Lecture 6: Basic Query Composition using SQL DML

CSX3006 DATABASE SYSTEMS

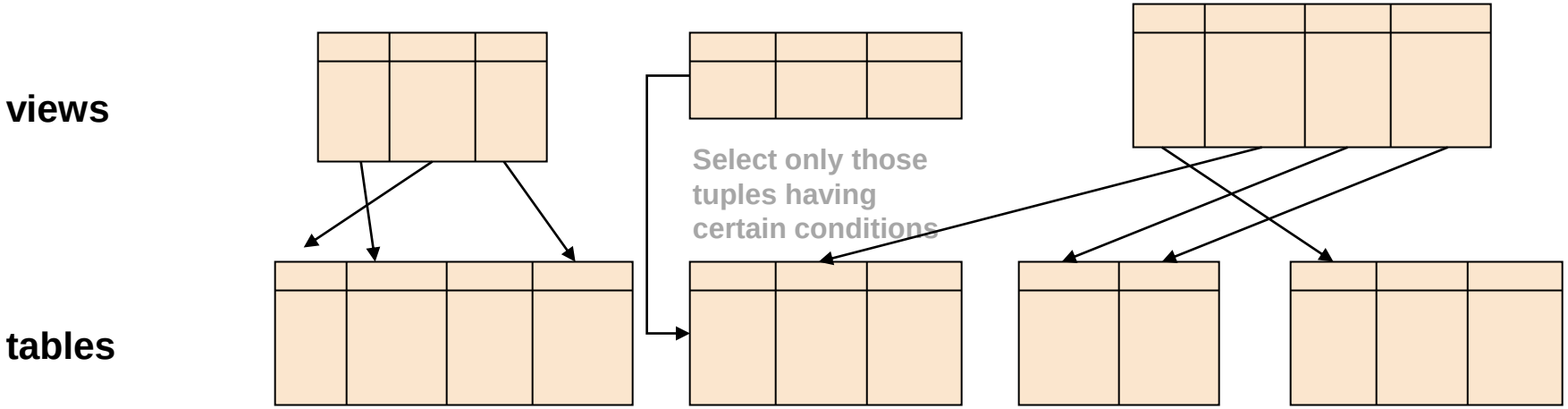
ITX3006 DATABASE MANAGEMENT SYSTEMS

Outline

- Views and View Expansions
- Modification Operations in SQL DML
  - Deletion operations
  - Insertion operations
- Update operations

What is a View? - 1

- **What is a view?** (in the context of Database and DBMS)
  - **A named query**;
    - a tailored presentation of the data contained in one or more tables (or other views)
  - A **'window'** into the underlying tables



What is a View? - 2

- For DBMS
  - Views are named or derived 'virtual' table

- For end-users
  - Views are just like normal tables (which in fact do not exist!)

View vs. Table

- Views are created, dropped, modified in a similar way to tables are
  - When you create a view, you define the view in terms of other tables (or other views)
- So how do views differ from tables?

	TABLE	VIEW
Is definition of a table stored in the data dictionary of DBMS?	Yes	Yes
Contains **actual data**? (table instance)	Yes	No (Just 'window' to actual data contained in the real tables)
Can have **indexes** on its attributes?	Yes	No

Why Views? - 1

- Query Simplification
  - Queries on the view are simpler to express than directly expressing queries on the underlying tables
    - A view joining data from multiple tables can be created, ***hiding the fact*** that the data are obtained by joining multiple tables
- Convenient Usage
  - Views can also be **used as inputs** to other queries
- Allows different perspective
  - A view creates a different perspective of enterprise data than those reflected in the logical database schema
  - Allows **customization** of data models **for different purposes** and/or **users**

Why Views? - 2

- Schema Transparency / Location Transparency
  - Ability to **hide underlying logical database schema** from the application, therefore the user.
  - **Changes** to **schemas** of underlying tables do **not affect views**, therefore, the **application based on the views**, *as long as the columns used in the definition of the view* ***are not modified***

- Security
  - To provide an additional level of table security
    - Restricting access to a pre-determined set of rows and/or columns of a table

How to define and create views? - 1

- 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

v is the view name

- *v* can be used to **refer to** the **virtual table** that the view generates.
- When a view is created, the query expression is stored in the database;
  - **Only the definition of the view** (not data) **is stored** in the data dictionary of DBMS

