UCS310 Database Management System

Introduction to SQL

Lecture-07

Date: 30 Jan 2023

Dr. Sumit Sharma
Assistant Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala

Outline

- Overview of The SQL Query Language
- SQL Data Definition
- Basic Query Structure of SQL Queries
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

SQL Parts

- **DML** -- provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- **integrity** – the DDL includes commands for specifying integrity constraints.
- **View definition** -- The DDL includes commands for defining views.
- **Transaction control** –includes commands for specifying the beginning and ending of transactions.
- **Embedded SQL and dynamic SQL** -- define how SQL statements can be embedded within general-purpose programming languages.
- **Authorization** – includes commands for specifying access rights to relations and views.

Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Domains Types in SQL

- **char(n)**. Fixed length character string, with user-specified length n.
- **varchar(n)**. Variable length character strings, with user-specified maximum length n.
- **int**. Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d)**. Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point. (ex., numeric(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)
- real, double precision. Floating point and double-precision floating point numbers, with machine-dependent precision.
- float(n). Floating point number, with user-specified precision of at least n digits.

Create Table Construct

- An SQL relation is defined 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$_k$))

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i

- Example:

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20)**,**
  dept_name **varchar**(20),
  salary **numeric**(8,2))

Integrity Constraints in Create Table

- Types of integrity constraints
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references s**
  - **not null**
- SQL prevents any update to the database that violates an integrity constraint.
- Example:

create table *instructor* (
ID **char**(5),
name **varchar**(20) **not null,**
dept_name **varchar**(20),
salary **numeric**(8,2),
primary key (*ID*),
foreign key *(dept_name*) **references** *department);*

Integrity Constraints in Create Table

- Types of integrity constraints
  - **primary key** (A_1, ..., A_n):

- The primary key attributes are required to be *nonnull* 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

.

Integrity Constraints in Create Table

- Types of integrity constraints
  - **foreign key** (A_m, ..., A_n) **references** s

    the values of attributes (A_{k1} , A_{k2}, . . . , A_{kn}) for any tuple in the relationship must correspond to values of the primary key attributes of some tuple in relation s.

  - **not null**

    the constraint excludes the null value from the domain of that attribute

Integrity Constraints in Create Table

- Types of integrity constraints
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references s**
  - **not null**
- SQL prevents any update to the database that violates an integrity constraint.
- Example:

```
create table instructor (
        ID              char(5),
        name            varchar(20) not null,
        dept_name       varchar(20),
        salary          numeric(8,2),
        primary key (ID),
        foreign key (dept_name) references department);
```

And a Few More Relation Definitions

- **create table** *student* (
  ID **varchar**(5),
  name **varchar**(20) not null,
  dept_name **varchar**(20),
  tot_cred **numeric**(3,0),
  primary key *(ID),*
  foreign key *(dept_name*) **references** *department*);

- **create table** *takes* (
  ID **varchar**(5),
  course_id **varchar**(8),
  sec_id **varchar**(8),
  semester **varchar**(6),
  year **numeric**(4,0),
  grade **varchar**(2),
  primary key *(ID, course_id, sec_id, semester, year)* ,
  foreign key (*ID*) **references** *student,*
  foreign key (*course_id, sec_id, semester, year*) **references** *section*);

And a Few More…

- **create table** *course* (
  course_id **varchar**(8),
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  primary key *(course_id),*
  foreign key *(dept_name*) **references** *department*);

Updates to tables

- **Insert**
  - **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **Delete**
  - Remove all tuples from the *student* relation
    - **delete from** *student*
- **Drop Table**
  - **drop table** *r*
- **Alter**
  - **alter table** *r* **add** *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
    - All exiting tuples in the relation are assigned *null* as the value for the new attribute.
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.

Basic Query Structure

- A typical SQL query has the form:

$$\textbf{select } A_1, A_2, \ldots, A_n$$
$$\textbf{from } r_1, r_2, \ldots, r_m$$
$$\textbf{where } P$$

  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.
- The result of an SQL query is a relation.

