



# 3 SQL进阶

{课本第4、5章}

刘跃文 博士，副教授

西安交通大学管理学院  
信息管理与电子商务系

[liuyuewen@mail.xjtu.edu.cn](mailto:liuyuewen@mail.xjtu.edu.cn)

V1, 2018-12-2

# 提纲

- 4.1 Join Expressions 连接查询 [掌握]
- 4.2 Views 视图 [掌握]
- 4.3 Transactions 事务
- 4.4 Integrity Constraints 完整性约束
- 4.5 SQL Data Types and Schemas 数据类型与模式
- 4.6 Authorization 授权

- 5.1 Accessing SQL From a Programming Language [了解]
  - Dynamic SQL
    - JDBC and ODBC
  - Embedded SQL
- 5.2 Functions and Procedural Constructs [了解]
- 5.3 Triggers [了解]
- 5.4 Advanced Aggregation Features [了解]
- 5.5 OLAP [了解]

## 4.1 Joined Relations 连接查询 [掌握]

- **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.

*Join types*

**inner join**

**left outer join**

**right outer join**

**full outer join**

# Example [掌握]

## A

ID	Name
1	A
2	B
3	C

## B

ID	Name
2	B
3	C
4	D

- select \* from A **inner join** B on A.ID=B.ID

ID	Name	ID	Name
2	B	2	B
3	C	3	C

- select \* from A **full outer join** B on A.ID=B.ID

ID	Name	ID	Name
1	A	null	null
2	B	2	B
3	C	3	C
null	null	4	D

- select \* from A **left outer join** B on A.ID=B.ID

ID	Name	ID	Name
1	A	null	null
2	B	2	B
3	C	3	C

- select \* from A **right outer join** B on A.ID=B.ID

ID	Name	ID	Name
2	B	2	B
3	C	3	C
null	null	4	D

## 4.2 Views 视图 [掌握]

- 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” 虚拟关系 is called a **view**.



# View Definition 定义视图 [掌握]

- 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.

# 视图的几个主要作用

## 视图的特点：

- 1 没有实际数据
- 2 视图名可以像表名一样被直接查询

## 视图的好处：

- 1 在仅存一份原始数据的基础上，提供各种数据使用方式，不会带来数据冗余
- 2 保护原始数据，不被非法使用
- 3 视图可以像表一样被操作，因此增加数据使用的便利性

# Example Views [理解]

- View的定义与使用
- A view of instructors without their salary (保护隐私)  
**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* **as**  
    **select** *dept\_name, sum (salary) as total\_salary*  
    **from** *instructor*  
    **group by** *dept\_name*;

# Views Defined Using Other Views [理解]

- 基于view创建新的view
- **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'* ;

## 4.3 Transactions 事务 [理解]

- Unit of work
- Atomic transaction 事务原子性 [或者不做， 或者做完]
  - 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`
- Not supported on most databases
- Yuewen Add:
  - 数据库是一门工程技术， 因此， 不同的软件实现方式都不同
  - 不用记得每一个语法细节， 在实践时能找到解决方案即可

## 4.4 Integrity Constraints 完整性约束 [理解]

- 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

## 单个关系上的完整性约束 [部分掌握]

- **not null**
- **primary key**
- **unique**
- **check** (P), where P is a predicate

# Not Null and Unique Constraints 非空和唯一性约束 [掌握 not null]

- **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** or a **super key**?
- Candidate/Super keys are permitted to be null (in contrast to primary keys).



# The check clause

## check子句 [了解]

- **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

## 参照完整性 [了解]

- 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.
- Yuewen Add:
  - 参照完整性和外键约束的区别：参照完整性不要求所参照属性是源表的主键。
  - 唯一性约束和主键约束的区别：唯一性约束不要求值非空。

## 4.5 Built-in Data Types in SQL

### 其它数据类型 [了解]

- **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'

# Yuewen Add : 日期类函数

- DATEADD在向指定日期加上一段时间的基础上，返回新的datetime值。
- DATEDIFF返回跨两个指定日期的日期和时间边界数
- DATENAME返回代表指定日期的指定日期部分的字符串
- DATEPART返回代表指定日期的指定日期部分的整数
- DAY返回代表指定日期的天的日期部分的整数
- GETDATE按datetime值的Microsoft® SQL Server™标准内部格式返回当前系统日期和时间
- GETUTCDATE返回表示当前UTC时间（世界时间坐标或格林尼治标准时间）的datetime值
- MONTH返回代表指定日期月份的整数
- YEAR返回表示指定日期中的年份的整数

