Chapter 10
Advanced Topics in Relational Databases
Security and User Authorization in SQL
Database users have different privileges: read, write and execute

Nine types of privileges:
1. SELECT
2. INSERT
3. DELETE
4. UPDATE
5. REFERENCES
6. USAGE
7. TRIGGER
8. EXECUTE
9. UNDER
Nine types of privileges:

1. SELECT
2. INSERT
3. DELETE
4. UPDATE
5. REFERENCES
6. USAGE
7. TRIGGER
8. EXECUTE
9. UNDER

apply to a relation
(or some attributes of a relation)
Nine types of privileges:
1. SELECT
2. INSERT
3. DELETE
4. UPDATE
5. REFERENCES
6. USAGE
7. TRIGGER
8. EXECUTE
9. UNDER

The REFERENCES privilege on a relation is the right to refer to that relation in a constraint.
Nine types of privileges:
1. SELECT
2. INSERT
3. DELETE
4. UPDATE
5. REFERENCES
6. USAGE
7. TRIGGER
8. EXECUTE
9. UNDER

The right to use schema elements in one’s declarations. (We did not discuss them in class.)
Nine types of privileges:

1. SELECT
2. INSERT
3. DELETE
4. UPDATE
5. REFERENCES
6. USAGE
7. TRIGGER
8. EXECUTE
9. UNDER

The TRIGGER privilege on a relation is the right to define triggers on that relation.
Nine types of privileges:
1. SELECT
2. INSERT
3. DELETE
4. UPDATE
5. REFERENCES
6. USAGE
7. TRIGGER
8. EXECUTE
9. UNDER

EXECUTE is the right to execute a piece of code, such as a PSM procedure or function. (We did not discuss this in class.)
Nine types of privileges:
1. SELECT
2. INSERT
3. DELETE
4. UPDATE
5. REFERENCES
6. USAGE
7. TRIGGER
8. EXECUTE
9. UNDER

UNDER is the right to create subtypes of a type. (We did not discuss this in class.)
Example:  INSERT INTO Studio (name)
   SELECT DISTINCT studioName
   FROM Movies
   WHERE studioName NOT IN
   (SELECT name
    FROM Studio);

For the above SQL statement, we need privileges:
1) INSERT privilege on Studio(name)
2) SELECT privilege on Movies(studioName) and Studio(name)
Recursion in SQL
The **WITH** statement in SQL allows us to define temporary relations, recursive or not.

A simple form of **WITH** statement:

```
WITH R AS <definition of R> <query involving R>
```

The temporary relation is not available outside the query that is part of the **WITH** statement.
A more general form of WITH statement, with possible recursion:

WITH

[RECURSIVE] R1 AS <definition of R1>,
[RECURSIVE] R2 AS <definition of R2>,
......
[RECURSIVE] Rn AS <definition of Rn>

<query involving R1, R2, ..., Rn>

If a relation Ri is defined recursively (in terms of itself or other relations here), it must be preceded by the keyword RECURSIVE.
Example:

**Flights (airline, from, to, departure time, arrival time)**

<table>
<thead>
<tr>
<th>airline</th>
<th>from</th>
<th>to</th>
<th>departure time</th>
<th>arrival time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UA</td>
<td>SF</td>
<td>DEN</td>
<td>9:30</td>
<td>12:30</td>
</tr>
<tr>
<td>AA</td>
<td>SF</td>
<td>DAL</td>
<td>9:00</td>
<td>14:30</td>
</tr>
<tr>
<td>UA</td>
<td>DEN</td>
<td>CHI</td>
<td>15:00</td>
<td>18:00</td>
</tr>
<tr>
<td>UA</td>
<td>DEN</td>
<td>DAL</td>
<td>14:00</td>
<td>17:00</td>
</tr>
<tr>
<td>AA</td>
<td>DAL</td>
<td>CHI</td>
<td>15:30</td>
<td>17:30</td>
</tr>
<tr>
<td>AA</td>
<td>DAL</td>
<td>NY</td>
<td>15:00</td>
<td>19:30</td>
</tr>
<tr>
<td>AA</td>
<td>CHI</td>
<td>NY</td>
<td>19:00</td>
<td>22:00</td>
</tr>
<tr>
<td>UA</td>
<td>CHI</td>
<td>NY</td>
<td>18:30</td>
<td>21:30</td>
</tr>
</tbody>
</table>

