



اصول طراحی پایگاه داده

By Dr. Taghinezhad

Mail:

a0taghinezhad@gmail.com

SEVENTH EDITION

Database System Concepts





Chapter 4 : Intermediate SQL

Database System Concepts, 7th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Outline

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Index Definition in SQL
- 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
- Three types of joins:
 - Natural join
 - Inner join
 - Outer join



Natural Join in SQL

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column.
- List the names of instructors along with the course ID of the courses that they taught
 - **select** *name, course_id*
from *students, takes*
where *student.ID = takes.ID;*
- Same query in SQL with “natural join” construct
 - **select** *name, course_id*
from *student natural join takes;*



Natural Join in SQL (Cont.)

- The **from** clause can have multiple relations combined using natural join:

```
select A1, A2, ... An  
from r1 natural join r2 natural join .. natural join rn  
where P;
```



Student and Takes Relation Relation

| ID | name | dept_name | tot_cred |
|-------|----------|------------|----------|
| 00128 | Zhang | Comp. Sci. | 102 |
| 12345 | Shankar | Comp. Sci. | 32 |
| 19991 | Brandt | History | 80 |
| 23121 | Chavez | Finance | 110 |
| 44553 | Peltier | Physics | 56 |
| 45678 | Levy | Physics | 46 |
| 54321 | Williams | Comp. Sci. | 54 |
| 55739 | Sanchez | Music | 38 |
| 70557 | Snow | Physics | 0 |
| 76543 | Brown | Comp. Sci. | 58 |
| 76653 | Aoi | Elec. Eng. | 60 |
| 98765 | Bourikas | Elec. Eng. | 98 |
| 98988 | Tanaka | Biology | 120 |

| ID | course_id | sec_id | semester | year | grad |
|-------|-----------|--------|----------|------|------|
| 00128 | CS-101 | 1 | Fall | 2017 | A |
| 00128 | CS-347 | 1 | Fall | 2017 | A- |
| 12345 | CS-101 | 1 | Fall | 2017 | C |
| 12345 | CS-190 | 2 | Spring | 2017 | A |
| 12345 | CS-315 | 1 | Spring | 2018 | A |
| 12345 | CS-347 | 1 | Fall | 2017 | A |
| 19991 | HIS-351 | 1 | Spring | 2018 | B |
| 23121 | FIN-201 | 1 | Spring | 2018 | C+ |
| 44553 | PHY-101 | 1 | Fall | 2017 | B- |
| 45678 | CS-101 | 1 | Fall | 2017 | F |
| 45678 | CS-101 | 1 | Spring | 2018 | B+ |
| 45678 | CS-319 | 1 | Spring | 2018 | B |
| 54321 | CS-101 | 1 | Fall | 2017 | A- |
| 54321 | CS-190 | 2 | Spring | 2017 | B+ |
| 55739 | MU-199 | 1 | Spring | 2018 | A- |
| 76543 | CS-101 | 1 | Fall | 2017 | A |
| 76543 | CS-319 | 2 | Spring | 2018 | A |
| 76653 | EE-181 | 1 | Spring | 2017 | C |
| 98765 | CS-101 | 1 | Fall | 2017 | C- |
| 98765 | CS-315 | 1 | Spring | 2018 | B |
| 98988 | BIO-101 | 1 | Summer | 2017 | A |
| 98988 | BIO-301 | 1 | Summer | 2018 | nu |



