
```

Parameters

- **CASCADE** - Automatically drop objects that depend on the table (such as views), and in turn all objects that depend on those objects
- **RESTRICT** - Refuse to drop the table if any objects depend on it. This is the default.

```
DROP TABLE films, distributors CASCADE;
```

TRUNCATE

TRUNCATE is a **DDL command** used to quickly remove **all rows** from a table while keeping the table structure intact.

It is much faster than DELETE because it does not scan each row individually and does not generate a large number of row-level locks.

TRUNCATE TABLE table_name;

TRUNCATE parameters

- **RESTART IDENTITY** – Resets identity/serial columns to their starting values (default).
- **CONTINUE IDENTITY** – Keeps the current sequence values.
- **CASCADE** – Automatically truncates tables that have foreign-key references to the target table.
- **RESTRICT** – Refuses to truncate if there are foreign-key references (default behavior).

```
TRUNCATE TABLE enrollments RESTART IDENTITY;
```

Best Practices and Precautions

- Always Backup Before Dropping
- Use Transactions
- Use IF EXISTS
- Prefer RESTRICT Over CASCADE
- Check Dependencies Before Dropping

DATA TYPES

- PostgreSQL has a rich set of native data types available to users.
- Users can add new types to PostgreSQL using the CREATE TYPE command.

DATA TYPES

- **Numeric** - This type of data stores numerical values. Following Data types fall in this category: Integer, Float, Real, Numeric, or Decimal.
- **Character String** - This type of data stores character values. The two common types are CHAR(n) and VARCHAR(n).
- **Date/Datetime** - This type of data allows us to store date or datetime in a database table
- **Binary Data types**
- **Boolean types**
- **Arrays**

DATA TYPES

Numeric - This type of data stores numerical values.

Data Type	Description	Syntax	Storage
Small Integer	Integers with a small range	smallint	2 bytes
Integer	Most common choice for integers. Integers with medium range.	int	4 bytes
Big Integer	Integer with a large range.	bigint	8 bytes
Serial	Integer with an auto-incrementing value	serial	4 bytes
Float integer	Floating-point numbers with n precision	float(n)	8 bytes
Float integer	Floating-point number with 4-bytes.	float8 Or real	4 bytes
Real Number	Integer number with p digits and s digit after the decimal point.	numeric Or numeric(p,s)	

DATA TYPES

Character String - PostgreSQL supports several general-purpose character types.

Data Type	Syntax	Description
Character string	char(n)	Used for strings with padded spaces and a fixed length of characters.
Varchar string	varchar(n)	Variable-length character string with no padding, and a smaller data length than n.
Text string	text	Variable-length character string for storing data of unlimited length.

DATA TYPES

Date/Datetime - This type of data allows us to store date objects in a database table.

Data Type	Description	Syntax
Date	Store dates only.	date
Time	Store time of day values.	time
Timestamp	Store date and time values.	timestamp
TIMESTAMPTZ	Store date and time with time zone	timestamptz
INTERVAL	Store periods of time.	interval

DATA TYPES

Boolean data types can take on one of three possible values: true, false, or null.

Data Type	Description	Storage Size
Boolean	Indicate state of true/false.	1-Byte

DML - Data Manipulation Language

- **SELECT** is used to query data from one or more tables
- **INSERT INTO** is used to add new rows to a table.
- **UPDATE** is used to modify existing rows in a table
- **DELETE** removes rows from a table based on a condition.

INSERT INTO

The INSERT INTO statement is used to add new records into a database table

Syntax

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...)
```

table_name - name of the table
column1, etc - column names
value1, etc - are the values you want to insert into the corresponding columns. The order of the values must match the order of the columns (i.e. value1 goes into column1, etc).

INSERT INTO

The INSERT INTO statement is used to add new records into a database table

Example

Single row

```
INSERT INTO student (student_ID, first_name, last_name, date_of_birth,
gpa, major_ID, city)
VALUES
    (1, 'Amina', 'Abdulova', '14/05/2003', 3.8, 1, 'Almaty');
```

INSERT INTO

The INSERT INTO statement is used to add new records into a database table

Example

Multiple rows

```
INSERT INTO student (student_ID, first_name, last_name, date_of_birth, gpa, major_ID, city)
VALUES
    (2, 'Aidar', 'Kazimov', '23/08/2004', 3.5, 2, 'Astana'),
    (3, 'Gulnara', 'Ismailova', '30/11/2005', 3.9, 3, 'Almaty'),
    (4, 'Bakhytzhan', 'Nurpeisov', '17/02/2002', 3.2, 4, 'Shymkent');
```

SELECT

The SELECT statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

```
SELECT column_name1, column_name2 FROM table_name;

SELECT * FROM table_name
```

SELECT

Examples

Select one column

```
SELECT first_name FROM student;
```

Select multiple columns

```
SELECT first_name, last_name FROM student;
```

THANK YOU!