CS2102 Database Systems

Slides adapted from Prof. Chan Chee Yong

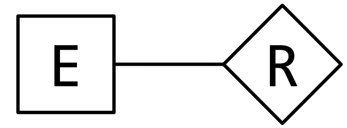
LECTURE 06

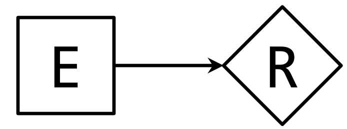
SQL #3

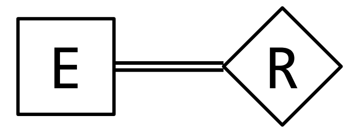
PREVIOUS LECTURE

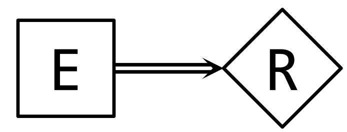
Relationship constraints

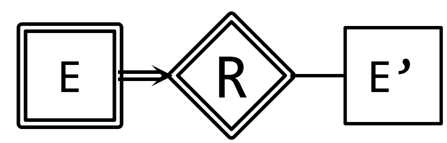
Types

- Many-to-many
  Each instance of E participates in _0 or more_ instance of R
  

- Key
  Each instance of E participates in _at most 1_ instance of R
  

- Total
  Each instance of E participates in _at least 1_ instance of R
  

- Key & total
  Each instance of E participates in _exactly one_ instance of R
  

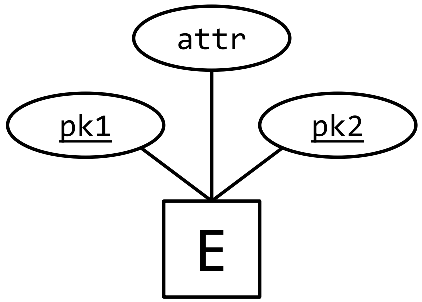
- Weak entity
  E is a weak entity set with identifying owner E’ and identifying relationship set R
  

PREVIOUS LECTURE

ER diagram to SQL

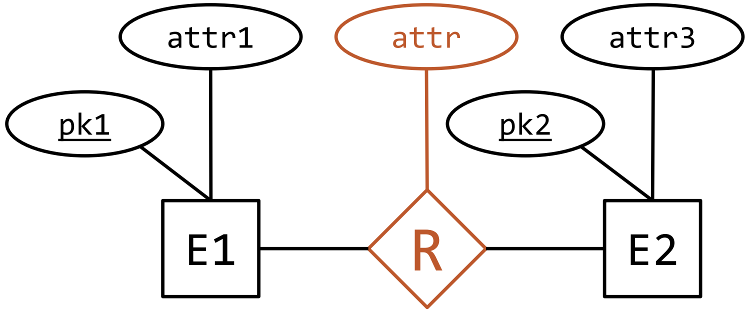
Entity sets

```
CREATE TABLE E (
  pk1   type,
  pk2   type,
  attr  type,
  PRIMARY KEY (pk1, pk2)
);
```



Many-to-many

```
CREATE TABLE R (
  pk1   type REFERENCES E1,
  pk2   type REFERENCES E2,
  attr  type,
  PRIMARY KEY (pk1, pk2)
);
```

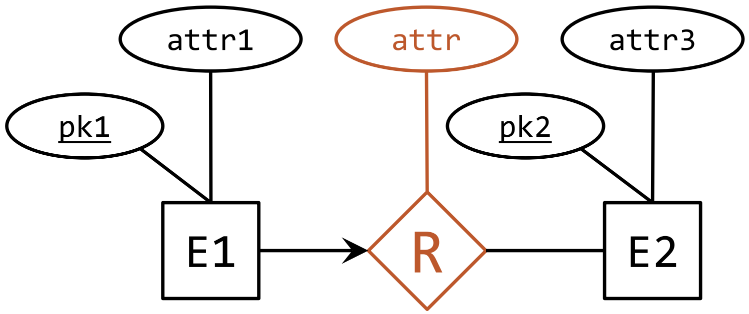


PREVIOUS LECTURE

ER diagram to SQL

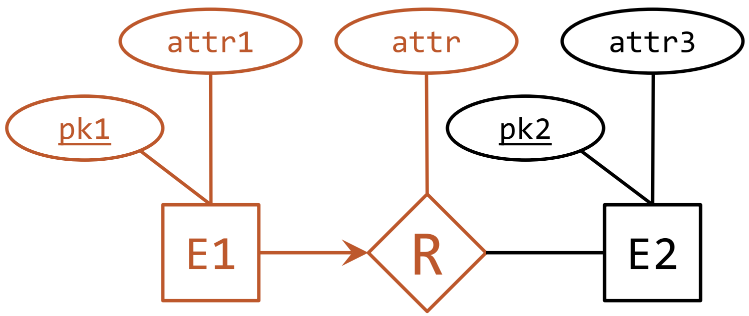
Key constraints approach #1

```
CREATE TABLE R (
  pk1   type REFERENCES E1,
  pk2   type REFERENCES E2,
  attr  type,
  PRIMARY KEY (pk1)
);
```