# 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*

## 4.6 Authorization 授权 [了解]

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

## SQL中的授权声明 [了解]

- 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

## SQL中的权限 [理解]

- **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

## 收回权限 [理解]

- 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 角色 [理解]

- **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;

## 5.1 JDBC and ODBC [了解]

- API (application-program interface) for a program to interact with a database server
- Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API' s such as ADO.NET sit on top of ODBC
- JDBC (Java Database Connectivity) works with Java

# JDBC

- **JDBC** is a Java API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:
  - Open a connection
  - Create a “statement” object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

# JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
            "jdbc:oracle:thin:@db.yale.edu:2000:univdb", userid, passwd);
        Statement stmt = conn.createStatement();
        ... Do Actual Work ...
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

## JDBC Code (Cont.)

- Update to database

```
try {  
    stmt.executeUpdate(  
        "insert into instructor values(' 77987' , ' Kim' , ' Physics' ,  
698000)");  
} catch (SQLException sqle)  
{  
    System.out.println("Could not insert tuple. " + sqle);  
}
```

- Execute query and fetch and print results

```
ResultSet rset = stmt.executeQuery(  
    "select dept_name, avg (salary)  
    from instructor  
    group by dept_name");  
while (rset.next()) {  
    System.out.println(rset.getString("dept_name") + " " +  
        rset.getFloat(2));  
}
```

# SQL Injection SQL注入攻击 [了解]

- Suppose query is constructed using
  - "select \* from instructor where name = ' " + name + "' "
- Suppose the user, instead of entering a name, enters:
  - X' or ' Y' = ' Y
- then the resulting statement becomes:
  - "select \* from instructor where name = ' " + "X' or ' Y' = ' Y" + "' "
  - which is:
    - select \* from instructor where name = ' X' or ' Y' = ' Y'
  - User could have even used
    - X' ; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:  
"select \* from instructor where name = ' X\ ' or \ ' Y\ ' = \ ' Y'
  - **Always use prepared statements, with user inputs as parameters**

# ODBC [了解]

- Open DataBase Connectivity(ODBC) standard
  - standard for application program to communicate with a database server.
  - application program interface (API) to
    - open a connection with a database,
    - send queries and updates,
    - get back results.
- Applications such as GUI, spreadsheets, etc. can use ODBC
- Was defined originally for Basic and C, versions available for many languages.



## 5.2 Procedural Extensions and Stored Procedures

### 存储过程 [了解]

- SQL provides a **module** language
  - Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc.
- Stored Procedures
  - Can store procedures in the database
  - then execute them using the **call** statement
  - permit external applications to operate on the database without knowing about internal details
- Object-oriented aspects of these features are covered in Chapter 22 (Object Based Databases)

# Functions and Procedures

## 函数和存储过程 [了解]

- SQL:1999 supports functions and procedures
  - Functions/procedures can be written in SQL itself, or in an external programming language.
  - Functions are particularly useful with specialized data types such as images and geometric objects.
    - Example: functions to check if polygons overlap, or to compare images for similarity.
  - Some database systems support **table-valued functions**, which can return a relation as a result.
- SQL:1999 also supports a rich set of imperative constructs, including
  - Loops, if-then-else, assignment
- Many databases have proprietary procedural extensions to SQL that differ from SQL:1999.

# SQL Functions

## 函数 [了解]

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

```
create function dept_count (dept_name varchar(20))  
returns integer  
begin  
  declare d_count integer;  
  select count (*) into d_count  
  from instructor  
  where instructor.dept_name = dept_name  
  return d_count;  
end
```

- Find the department name and budget of all departments with more that 12 instructors.

```
select dept_name, budget  
from department  
where dept_count (dept_name) > 1
```

# Table Functions 表函数 [了解]

- SQL:2003 added functions that return a relation as a result
- Example: Return all accounts owned by a given customer

```
create function instructors_of(dept_name char(20))
```

```
  returns table (ID varchar(5),  
                name varchar(20),  
                dept_name varchar(20),  
                salary numeric(8,2))
```

```
return table
```

```
(select ID, name, dept_name, salary  
  from instructor  
  where instructor.dept_name = instructors_of.dept_name)
```

