Introduction to SQL

Overview

- **Data-definition language (DDL)**:
  - defining and modifying relation schemas
  - integrity constrains
  - view definition
  - authorization: access rights to relations and views.
- **Data-manipulation language (DML)**: querry information from and modify tuples in relations
- **Transaction Control**: Specifying beginning and end points of transactions.
- **Embedded/dynamic SQL**: how SQL is embedded in general-purpose programmig languages.

SQL DDL

following can be specified:

- schema for each relation
- set of indices to be maintained for each relation
- security and authorization information for each relation
- physical storage structure of each relation on disk

Basic Types

- **char(n)**: fixed n-long string. (full form: **character(n)**)
- **varchar(n)**: variable-length character string with max length n. (full form **character varying(n)**)
- **int** (full form: **integer**)
- **smallint**
- **numeric(p, d)**: p digits in total, d of the digits after decimal point.
- **real**, **double precision**: floating-point / double precision floating point.
- **float(n)**: Floating-point with precision of at least n digits.

Basic Schema Definition

```
create table department (
    dept_name   varchar(20),
    building    varchar(15),
    budget      numeric(12,2),
    primary key (dept_name) --primary key integrity constraint
);
```

general form:

```
create table r(
    a1 domain1,
    a2 domain2,
    ...
    a_n domain_n
    [integrity constraint 1],
    ...
    [integrity constraing 2] -- these are optional
)
```

Basic Constraints:

- **primary key**: a1, ..., a_n together form the primary key of the relation:

  ```
  primary key(a1, a2, ..., a_n)
  ```

- **foreign key**: a1, ..., a_n together form a foreign key over a relation s, i.e. a1, ..., a_n must be a primary key of some tuple in s (existence: referential integrity)

  ```
  foreign key (a1, a2, ..., a_n) references s
  ```

- **not null**:

  ```
  name varchar(20) not null, --name can not be null
  ```

concrete example:

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

Note that explicitly specifying the primary key of the referenced relation

```
foreign key (dept_name) references department(dept_name)
```

is also possible but not required.

Altering the Schema

- remove a relation completely from the database schema:

  ```
  drop table r; --not to be confused with 'delete from'
  ```

- modify a relation schema:

  ```
  alter table r add a1 a2; -- add the attributes a1, a2 to the
   ↪  relation r
  ```

  ```
  alter table r drop a; -- drop/remove attribute a from the relation
   ↪  r
  ```

  some dbs' don't support dropping single attributes but only whole tables.

Basic SQL Querries

A typical sql querry has the form

```
select a1, ..., a_n
from r1, ..., r_m
where P -- P is a predicate/condition
```

```
select name
from instructor
```

Table 1: Displaying records 1 - 10

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

```
select dept_name
from instructor
```

Table 2: Displaying records 1 - 10

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

```
select distinct dept_name --removes duplicates
from instructor
```

Table 3: 7 records

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

Select allows arbitrary expressions of attributes. we can output 10% salary raise for instructors

```
select id, name, dept_name, salary * 1.1
from instructor
```

Table 4: Displaying records 1 - 10

id	name	dept_name	?column?
10101	Srinivasan	Comp. Sci.	71500
12121	Wu	Finance	99000
15151	Mozart	Music	44000
22222	Einstein	Physics	104500
32343	El Said	History	66000
33456	Gold	Physics	95700
45565	Katz	Comp. Sci.	82500
58583	Califieri	History	68200
76543	Singh	Finance	88000
76766	Crick	Biology	79200

Where clause.

