Summary of Lectures #1 and #2

- Codd’s Relational Model based on the concept of a relation (table)

- Relational algebra as a database query language
  - Five basic operators ($\cup$, $-$, $\times$, $\pi$, $\sigma$)
  - Relational algebra expressions obtained by combining these operators: can express intersection, join, quotient, ...
  - Independence of the basic operators

- Notion of a relationally complete database query language

- Influence of relational algebra on SQL
  - SELECT ... FROM ... WHERE ... is a combination of $\pi$, $\sigma$, $\times$
  - SQL is relationally complete (have to use aliases for self-joins)
  - Set semantics vs. bag (multiset) semantics
Roadmap for Lectures #3 and #4

- Relational Calculus as a database query language
  - Motivation
  - Syntax
  - Informal semantics, illustrations of its expressive power
  - More formal semantics, making the domain explicit, the issue of domain independence.
- Codd’s Theorem:
  Equivalence of relational algebra and relational calculus
- Algorithmic aspects of domain independence
- SQL Revisited:
  - Influence of relational calculus on SQL
  - Features of SQL beyond relational algebra and relational calculus.
In addition to relational algebra, Codd introduced relational calculus. Relational calculus is a declarative database query language based on first-order logic. Codd’s main technical result is that relational algebra and relational calculus have essentially the same expressive power. More precisely, relational algebra has the same expressive power as domain independent relational calculus. Relational calculus comes into two different flavors:
- Tuple relational calculus (this is the version studied by Codd)
- Domain relational calculus.
Here, we will focus on domain relational calculus (which is the version considered in the AHV textbook). Refer to Codd’s 1972 paper for the precise definition of tuple relational calculus.
Propositional Logic (aka Boolean Logic) Reminder

- **Propositional variables**: $x, y, z, \ldots$
  - They take values 0 (True) and 1 (False).

- **Propositional connectives**: $\land, \lor, \lnot, \rightarrow$

- **Propositional formulas**: expressions built from propositional variables and propositional connectives
  - **Syntax**: $\varphi := x, y, z, \ldots | (\psi \land \chi) | (\psi \lor \chi) | \lnot \psi | (\psi \rightarrow \chi)$
  - **Semantics**: Truth-table semantics

- **Application**: Propositional formulas express Boolean functions
  - $(x \lor y) \land (\lnot x \lor \lnot y)$  XOR-Gate
  - $(x \land y) \lor (x \land z) \lor (y \land z)$  Majority Gate

*In fact, every Boolean function can be expressed by a propositional formula.*
First-Order Logic - Motivation

- **First-Order Logic** is a formalism for expressing properties of mathematical structures (graphs, trees, partial orders, ...).

- **Example:** Consider a graph $G=(V,E)$ (nodes are in $V$, edges are in $E$)
  - There is a self-loop.
  - Every two nodes are connected via a path of length 2.
  - Every node has exactly three distinct neighbors.
  - There is a path of length 3 from node $x$ to node $y$.
  - Node $x$ has at least four distinct neighbors.

These and many other similar properties are expressible as formulas of first-order logic on graphs.

- One of Codd’s key insights was that first-order logic can also be used to express relational database queries.
First-Order Logic

- **Question:** What is First-Order Logic?

- **Answer:** Informally,

  “First-Order Logic = Propositional Logic + (∃ and ∀)”,
  where
  ∃ and ∀ range over *possible values occurring in relations.*
Relational Calculus (First-Order Logic for Databases)

- **First-order variables**: \( x, y, z, \ldots, x_1, \ldots, x_k, \ldots \)
  - They range over values that may occur in tables.
- **Relation symbols**: \( R, S, T, \ldots \) of specified arities (names of relations)
- **Atomic (Basic) Formulas**:
  - \( R(x_1,\ldots,x_k) \), where \( R \) is a \( k \)-ary relation symbol
    (alternatively, \( (x_1,\ldots,x_k) \in R \); the variables need not be distinct)
  - \( (x \text{ op } y) \), where op is one of \( =, \neq, <, >, \leq, \geq \)
  - \( (x \text{ op } c) \), where \( c \) is a constant and op is one of \( =, \neq, <, >, \leq, \geq \).
- **Relational Calculus Formulas**:
  - Every atomic formula is a relational calculus formula.
  - If \( \varphi \) and \( \psi \) are relational calculus formulas, then so are:
    - \( (\varphi \land \psi), (\varphi \lor \psi), \neg \psi, (\varphi \rightarrow \psi) \) (propositional connectives)
    - \( (\exists x \varphi) \) (existential quantification)
    - \( (\forall x \varphi) \) (universal quantification).
Relational Calculus

