SQL – Part 2

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

- develop working knowledge of the SQL data definition language (DDL)
  - SQL data types
  - creating tables and constraints
- learn additional SQL data manipulation language (DML) concepts
  - insert, update, and delete rows
  - string pattern matching
  - inner and outer joins
  - views
  - auto-increment attributes
  - ~~stored procedures~~
  - ~~cursors~~
  - ~~triggers~~
- Textbook sections (6th ed.): 3.2, 3.6, 3.9, 4.1, 4.2, 4.4, ~~5.1, 5.3~~

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*)
- *department* = (*dept_name*, *building*, *budget*)
- …

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

SQL DDL

Data Definition Language

The SQL **data-definition language (DDL)** is used to specify of information about relations, including:

- the schema for each relation
- the domain of values associated with each attribute
- integrity constraints
- the set of indexes to be maintained for each relation
- the physical storage structure of each relation (e.g., choice of InnoDB vs. MyISAM storage engine in MySQL)

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character string, with user-specified maximum length *n.*
- **int.** Integer (a machine-dependent finite subset of the integers).
- **smallint.** Small integer (a machine-dependent subset of int).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* significant digits, with *d* 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.
- **date, time**. Calendar date (YYYY-MM-DD format), and time of day (hh:mm:ss format).

Create Table Construct

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

  create table r (A_1 D_1,
  ...,
  A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$)
)

  [Note: r is a relation, A_i is an attribute, and D_i is the domain of A_i]

- Example:

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

Drop and Alter Table Constructs

- **drop table** *student*
  - Deletes the table and its contents. Assumes table exists.
- **drop table if exists** *student*
  - Deletes the table and its contents if table exists.
- **delete from** *student*
  - Deletes all tuples from table, but retains table.
- **alter table**
  - **alter table** *r* **add** *A D*
    - Adds attribute with name *A* and domain *D* to relation *r.* All existing tuples in the relation are assigned *null* as the value for the new attribute.
  - **alter table** *r* **drop** *A*
    - Drops attribute *A* from relation *r*. Not supported by many databases.

Declaring Integrity Constraints

- **not null:** disallows null values
- **primary key** (A_1, ..., A_n): ensures uniqueness
- **unique** (A_1, ..., A_n): ensures uniqueness (think superkey)
- **foreign key** (A_m, ..., A_n) **references** r (A'_m, ..., A'_n): defines a foreign key in the child (referencing) table that points to a referenced key in a parent (referenced) table r
- **default** V: makes V the default value for an attribute
- Example:

```
create table instructor (
        ID              char(5),
        name            varchar(20) not null,
        dept_name       varchar(20),
        salary          numeric(8,2) default 0,
        primary key (ID),
        foreign key (dept_name) references department (dept_name)
    )
```

Primary Keys and Superkeys

- Example of **primary key** and **unique** constraints in SQL:

```
create table customer (
        customer_id                     int,
        social_insurance_num            numeric(9,0),
        first_name                      varchar(20),
         last_name                      varchar(20),
         primary key (customer_id),
         unique (social_insurance_num)
)
```

- Note 1: **primary key** and **unique** both identify superkeys. You can use **primary key** at most once per table but you can use **unique** more than once.
- Note 2: A primary key attribute cannot be null. The **primary key** constraint on an attribute implies the **not null** constraint.
- Note 3: For a **unique** attribute, the number of tuples that may have a *null* value is system-dependent (zero, one, or many).

Foreign Keys

- Example of a **foreign key** (a.k.a. **referential integrity**) constraint:

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

create table *department* (
dept_name **varchar**(20) **primary key**,
building **varchar**(20) **not null**,
budget **int**
)

- Note: In MySQL the referenced key (in this case *dept_name*) must be a superkey of the referenced table, or a prefix of a multi-attribute primary key of the referenced table.