Key constraints approach #2

```
CREATE TABLE R (
  pk1   type PRIMARY KEY,
  pk2   type REFERENCES E2,
  attr  type,
  attr1 type
);
```

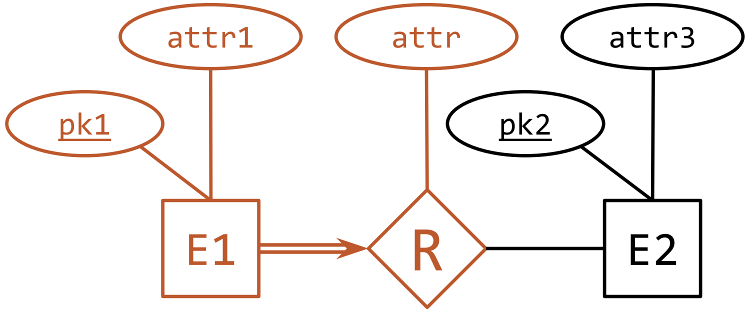


PREVIOUS LECTURE

ER diagram to SQL

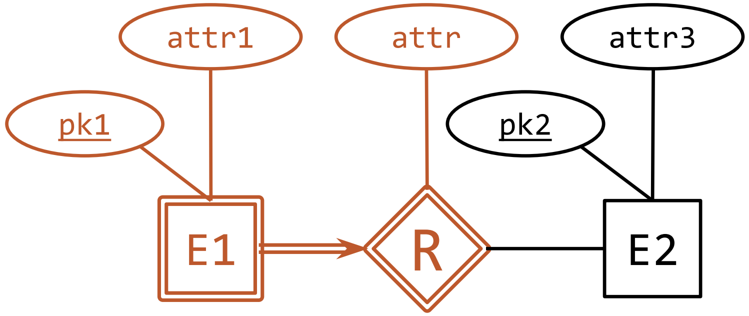
Key & total constraints

```
CREATE TABLE R (
  pk1   type PRIMARY KEY,
  pk2   type NOT NULL,
  attr  type,
  attr1 type,
  FOREIGN KEY (pk2) REFERENCES E2
);
```



Weak entity sets

```
CREATE TABLE R (
  pk1   type,
  pk2   type REFERENCES E2
    ON DELETE cascade,
  attr  type,
  attr1 type,
  PRIMARY KEY (pk1, pk2)
);
```



PREVIOUS LECTURE

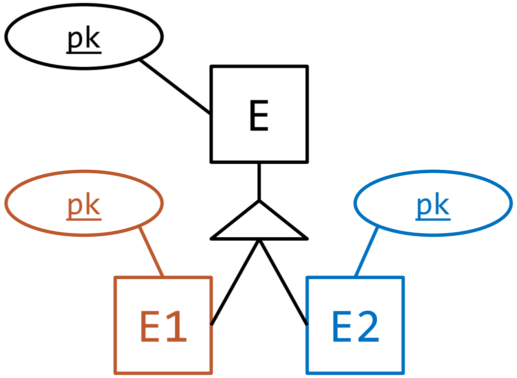
Additional ER concepts

ISA hierarchies approach #1

```
CREATE TABLE E (
   pk     type PRIMARY KEY
);

CREATE TABLE E1 (
   pk     type
          PRIMARY KEY
          REFERENCES E ON DELETE cascade
);

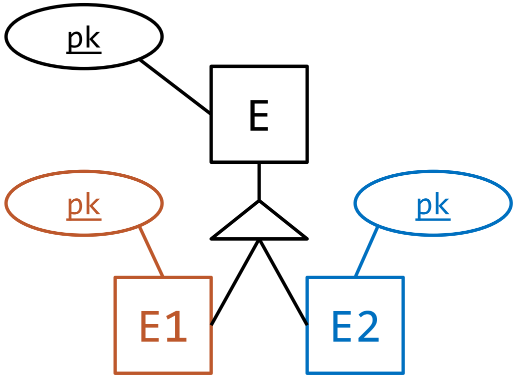
CREATE TABLE E2 (
   pk     type
          PRIMARY KEY
          REFERENCES E ON DELETE cascade
);
```



