



# Chapter 4: Intermediate SQL

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use



# 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 and Natural Join

⑩ What is the difference between the two query statements?

1) **select \***

**from** student **join** takes **on** student.ID =takes.ID;

**select \***

**from** student ,takes

**where** student.ID =takes.ID;

2) **Select \***

**from** student **natural join** takes ;



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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

**Figure 2.1** The *instructor* relation.

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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

**Figure 2.7** The *teaches* relation.

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

**Figure 3.8** The natural join of the *instructor* relation with the *teaches* relation.

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

**Figure 4.1** The *student* relation.

12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	null

**Figure 4.2** The *takes* relation.

ID	ID	name	dept_name	tot_cred	course_id	sec_id	semester	year	grade
00128	00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2009	A
00128	00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2009	A-
12345	12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2009	C
12345	12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2009	A
12345	12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2010	A
12345	12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2009	A
19991	19991	Brandt	History	80	HIS-351	1	Spring	2010	B
23121	23121	Chavez	Finance	110	FIN-201	1	Spring	2010	C+
44553	44553	Peltier	Physics	56	PHY-101	1	Fall	2009	B-
45678	45678	Levy	Physics	46	CS-101	1	Fall	2009	F
45678	45678	Levy	Physics	46	CS-101	1	Spring	2010	B+
45678	45678	Levy	Physics	46	CS-319	1	Spring	2010	B
54321	54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2009	A-
54321	54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2009	B+
55739	55739	Sanchez	Music	38	MU-199	1	Spring	2010	A-
76543	76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2009	A
76543	76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2010	A
76653	76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2009	C
98765	98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2009	C-
98765	98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2010	B
98988	98988	Tanaka	Biology	120	BIO-101	1	Summer	2009	A
98988	98988	Tanaka	Biology	120	BIO-301	1	Summer	2010	null

**Figure 4.3** The result of *student* join *takes* on *student.ID* = *takes.ID* with second occurrence of *ID* omitted.



# Join operations – Example

## ■ 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

Suppose that there is some student who takes no courses. Then the tuple in the *student* relation for that particular student would not satisfy the condition of a natural join with any tuple in the *takes* relation, and that student's data would not appear in the result

## ■ Relation *prereq*

<i>course_id</i>	<i>prereq_id</i>
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



# 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

- *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>



# 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



# 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.

<i>Join types</i>	<i>Join Conditions</i>
inner join left outer join right outer join full outer join	natural on <predicate> using $(A_1, A_1, \dots, A_n)$



# Full Outer Join

■ *course* **natural full 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>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

but, full outer join is not supported in  
Mysql



# Joined Relations – Examples

- **course inner join prereq on**  
*course.course\_id = prereq.course\_id*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
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*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>	<i>course_id</i>
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	<i>null</i>	<i>null</i>



