CS2102
Structured Query Language (SQL)
Part 2

Subquery Expressions

- EXISTS subqueries
- IN subqueries
- ANY/SOME subqueries
- ALL subqueries
- UNIQUE subqueries

EXISTS Subqueries

- EXISTS (subquery)
- Returns *true* if the result subquery is non-empty; otherwise, *false*

EXISTS Subqueries (cont.)

Find distinct customers who like some pizza sold by "Corleone Corner"

Likes

cname	pizza
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Sciliana
Ralph	Diavola

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

cname
Homer
Ralph

EXISTS Subqueries (cont.)

```
select   distinct cname
from     Likes L
where    exists (
         select   1
         from     Sells S
         where    S.rname = 'Corleone Corner'
         and      S.pizza = L.pizza
         );

select   distinct L.cname
from     Likes L inner join Sells S
         on S.pizza = L.pizza
where    S.rname = 'Corleone Corner';
```

Subquery Expressions: Scoping Rules

- Queries with subquery expressions are also called nested queries
- A subquery expression is referred to as an inner query that is nested within an outer query
- **Scoping rules for table alias (a.k.a. tuple variable)**:
  - A tuple variable declared in a subquery/query Q can be used only in Q and any subquery nested within Q
  - If a tuple variable is declared both locally in a subquery *Q* as well as in an outer query, the local declaration applies in *Q*

NOT EXISTS Subqueries

Find distinct customers who does not like any pizza sold by “Corleone Corner”

Customers

cname	area
Homer	West
Lisa	South
Maggie	East
Moe	Central
Ralph	Central
Willie	North

Likes

cname	pizza
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Sciliana
Ralph	Diavola

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma’s Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

cname
Lisa
Maggie
Moe
Willie

NOT EXISTS Subqueries (cont.)

```
select   cname
from     Customers C
where    not exists (
         select   1
         from     Likes L natural join Sells S
         where    S.rname = 'Corleone Corner'
         and      L.cname = C.cname
         );

select   cname   from Customers
except
select   cname
from     Likes natural join Sells
where    rname = 'Corleone Corner';
```

IN Subqueries

- expression IN (subquery)
- Subquery must return exactly one column
- Returns *false* if result of subquery is empty; otherwise return the result of the boolean expression

$$((v = v_1) \text{ or } (v = v_2) \text{ or } \cdots \text{ or } (v = v_n))$$

where

  - v denote the result of expression
  - $\{v_1, v_2, \cdots, v_n\}$ denote the result of subquery

IN Subqueries (cont.)

Find distinct customers who like some pizza sold by "Corleone Corner"

```
select   distinct cname
from     Likes
where    pizza in (
         select    pizza
         from      Sells
         where     rname = 'Corleone Corner'
         );
```

Another Form of IN Predicate

- expression IN (value1, value2, · · · , valuen)

  Example: Find pizzas that contain ham or seafood

```
select   distinct pizza  from  Contains
where    ingredient in ('ham', 'seafood');

select   distinct pizza from  Contains
where    ingredient = 'ham'  or  ingredient = 'seafood';

select   pizza from Contains where ingredient = 'ham'
union
select   pizza from Contains where ingredient = 'seafood';
```

(Non-)Correlated Nested Queries

- A nested query with a subquery that references a tuple variable declared in an outer query is called a correlated nested query
- Example of correlated nested query

```
select   distinct cname from  Likes L where  exists (
         select     1 from Sells S
         where      (S.rname = 'Corleone Corner') and (S.pizza = L.pizza));
```

- Example of non-correlated nested query

```
select   distinct cname from  Likes
where    pizza in (select pizza from Sells where rname = 'Corleone Corner');
```

ANY/SOME Subqueries

- expression operator ANY (subquery)
- Subquery must return exactly one column
- Returns *false* if result of subquery is empty; otherwise return the result of the boolean expression

$$((v\ op\ v_1)\ \text{or}\ (v\ op\ v_2)\ \text{or}\ \cdots\ \text{or}\ (v\ op\ v_n))$$

where

  - v denote the result of expression
  - $\{v_1, v_2, \cdots, v_n\}$ denote the result of subquery
  - *op* denote operator

ANY/SOME Subqueries (cont.)

