同济大学 TONGJI UNIVERSITY

ADMIS Lab
先进数据与机器智能系统实验室

SQL介绍
Introduction to SQL

李文根/Wengen Li
Email: lwengen@tongji.edu.cn

先进数据与机器智能系统实验室（ADMIS）
https://admis.tongji.edu.cn/main.htm

同济大学 TONGJI UNIVERSITY 电子与信息工程学院 College of Electronics and Information Engineering 计算机科学与技术系 Department of Computer Science and Technology

课程概要

- **Part 0: Overview**
  - Ch1: Introduction
- **Part 1 Relational Databases**
  - Ch2: Relational model
  - **Ch3: Introduction to SQL**
  - Ch4: Intermediate SQL
  - Ch5: Advanced SQL
- **Part 2 Database Design**
  - Ch6: Database design based on E-R model
  - Ch7: Relational database design
- **Part 3 Application Design & Development**
  - Ch8: Complex data types
  - Ch9: Application development
- **Part 4 Big data analytics**
  - Ch10: Big data
  - Ch11: Data analytics
- **Part 5 Data Storage & Indexing**
  - Ch12: Physical storage system
  - Ch13: Data storage structure
  - Ch14: Indexing
- **Part 6 Query Processing & Optimization**
  - Ch15: Query processing
  - Ch16: Query optimization
- **Part 7 Transaction Management**
  - Ch17: Transactions
  - Ch18: Concurrency control
  - Ch19: Recovery system
- **Part 8 Parallel & Distributed Database**
  - Ch20: Database system architecture
  - Ch21-23: Parallel & distributed storage, query processing & transaction processing
- **Part 9**
  - DB Platform: **OceanBase**, MongoDB, Neo4J

目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

SQL概览

- IBM Sequel language is developed as part of System R project at the IBM San Jose Research Laboratory in the early 1970s
- Renamed **Structured Query Language (SQL)**
- ANSI (美国国家标准学会) and ISO (国际标准化组织) standard SQL
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL-99
  - SQL: 2003, 2006, 2008, 2011, 2016
- Commercial systems offer most SQL-92 features, plus varying feature sets from later standards and special proprietary features
  - Not all examples here may work on particular DBMS systems

SQL的组成

- **数据定义语言（Data definition language，DDL)**
  - Relation schemas
  - Integrity constraints
  - View
  - Authorization
- **数据操纵语言（Data manipulation language，DML)**
  - Queries
  - Insertion
  - Deletion
  - Updates
  - Transaction processing

目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

数据定义语言DDL

- **定义关系及其相关信息，包括：**
  - **关系的模式**
  - 属性的域(domain)
  - 完整性约束
  - 索引结构
  - 安全性和权限信息
  - 物理存储结构

SQL中的数据类型

- **char(n)**
  - fixed length character string with user-specified length n
- **varchar(n)**
  - variable length character strings with user-specified maximum length n
- **int**
  - Integer, a finite subset of the integers that is machine-dependent
- **smallint**
  - small integer, a machine-dependent subset of the integer domain type
- **numeric(p，d)**
  - fixed point number (定点数), with user-specified precision of p digits, and with d digits to the right of decimal point
  - E.g., numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 nor 0.32 can be stored exactly

SQL中的数据类型(续)

- **real, double precision**
  - floating point and double-precision floating point numbers, with machine-dependent precision
- **float(n)**
  - floating point number with user-specified precision of at least n digits
- **null value**
  - allowed in all domain types. Declaring an attribute to be not null prohibits null values for that attribute

- In SQL-92, we can create user-defined domain types
  - **create domain** person_name char(20) **not null**

模式定义

- Relation is defined using the *create table* command:

  create table $r(A_1\ D_1, A_2\ D_2, \ldots, A_n\ D_n,$
  $(integrity\_constraint_1), \ldots, (integrity\_constraint_k))$

  - r: the relation name
  - A_i: an attribute name in the schema of relation r
  - D_i: the data type of attribute A_i
- 例:

