CS2102 Database Systems

Slides adapted from Prof. Chan Chee Yong

LECTURE 05

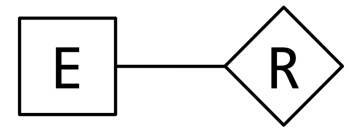
SQL #2

PREVIOUS LECTURE

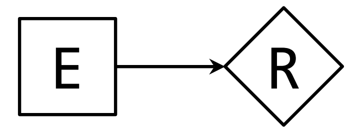
Relationship constraints

Types

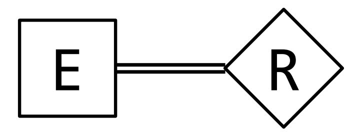
- Many-to-many — Each instance of E participates in *0 or more* instance of R



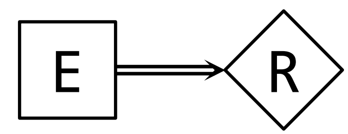
- Key — Each instance of E participates in *at most 1* instance of R



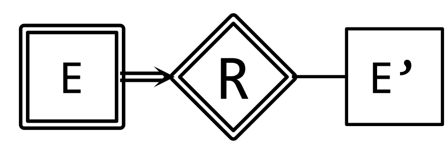
- Total — Each instance of E participates in *at least 1* instance of R



- 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

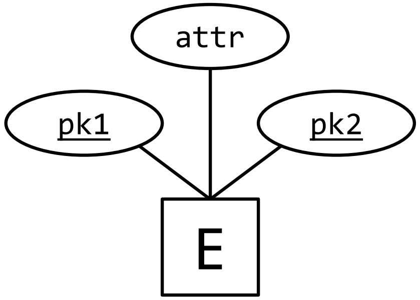


PREVIOUS LECTURE

ER diagram to SQL

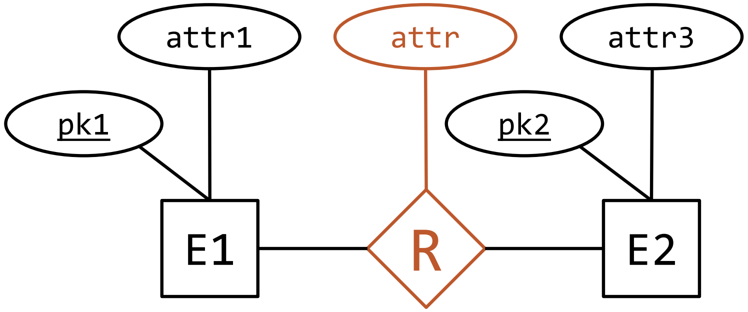
Entity sets

```
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)
);
```

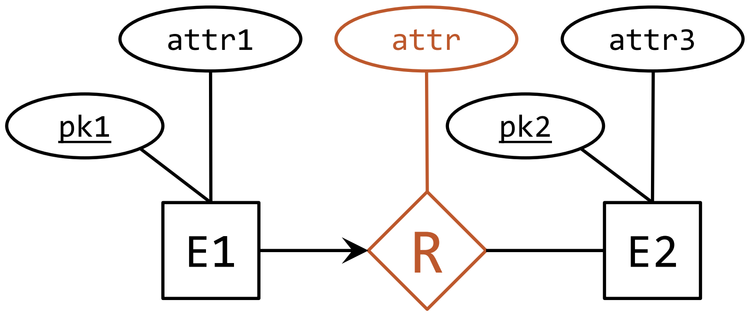


PREVIOUS LECTURE

ER diagram to SQL

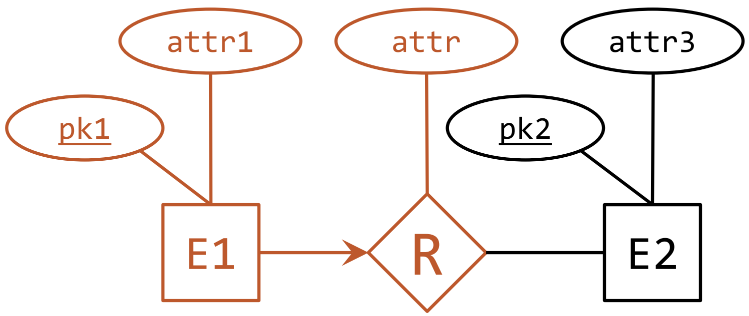
Key constraints approach #1

```
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
);
```

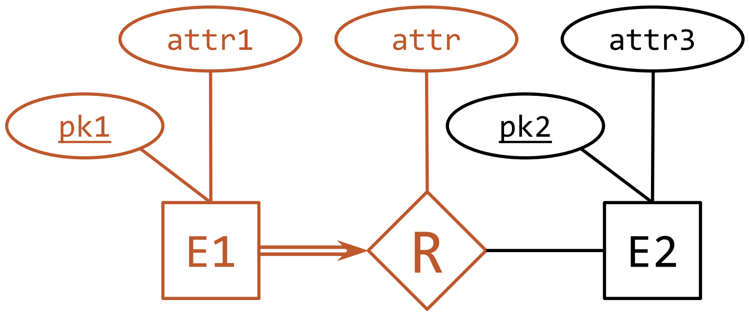


PREVIOUS LECTURE

ER diagram to SQL

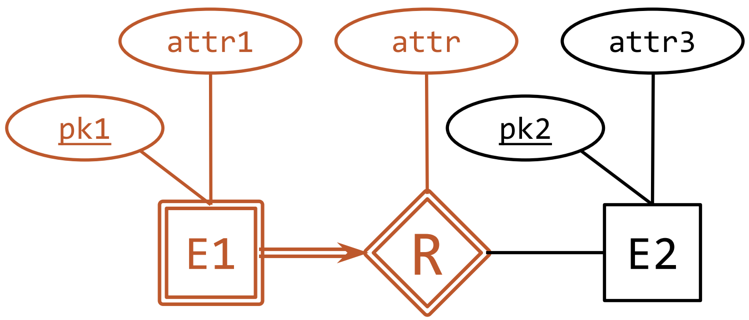
Key & total constraints

```
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)
);
```

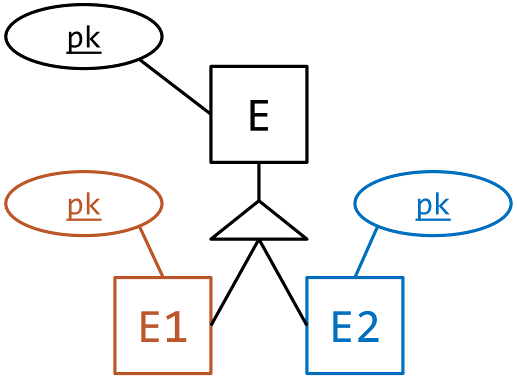


PREVIOUS LECTURE

Additional ER concepts

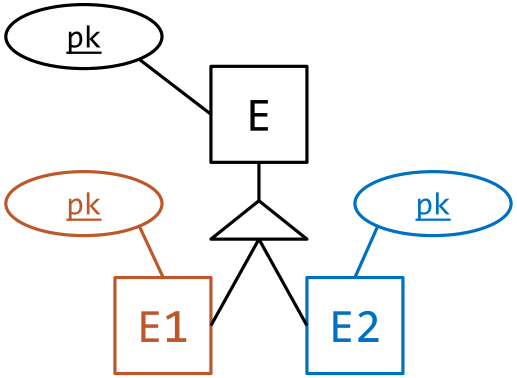
ISA hierarchies approach #1

```
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
);
```