- Names of all instructors in the CS department who have salary greater than $70,000:

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

Table 5: 2 records

name
Katz
Brandt

Joining Tables

- names of instructors, names of their departments and the names of the buildings where departments are located:

```
select i.name, d.name as dept_name, building
from instructor i, department d
where i.dept_name = d.name
```

Table 6: Displaying records 1 - 10

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

- names of instructors and identifiers of courses they have tought:

```
select i.name, t.course_id
from instructor i, teaches t
where i.id = t.instructor_id
```

Table 7: Displaying records 1 - 10

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

- names of instructors from the CS department and identifiers of courses that they have tought:

```
select i.name, t.course_id
from instructor i, teaches t
where i.id = t.instructor_id and i.dept_name = 'Comp. Sci.'
```

Table 8: 8 records

name	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Katz	CS-101
Katz	CS-319
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319

Renaming

```
select i.name as instructor_name, t.course_id -- as can be omitted
from instructor as i, teaches as t -- as can be ommited
where i.id = t.instructor_id
```

Table 9: Displaying records 1 - 10

instructor_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

Renaming useful when comparing tuples in the same relation:

- names of instructors whose salary is greater than at least one instructor in the biology department ≈ names of all instructors who earn more than the lowest paid instructor in the Biology department:

```
select distinct i.name
from instructor i , instructor i2
where i.salary > i2.salary and i2.dept_name = 'Biology'
```

Table 10: 7 records

name
Einstein
Katz
Singh
Kim
Brandt
Wu
Gold

- **correlation name = table alias = tuple variable**

String Operations

- concatenation with `||`

```
values
('hey' || ' there!')
```

Table 11: 1 records

column1
hey there!

- `upper()` and `lower()`

```
values
(upper('hey there')),
(lower('HEY THERE'))
```

Table 12: 2 records

column1
HEY THERE
hey there

- removing spaces with `trim()`

```
values
(trim('hey   ' ) || ' there!')
```

Table 13: 1 records

column1
hey there!

Pattern Matching

- `%`: matches any string
- `_`: matches any character
- examples:
  - `intro%`: any string beginning with `Intro'
  - `%Comp%`: any string containing `Comp' as a substring

  - `___`: any string of exactly three characters
  - `___%`: any string of at least three characters

- concrete example; information of courses that have `comp' as a substring in their title:

```
select *
from course c
where c.title ilike '%comp%' --ilike is case insensitive like
```

Table 14: 2 records

id	title	dept_name	credits
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4

*Escaping special characters like ``%'' with ``\'':

```
select *
from (
    values
    ('%15')
) r(a)
where a like '\%%'
```

Table 15: 1 records

a
%15

- Defining custom escape characters other than ``\'':

```
select *
from (
    values
    ('%15')
) r(a)
where a like '^%%' escape '^' -- '^' is defined as the escape character
```

Table 16: 1 records

a
%15

Ordering Display of Tuples

- ordering

```
select name
from instructor
where dept_name  = 'Physics'
order by name;
```

Table 17: 2 records

name
Einstein
Gold

- ordering order, multiple attributes

```
select *
from instructor
order by salary desc, name asc
```

Table 18: Displaying records 1 - 10

id	name	dept_name	salary
22222	Einstein	Physics	95000
83821	Brandt	Comp. Sci.	92000
12121	Wu	Finance	90000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000
76543	Singh	Finance	80000
45565	Katz	Comp. Sci.	75000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000

id	name	dept_name	salary
58583	Califieri	History	62000

Between

```
SELECT name
from instructor i
where i.salary BETWEEN 90000 and 100000
```

Table 19: 3 records

name
Wu
Einstein
Brandt

Tuples in Where Predicates

```
SELECT i.name, t.course_id
from instructor i, teaches t
where i.id = t.instructor_id and i.dept_name = 'Biology'
```

is equivalent to

```
SELECT i.name, t.course_id
from instructor i, teaches t
where (i.id, i.dept_name) = (t.instructor_id, 'Biology')
```

Table 20: 2 records

name	course_id
Crick	BIO-101
Crick	BIO-301

Set Operations

Set operations eliminate duplicates by default. Duplicates retained by `all` keyword.

Union

corresponds to ∪.

