SQL – Part 1

MSCI346
Lukasz Golab

Acknowledgement: slides derived from material provided by Sliberschatz, Korth and Sudarshan, copyright 2019, www.db-book.com, and by Prof. Wojciech Golab, ECE356

Learning Outcomes

- brief history of SQL
- understand the difference between SQL DDL and DML
- understand multiset relations
- develop working knowledge of basic SQL syntax and semantics
  - selection and projection
  - Cartesian product and joins
  - renaming
  - set operations
  - aggregation and grouping
  - ordering results
  - nested subqueries
  - ~~JDBC, SQL injection~~
- Textbook sections (6th ed.): 3.1, 3.3, 3.4, 3.5, 3.7, 3.8, 4.1, 5.1.1

Brief History of SQL

- SEQUEL (Structured English Query Language) developed for *System R* project at IBM San Jose Research Lab in early 1970s. Later shortened to SQL (Structured Query Language).
- Standardized by ANSI (since 1986) and ISO (since 1987):
  - SQL-86 (first cut)
  - SQL-89 (added integrity constraints)
  - **SQL-92** (DATE and VARCHAR types, NATURAL JOIN)
  - SQL:1999 (regular expressions, recursive queries, triggers)
  - SQL:2003 (XML support, auto-generated values)
  - SQL:2008 (revisions to cursors and triggers)
  - SQL:2011 (support for temporal databases)
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

SQL DDL, DML, DCL, TCL

- The SQL **Data Definition Language (DDL)** provides the ability to specify schema includes domains and integrity constraints.
  - e.g., creating tables and declaring foreign keys
- The SQL **Data Manipulation Language (DML)** provides the ability to query data, as well as insert/delete/update tuples.
  - e.g., SELECT statement
- The SQL **Data Control Language (DCL)** provides the ability to grant access privileges to users and to revoke such privileges.
- The SQL **Transaction Control Language (TCL)** provides the ability to control the execution of transactions.
- We will cover **SQL DML** in this module. The next module will cover additional DML concepts as well as **SQL DDL**.

Schema for Running Example

- *instructor* (*ID*, *name*, *dept_name*, *salary*)
- *teaches* (*ID*, *course_id*, *sec_id*, *semester*, *year*)
- *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_no*, *time_slot_id*)
- *course* (*course_id*, *title*, *dept_name*, *credits*)

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

Multiset Relations

- SQL borrows concepts from both relational algebra and relational calculus.
- Because removing duplicates is expensive, in practice relational databases allow duplicate tuples in relations, including in query results.
- In other words, relations are treated as **multisets.**
- Example:

select dept_name **from** Instructor;

Π_{dept_name} (*instructor*)

Note: primary keys avoid duplicate tuples in tables but not in intermediate query results.

dept_name
ECE
ECE
ECE
SYDE
ME
ECE
ECE
SYDE
ECE

Multiset Relations (Cont.)

- Basic SQL constructs can be defined formally using relational algebra (RA) over multiset relations.
- Given multiset relations r_1 and r_2, we define multiset RA operators as follows:

1. $\sigma_\theta(r_1)$: If there are c1 copies of tuple t1 in r1 and t1 satisfies θ, then there are c1 copies of t1 in $\sigma\theta$ (r1).
2. $\Pi_A(r)$: For each copy of tuple t_1 in r_1, then there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1.
3. $r_1 \times r_2$: If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, then there are $c_1 \times c_2$ copies of the tuple $t_1.t_2$ in $r_1 \times r_2$.

Multiset Relations: Example

- Suppose that multiset relations r_1 (*A, B*) and r_2 (*C*) are defined as follows:

