Chapter 4

Intermediate SQL

Slides by Silbershatz, Modifications by Rogers and Brown

Chapter 4 : Intermediate SQL

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

Joined Relations

- Join operations take two relations and return as a result another relation
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join.
- The join operations are typically used as subquery expressions in the from clause

Join Operations – Example

- 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

- Note: prereq information missing for CS-315 and course information missing for CS-437

Outer Join

- An extension of the join operation that avoids loss of information
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join
- Uses null values
- Has three (3) options for how you want to combine the data

Join #1 – Left Outer Join

- **`course natural left outer join prereq`**

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

Note : read prere_id as prereq_id

Join #2 – Right Outer Join

- **course natural right outer join prereq**

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

Join #3 – Full Outer Join

- **course natural full outer join prereq**

course_id	title	dept_name	credits	prere_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

Joined Relations

- Join operations take two relations and return as a result another relation
- These additional operations are typically used as subquery expressions in the from clause
- 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_2, .., A_n)

Joined Relations – Examples

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

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

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

course_id	*title*	*dept_name*	*credits*	*prere_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*

Joined Relations – Examples

- **course natural right outer join prereq**

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

- **course right outer join prereq using (course_id)**

course_id	title	dept_name	credits	prere_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

Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database)
- Consider a person who needs to know an instructors' name and department but not salary. This person should see a relation described, in SQL, by

```
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 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 has the form:

```
create view v as <query expression>
```

  where <query expression> is any legal SQL expression. The view name is represented by v
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

Example Views

- A view of instructors without their salary

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

- Now, 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;
```

Views Defined Using Other Views

```
• create view physics_fall_2009 as
        select course.course_id, sec_id, building,
             room_number
        from course, section
        where course.course_id = section.course_id
             and course.dept_name = 'Physics'
             and section.semester = 'Fall'
             and section.year = '2009';
• create view physics_fall_2009_watson as
        select course_id, room_number
        from physics_fall_2009
        where building = "Watson";
```

View Expansion

- A way to define the meaning of views defined in terms of other views
- Let view v1 be defined by an expression e1 that may itself contain uses of view relations
- View expansion of an expression repeats the following replacement step:

  repeat

    Find any view relation v_i in e_1

    Replace the view relation v_i by the expression defining v_i

  until no more view relations are present in e_1

- As long as the view definitions are not recursive, this loop will terminate

View Expansion

- Expand use of a view in a query/another view

```
create view physics_fall_2009_watson as
    (select course_id, room_number
     from (select course.course_id, building,
     room_number
          from course, section
          where course.course_id = section.course_id
               and course.dept_name = 'Physics'
               and section.semester = 'Fall'
               and section.year = '2009')
     where building = 'Watson')
```

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to depend directly on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to depend on view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be recursive if it depends on itself

Update of a View

- Add a new tuple to faculty view (which we defined earlier)

```
insert into faculty values ('30765',
    'Green', 'Music');
```

this insertion must be represented by the insertion of the tuple

('30765', 'Green', 'Music', null)

into the instructor relation

Some Updates Cannot be Translated Uniquely…

-
```
create view instructor_info as
        select ID, name, building
        from instructor, department
        where instructor.dept_name =
              department.dept_name;
```
-
```
insert into instructor_info values ('69987', 'White',
    'Taylor')
```
  - Which department, if multiple departments are in Taylor?
  - What if NO department is in Taylor?
- Most SQL implementations allow updates only on simple views
  - The from clause has only one database relation
  - The select clause contains only attribute names of the relation, and does not have any expressions, aggregates or distinct specifications
  - Any attribute not list in the select clause can be set to null
  - The query does not have a group by or having clause

…And Some Not at All

```
create view history_instructors as
        select *
        from instructor
        where dept_name = 'History';
Insert ('25566', 'Brown', 'Biology', 100000)
        into history_instructors
```

Transactions

- Unit of Work
- Atomic Transaction
  - Either fully executed or rolled back as if it never occurred
- Isolation from Concurrent Transactions
- Transactions begin implicitly
  - Ended by **`commit work`** or **`rollback work`**
- But default on most databases : each SQL statement commits automatically
  - Can turn off auto commit for a session (e.g. using API)
  - In SQL:1999 can use: **`begin atomic…. endl`**

Integrity Constraints

- Integrity Constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency
  - A Checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $14.00 an hour
  - A customer must have a (non-null) phone number

Constraints on a Single Relation

- not null
- primary key
- unique
- check (P)
  - P is a predicate
    - Quick check : what is a predicate?

Not Null and Unique Constraints

- not null
  - Declare name and budget to be not null

```
name      varchar (20) not null
budget    numeric (12, 2) not null
```

- unique (A_1, A_2, A_3, … , A_n)
  - The unique specification states that the attributes A_1, A_2, …, A_n form a candidate key
  - Candidate keys are permitted to be null (in contrast to primary keys)

The check Clause

- check (P)
  - P is a predicate
- Example : ensure that semester is one of Fall, Winter, Spring or Summer

```
create table section (
        course_id           varchar (8),
        sec_id              varchar (8),
        semester            varchar (6),
        year                numeric (4,0),
        building            varchar (15),
        room_number         varchar (7),
        time_slot_id        varchar (4),
        primary key (course_id, sec_id, semester,
                     year),
        check (semester in ('Fall', 'Winter',
                            'Spring', 'Summer'))
);
```

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation
  - Example : If 'Biology' is a department name appearing in one of the tuples in the instructor relation, then there exists a tuple in the department relation for 'Biology'
- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a foreign key of R if for any values of A appearing in R these values also appear in S