Additional ER concepts

ISA hierarchies approach #2



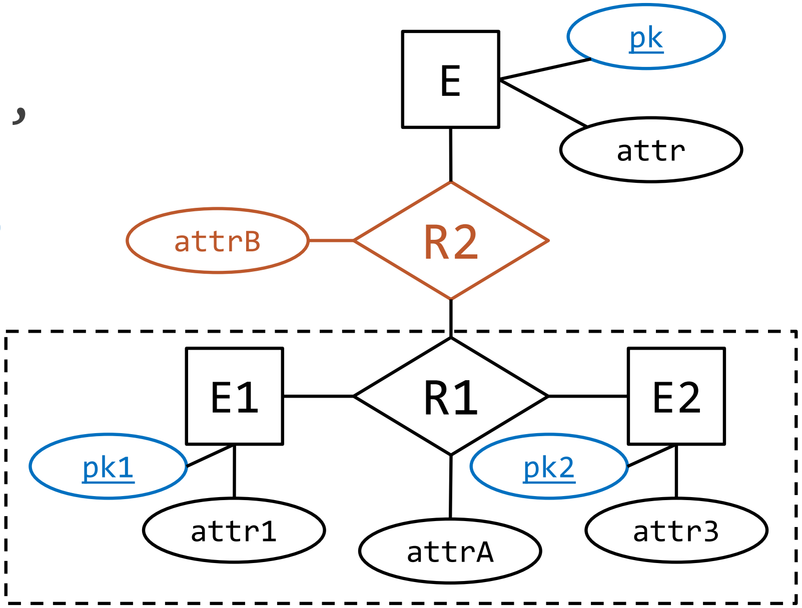

```
CREATE TABLE E1 (
  pk     type
         PRIMARY KEY

);

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)
);
```



- More conditions
  - Pattern matching
  - Conditional expressions: case analysis
  - Conditional expressions: null analysis
- Multi-relation queries
  - Set operations
  - Cross-product
  - Join: inner join, left outer join, right outer join, full outer join, natural join
- Views

Overview

- More conditions
  - Pattern matching
  - Conditional expressions: case analysis
  - Conditional expressions: null analysis
- Multi-relation queries
  - Set operations
  - Cross-product
  - Join: inner join, left outer join, right outer join, full outer join, natural join
- Views

More conditions

More conditions

Basic

- Syntax

```
SELECT [ DISTINCT ]  select_list
FROM                 from_list
WHERE                condition
```

- What can the *condition* be?
  - Pattern matching `LIKE`
  - Conditional expression
    - Case analysis `CASE ... END`
    - Null analysis `NULLIF, COALESCE`

More conditions

Pattern matching

- Syntax

  `attr LIKE pattern`
- Rules for pattern
  - Underscore (_) match any <u>*single character*</u>
  - Percent (%) match sequence of <u>*0 or more characters*</u>
- Example:
  - `‘abc’ LIKE ‘abc’`
  - `‘abc’ LIKE ‘a%’`
  - `‘abc’ LIKE ‘_b_’`
  - `‘abc’ LIKE ‘c’`

More conditions

Pattern matching

- Syntax

  `attr LIKE pattern`

- Rules for pattern
  - Underscore (_) match any *single character*
  - Percent (%) match sequence of *0 or more characters*
- Example:
  - `‘abc’ LIKE ‘abc’` True
  - `‘abc’ LIKE ‘a%’` True
  - `‘abc’ LIKE ‘_b_’` True
  - `‘abc’ LIKE ‘c’` False

More conditions

Pattern matching

- Question:
  - Find all customer names ending with "e" that consists of at least four characters
- Query:

```
SELECT cname FROM Customers
WHERE cname LIKE '_ _ _ % e';
```

Customers

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

Output

cname
Maggie
Willie

More conditions

Pattern matching

- Question:
  - Find all customer names starting with “M” and ending with “e”
- Query:

```
SELECT cname FROM Customers
WHERE cname LIKE           ;
```

Customers

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

Output

cname
Maggie
Moe

More conditions

Pattern matching

- Question:
  - Find all customer names starting with "M" and ending with "e"
- Query:

```
SELECT cname FROM Customers
WHERE cname LIKE  'M % e'  ;
```

Customers

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

Output

cname
Maggie
Moe

More conditions

Conditional expressions

- Case analysis
  - Syntax

```
CASE [expression]
  WHEN condition THEN result
  [ WHEN ... ]
  [ ELSE result ]
END
```

More conditions

Conditional expressions

- Case analysis
  1. If the student get marks >= 85, the student get grade ‘A’
  2. Else if the student get marks >= 75 on marks, the student get grade ‘B’
  3. Else if the student get marks >= 65 on marks, the student get grade ‘C’
  4. Otherwise, the student get grade ‘D’
- Query

```
SELECT name, CASE
  WHEN ___________ THEN
  WHEN ___________ THEN
  WHEN ___________ THEN
  ELSE
END AS grade
FROM Scores;
```

Scores

name	*marks*
Alice	92
Bob	78
Carol	65
Dave	47

Output

name	*grades*
Alice	A
Bob	B
Carol	C
Dave	D

More conditions

Conditional expressions

- Case analysis
  1. If the student get marks >= 85, the student get grade ‘A’
  2. Else if the student get marks >= 75 on marks, the student get grade ‘B’
  3. Else if the student get marks >= 65 on marks, the student get grade ‘C’
  4. Otherwise, the student get grade ‘D’
- Query

```
SELECT name, CASE
  WHEN marks >= 85 THEN ‘A’        -- #1
  WHEN marks >= 75 THEN ‘B’        -- #2
  WHEN marks >= 65 THEN ‘C’        -- #3
  ELSE ‘D’ -- #4
END AS grade
FROM Scores;
```

Scores

name	*marks*
Alice	92
Bob	78
Carol	65
Dave	47

Output

name	*grades*
Alice	A
Bob	B
Carol	C
Dave	D

More conditions

Conditional expressions

- Null analysis
  - **Scenario** student may attempt a quiz
    - On passing the value is ‘`pass`’; the status is marked as ‘`pass`’
    - On failing the value is ‘`fail`’, the status is marked as ‘`fail`’
    - On absent the value is ‘`absent`’, the status is marked as `null`

```
SELECT name, CASE
  WHEN (_______________) ____ (_______________)
  THEN result
END AS status
FROM Quizzes;
```

Quizzes

name	*result*
Alice	absent
Bob	fail
Carol	pass
Dave	absent
Eve	pass

Output

name	*status*
Alice	null
Bob	fail
Carol	pass
Dave	null
Eve	pass

More conditions

Conditional expressions

- Null analysis
  - **Scenario** student may attempt a quiz
    - On passing the value is ‘`pass`’; the status is marked as ‘`pass`’
    - On failing the value is ‘`fail`’, the status is marked as ‘`fail`’
    - **On absent the value is ‘`absent`’, the status is marked as `null`**

```
SELECT name, CASE
  WHEN (result = ‘pass’) OR (result = ‘fail’)
  THEN result
END AS status
FROM Quizzes;
```

Quizzes

name	***result***
Alice	absent
Bob	fail
Carol	pass
Dave	absent
Eve	pass

Output

name	***status***
Alice	null
Bob	fail
Carol	pass
Dave	null
Eve	pass

More conditions

Conditional expressions

- Null analysis
  - **Scenario** student may attempt a quiz
  - On passing the value is ‘`pass`’; the status is marked as ‘`pass`’
  - On failing the value is ‘`fail`’, the status is marked as ‘`fail`’
  - **On absent the value is ‘`absent`’, the status is marked as `null`**

```
SELECT name, NULLIF (result, ‘absent’) AS status
FROM Quizzes;
```

