Chapter 5: Advanced SQL

Database System Concepts, 7th Ed.

Outline

- Functions and Procedures
- Triggers
- Recursive Queries
- Advanced Aggregation Features

Functions and Procedures

Functions and Procedures

- Functions and procedures allow "business logic" to be stored in the database and executed from SQL statements.
- These can be defined either by the procedural component of SQL or by an external programming language such as Java, C, or C++.
- The syntax we present here is defined by the SQL standard.
  - Most databases implement nonstandard versions of this syntax.

-
با فانکشن می تونیم از کارای تکراری جلوگیری کنیم
اینجا می خوایم خودمون یه سری فانکشن بنویسیم
توی فصل سوم یه سری فانکشن هایی دیدیم همون دو صفحه ای میشه که شبیه جدول بود
تفاوت بین function , procedures : تفاوت اصلی این است که توی procedures ما اجازه تغییر دادن داده های جدول ها رو داریم ینی این امکان رو به ما می ده که اپدیت بکنیم داده های یک جدول را ولی توی فانکشن ما اجازه تغییر داده ها رو نداریم

Declaring SQL Functions

- 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
```

تایپ داده ای که می خواد توی خروجی برگردونده بشه رو اینجا می نویسیم

اینجا یه متغییر تعریف شده به اسم d_count از نوع integer نی با دستور declare می تونیم متغییر تعریف کنیم

نکته: توی بدنه فانکشن ما اجازه دسترسی به جداول مختلف پایگاه داده و ویوهای مختلف داخل پایگاه داده رو داریم ولی تغییری نمی تونیم روی داده های جداول انجام بدیم

با into d_count میاد مقدار رو پاس میده توی متغییرمون

-
مثال: اسم دانشکده رو به عنوان ورودی به فانکشن بدیم و تعداد اساتید اون دانشکده رو به ما بده

Table Functions

- The SQL standard supports functions that can return tables as results; such functions are called **table functions**
- Example: Return all instructors in a given department

```
create function instructor_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 = instructor_of.dept_name)
```

این نوع خروجی ما است

- Usage

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

به این صورت از فانکشن استفاده میکنیم

نکته: فانکشن رو می تونیم توی select فراخوانی بکنیم یا توی where استفاده بکنیم

-
table function به این معناست که فانکشن ما دیگه نمیخواد یک متغییر از جنس integer یا string رو برگردونه قراره یک جدول توی خروجی به ما بده

این توی محیط sql server است

تابع اسکالر: این توابع تک خروجی هستند

```
CREATE  FUNCTION [ owner_name. ] function_name
   ( @parameter_name [AS] data_type )
RETURNS data_type
 AS
BEGIN

   function_body

   RETURN scalar_expression
END
```

تابعی که n را دریافت کرده و n*(n-1) را خروجی می دهد.

```
create function [dbo].[mul] (@n int)
returns bigint
AS
begin
 declare @r bigint
  set @r=(@n-1)*@n
 return @r
 end
```

اسم فانکشن

تایپ داده ای که میخواد برگردونه bigint است

```
select [dbo].[mul](10) as result
```

00 %

Results | Messages

	result
1	90

با set می تونیم توی متغییر مقدار بریزیم

تابعی که شماره یک دانشجو راگرفته و نام ونام خانوادگی وی را نمایش می دهد .

```
Create FUNCTION fn_Nameret(@s# int)
 RETURNS NVARCHAR(40)
AS
BEGIN
        DECLARE @ret_Value  NVARCHAR(40);
        set @ret_Value = (SELECT  Name + ',' +Family
        FROM STD WHERE S# =@S#)

        RETURN (@ret_Value)
End
```

نکته: توی فانکشن اجازه دسترسی به جداول رو داریم

توابع table-valued: یک جدول خروجی می دهند.

ینی خروجی یک تک مقدار نیست

```
CREATE  FUNCTION [ owner_name. ] function_name
   ( @parameter_name [AS] data_type )
RETURNS table
 AS
BEGIN
   function_body
   RETURN (select)
END
```

تفاوت اصلی این تعریف با توابع اسکالر در این است که نوع خروجی یک جدول تعریف شده است.
محدودیتی که روی این نوع تعریف وجود دارد این است که خروجی تابع باید توسط یک دستور select ایجاد شود.

در این مثال ما یک تابع با مقدار جدول درون خطی ایجاد خواهیم کرد که رکوردهای تمام دانش آموزانی را که DOB آنها کمتر از DOB ارسال شده به تابع است، بازیابی می کند.

In this example we will create an inline table-valued function that will retrieve records of all the students whose DOB is less than the DOB passed to the function.

