Relational model

• A particular way of structuring data (relations)
• Simple
• Mathematically based
  – Expressions (queries) can be analyzed by DBMS
  – Transformed to equivalent expressions automatically (query optimization) Optimizers have limits (⇒ programmer needs to know how queries are evaluated and optimized)

Relation Instance

• Relation is a set of tuples
  – Tuple ordering immaterial
  – No duplicates
  – Cardinality of relation = number of tuples
• All tuples in a relation have the same structure; constructed from the same set of attributes.
  – Attributes named (⇒ ordering immaterial)
  – Value of an attribute drawn from the attribute’s domain
  – Arity = number of attributes

Example

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>11111111</td>
<td>John</td>
<td>123 Miami</td>
<td>freshman</td>
</tr>
<tr>
<td>2345678</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophomore</td>
</tr>
<tr>
<td>4433322</td>
<td>Art</td>
<td>77 Sycamore</td>
<td>senior</td>
</tr>
<tr>
<td>7654321</td>
<td>Pat</td>
<td>88 5th Avenue</td>
<td>sophomore</td>
</tr>
</tbody>
</table>

Relation Schema

• Relation name
• Attribute names and domains (atomic values)
• Integrity constraints
• Default values - missing values
Relational Database

- Finite set of relations
- Each relation consists of a schema and an instance
- Database schema = set of relation schemas (and other things)
- Database instance = set of (corresponding) relation instances

Examples of schemas

- Student (Id: INT, Name: STRING, Address: STRING, Status: STRING)
- Professor (Id: INT, Name: STRING, DeptId: DEPTS)
- Course (DeptId: DEPTS, CrsName: STRING, CrsCode: COURSES)
- Enrolled (CrsCode: COURSES, StudId: INT, Grade: GRADES, Semester: SEMESTERS)
- Department(DeptId: DEPTS, Name: STRING)
- Teaching(ProfId:INTEGER, CrsCode: COURSES, Semester:SEMESTERS)

Integrity Constraints

- Part of schema
- Restriction on state (or sequence of states) of database
- Enforced by DBMS
- Intra-relational - involve only one relation – all IDs are unique
- Inter-relational - involve several relations

Kinds of Integrity Constraints

- Static - limitation on state of database
  - Syntactic (structural) e.g., all values in a column must be unique
  - Semantic (involve meaning of attributes) e.g., cannot register for more than 18 credits.
- Dynamic - limitation on sequence of database states (supported by some DBMSs, but not in current SQL standard) e.g., cannot raise salary by more than 5%
Key Constraint

- A **key** (or **superkey**) is a set of attributes that uniquely identifies a row.
  - e.g., Id in Student, e.g., (StudId, CrsCode, Semester) in Enrolled
- Minimality - Candidate key - No subset of a candidate key is a key.
- Primary key
- Every relation has a key.
- Relation can have several keys.

Foreign key

- a1 is a **foreign key** of R1 referring to a2 in R2 =>
  if v is a value of a1, there is a unique tuple of R2 in which a2 has value v.
  - This is a special case of referential integrity.
  - a2 must be a candidate key of R2.
  - If no row exists in R2 => violation of referential integrity.
  - Not all rows of R2 need to be referenced.
  - Value of a foreign key might not be specified (null).
- Names of a1 and a2 need not be the same.
- R1 and R2 need not be distinct.
  - Example: Employee(Id : INT, MgrId : INT, ...)
  - Employee(MgrId) references Employee(Id).
  - Every manager is also an employee and hence has a unique row in Employee.
- Foreign key might consist of several columns.
  - Example: (CrsCode, Semester) of Enrolled references (CrsCode, Semester) of Teaching.

