Welcome to:

CMPS 277 – Principles of Database Systems

https://courses.soe.ucsc.edu/courses/cmps277/Fall11/01

Instructor: Phokion G. Kolaitis
Instructor: Phokion G. Kolaitis
- E2 345A, kolaitis@cs.ucsc.edu, (831) 459-4768
- Office Hours: Tu 4:00-5:00pm, Th 11:00am-12:00pm, E2 345A

Recommended Textbook:

Coursework:
- Written homework assignments.
- Term project or term paper.
- Take-home final exam:

Presentations of term projects and term papers:
- Friday, December 2, 2011; time to be determined.
What CMPS 277 is about

Goals:
- Cover a coherent body of basic material in the foundations of databases
- Prepare students for further study and research in database systems

Topics:
- Fundamental concepts and methods in relational database systems
- Database query languages: their expressive power and computational complexity
- Conjunctive query evaluation and query containment
- Recursive queries and Datalog
- Database dependencies and the chase procedure
- Applications to information integration:
  - Schema mappings
  - Data integration and data exchange
What CMPS 277 is **not** about

- Implementation techniques in database systems
- Database statistics and query processing
- Database transactions and concurrency control
- Database recovery and database tuning
- Data mining.

These and other related topics are covered in **CMPS 278, Design and Implementation of Database Systems**

- Also, this course is **not** about making you a better Oracle or a better DB2 database administrator ...
Graduate Courses in Databases at UCSC

- CMPS 277 – Principles of Database Systems

- CMPS 278 – Design and Implementation of Database Systems
  - Winter 2012, Instructor: Neoklis (Alkis) Polyzotis

- CMPS 290H – Topics in Database Systems
  - Not offered in 2011-12

- CMPS 280D – Seminar in Database Systems (2-unit course)
  - Meets once a week throughout the year.
  - Fall 2011: Thursday 12:00-1:30 pm in E2-398.
  - Everyone is welcome to attend and participate.
Database Research at UC Santa Cruz

- **Faculty:**
  - Neoklis (Alkis) Polyzotis
    - approximation techniques for semi-structured data, on-line database tuning, data management in peer-to-peer systems.
  - Wang-Chiew Tan (currently on leave at IBM Research–Almaden)
    - Data provenance, annotations, archiving, information integration.
  - Phokion Kolaitis
    - Databases and logic, database query languages, information integration, inconsistent databases.

- **Postdocs:**
  - Balder ten Cate (U. of Amsterdam, The Netherlands)
  - Gaelle Fontaine (U. of Amsterdam, The Netherlands)
  - Trung-Qiong Tran (National U. of Singapore, Singapore)
Relational Databases: A Very Brief History

- The history of relational databases is the history of a scientific and technological revolution. Edgar F. Codd, 1923-2003

- The scientific revolution started in 1970 by Edgar (Ted) F. Codd at the IBM San Jose Research Laboratory (now the IBM Almaden Research Center)

- Codd introduced the relational data model and two database query languages: relational algebra and relational calculus.
Relational Databases: A Very Brief History

- Researchers at the IBM San Jose Laboratory embark on the System R project, the first implementation of a relational database management system (RDBMS) – see the paper by Astrahan et al.
  - In 1974-1975, they develop SEQUEL, a query language that eventually became the industry standard SQL.
  - System R evolved to DB2 – released first in 1983.
  - Ingres is commercialized in 1983; later, it became PostgreSQL, a free software OODBMS (object-oriented DBMS).
- L. Ellison founds a company in 1979 that eventually becomes Oracle Corporation; Oracle V2 is released in 1979 and Oracle V3 in 1983.
- Ted Codd receives the ACM Turing Award in 1981.
According to Gartner, Inc., June 2007:

“Worldwide relational database management systems (RDBMS) total software revenue totaled $15.2 billion in 2006, a 14.2 percent increase from 2005 revenue of $13.3 billion.”

In 2007, the total RDBMS software revenue increased to $17.1 billion (figures released in July 2008).