```
CREATE FUNCTION BornBefore
 (
 @DOB AS DATETIME
 )
RETURNS TABLE
AS
BEGIN
        RETURN
        SELECT * FROM student
        WHERE DOB < @DOB
END
```

توی این مثال میگه یه تاریخ تولد به عنوان ورودی بگیر و بیا تمام دانشجویانی که تاریخ تولدشون کمتر از اون مقدار وارد شده است توی خروجی به ما بده

We will pass a date to "BornBefore" function. The student records retrieved by the function will have a DOB value lesser than that date.

از فانکشن صفحه قبل به این صورت استفاده میکنیم:

```
SELECT
        name, gender, DOB
FROM
        dbo.BornBefore('1980-01-01')
ORDER BY
        DOB
```

ما یک تاریخ را به تابع "BornBefore" ارسال می کنیم. سوابق دانشجویی بازیابی شده توسط تابع دارای مقدار DOB کمتر از آن تاریخ خواهد بود.

SQL Procedures

مثال: تعداد دانشکده رو به عنوان ورودی میگیره و تعداد استادان اون دانشکده رو به ما میده این دفعه با procedures می خوایم حل کنیم

- 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
```

متغییرهای ورودی رو با عبارت in مشخص میکنیم و متغییرهای خروجی رو با عبارت out مشخص میکنیم

- The keywords **in** and **out** are parameters that are expected to have values assigned to them and parameters whose values are set in the procedure in order to return results.
- 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);
```

به این صورت procedures رو فراخوانی میکنیم

توی این متغییر مقدار خروجی قرار میگیره

-
ما اجازه نداریم procedures روی توی select استفاده بکنیم
با دستور call می تونیم بیایم procedures رو فراخوانی بکنیم
با procedures می تونیم بیایم عملیات های اپدیت و دیلیت و insert رو انجام بدیم روی داده

Language Constructs (Cont.)

- **For** loop ما چه داخل فانکشن و چه procedures اجازه نوشتن لوپ رو داریم
  - Permits iteration over all results of a query
- Example: Find the budget of all departments

پس for زمانی استفاده میشود که انگار ما یه سری موجودیت هایی داریم که میخوایم روی هر کدوم از این موجودیت ها یه سری تغییراتی ایجاد شود مثلا یه سری مقدار توسط select داریم بعد این مقدار می ره توی r حالا عملیات ما روی این r انجام میشه هر بار

```
declare n integer default 0;
for r as
        select budget from department
        where dept_name = 'Music'
do
        set n = n + r.budget
end for
```

میتونیم توی این عملیاتی که مد نظرمان است انجام بشه

کل این با هم اجرا میشه ینی اگر بار اول r=10 باشه و مثلا توی این مرحله n=0 باشه پس n جدید میشه 10 بعد بار بعدی اگه توی حلقه r بشه مثلا 11 مقدار جدید n ما میشه 10+11 پش کل این بدنه با هم یه بار اجرا میشود و دوباره به همین صورت عمل میکند تا مقدار r تمام شود

نحوه استفاده کردن از for به این صورت است که:
r اندیس برای for است
بعد از as یک select داریم که این داره یه تعدادی رکوردی رو به ما برمیگردونه این تعداد رکورد باعث میشه لوپ ما هم به همون تعداد اجرا بشه

Language Constructs – if-then-else

ما چه داخل فانکشن و چه procedures می تونیم از if-then-else استفاده بکنیم

- Conditional statements (**if-then-else**)

if *boolean expression*
then *statement or compound statement*
elseif *boolean expression*
then *statement or compound statement*
else *statement or compound statement*
end if

مثل همون برنامه نویسی است

Procedure

- A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs.

- Stored procedures can access or modify data in a database, but it is not tied to a specific database or object, which offers a number of advantages.

Benefits of using stored procedures

- A stored procedure provides an important layer of security between the user interface and the database. It supports security through data access controls because end users may enter or change data, but do not write procedures.

- A stored procedure preserves data integrity because information is entered in a consistent manner. It improves productivity because statements in a stored procedure only must be written once.

-
مزایای stored procedures:
یک لایه ؟ برای DBMS میایم ایجاد میکنیم ؟
وقتی procedures می نویسیم می تونیم انواع چک کردن ها رو توی کد procedures داشته باشیم و اگر کاربر می خواد داده می خواد وارد کنه یا حذف کنه یا .. ینی هر عملیاتی که منحر به تغییر اطلاعات توی پایگاه داده میشه بیاد کنترل های خاص خود رو توی procedures داشته باشیم

Benefits of using stored procedures (Cont.)

- Stored procedures offer advantages over embedding queries in a graphical user interface (GUI). Since stored procedures are modular, it is easier to troubleshoot when a problem arises in an application.

- Stored procedures are also tunable, which eliminates the need to modify the GUI source code to improve its performance. It's easier to code stored procedures than to build a query through a GUI.

نواشتن پراسیجر

اینجا محیط sql server است

