A relation \( r \) consists of:
- a heading \((A_1, \ldots, A_m)\), which is a sequence of distinct attributes;
- a body, which is a finite collection of tuples \( t = (a_1, \ldots, a_m) \) of values.

We assume some infinite domain of values (or constants)

By a convenient abuse of notation, a relation is often identified with its body (e.g., \( t \in r \) means that \( t \) is a tuple in the body of \( r \))

We may refer to the \( i \)th value in a tuple \( t \in R \) as \( t_i \) or \( t[A_i] \)
It is convenient and common to view the database as a logical structure
- vocabulary = signature + built-in predicates (e.g., <, >, =), and
- interpretation = instance (more formality later)
- Database queries are viewed as logical formulas \( \phi(x) \) over the database:
  \[ Q(I) = \{ a \mid I \models \phi(a) \} \]
- But there are some significant restrictions:
  - We usually have only relation symbols (no function symbols)
  - We usually consider only finite structures (cf. finite-model theory)
  - Queries should be independent of the domain outside the database (cf. relational calculus, discussed later)

Consider the set \( F = \{ \varphi_i \mid i = 1, 2, \ldots \} \) of sentences where \( \varphi_i \) is the sentence "\( R \) has at least \( i \) distinct tuples"
- Each \( \varphi_i \) is expressible in first-order logic
- Every finite subset of \( F \) has a finite model, but there is no finite model for \( F \)
- Hence, the compactness theorem no longer holds in the finite

### Table of Contents

- Database Model
- Queries
- Constraints

### Definition (Query)
Let \( S \) be a schema. A query \( Q \) (over \( S \)) with the heading \( (A_1, \ldots, A_k) \) is a function that maps every database \( I \in \text{Inst}(S) \) into a relation \( Q(I) \) over \( (A_1, \ldots, A_k) \).

### Example
Find all pairs \((s, c)\) of students and courses, such that there exists a course number \( x \) where both \( \text{Takes}(s, x) \) and \( \text{Course}(x, c) \) hold.

<table>
<thead>
<tr>
<th>Takes</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>cno</td>
</tr>
<tr>
<td>Ahuva</td>
<td>1</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ \Rightarrow \]

<table>
<thead>
<tr>
<th>student</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td>AI</td>
</tr>
<tr>
<td>Ahuva</td>
<td>DB</td>
</tr>
</tbody>
</table>

### Boolean Queries
- A special case is where \( k = 0 \), and then the result either contains the empty tuple or is empty; in this case we say that the query is Boolean
- Example: Is it the case that for some \( x \), both \( \text{Takes}(\text{Ahuva}, x) \) and \( \text{Course}(x, \text{DB}) \) hold?
- We often denote
  - \( Q(I) = \{ () \} \) by \( Q(I) = \text{true or } I = Q \)
  - \( Q(I) = \emptyset \) by \( Q(I) = \text{false or } I \notin Q \)
- Boolean queries are very important in the analysis query languages (expressiveness, complexity, optimization and equivalence, etc.)
Relational Algebra

- Introduced by Codd [Cod72]
- Used by existing database systems, mainly for internal query-plan optimization
- A collection of operations over relations
  - Unary: $r \rightarrow t$; binary: $(r, s) \rightarrow t$
- Queries via:
  1. Applying the operators to the database relations
  2. Composition

Algebraic Operators

- **Union ($\cup$), difference ($-$)**
  - $R \cup S$ and $R - S$ allowed if $R$ and $S$ are union compatible, that is, the have the same heading
- **Cartesian product ($\times$)**
  - $R \times S$ allowed only when $R$ and $S$ have disjoint headings
- **Projection ($\pi$)**
  - $\pi_{A_1, \ldots, A_k}(R)$ allowed if $A_1, \ldots, A_k$ are distinct attributes of $R$
- **Selection ($\sigma$)**
  - $\sigma_C(R)$ allowed if $C$ is a condition over the attributes of $R$ (e.g., $A_1 = A_2$ or $A_1 \neq A_2$)
- **Renaming ($\rho$)**
  - $\rho_{A \rightarrow B}(R)$ allowed if $A$ is an attribute of $R$ and $B$ is not an attribute of $R$
Recall: in FO (First-Order Logic), a structure is a triple \((D, \sigma, I)\) where:
- \(D\) is a domain of values
- \(\sigma\) is a signature (collection of constants, function symbols, and relation symbols)
- \(I\) is an interpretation (that instantiates \(\sigma\) using values from \(D\))

In Relational Calculus (RC):
- The schema signature is the signature (only rel. symbols).
- The database interprets of the relations.
- Constants are interpreted using equality (e.g., “DB” is interpreted as the string “DB”).
- There is no concept of a domain.

