Relational Data Model
Outline

1. Relational Data Model
2. From ER Diagrams to Relational Schema
3. Relational Operations
Relational Data Model

• Key concept:

  In ER both *Entity sets* and *Relationships* become relations (tables in RDBMS)

• Database schema is the logical structure of the database.
• Database instance is a snapshot of the data in the database at a given instant in time.
Keys

• Let $K \subseteq R$, $K$ is a **superkey** of $R$ if values for $K$ are sufficient to identify a unique tuple of each possible relation $r(R)$
  • Example: \{ID\} and \{ID,name\} are both superkeys of instructor.

• Superkey $K$ is a **candidate key** if $K$ is minimal
  • Example: \{ID\} is a candidate key for Instructor

• One of the candidate keys is selected to be the **primary key**.
  • Which one?

• **Foreign key** constraint: Value in one relation must appear in another
  • Referencing relation
  • Referenced relation
  • Example: $dept\_name$ in instructor is a foreign key from instructor referencing department
Hacettepe University Computer Engineering Department

Schema Diagram for University Database

- **section**
  - course_id
  - sec_id
  - semester
  - year
  - building
  - room_number
  - time_slot_id

- **time_slot**
  - time_slot_id
  - day
  - start_time
  - end_time

- **classroom**
  - building
  - room_number
  - capacity

- **teaches**
  - ID
  - course_id
  - sec_id
  - semester
  - year

- **takes**
  - ID
  - course_id
  - sec_id
  - semester
  - year
  - grade

- **student**
  - ID
  - name
  - dept_name
  - tot_cred

- **course**
  - course_id
  - title
  - dept_name
  - credits

- **department**
  - dept_name
  - building
  - budget

- **advisor**
  - s_id
  - i_id

- **instructor**
  - ID
  - name
  - dept_name
  - salary

- **prereq**
  - course_id
  - prereq_id
From ER Diagrams to Database Instance

**Product**
(name: string, prince: double, category: string)

**CREATE TABLE** Product(
  name CHAR(50) PRIMARY KEY,
  price DOUBLE,
  category VARCHAR(30)
)

<table>
<thead>
<tr>
<th>name</th>
<th>price</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo1</td>
<td>99.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo2</td>
<td>19.99</td>
<td>Edible</td>
</tr>
</tbody>
</table>
From ER Diagrams to Database Instance

**Product**
- **name**: string, **price**: double, **category**: string

**Person**
- **firstname**: string, **lastname**: string

**Purchased**
- **name**: string, **firstname**: string, **lastname**: string, **date**: date

---

**CREATE TABLE** Purchased(
  name CHAR(50),
  firstname CHAR(50),
  lastname CHAR(50),
  date DATE,
  PRIMARY KEY (name, firstname, lastname),
  FOREIGN KEY (name)
  REFERENCES Product,
  FOREIGN KEY (firstname, lastname)
  REFERENCES Person
)

---

<table>
<thead>
<tr>
<th>name</th>
<th>firstname</th>
<th>lastname</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo1</td>
<td>Bob</td>
<td>Joe</td>
<td>01/01/15</td>
</tr>
<tr>
<td>Gizmo2</td>
<td>Joe</td>
<td>Bob</td>
<td>01/03/15</td>
</tr>
<tr>
<td>Gizmo1</td>
<td>JoeBob</td>
<td>Smith</td>
<td>01/05/15</td>
</tr>
</tbody>
</table>
Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.

- A database which conforms to an ER diagram can be represented by a collection of schemas.

- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.

- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

- Specification of domain (data types) for each column is optional but will be required in the data definition.
Representing Entity Sets

• A strong entity set reduces to a schema with the same attributes
  \(\text{student}(ID, \text{name, tot_cred})\)

• A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set
  \(\text{section (course_id, sec_id, sem, year)}\)

• Example
## Representation of Entity Sets with Composite Attributes

- Composite attributes are flattened out by creating a separate attribute for each component attribute.
  - Example: given entity set `instructor` with composite attribute `name` with component attributes `first_name` and `last_name` the schema corresponding to the entity set has two attributes `name_first_name` and `name_last_name`.
  - Prefix omitted if there is no ambiguity (`name_first_name` could be `first_name`).