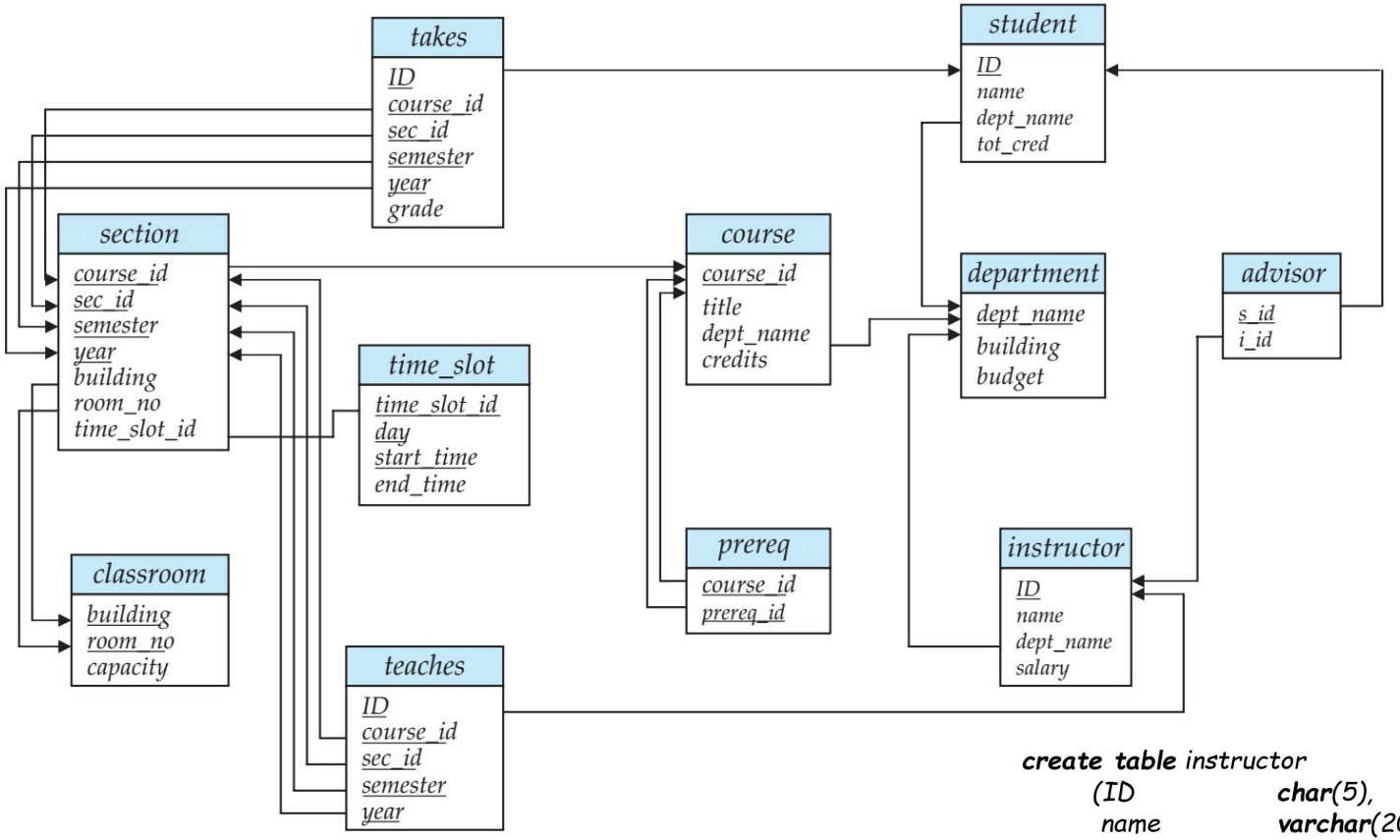
```
create table department(
    dept_name       varchar(20),
    building        varchar(15),
    budget          numeric(12, 2),
    primary key (dept_name));
```

完整性约束

- not null
- primary key $(A_1, ..., A_n)$
- foreign key $(A_{k1}, A_{k2} ..., A_{kn})$ references s
- check (P), where P is a predicate

```
create table instructor(
    ID          char(5),
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8, 2),
    primary key (ID),
    foreign key (dept_name) references department,
    check (salary >= 0));
```