- Usage

```
select *  
from table (instructors_of( 'Music' ))
```

# SQL Procedures

## 存储过程 [了解]

- The *dept\_count* function could instead be written as procedure:

```
create procedure dept_count_proc (in dept_name varchar(20), out d_count integer)
begin
  select count(*) into d_count
  from instructor
  where instructor.dept_name = dept_count_proc.dept_name
end
```

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the **call** statement.

```
declare d_count integer;
call dept_count_proc( 'Physics' , d_count);
```

Procedures and functions can be invoked also from dynamic SQL

- SQL:1999 allows more than one function/procedure of the same name (called name **overloading**), as long as the number of arguments differ, or at least the types of the arguments differ

## 5.5 Ranking 排名 [理解]

- Ranking is done in conjunction with an order by specification.
- Suppose we are given a relation *student\_grades*(*ID*, *GPA*) giving the grade-point average of each student
- Find the rank of each student.

```
select ID, rank() over (order by GPA desc) as s_rank  
from student_grades
```

- An extra **order by** clause is needed to get them in sorted order

```
select ID, rank() over (order by GPA desc) as s_rank  
from student_grades  
order by s_rank
```

- Ranking may leave gaps: e.g. if 2 students have the same top GPA, both have rank 1, and the next rank is 3
  - **dense\_rank** does not leave gaps, so next dense rank would be 2

# Ranking (Cont.)

- Ranking can be done within partition of the data.
- “Find the rank of students within each department.”

```
select ID, dept_name,  
        rank () over (partition by dept_name order by GPA desc)  
        as dept_rank  
from dept_grades  
order by dept_name, dept_rank,
```

- Multiple **rank** clauses can occur in a single **select** clause.
- Ranking is done *after* applying **group by** clause/aggregation
- Can be used to find top-*n* results
  - More general than the **limit** *n* clause supported by many databases, since it allows top-*n* within each partition

# Ranking (Cont.)

- Other ranking functions:
  - **percent\_rank** (within partition, if partitioning is done)
  - **cume\_dist** (cumulative distribution)
    - fraction of tuples with preceding values
  - **row\_number** (non-deterministic in presence of duplicates)
- SQL:1999 permits the user to specify **nulls first** or **nulls last**  

```
select ID,  
       rank ( ) over (order by GPA desc nulls last) as s_rank  
from student_grades
```
- For a given constant  $n$ , the ranking the function  $ntile(n)$  takes the tuples in each partition in the specified order, and divides them into  $n$  buckets with equal numbers of tuples.
- E.g.,  

```
select ID, ntile(4) over (order by GPA desc) as quartile  
from student_grades;
```



## 5.6 Data Analysis and OLAP [了解]

- **Online Analytical Processing (OLAP)**

- Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)
- Data that can be modeled as dimension attributes and measure attributes are called **multidimensional data**.
  - **Measure attributes**
    - measure some value
    - can be aggregated upon
    - e.g., the attribute *number* of the *sales* relation
  - **Dimension attributes**
    - define the dimensions on which measure attributes (or aggregates thereof) are viewed
    - e.g., attributes *item\_name*, *color*, and *size* of the *sales* relation

# Example sales relation

<i>item_name</i>	<i>color</i>	<i>clothes_size</i>	<i>quantity</i>
skirt	dark	small	2
skirt	dark	medium	5
skirt	dark	large	1
skirt	pastel	small	11
skirt	pastel	medium	9
skirt	pastel	large	15
skirt	white	small	2
skirt	white	medium	5
skirt	white	large	3
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
shirt	dark	small	2
shirt	dark	medium	6
...	...	...	...
...	...	...	...