student natural join takes

| <i>ID</i> | <i>name</i> | <i>dept_name</i> | <i>tot_cred</i> | <i>course_id</i> | <i>sec_id</i> | <i>semester</i> | <i>year</i> | <i>grade</i> |
|-----------|-------------|------------------|-----------------|------------------|---------------|-----------------|-------------|--------------|
| 00128 | Zhang | Comp. Sci. | 102 | CS-101 | 1 | Fall | 2017 | A |
| 00128 | Zhang | Comp. Sci. | 102 | CS-347 | 1 | Fall | 2017 | A- |
| 12345 | Shankar | Comp. Sci. | 32 | CS-101 | 1 | Fall | 2017 | C |
| 12345 | Shankar | Comp. Sci. | 32 | CS-190 | 2 | Spring | 2017 | A |
| 12345 | Shankar | Comp. Sci. | 32 | CS-315 | 1 | Spring | 2018 | A |
| 12345 | Shankar | Comp. Sci. | 32 | CS-347 | 1 | Fall | 2017 | A |
| 19991 | Brandt | History | 80 | HIS-351 | 1 | Spring | 2018 | B |
| 23121 | Chavez | Finance | 110 | FIN-201 | 1 | Spring | 2018 | C+ |
| 44553 | Peltier | Physics | 56 | PHY-101 | 1 | Fall | 2017 | B- |
| 45678 | Levy | Physics | 46 | CS-101 | 1 | Fall | 2017 | F |
| 45678 | Levy | Physics | 46 | CS-101 | 1 | Spring | 2018 | B+ |
| 45678 | Levy | Physics | 46 | CS-319 | 1 | Spring | 2018 | B |
| 54321 | Williams | Comp. Sci. | 54 | CS-101 | 1 | Fall | 2017 | A- |
| 54321 | Williams | Comp. Sci. | 54 | CS-190 | 2 | Spring | 2017 | B+ |
| 55739 | Sanchez | Music | 38 | MU-199 | 1 | Spring | 2018 | A- |
| 76543 | Brown | Comp. Sci. | 58 | CS-101 | 1 | Fall | 2017 | A |
| 76543 | Brown | Comp. Sci. | 58 | CS-319 | 2 | Spring | 2018 | A |
| 76653 | Aoi | Elec. Eng. | 60 | EE-181 | 1 | Spring | 2017 | C |
| 98765 | Bourikas | Elec. Eng. | 98 | CS-101 | 1 | Fall | 2017 | C- |
| 98765 | Bourikas | Elec. Eng. | 98 | CS-315 | 1 | Spring | 2018 | B |
| 98988 | Tanaka | Biology | 120 | BIO-101 | 1 | Summer | 2017 | A |
| 98988 | Tanaka | Biology | 120 | BIO-301 | 1 | Summer | 2018 | <i>null</i> |



Dangerous in Natural Join

- Beware of unrelated attributes with same name which get equated incorrectly
- Example -- List the names of students instructors along with the titles of courses that they have taken
 - Correct version

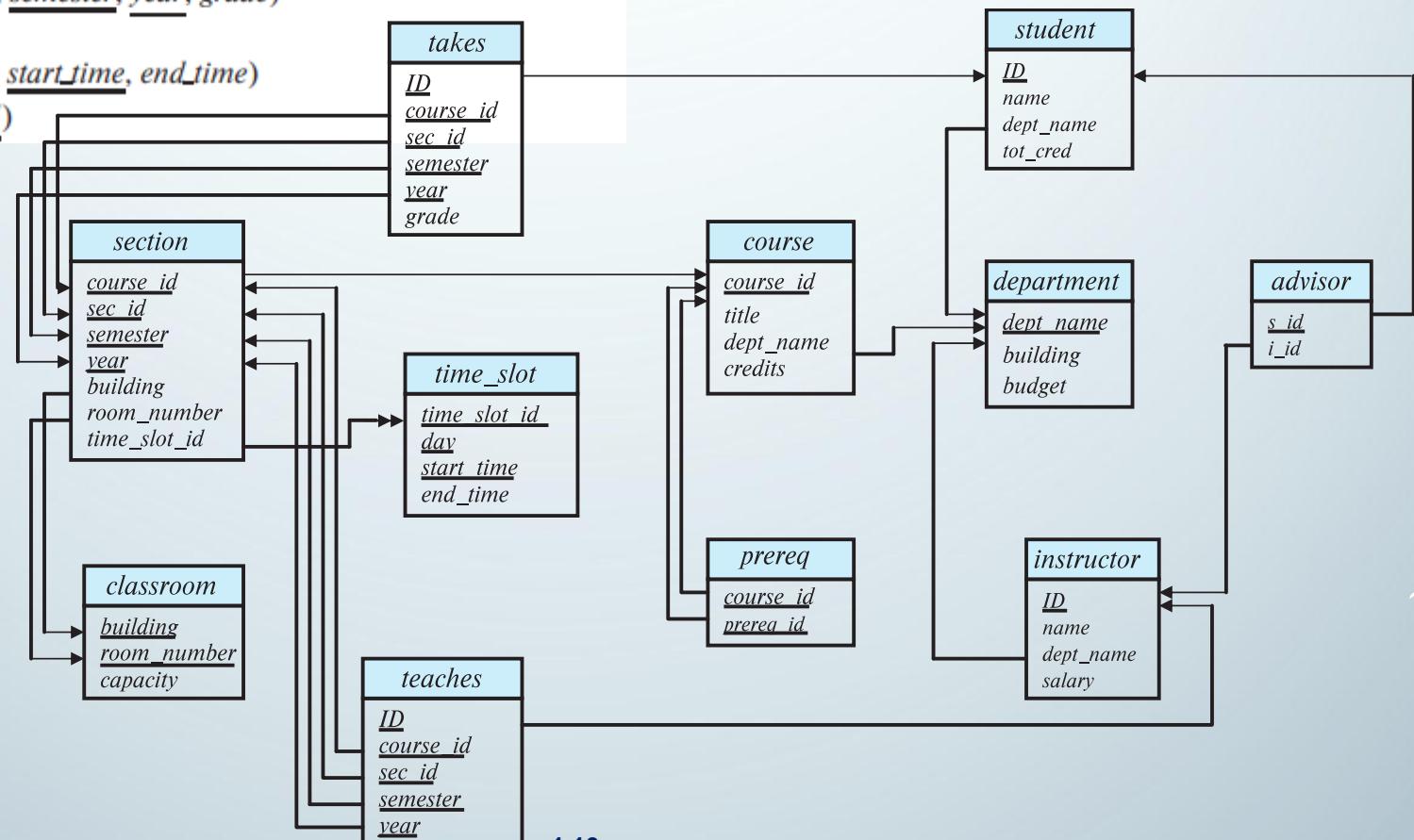
```
select name, title  
from student natural join takes, course  
where takes.course_id = course.course_id;
```