注意: **Primary key** declaration on an attribute automatically ensures **not null** and **unique** in SQL-92 onwards, needs to be explicitly stated in SQL-89



思考：用SQL定义student和advisor?

```
create table instructor
        (ID             char(5),
         name           varchar(20) not null,
         dept_name      varchar(20),
        salary          numeric(8, 2),
        primary key (ID),
        foreign key (dept_name) references department,
        check (salary >= 0));
```

删除或修改Table/Relation

- **drop table**: deletes all information about the dropped relation from the database
- **alter table**: add attributes to a relation or drop attributes from a relation

  alter table r add A D

  – All tuples in the relation are assigned null as the value for the new attribute

  alter table r drop A

  – Dropping of attributes is not supported by some databases

目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

SQL查询的基本结构

- 典型的SQL查询结构：

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

- The result of an SQL query is a relation and each query is equivalent to the relational algebra expression:

$$\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

for each tuple t_1 **in** relation r_1
  for each tuple t_2 **in** relation r_2
    …
    for each tuple t_m **in** relation r_m
      Concatenate $t_1, t_2, \ldots, t_m$ into a single tuple t
      Add t into the result relation

Select子句

- The select clause lists the attributes desired in the result of a query
  - correspond to the projection (投影) operation of the RA (relational algebra)
  - E.g., find the names of all departments in the instructor relation

```
select dept_name
from instructor
```

  - 对应的关系代数表达式：

$$\Pi_{dept\_name}(instructor)$$

  - **注意：**SQL is case insensitive

Select子句(续)

- SQL allows **duplicates** in relations. To eliminate the duplicates, insert the keyword **distinct** after select
  - Find the names of all departments in the instructor relation, and remove duplicates

```
select distinct dept_name
from instructor
```

  - The keyword all specifies that duplicates should not be removed

```
select all dept_name
from instructor
```

Select子句(续)

- An asterisk in the select clause denotes “all attributes”

  select ∗
  from *instructor*

- The select clause can contain arithmetic expressions involving the operation +, −, ∗, and /, and operating on constants or attributes of tuples
- E.g.,

  select *ID, dept_name, salary ∗ 1.1*
  from *instructor*

From子句

- The from clause lists the relations involved in the query
  - corresponds to the Cartesian product operation of the RA

- 例：find the Cartesian product $borrower \times loan$

```
select *
from borrower, loan
```

- 例：find the name, loan number and loan amount of all the customers that have a loan at the Jiading branch

```
select customer_name, borrower.loan_number, amount
from borrower, loan
where  borrower.loan_number=loan.loan_number and
       branch_name='Jiading'
```

Where子句

- The where clause specifies conditions that the result must satisfy
  - correspond to the selection predicate of the RA
  - 例：*find the names of those instructors who are with the Department of Computer Science and have the salary larger than 70000*

```
select name
from instructor
where dept_name = 'Computer Science' and salary > 70000
```

Comparison results can be combined using the logical connectives **and**, **or**, and **not**

Where子句(续)

- SQL includes a between comparison operator
  - 例：*find the loan numbers of those loans with loan amount between $90,000 and $100,000*

```
select loan_number
from loan
where amount between 90000 and 100000
```

自然连接(Natural Join)

select $A_1, A_2, \ldots, A_n$
from r_1 ***natural join*** r_2 ***natural join …natural join*** r_m
where *P;*

select *name, course_id*
from *instructor* ***natural join*** *teaches;*

select *name, title*
from *instructor* ***natural join*** *teaches, course*
where *teaches.course_id=course.course_id;*

join … using (…)

- natural join of instructor and teaches
  - (ID, name, dept_name, salary, course_id, sec_id)
- course
  - (course_id, title, dept_name, credits)

结果不等

```
select name, title
from instructor natural join teaches, course
where teaches.course_id= course.course_id;

select name, title
from instructor natural join teaches natural join course;

select name, title
from (instructor natural join teaches) join course using (course_id);
```