- Ignoring multivalued attributes, extended instructor schema is:
  - `instructor(ID, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth, phone_number) age()`)
A multivalued attribute $M$ of an entity $E$ is represented by a separate schema $EM$

Schema $EM$ has attributes corresponding to the primary key of $E$ and an attribute corresponding to multivalued attribute $M$

Example: Multivalued attribute $phone\_number$ of $instructor$ is represented by a schema:

\[ inst\_phone = (\text{ID}, \text{phone\_number}) \]

Each value of the multivalued attribute maps to a separate tuple of the relation on schema $EM$

- For example, an $instructor$ entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
  
  (22222, 456-7890) and (22222, 123-4567)
Representing Relationship Sets

- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.

- Example: schema for relationship set advisor

$$advisor = (s\_id, i\_id)$$

```
<table>
<thead>
<tr>
<th>instructor</th>
<th></th>
<th>student</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>name</td>
<td>ID</td>
</tr>
<tr>
<td>name</td>
<td>salary</td>
<td>name</td>
</tr>
<tr>
<td>salary</td>
<td></td>
<td>tot_cred</td>
</tr>
</tbody>
</table>
```
Redundancy of Schemas

- Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side.

- Example: Instead of creating a schema for relationship set `inst_dept`, add an attribute `dept_name` to the schema arising from entity set `instructor`.

- Example
Redundancy of Schemas (Cont.)

- For one-to-one relationship sets, either side can be chosen to act as the “many” side
  - That is, an extra attribute can be added to either of the tables corresponding to the two entity sets

- If participation is *partial* on the “many” side, replacing a schema by an extra attribute in the schema corresponding to the “many” side could result in null values
Redundancy of Schemas (Cont.)

- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.
- Example: The `section` schema already contains the attributes that would appear in the `sec_course` schema.
Specialization and Generalization

• **Top-down design process**; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
  - These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
  - Depicted by a *triangle* component labeled ISA (e.g., instructor “is a” person).
  - **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

• **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
  - Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
  - The terms specialization and generalization are used interchangeably.
Specialization Example

• **Overlapping** – employee and student
• **Disjoint** – instructor and secretary
• Total and partial
Method 1:
- Form a schema for the higher-level entity
- Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

<table>
<thead>
<tr>
<th>schema</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>ID, name, street, city</td>
</tr>
<tr>
<td>student</td>
<td>ID, tot_cred</td>
</tr>
<tr>
<td>employee</td>
<td>ID, salary</td>
</tr>
</tbody>
</table>

• Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema.
Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

<table>
<thead>
<tr>
<th>schema</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>ID, name, street, city</td>
</tr>
<tr>
<td>student</td>
<td>ID, name, street, city, tot_cred</td>
</tr>
<tr>
<td>employee</td>
<td>ID, name, street, city, salary</td>
</tr>
</tbody>
</table>

- Drawback: *name*, *street* and *city* may be stored redundantly for people who are both students and employees
Aggregation

- Consider the ternary relationship `proj_guide`, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project
Aggregation (Cont.)

• Relationship sets eval_for and proj_guide represent overlapping information
  • Every eval_for relationship corresponds to a proj_guide relationship
  • However, some proj_guide relationships may not correspond to any eval_for relationships
    • So we can’t discard the proj_guide relationship

• Eliminate this redundancy via aggregation
  • Treat relationship as an abstract entity
  • Allows relationships between relationships
  • Abstraction of relationship into new entity
Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:

- A student is guided by a particular instructor on a particular project
- A student, instructor, project combination may have an associated evaluation
Reduction to Relational Schemas

- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship,
  - The primary key of the associated entity set
  - Any descriptive attributes

- In our example:
  
  The schema `eval_for` is:
  
  $\text{eval}_\text{for} (s_{ID}, \text{project}_{id}, i_{ID}, \text{evaluation}_{id})$

  The schema `proj_guide` is redundant.
Relational Operations
Relational Operations

• Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
• The special value **null** is a member of every domain. Indicated that the value is “unknown”
  • The null value causes complications in the definition of many operations

• Relational operations take one or two relations as input and produce a new relation as their result.
• Six basic operations and corresponding operators in Relational Algebra
  • select: σ
  • project: Π
  • union: ∪
  • set difference: −
  • Cartesian product: ×
  • rename: ρ
Select Operation

• The select operation selects tuples that satisfy a given predicate.
• Notation: $\sigma_p (r)$, $p$ is called the selection predicate
• Example: select those tuples of the instructor relation where the instructor is in the “Physics” department.
  • Query
    $\sigma_{\text{dept\_name} = \text{"Physics"}} (\text{instructor})$
  • Result

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
</tr>
</tbody>
</table>
Select Operation (Cont.)