- Incorrect version
 - This query omits all (student name, course title) pairs where the student takes a course in a department other than the student's own department.
 - The correct version (above), correctly outputs such pairs.



Schema Diagram for University Database

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)





Natural Join with Using Clause

- To avoid the danger of equating attributes erroneously, we can use the “**using**” construct that allows us to specify exactly which columns should be equated.
- Query example

```
select name, title  
from (student natural join takes) join course using  
(course_id)
```



Join Condition

- The **on** condition allows a general predicate over the relations being joined
- This predicate is written like a **where** clause predicate except for the use of the keyword **on**
- Query example

```
select *  
from student join takes on student_ID = takes_ID
```

- The **on** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal.
- Equivalent to:

```
select *  
from student , takes  
where student_ID = takes_ID
```



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.
- Three forms of outer join:
 - left outer join
 - right outer join
 - full outer join



Outer Join Examples

- Relation *course*

| <i>course_id</i> | <i>title</i> | <i>dept_name</i> | <i>credits</i> |
|------------------|--------------|------------------|----------------|
| BIO-301 | Genetics | Biology | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |

- Relation *prereq*

| <i>course_id</i> | <i>prereq_id</i> |
|------------------|------------------|
| BIO-301 | BIO-101 |
| CS-190 | CS-101 |
| CS-347 | CS-101 |

- Observe that

course information is missing CS-347

prereq information is missing CS-315



Left Outer Join

- *course natural left outer join prereq*

| <i>course_id</i> | <i>title</i> | <i>dept_name</i> | <i>credits</i> | <i>prereq_id</i> |
|------------------|--------------|------------------|----------------|------------------|
| BIO-301 | Genetics | Biology | 4 | BIO-101 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 |
| CS-315 | Robotics | Comp. Sci. | 3 | <i>null</i> |

- In relational algebra: *course* \bowtie *prereq*



Right Outer Join

- *course natural right outer join prereq*

| <i>course_id</i> | <i>title</i> | <i>dept_name</i> | <i>credits</i> | <i>prereq_id</i> |
|------------------|--------------|------------------|----------------|------------------|
| BIO-301 | Genetics | Biology | 4 | BIO-101 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-101 |
| CS-347 | <i>null</i> | <i>null</i> | <i>null</i> | CS-101 |

- In relational algebra: *course* \bowtie *prereq*



Full Outer Join

- *course natural full 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-315 | Robotics | Comp. Sci. | 3 | null |
| CS-347 | null | null | null | CS-101 |

- In relational algebra: *course ⋗ prereq*



Joined Types and Conditions

- **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.
- **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

| <i>Join types</i> |
|-------------------------|
| inner join |
| left outer join |
| right outer join |
| full outer join |

| <i>Join conditions</i> |
|-----------------------------------------------------------------|
| natural |
| on < predicate > |
| using (A₁, A₂, ..., A_n) |



Joined Relations – Examples

- *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 |

- *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 |

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



Joined Relations – Examples

- **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 |

- What is the difference between the above, and a natural join?
- **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 |

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



Joined Relations – Examples

- *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 |

- *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 |

| course_id | title | dept_name | credits | course_id | prereq_id |
|-----------|-------------|------------|---------|-----------|-----------|
| BIO-301 | Genetics | Biology | 4 | BIO-301 | BIO-101 |
| CS-190 | Game Design | Comp. Sci. | 4 | CS-190 | CS-101 |
| CS-315 | Robotics | Comp. Sci. | 3 | CS-347 | 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 the 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 vas <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.