结果相等

目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

更名运算

- The SQL allows renaming relations and attributes using the as clause:
  old_name ***as*** *new_name*

- Find the *name*, *loan_number* and *loan_amount* of all customers, and rename the column name *loan_number* as *loan_id*:

```
select customer_name, borrower.loan_number as loan_id, amount
from  borrower, loan
where borrower.loan_number = loan.loan_number
```

更名运算(续)

- 找出每位老师的姓名和所授所有课程的ID

```
select T.name, S.course_id
from instructor as T, teaches as S
where  T.ID = S.ID
```

- 找出满足下列条件的所有教师的姓名，他们的工资至少比Biology系的某一位教师的工资高

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology'
```

字符串运算

- SQL includes a string-matching operator for comparisons on character strings
  - percent (%): matches any substring
  - underscore (_): matches any character
- like/not like: 找出所在建筑名称中包含子串“Watson”的所有系名

  select *dept_name*
  from *department*
  where *building* ***like*** *‘%Watson%’*
- 要匹配的字符中有百分号的情况，需要转义

  like ‘Watson\%’
- “*” denote “all attributes” : *select instructor.* from instructor*
- SQL supports a variety of string operations such as
  - concatenation（串联） (using “||”)
  - converting from upper case to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

元组的顺序

- List in alphabetic order the names of all customers having a loan in Jiading branch

```
select  distinct customer_name
from    borrower, loan
where borrower.loan_number = loan.loan_number and
        branch_name = 'Jiading'
order by customer_name
```

- Specify desc for descending order or asc (default) for ascending order, for each attribute

```
select *
from loan
order by amount desc, loan-number asc
```

Where子句谓词

- SQL includes a between\not between comparison operator
  - Example: find the names of all instructors with salary between $90,000 and $100,000

```
select name
from instructor
where salary between 90000 and 100000
```

- **Tuple comparison**

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

等价

```
select name, course_id
from instructor, teaches
where instructor.ID=teaches.ID and dept_name='Biology'
```

目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

集合运算

- The set operations union, intersect, and except operate on relations and correspond to the relational algebra operations ∪, ∩, −
- Each set operation automatically eliminates duplicates. To retain all duplicates use the corresponding multiset versions union all, intersect all and except all
  - Suppose a tuple occurs m times in r and n times in s, then, it occurs:
    - $m + n$ times in r union all s
    - $\min(m, n)$ times in r intersect all s
    - $\max(0, m - n)$ times in r except all s

集合运算(续)

- Find all customers who have a loan, an account, or both:

  (select customer_name from depositor)
  union [all]
  (select customer_name from borrower)

- Find all customers who have both a loan and an account.

  (select customer_name from depositor)
  intersect [all]
  (select customer_name from borrower)

- Find all customers who have an account but no loan.

  (select customer_name from depositor)
  except [all]
  (select customer_name from borrower)

目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

空值

- It is possible for tuples to have a null value, signifying an unknown value or a value that does not exist
- The predicate *is null* can be used to check for null values

  select *loan_number*
  from *loan*
  where *amount* ***is*** *null*

- The result of any arithmetic expression involving null is null
  - E.g., 5 + null returns null

空值与聚集操作

- Calculate the sum of all loan amounts

  select sum *(amount)*
  from *loan*

  – Above statement ignores null amounts
  – Result is null if there is no non-null amount
- All aggregate operations except count(∗) ignore tuples with null values on the aggregated attributes

目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

聚集函数(Aggregate Functions)

- These functions receive as input a set of values, and return a value
  - **avg**: average value
  - **min**: minimum value
  - **max**: maximum value
  - **sum**: sum of values
  - **count**: number of values

聚集函数(续)

- 找出Computer Science系教师的平均工资

  select avg *(salary)*
  from *instructor*
  where *dept_name= 'Computer Science'*

- 找出在2018年春季学期授课的教师总数

  select count *(distinct ID)*
  from *teaches*
  where semester = 'Spring' and year=2018

聚集函数：Group By

- Find the number of depositors for each branch

