CMSC 206: Database Management Systems

SQL

Thomas Laurent

10/10/2020

University of the Philippines Open University

Table of contents

Introduction

Introduction

SQL stands for Structured Query Language. It is the standard language to interact with RDBMSes. Still, each DBMS might extend the language in particular ways, or might not implement all features. Nevertheless, adapting SQL queries for each DBMS should require minimal effort, especially when using the core functions of the language.

While up until now we saw how to model our data to design our database, we will now see how to implement it with SQL. This means our language will change a bit. We will talk about *tables*, not relations, *columns*, not attributes, and *rows*, not tuples.

Introduction

SQL lets us define our DB, populate it, query it and administer it. It is thus composed of different sub languages that we will discover today.

We will stick to the more general functions of SQL here. Each specific DBMS has documentation for the more advanced functions.

DDL: Data Definition Language

DDL - Introduction

The DDL lets us:

- Create a database
- Define the infrastructure (schema) of the database
- Modify this infrastructure

The DDL deals with the metadata that the DBMS stores about the DB.

DDL - Main commands

The main commands in the DDL are:

- CREATE DATABASE: creates a database
- DROP DATABASE: deletes a database
- CREATE TABLE: creates a table
- CREATE VIEW: creates a view (a virtual table) from a query written in the DQL (a SELECT statement)
- DROP TABLE/VIEW: deletes a table or view
- ALTER TABLE/VIEW: modifies a table/view

DDL - CREATE DATABASE and DROP DATABASE

Syntax

```
CREATE DATABASE dbNAme
```

Creates a database of name *dbNAme*.

Every DBMS offers different options for this command, but they are not standardised.

Syntax

```
DROP DATABASE dbNAme
```

Deletes the database with name *dbNAme*.

DDL - CREATE TABLE

Syntax

CREATE TABLE tableName (cName cType [[CONSTRAINT constraint_name] cConstraint] , ..., [[CONSTRAINT constraint_name] tableConstraint])
CREATE TABLE tableName as SQLQuery

Creates a new table in the current DB with the specified schema.

Example

CREATE TABLE student (student_id serial PRIMARY KEY, student_first_name varchar(20) NOT NULL, student_number varchar(20) NOT NULL, UNIQUE(student_number))

DDL - CREATE TABLE - data types

Each DBMS proposes its own data types, and refers to them using different keywords. The main data types and their representation in postgreSQL are:

type	PostgreSQL
integer	integer
auto incremented integer	serial
real	numeric
character	char
string (fixed length)	varchar(length)
string	text
boolean	bool
date	date

DDL - CREATE TABLE - column constraints

The main column constraints are:

NOT NULL this column can not take the value NULL

UNIQUE each row must have a unique value for this column

PRIMARY KEY this column is the primary key of the table. It is unique and not null

REFERENCES tableName[(columnName)] [action] creates a foreign key constraint between this column and a column in tableName, its primary key by default.
action lets us define the behaviour when the referenced value is updated or deleted. For example ON UPDATE CASCADE, would reflect the update of the referenced value in the current table

CHECK(condition) the condition must be true for every row in the table

DDL - CREATE TABLE - table constraints

The main table constraints are:

PRIMARY KEY(columnName+) creates a primary key formed of the specified column(s)

UNIQUE(columnName+) declares the specified column(s) as a super key

FOREIGN KEY (cName+) REFERENCES tableName[(cName+)] [action] creates a foreign key constraint between column(s) of the table and column(s) in a referenced table
action lets us define the behaviour when the referenced value is updated or deleted. For example ON DELETE SET NULL, would set the current table's value to NULL if the referenced tuple is deleted. An error would be thrown otherwise.

CHECK(condition) the condition must be true for every row in the table

DDL - DROP TABLE

Syntax

```
DROP TABLE tableName [CASCADE]
```

Drops the specified table. If cascade is used, the deletion is propagated to tuples referencing the table.

Example

```
DROP TABLE student
```

DDL - ALTER TABLE

Syntax

ALTER TABLE tableName {RENAME TO newName | RENAME COLUMN cName TO newName | ADD COLUMN cName cType [[CONSTRAINT] cConstraint] | ALTER COLUMN cName alteration | DROP COLUMN cName [CASCADE] }

Alters the schema of the table. Can rename the table, a columns, add or remove a column, change a columns data type or constraints, etc.

Examples

ALTER TABLE student RENAME TO etudiant
ALTER TABLE student RENAME COLUMN student_name TO sName
ALTER TABLE student ALTER COLUMN student_name TYPE text

DDL - CREATE VIEW

Syntax

```
CREATE VIEW AS SQLQuery
```

Creates a view from the specified SQL query. A view acts as a virtual table without a primary key that reflects the results of the query. A view can be used in queries as any other table. The view represents the query, not its result at a certain time.