PREVIOUS LECTURE

Additional ER concepts

ISA hierarchies approach #2




```
CREATE TABLE E1 (
  pk    type
        PRIMARY KEY

);

CREATE TABLE E2 (
  pk    type
        PRIMARY KEY

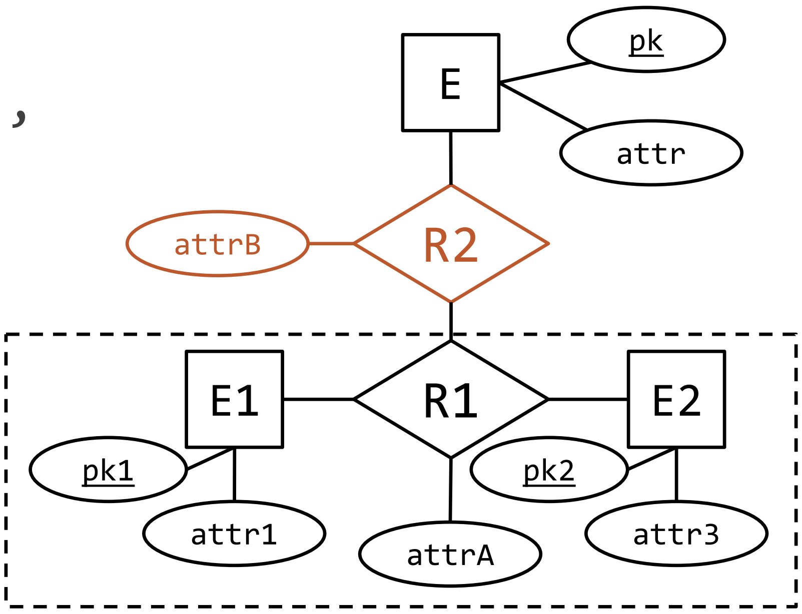
);
```

PREVIOUS LECTURE

Additional ER concepts

Aggregation

```
CREATE TABLE R2 (
  pk     type REFERENCES E,
  pk1    type,
  pk2    type,
  attrB  type,
  PRIMARY KEY (pk, pk1, pk2),
  FOREIGN KEY (pk1, pk2)
    REFERENCES R1 (pk1, pk2)
);
```



- Aggregate functions
  - Introduction
  - Order by clause
  - Group by clause
  - Having clause
- Subqueries
  - select clause
  - from clause
  - where clause
  - Queries with universal quantification

Overview

- Aggregate functions
  - Introduction
  - Order by clause
  - Group by clause
  - Having clause
- Subqueries
  - select clause
  - from clause
  - where clause
  - Queries with universal quantification

Aggregate functions

Aggregate functions

Introduction

- Aggregate function computes a *single value from a set of tuples*
- Types of aggregate functions:
  - `MIN`, `MAX`, `AVG`, `COUNT`, `SUM`
- Aggregate function can be used in different parts of SQL queries
  - select clause
  - having clause (*to be discussed later*)
  - order by clause (*to be discussed later*)

Aggregate functions

Query	Meaning
`SELECT MIN(A) FROM R`	Minimum values in A
`SELECT MAX(A) FROM R`	Maximum values in A
`SELECT AVG(A) FROM R`	Average 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

Query	Empty relation?	All `null` values with cardinality n?
`SELECT MIN(A) FROM R`	`null`	`null`
`SELECT MAX(A) FROM R`	`null`	`null`
`SELECT AVG(A) FROM R`	`null`	`null`
`SELECT SUM(A) FROM R`	`null`	`null`
`SELECT COUNT(A) FROM R`	`0`	`0`
`SELECT COUNT(*) FROM R`	`0`	`n`

Aggregate functions

Example

- Find the number, minimum price, maximum price, and average price of pizzas sold by Corleone Corner

Aggregate functions

Example

- Find the number, minimum price, maximum price, and average price of pizzas sold by Corleone Corner

```
SELECT COUNT(price), MIN(price),
       MAX(price)  , AVG(price)
FROM   Sells WHERE rname = ‘Corleone Corner’;
```

Aggregate functions

Example

- Find the number, minimum price, maximum price, and average price of pizzas sold by Corleone Corner

```
SELECT COUNT(price), 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
:	:	:

Output

count	*min*	*max*	*avg*
3	19	25	22.6667

Order by clause

Introduction

- Order by clause *sort the result* of SQL query
- Syntax
  - ORDER BY **column1** [ASC | DESC]
    [, **column2** [ASC | DESC] [...]]
- Example:
  - Find all customer name and the pizza the customer likes. Show the output in ascending order of the pizza, followed by in descending order of the customer name

```
SELECT   cname, pizza
FROM     Likes

                   ;
```

Likes

cname	*pizza*
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Siciliana
Ralph	Diavola

Output

cname	*pizza*
Ralph	Diavola
Moe	Funghi
Maggie	Funghi
Lisa	Funghi
Homer	Hawaiian
Homer	Margherita
Moe	Siciliana

Order by clause

Introduction

- Order by clause *sort the result* of SQL query
- Syntax
  - ```
    ORDER BY column1 [ ASC | DESC ]
        [ , column2 [ ASC | DESC ] [ ... ] ]
    ```
- Example:
  - Find all customer name and the pizza the customer likes. Show the output in ascending order of the pizza, followed by in descending order of the customer name

```
SELECT   cname, pizza
FROM     Likes
ORDER BY pizza ASC,
         cname DESC;
```