Find distinct restaurants that sell some pizza P1 that is more expensive than some pizza P2 sold by "Corleone Corner". P1 and P2 are not necessarily the same pizza. Exclude "Corleone Corner" from the query result.

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname
Lorenzo Tavern
Mamma's Place
Pizza King

ANY/SOME Subqueries (cont.)

Find distinct restaurants that sell some pizza P1 that is more expensive than some pizza P2 sold by "Corleone Corner". P1 and P2 are not necessarily the same pizza. Exclude "Corleone Corner" from the query result.

```
select   distinct rname
from     Sells
where    rname <> 'Corleone Corner'
and      price > any (
         select    price
         from      Sells
         where     rname = 'Corleone Corner'
         );
```

ANY/SOME Subqueries (cont.)

Find distinct restaurants that sell some pizza P1 that is more expensive than some pizza P2 sold by "Corleone Corner". P1 and P2 are not necessarily the same pizza. Exclude "Corleone Corner" from the query result.

```
select   distinct rname
from     Sells S1
where    rname <> 'Corleone Corner'
and      exists (
         select   1
         from     Sells S2
         where    S2.rname = 'Corleone Corner'
         and      S1.price > S2.price
         );
```

ALL Subqueries

- expression operator ALL (subquery)
- Subquery must return exactly one column
- Returns *true* if result of subquery is empty; otherwise return

  $$((v\ op\ v_1)\ \text{and}\ (v\ op\ v_2)\ \text{and}\ \cdots\ \text{and}\ (v\ op\ v_n))$$

  where
  - v denote the result of expression
  - $\{v_1, v_2, \cdots, v_n\}$ denote the result of subquery
  - op denote operator

ALL Subqueries (cont.)

For each restaurant, find the name and price of its most expensive pizzas. Exclude restaurants that do not sell any pizza.

Sells

rname	pizza	price
Corleone Corner	Diavola	25
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	pizza	price
Corleone Corner	Diavola	25
Corleone Corner	Hawaiian	25
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Hawaiian	21

ALL Subqueries (cont.)

For each restaurant, find the name and price of its most expensive pizzas. Exclude restaurants that do not sell any pizza.

```
select  rname, pizza, price
from    Sells S1
where   price >= all (
        select  S2.price
        from    Sells S2
        where   S2.rname = S1.rname
        );
```

ALL Subqueries (cont.)

For each restaurant, find the name and price of its most expensive pizzas. Exclude restaurants that do not sell any pizza.

```
select   rname, pizza, price  from Sells
except
select   rname, pizza, price
from     Sells S1
where    price < any (
         select   S2.price
         from     Sells S2
         where    S2.rname = S1.rname
         );
```

UNIQUE Subqueries

- UNIQUE (subquery)
- Returns *false* if the result subquery contains at least two distinct records t_1 and t_2 such that "$t_1 = t_2$" evaluates to *true*; otherwise, *true*
  - "$t_1 = t_2$" is evaluated as "$(t_1.a_1 = t_2.a_1)$ and $\cdots$ and $(t_1.a_n = t_2.a_n)$" where $\{a_1, \cdots, a_n\}$ are the attributes in the schema of the subquery result
- UNIQUE subqueries are not widely supported

UNIQUE Subqueries (cont.)

Find distinct pizzas that are sold by at most one restaurant in each area; exclude pizzas that are not sold by any restaurant

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Margherita	19
Gambino Oven	Hawaiian	25
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

Restaurants

rname	area
Corleone Corner	East
Gambino Oven	Central
Lorenzo Tavern	Central
Mamma's Place	South
Pizza King	East

pizza
Margherita
Hawaiian
Sciliana
Funghi
Marinara

UNIQUE Subqueries (cont.)