• We allow comparisons using $=, \neq, >, \geq, <, \leq$ in the selection predicate.

• We can combine several predicates into a larger predicate by using the connectives:

  $\land$ (and), $\lor$ (or), $\neg$ (not)

• Example: Find the instructors in Physics with a salary greater $90,000$, we write:

  $\sigma_{dept\_name=\text{"Physics"}} \land salary > 90,000$ (instructor)

• The select predicate may include comparisons between two attributes.

  • Example, find all departments whose name is the same as their building name:

  $\sigma_{dept\_name=\text{building}}$ (department)
Project Operation

• A unary operation that returns its argument relation, with certain attributes left out.

• Notation:

\[ \Pi_{A_1, A_2, A_3, \ldots, A_k} (r) \]

where \( A_1, A_2, \ldots, A_k \) are attribute names and \( r \) is a relation name.

• 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
Project Operation Example

• Example: eliminate the dept_name attribute of instructor

• Query:
\[ \Pi_{ID, name, salary} (instructor) \]

• Result:

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>90000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>40000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>95000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>87000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
</tr>
<tr>
<td>58583</td>
<td>Caliﬁeri</td>
<td>62000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>72000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>92000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
</tr>
</tbody>
</table>
Composition of Relational Operations

• The result of a relational operation is relation and therefore of operations can be composed together into a single expression.

• Consider the query -- Find the names of all instructors in the Physics department.

\[ \Pi_{name}(\sigma_{\text{dept\_name} = \text{"physics"}}(\text{instructor})) \]

• Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.
Cartesian-Product Operation

• To combine information from any two relations. Denoted by $X$

  $instructor \times teaches$

• Since the instructor $ID$ appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
  • $instructor.ID$
  • $teaches.ID$
The **instructor** table **teaches** table

<table>
<thead>
<tr>
<th>instructor.ID</th>
<th>name</th>
<th>dept.name</th>
<th>salary</th>
<th>teaches.ID</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
</tbody>
</table>
Join Operation

• The Cartesian-Product
  \[ \text{instructor} \times \text{teaches} \]
  associates every tuple of instructor with every tuple of teaches.

• To get only those tuples of “\text{instructor} \times \text{teaches} “ that pertain to instructors and the courses that they taught, we write:

  \[ \sigma_{\text{instructor}.id = \text{teaches}.id} (\text{instructor} \times \text{teaches}) \]
Join Operation (Cont.)

- The table corresponding to:

\[ \sigma_{\text{instructor.id} = \text{teaches.id}}(\text{instructor } \times \text{teaches}) \]

<table>
<thead>
<tr>
<th>instructor.ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
<th>teaches.ID</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2017</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
<td>32343</td>
<td>HIS-351</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
<td>45565</td>
<td>CS-101</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
<td>45565</td>
<td>CS-319</td>
<td>1</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
<td>76766</td>
<td>BIO-101</td>
<td>1</td>
<td>Summer</td>
<td>2017</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
<td>76766</td>
<td>BIO-301</td>
<td>1</td>
<td>Summer</td>
<td>2018</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td>83821</td>
<td>CS-190</td>
<td>1</td>
<td>Spring</td>
<td>2017</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td>83821</td>
<td>CS-190</td>
<td>2</td>
<td>Spring</td>
<td>2017</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td>83821</td>
<td>CS-319</td>
<td>2</td>
<td>Spring</td>
<td>2018</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
<td>98345</td>
<td>EE-181</td>
<td>1</td>
<td>Spring</td>
<td>2017</td>
</tr>
</tbody>
</table>
Union Operation

• To combine two relations. Notation: $r \cup s$

• 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: 2$^{nd}$ column of $r$ deals with the same type of values as does the 2$^{nd}$ column of $s$)

Example: to find all courses taught in the Fall 2017 semester, or in the Spring 2018 semester, or in both

$$\Pi_{course\_id} (\sigma\ semester=\text{"Fall"} \land year=2017 (section)) \cup \Pi_{course\_id} (\sigma\ semester=\text{"Spring"} \land year=2018 (section))$$
Union Operation (Cont.)