- **Examples:** Assume $E$ is a binary relation symbol
  - $(\exists x)E(x,x)$
  - $(\forall x)(\forall y)(\exists z)(E(x,z) \land E(z,y))$
  - $(\exists z_1)(\exists z_2)(E(x,z_1) \land E(z_1,z_2) \land E(z_2,y))$
  - $(\exists y)(\exists z)(E(x,y) \land E(x,z) \land (y \neq z))$

- **Free and bound variables:**
  - In the first two formulas above, no variable is free.
  - In the third formula above, the free variables are $x$ and $y$.
  - In the fourth formula above, the only free variable is $x$.
  - Intuitively, a variable is free in a formula if the variable must be assigned a value in order to tell if the formula is true or false.
Free and Bound Variables

- A sentence is a first-order formula $\psi$ with no free variables.
  - $(\exists x)E(x,x)$
  - $(\forall x)(\exists z)(E(x,z) \land E(z,y))$
  - $(\forall x)(\forall y)(x < y \rightarrow (\exists z)(x < z \land z < y))$
  - On every relational database $I$, a sentence is either true or false.
    - Either $I \models \psi$ or $I \models \neg \psi$

- If a first-order formula has at least one free variable, then it makes no sense to tell whether it is true or false on a relational database $I$. Instead, we need to also assign values to its free variables
  - $I, 3, 5 \models \exists z (x < z \land z < y)$
  - $I, 3, 4 \models \neg \exists z (x < z \land z < y)$, where $I$ is the linear order $<$ on the natural numbers $1, 2, 3, ...$
Queries

Definition: Let $S$ be a relational database schema. A $k$-ary query on $S$ is a function $q$ defined on the relational database instances over $S$ such that if $I$ is a relational database instance over $S$, then $q(I)$ is a $k$-ary relation (i.e., a set of $k$-tuples).

Note: All “queries” that we have expressed in relational algebra and/or in SQL thus far are queries in the above formal sense.

- Find the salaries of department chairs (binary query)
- Find the students who are enrolled in every course taught by Victor Vianu (unary query)
- The natural join $R \bowtie S$ of $R(A,B,C)$ and $S(B,C,D)$ (4-ary query)
Queries: Syntax vs. Semantics

- Queries are, by definition, **semantic** objects (functions)

- Database query languages provide a **syntax** for defining (expressing) queries. In particular,
  - Every **relational algebra expression** defines a query
  - Every **SQL expression** defines a query

- One of Codd’s key insight was that **formulas of first-order logic** (aka **formulas of relational calculus**) can also be used to define queries.
Relational Calculus as a Database Query Language

Definition:
- A relational calculus expression is an expression of the form
  \[ \{ (x_1, \ldots, x_k) : \varphi(x_1, \ldots, x_k) \} , \]
  where \( \varphi(x_1, \ldots, x_k) \) is a relational calculus formula with \( x_1, \ldots, x_k \) as its free variables.
- When applied to a relational database \( I \), this relational calculus expression returns the \( k \)-ary relation that consists of all \( k \)-tuples \( (a_1, \ldots, a_k) \) that make the formula “true” on \( I \).
- Thus, every relational calculus expression as above defines a \( k \)-ary query.

Example: The relational calculus expression
\[
\{ (x,y) : (\exists z)(E(x,z) \land E(z,y)) \}
\]
returns the set \( P \) of all pairs of nodes \( (a,b) \) that are connected via a path of length 2.
Example:  FACULTY(name, dpt, salary), CHAIR(dpt, name)
Give a relational calculus expression for C-SALARY(dpt,salary)
(find the salaries of department chairs).

\{ (x,y): \exists u(FACULTY(u,x,y) \land CHAIR(x,u)) \} 

Here is another relational calculus expression for the same task:

\{ (x,y): \exists u \exists v(FACULTY(u,x,y) \land CHAIR(x,v) \land (u=v)) \}
Relational Calculus as a Database Query Language

Example: FACULTY(name, dpt, salary)
Find the names of the highest paid faculty in CS
\{ x: \varphi(x) \}, where \varphi(x) is the formula:

\exists y,z (FACULTY(x,y,z) \land y = "CS" \land
(\forall u,v,w(FACULTY(u,v,w) \land v = "CS" \rightarrow z \geq w)))

Exercise: Express this query in relational algebra and in SQL.

