PURDUE
UNIVERSITY

# CS34800 Information Systems

*Relational Algebra*

Prof. Chris Clifton

29 August, 2016

Indiana Center for Database Systems ™

1

## Data Types in SQL

- Values in relational model have a data type
  - Implicit in the formal relational model
- Types available vary by manufacturer
  - Number, String, Date, …
  - Constants, variables – familiar concepts
  - Often extensible (similar to a new class in Java)
- For now, just assume whatever is convenient
  - You'll need to use Oracle conventions for projects

2

## Data Type Example: Date and Time

- **date**: Dates, containing a (4 digit) year, month and date
  - Example: **date** '2005-7-27'
- **time**: Time of day, in hours, minutes and seconds.
  - Example: **time** '09:00:30'  **time** '09:00:30.75'
- **timestamp**: date plus time of day
  - Example: **timestamp** '2005-7-27 09:00:30.75'
- **interval**: period of time
  - Example: interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values
- ***Oracle uses a "Date" type that corresponds to timestamp***
  - Very specific (and adjustable, but inflexible) constant format



## Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

    **select** *name*
    **from** *instructor*
    **where** *salary* **is null**

## Null Values and Three Valued Logic

- Three values – *true*, *false*, *unknown*
- Any comparison with *null* returns *unknown*
  - Example*: 5 < null   or   null <> null    or    null = null*
- Three-valued logic using the value *unknown*:
  - OR: (*unknown* **or** *true*)   = *true*,
    (*unknown* **or** *false*)  = *unknown*
    (*unknown* **or** *unknown)* = *unknown*
  - AND: *(true* **and** *unknown)*  = *unknown,*
    *(false* **and** *unknown)* = *false,*
    *(unknown* **and** *unknown)* = *unknown*
  - NOT*:  (***not** *unknown)* = *unknown*
  - "*P*  **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*



## Relational Algebra

*Mathematical view of SQL queries*

- Formal view of what we've seen in SQL
  - Selection (where clause)
  - Projection (select clause)
- Cartesian Product
- Join
- Set operations
- Aggregation

*And how these are done in SQL*

# "Core" Relational Algebra

A small set of operators that allow us to manipulate relations in limited but useful ways. The operators are:

1. Union, intersection, and difference: the usual set operators.
   – But the relation schemas must be the same.
2. *Selection*: Picking certain rows from a relation.
3. *Projection*: Picking certain columns.
4. *Products and joins*: Composing relations in useful ways.
5. *Renaming* of relations and their attributes.

# Relational Algebra

| | | |
|---|---|---|
| σ SELECT | UNARY | FUNDAMENTAL |
| π PROJECT | UNARY | FUNDAMENTAL |
| X CARTESIAN PRODUCT | BINARY | FUNDAMENTAL |
| U UNION | BINARY | FUNDAMENTAL |
| – SET-DIFFERENCE | BINARY | FUNDAMENTAL |
| ∩ SET-INTERSECTION | BINARY | CAN BE DEFINED IN TERMS OF FUNDAMENTAL OPS |
| $\bowtie_\theta$ THETA-JOIN | BINARY | CAN BE DEFINED IN TERMS OF FUNDAMENTAL OPS |
| $\bowtie$ NATURAL JOIN | BINARY | CAN BE DEFINED IN TERMS OF FUNDAMENTAL OPS |
| ÷ DIVISION or QUOTIENT | BINARY | CAN BE DEFINED IN TERMS OF FUNDAMENTAL OPS |

- Properties:
  – Limited expressive power (subset of possible queries), but rich enough to be useful
  – Closed (input is relation, output is relation)
  – Finite (result always defined)
  – Easy to automatically determine how to efficiently process

# Select Operation

- Notation: $\sigma_p(r)$
- *p* is called the **selection predicate**
  - *Corresponds to SQL where clause*
- Defined as:

$$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where *p* is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq. <. \leq$