• Result of:

\[ \Pi_{course\_id} \left( \sigma_{semester=\text{"Fall"} \land year=2017} (\text{section}) \right) \cup \Pi_{course\_id} \left( \sigma_{semester=\text{"Spring"} \land year=2018} (\text{section}) \right) \]
Set-Intersection Operation

- To find tuples that are in both the input relations. Notation: \( r \cap s \)
- Assume:
  - \( r, s \) have the *same arity*
  - attributes of \( r \) and \( s \) are compatible
- Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.
  \[
  \Pi_{\text{course}\_id} (\sigma_{\text{semester}=\text{"Fall"}} \land \text{year}=2017 (\text{section})) \cap \Pi_{\text{course}\_id} (\sigma_{\text{semester}=\text{"Spring"}} \land \text{year}=2018 (\text{section}))
  \]
- Result

<table>
<thead>
<tr>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS-101</td>
</tr>
</tbody>
</table>
Set Difference Operation

- To find tuples that are in one relation but are not in another. Notation $r - s$
- Set differences must be taken between \textbf{compatible} relations.
  - $r$ and $s$ must have the \textit{same} arity
  - attribute domains of $r$ and $s$ must be compatible
- Example: to find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

\[
\Pi_{\text{course}_id} (\sigma_{\text{semester} = "Fall" \land \text{year}=2017} (\text{section})) - \Pi_{\text{course}_id} (\sigma_{\text{semester} = "Spring" \land \text{year}=2018} (\text{section}))
\]
The Assignment Operation

• Creates temporary relations.
• Denoted by $\leftarrow$ and works like assignment in a programming language.
• Example: Find all instructor in the “Physics” and Music department.

$$Physics \leftarrow \sigma_{dept\_name= “Physics”}(instructor)$$
$$Music \leftarrow \sigma_{dept\_name= “Music”}(instructor)$$
$$Physics \cup Music$$
The Rename Operation

• The results of relational expressions do not have a name that we can use to refer to them. The rename operator, $\rho$, is provided for that purpose.

• The expression:

$$\rho_x (E)$$

returns the result of expression $E$ under the name $x$.

• Another form of the rename operation:

$$\rho_{x(A1,A2,..An)} (E)$$
Equivalent Queries

• There is more than one way to write a query
• Example: Find information about courses taught by instructors in the Physics department with salary greater than 90,000
• Query 1
  \[ \sigma_{dept\_name=\text{“Physics”}} \land sal > 90,000 \ (\text{instructor}) \]
• Query 2
  \[ \sigma_{dept\_name=\text{“Physics”}}(\sigma_{sal > 90.000} \ (\text{instructor})) \]
• The two queries are not identical; they are, however, equivalent -- they give the same result on any database.
Equivalent Queries

• There is more than one way to write a query in relational algebra.
• Example: Find information about courses taught by instructors in the Physics department
• Query 1
  \[ \sigma_{\text{dept\_name} = "Physics"}(\text{instructor} \bowtie_{\text{instructor.ID} = \text{teaches.ID}} \text{teaches}) \]
• Query 2
  \[ (\sigma_{\text{dept\_name} = "Physics"}(\text{instructor})) \bowtie_{\text{instructor.ID} = \text{teaches,ID}} \text{teaches} \]

• The two queries are not identical; they are, however, equivalent -- they give the same result on any database.
Acknowledgements

The course material used for this lecture is mostly taken and/or adopted from

• From the slides of the textbook Database System Concepts, Seventh Edition by Avi Silberschatz, Henry F. Korth, S. Sudarshan.

• The course materials of the CS145 Introduction to Databases lecture given by Christopher Ré at Stanford University.