Abbreviation:
- \exists x_1,\ldots,x_k stands for \exists x_1,\ldots,\exists x_k
- \forall x_1,\ldots,x_k stands for \forall x_1,\ldots,\forall x_k
Example: Let $R(A,B,C)$ and $S(B,C,D)$ be two ternary relation schemas.

- Recall that, in relational algebra, the natural join $R \bowtie S$ is given by
  \[
  \pi_{R.A,R.B,R.C,S.D} \left( \sigma_{R.B = S.B \land R.C = S.C} (R \times S) \right).
  \]

- Give a relational calculus expression for $R \bowtie S$
  \[
  \{ (x_1,x_2,x_3,x_4) : R(x_1,x_2,x_3) \land S(x_2,x_3,x_4) \}
  \]

Note: The natural join is expressible by a quantifier-free formula of relational calculus.
Quotient in Relational Calculus

- Recall that the quotient (or division) $R \div S$ of two relations $R$ and $S$ is the relation of arity $r - s$ consisting of all tuples $(a_1, \ldots, a_{r-s})$ such that for every tuple $(b_1, \ldots, b_s)$ in $S$, we have that $(a_1, \ldots, a_{r-s}, b_1, \ldots, b_s)$ is in $R$.

- Assume that $R$ has arity 5 and $S$ has arity 2. Express $R \div S$ in relational calculus (3-ary query)

$$\{ (x_1, x_2, x_3): (\forall x_4)(\forall x_5) (S(x_4, x_5) \rightarrow R(x_1, x_2, x_3, x_4, x_5)) \}$$

- Much simpler than the relational algebra expression for $R \div S$. 

The need for more formal semantics for Relational Calculus

- The semantics of the relational calculus expressions considered thus far have been unambiguous (and consistent with our intuition).

- However, consider the following relational calculus expressions:
  - \{ (x_1,...,x_k) : \neg R(x_1,...,x_k) \}
  - \{ (x,y) : \exists z (\text{CHAIR}(x,z) \land y \neq z) \}, where \text{CHAIR}(dpt,name)
  - \{ x : \forall y,z \text{ ENROLLS}(x,y,z) \}, with \text{ENROLLS}(s-name,course,term)

- **Question**: What is the semantics of each of these expressions?
The need for more formal semantics for Relational Calculus

Fact:
- To evaluate \( \{ (x_1, \ldots, x_k) : \neg R(x_1, \ldots, x_k) \} \) we need to know what the possible values for the variables \( x_1, \ldots, x_k \) are.
- If the variables \( x_1, \ldots, x_k \) range over a domain \( D \), then
  \[
  \{ (x_1, \ldots, x_k) : \neg R(x_1, \ldots, x_k) \} = D^k - R.
  \]

Note:
- Intuitively, the relational calculus expression
  \[
  \{ (x_1, \ldots, x_k) : \neg R(x_1, \ldots, x_k) \}
  \]
  is not “domain independent”.
- In contrast, the relational calculus expression
  \[
  \{ (x_1, \ldots, x_k) : S(x_1, \ldots, x_k) \land \neg R(x_1, \ldots, x_k) \}
  \]
  is “domain independent”.
The need for more formal semantics for Relational Calculus

**Note:** The three relational calculus expressions

- \( \{ (x_1,\ldots,x_k) : \neg R(x_1,\ldots,x_k) \} \)

- \( \{ (x,y) : \exists z (\text{CHAIR}(x,z) \land y \neq z) \} \), where \( \text{CHAIR}(dpt,name) \)

- \( \{ x : \forall y,z \text{ENROLLS}(x,y,z) \} \), with \( \text{ENROLLS}(s\text{-name},\text{course},\text{term}) \)

produce different answers when we consider different domains over which the variables are interpreted.

**Fact:** None of these three expressions is “domain independent”.


The need for more formal semantics of Relational Calculus

- **Conclusion:**
  - To give rigorous and unambiguous semantics to relational calculus expressions, we need to make explicit the domain over which the variables (and the quantifiers) take values.
  - We also need to formalize the notion of domain independence.

- **Note:**
  In mathematical logic, the domain of the possible values of the variables is always given explicitly (it is called the universe of the structure on which the formula is evaluated).

