Database Structuring and Querying with SQL

Hiba ALQASIR

2021-2022

télécom saint-étienne

école d'ingénieurs / nouvelles technologies

SQL

- Structured Query Language
- A standard language designed for managing data held in a relational database management system (RDBMS).
- Very highlevel language for querying and manipulating data.
- Helps in defining the structure of the data, modifying the data and specifying the security constraints.
- Uses a combination of relational algebra and relational calculus constructs.

Syntax

- SQL keywords are case-insensitive,
  however, they are often written in uppercase.
  Same: SELECT, Select, select
  Same: City, city
- Values are case-sensitive.
  Different: 'Paris', 'paris'
- Some database systems require a semicolon at the end of each SQL statement.
- Each clause in a statement begin on a new line.

```
SELECT *
FROM Capitals
WHERE name='Paris' ;
```

télécom saint-étienne

Tables

- A relation or table is a multiset of tuples
  having the attributes specified by the schema.
- An attribute (or column) is a typed data entry
  present in each tuple in the relation.
- A tuple (or row) is a single entry in the table having the attributes.
  Also referred to sometimes as a **record** specified by the schema.

Data types

- Numbers: INT, SMALLINT, BIGINT, FLOAT
- Characters: CHAR(20), VARCHAR(50)
- Others: MONEY, DATETIME, ...

Table Schema

- The schema of a table is the table name, its attributes, and their types.
- A key is an attribute whose values are unique; we underline a key.

Building (id:float, name:string, address:string)

DDL and DML

SQL is a:

- Data Definition Language (DDL)
  Define relational schema.
  Create/alter/delete tables and their attributes.
- Data Manipulation Language (DML)
  Insert/delete/modify tuples in tables.
  Query one or more tables.

Data Manipulation

- SELECT, FROM, WHERE
- ORDER BY
- DISTINCT, AND, OR, NOT, LIKE
- JOINs

SELECT ... FROM

The operation of producing an output table that have a subset of the prior attributes.

```
SELECT <attributes>
FROM <relations>;
```

To select all the fields available in the table:

```
SELECT *
FROM <relations>;
```

SELECT ... FROM

```
SELECT id, surface, level
FROM Apartment;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

id	surface	level
102	200	2
103	50	5
104	75	3
200	150	0
201	250	1
202	250	2

DISTINCT

To eliminate duplicates:

```
SELECT DISTINCT <attributes>
FROM <relations>;
```

DISTINCT

```
SELECT idBuilding
FROM Apartment;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

idBuilding
1
1
1
2
2
2

DISTINCT

```
SELECT DISTINCT idBuilding
FROM Apartment;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

idBuilding
1
2

WHERE

The operation of filtering a relation's tuples on some condition.

```
SELECT <attributes>
FROM <relations>
WHERE <conditions>;
```

WHERE clause is not only used in SELECT statements,
it is also used in UPDATE, DELETE, ...

WHERE

```
SELECT *
FROM Apartment
WHERE level = 2;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

id	no	surface	level	idBuilding
102	51	200	2	1
202	3	250	2	2

AND, OR and NOT

To filter records based on more than one condition:

```
WHERE NOT condition;
WHERE condition1 AND condition2;
WHERE condition1 OR  condition2;
```

- AND operator displays a record if all the conditions separated by AND are TRUE.
- OR operator displays a record if any of the conditions separated by OR is TRUE.
- NOT operator displays a record if the condition(s) is NOT TRUE.

AND

```
SELECT *
FROM Apartment
WHERE idBuilding = 1 AND surface > 60;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

id	no	surface	level	idBuilding
102	51	200	2	1
104	43	75	3	1

OR

```
SELECT *
FROM Apartment
WHERE idBuilding = 1 OR surface > 60;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

NOT

```
SELECT *
FROM Apartment
WHERE NOT idBuilding = 1;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

id	no	surface	level	idBuilding
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

LIKE

For simple string pattern matching.