- courses offered in 2017 Fall **or** 2018 Spring:

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

Table 21: 8 records

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

Intersect

Corresponds to ∩.

- Courses offered in **both** Fall 2017 and Spring 2018:

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

Table 22: 1 records

course_id
CS-101

Except

Corresponds to \.

- courses offered in the Fall 2017 but not in Spring 2018:

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

Table 23: 2 records

course_id
PHY-101
CS-347

course_id

- Illustrating how `all` works:

`union all`:

```
with r(a) as (
    values
    (1),
    (1)
), s(a) as (
    values
    (1)
)
(select *
from r )
union  all
(select *
from s)
```

Table 24: 3 records

a
1
1
1

`intersect all`:

```
with r(a) as (
    values
    (1),
    (1),
    (1)
), s(a) as (
    values
    (1),
    (1)
)
(select *
```

```
from r )
INTERSECT all
(select *
from s)
```

Table 25: 2 records

a
1
1

except all:

```
with r(a) as (
    values
    (1),
    (1)
), s(a) as (
    values
    (1)
)
(select *
from r )
except  all
(select *
from s)
```

Table 26: 1 records

a
1

```
with r(a) as (
    values
    (1),
    (1)
), s(a) as (
    values
```

```
    (1)
)
(select *
from s )
except  all
(select *
from r)
```

Table 27: 0 records

a

Null

- `null` represents values that are not known. (can be anything).
- arithmetic expressions involving `null` produce `null`:

```
values
(1 + null)
```

Table 28: 1 records

column1
NA

- boolean expressions and predicates involving `null` (other than `is [not] null`) have a special truth value `unknown`.
  - e.g. `(1 < unknown) = uknown`
- truth tables for `unknown`:

p	q	p and q
`false`	`unknown`	`false`
`true`	`unknown`	`unknown`

p	q	p or q
false	unknown	unknown
true	unknown	true

`not unknown = uknown`

- tuples that evaluate to `uknown` in the `where` clause are not included in the result (just like the ones that evaluate to `false`)
- we can test if a value is or isn't null:

```
select a
from (
    values
    (1, 3),
    (2, null)
) r(a, b)
where b is null
```

Table 31: 1 records

a
2

```
select a
from (
    values
    (1, 3),
    (2, null)
) r(a, b)
where b is not null
```

Table 32: 1 records

a
1

- we can test whether the result of a predicate is or isn't `uknown`

```
select a
from (
    values
    (1, 3),
    (2, null)
) r(a, b)
where b > 2 is unknown
```

Table 33: 1 records

a
2

```
select a
from (
    values
    (1, 3),
    (2, null)
) r(a, b)
where b > 2 is not unknown
```

Table 34: 1 records

a
1

Aggregate Functions

- Take collection (i.e. aggregate, multiset) of values as input and return a **single** value.
- Built in aggregate functions in sql:
  - `avg`: input must be numeric
  - `min`
  - `max`
  - `sum`: computes the total sum of the values in the aggregate, input must be numeric
  - `count`

Basic Aggregation

- Average salary of instructors in the CS department:

```
select avg(salary) as avg_salary
from instructor
where dept_name = 'Comp. Sci.'
```

Table 35: 1 records

avg_salary
77333.33

Duplicates are retained. Retention of duplicates is obviously important for calculating averages. But sometimes we may want to eliminate duplicates:

- Find the total number of instructors that teach a course in the Spring 2018 semester:

```
select count(distinct t.instructor_id)
from teaches t
where semester = 'Spring' and year = 2018
```

Table 36: 1 records

count
6

as opposed to

```
select count(t.instructor_id)
from teaches t
where semester = 'Spring' and year = 2018
```

Table 37: 1 records

count
7

- counting all attributes with `count(*)`:

```
select count(*)
from course
```

Table 38: 1 records

count
13

`count(*)` retains `null` values. It is the only aggregate function that does so. All other aggregate functions ignore `null` values, including count functions applied on a single attribute. That is `count(*)` and `count(attr)` are different:

```
select count(*) as count_star, count(a) as count_attribute
from (
    values
    (1),
    (null)
) r(a)
```

