SQL grouping, views & modifying data

- Grouping (GROUP BY)
- Views
- INSERT
- UPDATE
- DELETE

Steen Jensen, autumn 2017

SQL query - grouping

- Grouping makes it possible to show information listed in groups
- Conditions can be made by using the ***HAVING*** clause – not mandatory
- Syntax:

```
SELECT <fieldlist>
FROM <tablename>
GROUP BY <fieldlist>
HAVING <criteria>
```

```
SELECT count(*), types, sum(price)
FROM room
GROUP BY types
HAVING sum(price)>3000 ;
```

SQL query - grouping

- But wait a minute…
- …isn't HAVING the same as WHERE..?
- Not quite
  - WHERE filters specific records
  - HAVING filters specific groups from the final result
- You can't use an aggregate function in a WHERE clause

Exercise – grouping

- With the data in place for the ***hotel database*** then run the below queries:
  - What is the (average) number of bookings for each hotel in this month? (February in 2011)
  - What is the lost income from unoccupied rooms at each hotel this month? As “This month” pick a month represented in the SQL table Booking.

CREATE VIEW intro

- A view is defined as a query on one or more base tables or views
- To the database user, a view appears just like a real table with columns & rows
- ***However,*** unlike a base table, a view is not necessarily stored in the database
- The DBMS stores the ***definition*** of the view in the database

The format of the CREATE VIEW statement is:

```
CREATE VIEW ViewName [(newColumnName [, . . . ])]
AS subselect [WITH [CASCADED | LOCAL] CHECK OPTION]
```

- The ***subselect*** is known as the ***defining query***

CREATE VIEW– example 1

Example 6.3 Create a horizontal view

Create a view so that the manager at branch B003 can see only the details for staff who work in his or her branch office.

A horizontal view restricts a user's access to selected rows of one or more tables.

```
CREATE VIEW Manager3Staff
AS SELECT *
    FROM Staff
    WHERE branchNo = 'B003';
```

```
SELECT * FROM Manager3Staff;
```

Table 6.3 Data for view Manager3Staff.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

CREATE VIEW– example 2

Example 6.4 Create a vertical view

Create a view of the staff details at branch B003 that excludes salary information, so that only managers can access the salary details for staff who work at their branch.

A vertical view restricts a user's access to selected columns of one or more tables.

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
    FROM Staff
    WHERE branchNo = 'B003';
```

Note that we could rewrite this statement to use the Manager3Staff view instead of the Staff table, thus:

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
    FROM Manager3Staff;
```

Table 6.4 Data for view Staff3.

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

CREATE VIEW– example 3

Example 6.5 Grouped and joined views

Create a view of staff who manage properties for rent, which includes the branch number they work at, their staff number, and the number of properties they manage (see Example 5.27).