Find distinct pizzas that are sold by at most one restaurant in each area; exclude pizzas that are not sold by any restaurant

```
select   distinct pizza
from     Sells S
where    unique (
             select   R.area
             from     Restaurants R
             where    R.pizza = S.pizza
             );
```

Scalar Subqueries

- A scalar subquery is a subquery that returns at most one tuple with one column
  - If the subquery's result is empty, its return value is `null`
- A scalar subquery can be used as a scalar expression

Scalar Subqueries (cont.)

For each restaurant that sells Funghi, find its name, area, and selling price.

```
select   R.rname, R.area, S.price
from     Sells S, Restaurants R
where    S.pizza = 'Funghi'
and      S.rname = R.rname;
```

```
select   rname,
         (select R.area from Restaurants R
         where R.rname = S.rname), price
from     Sells S
where    pizza = 'Funghi';
```

Usage of Subqueries

- Non-scalar subquery expressions can be used in different parts of SQL queries:
  - `WHERE` clause
  - `FROM` clause (to be illustrated later)
  - `HAVING` clause (to be discussed later)

Database Modifications with Subqueries

```
create table Students (
    studentId       integer,
    name            varchar(100),
    birthDate       date,
    year            integer,
    primary key (studentId));
```

```
create table Enrolls (
    sid     integer
            references Students,
    cid     integer
            references Courses,
    grade   char(2),
    primary key (sid, cid));
```

```
-- Enroll all first-year students in the course 101
insert into Enrolls (sid, cid)
   select studentId, 101
   from   students
   where year = 1;
```

Aggregate Functions

- Aggregate function computes a single value from a set of tuples
- **Example**: Find the minimum, maximum, and average prices of pizzas sold by Corleone Corner

```
select   min (price), max (price), avg (price)
from     Sells
where    rname = 'Corleone Corner'
```

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

min	max	avg
19	25	22.6666666666666667

Aggregate Functions (cont.)

Query	Meaning
select min(A) **from** R	Minimum value in A
select max(A) **from** R	Maximum value in A
select avg(A) **from** R	Average of values in A
select sum(A) **from** R	Sum of values in A
select count(A) **from** R	Count number of non-null values in A
select count(*) **from** R	Count number of rows in R
select avg(**distinct** A) **from** R	Average of distinct values in A
select sum(**distinct** A) **from** R	Sum of distinct values in A
select count(**distinct** A) **from** R	Count number of distinct non-null values in A

Aggregate Functions (cont.)

- Let *R* be an empty relation
- Let *S* be a relation with cardinality = n where all values of A are null values

Query	Result
select min(A) **from** R	`null`
select max(A) **from** R	`null`
select avg(A) **from** R	`null`
select sum(A) **from** R	`null`
select count(A) **from** R	0
select count(*) **from** R	0

Query	Result
select min(A) **from** S	`null`
select max(A) **from** S	`null`
select avg(A) **from** S	`null`
select sum(A) **from** S	`null`
select count(A) **from** S	0
select count(*) **from** S	*n*

Usage of Aggregate Functions

- Aggregate functions can be used in different parts of SQL queries:
  - `SELECT` clause
  - `HAVING` clause (to be discussed later)
  - `ORDER BY` clause (to be discussed later)

Usage of Aggregate Functions (cont.)

Find the number of items ordered and the maximum order cost for an item

Orders

item	price	qty
A	2.50	100
B	4.00	100
C	7.50	100

count	max
3	750.00

select **count**(∗), **max**(price ∗ qty) **from** Orders;

Usage of Aggregate Functions (cont.)

Find the most expensive pizzas and the restaurants that sell them (at the most expensive price)

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	25
Pizza King	Diavola	17
Pizza King	Hawaiian	21

pizza	rname
Hawaiian	Corleone Corner
Marinara	Mamma's Place

```
select  pizza, rname from Sells
where   price = (select max(price) from Sells);
```

ORDER BY Clause

For each restaurant that sells some pizza, find its name, area, and the pizzas it sells together with their prices. Show the output in ascending order of the area, followed by in descending order of the price.