- `NULLIF (`$value_1$`, `$value_2$`)`
- Returns `null`
  - if $value_1$ equals to $value_2$
  - otherwise, $value_1$

Quizzes

name	***result***
Alice	absent
Bob	fail
Carol	pass
Dave	absent
Eve	pass

Output

name	***status***
Alice	null
Bob	fail
Carol	pass
Dave	null
Eve	pass

More conditions

Conditional expressions

- Null analysis
  - **Scenario** student may attempt quiz up to 3 times
    - On passing the value is ‘`pass`’; cannot attempt further quizzes
    - On failing the value is ‘`fail`’, may attempt further quizzes
    - On absent the value is `null`, may attempt further quizzes

```
SELECT name, CASE
  WHEN (______________) OR (______________) OR
       (______________)
  THEN ‘pass’
  ELSE ‘fail’
END AS result
FROM Quiz3;
```

Quiz3

name	*first*	*second*	*third*
Alice	pass	null	null
Bob	fail	pass	null
Carol	fail	fail	pass
Dave	fail	null	fail
Eve	fail	fail	null

Output

name	*result*
Alice	pass
Bob	pass
Carol	pass
Dave	fail
Eve	fail

More conditions

Conditional expressions

- Null analysis
  - **Scenario** student may attempt quiz up to 3 times
    - On passing the value is '`pass`'; cannot attempt further quizzes
    - On failing the value is '`fail`', may attempt further quizzes
    - On absent the value is `null`, may attempt further quizzes

```
SELECT name, CASE
  WHEN (first = 'pass') OR (second = 'pass') OR
       (third = 'pass')
  THEN 'pass'
  ELSE 'fail'
END AS result
FROM Quiz3;
```

Quiz3

<u>*name*</u>	*first*	*second*	*third*
Alice	pass	null	null
Bob	fail	pass	null
Carol	fail	fail	pass
Dave	fail	null	fail
Eve	fail	fail	null

Output

<u>*name*</u>	*result*
Alice	pass
Bob	pass
Carol	pass
Dave	fail
Eve	fail

More conditions

Conditional expressions

- Null analysis
  - **Scenario** student may attempt quiz up to 3 times
    - On passing the value is ‘`pass`’; cannot attempt further quizzes
    - On failing the value is ‘`fail`’, may attempt further quizzes
    - On absent the value is `null`, may attempt further quizzes

```
SELECT name,
       COALESCE (third, second, first) AS result
FROM Quiz3;  -- does order matter?
```

- `coalesce` returns the first non-null value in its argument
- `null` is returned if all arguments are `null`

Quiz

name	*first*	*second*	*third*
Alice	pass	null	null
Bob	fail	pass	null
Carol	fail	fail	pass
Dave	fail	null	fail
Eve	fail	fail	null

Output

name	*result*
Alice	pass
Bob	pass
Carol	pass
Dave	fail
Eve	fail

- More conditions
  - Pattern matching
  - Conditional expressions: case analysis
  - Conditional expressions: null analysis
- Multi-relation queries
  - Set operations
  - Cross-product
  - Join: inner join, left outer join, right outer join, full outer join, natural join
- Views

Multi-relation queries

Multi-relation queries

Set operations

- Let Q_1 and Q_2 denote SQL queries that output union-compatible relations
  - $Q_1 \cup Q_2$ `Q₁ UNION Q₂` `Q₁ UNION ALL Q₂`
  - $Q_1 \cap Q_2$ `Q₁ INTERSECT Q₂` `Q₁ INTERSECT ALL Q₂`
  - $Q_1 - Q_2$ `Q₁ EXCEPT Q₂` `Q₁ EXCEPT ALL Q₂`

  - `UNION, INTERSECT, EXCEPT`
    - eliminates duplicate records
  - `UNION ALL, INTERSECT ALL, EXCEPT ALL`
    - preserves duplicate records

Multi-relation queries

Set operations

- **Union**: *Find all customer/restaurant names*

```
SELECT cname FROM Customers
UNION
SELECT rname FROM Restaurants;
```

- **Intersect**: *Find all pizzas that contain both cheese and chilli*

```
SELECT pizza FROM Contains WHERE ingredient = ‘cheese’
INTERSECT
SELECT pizza FROM Contains WHERE ingredient = ‘chilli’;
```

- **Set difference**: *Find all pizzas that contain cheese but not chilli*

```
SELECT pizza FROM Contains WHERE ingredient = ‘cheese’
EXCEPT
SELECT pizza FROM Contains WHERE ingredient = ‘chilli’;
```

Multi-relation queries

Set operations

- ALL vs no ALL
  - Q1:
    ```
    SELECT B FROM R
    EXCEPT
    SELECT D FROM S;
    ```
  - Q2:
    ```
    SELECT B FROM R
    EXCEPT ALL
    SELECT D FROM S;
    ```

R

<u>A</u>	B
10	1
20	1
30	1
40	2
50	3
60	4
70	4

S

<u>C</u>	D
10	1
20	5
30	2
40	2
50	3

Q1

B
4

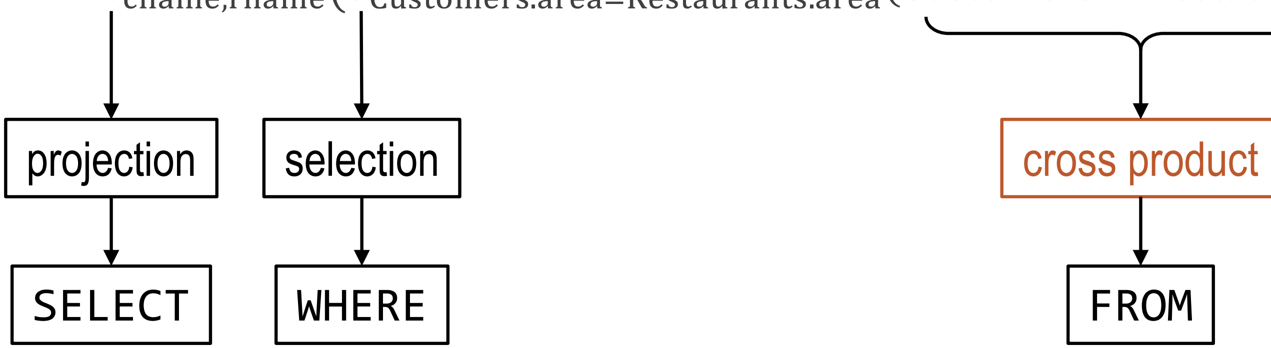
Q2

B
1
1
4
4

Multi-relation queries

Cross-product

- Find distinct pairs of customers and restaurants that are located in the same area
- Relational algebra expression
  - $\pi_{\text{cname,rname}}\left(\sigma_{\text{Customers.area=Restaurants.area}}(\text{Customers} \times \text{Restaurants})\right)$