Why should we care?

- Very convenient tool to reason about expressiveness in RA and SQL
- If you can phrase it in FO, you can write it in RA
- On the other hand, tools from Logic (or, more precisely, Finite Model Theory) can help establishing negative results

Domain Dependence

- What is the meaning of:
  - \(\{x \mid R(x)\}\)?
  - \(\{x \mid \neg R(x)\}\)?
- In FO it makes sense, since we have a domain
- In databases we wish to forbid queries that depend on the domain
- Next, we formalize this idea

Domain Independence

- Let \(S\) be a schema, \(I\) an instance, and \(Q = \{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}\) an RC query
- The active domain is the set of constants that appear in either \(I\) or \(Q\)
- If \(D\) is a domain that contains the active domain, then:
  - An answer for \(Q\) is an assignment to \(x_1, \ldots, x_n\) that makes \(\varphi(x_1, \ldots, x_n)\) true in the interpretation \(I\)
  - Recall that, now, we have a complete FO structure
  - \(Q^D(I)\) is the set of all answers for \(Q\)
- \(Q\) is domain independent if \(Q^D(I) = Q^E(I)\) for all instances \(I\) and domains \(D\) and \(E\) that contain the active domain

Example: Division

- Let \(R(A, B)\) and \(S(B)\) be two relation schemas
- Recall: \(R \div S\) consists of all \(A\) of \(R\) that occur in \(R\) with every \(B\) of \(S\)
  - Example: \(\text{Take}(A, C) \land \text{Course}(C)\) consists of all the students that take every course
- In RA: \(\pi_A R \land \pi_A (\pi_B R \times S) \setminus R\)
- In RC: \(\{(x) \mid \exists y[R(x, z)] \land \forall y[S(y) \rightarrow R(x, y)]\}\)
We denote by RA the class of queries that can be phrased in relational algebra, where only equality is allowed in selection predicates.

Two queries $Q_1$ and $Q_2$ (possibly in different formalisms) over the same schema $S$ are equivalent, denoted $Q_1 \equiv Q_2$, if for all instances $I$ of $S$ we have $Q_1(I) = Q_2(I)$.

**Theorem** [Cod72]
RA and domain-independent RC have the same expressive power. That is, for all schemas $S$:
- For every RA expression $\alpha$ there is a domain-independent RC query $Q$ such that $\alpha \equiv Q$
- For every domain-independent RC query $Q$ there is an RA expression $\alpha$ such that $\alpha \equiv Q$

Can we test whether a given RC query is domain independent?
Unfortunately, this problem is undecidable [Pao69]
Nevertheless, there is an effective syntax for domain-independent RC queries; that is, a fragment of safe queries where:
- Every safe query is domain independent
- Safety can be detected in polynomial time
- Every domain-independent RC query is equivalent to some safe query

SQL (Structured Query Language) is natural language to express relational algebra
- SELECT (projection) ... AS (rename)
- FROM (Cartesian product)
- WHERE (selection)
- UNION
- MINUS
- And much more, e.g., aggregate operators (e.g., COUNT, SUM), clustering operators (e.g., GROUP BY, HAVING), ranking (e.g., ORDER BY, LIMIT), and more

Conjunctive Queries (CQs) are SELECT-FROM-WHERE queries (no MINUS, UNION, etc.) such that all the WHERE conditions are equalities among attributes
- CQs are typically represented in the following FOL notation:

$$Q(x) \equiv \exists y [\varphi_1(x, y) \land \cdots \land \varphi_k(x, y)]$$

where:
- $x$ and $y$ are disjoint sequences of variables
- Each $\varphi_i(x, y)$ is a an atomic formula of the form $R(t_1, \ldots, t_m)$ where $R$ is an $m$-ary relation in the schema and each $t_i$ is either a variable in $x$, a variable in $y$, or a constant value (e.g., 7 or ‘Ahuva’)
- Every variable in $x$ occurs at least once on the right hand side