Restaurants

rname	area
Corleone Corner	North
Gambino Oven	Central
Lorenzo Tavern	Central
Mamma's Place	South
Pizza King	East

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	area	pizza	price
Lorenzo Tavern	Central	Funghi	23
Gambino Oven	Central	Siciliana	16
Pizza King	East	Hawaiian	19
Pizza King	East	Diavola	17
Corleone Corner	North	Hawaiian	25
Corleone Corner	North	Diavola	24
Corleone Corner	North	Margherita	19
Mamma's Place	South	Marinara	22

ORDER BY Clause (cont.)

For each restaurant that sells some pizza, find its name, area, and the pizzas it sells together with their prices. Show the output in ascending order of the area, followed by in descending order of the price.

```
select   *
from     Restaurants natural join Sells
order by area asc, price desc;
```

```
select   *
from     Restaurants natural join Sells
order by area, price desc;
```

LIMIT Clause

Find the top three most expensive pizzas. Show the pizza name, the name of the restaurant that sells it, and its selling price for each output tuple; and show the output in descending order of price.

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

pizza	rname	price
Hawaiian	Corleone Corner	25
Diavola	Corleone Corner	24
Funghi	Lorenzo Tavern	23

```
select   pizza, rname, price
from     Sells
order by price desc
limit    3;
```

Views

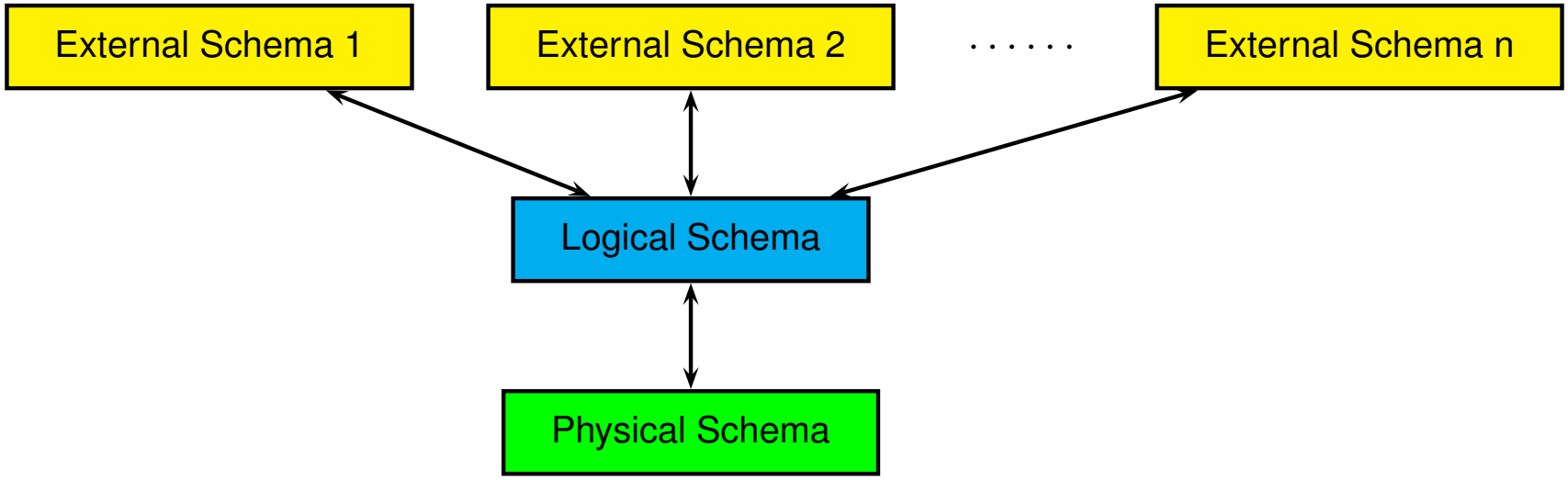
- A view defines a virtual relation that can be used for querying

- **Example**: Consider the following database schema:

  `Courses` (courseId, cname, credits, profId, lectureTime)
  `Profs` (profId, pname, officeRoom, contactNum)
  `Students` (studentId, sname, email, birthDate)
  `Enrolls` (sid, cid, grade)

```
create view CourseInfo (cname, pname, lectureTime, numStudent) as
    select C.cname, P.pname, C.lectureTime,
           (select count(*) from Enrolls E where E.cid = C.courseId)
    from   Courses C natural join Profs P;
```