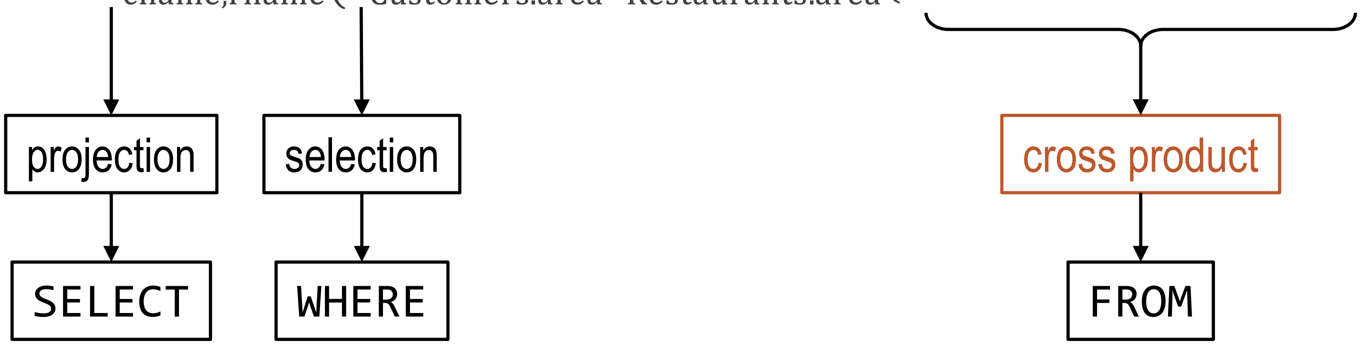
- SQL

```
SELECT cname, rname
FROM   Customers, Restaurants
WHERE  Customers.area = Restaurants.area;
```

Multi-relation queries

Cross-product

- Find distinct pairs of customers and restaurants that are located in the same area
- Relational algebra expression
  - $\pi_{\text{cname,rname}}\left(\sigma_{\text{Customers.area=Restaurants.area}}(\text{Customers} \times \text{Restaurants})\right)$



- SQL *with renaming*

```
SELECT C.cname, R.rname
FROM   Customers AS C, Restaurants AS R
WHERE  C.area = R.area;
```

Multi-relation queries

Cross-product

- Find distinct pairs of customers and restaurants that are located in the same area
- Relational algebra expression
  - $\pi_{\text{cname,rname}}\left(\sigma_{\text{Customers.area=Restaurants.area}}(\text{Customers} \times \text{Restaurants})\right)$

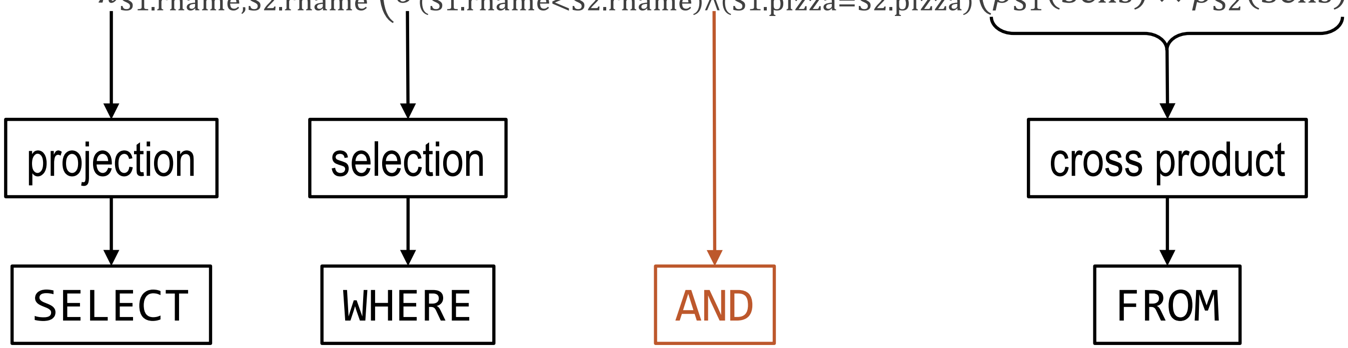
- SQL *with renaming*

```
SELECT C.cname, R.rname
FROM   Customers C, Restaurants R
WHERE  C.area = R.area;
```

Multi-relation queries

Cross-product

- Find distinct pairs of restaurants (R_1, R_2) where $R_1 < R_2$ and they sell some common pizza
- Relational algebra expression
  - $\pi_{\text{S1.rname,S2.rname}}\left(\sigma_{(\text{S1.rname}<\text{S2.rname})\wedge(\text{S1.pizza}=\text{S2.pizza})}\left(\rho_{\text{S1}}(\text{Sells}) \times \rho_{\text{S2}}(\text{Sells})\right)\right)$



- SQL *with renaming*

```
SELECT DISTINCT S1.rname, S2.rname
FROM   Sells S1, Sells S2
WHERE  S1.rname < S2.rname
   AND S1.pizza = S2.pizza;
```

Join

Join operators in relational algebra

- A join operator *combines* cross-product, selection, and possibly projection operators
  - More convenient that plain cross-product operator
- Types:
  - Inner join a.k.a. join $\bowtie_c$
  - Left outer join a.k.a. left join $\rightarrow_c$
  - Right outer join a.k.a. right join $\leftarrow_c$
  - Full outer join a.k.a. full join $\leftrightarrow_c$
  - Natural join $\bowtie$
  - *Others: natural left/right/full outer joins*

Join

Inner join: $R \bowtie_c S$

- The inner join of R and S is defined as $R \bowtie_c S \equiv \sigma_c(R \times S)$
- Example:
  - Find customer pairs (C_1, C_2) such that they like some common pizza and $C_1 < C_2$
  - $\pi_{\text{cname1,cname2}}\left(\sigma_c\left(\text{Likes} \times \rho_{(\text{cname2,pizza2})}(\text{Likes})\right)\right)$
  - $\pi_{\text{cname1,cname2}}\left(\quad\left(\text{Likes} \bowtie_c \rho_{(\text{cname2,pizza2})}(\text{Likes})\right)\right)$

where $c = (\text{pizza} = \text{pizza2}) \wedge (\text{cname} < \text{cname2})$

Likes

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

$\pi_{\text{cname1,cname2}}\left(\left(\text{Likes} \bowtie_c \rho_{(\text{cname2,pizza2})}(\text{Likes})\right)\right)$

cname	*cname2*
Lisa	Maggie
Lisa	Moe
Maggie	Moe

Join

Inner join: $R \bowtie_c S$

- The inner join of R and S is defined as $R \bowtie_c S \equiv \sigma_c(R \times S)$
- Example:
  - Find customer pairs (C_1, C_2) such that they like some common pizza and $C_1 < C_2$

```
SELECT DISTINCT L1.cname, L2.cname
FROM   Likes L1, Likes L2
WHERE  L1.cname < L2.cname
  AND  L1.pizza = L2.pizza;
```

Likes

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

where $c = (\text{pizza} = \text{pizza2}) \wedge (\text{cname} < \text{cname2})$

$\pi_{\text{cname1,cname2}}\left(\left(\text{Likes} \bowtie_c \rho_{(\text{cname2,pizza2})}(\text{Likes})\right)\right)$

cname	*cname2*
Lisa	Maggie
Lisa	Moe
Maggie	Moe

Join

Inner join: $R \bowtie_c S$

- The inner join of R and S is defined as $R \bowtie_c S \equiv \sigma_c(R \times S)$
- Example:
  - Find customer pairs (C_1, C_2) such that they like some common pizza and $C_1 < C_2$

```
SELECT DISTINCT L1.cname, L2.cname
FROM   Likes L1 INNER JOIN Likes L2
       ON  (L1.pizza = L2.pizza)
       AND (L1.cname < L2.cname);
```