<table>
<thead>
<tr>
<th>Company</th>
<th>2006 Revenue</th>
<th>2006 Market Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>7.168B</td>
<td>47.1%</td>
</tr>
<tr>
<td>IBM</td>
<td>3.204B</td>
<td>21.1%</td>
</tr>
<tr>
<td>Microsoft</td>
<td>2.654B</td>
<td>17.4%</td>
</tr>
<tr>
<td>Teradata</td>
<td>494.2M</td>
<td>3.2%</td>
</tr>
<tr>
<td>Sybase</td>
<td>486.7M</td>
<td>3.2%</td>
</tr>
<tr>
<td>Other</td>
<td>1.2B</td>
<td>7.8%</td>
</tr>
<tr>
<td>Total</td>
<td>15.2B</td>
<td>100%</td>
</tr>
</tbody>
</table>
Database Research Today

- A very vibrant community comprising several thousand researchers around the world.
- Several major annual conferences in database research:
  - SIGMOD, PODS, VLDB, ICDE, EDBT, ICDT (top six).
  - Numerous other conferences and workshops.
- Several major scholarly journals dedicated to database research:
  - ACM TODS, VLDB Journal, IEEE TKDE, ...
- Strong database research groups in academia.
- Several world-class database groups in industrial research laboratories:
  - IBM Almaden Research Center, San Jose
  - Microsoft Research – Redmond
  - AT&T Labs
  - HP Labs
  - Yahoo! Research.
Database Management Systems (DBMS)

- **Definition** (informal): A *database* is a collection of interrelated data organized in particular ways.

- **Definition** (informal): A *DBMS* is a set of computer programs that allow us to:
  - Create a database.
  - Insert, delete, modify data, and query data (extract information from data).

- **Key characteristics of DBMS’s:**
  - Ability to manage persistent data (as opposed to transient data).
  - Ability to access large amounts of data conveniently and efficiently.

- **Note:** DBMS ≠ File Systems
Features of DBMS

DBMS provide support for:

- At least one data model (a mathematical abstraction for representing data);
- At least one high level data language (language for defining, updating, manipulating, and retrieving data);
- Transaction management & concurrency control mechanisms;
- Access control (limit access of certain data to certain users);
- Resiliency (ability to recover from crashes).
Data Models and Data Languages

- A data model is a mathematical formalism for describing and representing data.
- A data model is accompanied by a data language that has two parts: a data definition language and a data manipulation language.
  - A data definition language (DDL) has a syntax for describing “database templates” in terms of the underlying data model.
  - A data manipulation language (DML) supports the following operations on data:
    - Insertion
    - Deletion
    - Update
    - Retrieval and extraction of data (query the data).
  - The first three operations are fairly standard. However, there is much variety on data retrieval and extraction (query languages).
The Relational Data Model (E.F. Codd – 1970)

- The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.
- **Question:** What is a relation?
- **Answer:**
  - Formally, a relation is a subset of a cartesian product of sets.
  - Informally, a relation is a “table” with rows and columns.
The Relational Data Model (E.F. Codd – 1970)

The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.

**Question:** What is a relation?

**Answer:**
- Formally, a relation is a subset of a cartesian product of sets.
- Informally, a relation is a "table" with rows and columns.

**CHECKING-ACCOUNT Table**

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-no</th>
<th>customer-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orsay</td>
<td>10991-06284</td>
<td>Abiteboul</td>
<td>$3,567.53</td>
</tr>
<tr>
<td>Hawthorne</td>
<td>10992-35671</td>
<td>Hull</td>
<td>$11,245.75</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
A $k$-tuple is an ordered sequence of $k$ objects (need not be distinct)

- $(2,0,1)$ is a 3-tuple; $(a,b,a,a,c)$ is a 5-tuple, and so on.

If $D_1, D_2, \ldots, D_k$ are $k$ sets, then the **cartesian product** $D_1 \times D_2 \times \ldots \times D_k$ of these sets is the set of all $k$-tuples $(d_1,d_2,\ldots,d_k)$ such that $d_i \in D_i$, for $1 \leq i \leq k$.

**Fact:** Let $|D|$ denote the cardinality (= # of elements) of a set $D$. Then $|D_1 \times D_2 \times \ldots \times D_k| = |D_1| \times |D_2| \times \ldots \times |D_k|$.