  $r_1 = \{(1,a)\ (2,a)\}$ $\quad$ $r_2 = \{(2),\ (3),\ (3)\}$

- Then $\Pi_B(r_1)$ would be

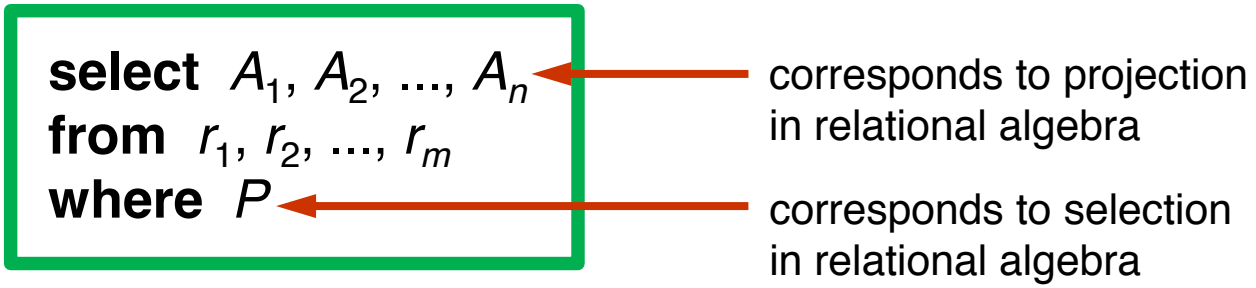
  {(a), (a)}

  while $\Pi_B(r_1)$ x r_2 would be

  {(*a*,2), (*a*,2), (*a*,3), (*a*,3), (*a*,3), (*a*,3)}

Basic Query Structure

- A simple SQL query has the form:



  - A_i represents an attribute
  - r_i represents a relation
  - P is a predicate over the relations

- The result of an SQL query is a multiset relation.
- The query is equivalent to the multiset version of the relational algebra expression

$$\prod_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

The select Clause

- The **select** clause lists the requested attributes in the result of a query. Corresponds to projection in relational algebra.
- Example: find the names of all instructors:

select *name*
from *instructor*

- In general relation names are case-sensitive but attribute names are not. (Some exceptions, like MySQL running on Windows.)

The select Clause (Cont.)

- The keyword **distinct** eliminates duplicates. The keyword **all** keeps duplicates. In MySQL, the default is **all**.
- Example: find the names of all departments with at least one instructor, and remove duplicates.

```
select distinct dept_name
from instructor
```

- Example: find the names of all departments with at least one instructor, and keep duplicates.

```
select all 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 operators +, –, ∗, 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 (and renamed "salary/12").

The where Clause

- The **where** clause specifies conditions that the result must satisfy. Corresponds to selection in relational algebra.
- Example: find all instructors in Physics with *salary* > 80000.

  select *name*
  from *instructor*
  where *dept_name* = 'Physics' **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.

The from Clause

- The **from** clause lists the relations involved in the query. Corresponds to the Cartesian product in relational algebra.
- Example: find the Cartesian product *instructor* x *teaches*.

select *
from *instructor, teaches*

This query generates every possible instructor-teaches pair, with all attributes from both relations.

Warning: the result set of a Cartesian product can be very large!

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
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Theta Join

- A Theta join is obtained using an **inner join** with the **on** clause.
- Example: find the course ID, semester, year and title of each course offered in the Physics department.

select *section.course_id, semester, year, title*
from *section* **inner join** *course*
on (*section.course_id = course.course_id* **and**
dept_name = 'Physics')

Equivalent to:

$\Pi_{section.course\_id,\ semester,\ year,\ title}$ (
$section \bowtie_{section.course\_ID=course.course\_ID \wedge dept\_name=\text{'Physics'}} course$)

Equivalent to:

$\Pi_{section.course\_id,\ semester,\ year,\ title}$ (
$\sigma_{section.course\_ID=course.course\_ID \wedge dept\_name=\text{'Physics'}}$ ($section \times course$))

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common attribute.
- Example: **select** * **from** *instructor* **natural join** *teaches*
- Equivalent to *instructor* ⋈ *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: find the names of instructors along with the course ID of the courses they teach.

select *name*, *course_id*
from *instructor* **natural join** *teaches*

select *name*, *course_id*
from *instructor* **inner join** *teaches* **using** (*ID*)

Note: In this case the inner join retains only one copy of *ID* because *ID* is specified in the **using** clause.

Natural Join (Cont.)

- Danger in natural join: beware of unrelated attributes with same name that get equated incorrectly.
- Example: list the names of instructors along with the titles of courses that they teach.
  - Incorrect solution (makes course.dept_name = instructor.dept_name):
    - **select** *name*, *title*
      from *instructor* **natural join** *teaches* **natural join** *course*
  - Correct solution:
    - **select** *name*, *title*
      from *instructor* **natural join** *teaches*, *course*
      where *teaches.course_id* = *course.course_id*
  - Another correct solution:
    - **select** *name*, *title*
      from (*instructor* **natural join** *teaches*)
      inner join *course* **using** (*course_id*)

The Rename Operation

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

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

- Example:
  - **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 'Physics'.
  - **select distinct** *T.name*
    from *instructor* **as** *T, instructor* **as** *S*
    where *T.salary* > *S.salary* **and** *S.dept_name* = '*Physics*'

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

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

(Note: keyword **as** must be omitted in Oracle)

Set Operations

- Find courses offered in Fall 2009 or in Spring 2010

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

- Find courses offered in Fall 2009 and in Spring 2010

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

- Find courses offered in Fall 2009 but not in Spring 2010

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

Note: MySQL does not support **intersect** or **except.** Remedy?

Set Operations

- Set operations **union**, **intersect**, and **except** automatically eliminate 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*

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- Use the keyword **distinct** to eliminate duplicates.
  Example: … **count**(**distinct** *ID*) …

Aggregate Functions (Cont.)

- Example: find the average salary of instructors in the Physics department.
  - **select avg**(*salary*)
    from *instructor*
    where *dept_name*= 'Physics'
- Example: find the total number of instructors who teach a course in the Spring 2010 semester.
  - **select count**(**distinct** *ID*)
    from *teaches*
    where *semester* = 'Spring' **and** *year* = 2010
- Example: find the number of tuples in the *course* relation.
  - **select count**(*)
    from *course*

- Note: avoid spaces between the aggregate function and (

Aggregation with Grouping

- Find the average salary of instructors in each department
  - **select** *dept_name*, **avg**(*salary*)
    from *instructor*
    group by *dept_name*
  - Note: departments with no instructor will not appear in result

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation (Cont.)

- Attributes in **select** clause outside of aggregate functions must appear in the **group by** list:
  - /* incorrect */
    select *dept_name*, *name*, **avg**(*salary*)
    from *instructor*
    group by *dept_name*

Returns a group for each distinct *dept_name* in relation *instructor*.

Note: MySQL 5.5 does accept the above syntax. For each department group it will return one instructor name from that group.

Aggregation with "Having" Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000.