```
select branch_name, count (distinct customer_name)
from depositor, account
where depositor.account_number = account.account_number
group by branch_name
```

- 注意：Attributes in select clause outside of aggregate functions must appear in group by list

```
/*erroneous query*/
select dept_name, ID, avg(salary)
from instructor
group by dept_name
```

聚集函数：Having子句

- At times, it is useful to state a condition that applies to groups rather than to tuples
- 例：find the names of all branches where the average account balance is more than $1,200

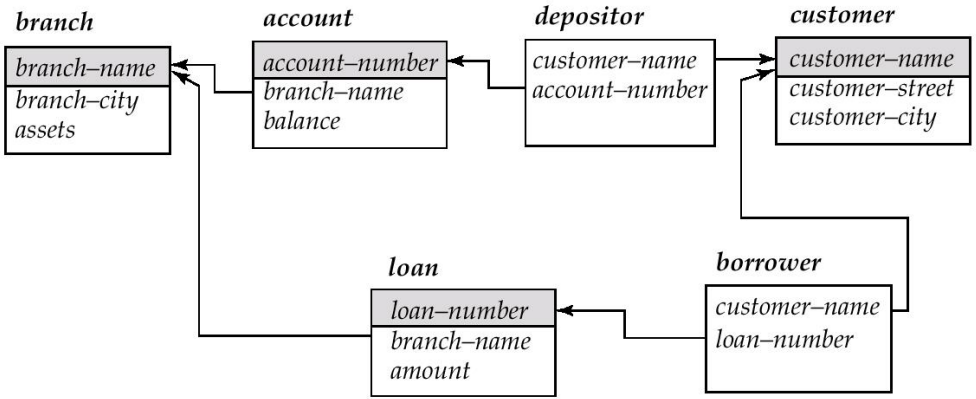
  select *branch_name,* ***avg*** *(balance)*
  from *account*
  group by *branch_name*
  having avg *(balance) > 1200*

- 注意：
  - predicates in the having clause are applied after the information of groups
  - predicates in the where clause are applied before forming groups

聚集函数：Having子句(续)

- 例：find the average balance for each customer who lives in Shanghai and has at least three accounts

```
select depositor.customer_name, avg (balance)
from depositor, account, customer
where depositor.account_number=account.account_number
    and depositer.customer_name=customer.customer_name
    and customer_city='Shanghai'
group by depositor.customer_name
having count(distinct depositor.account_number) >=3
```



目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

嵌套子查询(Nested Subqueries)

- A subquery is a select-from-where expression that is nested within another query in the where clause or from clause
- A common use of subqueries is to perform
  - tests for set membership (测试集合成员资格)
  - make set comparisons (集合比较)
  - determine set cardinality (确定集合基数)

集合包含

- Find all customers who have both an account and a loan at the bank

```
select distinct customer_name
from borrower
where customer_name in (select customer_name
                        from depositor)
```

- Find all customers who have a loan but do not have an account at the bank

```
select distinct customer_name
from borrower
where customer_name not in (select customer_name
                            from depositor)
```

```
select distinct name
from instructor
where name not in ('Mozart', 'Einstein');
```

集合包含(续)

- Find all customers who have both an account and a loan at the Jiading branch

```
select distinct customer_name
from borrower, loan
where borrower.loan_number=loan.loan_number and
        branch_name="Jiading" and
        (branch_name, customer_name) in
        (select branch_name, customer_name
         from depositor, account
         where depositor.account_number =account.account_number)
```

集合比较

- Find all branches that have greater assets than some branch located in Brooklyn

```
select distinct  T.branch_name
from  branch as T, branch as S
where T.assets > S.assets and S.branch_city = 'Brooklyn'
```

- Same query using **>some** clause