Likes

cname	*pizza*
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Siciliana
Ralph	Diavola

Output

cname	*pizza*
Ralph	Diavola
Moe	Funghi
Maggie	Funghi
Lisa	Funghi
Homer	Hawaiian
Homer	Margherita
Moe	Siciliana
```

Order by clause

Limit and offset

- Limit and offset allows retrieval of only a *portion of rows*
- Syntax
  - `LIMIT { number | ALL }`
  - `OFFSET number`
- Example:
  - Find the top 3 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

```
SELECT   pizza, rname, price
FROM     Sells
ORDER BY price DESC
          ;
```

Output

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

Order by clause

Limit and offset

- Limit and offset allows retrieval of only a *portion of rows*
- Syntax
  - `LIMIT { number | ALL }`
  - `OFFSET number`
- Example:
  - Find the top 3 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

```
SELECT   pizza, rname, price
FROM     Sells
ORDER BY price DESC
LIMIT    3;
```

Output

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

Order by clause

Limit and offset

- Limit and offset allows retrieval of only a *portion of rows*
- Syntax
  - `LIMIT { number | ALL }`
  - `OFFSET number`
- Example:
  - For each pizza that is sold by some restaurant, find the pizza name, the restaurant name, and its selling price; show the output in descending order of price and exclude the top 3 pizzas

```
SELECT   pizza, rname, price
FROM     Sells
ORDER BY price DESC
           ;
```

Output

pizza	*rname*	*price*
Marinara	Mamma's Place	23
Hawaiian	Pizza King	21
Margherita	Corleone Corner	19
Diavola	Pizza King	17
Sciliana	Gambino Oven	16

Order by clause

Limit and offset

- Limit and offset allows retrieval of only a *portion of rows*
- Syntax
  - `LIMIT { number | ALL }`
  - `OFFSET number`
- Example:
  - For each pizza that is sold by some restaurant, find the pizza name, the restaurant name, and its selling price; show the output in descending order of price and exclude the top 3 pizzas

```
SELECT   pizza, rname, price
FROM     Sells
ORDER BY price DESC
OFFSET   3;
```

Output

pizza	*rname*	*price*
Marinara	Mamma's Place	23
Hawaiian	Pizza King	21
Margherita	Corleone Corner	19
Diavola	Pizza King	17
Sciliana	Gambino Oven	16

Order by clause

Limit and offset

- Limit and offset allows retrieval of only a *portion of rows*
- Syntax
  - `LIMIT { number | ALL }`
  - `OFFSET number`
- Example:
  - Find the 4th and 5th most expensive pizzas. Show the pizza name, the name of the restaurant the sells it, and its selling price; show the output in descending order of price

```
SELECT   pizza, rname, price
FROM     Sells
ORDER BY price DESC

             ;
```

Output

pizza	*rname*	*price*
Marinara	Mamma's Place	23
Hawaiian	Pizza King	21

Order by clause

Limit and offset

- Limit and offset allows retrieval of only a _portion of rows_
- Syntax
  - `LIMIT { number | ALL }`
  - `OFFSET number`
- Example:
  - Find the 4th and 5th most expensive pizzas. Show the pizza name, the name of the restaurant the sells it, and its selling price; show the output in descending order of price

```
SELECT   pizza, rname, price
FROM     Sells
ORDER BY price DESC
LIMIT    2
OFFSET   3;
```

Output

pizza	*rname*	*price*
Marinara	Mamma's Place	23
Hawaiian	Pizza King	21

Group by clause

Introduction

- Group by clause *<u>divides the rows into groups</u>* such that aggregate functions can be applied to each group
- Syntax
  - `GROUP BY column1 [ , column2 [ ... ] ]`
- Example:
  - 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

Group by clause

Introduction

- Example:
  - For each restaurant that sells some pizza, find the minimum and maximum prices of its pizzas
- Conceptual steps:
  1. Partition the tuples in `Sells` into groups based on `rname`

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

Group by clause

Introduction

- Example:
  - For each restaurant that sells some pizza, find the minimum and maximum prices of its pizzas
- Conceptual steps:
  1. Partition the tuples in `Sells` into groups based on `rname`
  2. Compute `MIN(price)` and `MAX(price)` for each group

Sells

<u>*rname*</u>	<u>*pizza*</u>	*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**

Group by clause

Introduction

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

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

Output

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

Introduction

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

```
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  ...
;
```

Output

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

Introduction

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