A view can let us expose some data to users without them have to know how the data was constructed (without having to know the query that resulted in this view), and without them needing query permissions on the original tables.

DML: Data Manipulation Language

DML - Introduction

The DML lets us:

- Insert data in our database
- Modify data in our database
- Delete data from our database

DML - Main commands

The main commands in the DML are:

- INSERT INTO: inserts data (rows) in a table
- UPDATE: updates data (rows) in a table
- DELETE FROM: deletes data (rows) from a table

DML - INSERT INTO

Syntax

INSERT INTO tableName($attr_1$, $attr_2$, ..., $attrn_n$) values (v_{1_1}, v_{1_2}, ..., v_{1_n}), (v_{2_1}, v_{2_2}, ..., v_{2_n}), ...
INSERT INTO tableName values (v_{1_1}, v_{1_2}, ..., v_{1_n}), (v_{2_1}, v_{2_2}, ..., v_{2_n}), ...
INSERT INTO tableName [($attr_1$, $attr_2$, ..., $attrn_n$)] SQLQuery

Insert the specified values in the table. If the attributes are specified then the values are inserted in each column in order (v_{1_1} for $attr_1$, etc), otherwise the order of the columns in the schema (the CREATE TABLE statement). If the attributes are not specified and the schema changes (UPDATE TABLE) then it will lead to errors or problems!

We can also insert values that result from an SQL query written in the DQL.

The insert will fail if the constraints are not respected. If we insert a NULL value in a NON NULL column, or if we insert a value that already exists in a UNIQUE column, for example. The whole insert will fail even if only one row fails, rember the A in ACID.

DML - INSERT INTO - Examples

Examples

INSERT INTO student (student_fist_name, student_number) values ('Thomas', 1), ('Ria', 42)

INSERT INTO student values (1, 'Thomas', 1), (2, 'Ria', 42)

INSERT INTO student (student_fist_name, student_number) values SELECT first_name, student_number FROM old_students

DML - UPDATE

Syntax

UPDATE tableName SET $attr_1 = val_1, attr_2 = val_2, \ldots$ [WHERE condition]

Sets the values of the specified attributes in the rows of table tableName. If a WHERE clause is used then only the rows meeting the condition are updated.

Examples

UPDATE student SET student_first_name='EveryoneHasTheSameName'

UPDATE student SET student_first_name='ThomasIsNowJohn' WHERE student_first_name='Thomas'

DML - DELETE FROM

Syntax

```
DELETE FROM tableName [WHERE condition]
```

Deletes rows from table tableName. If a WHERE is used then only rows meeting the condition are deleted.

Examples

```
DELETE FROM student

DELETE FROM student WHERE student_number = 42
```

DQL: Data Querying Language

DQL - Introduction

The DQL lets us retrieve data from our database and is based on the relational algebra we saw last week.

The only statement in the DQL is the SELECT statement. A SELECT statement will always return a (temporary) table. A main difference with the relational algebra is that the returned table can contain duplicate rows.

DQL - Simple SELECT

Syntax

SELECT col_1, col_2, col_3, ... FROM tableName

Returns a table containing only the specified columns of table tableName. Corresponds to the projection operation in relational algebra, except that duplicate rows can be returned if a superKey is not included in the selected columns.

The * operator selects all columns, i.e. just returns table tableName.

Examples

SELECT * FROM student

SELECT student_name FROM student

DQL - SELECT DISTINCT

Syntax

SELECT DISTINCT col_1, col_2, col_3, ... FROM tableName

The DISTINCT constraints does not allow duplicate rows in the resulting table, the behaviour is the same as the projection operator in relation algebra.

Examples

SELECT DISTINCT * FROM student

SELECT DISTINCT student_first_name FROM student

DQL - WHERE

Syntax

SELECT [DISTINCT] col_1, col_2, col_3, ... FROM tableName [WHERE conditions]

The WHERE clause let's us define conditions that the selected rows must meet. Conditions are expressed as comparisons between a column's value and a constant or another column's value. We can also check if a value is NULL or not with operators IS NULL and IS NOT NULL. Multiple conditions can be used with the operators AND and OR. We can also negate a condition using the operator NOT.