```
select branch_name
from  branch
where assets > some
            (select assets
             from branch
             where branch_city = 'Brooklyn')
```

Some子句

- $E < comp > \text{some}\ r \Leftrightarrow \exists\ t \in r$ such that $(E < comp > t)$, where <comp> can be: $<$, $\leq$, $>$, $\geq$, $=$, $\neq$

(5< some | 0 / 5 / 6) = true (read: 5 < some tuple in the relation)

(5< some | 0 / 5) = false

(5 = some | 0 / 5) = true

(5 ≠ some | 0 / 5) = true (since 0 ≠ 5)

(= some) ≡ in
However,
(≠ some) ≢ not in, why?

All子句

- $E < comp > all\ r \Leftrightarrow \forall t \in r\ (E < comp > t)$

(5< all $\begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}$) = false

(5< all $\begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}$) = true

(5 = all $\begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}$) = false

(5 ≠ all $\begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}$) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ all) ≡ not in
However,
(= all) ≠ in, why?

All子句(续)

- Find the names of all branches that have greater assets than all the branches located in Brooklyn.

```
select branch_name
from branch
where assets > all
        (select assets
         from branch
         where branch_city = 'Brooklyn')
```

```
select dept_name
from instructor
group by dept_name
having avg (salary) >= all (select avg (salary)
                            from instructor
                            group by dept_name);
```

思考：左边SQL语句的查询内容是什么?

空关系测试

- The exists construct returns the value true if the argument subquery is nonempty
  - exists $r \Leftrightarrow r \neq \emptyset$
  - not exists $r \Leftrightarrow r = \emptyset$

- E.g., find all the customers who have both account(s) and loan(s) at the bank

```
select distinct customer_name
from borrower
where exists (
    select *
    from depositor
    where depositor.customer_name=borrower.customer_name)
```

空关系测试(续)

- We can write “relation A contains relation B” as “not exists (B except A).”
- E.g., find those students who have taken all the courses offered by the Biology Department

```
select ~~distinct~~ S.ID, S.name
from student as S
where not exists ((select course_id
                   from course
                   where dept_name = 'Biology')
                   except
                   (select T.course_id
                   from takes as T
                   where S.ID = T.ID));
```

空关系测试(续)

- Find all the customers who have accounts at all branches located in Brooklyn

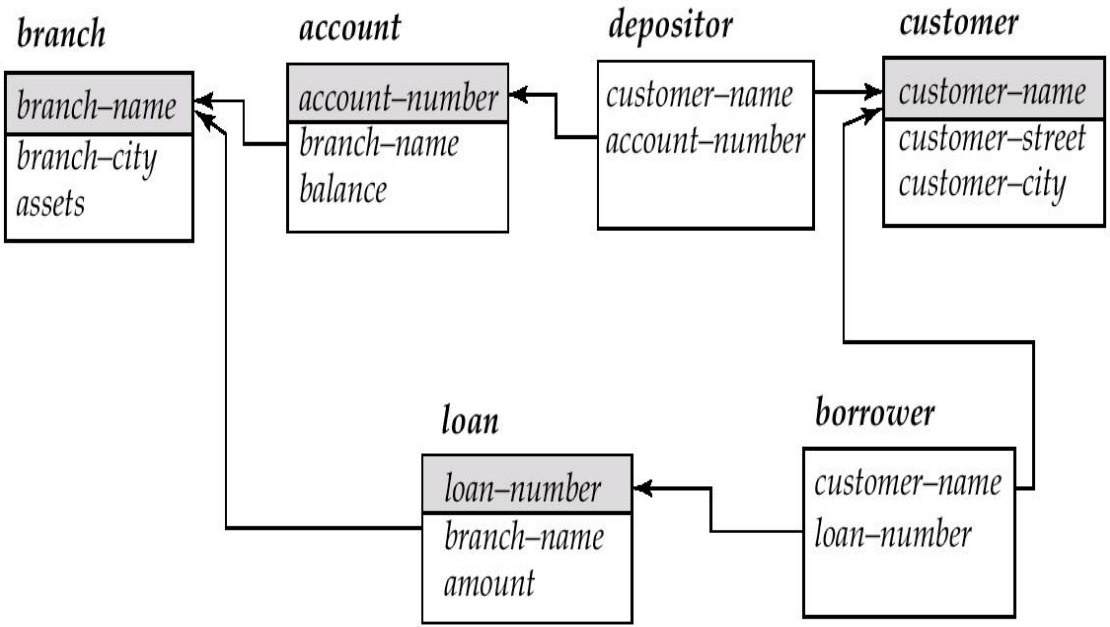
```
select distinct S.customer_name
from depositor as S
where not exists (
        (select branch_name      /* all branches in Brooklyn  */
         from branch
         where branch_city = 'Brooklyn')
        except
        (select R.branch_name   /* finds all the branches at which customer
S.customer_name has an account */
         from depositor as T, account as R
         where T.account_number = R.account_number and
               T.customer_name = S.customer_name))
```

- **Note**: *not exists* $(X - Y) \Leftrightarrow X - Y = \emptyset \Leftrightarrow X \subseteq Y$

重复元组测试

- The unique construct tests whether a subquery has any duplicate tuples in its result
- E.g., find all the customers who have at most one account at the Jiading branch