# Cross Tabulation of sales by item\_name and color

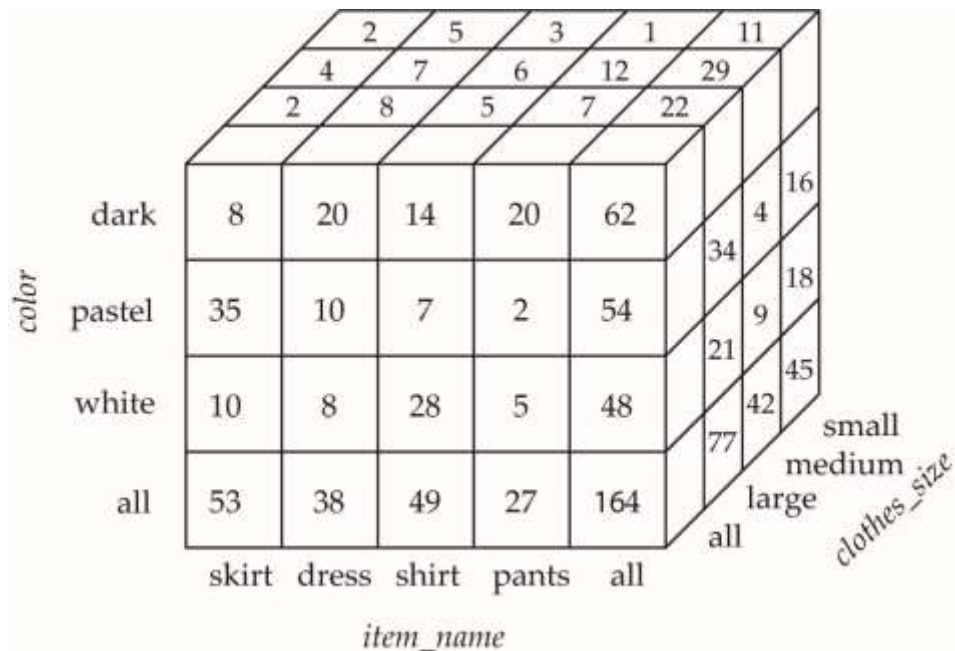
- The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.
  - Values for one of the dimension attributes form the row headers
  - Values for another dimension attribute form the column headers
  - Other dimension attributes are listed on top
  - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.

*clothes\_size* **all**

		<i>color</i>			
		dark	pastel	white	total
<i>item_name</i>	skirt	8	35	10	53
	dress	20	10	5	35
	shirt	14	7	28	49
	pants	20	2	5	27
	total	62	54	48	164

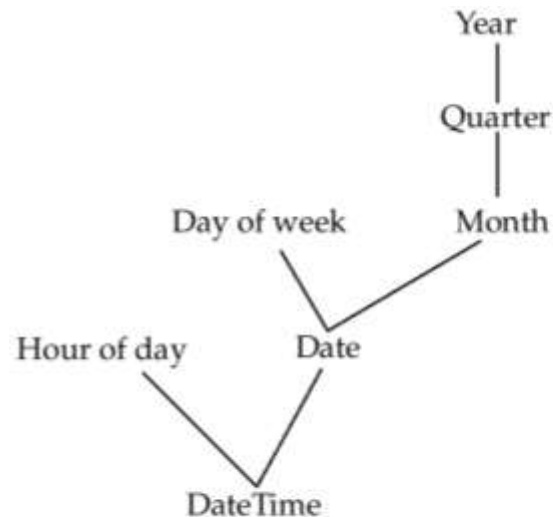
# Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have  $n$  dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube

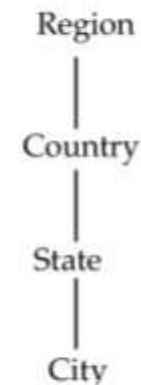


# Hierarchies on Dimensions

- Hierarchy on dimension attributes: lets dimensions to be viewed at different levels of detail
  - E.g., the dimension DateTime can be used to aggregate by hour of day, date, day of week, month, quarter or year



a) Time Hierarchy



b) Location Hierarchy

# Cross Tabulation With Hierarchy

- Cross-tabs can be easily extended to deal with hierarchies
  - Can drill down or roll up on a hierarchy

*clothes\_size:* **all**

<i>category</i>	<i>item_name</i>	<i>dark</i>	<i>pastel</i>	<i>white</i>	<i>total</i>	
womenswear	skirt	8	8	10	53	
	dress	20	20	5	35	
	subtotal	28	28	15		88
menswear	pants	14	14	28	49	
	shirt	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164


# Online Analytical Processing Operations

- **Pivoting:** changing the dimensions used in a cross-tab is called
- **Slicing:** creating a cross-tab for fixed values only
  - Sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- **Rollup:** moving from finer-granularity data to a coarser granularity
- **Drill down:** The opposite operation - that of moving from coarser-granularity data to finer-granularity data

# 小结

- join: inner join, left outer join, right outer join, full outer join, on
- 视图: 视图的涵义, create view
- 授权与收回权限：
  - **grant select on *instructor* to  $U_1, U_2, U_3$**
  - **revoke select on *branch* from  $U_1, U_2, U_3$**





谢谢！

liuyuewen@xjtu.edu.cn