Views: Providing Logical Data Independence



- Logical Schema - logical structure of data in DBMS
- Physical Schema - how the data described by logical schema is physically organized in DBMS
- External Schema - A customized view of logical schema
- Logical (Physical) Data independence: Insulate users/applications from changes to logical (physical) schema

GROUP BY Clause

For each restaurant that sells some pizza, find the minimum and maximum prices of its pizzas

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	min	max
Corleone Corner	19	25
Gambino Oven	16	16
Lorenzo Tavern	23	23
Mamma's Place	22	22
Pizza King	17	21

GROUP BY Clause (cont.)

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	min	max
Corleone Corner	19	25
Gambino Oven	16	16
Lorenzo Tavern	23	23
Mamma's Place	22	22
Pizza King	17	21

Conceptual processing steps:

1. Partition the tuples in `Sells` into groups based on `rname`
2. Compute `min(price)` and `max(price)` for each group
3. Output one tuple for each group

GROUP BY Clause (cont.)

For each restaurant that sells some pizza, find the minimum and maximum prices of its pizzas

Bad solution!

```
select rname, min(price), max(price) from Sells where rname = 'Corleone Corner'
union
select rname, min(price), max(price) from Sells where rname = 'Gambino Oven'
union
. . . . . .
union
select rname, min(price), max(price) from Sells where rname = 'Pizza King';
```

GROUP BY Clause (cont.)

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	min	max
Corleone Corner	19	25
Gambino Oven	16	16
Lorenzo Tavern	23	23
Mamma's Place	22	22
Pizza King	17	21

```
select   rname, min(price), max(price)
from     Sells
group by rname;
```

GROUP BY Clause (cont.)

Find the number of students for each (dept,year) combination. Show the output in ascending order of (dept,year).

Students

studentId	name	year	dept
12345	Alice	1	Maths
67890	Bob	2	CS
11123	Carol	4	Maths
20135	Dave	4	CS
20135	Eve	3	CS
18763	Fred	3	Maths
60031	George	1	Maths
87012	Hugh	2	CS
96410	Ivy	4	CS

dept	year	count
CS	2	2
CS	3	1
CS	4	2
Maths	1	2
Maths	3	1
Maths	4	1

select dept, year, **count**(∗) **from** Students
group by dept, year **order by** dept, year;

GROUP BY Clause (cont.)

For each restaurant that sells some pizza, find its average pizza price. Show the restaurants in descending order of their average pizza price.

```
select    rname, avg(price) as avgPrice
from      Sells
group by  rname
order by  avgPrice desc;
```

Sells

rname	pizza	price
Corleone Corner	Diavola	24
Corleone Corner	Hawaiian	25
Corleone Corner	Margherita	19
Gambino Oven	Siciliana	16
Lorenzo Tavern	Funghi	23
Mamma's Place	Marinara	22
Pizza King	Diavola	17
Pizza King	Hawaiian	21

rname	avg
Lorenzo Tavern	23.0000000000000000
Corleone Corner	22.6666666666666667
Mamma's Place	22.0000000000000000
Pizza King	19.0000000000000000
Gambino Oven	16.0000000000000000

GROUP BY Clause: Properties

- In a query with "GROUP BY $a_1, a_2, \cdots, a_n$", two tuples t & t' belong to the same group if the following expression evaluates to `true`:

$$(t.a_1 \text{ IS NOT DISTINCT FROM } t'.a_1) \text{ AND } \cdots \text{ AND } (t.a_n \text{ IS NOT DISTINCT FROM } t'.a_n)$$

- **Example**: Four groups in R if R is grouped by $\{\text{A, C}\}$

R

A	B	C
null	4	19
null	21	19
6	1	null
6	20	null
20	2	10
1	1	2
1	18	2

GROUP BY Clause: Properties (cont.)