```
CREATE  PROCEDURE [ owner_name. ] procedure_name
   ( @parameter_name [AS] data_type )
 AS
BEGIN

   procedure_body

END
```

این پراسیجر سه تا مقدار رو به عنوان ورودی گرفته و می خواد این سه مقدار رو توی جدول insert بکنه

مثال ۱

```
1. CREATE PROCEDURE stpInsertMember
2. @MemberName varchar(50),
3. @MemberCity varchar(25),
4. @MemberPhone varchar(15)
5. AS
6. BEGIN
7.    Insert into tblMembers (MemberName,MemberCity,MemberPhone)
                  Values (@MemberName,@MemberCity, @MemberPhone)

8.   END
```

توی جدول tblMembers این سه مقدار رو میخواد بریزه

مثال ۲

```
1. CREATE PROCEDURE stpUpdateMemberByID
2. @MemberID int,
3. @MemberName varchar(50),
4. @MemberCity varchar(25),
5. @MemberPhone varchar(15)
6.
7. AS
8. BEGIN
9.     UPDATE tblMembers
10.     Set MemberName = @MemberName,
11.         MemberCity = @MemberCity,
12.         MemberPhone = @MemberPhone
13.     Where MemberID = @MemberID
14. END
```

اینجا داریم داده های جدول رو اپدیت میکنیم

فراخوانی

```
1.  CREATE PROCEDURE stpUpdateMemberByID
2.  @MemberID int,
3.  @MemberName varchar(50),
4.  @MemberCity varchar(25),
5.  @MemberPhone varchar(15)
6.  AS
7.  BEGIN
8.      UPDATE tblMembers
9.      Set MemberName = @MemberName,
10.         MemberCity = @MemberCity,
11.         MemberPhone = @MemberPhone
12.     Where MemberID = @MemberID
13. END
```

یعنی اگر شرط برقرار بود اپدیت رخ میده

```
EXEC  stpUpdateMemberByID @MemberID = 10,
 @MemberName = 'Ali' , @MemberCity = ‘Tehran‘,
@MemberPhone =‘0912912912‘;
```

سوال

- تابعی بنویسید که یک رشته به طول حداکثر ۲۰ و یک الگو به طول حداکثر ۳ دریافت نماید. این تابع باید تمام تکرارهای الگو را از رشته بزرگتر حذف کرده و رشته باقی مانده را در برگرداند. اگر با حذف یکباره دوباره رشته شامل آن الگو بود باید مجدد حذف شود.

- تابعی را در SQL SERVER بنویسید که مشابه تابع LPAD در اوراکل کار کند.

Triggers

Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
  - Syntax illustrated here may not work exactly on your database system; check the system manuals

-
یه سری سازوگاری های داده ای رو بیایم با استفاده از triggers چک بکنیم و کنترل بکنیم
هر وقت ما بیایم داده رو توی حدول اپدیت یا insert یا دیلیت بکنیم ینی تغییری روی داده های یک جدول ایجاد کنیم پایگاه داده این امکان رو به ما میده که یک اسکریپت بنویسیم که به عنوان triggers شناخته میشه و با استفاده از اون اسکریپت بیایم کنترل بکنیم داده هایی رو که توی یک جدول داره insert یا اپدیت یا دیلیت میشه که این کنترل شدن فقط خاص اون جدول نیست ینی توی اون اسکریپت می تونیم هر کاری انجام بدیم
کلا می تونیم شرایط پایگاه داده رو کنترل و چک بکنیم و اگر دیتای معقولی می خواست insert بشه داده رو اجازه داره insert بشه یا مثلا اگه می خواد دیلیت بشه داده رو خراب نمی کنه و واسه اپدیت هم همینه
کلا دوتا کار میشه انجام داد:
1- قبل از دیلیت و اپدیت و insert یک چکی رو انجام بدیم ینی دیلیت یا اپدیت یا insert فراخوانی شده ولی قبل از اینکه شروع به کار کنه اون اسکریپت رو فراخوانی میکنه که بیاد یه چکی انجام بده
2- یا اینکه بعد از اینکه اپدیت یا دیلیت یا insert تموم شد بیاد اون اسکریپت رو فراخوانی بکنه و چک بکنه

Triggering Events and Actions in SQL

برای اپدیت ما برای مقدار های قدیمی + قرار میدیم و برای مقدارهای جدید / رو قرار میدیم

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - For example, **after update of** *takes* **on** *grade*
- Values of attributes before and after an update can be referenced
  - **referencing old row as** : for deletes and updates
  - **referencing new row as** : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. For example, convert blank grades to null.

به ازای هر رکوردی که داره براش اپدیت انجام میشه میگه اگه مقدار grade توی رکورد جدید خالی بود مقدار grade شو برابر با null بذار