How to define and create views? - 2

```
create view account_customer (customer_name, account_number, balance) as
    select customer_name, depositor.account_number, balance
    from depositor, account
    where depositor.account_number = account.account_number;
```

- **account_customer** can be thought as a **'virtual table'** with the following schema: (customer_name, account_number, balance)

- A view can be used as a relation (table).

```
select customer_name
from account_customer
group by customer_name
having count(account_number) >= 2 and sum(balance) > 1000;
```

More example of a view

- ```
  create view branch_total_loan (branch_name, total_loan) as
  select branch_name, sum(amount)
  from loan
  group by branch_name;
  ```

- Defines a ‘virtual table’ called **branch_total_loan** with the following schema: **(branch_name, total_loan)**
  ```

More detail about view

- The definition of the view is stored in the data dictionary of DBMS permanently until you explicitly remove the view from your database
  - In the same way as meta-data about your table schema are stored in the DBMS
- This is in contrast to 'temporary' relations defined in the from clause as a **subquery**, which are only **available temporarily** during the evaluation of the 'outer query' which uses the temporary relations

Delete View

- Unneeded views can be removed explicitly from the database

```
drop view branch_total_loan
```

- Realize that when a view is removed from the database, no data is lost since a view is a 'virtual table'; Only the definition of the view is removed
  - This is contrast to the dropping of tables, which will cause all data to be deleted

View Defined Using Other Views

```
create view all_customer as
(           select branch_name, customer_name
            from depositor, account
            where depositor.account_number = account.account_number  )
union
(           select branch_name, customer_name
            from borrower, loan
            where borrower.loan_number = loan.loan_number  )
```

- Definition of view perryridge_customer is defined in terms of another view, **all_customer**

```
create view perryridge_customer as
            select customer_name
            from all_customer
            where branch_name = 'Perryridge'
```

Database Modification Operations

- How do we make changes to the database?
  - Inserting new tuples into a relation
  - Deleting certain tuples from a relation
  - Updating values of certain attributes of existing tuples

Delete Command

- Syntax

  delete from *r* where *P*

- Deletes those tuples from *r* which satisfies the predicate condition *P*
  - First find all tuples *t* in *r* for which *P(t)* is true, and then deletes them from *r*
  - Deletes whole tuples; cannot delete values on only particular attributes
  - delete command operates on only one table at a time
    - If we need to delete tuples from multiple tables, we must issue one delete command for each table.
  - where clause in the delete command can be *as complex as* where in select command

Deletion Example - 1

```
delete from account;
```

- Delete **all rows** from the table account.

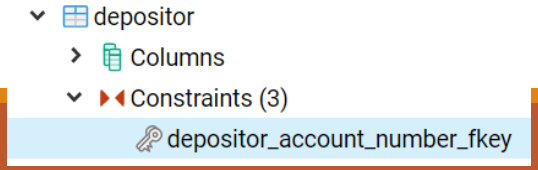
Deletion Example - 2

- Delete all accounts at the Perryridge branch

  delete from account

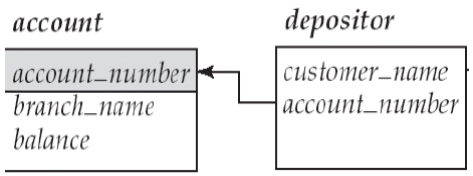
  where branch_name = ‘Perryridge’;

- Note that depositor.account_number is a foreign key referencing account.account_number
- `ERROR: update or delete on table "account"` **`violates foreign key constraint`** `"depositor_account_number_fkey" on table "depositor" DETAIL: Key (account_number)=(A-102 ) is still referenced from table "depositor". SQL state: 23503`
  - Deletion of tuples from account branch may be prevented, cause cascading deletion on depositor relation or may cause null or default values to be specified in depositor.account_number **depending on the referential integrity settings** specified on the **depositor** relation.



Referential Integrity Constraints specified in the referencing table