- Example of selection:

$$\sigma_{dept\_name=\text{“}Physics\text{”}}(instructor)$$



# Project Operation

- Notation: $\prod_{A_1, A_2, \ldots, A_k}(r)$

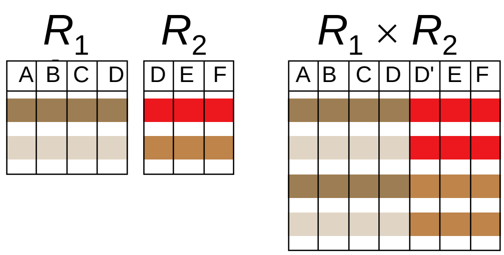
where $A_1$, $A_2$ are attribute names and *r* is a relation name.
  - *Corresponds to SQL Select distinct clause*
- The result is defined as the relation of *k* columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\prod_{ID,\ name,\ salary}(instructor)$$

# Cartesian Product

$R = R_1 \times R_2$

pairs each tuple $t_1$ of $R_1$ with each tuple $t_2$ of $R_2$ and puts in $R$ a tuple $t_1 t_2$.




# Cartesian Product Operation

- Notation *r* x *s*
- Defined as:

  $r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
  - If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used (later)
- SQL: This is the "from" clause
  - select *
    from r, s

# The from Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  **select** ∗
  **from** *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).



# Cartesian Product

*instructor*

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |

*teaches*

| *ID* | *course_id* | *sec_id* | *semester* | *year* |
|---|---|---|---|---|
| 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | FIN-201 | 1 | Spring | 2010 |
| 15151 | MU-199 | 1 | Spring | 2010 |
| 22222 | PHY-101 | 1 | Fall | 2009 |

| *Inst.ID* | *name* | *dept_name* | *salary* | *teaches.ID* | *course_id* | *sec_id* | *semester* | *year* |
|---|---|---|---|---|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 10101 | CS-347 | 1 | Fall | 2009 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 12121 | FIN-201 | 1 | Spring | 2010 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 15151 | MU-199 | 1 | Spring | 2010 |
| 10101 | Srinivasan | Comp. Sci. | 65000 | 22222 | PHY-101 | 1 | Fall | 2009 |
| … | … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … | … |
| 12121 | Wu | Finance | 90000 | 10101 | CS-101 | 1 | Fall | 2009 |
| 12121 | Wu | Finance | 90000 | 10101 | CS-315 | 1 | Spring | 2010 |
| 12121 | Wu | Pinance | 90000 | 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | Wu | Pinance | 90000 | 12121 | FIN-201 | 1 | Spring | 2010 |
| 12121 | Wu | Finance | 90000 | 15151 | MU-199 | 1 | Spring | 2010 |
| 12121 | Wu | Pinance | 90000 | 22222 | PHY-101 | 1 | Fall | 2009 |
| … | … | … | … | … | … | … | … | … |
| … | … | … | … | … | … | … | … | … |



## Examples

- Find the names of all instructors who have taught some course and the course_id
  - **select** *name, course_id*
    **from** *instructor , teaches*
    **where** *instructor.ID = teaches.ID*

- Find the names of all instructors in the Art department who have taught some course and the course_id
  - **select** *name, course_id*
    **from** *instructor , teaches*
    **where** *instructor.ID = teaches.ID* ***and*** *instructor. dept_name* = 'Art'



PURDUE UNIVERSITY

# CS34800 Information Systems

*Relational Algebra*

Prof. Chris Clifton

31 August, 2016

Indiana Center for Database Systems ™

22

# Theta-Join

- $R = R_1 \bowtie_C R_2$

*instructor*

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |

*teaches*

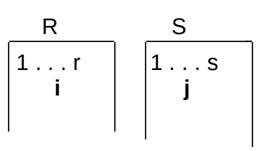
| *ID* | *course_id* | *sec_id* | *semester* | *year* |
|---|---|---|---|---|
| 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | FIN-201 | 1 | Spring | 2010 |
| 15151 | MU-199 | 1 | Spring | 2010 |
| 22222 | PHY-101 | 1 | Fall | 2009 |

- What would be an interesting C?
  - *instructor*.ID = *teaches*.ID
- We said this isn't fundamental, why not?
  - Equivalent to $R = \sigma_C(R_1 \times R_2)$.