Referential Actions

- **Referential actions** define the behavior of the DB in cases when an update or deletion on the parent (i.e., referenced) table SQL statement affects a value referenced by a child (i.e., referencing) table.
- SQL 92 defines four actions:
  - **cascade**: automatically update or delete foreign key in matching rows of child table.
  - **set null**: set foreign key columns to *null* in matching rows of child table. (Assumes foreign key is nullable.)
  - **set default**: set foreign key columns to the default value in matching rows of child table.
  - **no action**: reject operation and generate error. (Also known as **restrict** in MySQL.)

Referential Actions (Cont.)

- Example: cascade on update, set null on delete

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

SQL DML

Modifying Relations – Insertion

- Example: Add a new tuple to *course.*

  insert into *course*
  values ('ECE-356', 'Databases', 'ECE', 0.5)

- Example: Add a new tuple to *course* with *credits* set to *null*.

  insert into *course*
  values ('ECE-356', 'Databases', 'ECE', *null*)

Modifying Relations – Updates

- Example: Give a 3% salary increase to all instructors whose salary is below $80,000.

update *instructor*
    set *salary* = *salary* * 1.03
    where *salary* < 80000

Modifying Relations – Deletion

- Example: Delete all instructors.

  delete from *instructor*

- Example: Delete all instructors from the Math department.

  delete from *instructor*
  where *dept_name*= 'Math'

- Example: Delete all instructors whose departments are in the EIT building.

  delete from *instructor*
  where *dept_name* **in** (**select** *dept_name*
  from *department*
  where *building* = 'EIT')

String Pattern Matching

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

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

- Example: Match the string “100 %”.

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

- Note: Patterns are case-sensitive.

More Joins

- **Join operations** take two relations and return another relation as a result.
- **Join condition:** defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type:** defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

More Joins (Cont.)

- Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  prereq information is missing for CS-315 and

  course information is missing for CS-437

Inner and Outer Joins

- **select** * **from** *course* **inner join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

Note: some databases allow you to omit the keyword **"inner"**.

- **select** * **from** *course* **left outer join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	*null*	*null*

Outer Joins

- **select** * **from** *course* **natural right outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

- **select** * **from** *course* **full outer join** *prereq* **using** (*course_id*)

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Note: full outer join is not supported in MySQL 5.0.

Views

- In some cases, it is not desirable for all users to see all the relations stored in a database instance.
- Consider a person who needs to know an instructor's name and department, but not the salary. This person should see a relation described in SQL by the following query:

  select *ID*, *name*, *dept_name*
  from *instructor*

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not part of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

View Definition

- A view is defined using the **create view** statement, which follows the form

  create view *view_name* **as** < query expression >

  where <query expression> is any legal SQL query.
- Once a view is defined, the view name can be used to refer to the virtual relation.
- A view need not be defined over a single table. It can be defined over the result set of a join or group by query.
- Note: Views are dynamic. In other words, changing the data in the relations referenced by a view causes analogous changes in the virtual relation corresponding to the view.

Example Views

- A view of instructors without their salary

  create view *faculty* **as**
  select *ID*, *name*, *dept_name*
  from *instructor*

- Find all instructors in the Biology department

  select *name*
  from *faculty*
  where *dept_name* = 'Biology'

- Create a view of department salary totals

  create view *departments_total_salary*(*dept_name*, *total_salary*) **as**
  select *dept_name*, **sum**(*salary*)
  from *instructor*
  group by *dept_name*

  select * **from** *departments_total_salary*

Auto-increment Attributes

- Auto-increment attributes can be used to automatically generate primary key values.
- Example:

```
create table instructor_auto (
        ID          int auto_increment primary key,
        name        varchar(20),
        dept_name           varchar(20),
        salary      numeric(8,2)
)

insert into instructor_auto (name, dept_name, salary)
values ('Watson', 'Biology', 90210)

select * from instructor_auto
```

- Note 1: *ID* = 1 should be assigned automatically to the first tuple.
- Note 2: The auto-increment type does not exist in the ER model and therefore it does not eliminate the need for weak entity sets.