```
SELECT rname, MIN(price), MAX(price)
FROM   Sells
GROUP BY rname;
```

Output

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

Introduction

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

Output

rname	*avgPrice*
Lorenzo Tavern	23.0000
Corleone Corner	22.6667
Mamma's Place	22.0000
Pizza King	19.0000
Gambino Oven	16.0000

```
SELECT   rname, AVG(price) AS avgPrice
FROM     Sells
GROUP BY rname
ORDER BY avgPrice DESC;
```

Group by clause

Introduction

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

Students

sid	*name*	*year*	*dept*
12345	Alice	1	CS
67890	Bob	1	Eng
11123	Carol	1	CS
20135	Eve	2	Eng
87012	Fred	2	CS

Output

dept	*year*	*count*
CS	1	2
CS	2	1
Eng	1	1
Eng	2	1

```
SELECT   dept, year, COUNT(*)
FROM     Students
GROUP BY dept, year
ORDER BY dept, year;
```

Group by clause

Grouping

- In a query with "`GROUP BY` $a_1, a_2, \dots, a_n$", two tuples t & t' belong to the same group if the following expression evaluates to true
  - $(t.a_1 \text{ IS NOT DISTINCT FROM } t'.a_1) \wedge \cdots \wedge (t.a_n \text{ IS NOT DISTINCT FROM } t'.a_n)$
  - In other words, all values are NOT DISTINCT
    - Two null values are considered non-distinct
- Example:
  - How many groups in R if R is grouped by $\{A, C\}$

R

A	*B*	*C*
null	1	19
null	2	19
6	1	null
6	20	null
20	2	10
1	1	2
1	18	2

Group by clause

Grouping

- In a query with "`GROUP BY` $a_1, a_2, \ldots, a_n$", two tuples t & t' belong to the same group if the following expression evaluates to true
  - $(t.a_1 \text{ IS NOT DISTINCT FROM } t'.a_1) \wedge \cdots \wedge (t.a_n \text{ IS NOT DISTINCT FROM } t'.a_n)$
  - In other words, all values are NOT DISTINCT
    - Two null values are considered non-distinct
- Each output tuple corresponds to one group

R

A	*B*	*C*
null	1	19
null	2	19
6	1	null
6	20	null
20	2	10
1	1	2
1	18	2

Group by clause

Grouping

- Condition
  - For each column A in relation R that appears in `SELECT`, *<u>one of the following condition must hold</u>*
    1. Column A appears in the `GROUP BY` clause
    2. Column A appears in aggregated expression in `SELECT` (e.g., `MIN(A)`)
    3. The primary (or candidate) key of R appears in the `GROUP BY` clause
  - If an aggregate function appears in `SELECT`, and there is no `GROUP BY` clause
    - Then the `SELECT` must not contain any column that is not in an aggregated expression

Group by clause

Grouping

- Condition
  - For each column A in relation R that appears in `SELECT`, <u>*one of the following condition must hold*</u>
    1. Column A appears in the `GROUP BY` clause
    2. Column A appears in aggregated expression in `SELECT` (e.g., `MIN(A)`)
    3. The primary (or candidate) key of R appears in the `GROUP BY` clause

Students

sid	*name*	*year*	*dept*
12345	Alice	1	CS
11123	Carol	1	CS
87012	Fred	2	CS
67890	Bob	1	Eng
20135	Eve	2	Eng

```
SELECT   dept, year, COUNT(*)
FROM     Students
GROUP BY dept;
```

Group by clause

Grouping

- Condition
  - If an aggregate function appears in `SELECT`, and there is no `GROUP BY` clause
    - Then the `SELECT` must not contain any column that is not in an aggregated expression

```
SELECT rname, MIN(price), MAX(price)
FROM   Sells;
```

```
SELECT MIN(price), MAX(price)
FROM   Sells;
```

Group by clause

Example

- Question:
  - For each restaurant that sells some pizza, find its name, area, and the average price of its pizzas
- Conceptual steps
  1. Get restaurants that sells pizza
  2. Partition by rname
  3. Output one tuple for each group

Sells NATURAL JOIN Restaurants

rname	*area*	*price*
Corleone Corner	North	24
Corleone Corner	North	25
Corleone Corner	North	19
Gambino Oven	Central	16
Lorenzo Tavern	Central	23
Mamma’s Place	South	22
Pizza King	East	17
Pizza King	East	21

output

rname	*area*	*avg*
Corleone Corner	North	19.00
Gambino Oven	Central	16.00
Lorenzo Tavern	Central	22.67
Mamma’s Place	South	23.00
Pizza King	East	23.00

Group by clause

Example

- Question:
  - For each restaurant that sells some pizza, find its name, area, and the average price of its pizzas
- Conceptual steps
  1. Get restaurants that sells pizza
  2. Partition by rname
  3. Output one tuple for each group

