Database Management Systems

Module 08: Introduction to SQL/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced relational algebra
- Familiarized with the operators of relational algebra

Module Objectives

- To understand relational query language
- To understand data definition and basic query structure

Module Outline

- History of SQL
- Data Definition Language (DDL)
- Data Manipulation Language (DML): Query Structure

History of SQL

History of Query Language

- IBM developed *Structured English Query Language* (**SEQUEL**) as part of System R project. Renamed Structured Query Language (SQL: *pronounced still as* **SEQUEL**)
- ANSI and ISO standard SQL:

SQL-86	First formalized by **ANSI**
SQL-89	+ **Integrity Constraints**
SQL-92	Major revision (**ISO/IEC 9075 standard**), **De-facto Industry Standard**
SQL:1999	+ **Regular Expression Matching**, **Recursive Queries**, **Triggers**, **Support for Procedural and Control Flow Statements**, Nonscalar types (Arrays), and Some OO features (structured types), **Embedding SQL in Java (SQL/OLB)**, and **Embedding Java in SQL (SQL/JRT)**
SQL:2003	+ **XML features (SQL/XML)**, Window Functions, Standardized Sequences, and Columns with Auto-generated Values (identity columns)
SQL:2006	+ Ways of **importing and storing XML data** in an SQL database, **manipulating** it within the database, and **publishing both XML and conventional SQL-data in XML form**
SQL:2008	Legalizes ORDER BY outside Cursor Definitions + INSTEAD OF Triggers, TRUNCATE Statement, and FETCH Clause
SQL:2011	+ Temporal Data (PERIOD FOR) Enhancements for Window Functions and FETCH Clause
SQL:2016	+ Row Pattern Matching, Polymorphic Table Functions, and **JSON**
SQL:2019	+ Multidimensional Arrays (MDarray type and operators)

History of Query Language (2): Compliance

- SQL is the de facto industry standard today for relational or structred data systems
- Commercial systems as well as open systems may be fully or partially compliant to one or more standards from SQL-92 onward
- Not all examples here may work on your particular system. Check your system's SQL documentation

History of Query Language (3): Alternatives

- There aren't any alternatives to SQL for speaking to relational databases (that is, SQL as a protocol), but there are many alternatives to writing SQL in the applications
- These alternatives have been implemented in the form of frontends for working with relational databases. Some examples of a frontend include (for a section of languages):
  - SchemeQL and CLSQL, which are probably the most flexible, owing to their Lisp heritage, but they also look like a lot more like SQL than other frontends
  - LINQ (in .Net)
  - ScalaQL and ScalaQuery (in Scala)
  - SqlStatement, ActiveRecord and many others in Ruby
  - HaskellDB
  - ...the list goes on for many other languages.

Source: *What are good alternatives to SQL (the language)?*

History of Query Language (4): Derivatives

- There are several query languages that are derived from or inspired by SQL. Of these, the most popular and effective is SPARQL.
  - **SPARQL** (pronounced *sparkle*, a recursive acronym for *SPARQL Protocol and RDF Query Language*) is an RDF query language
    - A semantic query language for databases - able to retrieve and manipulate data stored in Resource Description Framework (RDF) format.
    - It has been standardized by the W3C Consortium as key technology of the semantic web
    - Versions:
      - SPARQL 1.0 (January 2008)
      - SPARQL 1.1 (March, 2013)
    - Used as the query languages for several NoSQL systems - particularly the Graph Databases that use RDF as store

Data Definition Language (DDL)

Data Definition Language (DDL)

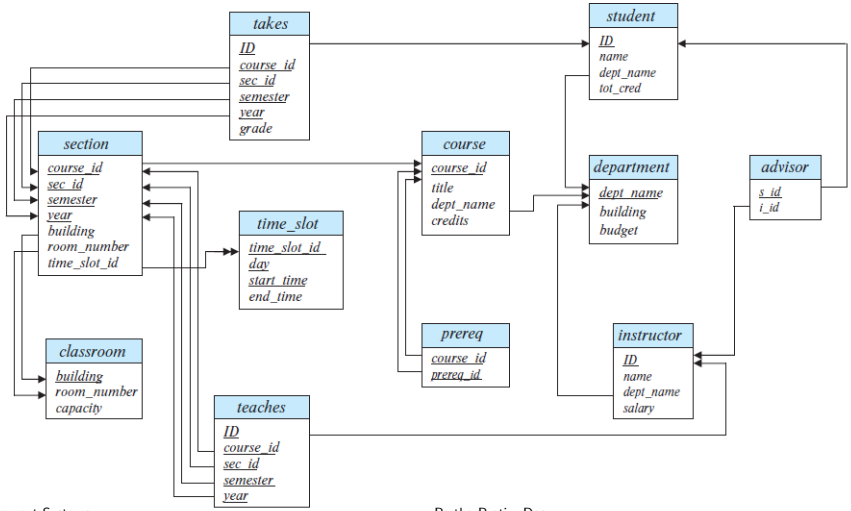
The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The *Schema* for each *Relation*
- The *Domain* of values associated with each *Attribute*
- *Integrity Constraints*
- And, as we will see later, also other information such as
  - The set of *Indices* to be maintained for each relations
  - *Security and Authorization* information for each relation
  - The *Physical Storage Structure* of each relation on disk