Examples

SELECT * FROM student WHERE student_first_name = 'Ria'

SELECT DISTINCT student_first_name FROM student WHERE student_number > 42

SELECT DISTINCT student_first_name FROM student WHERE student_number <> 42

DQL - LIKE

Syntax

SELECT [DISTINCT] col_1, col_2, col_3, ... FROM tableName [WHERE stringColumns LIKE value]

The LIKE operator lets us perform comparisons of strings more general than equality. In the value to be compared we can use operator % to represent any string and operator _ to represent one character

Examples

SELECT * FROM student WHERE student_first_name LIKE '_ia'

SELECT * FROM student WHERE student_first_name LIKE 'T%'

The first example will return students named Ria, Tia, etc, while the second one will return student name Tom, Thomas, Theodore, etc.

DQL - IN

Syntax

SELECT [DISTINCT] col_1, col_2, col_3, ... FROM tableName [WHERE cName IN valueList]

The IN operator lets us specify a list of values for a column. The condition is met if the row's value for this column is in the list. The list can be the result of another query.

Examples

SELECT * FROM student WHERE student_first_name IN ('Ria', 'Thomas')

SELECT * FROM student WHERE student_first_name IN (SELECT student_first_name FROM student WHERE student_number = 2)

The first example will return students named Ria or Thomas. The second example will return students with student number 2. Writing this query in this way is uselessly complicated though.

DQL -JOIN

Joins let us query data from multiple tables. Joins are done in the FROM clause. The comma "," operator performs a cartesian product of two tables, creating all combinations of rows. The JOIN ... ON operator perform a theta join following the condition specified in the ON clause.

A NATURAL JOIN operator is also available to join two tables that have common columns. LEFT and RIGHT OUTER JOIN operator are also available to perform outer joins.

If two tables have a column with the same name, there can be ambiguity in which column we are referring to, we must then prefix the column name with the table name to clear the ambiguity.

DQL - JOIN - Syntax

Syntax

SELECT [DISTINCT] col_1, col_2, col_3, ... FROM $tableName_1$, $tablename_2$, ... [WHERE condition]

SELECT [DISTINCT] col_1, col_2, col_3, ... FROM $tableName_1$ JOIN $tablename_2$ ON condition JOIN ... [WHERE condition]

SELECT [DISTINCT] col_1, col_2, col_3, ... FROM $tableName_1$ (LEFT | RIGHT) OUTER JOIN $tablename_2$ ON condition JOIN ... [WHERE condition]

SELECT [DISTINCT] col_1, col_2, col_3, ... FROM $tableName_1$ NATURAL JOIN $tablename_2$ JOIN ... [WHERE condition]

DQL - JOIN - Examples

Examples

```
SELECT * FROM professor, student

SELECT * FROM professor JOIN student ON professor.id = tutor_id

SELECT * FROM professor JOIN student ON professor.id = tutor_id
WHERE professor.first_name = 'Thomas'
```

The first example gives us all the professor - student combinations.

The second example will return a table containing all student - tutor pairs. Both professor and student tables have an id column so we must write professor.id in the ON clause.

The third example will return a table containing all student - tutor pairs where the tutor's name is Thomas. Both professor and student tables have a first_name column so we must write professor.first_name in the WHERE clause.

DQL - Complex queries and functions

The DQL offers many other functionalities, such as ordering of the results (ORDER BY clause), grouping results (GROUP BY clause) and aggregation (MIN, MAX, AVG functions) and use of aliases (AS clause). We do not cover these here, but be aware of these possibilities.

Several functions vary from DBMS to DBMS so always refer to the documentation of you DBMS when trying to implement something complex.

DCL: Data control language

DCL - Introduction

The DCL lets us define authority of users on the data: who owns what, who can access and or modify what. The available granularity of the control varies from DBMS to DBMS.

The main operations of the DCL are:

- GRANT: grant a permission to users
- DENY: prohibit an operation
- REVOKE: revoke a granted permission
- ALTER ... OWNER TO: change the owner of a database/table

I will not go into details into the DQL as it is more relevant to system administrators than to users/programmers. Again, each DBMS should have extensive documentation on available permission mechanisms and how to set them.

Practising with SQL

Practising SQL

There was a lot of information here and it might not all make sense to you now. You can think of these slides as a memo but the important thing will be to practise writing SQL queries and instructions.

A useful resource for this is sql bot, that you can find *here*. There are other such sites that give you some SQL exercises and an interactive environment to practice, do not hesitate to shop around.

PostgreSQL

When you submit SQL code to me, I will run it through postgreSQL, so please make sure you either stick to the SQL standard and/or try out your queries in postgreSQL.

PostgreSQL is a free DBMS, you can install in on your system or use a database-as-a-service system such as the one here.

If the command line is scary you can also connect to your database using pgAdmin, available here, which offers a graphical interface.