```
select T.customer_name
from depositor as T      /* Better to use from customer. why? */
where unique (
    select R.customer_name
    from account, depositor as R
    where T.customer_name = R.customer_name and
          R.account_number = account.account_number and
          account.branch_name = 'Jiading')
```


branch
branch–name
branch–city
assets
account
account–number
branch–name
balance
depositor
customer–name
account–number
customer
customer–name
customer–street
customer–city
loan
loan–number
branch–name
amount
borrower
customer–name
loan–number

例

- **找出所有在2017年最多开设一次的课程**

```
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id and
                    R.year = 2017);
```

等价

```
select T.course_id
from course as T
where 1 >= (select count(R.course_id)
            from section as R
            where T.course_id= R.course_id and
                  R.year = 2017);
```

例

- Find all customers who have at least two accounts at the Jiading branch.

```
select distinct T.customer_name
from depositor T
where not unique(
        select R.customer_name
        from account, depositor as R
        where T.customer-name = R.customer_name and
              R.account-number = account.account_number and
            account.branch_name = 'Jiading')
```

```
select T.course_id
from course as T
where not unique (select R.course_id
                  from section as R
                  where T.course_id= R.course_id and
                        R.year = 2017);
```

找出所有在2017年至少开设两次的课程

思考：找出所有在2017年至少开设三次的课程?

视图(View)

- In some cases, it is not desirable for all users to see the entire logical model, i.e., all the actual relations stored in the database
- Consider a person who needs to know a customer's name, loan number and branch name, but has no need to see the loan amount

```
select customer_name, borrower.loan_number, branch_name
from borrower, loan
where borrower.loan_number = loan.loan_number
```

- A view provides a mechanism to hide certain data from the view of certain users. It's a **virtual relation**

视图的定义

- A view is defined using the create view statement

  create view *v* ***as*** < *query expression* >

  where *v* is the view name, and <query expression> is any legal SQL expression

- Once a view is defined, the view name can be used to refer to the virtual relation, and the query expression is stored in the database. The expression is substituted into queries when the view is used.

视图：举例

- A view consisting of branches and their customers

```
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)
```

- Find all customers of the Jiading branch

```
select customer_name
from all_customer
where branch_name = 'Jiading'
```

导出关系(Derived Relations)

- **Derived Relations**
  - Find the average account balance of those branches which have the average account balance greater than $1200

```
select branch_name, avg (balance)
from account
group by branch_name
having avg (balance) > 1200
```

```
select branch_name, avg_balance
from (select branch_name, avg (balance)
      from account
      group by branch_name)
      as result (branch_name, avg_balance)
where avg_balance > 1200
```

  - **Note**: we do not need to use the having clause, since we compute the temporary (view) relation result in the from clause, and the attributes of result can be used directly in the where clause

导出关系(续)

- Find the maximum total balance across all branches

```
select max(tot_balance)
from (select branch_name, sum (balance)
        from account
        group by branch_name)
        as branch_total (branch_name, tot_balance)
```

With子句

- With clause allows views to be defined locally to a query, rather than globally. Analogous to procedures in a programming language
- Find all accounts with the maximum balance