Referential integrity

- Verification that the data used in a tuple to design the data of another tuple is valid, in particular the tuple must exist.
- Example:
  - Employee(ssn, name, address, job, #dept)
  - Department(#dept, dname, chair – ssn)
  - A tuple from Employee refers to a tuple of Department via the attribute #dept.
  - If we have a tuple with #dept = CS in Employee, CS must be in the relation Department.
  - A tuple from Department refers to a tuple of Employee via the attribute chair – ssn.
  - If we have a tuple with chair – ssn = 7778889999 in Department, 7778889999 must be in the relation Employee.
- Referential constraint implies an order in the creation or destruction of entities.
- Example:
  - One cannot create a department if the tuple corresponding to the chair does not exist.
  - One cannot destruct the chair of a department if the tuple corresponding to the department still exists.

Inclusion Dependency

- Why are they called inclusion dependencies?
- Referential integrity constraint that is not a foreign key constraint.
  - Teaching(CrsCode, Semester) references Enrolled(CrsCode, Semester) (no empty classes).
  - Reverse relationship is a foreign key.
- Target attributes are not a candidate key.
- No simple enforcement mechanism in SQL.
**Semantic Constraints**

- Domain, primary key, and foreign key are examples of structural (syntactic) constraints.
- Semantic constraints express rules of application: e.g., number of registered students (maximum enrollment).

**Relational languages**

- DDL and DML
- 2 classes of languages:
  - Relational Algebra (intermediate language within DBMS and procedural)
  - Predicate language
- Implementation:
  - Relational Algebra => SQL (declarative - non procedural)
  - Predicate language => QBE (Query by example) (See ACCESS)

**Relational Algebra**

- Based on operators and a domain of values.
- Operators map arguments from domain into another domain value.
- Hence, an expression involving operators and arguments produces a value in the domain. We refer to the expression as a query and the value produced as the result of that query.
Relational Algebra

- Domain: set of relations
- Basic operators: select, project, union, set difference, Cartesian product
- Derived operators: set intersection, division, join
- Procedural: Relational expression specifies query by describing an algorithm (the sequence in which operators are applied) for determining the result of an expression.

Select - Example

- Person relation:

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>John</td>
<td>123 Main</td>
<td>freshman</td>
<td>hiking</td>
</tr>
<tr>
<td>5678</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophomore</td>
<td>hiking</td>
</tr>
<tr>
<td>1322</td>
<td>Art</td>
<td>77 Sycamore</td>
<td>senior</td>
<td>hiking</td>
</tr>
<tr>
<td>4321</td>
<td>Pat</td>
<td>88 5th Avenue</td>
<td>sophomore</td>
<td>stamps</td>
</tr>
</tbody>
</table>

\[ \sigma_{Id>3000 \text{ AND } \text{Hobby}=\text{hiking}}(\text{Person}) \]

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>5678</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophomore</td>
<td>hiking</td>
</tr>
</tbody>
</table>

\[ \sigma_{Id>3000 \text{ AND } Id<3999}(\text{Person}) \]

Project Operator

- Produce table containing subset of columns of argument table.
  \[ \Pi_{\text{attributes list}}(\text{relation}) \]
  or
  \[ \Pi(\text{relation}, \text{attribute list}) \]
- The result is a table (no duplicates).
### Project - Example

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>John</td>
<td>123 Main</td>
<td>freshman</td>
<td>stamps</td>
</tr>
<tr>
<td>5678</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophomore</td>
<td>coins</td>
</tr>
<tr>
<td>1322</td>
<td>Art</td>
<td>77 Sycamore</td>
<td>senior</td>
<td>hiking</td>
</tr>
<tr>
<td>4321</td>
<td>Pat</td>
<td>88 5th Avenue</td>
<td>sophomore</td>
<td>stamps</td>
</tr>
</tbody>
</table>

- \( \Pi_{\text{Name}, \text{Hobby}}(\text{Person}) \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>stamps</td>
</tr>
<tr>
<td>Mary</td>
<td>coins</td>
</tr>
<tr>
<td>Art</td>
<td>hiking</td>
</tr>
<tr>
<td>Pat</td>
<td>stamps</td>
</tr>
</tbody>
</table>

\* Set Operators *

- Relation is a set of tuples \( \Rightarrow \) set operations should apply.

- Result of combining two relations with a set operator is a relation \( \Rightarrow \) all its elements must be tuples having same structure.