# Joined Relations – Examples

- **course 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

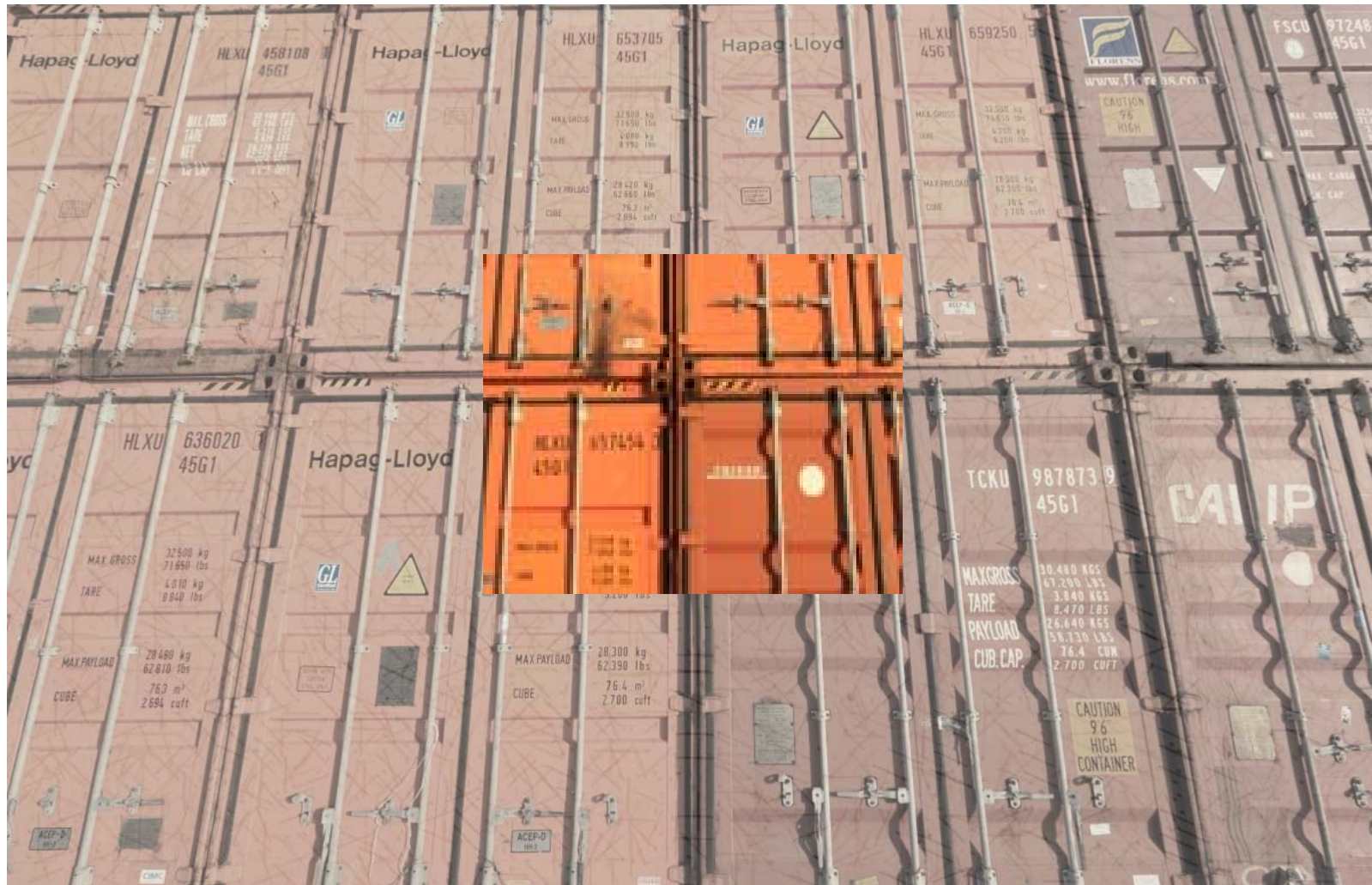
- **course full outer join prereq using (course\_id)**

<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>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101





# Views





# 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** *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
```

- Find all instructors in the Biology department

```
create view bio_instructor as
```

```
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

- 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* (indirect) 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.



# View Expansion

- 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



# 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 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** 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.





## 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*?
- add **with check option** at the end of view definition statement, if not satisfy where condition, the insertion would be rejected.



# Materialized Views

- **Materializing a view**: create a physical table containing all the tuples in the result of the query defining the view
- 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.



# 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**(can not recover after commit 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 .... end**
    - ▶ but Not supported on most databases



# 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.
- Some 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**
- **primary key**
- **unique**
- **check (P)**, where P 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 ( $A_1, A_2, \dots, A_m$ )

- 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

## ■ **check** (P)

where 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**,
- foreign key can be null if it is not declared



# 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 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 (next slide)



# Complex Check Clauses

- **check** (*time\_slot\_id* in (**select** *time\_slot\_id* **from** *time\_slot*)) in the definition of section
  - why not use a foreign key here?
- Every section has at least one instructor teaching the section.
  - how to 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 year (4 digit), 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



# Index Creation

- **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*)
- Indices are data structures used to speed up access to records with specified values for index attributes
  - e.g. **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*

*More on indices in Chapter 11*



# 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'));**



# 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



# 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 (will be discussed on more 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 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 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 <revokee-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

- **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*?



