

Using basic SQL

Outline

1. The SQL language
2. Create objects
3. Handling data
4. Querying data

1. The SQL language

- One of the ways to interact with PostgreSQL is to use the standard SQL query language
- Can access your PostgreSQL system from the psql program, a fancy Java or .NET application → knowing how to use SQL is an important skill to have
- The better your SQL skills, the better your application will perform

SQL History

- The Structured Query Language (**SQL**) has been around since the early 1970s as a language for interacting with relational database systems
- The first commercial SQL product, called Structured English Query Language (**SEQUEL**), was released by IBM in 1974
- In 1986 the American National Standards Institute (ANSI) formulated the first attempt to standardize SQL named **ANSI SQL89**
- additional updates have been made to the ANSI SQL standard, resulting in **SQL92** (SQL 2) and **SQL99**
- **PostgreSQL** conforms to the **ANSI SQL 92/99** standards

PostgreSQL SQL format

- A SQL command consists of ***tokens***, separated by white space, and **terminated by a semicolon**
- The command tokens identify actions, and data used in the command
 - Keywords
 - Identifiers
 - Literals

PostgreSQL SQL Keywords

SQL Keyword	Description
ALTER	Change (alter) the characteristics of an object.
CLOSE	Remove (close) an active cursor in a transaction or session.
COMMIT	Commit a transaction to the database.
CREATE	Create objects.
DELETE	Remove database data from a table.
DROP	Remove an object from the database.
END	Define the end of a transaction.
GRANT	Set object privileges for users.
INSERT	Add data to a table.
RELEASE	Delete a savepoint defined in a transaction.
REVOKE	Remove object privileges from users.
ROLLBACK	Undo a transaction.
SAVEPOINT	Define a point in a transaction where commands can be rolled back. This allows transactions within transactions.
SELECT	Query database table data.
START TRANSACTION	Start a set of database commands as a block.
UPDATE	Alter database data stored in a table.

SQL Literals

- **String** data types (characters, variable-length characters, time strings, and date strings) must be **enclosed in single quotes**
- If there is single quote within a string → preceding it with a **backslash**: 'O\'Leary'

SQL Identifiers

- SQL command *identifiers* define **database objects** used in the command (database name, schema name, or table name)
- Identifiers are **case sensitive** in PostgreSQL
 - Customer, CUSTOMER, CusTomer → customer
 - “”: store."Customer" , "Store"."Customer"
- Identifier names **vs. keywords**
 - SELECT * from SELECT; **:NO**
→ SELECT * from "select"; **: YES**
 - Using **keywords as table names** is an **extremely bad habit** to acquire
- Try to avoid using keywords as identifiers at all cost

2. Create objects

Creating a Database

- **CREATE DATABASE** *name* [**[WITH]**
[**OWNER** *owner*]
[**TEMPLATE** *template*]
[**ENCODING** *encoding*]
[**TABLESPACE** *tablespace*]
[**CONNECTIONLIMIT** *connlimit*]]
- The default template used to create the database will be template1
- \l: list of databases
- **DROP DATABASE** [**IF EXISTS**] *name*
 - recover from the DROP DATABASE command is to restore the database from the last backup
- You must be a **superuser** or have the special **CREATEDB** **privilege**

```

C:\Program Files\PostgreSQL\8.2\bin>psql postgres postgres
Password for user postgres:
postgres=# create database test2;
CREATE DATABASE
postgres=# create database test2;
ERROR: database "test2" already exists
postgres=# \c test2
You are now connected to database "test2".
test2=#
test2=# \l

```

List of databases

Name	Owner	Encoding
postgres	postgres	SQL_ASCII
template0	postgres	SQL_ASCII
template1	postgres	SQL_ASCII
test	postgres	UTF8
test2	postgres	SQL_ASCII

(5 rows)

Creating a Schema

- **CREATE SCHEMA** [IF NOT EXISTS] [*schemaname*]
[AUTHORIZATION *username* [*schema elements*]]

Example: **create schema store authorization fred;**

- \dn: List of schemas
- the invoking user must have the **CREATE privilege** for the current database
- **DROP SCHEMA** [IF EXISTS] *schemaname* [CASCADE | RESTRICT]
 - Default: RESTRICT → remove only if empty
 - NOT empty → CASCADE: Automatically drop objects (tables, functions, etc.) that are contained in the schema