View Definition and Use

- 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;
```



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.



Views Defined Using Other Views

- **create view *physics_fall_2017* 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 = '2017';*
- **create view *physics_fall_2017_watson* as**
select *course_id, room_number*
from *physics_fall_2017*
where *building= 'Watson';*



View Expansion

- Expand the view :

```
create view physics_fall_2017_watson as  
  select course_id, room_number  
  from physics_fall_2017  
  where building= 'Watson'
```

- To:

```
create view physics_fall_2017_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 = '2017')  
  where building= 'Watson';
```



View Expansion (Cont.)

- A way to define the meaning of views defined in terms of other views.
- Let view v_1 be defined by an expression e_1 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 (Cont.)

- **Example: View Definitions:**
 - CREATE VIEW v1 AS SELECT a, b FROM t1;
 - CREATE VIEW v2 AS SELECT v1.a, t2.c FROM v1, t2 WHERE v1.b = t2.b;
- **View Expansion Process:**
 - **Identify:** v1 is a view relation.
- **Replace v1 with its definition:**
 - SELECT t1.a, t2.c FROM *(SELECT a, b FROM t1) AS v1*, t2 WHERE v1.b = t2.b;
- **Final Expression:**
 - The expanded view expression no longer contains any view relations.
- **Key Points:**
 - **Non-recursive Definitions:** The process works as long as the view definitions are not recursive. Recursive definitions would create an infinite loop.
 - **Efficiency:** This expansion helps understand and optimize queries, ensuring correct and efficient execution.
 - By using view expansion, you break down complex nested view definitions into their fundamental components, making the overall query more transparent and manageable.



Materialized Views

- Certain database systems allow view relations to be physically stored.
 - Physical copy created when the view is defined.
 - Such views are called **Materialized view**:

Challenge: If relations used in the query are updated, the materialized view result becomes out of date

Need to **Maintain** the view, by updating the view whenever the underlying relations are updated.



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 into the *instructor* relation
 - Must have a value for salary attribute in the table.
- Two approaches
 - **Reject the insert**
 - **Insert 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');
- Issues
 - Which department, if multiple departments in Taylor?
 - What if no department is in Taylor?



And Some Not at All

- **create view** *history_instructors* **as**
select *
from *instructor*
where *dept_name*= 'History';
- What happens if we insert
('25566', 'Brown', 'Biology',
100000)
into *history_instructors*?



View Updates in SQL

- 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 specification**.
 - Any attribute not listed in the **select** clause can be set to null
 - The query does not have a **group by** or **having** clause.



Transactions

- A **transaction** consists of a sequence of query and/or update statements and is a “unit” of work
- The SQL standard specifies that a transaction begins implicitly when an SQL statement is executed.
- The transaction must end with one of the following statements:
 - **Commit work.** The updates performed by the transaction become permanent in the database.
 - **Rollback work.** All the updates performed by the SQL statements in the transaction are undone.
- Atomic transaction
 - either fully executed or rolled back as if it never occurred
- Isolation from concurrent transactions



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 \$4.00 an hour
 - A customer must have a (non-null) phone number



Constraints on a Single Relation

- **not null**
- **primary key**
- **unique**
- **check (P)**, where P is a predicate



Not Null Constraints

- **not null**

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

name **varchar(20) not null**

budget **numeric(12,2) not null**



Unique Constraints

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



The check clause

- The **check** (P) clause specifies a **predicate P that must be satisfied by every tuple in a relation**.
- 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.



Referential Integrity (Cont.)

- Foreign *keys can be* specified as part of the SQL **create table** statement
foreign key (dept_name) references department
- By default, a foreign key references the primary-key attributes of the referenced table.
- SQL allows a list of attributes of the referenced relation to be specified explicitly.
foreign key (dept_name) references department (dept_name)



Cascading Actions in Referential Integrity

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.
- An alternative, in case of delete or update is to cascade

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

- Instead of cascade we can use :
 - **set null**,
 - **set default**



Integrity Constraint Violation During Transactions

- Consider:

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

- How to insert a tuple without causing constraint violation?
 - Insert father and mother of a person before inserting person
 - OR, set **father and mother** to null initially, update after inserting all persons (not possible if father and mother attributes declared to be **not null**)
 - OR defer constraint checking



Complex Check Conditions

- The predicate in the check clause can be an arbitrary predicate that can include a subquery.