**Example:** If $D_1 = \{0,1\}$ and $D_2 = \{a,b,c,d\}$, then $|D_1| \times |D_2| = 8$.

**Warning:** Computing cartesian products is an expensive operation!
A k-ary relation $R$ is a subset of a cartesian product of $k$ sets, i.e.,

- $R \subseteq D_1 \times D_2 \times \ldots \times D_k$.

**Examples:**
- **Unary** $R = \{0,2,4,\ldots,100\}$ (R $\subseteq D$)
- **Binary** $T = \{(a,b): a \text{ and } b \text{ have the same birthday}\}$
- **Ternary** $S = \{(m,n,s): s = m+n\}$
- ...
Relations and Attributes

- **Note:**
  \[ R \subseteq D_1 \times D_2 \times \ldots \times D_k \] can be viewed as a table with \( k \) columns

Table S

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>150</td>
<td>100</td>
<td>250</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- In the relational data model, we want to have names for the columns; these are the **attributes** of the relation.
Relations and Attributes

In the **CHECKING-ACCOUNT** Table below, the attributes are **branch-name**, **account-no**, **customer-name**, and **balance**.

**CHECKING-ACCOUNT** Table

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-no</th>
<th>customer-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orsay</td>
<td>10991-06284</td>
<td>Abiteboul</td>
<td>$3,567.53</td>
</tr>
<tr>
<td>Hawthorne</td>
<td>10992-35671</td>
<td>Hull</td>
<td>$11,245.75</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
A k-ary relation schema $R(A_1, A_2, ..., A_k)$ is an ordered sequence $(A_1, A_2, ..., A_k)$ of k attributes.

- **COURSE**(course-no, course-name, term, instructor, room, time)
- **CITY-INFO**(name, state, population)

Thus, a k-ary relation schema is a “blueprint”, a “template” for some k-ary relation.

An instance of a relation schema is a relation conforming to the schema (arities match; also, in DBMS, data types of attributes match).

A relational database schema is a set of relation schemas $R_i(A_1, A_2, ..., A_{k_i})$, for $1 \leq i \leq m$.

A relational database instance of a relational schema is a set of relations $R_i$ each of which is an instance of the relation schema $R_i$, $1 \leq i \leq m$. 
Relational Database Schemas - Examples

- BANKING relational database schema with relation schemas
  - CHECKING-ACCOUNT(branch, acc-no, cust-id, balance)
  - SAVINGS-ACCOUNT(branch, acc-no, cust-id, balance)
  - CUSTOMER(cust-id, name, address, phone, email)
  - ...

- UNIVERSITY relational database schema with relation schemas
  - STUDENT(student-id, student-name, major, status)
  - FACULTY(faculty-id, faculty-name, dpt, title, salary)
  - COURSE(course-no, course-name, term, instructor)
  - ENROLLS(student-id, course-no, term)
  - ...
Schemas vs. Instances

Keep in mind that there is a clear distinction between
- relation schemas and instances of relation schemas and between
- relational database schemas and relational database instances.

<table>
<thead>
<tr>
<th><strong>Syntactic Notion</strong></th>
<th><strong>Semantic Notion (discrete mathematics notion)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation Schema</td>
<td>Instance of a relation schema (i.e., a relation)</td>
</tr>
<tr>
<td>Relational Database Schema</td>
<td>Relational database instance (i.e., a database)</td>
</tr>
</tbody>
</table>
There are two main paradigms of programming languages: imperative (or procedural) languages and declarative languages.

- **Imperative (Procedural) Languages**: programs are expressed by specifying how the task is to be accomplished (sequence of operations).
  - FORTRAN, C, ...

- **Declarative Languages**: programs are expressed by specifying what has to be accomplished (as opposed to “how”).
  - LISP (functional programming), PROLOG (logic programming), ...
Query Languages for the Relational Data Model

Codd introduced two different query languages for the relational data model:

- **Relational Algebra**, which is a *procedural* language.
  - It is an *algebraic formalism* in which queries are expressed by applying a sequence of operations to relations.