- Each output tuple corresponds to one group
- For each column *A* in relation *R* that appears in the `SELECT` clause, one of the following conditions must hold:
  1. *A* appears in the `GROUP BY` clause,
  2. *A* appears in an aggregated expression in the `SELECT` clause (e.g., **min**(A)), or
  3. the primary (or a candidate) key of *R* appears in the `GROUP BY` clause

GROUP BY Clause: Properties (cont.)

Students

studentId	name	year	dept
12345	Alice	1	Maths
11123	Carol	4	Maths
18763	Fred	3	Maths
60031	George	1	Maths
67890	Bob	2	CS
20135	Dave	4	CS
20135	Eve	3	CS
87012	Hugh	2	CS
96410	Ivy	4	CS

This query is invalid!

```
select   dept, year, count(*)
from     Students
group by dept;
```

GROUP BY Clause: Properties (cont.)

For each restaurant that sells some pizza, find its name, area, and the average price of its pizzas

```
select    R.rname, R.area, avg(S.price)
from      Sells S, Restaurants R
where     S.rname = R.rname
group by  R.rname;
```

```
select    R.rname, R.area, avg(S.price)
from      Sells S, Restaurants R
where     S.rname = R.rname
group by  R.rname, R.area;
```

GROUP BY Clause: Properties (cont.)

- If an aggregate function appears in the `SELECT` clause and there is no `GROUP BY` clause, then the `SELECT` clause must not contain any column that is not in an aggregated expression
- **Example**: The following query is invalid!

```
select    rname, min(price), max(price)
from      Sells
```

HAVING Clause

Find restaurants that sell pizzas with an average selling price of at least $22

Sells

rname	pizza	price	
Corleone Corner	Diavola	24	
Corleone Corner	Hawaiian	25	avg(price) = 22.67
Corleone Corner	Margherita	19	
Gambino Oven	Siciliana	16	
Lorenzo Tavern	Funghi	23	
Mamma's Place	Marinara	22	
Pizza King	Diavola	17	
Pizza King	Hawaiian	21	avg(price) = 19

rname
Corleone Corner
Lorenzo Tavern
Mamma's Place

```
select   rname
from     Sells
group by rname
having   avg(price) >= 22;
```

HAVING Clause (cont.)

Find restaurants that sell pizzas with an average selling price higher than the minimum selling price at Pizza King

```
select      rname
from        Sells
group by    rname
having      avg(price) >=
            (select   min(price)
            from      Sells
            where     rname = 'Pizza King');
```

HAVING Clause: Properties

- For each column *A* in relation *R* that appears in the `HAVING` clause, one of the following conditions must hold:
  1. *A* appears in the `GROUP BY` clause,
  2. *A* appears in an aggregated expression in the `HAVING` clause, or
  3. the primary (or a candidate) key of *R* appears in the `GROUP BY` clause

Students

studentId	name	year	dept
12345	Alice	1	Maths
11123	Carol	4	Maths
18763	Fred	3	Maths
60031	George	1	Maths
67890	Bob	2	CS
20135	Dave	4	CS
20135	Eve	3	CS
87012	Hugh	2	CS
96410	Ivy	4	CS

This query is invalid!

```
select   dept, count(*)
from     Students
group by dept
having   year = 3;
```

Conceptual Evaluation of Queries

select	**distinct** select-list
from	from-list
where	where-condition
group by	groupby-list
having	having-condition
order by	orderby-list
limit	limit-specification

1. Compute the cross-product of the tables in **from-list**
2. Select the tuples in the cross-product that evaluate to *true* for the **where-condition**
3. Partition the selected tuples into groups using the **groupby-list**
4. Select the groups that evaluate to *true* for the **having-condition** condition
5. For each selected group, generate an output tuple by selecting/computing the attributes/expressions that appear in the **select-list**
6. Remove any duplicate output tuples
7. Sort the output tuples based on the **orderby-list**
8. Remove the appropriate output tuples based on the **limit-specification**

Queries with Universal Quantification

- **Example**: Find the names of all students who have enrolled in all the courses offered by CS department

Courses (courseId, name, dept)
Students (studentId, name, birthDate)
Enrolls (sid, cid, grade)

Queries with Universal Quantification (cont.)