- Hence, scope of set operations limited to union compatible relations.

\* Union Compatible Relations *

- Two relations are union compatible if
  - Both have same number of columns
  - Names of attributes are the same in both
  - Attributes with the same name in both relations have the same domain

- Union compatible relations can be combined using:
  - union,
  - intersection, and
  - set difference

\* Expressions *

- Person:

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>John</td>
<td>123 Main</td>
<td>freshman</td>
<td>stamps</td>
</tr>
<tr>
<td>5678</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophomore</td>
<td>coins</td>
</tr>
<tr>
<td>1322</td>
<td>Art</td>
<td>77 Sycamore</td>
<td>senior</td>
<td>hiking</td>
</tr>
<tr>
<td>4321</td>
<td>Pat</td>
<td>88 5th Avenue</td>
<td>sophomore</td>
<td>stamps</td>
</tr>
</tbody>
</table>

- \( \Pi_{\text{Id}, \text{Name}}(\sigma_{\text{Hobby}=\text{stamps}} \lor \text{Hobby}=\text{coins}(\text{Person})) \)

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>John</td>
</tr>
<tr>
<td>5678</td>
<td>John</td>
</tr>
<tr>
<td>4321</td>
<td>Pat</td>
</tr>
</tbody>
</table>
Union-compatible - Example

- \textit{Person}(ssn, name, address, hobby)
- \textit{Professor}(id, name, office, phone)
- \textit{Person} and \textit{Professor} are not union compatible.
- \( \Pi_{\text{name}}(\text{Person}) \) and \( \Pi_{\text{name}}(\text{Professor}) \) are union-compatible.
  \( \Pi_{\text{name}}(\text{Person}) - \Pi_{\text{name}}(\text{Professor}) \) makes sense.
  \( \Pi_{\text{name}}(\text{Person}) \cap \Pi_{\text{name}}(\text{Professor}) \) makes sense.
  \( \Pi_{\text{name}}(\text{Person}) \cup \Pi_{\text{name}}(\text{Professor}) \) makes sense.

Cartesian Product

- If \( R \) and \( S \) are two relations, \( R \times S \) is the set of all concatenated tuples \((x, y)\), where \( x \) is a tuple in \( R \) and \( y \) is a tuple in \( S \).
- \( R \) and \( S \) need not be union compatible.
- \( R \times S \) is expensive to compute.
- Optimization.

Example

- \textit{Enrolled}(StudId, CrsCode, Semester, Grade)
- \textit{Teaching}(ProfId, CrsCode, Semester)
- \( \Pi_{\text{StudId,CrsCode}}(\text{Enrolled})[\text{StudId, CrsCode}] \times \Pi_{\text{ProfId,CrsCode}}(\text{Teaching})[\text{ProfId, PCrsCode}] \)

  The obtained relation is of the form:

<table>
<thead>
<tr>
<th>StudId</th>
<th>CrsCode</th>
<th>ProfId</th>
<th>PCrsCode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Renaming

- Attributes of relation must have distinct names. This is not guaranteed with Cartesian product.
- Renaming operator tidies this up.
  To assign new names to the attributes of the new created relation.
**Derived Operation: Join**

- The expression:
  \[ \sigma_{\text{join-condition}}(RS) \]
  where \( \text{join-condition} \) is a conjunction of terms \( A_i \) \( \text{oper} B_i \) in which \( A_i \) is an attribute of \( R \) and \( B_i \) is an attribute of \( S \) and \( \text{oper} \) is one of \( =, <, \geq, \neq, \leq \)
  is referred as a theta-join formula denoted:
  \( R \bowtie_{\text{join-condition}} S \).

**Join and Renaming**

- Problem: \( R \) and \( S \) might have attributes with the same name - in which case the Cartesian product is not defined.
- Solution:
  - Rename attributes prior to forming the product and use new names in \( \text{join-condition} \).
  - Common attribute names are qualified with relation names in the result of the join.