```
create trigger setnull_trigger before update of takes
referencing new row as nrow
for each row
    when (nrow.grade = ' ')
    begin atomic
        set nrow.grade = null;
end;
```

اسم رکورد های جدید رو nrow گذاشته

میگه قبل از اینکه جدول takes اپدیت بیشه این تیکه کدی که براش نوشتیم میاد فراخوانی میشه

نکته: trigger یک پراسیجری که اتوماتیک فراخوانی میشه خودش نیاز نیست ما فراخوانیش بکنیم ینی هر جایی که اپدیت یا دیلیت یا insert می خواد انجام بشه این تیکه برنامه اجرا میشه خودش انگار وصله به اون جدول

Trigger to Maintain credits_earned value

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> 'F' and nrow.grade is not null
   and (orow.grade = 'F' or orow.grade is null)
begin atomic
    update student
    set tot_cred= tot_cred +
        (select credits
         from course
         where course.course_id= nrow.course_id)
    where student.id = nrow.id;
end;
```

grade ستون ما است

ینی مخالف

این ینی tot_cred قبلی رو به اضافه credits که بعد از پاس کردن به دست میاد میکنه

یک تغییر نمره ای توی جدول takes انجام شده ینی موقعی که اپدیت شد بیا این کارای بالا رو انجام بده

نکته: طول نوشتن trigger مون محدودیتی نداره

Recursive Queries

کوئری های بازگشتی

Recursion in SQL

پیش نیازهای یک درس رو می خوایم توی خروجی نمایش بدیم ینی ممکنه یک درس یک پیش نیاز داشته باشه و اون درس هم خودش باز پیش نیاز داشته باشه

- SQL:1999 permits recursive view definition
- Example: find which courses are a prerequisite, whether directly or indirectly, for a specific course

```
with recursive rec_prereq(course_id, prereq_id) as (
        select course_id, prereq_id
        from prereq
    union
        select rec_prereq.course_id, prereq.prereq_id,
        from rec_rereq, prereq
        where rec_prereq.prereq_id = prereq.course_id
    )
select ∗
from rec_prereq;
```

فیلد اول درس اصلی رو میگه و فیلد دوم درس پیش نیازو

اینا پیش نیاز مستقیم میشن

اینجا می تونه اون تفاضل و اشتراک هم باشه به جای اجتماع

اینا پیش نیازهای غیر مستقیم

This example view, *rec_prereq,* is called the *transitive closure* of the *prereq* relation

The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.
  - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *prereq* with itself
    - This can give only a fixed number of levels of managers
    - Given a fixed non-recursive query, we can construct a database with a greater number of levels of prerequisites on which the query will not work
    - Alternative: write a procedure to iterate as many times as required
      - See procedure *findAllPrereqs* in book

Example of Fixed-Point Computation

course_id	*prereq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-190
CS-319	CS-101
CS-319	CS-315
CS-347	CS-319

Iteration Number	*Tuples in c1*
0	
1	(CS-319)
2	(CS-319), (CS-315), (CS-101)
3	(CS-319), (CS-315), (CS-101), (CS-190)
4	(CS-319), (CS-315), (CS-101), (CS-190)
5	done

Advanced Aggregation Features

Ranking

میخوایم برای هر کدوم از رکورد ها یک رنک مشخص بکنیم ینی فانکشن رنک میاد همین کارو میکنه و اون ترتیب رو مشخص می کنه

- 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

یک جدولی داریم که داره نمره های دانشجوها رو مشخص میکنه GPA میشه معدل دانشجو

- 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*

  بینی توی خروجی هم براساس ترتیب نشون بده
- 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

این اگریگیشن برخلاف group by میاد برای هر ردیف یک عدد مشخص میکنه ینی اگر 20 تا رکورد داریم تهش هم 20 تا داریم باز

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.