Domain 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**(n). 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
- More are covered in Chapter 4

Schema Diagram for University Database



Create Table Construct

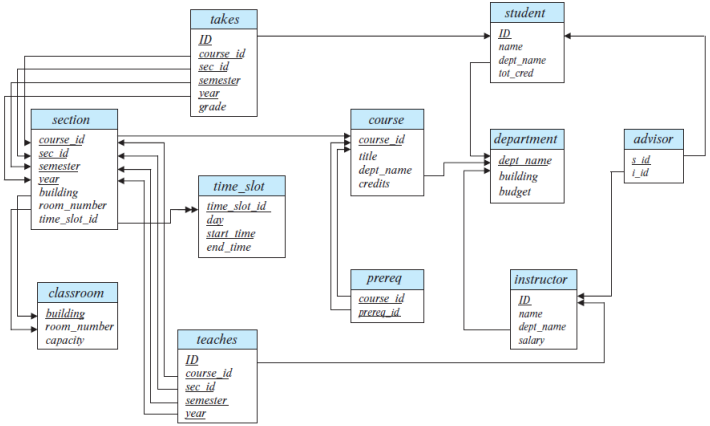
- An SQL relation is defined using the **create table** command:

  create table r $(A_1 D_1, A_2 D_2, \ldots, A_n D_n)$,
  $(integrity\text{-}constraint_1)$,
  $\ldots$
  $(integrity\text{-}constraint_k)$);

  - 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

Create Table Construct (2)

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



Create Table Construct (3): Integrity Constraints

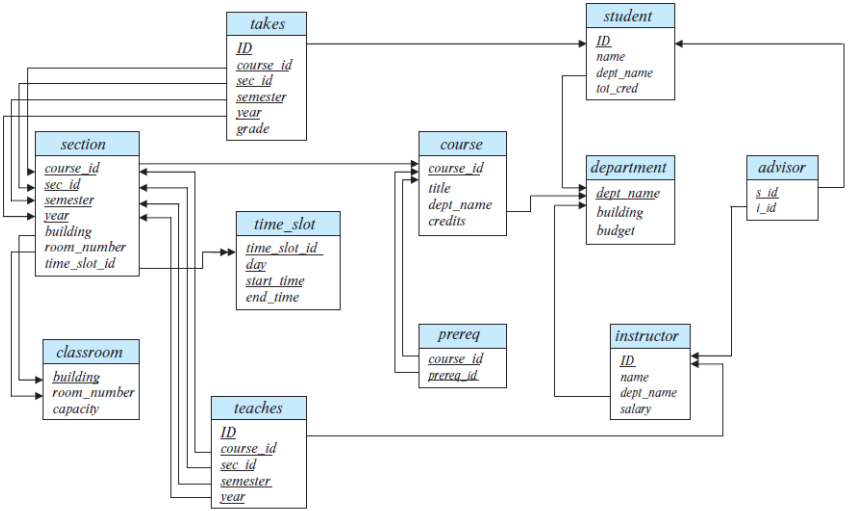
- **not null**
- **primary key** $(A_1, \ldots, A_n)$
- **foreign key** $(A_m, \ldots, A_n)$ **references** r

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

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*));

primary key declaration on an attribute automatically ensures **not null**

University Schema



Create Table Construct (4): More Relations

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 *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*);

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*);

- **Note**: *sec_id* can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester

Update Tables

- **Insert (DML command)**
  - **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **Delete (DML command)**
  - Remove all tuples from the *student* relation

    delete from *student*
- **Drop Table (DDL command)**
  - **drop table** *r*
- **Alter (DDL command)**
  - **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 existing 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

Data Manipulation Language (DML): Query Structure

Basic Query Structure

- A typical SQL query has the form:

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

  - A_i represents an attribute from r_i's
  - r_i represents a relation
  - P is a predicate
- The result of an SQL query is a relation

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 (that is, you may use upper-case or lower-case letters)
  - $Name \equiv NAME \equiv name$
  - Some people use upper case wherever we use bold font

Select Clause (2)

- **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 **all** specifies that duplicates should not be removed

  select all *dept_name*
  from *instructor*

Select Clause (3)

- 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'

Select Clause (4)

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 "*salary*/12" using the **as** clause:

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

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.'
- Comparison results can be combined using the logical connectives **and**, **or**, and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

    select *name*
    from *instructor*
    where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000
- Comparisons can be applied to results of arithmetic expressions

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 (for example, *ID*), the attributes in the resulting table are renamed using the relation name (for example, *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Cartesian Product

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821			
98345			

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
				2010
				2009
				2009
				2010
				2009

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Module Summary

- Introduced relational query language
- Familiarized with data definition and basic query structure

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".