

DATABASE SYSTEM PRINCIPLE - INTERMEDIATE SQL

李旭东 Li-Xudong

LEEUDONG@NANKAI.EDU.CN
NANKAI UNIVERSITY

OBJECTIVES

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

©LXD

JOINED RELATIONS

- **Join operations** 连接谓词 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

©LXD

JOIN OPERATIONS – EXAMPLE

- Relation *course*

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

- Relation *prereq*

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Observe that

1. prereq information is missing for **CS-315**
2. course information is missing for **CS-437**

©LXD

THE RESULT OF NATURAL JOIN

```
select *
from course natural 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

Some information lost:

course information is missing for **CS-315**

©LXD

OUTER JOIN 外连接

- 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
 - preserves tuples only in the relation named **before** (to the left of) the operation.
- right outer join
- full outer join

©LXD

LEFT OUTER JOIN

```
select *
from course natural left outer join prereq;
```

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

©LXD

RIGHT OUTER JOIN

```
select *
from course natural right outer join prereq;
```

course_id	title	dept_name	credits	prere_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	null	null	null	CS-101

©LXD

FULL OUTER JOIN

```
select *
from course natural full outer join prereq;
```

course_id	title	dept_name	credits	prere_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

©LXD

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.

©LXD

JOINED RELATIONS

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

©LXD

JOINED RELATIONS – EXAMPLES

- **course inner join prereq on**
 $course.course_id = prereq.course_id$

course_id	title	dept_name	credits	prere_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

©LXD

JOINED RELATIONS – EXAMPLES

What is the difference between the following sql, and a natural join?

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>

©LXD

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

©LXD

JOINED RELATIONS – EXAMPLES

course **full 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

©LXD

OUTER JOIN:
ON V.S. WHERE

- `select *`
`from student left outer join takes on student. ID = takes. ID`
- `select *`
`from student left outer join takes on true`
- `where student. ID = takes. ID`

©LXD

OBJECTIVES

- Join Expressions
- **Views**
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

©LXD

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
```

©LXD

VIEWS

- 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**.

©LXD

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.

©LXD

VIEW DEFINITION

- 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.

©LXD

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 biology_instructor as
select *
from faculty
where dept_name = 'Biology'

©LXD

EXAMPLE VIEWS

- 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;

©LXD

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';

©LXD

VIEWS DEFINED USING OTHER VIEWS

- create view *physics_fall_2009_watson* as
 select *course_id*, *room_number*
 from *physics_fall_2009*
 where *building*= 'Watson';

©LXD

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');
```

©LXD

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.

©LXD

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

©LXD

UPDATE OF A VIEW

- A view of instructors without their salary
 create view *faculty* as
 select *ID*, *name*, *dept_name*
 from *instructor*

©LXD

UPDATE OF A VIEW (CONT.,)

- 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

©LXD

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?

©LXD

UPDATABLE VIEW

- 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.

©LXD

UPDATE OF A VIEW - AND SOME NOT AT ALL

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

©LXD

MATERIALIZED VIEWS 物化视图

- **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
 - Materialized view maintenance 物化视图维护

©LXD

OBJECTIVES

- Join Expressions
- Views
- **Transactions**
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

©LXD

TRANSACTIONS 事务

- Transaction: Unit of work
- Atomic transaction: either fully executed or rolled back as if it never occurred
- Isolation from concurrent transactions
- Transactions begin implicitly 隐式地: 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 mysql: `set autocommit=0` or `start transaction`
 - In psql (postgresql): `set autocommit off`
 - In SQL:1999, can use: **begin atomic end**

©LXD

Not supported on most databases

TEST TRANSACTION

```
#!/usr/bin/python3
...
conn = mysql.connector.connect(**config)
cursor = conn.cursor()
try:
    cursor.execute('select name,salary from instructor01')
    for name,salary in cursor:
        print(name.decode('utf-8'),' ',salary)
    cursor.execute('update instructor01 set salary=salary-100 where name="Bondi"')
    cursor.execute('update instructor01 set salary=salary+100 where name="Dale"')
    cursor.execute('select name,salary from instructor01')
    for name,salary in cursor:
        print(name.decode('utf-8'),' ',salary)
except mysql.connector.Error as e:
    print(e)
finally:
    cursor.close()
    conn.close()
    print('Transaction test completed')
```