Cascading Actions in Referential Integrity

- ```
  create table course (
      course_id      char (5), primary key
      title          varchar (20)
      dept_name      varchar (20) references
                     department);
  ```
- ```
  create table course (
      …
      dept_name varchar (20),
      foreign key (dept_name) references
                department
                on delete cascade
                on update cascade,
          …);
  ```
- alternative actions to cascade: set null, set default

Integrity Constraint Violation During Transactions

- E.g.,

```
create table person (
        ID          char (10),
        name        char (40),
        mother      char (10),
        father      char (10),
        primary key ID,
        foreign key father references person,
        foreign key mother references person);
```

- How to insert a tuple?
- What if mother or father is declared not null?
  - `constraint father_ref foreign key father references person, constraint mother_ref foreign key mother references person)`
  - `set constraints father_ref, mother_ref deferred`

Complex Check Clauses

- **`check (time_slot_id in (select time_slot_id from time_slot))`**
  - why not use a foreign key here?
- Every section has at least one instructor teaching the section
  - How do I write this?
- Unfortunately : subquery in check clause not supported by pretty much any database
  - Alternative : triggers (later)
- create assertion <assertion-name> check <predicate>
  - Also not supported by anyone

Built-in Data Types in SQL

- date
  - Dates, containing a (4 digit) year, month and date
  - Example : date '2005-7-28'
- time
  - Time of day, in hours, minutes and seconds
  - Example : time '09:13:03' time '08:15:28.74'
- timestamp
  - date plus time of day
  - Example : timestamp '2005-7-28 08:15:28.74'
- interval
  - Example : interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

Other Features

- ```
  create table student(
      ID          varchar (5) primary key,
      name        varchar (20) not null,
      dept_name   varchar (20),
      tot_cred    numeric (3,0) default 0)
  ```
- ```
  create index studentID_index on student(ID)
  ```
- Large objects
  - book review clob (10KB)
  - image blob (10 MB)
  - movie blob (2 GB)

Large-Object Types

- Large objects (photos, videos, CAD Files, etc) are stored as a large object:
  - blob
    - Binary large object : object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - clob
    - Character large object : object is a large collection of character data
- When a query returns a large object, a pointer is returned rather than the large object itself

User-Defined Types

- create type construct in SQL creates user-defined type

```
create type Dollars as numeric (12, 2) final
create table department(
        dept_name       varchar (20),
        building        varchar (15),
        budget          Dollars);
```

Domains

- create domain construct in SQL-92
  - Creates user defined domain types

```
create domain person_name char (20) not null
```

- Types and domains are similar.
  - Domains can have constraints such as not null specified on them

```
create domain degree_level varchar (10)
constraint degree_level_test
check (value in ('Bachelors', 'Masters',
                 'Doctorate'));
```

Authorization

- Forms of authorization on parts of the database :
  - Read : allows reading, but not modification of data
  - Insert : allows insertion of new data, but not modification of existing data
  - Update : allows modification, but not deletion of data
  - Delete : allows deletion of data
- Forms of authorization to modify the database schema :
  - Index : allows creation and deletion of indices
  - Resources : allows creation of new relations
  - Alteration : allows addition or deletion of attributes in a relation
  - Drop : allows deletion of relations

Authorization Specification in SQL

- The grant statement is used to confer authorization
  grant <privilege list>
  on <relation name or view name> to <user list>
- <user list> is :
  - a user-id
  - public, which allows all valid users the privilege granted
  - a role (more on this later)
- Granting a privilege on a view does not imply granting any privilege on the underlying relations
- The grantor of the privilege must already hold the privilege on the specified item (or be the DBA)

Privileges in SQL

- select : allows read access to relation, or the ability to query using the view
  - Example : grant users U_1, U_2 and U_3 select authorization on the branch relation

```
grant select on instructor to U1, U2, U3
```

- insert : the ability to insert tuples
- update : the ability to update using the SQL update statement
- delete : the ability to delete tuples
- all privileges : used as a short form for all the allowable privileges

Revoking Authorization in SQL

- The revoke statement is used to revoke authorization
  revoke <privilege list>
  on <relation name or view name> from <user list>
- Example :
  revoke select on *branch* from U_1, U_2, U_3
- <privilege list> may be all to revoke all privileges the revokee may hold
- If <user list> includes public, all users lose the privilege except those granted in explicitly
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation
- All privileges that depend on the privilege being revoked are also revoked

Roles

- create role instructor
- Privileges can be granted to roles:
  - grant select on takes to instructor,
- Roles can be granted to users, as well as to other roles
  - create roll student;
  - grant instructor to Amit;
  - create role dean;
  - grant instructor to dean;
  - grant dean to Satoshi;

Authorization on Views

- ```
  create view geo_instructor as(
      select *
      from instructor
      where dept_name = 'Geology');
  ```
- ```
  grant select on geo_instructor to staff
  ```
- Suppose that a staff member issues
  ```
  select *
  from geo_instructor
  ```
- What if
  - Staff does not have permissions on instructor
  - Creator of the view did not have some permissions on instructor?

Other Authorization Features

- references privilege to create foreign key
  - ```
    grant reference (dept_name) on department to Mariano;
    ```
  - Why is this required?
- transfer of privileges
  - ```
    grant select on department to Amit with grant option;
    ```
  - ```
    revoke select on department from Amit, Satoshi cascade;
    ```
  - ```
    revoke select on department from Amit, Satoshi restrict;
    ```
- Read 4.6 for others

End of Chapter 4