Find all pairs $(s, c)$ of students and courses, such that there exists a course number $x$ where both $\text{Takes}(s, x)$ and $\text{Course}(x, c)$ hold.

```
SELECT S.student, C.cname
FROM Takes T, Course C
WHERE T.cno = C.cno
```
**CQ Terminology and Notation**

\[ Q(x) := \exists y[\varphi_1(x, y) \land \ldots \land \varphi_k(x, y)] \]

For simplification, quantification and conjunction are omitted:

\[ Q(x) := \varphi_1(x, y), \ldots, \varphi_k(x, y) \]

A variable in \( x \) is called a **free** or **head** variable, and a variable in \( y \) is called an **existential** variable.

---

**The Example as CQ**

<table>
<thead>
<tr>
<th>Takes</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>cno</td>
</tr>
<tr>
<td>Ahuva</td>
<td>1</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

Find all pairs \((s, c)\) of students and courses, such that there exists a course number \( x \) where both \( \text{Takes}(s, x) \) and \( \text{Course}(x, c) \) hold.

\[ Q(s, c) := \text{Takes}(s, x) \land \text{Course}(x, c) \]

---

**Why Are CQs Interesting?**

- This is the class of all the queries that can be phrased in RA when using only:
  - Selection with equality predicate
  - Projection
  - Join
- For that reason, CQs are often called SPJ queries
- CQs are the building block of expressive query languages, such as Datalog
- Useful queries that are simple enough to perform deep investigation for various database problems
  - Simple: evaluation characterized by **homomorphism** (next)
- As we shall see, there are significant deep insights and algorithms that apply only to conjunctive queries

**Homomorphism**

- Let \( Q \) be a CQ and \( I \) an instance of the same schema as \( Q \)
- Let \( \text{trm}(Q) \) be the set of terms (variables or constants) in \( Q \)
- Let \( \text{dom}(I) \) be the set of values (constants) in \( I \)
- A **homomorphism from** \( Q \) to \( I \) is a function \( h : \text{trm}(Q) \rightarrow \text{dom}(I) \) such that:
  - \( h(c) = c \) for all constants \( c \)
  - For each atom \( R(\tau_1, \ldots, \tau_k) \) of \( Q \), the fact \( R(h(\tau_1), \ldots, h(\tau_k)) \) belongs to \( I \)

**Proposition**

Let \( Q \) and \( I \) be a Boolean CQ and an instance, respectively, over the same schema. Then \( Q(I) = \text{true} \) if and only if there is a homomorphism from \( Q \) to \( I \).
What are the homomorphisms here?

### Takes
- student: Ahuva, student: Alon
- cno: 1, 1

### Course
- cno: 1, 2, 3
- cname: AI, DB, PL

Q₁: \(\text{Takes}(s, x) \land \text{Course}(x, c)\)

Q₂: \(\text{Takes}(\text{Ahuva}, x) \land \text{Course}(x, c)\)

Why Do We Care about Constraints?

- Allow to enforce database coherence and avoid bugs
- Allow to formally determine what inconsistency means
  - Very relevant to us
- May have dramatic effect on algorithms and complexity
- By focusing on specific classes of constraint languages, we allow for nontrivial analysis

### Formal Definition
- Let \(S\) be a schema
- A **Functional Dependency** (FD) over \(S\) is an expression of the form \(R: U \rightarrow V\), where \(U\) and \(V\) are sets of attributes of \(R\)
- An instance \(I\) over \(S\) satisfies the FD \(R: U \rightarrow V\) if for every two tuples \(t₁\) and \(t₂\) of \(R\):
  - \(t₁\) and \(t₂\) agree on \(U \Rightarrow t₁\) and \(t₂\) agree on \(V\)
    - By “agree on \(W\)” we mean that \(t₁\) and \(t₂\) have the same value in every position that corresponds to an attribute of \(W\)
  - If \(U\) and \(V\) cover all the attributes of \(R\), then \(R: U \rightarrow V\) is a **key constraint** and \(U\) is said to be a **key** for \(R\)
  - As a simplified notation, we write \(U\) and \(V\) by simply listing their attributes (no set notation)