- Let R denote the set of all students who have enrolled in all the courses offered by CS department
- Let $\overline{R}$ = Students $-$ R
- $\overline{R}$ = the set of all students who have not enrolled in all the courses offered by CS department
- A student $s \in \overline{R}$ iff there exists some CS course c such that s is not enrolled in c
- Given a studentId x, let F(x) = set of courseIds of CS courses that are not enrolled by student with studentId x
- $\overline{R} = \{s \in \text{Students} \mid F(\text{s.studentId}) \neq \emptyset\}$

Queries with Universal Quantification (cont.)

- $\overline{R} = \{s \in \text{Students} \mid F(\text{s.studentId}) \neq \emptyset\}$
- $\overline{R}$ can be computed by the following pseudo SQL query:

  select s.studentId **from** Students **where exists** (F(s.studentId))

- R can be computed by the following pseudo SQL query:

  select s.studentId **from** Students **where not exists**
  (F(s.studentId))

Queries with Universal Quantification (cont.)

```
--F(x): set of courseIds of CS courses that are not enrolled
--by student with studentId x
select courseId
from   Courses C
where dept = 'CS'
and    not exists (
       select 1
       from   Enrolls E
       where E.cid = C.courseId
       and    E.sid = x
);
```

Queries with Universal Quantification (cont.)

```
--Names of students who have enrolled in all CS Courses
select name
from   Students S
where not exists (
        select courseId
        from   Courses C
        where dept = 'CS'
        and not exists (
              select 1
              from   Enrolls E
              where E.cid = C.courseId
              and    E.sid = S.studentId
        )
);
```

Table Expressions

Given the following database schema, find the courses where the total number of enrolled students is higher than that for the course named “Database Systems”. Output the cname and the total number of enrolled students for each selected course.

`Courses`	(cid, cname, credits)
`Enrolls`	(sid, cid, grade)

Assume that `cname` is a candidate key of `Courses`.

Table Expressions (cont.)

Find the courses where the total number of enrolled students is higher than that for the course named "Database Systems". Output the cname and the total number of enrolled students for each selected course.

```
select   C.cname,
         (select   count(*)
         from      Enrolls E
         where     E.cid = C.cid) as numEnroll
from     Courses C natural join Enrolls E
group by C.cid
having   count(*) >
         (select   count(*)
         from      Courses C natural join Enrolls E
         where     C.cname = 'Database Systems');
```

Table Expressions (cont.)

Find the courses where the total number of enrolled students is higher than that for the course named "Database Systems". Output the cname and the total number of enrolled students for each selected course.

```
select   cname, numEnroll
from     (select   C.cid, C.cname, count(*) as numEnroll
          from     Courses C natural join Enrolls E
          group by C.cid) as X
where    numEnroll >
          (select   count(*)
          from     Courses C natural join Enrolls E
          where    C.cname = 'Database Systems');
```

Common Table Expressions (CTEs)

Find the courses where the total number of enrolled students is higher than that for the course named "Database Systems". Output the cname and the total number of enrolled students for each selected course.

```
with CourseEnroll as
        (select    C.cid, C.cname, count(*) as numEnroll
        from       Courses C natural join Enrolls E
        group by   C.cid)
select   cname, numEnroll
from     CourseEnroll
where    numEnroll >
            (select   numEnroll
            from      CourseEnroll
            where     cname = 'Database Systems');
```

Common Table Expressions (CTEs)

```
with
        R1 as (Q1),
        R2 as (Q2),
        ...,
        Rn as (Qn)
select/insert/update/delete statement S;
```

- Each `Ri` is the name of a temporary relation defined by a query `Qi`
- `S` is a SQL statement that references Rn & possibly R1,R2,$\cdots$
- CTEs can be used for writing recursive queries (not covered)

Summary

- Conceptual evaluation of queries

select	**distinct** select-list
from	from-list
where	where-condition
group by	groupby-list
having	having-condition
order by	orderby-list
limit	limit-specification

- Non-scalar subqueries can be used in `FROM`, `WHERE`, and `HAVING` clauses
- Aggregate functions can be used in `SELECT`, `HAVING`, and `ORDER BY` clauses