**Theta Join - Example**

**Equijoin Join - Example**

- Join condition is a conjunction of equalities.
- \( \Pi_{\text{Name},\text{CrsCode}}(\text{Student} \bowtie_{\text{Id}=\text{Id}} \sigma_{\text{grade}=A}(\text{Enrolled})) \)

**Student:**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>John</td>
<td>123 Main</td>
<td>freshman</td>
</tr>
<tr>
<td>222</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophomore</td>
</tr>
<tr>
<td>333</td>
<td>Bill</td>
<td>77 Sycamore</td>
<td>senior</td>
</tr>
<tr>
<td>444</td>
<td>Joe</td>
<td>88 5th Avenue</td>
<td>sophomore</td>
</tr>
</tbody>
</table>

**Enrolled:**

<table>
<thead>
<tr>
<th>StudId</th>
<th>CrsCode</th>
<th>Semester</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>cs387</td>
<td>S00</td>
<td>B</td>
</tr>
<tr>
<td>222</td>
<td>cs386</td>
<td>S99</td>
<td>A</td>
</tr>
<tr>
<td>333</td>
<td>cs385</td>
<td>F88</td>
<td>A</td>
</tr>
</tbody>
</table>

**Result:**

<table>
<thead>
<tr>
<th>Name</th>
<th>CrsCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>cs386</td>
</tr>
<tr>
<td>Bill</td>
<td>cs385</td>
</tr>
</tbody>
</table>

**Natural Join**

- Special case of equi-join:
  - join condition equates all and only those attributes with the same name (condition doesn't have to be explicitly stated).
  - duplicate columns eliminated from the result Natural Join.
- Example:
  \[ \text{Enrolled}(\text{StudId}, \text{CrsCode}, \text{Semester}, \text{Grade}) \]
  \[ \text{Teaching}(\text{ProfId}, \text{CrsCode}, \text{Semester}) \]
  \[ \text{Enrolled} \bowtie \text{Teaching} = \]
  \[ \Pi_{\text{StudId},\text{CrsCode},\text{Semester},\text{Grade},\text{ProfId}}(\text{Enrolled} \bowtie_{\text{StudId}=\text{StudId}} \sigma_{\text{CrsCode}=\text{CrsCode}}(\text{Teaching})) \]

- More generally:
  \[ R \bowtie S = \]
  \[ \Pi_{\text{attr}(R) \cup \text{attr}(S)}(R \bowtie_{\text{join-condition}} S) \]
  - In \( \text{attr}(R) \cup \text{attr}(S) \) duplicates are eliminated.
  - \( \text{join-condition} \) is of the form \( A_1 = A_2 \text{ AND } A_2 = A_2 \) where \( \{A_1, ..., A_n\} = \text{attr}(R) \cap \text{attr}(S) \).
**Natural Join - Example**

- List all Id's of students who took at least two different courses.
- \( \Pi_{\text{StudId}}(\sigma_{\text{CrsCode} \neq \text{CrsCode}2}(\text{Enrolled} \bowtie \text{Enrolled}[\text{StudId}, \text{CrsCode2}, \text{Semester2}, \text{Grade2}])) \)

**Division**

- Goal: Produce the tuples in one relation, \( R \), that match all tuples in another relation, \( S \).
  \( R(A_1, \ldots, A_n, B_1, \ldots, B_m) \)
  \( S(B_1, \ldots, B_m) \)
- \( R/S \), with attributes \( A_1, \ldots, A_n \), is the set of all tuples \( <a> \) such that for every tuple \( <b> \) in \( S \), \( <a, b> \) is in \( R \).
  Can be expressed in terms of projection, set difference and product.

**Division - Example**

- List the Ids of students who have passed all courses that were taught in spring 2000.
- Numerator: StudId and CrsCode for every course passed by every student.
  \( \Pi_{\text{StudId}, \text{CrsCode}}(\sigma_{\text{grade} \neq \text{F}}(\text{Enrolled})) \)
- Denominator: CrsCode of all courses taught in spring 2000.
  \( \Pi_{\text{CrsCode}}(\sigma_{\text{Semester} = \text{Spring 2000}}(\text{Teaching})) \).
- Result is numerator/denominator.