```
SELECT <attributes>
FROM <relations>
WHERE A LIKE P;
```

A LIKE P: pattern matching on strings
P may contain two special symbols:

- % = any sequence of characters
- _ = any single character

LIKE

```
SELECT *
FROM Building
WHERE address LIKE '%Grand%';
```

id	name	address
1	Koudalou	3 rue des Martyrs
2	Barabas	2 allée du Grand Turc

Building

id	name	address
2	Barabas	2 allée du Grand Turc

ORDER BY

Sorting the Results

```
SELECT <attributes>
FROM <relations>
ORDER BY <attributes>;
```

For descending order:

```
SELECT <attributes>
FROM <relations>
ORDER BY <attributes> DESC;
```

télécom saint-étienne

ORDER BY

```
SELECT *
FROM Apartment
ORDER BY surface;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

id	no	surface	level	idBuilding
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
102	51	200	2	1
201	2	250	1	2
202	3	250	2	2

ORDER BY

```
SELECT *
FROM Apartment
ORDER BY no DESC;
```

id	no	surface	level	idBuilding
102	51	200	2	1
103	52	50	5	1
104	43	75	3	1
200	1	150	0	2
201	2	250	1	2
202	3	250	2	2

Apartment

id	no	surface	level	idBuilding
103	52	50	5	1
102	51	200	2	1
104	43	75	3	1
202	3	250	2	2
201	2	250	1	2
200	1	150	0	2

Cartesian product

```
SELECT *
FROM <relation1, relation2, ...>;
```

Cartesian product

```
SELECT *
FROM R1, R2;
```

A	B
a	b
x	y

$\mathcal{R}_1$

C	D
c	d
u	v
x	y

$\mathcal{R}_2$

A	B	C	D
a	b	c	d
a	b	u	v
a	b	x	y
x	y	c	d
x	y	u	v
x	y	x	y

$\mathcal{R}_1 \times \mathcal{R}_2$

INNER JOIN

Returns records that have matching values in both tables.

```
SELECT <attributes>
FROM <relation1>
INNER JOIN <relation2>
ON  <condition>;
```

Example:

```
SELECT *
FROM Capitals
INNER JOIN Presidents
ON  Capitals.State = Presidents.State;
```

télécom saint-étienne

INNER JOIN

Capital	State
Paris	France
Madrid	Spain
Berlin	Germany

Capitals

State	President
France	Emmanuel Macron
Germany	Frank-Walter Steinmeier
Italy	Sergio Mattarella

Presidents

Capital	State	President
Paris	France	Emmanuel Macron
Berlin	Germany	Frank-Walter Steinmeier

Capitals $\bowtie$ Presidents

LEFT JOIN

Returns all records from the left table, and the matched records from the right table.

```
SELECT <attributes>
FROM <relation1>
LEFT JOIN <relation2>
ON  <condition>;
```

Example:

```
SELECT *
FROM Capitals
LEFT JOIN Presidents
ON  Capitals.State = Presidents.State;
```

télécom saint-étienne

LEFT JOIN

Capital	State
Paris	France
Madrid	Spain
Berlin	Germany

Capitals

State	President
France	Emmanuel Macron
Germany	Frank-Walter Steinmeier
Italy	Sergio Mattarella

Presidents

Capital	State	President
Paris	France	Emmanuel Macron
Madrid	Spain	NULL
Berlin	Germany	Frank-Walter Steinmeier

Capitals $\bowtie$ Presidents

RIGHT JOIN

Returns all records from the right table, and the matched records from the left table.

```
SELECT <attributes>
FROM <relation1>
RIGHT JOIN <relation2>
ON  <condition>;
```

Example:

```
SELECT *
FROM Capitals
RIGHT JOIN Presidents
ON  Capitals.State = Presidents.State;
```

RIGHT JOIN

Capital	State
Paris	France
Madrid	Spain
Berlin	Germany

Capitals

State	President
France	Emmanuel Macron
Germany	Frank-Walter Steinmeier
Italy	Sergio Mattarella

Presidents

Capital	State	President
Paris	France	Emmanuel Macron
Berlin	Germany	Frank-Walter Steinmeier
NULL	Italy	Sergio Mattarella