```
delete from account
        where branch_name = 'Perryridge';
```




```
create table depositor (
        customer_name        char(15),
        account_number       char(10),
        primary key (customer_name, account_number),
        foreign key (customer_name) references customer,
        foreign key (account_number ) references account);
```

Default is to **reject** the delete or update **operations** on branch relation *that will cause the referential integrity to be broken*

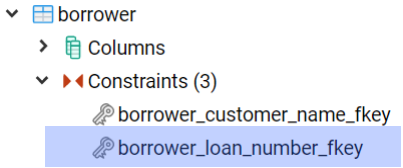
- when you do not specify any rule on the foreign key specification

Deletion Examples - 3

- Delete all loans with loan amounts between $1300 and $1500

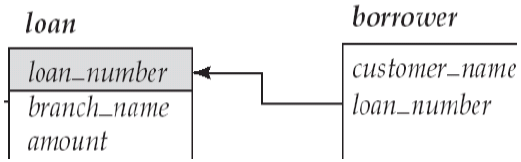
```
delete from loan
where amount between 1300 and 1500;
```

- Consider the database schema of the banking enterprise
  - What **integrity constraints** may be **related** to the above delete operation?
  - What are the possible actions that can be taken by the DBMS?



Referential Integrity Constraints specified in the referencing table

```
delete from loan
        where amount between 1300 and 1500;
```




```
create table borrower (
        customer_name       char(15),
        loan_number         char(10),
        primary key (customer_name, loan_number ),
        foreign key (customer_name) references customer,
        foreign key ( loan_number ) references loan);
```

Default is to **reject** the delete or update **operations** on branch relation *that will cause the referential integrity to be broken*

- when you do not specify any rule on the foreign key specification

Other Settings of Referential Integrity Constraints

b) Also delete all rows in **borrower** relation having the same loan_number as the deleted ones in **loan** relation → cascade

```
foreign key ( loan_number ) references loan
                                        on delete cascade
                                        on update cascade,
```

c) Set the loan_number of **borrower** relation to null → set null

```
foreign key ( loan_number ) references loan
                                        on delete set null
                                        on update cascade,
```

d) Set the loan_name of **borrower** relation to a default value → set default

```
foreign key ( loan_number ) references loan
                                        on delete set default
                                        on update cascade,
```

Delete Example - 4

- Delete all accounts at every branch located in Brooklyn

```
delete from account
where branch_name in (
        select branch_name
        from branch
        where branch_city = 'Brooklyn' );
```

- If the referential integrity on the **depositor** is set to **cascade** on delete, the DBMS will **also remove** tuples from the **depositor** relation whose account_number is managed by any branch in Brooklyn.

- Assume that no constraint setting is specified in the database schema, and you need to maintain the business logic manually.
  - What actions do you need to take?

Delete Examples – 4

What actions do you need to take? (Cont.)

1. Delete rows from **referencing table (depositor)**

```
delete from depositor
 where account_number in (
            select account_number
           from account natural join branch
where branch_city = 'Brooklyn' );
```

2. Delete rows from **referenced table (account)**

```
delete from account
where branch_name in (
            select branch_name
            from branch
            where branch_city = 'Brooklyn' )
```

Delete Example - 5

- Delete the record of all accounts with balances below the average at the bank.

```
delete from account
where balance < ( select avg(balance)
                  from account )
```

- Note that **nested query** in the where clause of the delete command can refer to the same relation from which tuples are to be deleted.
- How is the above query evaluated?
  - Step 1: The **sub-query is evaluated first** to find the average balance.
  - Step 2: Tuples whose balance is less than the average found in step 1 is selected.
  - Step 3: Those tuples selected in Step 2 are deleted.

Insertion - 1

- To insert data into a relation,
  - Tuples to be inserted must be *'compatible'* to the schema of a relation into which the tuples are to be inserted
    - The **same number** of **attributes**
    - Values must be in the **same domain** as the corresponding attributes
- Different ways of inserting a tuple:

```
insert into account
        values ('A-1234', 'Perryridge', 1200);

insert into account (account_number, branch_name, balance)
        values ('A-1234', 'Perryridge', 1200);