```
select dept_name, avg(salary)
from instructor
group by dept_name
having avg(salary) > 42000
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups.

Ordering the Display of Tuples

- Example: list in alphabetical order the names of all instructors.

  select distinct *name*
  from *instructor*
  order by *name*

- Specify **desc** for descending order or **asc** for ascending order, for each attribute. Ascending order is the default.
  - example: **order by** *name* **desc**
- Possible to sort on multiple attributes.
  - example:

    select distinct *dept_name, name*
    from *instructor*
    order by *dept_name* **asc**, *name* **desc**

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- Common uses of subqueries including performing tests for set membership (e.g., a value is in the set or not) and set cardinality (e.g., set is empty or not), as well as performing set operations (e.g., union, intersect, difference).
- The **in** construct is used for testing set membership. It returns **true** if the value on the left is an element of the relation generated by a subquery on the right.

  select … **from** … **where** *value* **in** (*subquery*)

  select … **from** … **where** *value* **not in** (*subquery*)

Testing for Set Membership

- Example: find courses offered in Fall 2009 and in Spring 2010.

```
select distinct course_id
from section
where semester = 'Fall' and year = 2009 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year = 2010)
```

- Example: find courses offered in Fall 2009 but not in Spring 2010.

```
select distinct course_id
from section
where semester = 'Fall' and year = 2009 and
        course_id  not in (select course_id
                           from section
                           where semester = 'Spring' and year = 2010)
```

Testing for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
  - **exists** $r \Leftrightarrow r \neq \emptyset$
  - **not exists** $r \Leftrightarrow r = \emptyset$
- Example: find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester.

```
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
        exists (select *
                from section as T
                where semester = 'Spring' and year = 2010
                      and S.course_id = T.course_id)
```

Correlated subquery: uses values from outer query in **where** clause.

Test for Empty Relations (Cont.)

- Example: find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course as C
                    where C.dept_name = 'Biology' and
                    not exists
                     (select T.course_id from takes as T
                      where C.course_id = T.course_id
                      and S.ID = T.ID) ) )
```

Note: This example is similar to set division.

Scalar Subquery

- **Scalar subquery** is one that yields a single value.
- Example:

```
select dept_name,
    (select count(*)
     from instructor
     where department.dept_name = instructor.dept_name)
as num_instructors
from department
```

- Example:

```
select name
from instructor
where  salary * 10 >
    (select budget  from department
     where department.dept_name = instructor.dept_name)
```

- Note: An error occurs if the subquery returns more than one result tuple.