```
check (time_slot_id in (select time_slot_id from  
time_slot))
```

The check condition states that the `time_slot_id` in each tuple in the `section` relation is actually the identifier of a time slot in the `time_slot` relation.

- The condition has to be checked not only when a tuple is inserted or modified in `section`, but also when the relation `time_slot` changes



Assertions

- An **assertion** is a predicate expressing a condition that we wish the database always to satisfy.
- The following constraints, can be expressed using assertions:
- For each tuple in the ***student* relation**, the value of the attribute ***tot_cred*** must equal the sum of credits of courses that the student has completed successfully.
- An instructor cannot teach in two different classrooms in a semester in the same time slot
- An assertion in SQL takes the form:
create assertion <assertion-name> **check** (<predicate>);



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.



User-Defined Types

- **create type** construct in SQL creates user-defined type

```
create type Dollars as numeric (12,2) final
```

- Example:

```
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.
- Example:

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



Index Creation

- Many queries reference only a small proportion of the records in a table.
- It is inefficient for the system to read every record to find a record with particular value
- An **index** on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation.
- We create an index with the **create index** command
create index <name> on <relation-name> (attribute);



Index Creation Example

- **create table student**
*(ID varchar (5),
name varchar (20) not null,
dept_name varchar (20),
tot_cred numeric (3,0) default 0,
primary key (ID))*
- **create index studentID_index on student(ID)**
- The query:

```
select *  
from student  
where ID = '12345'
```

can be executed by using the index to find the required record, without looking at all records of *student*



Authorization

- We may assign a user several forms of authorizations 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.
- Each of these types of authorizations is called a **privilege**. We may authorize the user all, none, or a combination of these types of privileges on specified parts of a database, such as a relation or a view.



Authorization (Cont.)

- 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 give authorization
grant <privilege list> on <relation or view > to <user list>
- <user list> is:
 - a user-id
 - **public**, which allows all valid users the privilege granted
 - A role (more on this later)
- Example:
 - **grant select on department to Akbar, Asghar**
- 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 the ability to query using the view
 - Example: grant users U_1 , U_2 , and U_3 **select** authorization on the *instructor* relation:
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



Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.
revoke <privilege list> on <relation or view> from <user list>
- Example:
revoke select on student from U₁, U₂, U₃
- <privilege-list> may be **all** to revoke all privileges the revoker may hold.
- If <revoker-list> includes **public**, all users lose the privilege except those granted it 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

- A **role** is a way to distinguish among various users as far as what these users can access/update in the database.
- To create a role we use:
create a role <name>
- Example:
 - **create role instructor**
- Once a role is created we can assign “users” to the role using:
 - **grant <role> to <users>**



Roles Example

- **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;



Authorization on Views

- **create view** *geo_instructor* **as**
(select *
from *instructor*
where *dept_name* = 'Geology');
- **grant select on** *geo_instructor* **to** *geo_staff*
- Suppose that a *geo_staff* member issues
 - **select ***
from *geo_instructor*,
- What if
 - *geo_staff* does not have permissions on *instructor*?
 - Creator of 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;**
 - And more!



Other Authorization Features

- GRANT REFERENCES (dept_name) ON department TO Mariano;
 - The REFERENCES privilege allows a user (in this case, Mariano) to create a **foreign key** that references the specified column (dept_name) in the department table.
- REVOKE SELECT ON department FROM Amit, Satoshi CASCADE;
 - **CASCADE** ensures that the revocation propagates, removing permissions for all indirect recipients of the privilege.
- REVOKE SELECT ON department FROM Amit, Satoshi RESTRICT;
- **RESTRICT** prevents the revocation if there are users who have received the privilege from Amit or Satoshi. This maintains the hierarchy of granted privileges.
- grant select on department to Amit with grant option;
 - Means that Amit can transfer his grant to other users



CASCADE/ RESTRICT GRANT FEATURES

| Aspect | CASCADE | RESTRICT |
|-----------------------|------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Behavior | Revokes the privilege from the specified users and all those who were granted the privilege by them. | Revokes the privilege only if no dependent users exist. Otherwise, it raises an error. |
| Use Case | Use when you want to remove all dependencies. | Use when you want to ensure no other users are affected. |
| Example Result | If Amit granted SELECT to Bob, Bob loses the privilege when Amit's privilege is revoked. | Amit's privilege cannot be revoked if Bob received the privilege from Amit. |



End of Chapter 4