```
SELECT   R.rname, R.area
         AVG(S.price)
FROM     Sells S NATURAL JOIN
         Restaurants R
GROUP BY R.rname;
```

Sells NATURAL JOIN Restaurants

rname	*area*	*price*
Corleone Corner	North	24
Corleone Corner	North	25
Corleone Corner	North	19
Gambino Oven	Central	16
Lorenzo Tavern	Central	23
Mamma’s Place	South	22
Pizza King	East	17
Pizza King	East	21

output

rname	*area*	*avg*
Corleone Corner	North	19.00
Gambino Oven	Central	16.00
Lorenzo Tavern	Central	22.67
Mamma’s Place	South	23.00
Pizza King	East	23.00

Having clause

Introduction

- Example:
  - Find restaurants that sell pizzas with an average selling price of at least $22
- Conceptual steps
  1. Get restaurants that sells pizza
  2. Partition by rname
  3. Output one tuple for each group

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

```
SELECT   rname
FROM     Sells S
GROUP BY rname
WHERE    AVG(price) >= 22;
```

output

rname
Corleone Corner
Lorenzo Tavern
Mamma's Place

Having clause

Introduction

- Example:
  - Find restaurants that sell pizzas with an average selling price of at least $22
- Conceptual steps
  1. Get restaurants that sells pizza
  2. Partition by rname
  3. Output one tuple for each group

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

```
SELECT   rname
FROM     Sells S
GROUP BY rname
HAVING   AVG(price) >= 22;
```

output

rname
Corleone Corner
Lorenzo Tavern
Mamma's Place

Having clause

Properties

- Condition
  - For each column A in relation R that appears in `HAVING`, *one of the following condition must hold*
    1. Column A appears in the `GROUP BY` clause
    2. Column A appears in aggregated expression in `HAVING` (e.g., `MIN(A)`)
    3. The primary (or candidate) key of R appears in the `GROUP BY` clause

- Similar to `GROUP BY` but `SELECT` is replaced with `HAVING`

Summary

- Conceptual evaluation of queries
  - Query:

```
SELECT      DISTINCT select-list
FROM        from-list
WHERE       where-condition
GROUP BY    groupby-list
HAVING      having-condition
ORDER BY    orderby-list
LIMIT       limit-spec
OFFSET      offset-spec
```

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 because of `DISTINCT`
7. Sort the output tuples based on the **`orderby-list`**
8. Remove the appropriate output tuples based on the **`limit-spec`** & **`offset-spec`**

- Aggregate functions
  - Introduction
  - Order by clause
  - Group by clause
  - Having clause
- Subqueries
  - select clause
  - from clause
  - where clause
  - Queries with universal quantification

Subqueries

Subqueries

Introduction

- We let an SQL query be of the form

```
SELECT [ DISTINCT ] select_list  -- select clause
FROM                from_list    -- from clause
[ WHERE             expression ] -- where clause
```

- A subquery is an SQL query that may be added into
  - select clause
    - Scalar subqueries
  - from clause
    - Common table expression
  - where clause
    - `EXISTS`; `IN`; `ANY/SOME`; `ALL`

Subqueries

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

Subqueries

Scoping rules

- 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
- Example:
  - Given `R(a,b)`
    - ```
      SELECT S.a FROM ( SELECT * FROM R ) AS S;
      ```
    - ```
      SELECT * FROM ( SELECT * FROM R ) AS S, R;
      ```
    - ```
      SELECT * FROM ( SELECT * FROM R ) AS S
      WHERE  S.a = 1;
      ```
    - ```
      SELECT *
      FROM ( SELECT * FROM R ) AS S,
           ( SELECT * FROM S ) AS T;
      ```

select clause

Scalar subqueries

- A scalar subquery is a subquery that returns *at most one tuple with one column*
  - If the result of the subquery is empty, its return value is `null`
- A scalar subquery can be used as a scalar expression
- Example:
  - ```
    SELECT rname
    FROM Restaurants
    WHERE rname = ‘Corleone Corner’;
    ```
  - ```
    SELECT *
    FROM Restaurants
    WHERE rname = ‘Corleone Corner’;
    ```

Output

rname
Corleone Corner

Output

rname	*area*
Corleone Corner	North

select clause

Scalar subqueries

- For each restaurant that sells Funghi, find its name, area, and selling price
  - name and price can be obtained from `Sells`
  - area can be obtained from `Restaurants`
  - Attempt #1
    - Natural join

```
SELECT R.rname, R.area, S.price
FROM   Sells S NATURAL JOIN Restaurants R
WHERE  S.pizza = 'Funghi';
```

  - Attempt #2
    - Scalar subquery