Likes

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

where $c = (\text{pizza} = \text{pizza2}) \wedge (\text{cname} < \text{cname2})$

$\pi_{\text{cname1,cname2}}\left(\left(\textbf{Likes} \bowtie_c \rho_{(\text{cname2,pizza2})}(\textbf{Likes})\right)\right)$

cname	*cname2*
Lisa	Maggie
Lisa	Moe
Maggie	Moe

Join

Inner join: $R \bowtie_c S$

- The inner join of R and S is defined as $R \bowtie_c S \equiv \sigma_c(R \times S)$
- Example:
  - Find customer pairs (C_1, C_2) such that they like some common pizza and $C_1 < C_2$

```
SELECT DISTINCT L1.cname, L2.cname
FROM   Likes L1 JOIN Likes L2
       ON  (L1.pizza = L2.pizza)
       AND (L1.cname < L2.cname);
```

Likes

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

where $c = (\text{pizza} = \text{pizza2}) \land (\text{cname} < \text{cname2})$

$\pi_{\text{cname1,cname2}}\left(\left(\text{Likes} \bowtie_c \rho_{(\text{cname2,pizza2})}(\text{Likes})\right)\right)$

cname	*cname2*
Lisa	Maggie
Lisa	Moe
Maggie	Moe

Join

Dangling tuple

- A dangling tuple is a tuple in a join operand that *does not participate* in the join operation
  - Inner join ignores dangling tuples
  - To preserve dangling tuple, use **outerjoins**

R

A	*B*	*C*
0	x	100
2	y	100
4	w	400
5	z	200

S

D	*E*	*F*	*G*
a	100	0	i
b	300	1	j
c	200	5	k

$R \bowtie_{R.A=S.F} S$

A	*B*	*C*	*D*	*E*	*F*	*G*
0	x	100	a	100	0	i
5	z	200	c	200	5	k

Join

Dangling tuple

- A dangling tuple is a tuple in a join operand that *does not participate* in the join operation
  - Inner join ignores dangling tuples
  - To preserve dangling tuple, use **outerjoins**

R

A	B	C
0	x	100
2	y	100
4	w	400
5	z	200

S

D	E	F	G
a	100	0	i
b	300	1	j
c	200	5	k

$R \bowtie_{R.A=S.F} S$

A	B	C	D	E	F	G
0	x	100	a	100	0	i
5	z	200	c	200	5	k

Join

Outerjoin

R

A	B	C
0	x	100
2	y	100
4	w	400
5	z	200

S

D	E	F	G
a	100	0	i
b	300	1	j
c	200	5	k

R $\bowtie_{R.A=S.F}$ S

A	B	C	D	E	F	G
0	x	100	a	100	0	i
5	z	200	c	200	5	k

R $\rightarrow_{R.A=S.F}$ S [left outer join]

A	B	C	D	E	F	G
0	x	100	a	100	0	i
5	z	200	c	200	5	k
2	y	100	null	null	null	null
4	w	400	null	null	null	null

R $\leftarrow_{R.A=S.F}$ S [right outer join]

A	B	C	D	E	F	G
0	x	100	a	100	0	i
5	z	200	c	200	5	k
null	null	null	b	300	1	j

R $\leftrightarrow_{R.A=S.F}$ S [full outer join]

A	B	C	D	E	F	G
0	x	100	a	100	0	i
5	z	200	c	200	5	k
2	y	100	null	null	null	null
4	w	400	null	null	null	null
null	null	null	b	300	1	j

Join

Outerjoin definition

- Let `attr(R)` denotes the list of attributes in the schema of `R`
  - Example: `attr(Sells) = rname, pizza, price`
- Let `null(R)` denotes an *n*-component tuple of `null` values
  - *n* is the *arity* of relation `R`
  - Example: `null(Sells) = (null, null, null)`

Join

Outerjoin definition

- Let `attr(R)` denotes the list of attributes in the schema of `R`
  - Example: `attr(Sells) = rname, pizza, price`
- Let `null(R)` denotes an *n*-component tuple of `null` values
  - *n* is the *arity* of relation `R`
  - Example: `null(Sells) = (null, null, null)`
- Let $R \rhd_c S$ be $R - \pi_{\text{attr}(R)}(R \bowtie_c S)$
  - Example: let c be $(\mathrm{R.A} = \mathrm{S.F})$

R

A	B	C
0	x	100
2	y	100
4	w	400
5	z	200

S

D	E	F	G
a	100	0	i
b	300	1	j
c	200	5	k

Join

Outerjoin definition

- Let `attr(R)` denotes the list of attributes in the schema of `R`
  - Example: `attr(Sells) = rname, pizza, price`
- Let `null(R)` denotes an *n*-component tuple of `null` values
  - *n* is the *arity* of relation `R`
  - Example: `null(Sells) = (null, null, null)`
- Let $R \rhd_c S$ be $R - \pi_{\mathrm{attr}(R)}(R \bowtie_c S)$
  - Example: let c be $(\mathrm{R.A} = \mathrm{S.F})$

R

A	B	C
0	x	100
2	y	100
4	w	400
5	z	200

S

D	E	F	G
a	100	0	i
b	300	1	j
c	200	5	k

$\pi_{\mathrm{attr(R)}}(\mathbf{R} \bowtie_c S)$

A	B	C
0	x	100
5	z	200

Join

Outerjoin definition

- Let `attr(R)` denotes the list of attributes in the schema of `R`
  - Example: `attr(Sells) = rname, pizza, price`
- Let `null(R)` denotes an *n*-component tuple of `null` values
  - *n* is the <u>*arity*</u> of relation `R`
  - Example: `null(Sells) = (null, null, null)`
- Let $R \rhd_c S$ be $R - \pi_{\text{attr}(R)}(R \bowtie_c S)$
  - Example: let c be $(\mathrm{R.A} = \mathrm{S.F})$

R

A	B	C
0	x	100
2	y	100
4	w	400
5	z	200

S

D	E	F	G
a	100	0	i
b	300	1	j
c	200	5	k

$\pi_{\text{attr(R)}}(\mathrm{R} \bowtie_c S)$

A	B	C
0	x	100
5	z	200

$\mathrm{R} - \pi_{\text{attr(R)}}(\mathrm{R} \bowtie_c \mathrm{S})$

A	B	C
2	y	100
4	w	400

Join

Outerjoin definition

- Let `attr(R)` denotes the list of attributes in the schema of `R`
  - Example: `attr(Sells) = rname, pizza, price`
- Let `null(R)` denotes an *n*-component tuple of `null` values
  - *n* is the *arity* of relation `R`
  - Example: `null(Sells) = (null, null, null)`
- Let $R \rhd_c S$ be $R - \pi_{\text{attr}(R)}(R \bowtie_c S)$
  - Example: let c be $(\mathrm{R.A} = \mathrm{S.F})$
  - In other words: *get all the dangling tuple*

R

A	B	C
0	x	100
2	y	100
4	w	400
5	z	200

S

D	E	F	G
a	100	0	i
b	300	1	j
c	200	5	k

$\pi_{\text{attr}(\mathrm{R})}(\mathrm{R} \bowtie_c S)$

A	B	C
0	x	100
5	z	200

$\mathrm{R} - \pi_{\text{attr}(\mathrm{R})}(\mathrm{R} \bowtie_c \mathrm{S})$

A	B	C
2	y	100
4	w	400

Join

Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv (R \bowtie_c S) \cup \left((R \rhd_c S) \times \{\text{null}(S)\}\right)$