- **Relational Calculus**, which is a *declarative* language.
  - It is a *logical formalism* in which queries are expressed as formulas of first-order logic.

**Codd’s Theorem**: Relational Algebra and Relational Calculus are *essentially equivalent in terms of expressive power*.
  - (but what does this really mean?)
Query Languages for the Relational Data Model

Question:
Is SQL a procedural language or a declarative language?

Answer:
- Neither and both.
- SQL is a hybrid of a procedural and a declarative language that combines features from both relational algebra and relational calculus. (More on this later.)
Desiderata for a Database Query Language

Desiderata:
I. The language should be sufficiently high-level to secure physical data independence, i.e., the separation between the physical level and the conceptual level of databases.
II. The language should have high enough expressive power to be able to pose useful and interesting queries against the database.
III. The language should be efficiently implementable to allow for the fast retrieval of information from the database.

Warning:
- There is a well-understood tension between desideratum II. and desideratum III.
- Increase in expressive power comes at the expense of efficiency.
Relational Algebra

- Relational algebra strikes a **good balance** between expressive power and efficiency.

- Codd’s key contribution was to identify a small set of **basic operations** on relations and to demonstrate that useful and interesting queries can be expressed by **combining** these operations.
  - Thus, relational algebra is a rich enough language, even though, as we will see later on, it suffers from certain **limitations** in terms of expressive power.

- The first RDBMS prototype implementations (System R and Ingres) demonstrated that the relational algebra operations can be implemented efficiently.
The Five Basic Operations of Relational Algebra

- **Group I**: Three standard set-theoretic binary operations:
  - Union
  - Difference
  - Cartesian Product.

- **Group II**: Two special unary operations on relations:
  - Projection
  - Selection.

- **Relational Algebra** consists of all expressions obtained by combining these five basic operations in syntactically correct ways.
Relational Algebra: Standard Set-Theoretic Operations

- **Union**
  - **Input:** Two k-ary relations R and S, for some k.
  - **Output:** The k-ary relation $R \cup S$, where $R \cup S = \{(a_1,\ldots,a_k) : (a_1,\ldots,a_k) \text{ is in } R \text{ or } (a_1,\ldots,a_k) \text{ is in } S\}$

- **Difference:**
  - **Input:** Two k-ary relations R and S, for some k.
  - **Output:** The k-ary relation $R - S$, where $R - S = \{(a_1,\ldots,a_k) : (a_1,\ldots,a_k) \text{ is in } R \text{ and } (a_1,\ldots,a_k) \text{ is not in } S\}$

- **Note:**
  - In relational algebra, both arguments to the union and the difference must be relations of the same arity.
  - In SQL, there is the additional requirement that the corresponding attributes must have the same data type.
  - However, the corresponding attributes need not have the same names; the corresponding attribute in the result can be renamed arbitrarily.
Relational Algebra: Standard Set-Theoretic Operations

- **Cartesian Product**
  - **Input:** An m-ary relation R and an n-ary relation S
  - **Output:** The (m+n)-ary relation R \( \times \) S, where

\[
R \times S = \{(a_1,...,a_m,b_1,...,b_n): (a_1,...,a_m) \text{ is in } R \text{ and } (b_1,...,b_n) \text{ is in } S\}
\]

- **Note:** As stated earlier,

\[
|R \times S| = |R| \times |S|
\]
Algebraic Laws for the Basic Set-Theoretic Operation

- **Union**
  - \( R \cup S = S \cup R \)
    - *(commutativity law – order is unimportant)*
  - \( R \cup (S \cup T) = (R \cup S) \cup T \)
    - *(associativity law – can drop parentheses)*

- **Difference:**
  - In general, \( R - S \neq S - R \) (why?)
  - Does associativity hold for the Difference? *(Exercise)*

- **Cartesian Product:**
  - In general, \( R \times S \neq S \times R \)
  - \( R \times (S \times T) = (R \times S) \times T \)
  - \( R \times (S \cup T) = (R \times S) \cup (R \times T) \)
    - *(distributivity law)*
Algebraic Laws

- **Question:**
  - Why are algebraic laws important?
  - Why should we care about algebraic laws?