<file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-createschema.html>

```
test2=# create schema store authorization fred;
```

```
CREATE SCHEMA
```

```
test2=# \dn
```

```
      List of schemas
```

Name	Owner
information_schema	postgres
pg_catalog	postgres
pg_toast	postgres
public	postgres
store	fred

```
(5 rows)
```

```
test2=#
```

Creating a Table

- **CREATE TABLE** command can be **extremely complex**
 - primary key, foreign keys, table constraints
- Instead of trying to include all of the information required to create a table
 - create a **base definition of a table** using the **CREATE TABLE** command
 - add additional elements using **ALTER TABLE** commands

Defining the Base Table

- `CREATE TABLE tablename (column1 datatype, column2 datatype, ...);`
- Database administrators often split the statement into several command-line entries

```
create table store."Customer" (  
    "CustomerID" varchar,  
    "LastName" varchar,  
    "FirstName" varchar,  
    "Address" varchar,  
    "City" varchar,  
    "State" char, "Zip" char(5), "Phone" char(8));
```

PRIMARY KEY

```
postgres=> create table Employee (  
postgres(> EmployeeID int4 primary key,  
postgres(> Lastname varchar,  
postgres(> Firstname varchar,  
postgres(> Department char(5) not null,  
postgres(> StartDate date default now(),  
postgres(> salary money);  
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "employee_pkey"  
for table "Employee"  
CREATE TABLE  
postgres=>
```


Adding Additional Table Elements

- Format: **ALTER TABLE** *tablename action*

- Test:

```
\d store."Customer"
```

```
alter table store."Customer" add primary key  
("CustomerID");
```

```
alter table store."Customer" alter column "Phone" set not  
null;
```

```
\d store."Customer"
```

<file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-altertable.html>

Adding Additional Table Elements

ALTER TABLE ... ADD CONSTRAINT CHECK (condition);

ALTER TABLE ... ADD CONSTRAINT

FOREIGN KEY (*column_name* [, ...]) REFERENCES *reftable* [(*refcolumn* [, ...])]

[MATCH FULL | MATCH SIMPLE]

[ON DELETE *action*] [ON UPDATE *action*]

[DEFERRABLE | NOT DEFERRABLE]

[INITIALLY DEFERRED | INITIALLY IMMEDIATE];

<file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-altertable.html>

Adding Additional Table Elements

...

[ON DELETE *action*] [ON UPDATE *action*]

...

action can be: RESTRICT | NO ACTION | CASCADE |
SET NULL | SET DEFAULT

- RESTRICT prevents deletion of a referenced row
- CASCADE specifies that when a referenced row is deleted (or updated), row(s) referencing it should be automatically deleted (or updated) as well.
- NO ACTION means that if any referencing rows still exist when the constraint is checked, an error is raised. **(default)**

Example

create table "Order"

("OrderID" varchar, "CustomerID" varchar,
"ProductID" varchar, "Quantity" integer);

ALTER TABLE "Order" ADD CONSTRAINT order_pk PRIMARY
KEY ("OrderID");

ALTER TABLE "Order" ADD CONSTRAINT order_fk2 FOREIGN
KEY ("CustomerID") REFERENCES "Customer"("CustomerID")
ON UPDATE ON ON DELETE ;