- **Question:**
  Is there a natural choice for the domain over which the variables take value?
Active Domain

Definition:

- The **active domain** \( \text{adom}(\varphi) \) of a relational calculus formula \( \varphi \) is the set of all constants that occur in \( \varphi \).
  - If \( \varphi \) is \( R(x,y) \), then \( \text{adom}(\varphi) = \emptyset \)
  - If \( \varphi \) is \( \exists y(R(x,y) \land y > 3) \), then \( \text{adom}(\varphi) = \{3\} \).
  - If \( \varphi \) is \( \forall y(P(x,2,y) \rightarrow R(x,y)) \), then \( \text{adom}(\varphi) = \{2\} \).

- The **active domain** \( \text{adom}(I) \) of a relational database instance \( I \) is the set of all values that occur in the relations of \( I \).
Active Domain and Relative Interpretations

**Definition:** Let $\varphi(x_1,...,x_k)$ be a relational calculus formula and let $I$ be a relational database instance.

- If $D$ is a domain such that $\text{adom}(\varphi) \cup \text{adom}(I) \subseteq D$, then $\varphi^D(I)$ is the result of evaluating $\varphi(x_1,...,x_k)$ over $D$ and $I$, that is, all variables and quantifiers are assumed to range over $D$, and the relation symbols in $\varphi$ are interpreted by the relations in $I$.

- $\varphi^{\text{adom}(I)}$ is $\varphi^D(I)$, where $D = \text{adom}(\varphi) \cup \text{adom}(I)$.

**Note:** $\text{adom}(\varphi) \cup \text{adom}(I)$ is the *smallest* domain on which it makes sense to evaluate $\varphi$. 
Active Domain and Relative Interpretation

Example: Let $\varphi$ be $\neg R(x,y)$ and $I = \{(1,2)\}$.

- $\varphi^{\text{adom}}(I) = \{(2,1), (1,1), (2,2)\}$

- If $D = \{1,2,3\}$, then

  $\varphi^{D}(I) = \{(2,1),(1,1),(2,2),(3,3),(1,3),(3,1),(2,3),(3,2)\}$

Note: This example shows that, in general,

$\varphi^{\text{adom}}(I) \neq \varphi^{D}(I)$
Active Domain and Relative Interpretation

Example: Let \( \varphi \) be \( \forall y R(x, y) \) and \( I = \{(1,2), (1,1)\} \).

- \( \text{adom}(\varphi) \cup \text{adom}(I) = \{1,2\} \)
- \( \varphi^{\text{adom}}(I) = \{1\} \)
- If \( D = \{1,2,3\} \), then \( \varphi^D(I) = \emptyset \)
  (1 is not in \( \varphi^D(I) \), because (1,3) is not in \( I \))

Note: This example also shows that, in general,
\( \varphi^{\text{adom}}(I) \neq \varphi^D(I) \)
**Domain Independence**

**Definition:** A relational calculus formula $\varphi$ is domain independent if for every relational instance $I$ and every domain $D$ such that $\text{adom}(\varphi) \cup \text{adom}(I) \subseteq D$, we have that $\varphi^D(I) = \varphi^{\text{adom}(I)}$.

**Examples:**
- $\neg R(x_1,...,x_k)$ is not domain independent (see slide 24).
- $\exists y R(x,y)$ is domain independent.  (Why?)
- $\forall y R(x,y)$ is not domain independent (see slide 25).
- $P(x) \land \forall y (R(x,y) \rightarrow y > 5)$ is domain independent.  (Why?)
Domain Independence

Examples: The following relational calculus expressions are not domain independent

- \( \{x: (\forall y)(\exists z) \ R(x,y,z)\} \)
- \( \{(x,y): \exists z(\text{CHAIR}(x,z) \land y \neq z)\} \), where CHAIR(dpt,name)
- \( \{x: \forall y,z \\text{ENROLLS}(x,y,z)\} \), where ENROLLS(s-name,course,term)
Theorem: The following are equivalent for a k-ary query \( q \): 
1. There is a relational algebra expression \( E \) such that \( q(I) = E(I) \), for every database instance \( I \) (in other words, \( q \) is expressible in relational algebra).

2. There is a domain independent relational calculus formula \( \varphi \) such that \( q(I) = \varphi^{\text{adom}}(I) \), for every database instance \( I \) (in other words, \( q \) is expressible in domain independent relational calculus).

3. There is a relational calculus formula \( \psi \) such that \( q(I) = \psi^{\text{adom}}(I) \), for every database instance \( I \) (in other words, \( q \) is expressible in relational calculus under the active domain interpretation).