Table 39: 1 records

count_star	count_attribute
2	1

Aggregation with Grouping

Instead of applying the aggregate function to a single aggregate (collection), we can apply it to multiple aggregates/collections, that consist of tuples grouped together w.r.t certain grouping attributes:

- find the average salary in each department:

```
select dept_name, avg(salary) as avg_dept_salary
from instructor
group by dept_name
order by avg(salary) desc
```

Table 40: 7 records

dept_name	avg_dept_salary
Physics	91000.00
Finance	85000.00
Elec. Eng.	80000.00
Comp. Sci.	77333.33
Biology	72000.00
History	61000.00
Music	40000.00

- find number of instructors working in each department:

```
select dept_name, count(*) cnt
from instructor
group by dept_name
order by dept_name
```

Table 41: 7 records

dept_name	cnt
Biology	1
Comp. Sci.	3
Elec. Eng.	1
Finance	2
History	2
Music	1
Physics	2

- find the number of instructors in each department who teach a course in the Spring 2018 semester:

```
select i.dept_name, count(distinct i.id) as instr_count
from instructor i , teaches t
where i.id = t.instructor_id
and t.semester = 'Spring' and t.year = 2018
group by i.dept_name
```

Table 42: 4 records

dept_name	instr_count
Comp. Sci.	3
Finance	1
History	1
Music	1

> **Warning**
>
> Note that only attributes allowed to appear in the select clause (other than the attribute being aggregated) are the attributes used in the `group by` clause. Thus
>
> ```
> select a, X, avg(b) -- X doesn't appear in the group by clause below
> from r -- some relation r
> where P -- some predicate
> group by a
> ```
>
> it not legal.

Having Clause

Specifies a condition that applies to **groups** rather than to tuples.

- departments where average salayr is morethan $42,000:

```
select i.dept_name, avg(i.salary)
from instructor i
group by dept_name
having avg(i.salary) > 42000
order by avg(i.salary) desc
```

Table 43: 6 records

dept_name	avg
Physics	91000.00
Finance	85000.00
Elec. Eng.	80000.00
Comp. Sci.	77333.33
Biology	72000.00

dept_name	avg
History	61000.00

Like with `select` the attributes that are allowed in the `having` clause are either present in the `group by` clause, it is the attribute that is being aggregated.

Semantics of Group by and Having

Can be understood roughly as:

1. `from` is evaluated to get a relation
2. `where` predicate is applied on each tupel to get a new relation
3. tupels that agree on values of those attributes listed in the `group by` clause are placed into groups.
4. `having` clause applied to each group, the ones that satisfy it are retianed to obtain a new relation
5. `select` clause is applied to the relation to obtain the resulting relation.

A query with both `where` and `having`:

- for each course section offered in 2017, find the average total credits (`tot_cred`) of all students enrolled in the section, if the section has at least 2 students:

```
select t.course_id, t.sec_id, t.semester, avg(s.tot_cred)
from takes t, student s
where t.student_id = s.id
and t."year" = 2017
group by course_id, sec_id, semester
having count(s.id) > 1;
```

Table 44: 3 records

course_id	sec_id	semester	avg
CS-101	1	Fall	65
CS-190	2	Spring	43
CS-347	1	Fall	67

Aggregation with null

All aggregate functions except of `count(*)` ignore `null` values

Aggregation of Boolean Values

- the aggregate function `some()` can be applied to an aggregate consisting of boolean values to compute the disjunction of these values
- the aggregate function `every()` can be applied to an aggregate consisting of boolean values to compute the conjunction of the values.

```
select every(a)
from (
    values
    (true),
    (true)
) r(a)
```

Table 45: 1 records

every
TRUE