The Select Clause

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

  select *name*

  from *instructor*
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever we use bold font.

The Select Clause

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select

- Find the department names of all instructors, and remove duplicates

```
select distinct dept_name
from instructor
```

- The keyword specifies that duplicates should not be removed.

```
select all dept_name
from instructor
```

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

The Select Clause

- An asterisk in the select clause denotes “all attributes”

```
select *
from instructor
```

- An attribute can be a literal with no from clause

```
select '437'
```

  - Results is a table with one column and a single row with value “437”
  - Can give the column a name using:

```
select '437' as FOO
```

- An attribute can be a literal with from clause

```
select 'A'
from instructor
```

  - Result is a table with one column and N rows (number of tuples in the instructors table), each row with value “A”

The Select Clause

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.
  - The query:

    select *ID, name, salary/12*
    from *instructor*

    would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.
  - Can rename "s*alary/12"* using the **as** clause:

    select *ID, name, salary/12* as *monthly_salary*

The where clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

- SQL allows the use of the logical connectives **and, or,** and **not**
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <>.
- Comparisons can be applied to results of arithmetic expressions
- To find all instructors in Comp. Sci. dept with salary > 70000

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 70000
```

name
Katz
Brandt

The from Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

```
select *
from instructor, teaches
```

  - generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Cartesian Product of instructor and teaches table

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

The from Clause with where condition

- **select** *
  from *instructor, teaches*
  where *instructor.ID = teaches.ID*

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Natural Join

- Find the names of all instructors who have taught some course and the course_id

  select *name, course_id*
  from *instructor , teaches*
  where *instructor.ID = teaches.ID*

- Can be written more concisely using natural join

  select name, course id

  from instructor natural join teaches;

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

The from Clause

- Find the names of all instructors who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID*
- Find the names of all instructors in the Art department who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID*
    and *instructor. dept_name* = 'Art'

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
  - **select distinct** *T.name*
    from *instructor* **as** *T*, *instructor* **as** *S*
    where *T.salary* > *S.salary* **and** *S.dept_name* = *'Comp. Sci.'*

- Keyword **as** is optional and may be omitted

  instructor **as** *T* ≡ *instructor T*

String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator **like** uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all instructors whose name includes the substring “dar”.

```
select name
from instructor
where name like '%dar%'
```

- Match the string “100%”

```
like '100 \%'  escape  '\'
```

in that above we use backslash (\) as the escape character.

String Operations

- Patterns are case sensitive.
- Pattern matching examples:
  - 'Intro%' matches any string beginning with “Intro”.
  - '%Comp%' matches any string containing “Comp” as a substring.
  - '_ _ _' matches any string of exactly three characters.
  - '_ _ _ %' matches any string of at least three characters.
- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors

```
select distinct name
from   instructor
order by name
```

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**
- Can sort on multiple attributes
  - Example: **order by** *dept_name* **desc**, *name* **asc**

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salaries between $90,000 and $100,000 (that is, >= $90,000 and <= $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000

- Tuple comparison
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor.ID*, *dept_name*) = (*teaches.ID*, 'Biology');

Set Operations

- Find courses that ran in Fall 2017 or in Spring 2018

```
(select course_id  from section where sem = 'Fall' and year = 2017)
 union
(select course_id  from section where sem = 'Spring' and year = 2018)
```

- Find courses that ran in Fall 2017 and in Spring 2018

```
(select course_id  from section where sem = 'Fall' and year = 2017)
 intersect
(select course_id  from section where sem = 'Spring' and year = 2018)
```

- Find courses that ran in Fall 2017 but not in Spring 2018

```
(select course_id  from section where sem = 'Fall' and year = 2017)
 except
(select course_id  from section where sem = 'Spring' and year = 2018)
```

Set Operations

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the
  - **union all**
  - **intersect all**
  - **except all**

Thanks!