```
with max_balance(value) as
    select max(balance)
    from account

select account_number
from account, max_balance
where account.balance = max_balance.value
```

使用with子句的复杂查询

- Find all branches where the total account deposit is greater than the average of the total account deposits at all branches

```
with branch_total (branch_name, value) as
select branch_name, sum (balance)
from account
group by branch_name

with branch_total_avg (value) as
select avg (value)
from branch_total

select branch_name
from branch_total, branch_total_avg
where branch_total.value >= branch_total_avg.value
```

标量子查询(Scalar Subquery)

- Scalar subquery (标量子查询) is used where a single value is expected
- List all departments along with the number of instructors in each department

```
select dept_name,
    (select count(*)
    from instructor
    where department.dept_name = instructor.dept_name)
    as num_instructors
from department;
```

- Runtime error if subquery returns more than one tuple

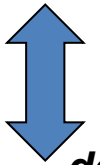
目录

- SQL概览
- SQL数据定义
- SQL查询的基本结构
- 附加基本运算
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

数据库的修改：Deletion

- Delete all accounts at every branch located in Needham city

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



```
delete from depositor
where account_number in
        (select account_number
         from branch, account
         where branch_city = 'Needham'
         and branch.branch_name = account.branch_name)
```

思考：删除操作是否存在问题?

数据库的修改：Deletion(续)

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

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

- **Note**: as we delete tuples from account, the average balance changes
- Solution used in SQL:
  - First, compute avg balance and find all tuples to delete
  - Next, delete all tuples found above (without recomputing avg or retesting the tuples)

数据库的修改：Insertion

- Add a new tuple to account

  insert into *account* ***values*** *('A-9732', 'Perryridge',1200)*

  or equivalently

  insert into *account (branch_name, balance, account_number)*
  values *('Perryridge', 1200, 'A-9732')*

- Add a new tuple to account with balance set to null

  insert into *account*
  values *('A-777','Perryridge', null)*

数据库的修改：Insertion(续)

- Provide as a gift for all loan customers of the Jiading branch, i.e., a $200 saving account. Let the loan number serve as the account number for the new saving account

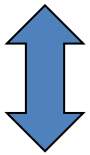
```
insert into account
    select loan_number, branch_name, 200
    from loan
    where branch_name = 'Jiading'
insert into depositor
    select customer_name, loan_number
    from loan, borrower
    where loan.loan_number = borrower.loan_number
        and branch_name = 'Jiading'
```

- The select from where statement is **fully evaluated** before its results are inserted into the relation. Otherwise, queries like *insert into table1 select * from table2* would cause problems

数据库的修改：Updates

- Increase all accounts with balances over $10,000 by 6%, and all other accounts receive an increase of 5%.
  - Write two update statements:

update *account*
set *balance = balance * 1.05*
where *balance ≤ 10000*



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

思考：插入操作是否存在问题?

- The **order** is important
- Can be done better using the case statement (next slide)

Case表达式

- Same query: increase all accounts with balances over $10,000 by 6%, and all other accounts receive 5%.

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

```
case
    when pred_1 then result_1
    when pred_2 then result_2
    ...
    when pred_n then result_n
    else result_0
end
```

小结

- **SQL**
  - DDL + DML
- **DDL**
  - 数据库模式、完整性约束等
- **SQL查询**
  - select子句、from子句、where子句
  - natural join, join … using (…)
- **SQL附加运算**
  - 更名运算、字符串运算、排序order by
- **集合运算**
  - union、intersect、except
- **空值**
- **聚集函数**
  - avg、min、max、sum、count
  - group by、having
- **嵌套子查询**
  - 集合包含：in、not in
  - 集合比较：some子句、all子句、exists、unique
  - 视图、导出关系、with子句、标量子查询
- **数据库的修改**
  - Deletion、Insertion、Updates

作业

- **Exercises**
  - 3.8, 3.9 (选择其中一个)
  - 3.15, 3.16, 3.17, 3.21 (选择其中两个)
- **Submission**
  - Canvas上提交，上传单个PDF文件
  - Deadline: 待定