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo
    GROUP BY s.branchNo, s.staffNo;
```

require a multi-table join. Note that we have to name the columns in the definition of the view because of the use of the unqualified aggregate function COUNT in the subselect.

Table 6.5 Data for view StaffPropCnt.

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

DROP VIEW

- A view is removed from the database with the DROP VIEW

DROP VIEW ViewName [**RESTRICT | CASCADE**]

DROP VIEW causes the definition of the view to be deleted from the database. For example, we could remove the Manager3Staff view using the statement:

DROP VIEW Manager3Staff;

If CASCADE is specified, DROP VIEW deletes all related dependent objects, in other words, all objects that reference the view. This means that DROP VIEW also deletes any views that are defined on the view being dropped. If RESTRICT is specified and there are any other objects that depend for their existence on the continued existence of the view being dropped, the command is rejected. The default setting is RESTRICT.

Advantages and disadvantages of views

Table 6.7 Summary of advantages/disadvantages of views in SQL.

Advantages	Disadvantages
Data independence	Update restriction
Currency	Structure restriction
Improved security	Performance
Reduced complexity	
Convenience	
Customization	
Data integrity	

Explanation of selected advantages

Currency

Changes to any of the base tables in the defining query are immediately reflected in the view.

Improved security

Each user can be given the privilege to access the database only through a small set of views that contain the data appropriate for that user, thus restricting and controlling each user's access to the database.

Reduced complexity

A view can simplify queries, by drawing data from several tables into a single table, thereby transforming multi-table queries into single-table queries.

Convenience

Views can provide greater convenience to users as users are presented with only that part of the database that they need to see. This also reduces the complexity from the user's point of view.

Customization

Views provide a method to customize the appearance of the database, so that the same underlying base tables can be seen by different users in different ways.

Explanation of selected disadvantages

Structure restriction

The structure of a view is determined at the time of its creation. If the defining query was of the form SELECT * FROM . . . , then the * refers to the columns of the base table present when the view is created. If columns are subsequently added to the base table, then these columns will not appear in the view, unless the view is dropped and recreated.

Performance

There is a performance penalty to be paid when using a view. In some cases, this will be negligible; in other cases, it may be more problematic. For example, a view defined by a complex, multi-table query may take a long time to process as the view resolution must join the tables together *every time the view is accessed*. View resolution requires additional computer resources. In the next section, we briefly discuss an alternative approach to maintaining views that attempts to overcome this disadvantage.

Exercise in CREATE VIEW

- Continue with the Hotel case

- Create a view containing the hotel name of hotels in XXX (replace XXX with a specific city from your Hotel table)
- Create a view containing the hotel name and the names of the guests staying at each hotel
- Create a view containing the hotel name and a count of the number of guests staying at each hotel
- Create a view containing the hotel name and the room type for each hotel – the different room types at each hotel should only be shown once (avoid duplicates)

INSERT

- Insert a row in a table:

  INSERT INTO <tableName>
  VALUES (<valuelist>)

- Example:

  INSERT INTO hotel
  VALUES (1,'The Pope','Vaticanstreet 1 1111 Bishopcity');

  NB! The first value (1) in VALUES is NOT necessary, as the field/column Hotel_No is assigned by the DBMS

INSERT

- Things to note about INSERT
  - The value list must match the field list for the table into which the record is inserted
  - If we try to insert a record with a key field which already exists, we will get an error
  - Null values can be inserted if the table definition allows it

The field list can be specified explicitly

```
INSERT INTO hotel
    (Hotel_No, Name, Address)
    VALUES (1,'The Pope','Vaticanstreet 1  1111
    Bishopcity');
```

UPDATE

- Updates values for specified field(s)

- Without a WHERE: all rows/records

UPDATE <tableName>
  SET field1 = value1, field2 = value2,…
  WHERE <condition>

UPDATE

- Example:

```
update hotel
set name ='The Great Pope'
where hotel_no= 1;
```

UPDATE

- Things to note about UPDATE
  - For each field update, the type of the value must match the type of the field
  - The WHERE clause is optional – if you leave it out, all records in the table are updated!
  - It is not considered an error if zero rows are changed, so pay attention to the condition in the WHERE clause…

DELETE

- SQL syntax
  - DELETE FROM table_name
    WHERE some_column=some_value
- Delete all rows / records from GUEST :
  - DELETE FROM Guest;
- Delete all rooms from hotel no 1:
  - DELETE FROM Room where Hotel_no = 1;

Exercise in modifying data

- With the data in place, run the below commands on the database
  - INSERT INTO Hotel VALUES (specify your own values)
  - INSERT INTO Room VALUES (specify your own values)
  - UPDATE Room SET Price = Price* 1.30;
  - DELETE FROM Room WHERE (Room_no = 8)

- Now formulate commands yourself, in order to:
  - Insert data about ” Scandic Roskilde” in the table Hotel (you can find the data on the Internet, or make it up yourself)
  - Insert data representing the fact that Hotel Scandic have 10 rooms with room numbers 101, 102, 103, 201, 202, 203, 301, 302, 303, 400
  - Update the name of the Hotel ”Scandic Roskilde” to ”The new Scandic Roskilde”
  - Insert data for a booking of a room at the hotel ”The new Scandic Roskilde”