```
SELECT rname,
       (SELECT R.area FROM Restaurants R
        WHERE R.rname = S.rname), price
FROM   Sells S   WHERE pizza = 'Funghi';
```

from clause

Common table expressions (CTEs)

- A table expression computes a table
  - Computing a table is a subquery!
- A common table expression is a *temporary named result* set that can be queried
  - Syntax:

```
WITH
   cte1 AS (subquery1) [,
   cte2 AS (subquery2) [ ... ] ]
query
```

  - Property:
    - Each $\texttt{cte}_i$ is the name of a temporary relation defined by $\texttt{subquery}_i$
    - `query` is the SQL statement that references cte
    - CTEs can be used for writing recursive queries (*not covered*)

from clause

Example

- Given
  - `Courses (cid, cname, credits)`
  - `Enrolls (sid, cid, grade)`
- 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.
  - Assume that `cname` is a candidate key for `Courses`
  - Idea:
    - Get the count for “Database Systems”
    - Compare the count
    - Get the pair (courses, number enrolled)

from clause

Example

- Idea:
  - Get the count for “Database Systems”
  - Compare the count
  - Get the pair (courses, number enrolled)

```
SELECT   C.cname, ( SELECT COUNT(*)
                    FROM   Enrolls E
                    WHERE  E.cid = C.cid ) AS num
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’ );
```

from clause

Example

- Somewhat duplicated?
  - If we rearrange

```
SELECT   cname, num
FROM   ( SELECT C.cid, C.cname, COUNT(*) AS num
         FROM   Courses C NATURAL JOIN Enrolls E
         GROUP BY C.cid ) AS X
WHERE    num >
       ( SELECT COUNT(*)
         FROM   Courses C NATURAL JOIN Enrolls E
         WHERE  C.cname = ‘Database Systems’ );
```

from clause

Example

- Somewhat duplicated?
  - If we rearrange
  - But we cannot refer to X!
    - Without CTE

```
WITH X AS
   ( SELECT   C.cid, C.cname, COUNT(*) AS num
     FROM     Courses C NATURAL JOIN Enrolls E
     GROUP BY C.cid )
SELECT cname, num
FROM   X
WHERE  num >
     ( SELECT num FROM   X
       WHERE  cname = 'Database Systems' );
```

where clause

Types of subqueries

- `EXISTS` subqueries
  - Returns true if the result subquery is non-empty
  - Otherwise, false
- `IN` subqueries
  - Subquery must return exactly one column
  - Returns false if the result of subquery is empty
  - Otherwise, for subquery result $= \{v_1, \ldots v_n\}$ and expression result v; return the result of the boolean expression

$$\big((v = v_1) \vee (v = v_2) \vee \cdots \vee (v = v_n)\big)$$

where clause

Types of subqueries

- `ANY/SOME` subqueries
  - Subquery must return exactly one column
  - Returns false if the result of subquery is empty
  - Otherwise, for subquery result $= \{v_1, \ldots v_n\}$, expression result v, and operator $\oplus$; return the result of the boolean expression
    $$\big((v \oplus v_1) \lor (v \oplus v_2) \lor \cdots \lor (v \oplus v_n)\big)$$
- `ALL` subqueries
  - Subquery must return exactly one column
  - Returns false if the result of subquery is empty
  - Otherwise, for subquery result $= \{v_1, \ldots v_n\}$, expression result v, and operator $\oplus$; return the result of the boolean expression
    $$\big((v \oplus v_1) \land (v \oplus v_2) \land \cdots \land (v \oplus v_n)\big)$$

where clause

EXISTS subqueries

- Example:
  - Find distinct customers who like some pizza sold by “Corleone Corner”
- Idea:
  - Find pizza sold by “Corleone Corner”
  - Check if customer likes the pizza
  - Remove duplicates
- Query:

```
SELECT DISTINCT L.cname
FROM   Likes L INNER JOIN Sells S
       ON S.pizza = L.pizza
WHERE  S.rname = ‘Corleone Corner’;
```

where clause

EXISTS subqueries

- Example:
  - Find distinct customers who like some pizza sold by “Corleone Corner”
- Idea:
  - Find pizza sold by “Corleone Corner”
  - Check if customer likes the pizza
  - Remove duplicates
- Query:

```
SELECT DISTINCT cname
FROM   Likes L
WHERE  EXISTS (
         SELECT 1 FROM Sells S
         WHERE  S.rname = ‘Corleone Corner’
           AND  S.pizza = L.pizza );
```

where clause

NOT EXISTS subqueries

- Example:
  - Find distinct customers who does not like some pizza sold by “Corleone Corner”
- Idea:
  - Find customers who likes pizza sold by “Corleone Corner”
  - Set difference
- Query:

