Database Management System (DBMS)

Lecture-24

Dharmendra Kumar

October 12, 2020

Basic Structure

The basic structure of an SQL expression consists of three clauses: select, from, and where.

- The **select** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
- The **from** clause corresponds to the Cartesian-product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
- The **where** clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause.

SQL

A typical SQL query has the form

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

Each A_i represents an attribute, and each r_i a relation. P is a predicate. The query is equivalent to the relational-algebra expression

$$\Pi_{A_1,A_2,...,A_n}(\sigma_P(r_1 \times r_2 \times ..... \times r_m))$$

Schema Definition in SQL

We define an SQL relation by using the create table command:
create table $r(A_1\ D_1, A_2\ D_2, ..., A_n\ D_n, < integrity-constraint_1 >, ..., < integrity-constraint_k >)$
where r is the name of the relation, each A_i is the name of an attribute in the schema of relation r, and D_i is the domain type of values in the domain of attribute A_i. The allowed integrity constraints include

- **primary key** $(A_{j_1}, A_{j_2}, ..., A_{j_m})$**:** The primary key specification says that attributes $A_{j_1}, A_{j_2}, ..., A_{j_m}$ form the primary key for the relation. The primary key attributes are required to be non-null and unique; that is, no tuple can have a null value for a primary key attribute, and no two tuples in the relation can be equal on all the primary-key attributes. Although the primary key specification is optional, it is generally a good idea to specify a primary key for each relation.
- **check(P):** The check clause specifies a predicate P that must be satisfied by every tuple in the relation.

SQL

Example: Consider the following definition of tables:-

- create table customer (customer-name char(20),
  customer-street char(30),
  customer-city char(30),
  primary key (customer-name))
- create table branch
  (branch-name char(15),
  branch-city char(30),
  assets integer,
  primary key (branch-name),
  check (assets $\geq$ 0))
- create table account
  (account-number char(10),
  branch-name char(15),
  balance integer,
  primary key (account-number),
  check (balance $\geq$ 0))

SQL

- create table depositor
  (customer-name char(20),
  account-number char(10),
  primary key (customer-name, account-number))
- create table student
  (name char(15) not null,
  student-id char(10),
  degree-level char(15),
  primary key (student-id),
  check (degree-level in ('Bachelors', 'Masters', 'Doctorate')))

Note: SQL also supports an integrity constraint

$$\text{unique } (A_{j_1}, A_{j_2}, ..., A_{j_m})$$

The unique specification says that attributes $A_{j_1}, A_{j_2}, ..., A_{j_m}$ form a candidate key.

Some queries

Consider the following relation schemas:-

Branch-schema = (branch-name, branch-city, assets)
Customer-schema = (customer-name, customer-street, customer-city)
Loan-schema = (loan-number, branch-name, amount)
Borrower-schema = (customer-name, loan-number)
Account-schema = (account-number, branch-name, balance)
Depositor-schema = (customer-name, account-number)

SQL

- Find the names of all branches in the loan relation.
  Solution:

  ```
  select branch-name
  from loan
  ```

- Find all loan numbers for loans made at the Perryridge branch with loan amounts greater that $1200.
  Solution:

  ```
  select loan-number
  from loan
  where branch-name = 'Perryridge' and amount > 1200
  ```

SQL

- Find the loan number of those loans with loan amounts between \$90,000 and \$100,000.

  Solution:

  ```
  select loan-number
  from loan
  where amount ≤ 100000 and amount ≥ 90000
  ```

- For all customers who have a loan from the bank, find their names,loan numbers and loan amount.

  Solution:

  ```
  select customer-name, borrower.loan-number, amount
  from borrower, loan
  where borrower.loan-number = loan.loan-number
  ```

SQL

- Find the customer names, loan numbers, and loan amounts for all loans at the Perryridge branch.

  Solution:

  select customer-name, borrower.loan-number, amount
  from borrower, loan
  where borrower.loan-number = loan.loan-number and
  branch-name = 'Perryridge'

Rename operation

SQL provides a mechanism for renaming both relations and attributes. It uses the **as** clause, taking the form:

old-name as new-name

The **as** clause can appear in both the select and from clauses.

Example:

select customer-name, borrower.loan-number as loan-id, amount

from borrower, loan

where borrower.loan-number = loan.loan-number

SQL

- For all customers who have a loan from the bank, find their names, loan numbers, and loan amount

  Solution:

  select customer-name, T.loan-number, S.amount
  from borrower as T, loan as S
  where T.loan-number = S.loan-number

- Find the names of all branches that have assets greater than at least one branch located in Brooklyn.

  Solution:

  select distinct T.branch-name
  from branch as T, branch as S
  where T.assets > S.assets and S.branch-city = 'Brooklyn'

String Operations

The most commonly used operation on strings is pattern matching using the operator **like**. We describe patterns by using two special characters:

- Percent (%): The % character matches any substring.
- Underscore (_): The _ character matches any character.

Patterns are case sensitive; that is, uppercase characters do not match lowercase characters, or vice versa. To illustrate pattern matching, we consider the following examples:

- 'Perry%' matches any string beginning with "Perry".
- '%idge%' matches any string containing "idge" as a substring, for example, 'Perryridge', 'Rock Ridge', 'Mianus Bridge', and 'Ridgeway'.
- '___' matches any string of exactly three characters.
- '___%' matches any string of at least three characters.

Example:

Find the names of all customers whose street address includes the substring ‘Main’.

Solution:

select customer-name

from customer

where customer-street like ’%Main%’

Ordering the Display of Tuples

To display the result in the sorted order, we use the **order by** clause.

Example: To list in alphabetic order all customers who have a loan at the Perryridge branch.

Solution:

```
select distinct customer-name
from borrower, loan
where borrower.loan-number = loan.loan-number and
branch-name = 'Perryridge'
order by customer-name
```

Example:

```
select *
from loan
order by amount desc, loan-number asc
```