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

Id	**Name**	**Address**	**Status**
1111111	John	123 Mian	freshman
2345678	Mary	456 Cedar	sophomore
4433322	Art	77 Sycamore	senior
7654321	Pat	88 5th Avenue	sophomore

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.

Foreigh 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:

  $Employee(\underline{ssn}, name, address, job, \#dept)$

  $Department(\underline{\#dept}, dname, chair - ssn)$

  - 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*, 777888999 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

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 Operator

- Produces a table containing subset of rows of argument table satisfying condition.

  $\sigma_{<condition>}(< relation >)$

  or $select(relation, < condition >)$

  or $\sigma(< relation >, < condition >)$
- The result is a table.
- Selection Condition:
  - Operators: $<, (, (, >, =, ($
  - Simple selection condition:
  - $< attribute >$ operator $< constant >$
  - $< attribute >$ operator $< attribute >$
  - $< condition >$ AND $< condition >$
  - $< condition >$ OR $< condition >$
  - NOT $< condition >$

Select - Example

- Person relation:

Id	**Name**	**Address**	**Status**	**Hobby**
1111	John	123 Mian	freshman	hiking
5678	Mary	456 Cedar	sophomore	hiking
1322	Art	77 Sycamore	senior	hiking
4321	Pat	88 5th Avenue	sophomore	stamps

- $\sigma_{Id>3000\ AND\ Hobby='hiking'}(Person)$

Id	**Name**	**Address**	**Status**	**Hobby**
5678	Mary	456 Cedar	sophomore	hiking

- $\sigma_{Id>3000\ AND\ Id<3999}(Person)$
- $\sigma_{NOT(Hobby='hiking')}(Person)$
- $\sigma_{Hobby='hiking'}(Person))$

Project Operator

- Produce table containing subset of columns of argument table.

  $\Pi_{<attributes\_list>}(< relation >)$

  or

  $\Pi(< relation >, < attribute\_lists >)$
- The result is a table (no duplicates).

Project - Example

-

Id	Name	Address	Status	Hobby
1111	John	123 Mian	freshman	stamps
5678	Mary	456 Cedar	sophomore	coins
1322	Art	77 Sycamore	senior	hiking
4321	Pat	88 5th Avenue	sophomore	stamps

- $\Pi_{name,hobby}(Person)$
-

Name	Hobby
John	stamps
Mary	coins
Art	hiking
Pat	stamps

Expressions

- Person:

Id	Name	Address	Status	Hobby
1111	John	123 Mian	freshman	stamps
5678	Mary	456 Cedar	sophomore	coins
1322	Art	77 Sycamore	senior	hiking
4321	Pat	88 5th Avenue	sophomore	stamps

- $\Pi_{Id,Name}(\sigma_{hobby='stamps'\ OR\ Hobby='coins'}(Person))$
-

Id	Name
1111	John
5678	Mary
4321	Pat

Set Operators

- Relation is a set of tuples => set operations should apply.
- Result of combining two relations with a set operator is a relation => 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

Union-compatible - Example

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

Cartesian Product

- If R and S are two relations, RXS 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.
- RXS is expensive to compute.
- Optimization.

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.

Example

- $Enrolled(StudId, CrsCode, Semester, Grade)$
- $Teaching(ProfId, CrsCode, Semester)$
- $\Pi_{StudId,CrsCode}(Enrolled)[StudId, SCrsCode]X$
  $\Pi_{ProfId,CrsCode}(Teaching)[ProfId, PCrsCode]$

The obtained relation is of the form:

StudId	SCrsCode	ProfId	PCrsCode

Derived Operation: Join

- The expression:

  $\sigma_{join-condition}(RXS)$

  where $join - condition$ is a conjunction of terms A_i $oper$ B_i in which A_i is an attribute of R and B_i is an attribute of S and $oper$ is one of $=, <, >, \geq, \neq, \leq$ is referred as a theta-jointure denoted:

  $R \infty_{join-condition} S$.

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 $join - condition$.

  – 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.

- $\Pi_{Name,CrsCode}(Student \infty_{Id=StudId}\ \sigma_{grade='A'}(Enrolled))$

- Student:

Id	Name	Address	Status
111	John	123 Mian	freshman
222	Mary	456 Cedar	sophomore
333	Bill	77 Sycamore	senior
444	Joe	88 5th Avenue	sophomore

- Enrolled:

StudId	CrsCode	Semester	Grade
111	cs387	S00	B
222	cs386	S99	A
333	cs385	F88	A

- Result:

Mary	cs386
Bill	cs385

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:

  $Enrolled(StudId, CrsCode, Semester, Grade)$

  $Teaching(ProfId, CrsCode, Semester)$

  $Enrolled \infty Teaching\ =$

  $\Pi_{StudId,CrsCode,Semester,Grade,ProfId}($

  $Enrolled \infty_{CrsCode=CrsCode,Semester=Semester}\ Teaching)$

- More generally:

  $R \infty S\ =$

  $\Pi_{attr(R)\cup attr(S)}(R \infty_{join-condition} S)$

  – In $attr(R) \cup attr(S)$ duplicates are eliminated.

  – $join - condition$ is of the form $A_1 = A_1\ AND\ A_2 = A_2 ...$ where $\{A_1, ..., A_n\} = attr(R) \cap attr(S)$.

Natural Join - Example

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

Division

- Goal: Produce the tuples in one relation, R, that match all tuples in another relation, S.

  $R(A1, ...An, B1, ...Bm)$

  $S(B1...Bm)$
- R/S, with attributes $A1, ...An$, 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.

  $\Pi_{StudId,CrsCode}(\sigma_{grade \neq 'F'}(Enrolled))$
- Denominator: CrsCode of all courses taught in spring 2000.

  $\Pi_{CrsCode}(\sigma_{semester='S2000'}(Teaching))$.
- Result is numerator/denominator.