```
SELECT cname FROM Customers
EXCEPT
SELECT L.cname
FROM   Likes L INNER JOIN Sells S
       ON S.pizza = L.pizza
WHERE  S.rname = ‘Corleone Corner’;
```

where clause

NOT EXISTS subqueries

- Example:
  - Find distinct customers who does not like some pizza sold by “Corleone Corner”
- Idea:
  - Invert condition
  - `NOT EXISTS`
- Query:

```
SELECT DISTINCT cname
FROM   Likes L
WHERE  NOT EXISTS (
         SELECT 1 FROM Sells S
         WHERE  S.rname = ‘Corleone Corner’
           AND  S.pizza = L.pizza );
```

where clause

IN subqueries

- Example:
  - Find pizzas that contain ham or seafood
  - UNION

```
SELECT pizza FROM Contains WHERE ingredient = ‘ham’
UNION
SELECT pizza FROM Contains WHERE ingredient =
‘seafood’;
```

  - OR

```
SELECT DISTINCT pizza FROM Contains
WHERE  ingredient = ‘ham’ OR ingredient = ‘seafood’;
```

  - IN

```
SELECT DISTINCT pizza FROM Contains
WHERE  ingredient IN (‘ham’, ‘seafood’);
```

where clause

IN subqueries

- Example:
  - Find distinct customers who like some pizza sold by “Corleone Corner”
- Idea:
  - Find pizza sold by “Corleone Corner”
  - Check if customer likes the pizza
  - Remove duplicates
- Query:

```
SELECT DISTINCT cname
FROM   Likes L
WHERE  pizza IN (
         SELECT pizza FROM Sells S
         WHERE  rname = ‘Corleone Corner’
       );
```

where clause

ANY/SOME subqueries

- Example:
  - 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.
- Idea:
  - Find pizza sold by “Corleone Corner”
  - Check if price more expensive
  - Remove duplicates and “Corleone Corner”
- Query:

```
SELECT DISTINCT rname FROM Sells S
WHERE  rname <> ‘Corleone Corner’ AND EXISTS (
   SELECT 1 FROM Sells S2 WHERE S.price > S2.price
      AND rname = ‘Corleone Corner’ );
```

where clause

ANY/SOME subqueries

- Example:
  - 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.
- Idea:
  - Find pizza sold by “Corleone Corner”
  - Check if price more expensive
  - Remove duplicates and “Corleone Corner”
- Query:

```
SELECT DISTINCT rname FROM Sells
WHERE rname <> ‘Corleone Corner’ AND price > ANY (
        SELECT price FROM Sells
        WHERE  rname = ‘Corleone Corner’ );
```

where clause

ANY/SOME subqueries

- Example:
  - For each restaurant, find the name, and price of its most expensive pizzas. Exclude restaurants that do no sell any pizza.
- Idea:
  - Find pizza that has other more expensive pizza
  - Remove these pizzas
- Query:

```
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
       );
```

where clause

ALL subqueries

- Example:
  - For each restaurant, find the name, and price of its most expensive pizzas. Exclude restaurants that do no sell any pizza.
- Idea:
  - Find pizza that is more expensive or as expensive than all other pizzas
- Query:

```
SELECT rname, pizza, price FROM Sells S1
WHERE  price >= ALL (
         SELECT S2.price FROM Sells S2
         WHERE S2.rname = S1.rname
       );
```

Queries with universal quantification

Example

- Given:
  - `Courses (courseID, name, dept)`
  - `Students (studentID, name, birthdate)`
  - `Enrolls (sid, cid, grade)`
- Question:
  - Find the names of all students who have enrolled in all the courses offered by CS department

Queries with universal quantification

Example

- Question:
  - Find the names of all students who have enrolled in all the courses offered by CS department
- Analysis:
  - Let R denote the set of all students who have enrolled in all the courses offered by CS department
  - Let $\bar{R} = \text{Students} - R$
    = set of all students who have NOT enrolled ... CS department
  - A student $s \in \bar{R}$ 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
  - $\bar{R}$ = $\{s \in \text{Students} \mid F(s.\text{studentID}) \neq \emptyset\}$

Queries with universal quantification

Example

- Question:
  - Find the names of all students who have enrolled in all the courses offered by CS department
- Analysis:
  - $\bar{R}$ = $\{s \in \text{Students} \mid F(s.\text{studentID}) \neq \emptyset\}$
  - $\bar{R}$ 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

Example

- Question:
  - Find the names of all students who have enrolled in all the courses offered by CS department
- Solution: F(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

Example

- Question:
  - Find the names of all students who have enrolled in all the courses offered by CS department
- Solution: R

```
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 = x )
);
```

Summary

- A subquery is an SQL query that may be added into
  - select clause
    - Scalar subqueries
    - One column and one tuple
  - from clause
    - Common table expression
  - where clause
    - `EXISTS`; `IN`; `ANY/SOME`; `ALL`