---

Theta-Join $R \bowtie_{i\,\theta\,j} S$

$\sigma_{\$i\ \theta\ \$(r+j)}\ (R \times S)$




arity(R) = r

arity(S) = s

arity $(R \bowtie_\theta S) = r + s$

$0 \le card(R \bowtie_\theta S) \le card(R) \times card(S)$

$\theta$ can be $< > = \neq \le \ge$

If equal (=), then it is an EQUIJOIN

$R \bowtie_c S = \sigma_c (R \times S)$

$R(A\ B\ C) \bowtie_{R.A<S.D} S(C\ D\ E)$

result has schema T(A B C C' D E)

| R(ABC) | S(CDE) | T(ABCC'DE) |
|---|---|---|
| 1 3 5 | 2 1 1 | 1 3 5 1 2 2 |
| 2 4 6 | 1 2 2 | 1 3 5 3 3 4 |
| 3 5 7 | 3 3 4 | 1 3 5 4 4 3 |
| 4 6 8 | 4 4 3 | 2 4 6 3 3 4 |
| | | 2 4 6 4 4 3 |
| | | 3 5 7 4 4 3 |

# Natural Join

$R = R_1 \bowtie R_2$

calls for the theta-join of $R_1$ and $R_2$ with the condition that all attributes of the same name be equated. Then, one column for each pair of equated attributes is projected out.

## Example

*instructor*

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| [illegible] | [illegible] | [illegible] | [illegible] |

*teaches*

| *ID* | *course_id* | *sec_id* | *semester* | *year* |
|---|---|---|---|---|
| 10101 | CS-101 | 1 | Fall | 2009 |
| 10101 | CS-315 | 1 | Spring | 2010 |
| 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | FIN-201 | 1 | Spring | 2010 |
| 15151 | MU-199 | 1 | Spring | 2010 |
| 22222 | PHY-101 | 1 | Fall | 2009 |

# Joined Relations

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause

## Join operations – Example

- Relation *course*

| *course_id* | *title* | *dept_name* | *credits* |
|---|---|---|---|
| BIO-301 | Genetics | Biology | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |

- Relation *prereq*

| *course_id* | *prereq_id* |
|---|---|
| BIO-301 | BIO-101 |
| CS-190 | CS-101 |
| CS-347 | CS-101 |

- Observe that

  prereq information is missing for CS-315 and

  course information is missing for CS-437



PURDUE UNIVERSITY

# CS34800 Information Systems

*Relational Algebra*

Prof. Chris Clifton

2 September 2016

Indiana Center for Database Systems ™

## Announcements

- Assignment 1 due today at 11:59pm
- Project 1 (SQL queries) will be released later today – watch web site and email
  - Due September 16
- *No class on Monday* – Labor day
  - PSOs will be held as usual next week

30

## Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_X(E)$$

returns the expression $E$ under the name $X$

- If a relational-algebra expression $E$ has arity $n$, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression $E$ under the name $X$, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

- SQL: **select** E.first=A1, E.second=A2 **as** x

## Union Operation

- Notation: $r \cup s$
- Defined as:

  $$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

  $$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \land year=2009}(section)) \cup$$
  $$\Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \land year=2010}(section))$$

- SQL: **select** * **from** r **union select** * **from** s



## Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r$ **and** $t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Note: $r \cap s = r - (r - s)$
- SQL: **select** * **from** r **intersect select** * **from** s

## Set Difference Operation

- Notation *r* – *s*
- Defined as:

  $r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$

- Set differences must be taken between **compatible** relations.
  - *r* and *s* must have the same arity
  - attribute domains of *r* and *s* must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

  $\Pi_{course\_id}(\sigma_{semester=\text{"Fall"} \wedge year=2009}(section)) -$

  $\Pi_{course\_id}(\sigma_{semester=\text{"Spring"} \wedge year=2010}(section))$

- SQL: **select** * **from** r **minus select** * **from** s



## Example

### Indoor

| Title | Date | Time | Room |
|---|---|---|---|
| Bloomberg Day | Aug 30 | 10:00am | LWSN Commons |
| Black Lives Matter Panel Discussion | Sep 13 | 6:30pm | STEW Fowler Hall |
| CERIAS New Student Welcome | Aug 30 | 6:00pm | LWSN 1142 |
| Global Fest | Sep 13 | 11:00am | Morton Center |
| Microsoft Day | Aug 31 | 10:00am | LWSN Commons |