Join

Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv (R \bowtie_c S) \cup \left(\underbrace{(R \rhd_c S)}_{\text{dangling tuple}} \times \underbrace{\{\text{null}(S)\}}_{\text{null values}}\right)$

Join

Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv \underbrace{(R \bowtie_c S)}_{\text{inner join}} \cup \left(\underbrace{(R \rhd_c S)}_{\text{dangling tuple}} \times \underbrace{\{\text{null}(S)\}}_{\text{null values}}\right)$

Join

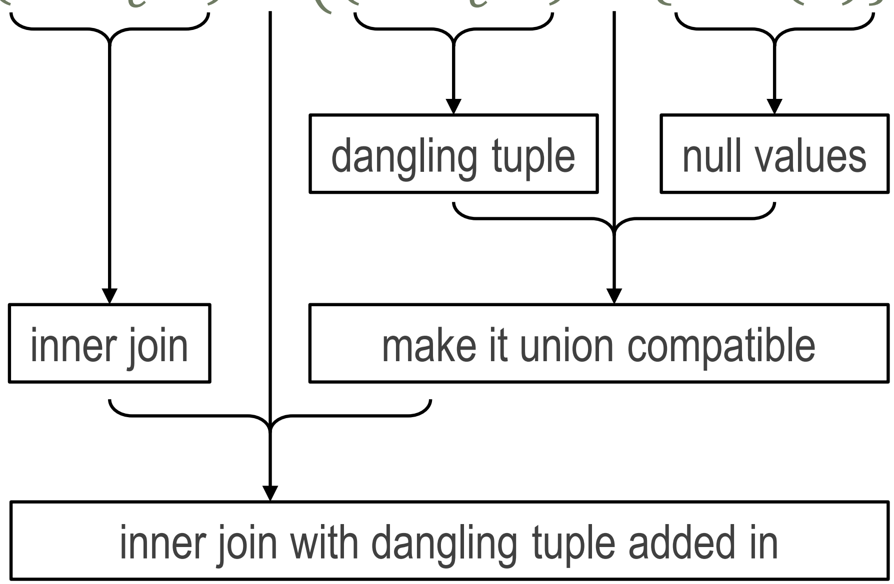
Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv \underbrace{(R \bowtie_c S)}_{\text{inner join}} \cup \underbrace{\left(\underbrace{(R \rhd_c S)}_{\text{dangling tuple}} \times \underbrace{\{\text{null}(S)\}}_{\text{null values}}\right)}_{\text{make it union compatible}}$

Join

Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv (R \bowtie_c S) \cup \left((R \rhd_c S) \times \{\text{null}(S)\}\right)$



Join

Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv (R \bowtie_c S) \cup \left((R \rhd_c S) \times \{\text{null}(S)\}\right)$

Join

Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv (R \bowtie_c S) \cup \left(\underbrace{(R \rhd_c S)} \times \underbrace{\{\text{null}(S)\}}\right)$

$R \rhd_c S$

A	*B*	*C*
2	y	100
4	w	400

$\{\text{null}(S)\}$

D	*E*	*F*	*G*
null	null	null	null

Join

Outerjoin definition

- Left outer join of R and S is defined as
  - $R \rightarrow_c S \equiv (R \bowtie_c S) \cup \left((R \rhd_c S) \times \{\text{null}(S)\}\right)$

$R \rhd_c S$

A	B	C
2	y	100
4	w	400

$\{\textbf{null}(S)\}$

D	E	F	G
null	null	null	null

$(R \bowtie_c S)$

A	B	C	D	E	F	G
0	x	100	a	100	0	i
5	z	200	c	200	5	k

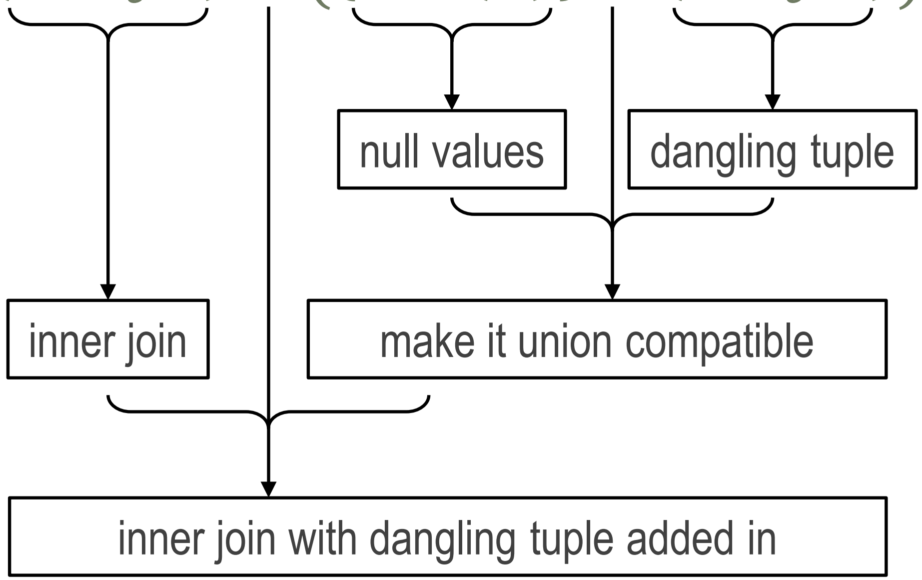
$(R \rhd_c S) \times \{\textbf{null}(S)\}$

A	B	C	D	E	F	G
2	y	100	null	null	null	null
4	w	400	null	null	null	null

Join

Outerjoin definition

- Right outer join of R and S is defined as
  - $R \leftarrow_c S \equiv (R \bowtie_c S) \cup \left(\{\text{null}(R)\} \times (S \rhd_c R)\right)$



Join

Outerjoin definition

- Right outer join of R and S is defined as
  - $R \rightarrow_c S \equiv (R \bowtie_c S) \cup \left(\{\text{null}(R)\} \times (S \rhd_c R)\right)$

$\{\text{null}(R)\}$

A	*B*	*C*
null	null	null

$\mathbf{S} \rhd_c \mathbf{R}$

D	*E*	*F*	*G*
b	300	1	j

$(R \bowtie_c S)$

A	*B*	*C*	*D*	*E*	*F*	*G*
0	x	100	a	100	0	i
5	z	200	c	200	5	k

$\{\text{null}(\mathbf{R})\} \times \mathbf{S} \rhd_c \mathbf{R}$

A	*B*	*C*	*D*	*E*	*F*	*G*
null	null	null	b	300	1	j

Join

Outerjoin definition

- Full outer join of R and S is defined as
  - $R \leftrightarrow_c S \equiv (R \rightarrow_c S) \cup (R \leftarrow_c S)$
  - $R \leftrightarrow_c S \equiv (R \rightarrow_c S) \cup \big((R \rhd_c S) \times \{\text{null}(S)\}\big)$
  - $R \leftrightarrow_c S \equiv \big(\{\text{null}(R)\} \times (S \rhd_c R)\big) \cup (R \leftarrow_c S)$
  - $R \leftrightarrow_c S \equiv (R \bowtie_c S) \cup \big((R \rhd_c S) \times \{\text{null}(S)\}\big) \cup \big(\{\text{null}(R)\} \times (S \rhd_c R)\big)$

Join

Example problems

- Find customers and the pizzas they like; include also customers who don't like any pizza
  - Attempt #1
    - Likes
      - *Does not include customer who don't like pizza*

```
SELECT * FROM Likes;
```