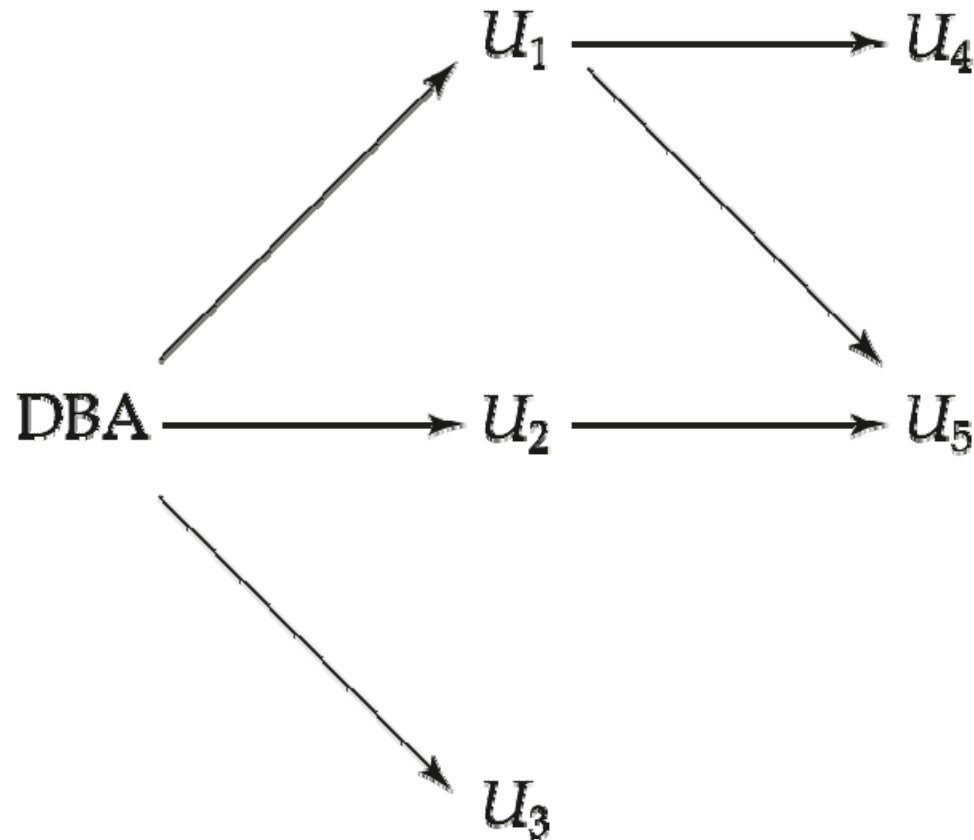
## Other Authorization Features

- **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**;  
(Amit can grant select on department to other users /roles)
  - **revoke select on** *department* **from** Amit, Satoshi **cascade**;  
(**cascade** is default)
  - **revoke select on** *department* **from** Amit, Satoshi **restrict**;
  - **revoke grant option for select on** department **from** Amit;
- Etc. read Section 4.6 for more details we have omitted here.





## Figure 4.08 (cascade revoke)



1. revoke the privileges of U1, what would happen?
2. suppose U2 was granted some privileges by the DBA, then U2 grants to U3(U3 was not granted by DBA directly), and U3 grants to U2, if the administrator revoke from U2, does U3 still have the privilege?