**Question:** For what pairs of cities \((x,y)\) is it possible to get from city \(x\) to city \(y\) by taking one or more flights?
Problem: Find all pairs of cities \((x,y)\) such that there is a directed path from \(x\) to \(y\).
We want to produce a relation

**Reaches (from, to)**

where every tuple \((x,y)\) means there is a directed path from city \(x\) to city \(y\).

Basic rules:

1. If there is a tuple \((a,x,y,d,r)\) in “Flights”, then \(x\) can reach \(y\) in graph, so \((x,y)\) should be a tuple in “Reaches”.

2. If there is a tuple \((x,y)\) and a tuple \((y,z)\) in “Reaches”, then the tuple \((x,z)\) should also be in “Reaches”. 
SQL solution:

WITH RECURSIVE Reaches(from, to) AS
    (SELECT from, to FROM Flights)
UNION
    (SELECT R1.from, R2.to
     FROM Reaches R1, Reaches R2
     WHERE R1.to = R2.from)
SELECT * FROM Reaches
Object-Relational Model
Relational databases have incorporated many ideas in object-oriented databases.

As a result, many DBMS products that used to be called “relational” are now called “object-relational”. They have features such as:

1. Structured types for attributes.
2. Methods
3. Identifiers for tuples
4. References (pointers)
## Nested Relations

Stars(name, address(street, city), birthdate, movies(title, year, length))

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>birthdate</th>
<th>movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher</td>
<td></td>
<td>9/9/99</td>
<td>Star Wars 1977</td>
</tr>
<tr>
<td></td>
<td>street</td>
<td></td>
<td>Empire 1980</td>
</tr>
<tr>
<td></td>
<td>city</td>
<td></td>
<td>Return 1983</td>
</tr>
<tr>
<td>Hamill</td>
<td></td>
<td>8/8/88</td>
<td>Star Wars 1977</td>
</tr>
<tr>
<td></td>
<td>street</td>
<td></td>
<td>Empire 1980</td>
</tr>
<tr>
<td></td>
<td>city</td>
<td></td>
<td>Return 1983</td>
</tr>
<tr>
<td></td>
<td>street</td>
<td></td>
<td>Oak</td>
</tr>
<tr>
<td></td>
<td>city</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## References

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>birthdate</th>
<th>movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher</td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maple</td>
<td>H’wood</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Locust</td>
<td>Malibu</td>
<td></td>
</tr>
<tr>
<td>Hamill</td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oak</td>
<td>B’wood</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>title</th>
<th>year</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>124</td>
</tr>
<tr>
<td>Empire</td>
<td>1980</td>
<td>127</td>
</tr>
<tr>
<td>Return</td>
<td>1983</td>
<td>133</td>
</tr>
</tbody>
</table>

**Figure 10.16:** Sets of references as the value of an attribute
User-Defined Types in SQL

Data types used in C programming
Example:

In relation

Movies(title, year, length, producerCert#)

although “length” and “producerCert#” are both integers, it does not make sense to compare them. To prevent any attempt to compare them, we can define them as two different types:

```
CREATE TYPE CertType AS INTEGER;
CREATE TYPE LengthType AS INTEGER;
```
More generally, a type can be defined as a list of attributes (like a relation).

Example:

```sql
CREATE TYPE AddressType AS (  
    Street CHAR(50),  
    City CHAR(20)  
);

CREATE TYPE StarType AS (  
    name CHAR(30),  
    address AddressType  
);
```
Following references

Example: Given a relation

    StarsIn(star, movie)

where “star” has the type “StarType”, to get all the movies that Brad Pitt was in, we can use the query:

    SELECT movie
    FROM   StarsIn
    WHERE star->name = ‘Brad Pitt’;