



Chapter 4: Intermediate SQL

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan

See 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 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

□ 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



Left Outer Join

- *course* **natural left outer join** *prereq*
- *It returns all rows from left table even if there are no matches in right table*

<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*
- *It returns all rows from right table even if there are no matches in left table*

<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



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



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)



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>prere_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>prere_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 natural right outer join prereq**

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prere_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 right outer join prereq using (course_id)**

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prere_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



Quiz Q1: Are (**r left outer join s**) and (**s right outer join r**) the same, if we ignore the order of the columns in the result?

- (1) Yes
- (2) No
- (3) depends on the schema of r and s
- (4) none of the above

Quiz Q2: Which of the following give exactly the same result, given relations r(A,B) and s(B, C)

(A) **r natural join s** (B) **r join s using (B)** (C) **r join s on (r.B=s.B)**

- (1) A and B
- (2) A and C
- (3) B and C
- (4) all three



View Definition

- 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 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
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';
```



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');
- We cannot add a tuple directly to a view
- Instead the insertion can be done by inserting 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*?

Quiz Q3: The insertion into the view

- (1) cannot be done by any update to *instructor*
- (2) can be done by a simple insert to *instructor*
- (3) can be done, for any department other than History
- (4) none of the above



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 end**



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**



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)



Deferred Checking of Constraints

- 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**
- Deferred constraints are checked at end of transaction
 - Even if father tuple does not exist when a particular person is inserted, no violation provided father is inserted before transaction commits.
- Particularly useful for cyclic references
 - E.g. add attribute *spouse* to a *married_person* relation as follows:
spouse **char**(10) **not null**;
constraint *spouse_ref* **foreign key** *spouse*
references *married_person*;
 - Since spouse cannot be null, without deferred constraints we cannot insert any tuples into *married_person*



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 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 (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

Quiz Q4: The expression

date '2010-12-14' + (date '2010-12-01' – date '2010-30-11')

is (1) valid and returns a date (2) valid and returns an interval
(3) invalid (4) none of the above



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



Large Objects

- Database restrict the size of char and varchar types
 - typical limit is less than 4KB
- Large object types can be used instead to store large sized data items such as text, images, videos etc.
- Character large object (**clob**) and binary large object (**blob**)
 - *book review* **clob**(10KB)
 - *image* **blob**(10MB)
 - *movie* **blob**(2GB)



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 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 *branch* 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.



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**;
- Etc. read Section 4.6 for more details we have omitted here.