# Chapter 4

- 4.1
- 4.5
- 4.6
- 4.8
- 4.9
- 4.12
- 4.16
- 4.18



# End of Chapter 4

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use



## Figure 4.01

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>tot_cred</i>
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



# Figure 4.02

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>grade</i>
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	<i>null</i>



## Figure 4.03

<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	2009	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2009	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2009	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2009	A
12345	Shankar	History	32	CS-315	1	Spring	2010	A
12345	Shankar	Finance	32	CS-347	1	Fall	2009	A
19991	Brandt	Music	80	HIS-351	1	Spring	2010	B
23121	Chavez	Physics	110	FIN-201	1	Spring	2010	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2009	B-
45678	Levy	Physics	46	CS-101	1	Fall	2009	F
45678	Levy	Physics	46	CS-101	1	Spring	2010	B+
45678	Levy	Physics	46	CS-319	1	Spring	2010	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2009	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2009	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2010	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2009	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2010	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2009	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2009	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2010	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2009	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2010	<i>null</i>



## Figure 4.04

ID	name	dept_name	tot_cred	course_id	sec_id	semester	year	grade
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2009	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2009	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2009	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2009	A
12345	Shankar	History	32	CS-315	1	Spring	2010	A
12345	Shankar	Finance	32	CS-347	1	Fall	2009	A
19991	Brandt	Music	80	HIS-351	1	Spring	2010	B
23121	Chavez	Physics	110	FIN-201	1	Spring	2010	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2009	B-
45678	Levy	Physics	46	CS-101	1	Fall	2009	F
45678	Levy	Physics	46	CS-101	1	Spring	2010	B+
45678	Levy	Physics	46	CS-319	1	Spring	2010	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2009	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2009	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2010	A-
70557	Snow	Physics	0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2009	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2010	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2009	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2009	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2010	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2009	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2010	<i>null</i>





# Figure 4.05

ID	course_id	sec_id	semester	year	grade	name	dept_name	tot_cred
00128	CS-101	1	Fall	2009	A	Zhang	Comp. Sci.	102
00128	CS-347	1	Fall	2009	A-	Zhang	Comp. Sci.	102
12345	CS-101	1	Fall	2009	C	Shankar	Comp. Sci.	32
12345	CS-190	2	Spring	2009	A	Shankar	Comp. Sci.	32
12345	CS-315	1	Spring	2010	A	Shankar	History	32
12345	CS-347	1	Fall	2009	A	Shankar	Finance	32
19991	HIS-351	1	Spring	2010	B	Brandt	Music	80
23121	FIN-201	1	Spring	2010	C+	Chavez	Physics	110
44553	PHY-101	1	Fall	2009	B-	Peltier	Physics	56
45678	CS-101	1	Fall	2009	F	Levy	Physics	46
45678	CS-101	1	Spring	2010	B+	Levy	Physics	46
45678	CS-319	1	Spring	2010	B	Levy	Physics	46
54321	CS-101	1	Fall	2009	A-	Williams	Comp. Sci.	54
54321	CS-190	2	Spring	2009	B+	Williams	Comp. Sci.	54
55739	MU-199	1	Spring	2010	A-	Sanchez	Music	38
70557	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	Snow	Physics	0
76543	CS-101	1	Fall	2009	A	Brown	Comp. Sci.	58
76543	CS-319	2	Spring	2010	A	Brown	Comp. Sci.	58
76653	EE-181	1	Spring	2009	C	Aoi	Elec. Eng.	60
98765	CS-101	1	Fall	2009	C-	Bourikas	Elec. Eng.	98
98765	CS-315	1	Spring	2010	B	Bourikas	Elec. Eng.	98
98988	BIO-101	1	Summer	2009	A	Tanaka	Biology	120
98988	BIO-301	1	Summer	2010	<i>null</i>	Tanaka	Biology	120





## Figure 4.06

<i>Join types</i>	<i>Join conditions</i>
<b>inner join</b> <b>left outer join</b> <b>right outer join</b> <b>full outer join</b>	<b>natural</b> <b>on</b> <predicate> <b>using</b> ( $A_1, A_2, \dots, A_n$ )



## Figure 4.07

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
69987	White	<i>null</i>	<i>null</i>

*instructor*

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000
<i>null</i>	Taylor	<i>null</i>

*department*