Capitals $\bowtie_{\subset}$ Presidents

FULL OUTER JOIN

Returns all records when there is a match in either left or right table.

```
SELECT <attributes>
FROM <relation1>
FULL OUTER JOIN <relation2>
ON  <condition>;
```

Example:

```
SELECT *
FROM Capitals
FULL OUTER JOIN Presidents
ON  Capitals.State = Presidents.State;
```

FULL OUTER JOIN

Capital	State
Paris	France
Madrid	Spain
Berlin	Germany

Capitals

State	President
France	Emmanuel Macron
Germany	Frank-Walter Steinmeier
Italy	Sergio Mattarella

Presidents

Capital	State	President
Paris	France	Emmanuel Macron
Madrid	Spain	NULL
Berlin	Germany	Frank-Walter Steinmeier
NULL	Italy	Sergio Mattarella

Capitals ⟗ Presidents

Data Definition

- Creating, deleting, altering a table (schema object).
- Inserting, deleting, updating values (schema instance).

Creating a table

```
CREATE TABLE R(
                    A1 D1  ,
                    A2 D2  ,
                    ...    ,
                    An Dn
);
```

- Each A_i represents an attribute in the schema of relation $\mathcal{R}$.
- Each D_i denotes the data type of values in the domain for the corresponding attribute A_i.

Creating a table

```
CREATE TABLE Apartment(
                id int,
                no int,
                surface float,
                level int,
                idBuilding int
);
```

id	no	surface	level	idBuilding

Deleting a table

To delete (drop) an existing table:

```
DROP TABLE R;
```

Example

```
DROP TABLE Apartment;
```

Altering a table

A typical SQL query for altering:

```
ALTER TABLE R <action> <description>;
```

- action: ADD, MODIFY, DROP, RENAME.
- description: the modification command associated with action.

Altering a table

A typical SQL query for altering a table by adding attributes:

```
ALTER TABLE R ADD Ai Di ;
```

Example:

```
ALTER TABLE Apartment ADD street varchar(255);
```

id	no	surface	level	idBuilding	street

Altering a table

A typical SQL query for altering a table by deleting attributes:

```
ALTER TABLE R DROP COLUMN Ai;
```

Example:

```
ALTER TABLE Apartment DROP COLUMN level;
```

id	no	surface	idBuilding	street

Inserting values

```
INSERT INTO R(A1, A2, ...)
VALUES (value1, value2, ...);
```

Or:

```
INSERT INTO R
VALUES (value1, value2, ... valuek);
```

- Each A_i represents an attribute in the schema of relation $\mathcal{R}$.
- Each $value_i$ denotes the value for the corresponding attribute A_i.

Inserting values

```
INSERT INTO Apartment(id,  no, surface, level, idBuilding)
VALUES ( 100, 1, 150, 14 , 1);

INSERT INTO Apartment
VALUES ( 101, 34, 150, 15 , 1);
```

id	no	surface	level	idBuilding

id	no	surface	level	idBuilding
100	1	150	14	1
101	34	50	15	1

Deleting values

```
DELETE FROM R
WHERE <condition>;
```

- WHERE clause specifies which record(s) should be deleted.

Deleting values

```
DELETE FROM Apartment
WHERE id = 101;
```

id	no	surface	level	idBuilding
100	1	150	14	1
101	34	50	15	1

id	no	surface	level	idBuilding
100	1	150	14	1

Updating values

```
UPDATE  R
SET A1 = value1 , A2 = value2 , ...
WHERE <condition>;
```

- Each A_i represents an attribute in the schema of relation $\mathcal{R}$.
- Each $value_i$ denotes the value for the corresponding attribute A_i.
- WHERE clause specifies which record(s) should be updated.

Updating values

```
UPDATE  Apartment
SET idBuilding = 77, no = 10
WHERE id = 100;
```

id	no	surface	level	idBuilding
100	1	150	14	1

id	no	surface	level	idBuilding
100	10	150	14	77

Create a table

SQLizer
Easily convert files into SQL databases
https://sqlizer.io

télécom saint-étienne

RAT
Relational Algebra Translator
http://www.slinfo.una.ac.cr/rat/rat.html