ینی براساس دانشکده بیا پارتیشن بندی انجام بده

ینی مثلا پارتیشن بندی کردیم بین دانشکده برق و کامپ بعد میگه حالا توی این پارتیشن بندی ها بیاد اردر بای بکنه ینی مثلا برای دانشکده برق بیا order by بزن توی اون دانشکده

اینجا هم باز 200 تا رکورد داریم فقط توی اون جدول براساس پارتیشن بندی نشونشون میده

Ranking (Cont.)

- It is often the case, especially for large results, that we may be interested only in the top-ranking tuples of the result rather than the entire list.
- For rank queries, this can be done by nesting the ranking query within a containing query whose **where** clause chooses only those tuples whose rank is lower than some specified value.
- For example, to find the top 5 ranking students based on GPA we could extend our earlier example by writing:

```
select *
from (select ID, rank() over (order by (GPA) desc) as s rank
from student grades)
where s rank <= 5;
```

Ranking (Cont.)

- Other ranking functions:
  - **percent_rank** (within partition, if partitioning is done)
    - If there are *n* tuples in the partition12
      and the rank of the tuple is *r*, then its percent rank is defined as $(r - 1)/(n - 1)$ (and as *null* if there is only one tuple in the partition).
  - **row_number:** sorts the rows and gives each row a unique number corresponding to its position in the sort order; different rows with the same ordering value would get different row numbers, in a nondeterministic fashion.

به صورت درصدی رنک رو بیان می کنه

Windowing

- Used to smooth out random variations.
- E.g., **moving average**: “Given sales values for each date, calculate for each date the average of the sales on that day, the previous day, and the next day”
- **Window specification** in SQL:
  - Given relation *sales(date, value)*

```
select date, sum(value) over
    (order by date between rows 1 preceding and 1 following)
 from sales
```

یک رکورد قبل از رکورد فعلی ویک رکورد بعد از رکورد فعلی

مبخواد چندتا رکورد رو مرتب کنه

Windowing

- Examples of other window specifications:
  - **between rows unbounded preceding and current**
  - **rows unbounded preceding**
  - **range between** 10 **preceding and current row**
    - All rows with values between current row value –10 to current value
  - **range interval** 10 **day preceding**
    - Not including current row

Windowing (Cont.)

- Can do windowing within partitions
- E.g., Given a relation *transaction* (*account_number, date_time, value*), where value is positive for a deposit and negative for a withdrawal
  - "Find total balance of each account after each transaction on the account"

```
select account_number, date_time,
    sum (value) over
                (partition by account_number
                order by date_time
                rows unbounded preceding)
      as balance
from transaction
order by account_number, date_time
```

OLAP

Data Analysis and OLAP

- **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

item_name	*color*	*clothes_size*	*quantity*
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*

