Seminar 2. SQL Queries – DML Subset

GROUP BY and HAVING

So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.

Consider: *Find the age of the youngest student for each group.*

- In general, we don't know how many groups exist
- Suppose we know that group values go from 110 to 119, we can write 10 similar queries. But when another group is added, a new query should be created.

Group By and *Having* clauses allow us to solve problems like this in only one SQL query. General syntax is:

```
SELECT [DISTINCT] target-list
FROM   relation-list
WHERE  qualification
GROUP BY  grouping-list
HAVING    group-qualification
```

The *target-list* contains

- attribute names (the attribute names must be a subset of *grouping-list*);
- terms with aggregate operations (e.g., MIN (*S.age*)).

Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Group By / *Having* conceptual evaluation:

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, *`unnecessary'* fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups. Expressions in *group-qualification* must have a single value per group!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

Sample: *Find the age of the youngest student with age ≥ 20 for each group with at least 2 such students*

```
SELECT  S.gr,  MIN (S.age)
FROM  Students S
WHERE  S.age >= 20
```

```
GROUP BY  S.gr
HAVING  COUNT (*) > 1
```

- Only S.gr and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `*unnecessary*'.
- 2nd column of result is unnamed.   (Use AS to name it.)

Sample: *Find the number of enrolled students and the grade average for each course with 6 credits*

```
SELECT  C.cid,  COUNT (*) AS scount, AVG(grade)
FROM  Students S, Enrolled E, Courses C
WHERE  S.sid=E.sid AND E.cid=C.cid AND C.credits=6
GROUP BY  C.cid
```

Insert a single record:

```
INSERT  [INTO]  table_name [(column_list)]
VALUES ( value_list)
```

Example:

```
INSERT INTO  Students (sid, name, email, age, gr)
VALUES  (53688, 'Smith', 'smith@math', 18, 311)
```

Bulk insert:

```
INSERT [INTO] table_name [(column_list)]
<select statement>
```

Example:

```
INSERT INTO  Enrolled  (sid, cid, grade)
SELECT sid, 'BD1', 10
FROM Students
```

Delete:

```
DELETE  [FROM]  table_name
[WHERE  qualification]
```

Example:

```
DELETE  FROM Students S
WHERE S.name = 'Smith'
```

Update:

```
UPDATE table_name
SET
      column1=value1,column2=value2,...
[WHERE qualification]
```

Example:

```
UPDATE Students S
SET S.age=S.age+1
WHERE S.sid = 53688
```

Note: in MS SQL Server insert, update and delete operations are transactional: all records are inserted/updated/deleted or no one.