### Outdoor

| Title | Date | Time | Location |
|---|---|---|---|
| Mosey Down Main Street | Sep 3 | 6:00pm | Main Street, Lafayette |
| Purdue Student Board (PSUB) Night | Aug 26 | 8:00pm | PMU Front Lawn |
| B-Involved Fair | Aug 27 | 3:00pm | Memorial Mall Square |

# select * from indoor union select * from outdoor

- A: Table at right
- B: Schema doesn't match, can't take Union
- C: Fails, need to rename
- D: Unsure, let's cover more

| Title | Date | Time | Location |
|---|---|---|---|
| Mosey Down Main Street | Sep 3 | 6:00pm | Main Street, Lafayette |
| Purdue Student Board (PSUB) Night | Aug 26 | 8:00pm | PMU Front Lawn |
| Bloomberg Day | Aug 30 | 10:00am | LWSN Commons |
| Black Lives Matter Panel Discussion | Sep 13 | 6:30pm | STEW Fowler Hall |
| CERIAS New Student Welcome | Aug 30 | 6:00pm | LWSN 1142 |
| Global Fest | Sep 13 | 11:00am | Morton Center |
| Microsoft Day | Aug 31 | 10:00am | LWSN Commons |
| B-Involved Fair | Aug 27 | 3:00pm | Memorial Mall Square |

41

# Example

**Indoor**

| Title | Date | Time | Room |
|---|---|---|---|
| Bloomberg Day | Aug 30 | 10:00am | LWSN Commons |
| Black Lives Matter Panel Discussion | Sep 13 | 6:30pm | STEW Fowler Hall |
| CERIAS New Student Welcome | Aug 30 | 6:00pm | LWSN 1142 |
| Global Fest | Sep 13 | 11:00am | Morton Center |
| Microsoft Day | Aug 31 | 10:00am | LWSN Commons |

**Outdoor**

| Title | Date | Time | Location |
|---|---|---|---|
| Mosey Down Main Street | Sep 3 | 6:00pm | Main Street, Lafayette |
| Purdue Student Board (PSUB) Night | Aug 26 | 8:00pm | PMU Front Lawn |
| B-Involved Fair | Aug 27 | 3:00pm | Memorial Mall Square |

42

## **select * from** inside **intersect select * from** outside

- A: Table at left
- B: null
- C: Fails, need to rename
- D: Table below

| Title | Date | Time | Location |
|---|---|---|---|

| Title | Date | Time | Location |
|---|---|---|---|
| Mosey Down Main Street | Sep 3 | 6:00pm | Main Street, Lafayette |
| Purdue Student Board (PSUB) Night | Aug 26 | 8:00pm | PMU Front Lawn |
| Bloomberg Day | Aug 30 | 10:00am | LWSN Commons |
| Black Lives Matter Panel Discussion | Sep 13 | 6:30pm | STEW Fowler Hall |
| CERIAS New Student Welcome | Aug 30 | 6:00pm | LWSN 1142 |
| Global Fest | Sep 13 | 11:00am | Morton Center |
| Microsoft Day | Aug 31 | 10:00am | LWSN Commons |
| B-Involved Fair | Aug 27 | 3:00pm | Memorial Mall Square |

43

## Combining Operations

Algebra =
1. Basis arguments +
2. Ways of constructing expressions.

For relational algebra:
1. Arguments = variables standing for relations + finite, constant relations.
2. Expressions constructed by applying one of the operators + parentheses.

- Query = expression of relational algebra.

# Operator Precedence

The normal way to group operators is:

1. Unary operators σ, π, and ρ have highest precedence.
2. Next highest are the “multiplicative” operators, $\bowtie$, $\bowtie_C$, and ×.
3. Lowest are the “additive” operators, ∪, ∩, and —.

- But there is no universal agreement, so we always put parentheses *around* the argument of a unary operator, and it is a good idea to group all binary operators with parentheses *enclosing* their arguments.

## Example

Group $R \cup \sigma S \bowtie T$ as $R \cup (\sigma(S) \bowtie T)$.