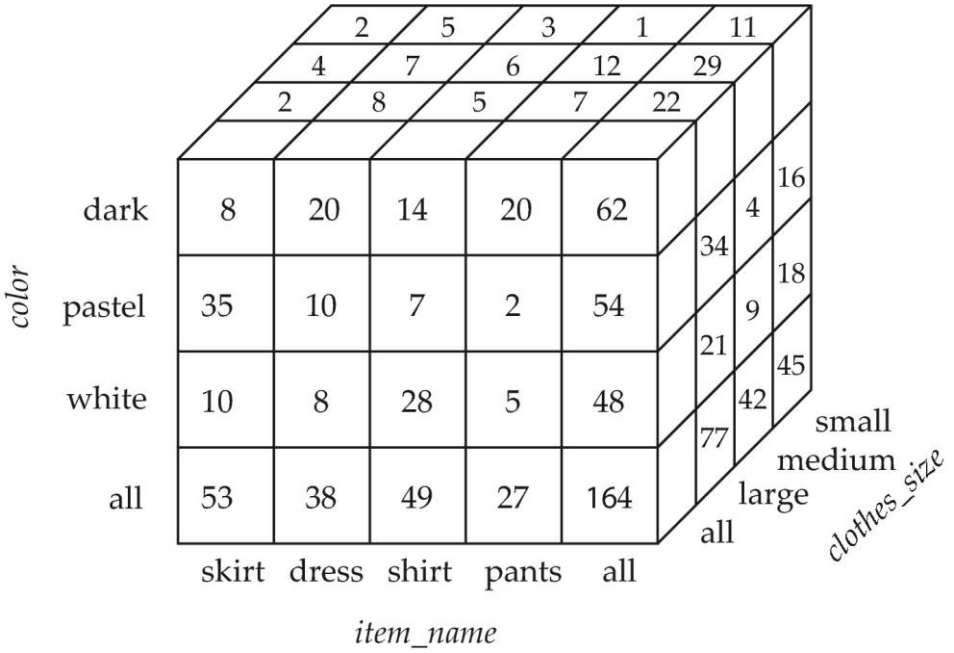
clothes_size **all**

item_name \ *color*	dark	pastel	white	total
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

- 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.

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



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**

category	*item_name*	*color*				
		dark	pastel	white	total	
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

Relational Representation of Cross-tabs

- Cross-tabs can be represented as relations
  - We use the value **all** is used to represent aggregates.
  - The SQL standard actually uses null values in place of **all** despite confusion with regular null values.

item_name	*color*	*clothes_size*	*quantity*
skirt	dark	**all**	8
skirt	pastel	**all**	35
skirt	white	**all**	10
skirt	**all**	**all**	53
dress	dark	**all**	20
dress	pastel	**all**	10
dress	white	**all**	5
dress	**all**	**all**	35
shirt	dark	**all**	14
shirt	pastel	**all**	7
shirt	White	**all**	28
shirt	**all**	**all**	49
pant	dark	**all**	20
pant	pastel	**all**	2
pant	white	**all**	5
pant	**all**	**all**	27
all	dark	**all**	62
all	pastel	**all**	54
all	white	**all**	48
all	**all**	**all**	164

Extended Aggregation to Support OLAP

- The **cube** operation computes union of **group by**'s on every subset of the specified attributes
- Example relation for this section
  sales(*item_name, color, clothes_size, quantity*)
- E.g., consider the query

  select *item_name, color, size,* **sum**(*number*)
  from *sales*
  group by cube(*item_name, color, size*)

  This computes the union of eight different groupings of the *sales* relation:

  { (*item_name, color, size*), (*item_name, color*),
  (*item_name, size*), (*color, size*),
  (*item_name*), (*color*),
  (*size*), ( ) }

  where ( ) denotes an empty **group by** list.
- For each grouping, the result contains the null value for attributes not present in the grouping.

Online Analytical Processing Operations

- Relational representation of cross-tab that we saw earlier, but with *null* in place of **all**, can be computed by
- **select** *item_name*, *color*, **sum**(*number*)
  from *sales*
  group by cube(*item_name, color*)
- The function **grouping()** can be applied on an attribute
  - Indicates whether a specified column expression in a GROUP BY list is aggregated or not. GROUPING returns 1 for aggregated or 0 for not aggregated in the result set.
  - **select** *item_name, color, size*, **sum**(*number*),
    grouping(*item_name*) **as** *item_name_flag*,
    grouping(*color*) **as** *color_flag*,
    grouping(*size*) **as** *size_flag*,
    from *sales*
    group by cube(*item_name, color, size*)

Online Analytical Processing Operations

- Can use the function **decode()** in the **select** clause to replace such nulls by a value such as **all**
  - E.g., replace *item_name* in first query by
    decode(**grouping**(item_*name*), 1, ‘all’, *item_name*)

Extended Aggregation (Cont.)

- The **rollup** construct generates union on every prefix of specified list of attributes
- E.g.,

```
select item_name, color, size, sum(number)
from sales
group by rollup(item_name, color, size)
```

  - Generates union of four groupings:

{ (*item_name, color, size*), (*item_name, color*), (*item_name*), ( ) }

- Rollup can be used to generate aggregates at multiple levels of a hierarchy.
- E.g., suppose table *itemcategory*(*item_name, category*) gives the category of each item. Then

```
select category, item_name, sum(number)
from sales, itemcategory
where sales.item_name = itemcategory.item_name
group by rollup(category, item_name)
```

would give a hierarchical summary by *item_name* and by *category.*

End of Chapter 5