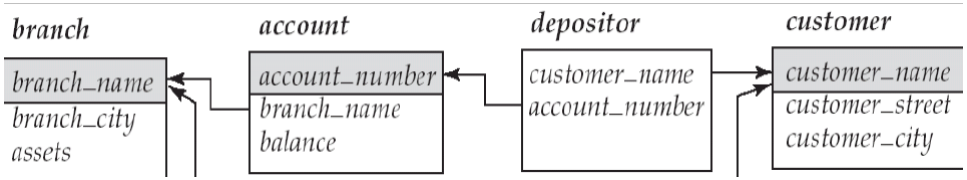
insert into account (branch_name, account_number, balance)
        values ('Perryridge', 'A-1234',1200);
```

Attributes' order could be swapped if explicitly specify attribute names

Insertion - 2

insert into account (account_number, branch_name, balance)
values ('A-1234', 'Perryridge', 1200);

- What integrity constraints should we need to make sure of?
  - There is a branch called 'Perryridge' in the branch table
  - You will also have to insert a tuple to a depositor table



Insertion example

- Provide as a gift for all loan customers of the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account

```
insert into account
        select loan_number, branch_name,  200
        from loan
        where branch_name = 'Perryridge' ;

insert into depositor
        select customer_name, loan_number
        from borrower natural join loan
        where branch_name = 'Perryridge' ;
```

- Be reminded that sub-query is fully evaluated before any insertion of tuples is performed

Updates

- Can change values of particular attributes of selected tuples in a relation

```
update r
set attribute = expression [, attribute2 = expression2]
where P
```

Update examples - 1

- Pay 5 percent interest to all the accounts in the bank

```
update account
set balance = balance * 1.05;
```

Update examples - 2

- Pay 5 percent interest to accounts whose balance is greater than average

```
update account
set balance = balance * 1.05
where balance > all (   select avg (balance)
                        from account ) ;
```

Note on updates

- Multiple attributes can be updated together by listing them in the **set** clause separated by comma
- where clause of the **update** can contain any legal constructs allowed in the **where** clause of the **select** command.
- Be reminded that where clause is fully evaluated before any updates to the attribute values are made

Case based update - 1

- All accounts with balances over $10000 receives 6 percent interest, whereas all others receive 5 percent.

```
update account
set balance = balance * 1.06
where balance > 10000 ;

update account
set balance = balance * 1.05
where balance <= 10000 ;
```

- Note that in the above, the order of the two update statements is important.
- What would happen if we changed the order?

Case based update - 2

- We can avoid possible mistakes by employing case clause

```
update account
   set balance =  case
                     when balance <= 10000 then balance *1.05
                     else   balance * 1.06
                  end
```

Practice 6-1

- Create the Database for the Library in Postgresql to keep records of students who borrow books.
  - Write SQL DDL to create database schema and necessary constraints.
  - Explain your constraints in your own words.

Library Database

Student relation

student_id	student_name	faculty

Borrow relation

student_id	isbn	borrow_month

Book relation

isbn	title	number_of_copies

Library Relations

Student relation

student_id	student_name	faculty
5725001	Paul Smith	Science and Technology
5815002	Alice Summerville	Science and Technology
5817013	Masha Winston	Laws
5819020	Tom Lee	Biology
5811051	Mark Cooper	BBA
5915004	Peter Highlander	BBA

Borrow relation

student_id	isbn	borrow_month
5725001	0760555841236	April
5815002	0760555841236	April
5817013	0251462157459	April
5819020	0760482456211	April
5815002	0584215622477	April
5815002	0584215622477	May
5819020	0154215871222	May
5915004	0154215871222	May

Book relation

isbn	title	number_of_copies
0584215622477	The Foundation in Mathematics	5
0154215871222	English Grammar in Use	5
0251462157459	Civil Laws	2
0760482456211	Bioinformatics	2
0760555841236	Python Programming	3

Practice 6-2

- Refer to database in the Practice 6-1

1. Insert all records into the tables
2. Retrieve a list of students' id, name and faculty who are in the faculty of Science and Technology.
3. Count number of books borrowed by students in each faculty.
4. Count number of books borrowed by students in each faculty in each month.
5. Retrieve the student's name who borrowed at least one book that is same as Alice Summerville.
6. Find the isbn and title of books that are borrowed by students in Biology Faculty but not by students in BBA Faculty.

Practice 6-3

7. Update the name of Mark Cooper to Sheldon Cooper
8. Update the ID of Peter Highlander to 6011001 and his faculty to Science and Technology
9. Create a view that shows a list of books' title and ISBN that are borrowed by students.
10. Delete the book Python Programming from the database.
11. Create 2 useful views that show the useful information from the database beside above questions.