# Each Expression Needs a Schema

- If ∪, ∩, — applied, schemas are the same, so use this schema.
- Projection: use the attributes listed in the projection.
- Selection: no change in schema.
- Product $R \times S$: use attributes of $R$ and $S$.
  - But if they share an attribute $A$, prefix it with the relation name, as $R.A$, $S.A$.
- Theta-join: same as product.
- Natural join: use attributes from each relation; common attributes are merged anyway.
- Renaming: whatever it says.

## Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let $E_1$ and $E_2$ be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, $P$ is a predicate on attributes in $E_1$
  - $\prod_S(E_1)$, $S$ is a list consisting of some of the attributes in $E_1$
  - $\rho_x (E_1)$, x is the new name for the result of $E_1$



## Bag Semantics

A relation (in SQL, at least) is really a *bag* or *multiset*.

- It may contain the same tuple more than once, although there is no specified order (unlike a list).
- Example: {1,2,1,3} is a bag and not a set.
- Select, project, and join work for bags as well as sets.
  - Just work on a tuple-by-tuple basis, and don't eliminate duplicates.
- *SQL by default uses bag semantics!*
  - *Except Union, Intersection, Difference*

## Bag Union

Sum the times an element appears in the two bags.

- Example: $\{1,2,1\} \cup \{1,2,3,3\} = \{1,1,1,2,2,3,3\}$.

## Bag Intersection

Take the minimum of the number of occurrences in each bag.

- Example: $\{1,2,1\} \cap \{1,2,3,3\} = \{1,2\}$.

## Bag Difference

Proper-subtract the number of occurrences in the two bags.

- Example: $\{1,2,1\} - \{1,2,3,3\} = \{1\}$.

## Laws for Bags Differ From Laws for Sets

- Some familiar laws continue to hold for bags.
  - Examples: union and intersection are still commutative and associative.
- But other laws that hold for sets do *not* hold for bags.

## Example

$R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T)$ holds for sets.

- Let $R$, $S$, and $T$ each be the bag $\{1\}$.
- Left side: $S \cup T = \{1,1\}$; $R \cap (S \cup T) = \{1\}$.
- Right side: $R \cap S = R \cap T = \{1\}$;
  $(R \cap S) \cup (R \cap T) = \{1\} \cup \{1\} = \{1,1\} \neq \{1\}$.

## Extended ("Nonclassical") Relational Algebra

Adds features needed for SQL, bags.

1. Duplicate-elimination operator $\delta$.
2. Extended projection.
3. Sorting operator $\tau$.
4. Grouping-and-aggregation operator $\gamma$.
5. Outerjoin operator $\overset{\circ}{\bowtie}$ .

## Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- **Multiset** versions of some of the relational algebra operators – given multiset relations $r_1$:
  1. $\boldsymbol{\sigma_\theta (r_1)}$**:** If there are $c_1$ copies of tuple $t_1$ in $r_1$, and $t_1$ satisfies selections $\sigma_{\theta}$, then there are $c_1$ copies of $t_1$ in $\sigma_\theta (r_1)$.
  2. $\boldsymbol{\Pi_A (r)}$**:** For each copy of tuple $t_1$ in $r_1$, there is a copy of tuple $\Pi_A (t_1)$ in $\Pi_A (r_1)$ where $\Pi_A (t_1)$ denotes the projection of the single tuple $t_1$.

## Duplicate Elimination

$\delta(R)$ = relation with one copy of each tuple that appears one or more times in $R$.

### Example

$R$ =

| $A$ | $B$ |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 1 | 2 |

$\delta(R)$ =

| $A$ | $B$ |
|---|---|
| 1 | 2 |
| 3 | 4 |

## Duplicates (Cont.)

- Example: Suppose multiset relations $r_1$ ($A$, $B$) and $r_2$ ($C$) are as follows:

  $r_1$ = {(1, $a$) (2,$a$)}    $r_2$ = {(2), (3), (3)}

- Then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1)$ x $r_2$ would be

  {($a$,2), ($a$,2), ($a$,3), ($a$,3), ($a$,3), ($a$,3)}

- SQL duplicate semantics:

  **select** $A_1, A_2, \ldots, A_n$
  **from** $r_1, r_2, \ldots, r_m$
  **where** $P$

  is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$