```
python3 transactiontest.py
Bondi , 9900.00
Dale , 10100.00
Bondi , 10000.00
Dale , 10000.00
```

EXAMPLE: TRANSACTION

T ₁	T ₂
<pre>read (A) A := A - 50 write (A) read (B) B := B + 50 write (B)</pre>	<pre>read (A) temp := A * 0.1 A := A - temp write (A) read (B) B := B + temp write (B)</pre>

- Initial
 - Account A:\$1000
 - Account B:\$2000
- Case 1:T1, T2
 - Result ?**
- Case 2:T2, T1

EXAMPLE: TRANSACTION

T ₁	T ₂
<pre>read (A) A := A - 50 write (A) read (B) B := B + 50 write (B)</pre>	<pre>read (A) temp := A * 0.1 A := A - temp write (A) read (B) B := B + temp write (B)</pre>

- Initial
 - Account A:\$1000
 - Account B:\$2000
- Case 3
 - Result ?**

OBJECTIVES

- Join Expressions
- Views
- Transactions
- Integrity Constraints**
- SQL Data Types and Schemas
- Authorization

INTEGRITY CONSTRAINTS完整性约束

- 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

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).

©LXD

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'))
)
```

©LXD

REFERENTIAL INTEGRITY 参照完整性

- 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.

©LXD

CASCADING 级联 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**

©LXD

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)
```

©LXD

INTEGRITY CONSTRAINT VIOLATION DURING TRANSACTIONS (CONT.,)

- 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** 延迟 constraint checking: **initially deferred**

©LXD

COMPLEX CHECK CLAUSES

- **check** (*time_slot_id* in (select *time_slot_id* from *time_slot*))
 - why not use a foreign key here?
- Unfortunately: subquery in check clause not supported by pretty much any database
 - Alternative: **triggers** 触发器
- **create assertion** <assertion-name> **check** <predicate>;
 - **assertion** 断言
 - Also not supported by anyone

©LXD

COMPLEX CHECK CLAUSES

- ASSERTION: CASE

```
create assertion credits_earned_constraint check
(not exists (select ID
             from student
             where tot_cred <> (select sum(credits)
                                from takes natural join course
                                where student.ID= takes.ID
                                and grade is not null and grade <> 'F' )
```

©LXD

OBJECTIVES

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- **SQL Data Types and Schemas**
- Authorization

©LXD

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'

©LXD

BUILT-IN DATA TYPES IN SQL

- **Functions of Time**
 - **current_date**
 - **current_time**
 - **localtime**
 - **current_timestamp**
 - **localtimestamp**

©LXD

BUILT-IN DATA TYPES IN SQL

- **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

©LXD

DEFAULT VALUES

```
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));
```

```
insert into student(ID, name, dept_name)
values ('12789', 'Newman', 'Comp. Sci.');
```

©LXD

INDEX CREATION 创建索引

- **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)**

©LXD

INDEX CREATION 创建索引 (CONT.,)

- 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*

©LXD

LARGE-OBJECT TYPES

- Large objects (photos, videos, CAD files, etc.) are stored as a large object
 - clob: character data
 - blob: binary data
- locator for a large object 定位器

```
book_review clob(10KB)
image blob(10MB)
movie blob(2GB)
```

©LXD

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**);

©LXD

USER-DEFINED TYPES

- **create type**
- **drop type**
- **alter type**

©LXD

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'));

©LXD

OBJECTIVES

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- **Authorization**

©LXD

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.

©LXD

AUTHORIZATION

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.

©LXD

AUTHORIZATION 授予 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)

©LXD

AUTHORIZATION SPECIFICATION IN SQL (CONT.,)

- 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).

©LXD

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

©LXD

REVOKING 收回 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

©LXD

REVOKING AUTHORIZATION IN SQL (CONT.,)

- <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.

©LXD

ROLES 角色

- **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*

©LXD

ROLES (CONT.,)

- Chain of roles
 - **create role dean**;
 - **grant instructor to dean**;
 - **grant dean to Satoshi**;

©LXD

AUTHORIZATION ON VIEWS

- **create view geo_instructor as**
(select *
from instructor
where dept_name = 'Geology');
- **grant select on geo_instructor to geo_staff**

©LXD

AUTHORIZATION ON VIEWS

- 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*?

©LXD

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;`

©LXD

SUMMARY

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

©LXD

Q&A?

THANKS!

leexudong@nankai.edu.cn