Output

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

Join

Example problems

- Find customers and the pizzas they like; include also customers who don't like any pizza
  - Attempt #2
    - $\pi_{\text{cname,pizza}}\left(\text{Customers} \bowtie_{c} \rho_{(\text{cname2,pizza})}(\text{Likes})\right)$
    - where c is $\text{cname} = \text{cname2}$
      - ➢ *Still does not include customer who don't like pizza*

```
SELECT C.cname, L.pizza
FROM   Customers C
       JOIN
       Likes L
       ON C.cname = L.cname;
```

Output

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

Join

Example problems

- Find customers and the pizzas they like; include also customers who don't like any pizza
  - Attempt #3
    - $\pi_{\text{cname,pizza}}\left(\text{Customers} \rightarrow_c \rho_{(\text{cname2,pizza})}(\text{Likes})\right)$
    - where c is $\text{cname} = \text{cname2}$

```
SELECT C.cname, L.pizza
FROM   Customers C
       LEFT JOIN
       Likes L
       ON C.cname = L.cname;
```

Output

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

Join

Example problems

- Find customers and the pizzas they like; include also pizza that no customer likes
  - Attempt #1
    - $\pi_{\text{cname,pizza}}\left(\text{Likes} \leftarrow_{c} \rho_{(\text{pizza2})}(\text{Pizzas})\right)$
    - where c is $\text{pizza} = \text{pizza2}$
      - ➢ *Why is is the same as* `Likes` *again?*

```
SELECT L.cname, L.pizza
FROM   Likes L
       RIGHT JOIN
       Pizzas P
       ON L.pizza = P.pizza;
```

Output

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

Join

Example problems

- Find customers and the pizzas they like; include also pizza that no customer likes
  - Attempt #2
    - $\pi_{\text{cname},\text{pizza2}}\left(\text{Likes} \leftarrow_{c} \rho_{(\text{pizza2})}(\text{Pizzas})\right)$
    - where c is $\text{pizza} = \text{pizza2}$

```
SELECT L.cname, P.pizza
FROM   Likes L
       RIGHT JOIN
       Pizzas P
       ON L.pizza = P.pizza;
```

Output

cname	*pizza2*
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Siciliana
Ralph	Diavola
null	Marinara

Join

Example problems

- Find customers and the pizzas they like; include also customers who don't like any pizza and pizza that no customer likes
  - Attempt #1

```
SELECT C.cname, P.pizza
FROM   Customers C
       FULL JOIN
       Likes L
       ON C.cname = L.cname
       FULL JOIN
       Pizzas P
       ON L.pizza = P.pizza;
```

Output

cname	*pizza2*
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Siciliana
Ralph	Diavola
Willie	**null**
null	**Marinara**

Join

Example problems

- Find customers and the pizzas they like; include also customers who don't like any pizza and pizza that no customer likes
  - Attempt #2

```
SELECT C.cname, L.pizza
FROM   Customers C LEFT JOIN Likes L
       ON C.cname = L.cname
UNION
SELECT L.cname, P.pizza
FROM   Likes L RIGHT JOIN Pizzas P
       ON L.pizza = P.pizza;
```

Output

cname	*pizza2*
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
Maggie	Funghi
Moe	Funghi
Moe	Siciliana
Ralph	Diavola
Willie	**null**
null	**Marinara**

Join

Natural join: $R \bowtie S$

- Natural join of R and S is defined as
  - $R \bowtie S \equiv \pi_l\big(\sigma_c(R \times S)\big)$
  - where
    - let A be common attributes between R and R
      - let the common attribute be $\{a_1, a_2, \ldots, a_n\}$
    - then c is $(R.a_1 = S.a_1) \wedge (R.a_2 = S.a_2) \ldots \wedge (R.a_n = S.a_n)$
    - and l is the list of attributes in A, followed by the list of attributes in R (excluding those in A) and the list of attributes in S (excluding those in A)
- In other words, it is an inner join such that all the common attributes (i.e., columns) are equal
  - And remove duplicated attributes (i.e., columns)

Join

Natural join: $R \bowtie S$

- Natural join of R and S is defined as
  - $R \bowtie S \equiv \pi_l\left(\sigma_c(R \times S)\right)$
  - where
    - let A be common attributes between R and R
      - let the common attribute be $\{a_1, a_2, \dots, a_n\}$
    - then c is $(R.a_1 = S.a_1) \wedge (R.a_2 = S.a_2) \dots \wedge (R.a_n = S.a_n)$
    - and l is the list of attributes in A, followed by the list of attributes in R (excluding those in A) and the list of attributes in S (excluding those in A)

R

D	B	A
0	x	100
2	y	100
4	w	400
5	z	200

S

E	A	D	C
a	100	0	i
b	300	1	j
c	200	5	k

$R \bowtie_{(R.A=S.A) \wedge (R.D=S.D)} S$

D	B	A	E	A	D	C
0	x	100	a	100	0	i
5	z	200	c	200	5	k

$R \bowtie S$

D	A	B	E	C
0	100	x	a	i
5	200	z	c	k

Join

Natural join: example problem

- For each restaurant, find its name, area, and the pizzas it sells together with their prices

Restaurants

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

```sql
SELECT R.rname, R.area,
       S.pizza, S.price
FROM   Restaurants R,
       Sells S
WHERE  R.rname = S.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

Output

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

Join

Natural join: example problem

- For each restaurant, find its name, area, and the pizzas it sells together with their prices

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

```
SELECT *
FROM   Restaurants
       NATURAL JOIN
       Sells;
```

Restaurants ⋈ Sells

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

Join

Natural join: example problem

- Find distinct names of restaurants that sell some pizza for under $20 that Homer likes

```
SELECT DISTINCT S.rname
FROM   Sells S, Likes L
WHERE  S.price < 20
   AND L.cname = ‘Homer’
   AND S.pizza = L.pizza;
```

Likes

cname	*pizza*
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
:	:

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

Join

Natural join: example problem

- Find distinct names of restaurants that sell some pizza for under $20 that Homer likes

```
SELECT DISTINCT S.rname
FROM   Sells S
       NATURAL JOIN Likes L
WHERE  S.price < 20
  AND  L.cname = 'Homer';
```

Likes

cname	*pizza*
Homer	Hawaiian
Homer	Margherita
Lisa	Funghi
:	:

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

- More conditions
  - Pattern matching
  - Conditional expressions: case analysis
  - Conditional expressions: null analysis
- Multi-relation queries
  - Set operations
  - Cross-product
  - Join: inner join, left outer join, right outer join, full outer join, natural join
- Views

Overview

Views

Introduction

- A view defines a <u>*virtual relation*</u> that can be used for querying

Example

- Consider the following database schema
  - `Courses (`<u>`cid`</u>`, cname, credit, pid, time, quota)`
  - `Profs (`<u>`pid`</u>`, pname, room, contact)`
  - `Enrollments (`<u>`cid`</u>`, ugrad, pgrad, exchange, audit)`
- Course information

