Chapter 4: Intermediate SQL
# 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.

<table>
<thead>
<tr>
<th>Join types</th>
<th>Join conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>inner join</td>
<td>natural</td>
</tr>
<tr>
<td>left outer join</td>
<td>on <code>&lt;predicate&gt;</code></td>
</tr>
<tr>
<td>right outer join</td>
<td>using ((A_1, A_2, \ldots, A_n))</td>
</tr>
<tr>
<td>full outer join</td>
<td></td>
</tr>
</tbody>
</table>
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.
Left Outer Join

Relation `course`

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

Relation `prereq`

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- `select *`
- `from course natural left outer join prereq`
Left Outer Join Queries

Relation course

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

Relation prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

- select *
  from course natural left outer join prereq

  = select course.course_id, title, dept_name, credits, prereq_id
  from course left outer join prereq
    on course.course_id = prereq.course_id

  = select *
  from course left outer join prereq using (course_id)
Right Outer Join

Relation course

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

Relation prereq

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

\[
\text{select } * \quad \text{from course natural right outer join prereq}
\]

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>CS-101</td>
</tr>
</tbody>
</table>
Full Outer Join

Relation `course`

<table>
<thead>
<tr>
<th>course_id</th>
<th>title</th>
<th>dept_name</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>Biology</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>Comp. Sci.</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>Comp. Sci.</td>
<td>3</td>
</tr>
</tbody>
</table>

Relation `prereq`

<table>
<thead>
<tr>
<th>course_id</th>
<th>prereq_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>BIO-101</td>
</tr>
<tr>
<td>CS-190</td>
<td>CS-101</td>
</tr>
<tr>
<td>CS-347</td>
<td>CS-101</td>
</tr>
</tbody>
</table>

```sql
*  
from `course` natural full outer join `prereq`
```
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 the salary. This person should see a relation described, in SQL, by

```sql
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.
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

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

- Find all instructors in the Biology department
  ```sql
  select name
  from faculty
  where dept_name = 'Biology'
  ```

- Create a view of department salary totals
  ```sql
  create view departments_total_salary(dept_name, total_salary) as
  select dept_name, sum(salary)
  from instructor
  group by dept_name;
  ```
Materialized Views

- When defining a view, simply create a physical table representing the view at the time of creation.
- Update is simple to handle.
- How are updates handled to the “base” relations on which the view was defined?
Integrity Constraints

- Integrity constraints guard against accidental damage to the database
  - Ensure that authorized changes to the database do not result in a loss of data consistency
- Examples
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $4.00 an hour
  - A customer must have a (non-null) phone number
Integrity Constraints on a Single Relation

- not null
- unique
- primary key
- check \( (P) \), where \( P \) is a predicate
Not Null and Unique Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**
  
  \[
  \text{name \texttt{varchar}(20) \texttt{not null}}
  \]
  
  \[
  \text{budget \texttt{numeric}(12,2) \texttt{not null}}
  \]

- **unique** \((A_1, A_2, \ldots, A_m)\)
  - The unique specification states that the attributes \(A_1, A_2, \ldots, A_m\) form a candidate key.
  - Candidate keys are permitted to be null (in contrast to primary keys).

- **primary key** \((A_1, A_2, \ldots, A_m)\)
  - **not null** + **unique**
The check clause

- **check** \((P)\), where \(P\) is a predicate

Example: ensure that semester value is one of fall, winter, spring or summer:

```sql
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),
title varchar(20),
department varchar(20),
primary key (course_id)
foreign key (dept_name) references department)`

- `create table course (  
...  
department varchar(20),
foreign key (dept_name) references department  
on delete cascade  
on update cascade,
...  
)`

- alternative actions to cascade: set null, set default
### Additional Built-in Data Types in SQL

- **date**: Dates, containing a (4 digit) year, month and date
  - Example: `date '2005-7-27'`

- **time**: Time of day, in hours, minutes and seconds.
  - Example: `time '09:00:30'  time '09:00:30.75'`

- **timestamp**: date plus time of day
  - Example: `timestamp '2005-7-27 09:00:30.75'`

- **interval**: period of time
  - 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
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.
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
  
  ```sql
  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 privileges on the underlying relations.

- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).
Privileges in SQL

- **select**: allows read access to relation, or to the view
  - Example: grant users $U_1$, $U_2$, and $U_3$ the **select** authorization on the *instructor* relation:
    
    ```sql
    grant select on instructor to U_1, U_2, U_3
    ```

- **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

- **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
Revoking Authorization in SQL

- The `revoke` statement is used to revoke authorization.
  
  ```sql
  revoke <privilege list>
  on <relation name or view name> from <user list>
  ```

- Example:
  ```sql
  revoke select on branch from U1, U2, U3
  ```

- `<privilege-list>` may be `all` to revoke all privileges the revokee may hold.

- All privileges that depend on the privilege being revoked are also revoked.

- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
Authorization-Grant Graph

Diagram showing the relationship between DBA and users U1, U2, U3, U4, and U5.
Roles

- **create role instructor;**
  - grant instructor to Amit;
- 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 role teaching_assistant;
  - grant teaching_assistant to instructor;
    - instructor inherits all privileges of teaching_assistant
- Chain of Roles
  - create role dean;
  - grant instructor to dean;
  - grant dean to Satoshi;
Transfer of Privileges

- 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;
End of Chapter 4