Chapter 3: Introduction to SQL

Chapter 3: Introduction to SQL

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

Structured Query Language (SQL)

- Introduced in 1974 by IBM
- “De facto” standard db query language
- Caveats
  - Standard has evolved (major revisions in 1992 and 1999, and later in 2003, 2008)
  - Semantics and Syntax may vary slightly among DBMS implementations
  - Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

Data Definition Language

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 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.** 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 *n* digits to the right of decimal point.
- **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.

Create Table Construct

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

  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) **not null,**
  dept_name **varchar**(20),
  salary **numeric**(8,2))
- **insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, 66000);
- **insert into** *instructor* **values** (‘10211’, null, ’Biology’, 66000);

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*

Example: Declare *ID* as the primary key for *instructor*

.

```
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**

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*);

  - 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

And more still

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

- Primary key declaration can be combined with attribute declaration as shown above

Drop and Alter Table Constructs

- **drop table** *student*
  - Deletes the table and its contents
- **delete from** *student*
  - Deletes all contents of table, but retains table
- **alter table**
  - **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 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

- The SQL **data-manipulation language (DML)** provides the ability to query information, and insert, delete and update tuples
- A typical SQL query has the form:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  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 list 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 some use bold font.

Instructor (ID, name, dept_name, salary)

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Result of “SELECT name FROM instructor”.

The select Clause (Cont.)

- 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 names of all departments with instructor, and remove duplicates

  SELECT DISTINCT *dept_name*
  FROM *instructor*

- The keyword **all** specifies that duplicates not be removed.

  SELECT ALL *dept_name*
  FROM *instructor*

Instructor (ID, name, dept_name, salary)

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

Result of “SELECT dept name FROM instructor”.

The select Clause (Cont.)

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

  SELECT *
  FROM *instructor*

- 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.

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 with salary > 80000

  SELECT *name*
  FROM *instructor*
  WHERE *dept_name* = 'Comp. Sci.' **AND** *salary* > 80000

- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions.

Instructor (ID, name, dept_name, salary)

name
Katz
Brandt

Result of “Find the names of all instructors in the Computer Science departm
who have salary greater than $70,000.”

What is the SQL Query?

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
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Cartesian Product: *instructor X teaches*

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

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

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	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Joins

- For all instructors who have taught some course, find their names and the course ID of the courses they taught.

```
SELECT name, course_id
FROM   instructor, teaches
WHERE  instructor.ID = teaches.ID
```

- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

```
SELECT section.course_id, semester, year, title
FROM   section, course
WHERE  section.course_id = course.course_id  AND
           dept_name = 'Comp. Sci.'
```

Instructor (ID, name, dept_name, salary)

Course (course_id, title, dept_name, credits)

Section (course_id, section_id, semester, year)

Try Writing Some Queries in SQL

Instructor (ID, name, dept_name, salary)

Course (course_id, title, dept_name, credits)

Section (course_id, section_id, semester, year)

Teaches (ID, course_id, sec_id, semester, year)

Department(dept_name, building, budget)

"Retrieve the names of all instructors, along with their department
names and department building name."

Instructor (ID, name, dept_name, salary)
Course (course_id, title, dept_name, credits)
Section (course_id, section_id, semester, year)
Teaches (ID, course_id, sec_id, semester, year)
Department(dept_name, building, budget)

name	*dept_name*	*building*
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Califieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor

In SQL??

```
SELECT name, dept_name, building
FROM Instructor, Department
WHERE Instructor.dept_name = Department.dept_name
```

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
- **SELECT** *
  FROM *instructor* **NATURAL JOIN** *teaches*;

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

Natural Join Example

Instructor (ID, name, dept_name, salary)

Teaches (ID, course_id, sec_id, semester, year)

- List the names of instructors along with the course ID of the courses that they taught.

  - **SELECT** *name*, *course_id*
    FROM *instructor, teaches*
    WHERE *instructor.ID = teaches.ID*;

  - **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

Result of “For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught.”

Instructor (ID, name, dept_name, salary)
Teaches (ID, course_id, sec_id, semester, year)

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

The natural join of the instructor relation with the teaches relation.

Natural Join (Cont.)

Instructor (ID, name, dept_name, salary)
Course (course_id, title, dept_name, credits)
Section (course_id, section_id, semester, year)

- Danger in natural join: beware of unrelated attributes with same name which get equated incorrectly
- List the names of instructors along with the the titles of courses that they teach
  - Incorrect version (makes course.dept_name = instructor.dept_name)
    - **SELECT** *name*, *title*
      FROM *instructor* **NATURAL JOIN** *teaches* **NATURAL JOIN** *course*;
  - Correct version
    - **SELECT** *name*, *title*
      FROM *instructor* **NATURAL JOIN** *teaches*, *course*
      WHERE *teaches.course_id = course.course_id*;
  - Another correct version
    - **SELECT** *name*, *title*
      FROM (*instructor* **NATURAL JOIN** *teaches*)
      JOIN *course* **USING** (*course_id*);

The Rename Operation

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

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

- E.g.
  - **SELECT** *ID, name, salary/12* **as** *monthly_salary*
    FROM *instructor*

- 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*
  - Keyword **as** must be omitted in Oracle

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  '\'
```

- **LIKE ‘**and\\or’ **escape** '\’ matches .....

String Operations (Cont.)

- 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, name*

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary 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 2009 or in Spring 2010

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
union
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 and in Spring 2010

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
intersect
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 but not in Spring 2010

(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
except
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

Set Operations

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

  Suppose a tuple occurs *m* times in *r* and *n* times in *s*, then, it occurs:
  - *m* + *n* times in *r* **union all** *s*
  - min(*m*,*n*) times in *r* **intersect all** *s*
  - max(0, *m* – *n*) times in *r* **except all** *s*

course_id
CS-101
CS-347
PHY-101

The c1 relation, listing courses taught in Fall 2009.

course_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

The c2 relation, listing courses taught in Spring 2010.

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

The result relation for c1 union c2.

course_id
CS-101

The result relation for c1 intersect c2.

course_id
CS-347
PHY-101

The result relation for c1 except c2.

Next:

- Null Values
- Aggregate Functions
- Nested Subqueries