### Example Revisited

### Lab
- name: SIPL, LCL, SSDL, STAT
- faculty: EE, CS, IE

### LabRoom
- lab: SIPL, LCL, SSDL
- building: Meyer, Taub
- room: 100, 101, 200

- A lab belongs to just one faculty (i.e., name is a key for Lab)
- A specific room in a specific building belongs to only one lab
- A lab may have multiple rooms, but all in the same building
A lab belongs to just one faculty (i.e., lab is a key for Lab)

\[ \text{Lab} : \text{name} \rightarrow \text{faculty} \]

A specific room in a specific building belongs to only one lab

\[ \text{LabRoom} : \text{building room} \rightarrow \text{lab} \]

A lab may have multiple rooms, but all in the same building

\[ \text{LabRoom} : \text{lab} \rightarrow \text{building} \]
There are various formalisms that naturally extend FDs to cross-relation dependencies.

Following are two popular examples:
- **Equality-Generating Dependencies (EGDs)**
- **Denial Constraints (DCs)**

The FD Lab : name → faculty can be phrased in FOL as:

\[ \forall x, y, z \left[ \text{Lab}(x, y) \land \text{Lab}(x, z) \rightarrow y = z \right] \]

An EGD is an expression of the form:

\[ \forall x \left[ \varphi(x) \rightarrow y_1 = y_2 \right] \]

\( \varphi(x) \) is a conjunction of atomic formulas

\( y_1 \) and \( y_2 \) are variables in \( x \)

Example:

\[ \text{Lab}(l_1, f_1), \text{Lab}(l_2, f_2), \text{LabRoom}(l_1, b, r_1), \text{LabRoom}(l_2, b, r_2) \rightarrow f_1 = f_2 \]

Every lab in LabRoom should be listed in the Lab relation (foreign key)

Let \( S \) be a schema.

An **Inclusion Dependency (IND)** over \( S \) is an expression \( \delta \) of the form

\[ R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m] \]

where:

- \( R \) and \( S \) are relation names in \( S \)
- \( R \) and \( S \) may be equal
- \( A_1, \ldots, A_m \) are distinct attributes of \( R \)
- \( B_1, \ldots, B_m \) are distinct attributes of \( S \)

An instance \( I \) over \( S \) satisfies \( \delta \) if

\[ \pi_{A_1,\ldots,A_m}(I) \subseteq \pi_{B_1,\ldots,B_m}(S) \]

Consider the relation Friend[person\(_1\), person\(_2\)]

Friend[person\(_1\), person\(_2\)] ≤ Friend[person\(_2\), person\(_1\)]

means that friendship is symmetric
Another Example

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>lab</td>
</tr>
<tr>
<td>SIPL</td>
<td>SIPL</td>
</tr>
<tr>
<td>LCL</td>
<td>SIPL</td>
</tr>
<tr>
<td>SSDL</td>
<td>LCL</td>
</tr>
<tr>
<td>STAT</td>
<td>SSDL</td>
</tr>
</tbody>
</table>

- Every lab should be listed in the Lab relation (foreign key)

LabRoom[lab] ⊆ Lab[name]

Tuple-Generating Dependencies

- The IND LabRoom[lab] ⊆ Lab[name] can be phrased in FOL as

$$\forall x, y, z [\text{LabRoom}(x, y, z) \rightarrow \exists w [\text{Lab}(x, w)]]$$

- A Tuple-Generating Dependency (TGD) is an expression of the form

$$\forall x [\psi(x) \rightarrow \exists y \psi(x, y)]$$

where $\psi(x)$ and $\psi(x, y)$ are conjunctions of atomic formulas.

- Example:

Researcher(p, l), LabRoom(l, b, r) → $\exists r' [\text{PersonRoom}(p, b, r')]$

Question

Can we express that Friends is transitive with TGDs?

Can we express that Friends has no triangles with TGDs?

References