```
CREATE VIEW CourseInfo AS
    SELECT C.cname, P.pname, C.time, C.quota,
      E.ugrad+E.pgrad+E.exchange+E.audit AS enrolled
    FROM Courses C, Profs P, Enrollments E
    WHERE C.pid = P.pid
      AND C.cid = E.cid;
```

Views

Introduction

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

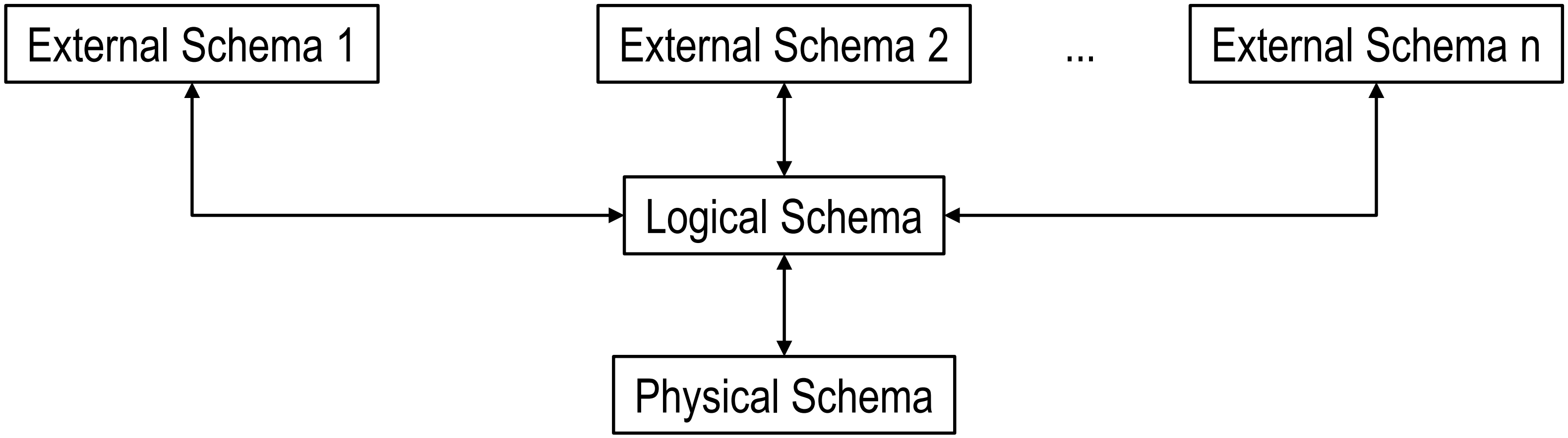
Example

- Consider the following database schema
  - Courses (cid, cname, credit, pid, time, quota)
  - Profs (pid, pname, room, contact)
  - Enrollments (cid, ugrad, pgrad, exchange, audit)
- Course information

```
CREATE VIEW CourseInfo(cname,pname,time,quota,enrolled) AS
    SELECT C.cname, P.pname, C.time, C.quota,
      E.ugrad+E.pgrad+E.exchange+E.audit
    FROM Courses C, Profs P, Enrollments E
    WHERE C.pid = P.pid
      AND C.cid = E.cid;
```

Views

Data independence

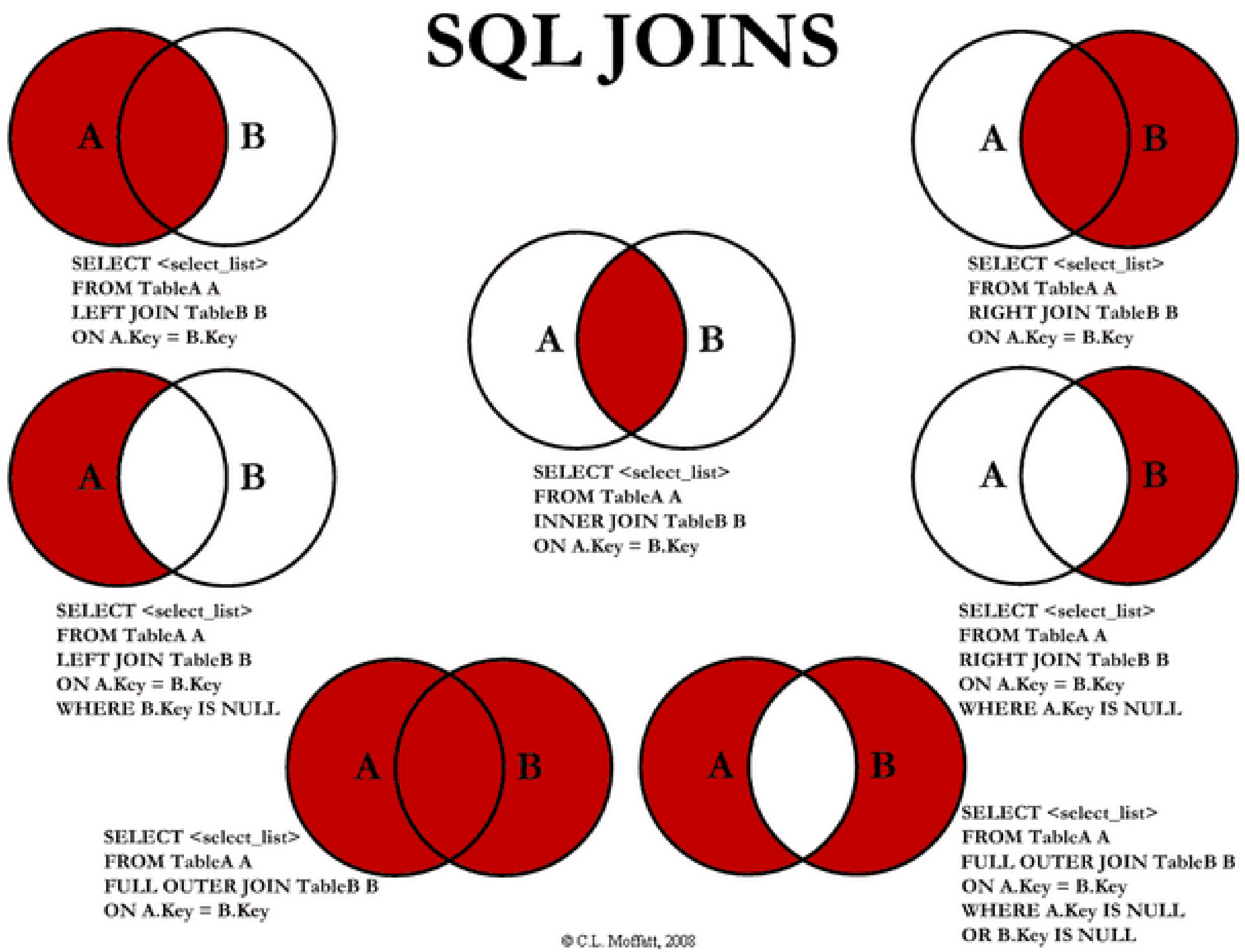


- Physical schema — how the data described by the logical schema is physically organized in DBMS
- Logical schema — logical structure of data in DBMS
- External schema — a customized view of logical schema
- Logical (physical) data independence
  - Insulate users/applications from changes to logical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

Summary

- **Pattern matching** `LIKE`
  - `_` single character
  - `%` 0 or more characters
- **Conditional**
  - `CASE [ `**`expr`**` ] WHEN `**`cond`**` THEN`
    `[ WHEN ... ] ELSE [`**`result`**`] END`
    - Similar to if-else in programming language
  - `NULLIF(`**`value1`**`, `**`value2`**`)`
    - If `value1` equals to `value2` returns `null`, otherwise `value1`
  - `COALESCE(`**`value1`**`, `**`value2`**`, ..., `**`value_n`**`)`
    - Returns the first non-`null` value from left-to-right
- **Set operations**
  - $\cup$ `UNION` `UNION ALL`
  - $\cap$ `INTERSECTION` `INTERSECTION ALL`
  - $-$ `EXCEPT` `EXCEPT ALL`

Summary



(source: The Code Project article by C.L. Moffatt)