<file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/sql-altertable.html>

ALTER TABLE Actions

ALTER Action	Description
ADD COLUMN <i>columnname</i>	Add a new column to the table.
DROP COLUMN <i>columnname</i>	Remove an existing column from the table.
ALTER COLUMN <i>columnname action</i>	Change the elements of an existing column. Can be used to change data type, add keys, or set constraints.
SET DEFAULT <i>value</i>	Set a default value for an existing column.
DROP DEFAULT	Remove a defined default value of an existing column.
SET NOT NULL	Define the NOT NULL constraint on an existing column.
DROP NOT NULL	Remove a NOT NULL constraint from an existing column.
SET STATISTICS	Enable statistic gathering used by the ANALYZE command.
SET STORAGE	Define the storage method used to store the column data.
ADD <i>constraint</i>	Add a new constraint to the table.
DROP <i>constraint</i>	Remove a constraint from the table.
DISABLE TRIGGER	Disable (but not remove) a trigger defined for the table.
ENABLE TRIGGER	Define a new trigger for the table.
OWNER <i>loginrole</i>	Set the table owner.
SET TABLESPACE <i>newspace</i>	Change the tablespace where the table is stored to <i>newspace</i> .
SET SCHEMA <i>newschema</i>	Change the schema location of the table to <i>newschema</i> .
RENAME COLUMN <i>oldname</i> TO <i>newname</i>	Change the name of table column <i>oldname</i> to <i>newname</i> .
RENAME TO <i>newname</i>	Change the name of the table to <i>newname</i> .

3. Handling data

<file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/ddl.html>

Inserting Data

- **INSERT INTO** *table* [(*columnlist*)] **VALUES** (*valuelist*)
 - insert into store."Customer" values ('BLU001', 'Blum', 'Rich', '123 Main St.', 'Chicago', 'IL', '60633', '555-1234');
- If you do not want to enter all of the values into a record, you can use the optional *columnlist* parameter
 - insert into store."Customer" ("CustomerID", "LastName", "Phone") values ('BLU002', 'Blum', '555-4321');
- Constraints
 - NOT NULL
 - DEFAULT VALUE → use DEFAULT or not list this column in the *columnlist* parameter

Modifying Data

- *UPDATE table SET column = value [WHERE condition]*
 - `update store."Customer" set "FirstName" = 'Barbara';`
- The WHERE clause allows you to restrict the records that the UPDATE command applies to
 - `update store."Customer" set "FirstName" = 'Rich' WHERE "CustomerID" = 'BLU001';`

Deleting Data

- DELETE FROM *table* [WHERE *condition*]
- delete from store."Customer" where "CustomerID" = 'BLU001';

4. Querying data

<file:///C:/Program%20Files/PostgreSQL/9.4/doc/postgresql/html/queries.html>

The Basic Query Format

- `SELECT columnlist FROM table`
- The output of the SELECT command is called a *result set*. By default, the records are not displayed in any particular order.
- Specify the order of the displayed records, you must use the ORDER BY clause

```
select "CustomerID", "LastName", "FirstName"  
from store."Customer"  
order by "FirstName";
```

Filtering Output Data

- The WHERE clause is used to determine what records satisfy the condition of the query.

```
select "CustomerID", "LastName", "FirstName"  
from store."Customer "  
where "City" = 'Gary';
```

Querying from Multiple Tables

- `select "Order"."OrderID",
"Customer"."CustomerID",
"Customer"."LastName", "Customer"."FirstName",
"Customer"."Address"
from store."Order", store."Customer"
where "Order"."OrderID" = 'ORD001' and
"Order"."CustomerID" =
"Customer"."CustomerID";`

Using Joins

- **SELECT** *columnlist*
FROM *table1 jointype JOIN table2 ON condition*
- Join types:
 - **INNER JOIN** Only display records found in both tables
 - **LEFT JOIN** Display all records in *table1* and the matching records in *table2* (outer joins)
 - **RIGHT JOIN** Display all records in *table2* and the matching records in *table1* (outer joins)
 - **CROSS JOIN** == t1 INNER JOIN t2 ON TRUE
- **NATURAL** keyword → join using the common column name

Using Joins

- **select** "Order"."OrderID", "Customer"."CustomerID",
"Customer"."LastName", "Customer"."FirstName",
"Customer"."Address"
from store."Order" **natural inner join** store."Customer";
- **select** "Order"."OrderID", "Customer"."CustomerID",
"Customer"."LastName", "Customer"."FirstName",
"Customer"."Address"
from store."Order" **natural right join** store."Customer"

Using Aliases

- **SELECT** *columnlist* **FROM** *table* **AS** *alias*
- **select** a."OrderID", b."CustomerID", b."LastName",
b."FirstName", b."Address"
from store."Order" **as** a, store."Customer" **as** b
where a."OrderID" = 'ORD001' and a."CustomerID" =
b."CustomerID";