- **Answer:**
  - Algebraic laws are important in query processing and optimization to transform a query to an equivalent one that may be less costly to evaluate.
  - Applying correct algebraic laws ensures the correctness of the transformations.
The Projection Operator

- **Motivation:**
  It is often the case that, given a table R, one wants to:
  - Rearrange the order of the columns
  - Suppress some columns
  - Do both of the above.

- **Fact:** The *Projection Operation* is tailored for this task
The Projection Operation

- **Projection** is a family of unary operations of the form
  \[ \pi_{\text{<attribute list>}} (<\text{relation name}>) \]

- The intuitive description of the projection operation is as follows:
  - When projection is applied to a relation R, it removes all columns whose attributes do not appear in the <attribute list>.
  - The remaining columns may be re-arranged according to the order in the <attribute list>.
  - Any duplicate rows are also eliminated.
The Projection Operation - Example

SAVINGS

<table>
<thead>
<tr>
<th>branch-name</th>
<th>acc-no</th>
<th>cust-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aptos</td>
<td>153125</td>
<td>Vianu</td>
<td>3,450</td>
</tr>
<tr>
<td>Santa Cruz</td>
<td>123658</td>
<td>Hull</td>
<td>2,817</td>
</tr>
<tr>
<td>San Jose</td>
<td>321456</td>
<td>Codd</td>
<td>9,234</td>
</tr>
<tr>
<td>San Jose</td>
<td>334789</td>
<td>Codd</td>
<td>875</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{cust-name,branch-name}}(\text{SAVINGS}) \]

<table>
<thead>
<tr>
<th>cust-name</th>
<th>branch-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vianu</td>
<td>Aptos</td>
</tr>
<tr>
<td>Hull</td>
<td>Santa Cruz</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
</tbody>
</table>
More on the Syntax of the Projection Operation

- In relational algebra, attributes can be referenced by position no.

- **Projection Operation:**
  - **Syntax:** \( \pi_{i_1,\ldots,i_m}(R) \), where \( R \) is of arity \( k \), and \( i_1, \ldots, i_m \) are distinct integers from 1 up to \( k \).
  - **Semantics:**
    \[
    \pi_{i_1,\ldots,i_m}(R) = \{(a_1,\ldots,a_m) : \text{there is a tuple } (b_1,\ldots,b_k) \text{ in } R \text{ such that } a_1 = b_{i_1}, \ldots, a_m = b_{i_m}\}
    \]

- **Example:** If \( R \) is \( R(A,B,C,D) \), then \( \pi_{C,A}(R) = \pi_{3,1}(R) \)
The Selection Operation

- **Motivation:**
  - Consider the table
    \[ \text{SAVINGS(\text{branch-name, acc-no, cust-name, balance})} \]
  - We may want to extract the following information from it:
    - Find all records in the Aptos branch
    - Find all records with balance at least $50,000
    - Find all records in the Aptos branch with balance less than $1,000

- **Fact:** The Selection Operation is tailored for this task.
Selection is a family of unary operations of the form

$$\sigma_\Theta (R),$$

where R is a relation and \( \Theta \) is a condition that can be applied as a test to each row of R.

When a selection operation is applied to R, it returns the subset of R consisting of all rows that satisfy the condition \( \Theta \).

Question: What is the precise definition of a “condition”? 
The Selection Operation

- **Definition:** A condition in the selection operation is an expression built up from:
  - Comparison operators $=, <, >, \neq, \leq, \geq$ applied to operands that are constants or attribute names or component numbers.
    - These are the basic (atomic) clauses of the conditions.
  - The Boolean logic operators $\land, \lor, \neg$ applied to basic clauses.

- **Examples:**
  - $\text{balance} > 10,000$
  - $\text{branch-name} = \text{“Aptos”}$
  - $(\text{branch-name} = \text{“Aptos”}) \land (\text{balance} < 1,000)$
Note:

- The use of the comparison operators <, >, ≤, ≥ assumes that the underlying domain of values is totally ordered.

- If the domain is not totally ordered, then only = and ≠ are allowed.

- If we do not have attribute names (hence, we can only reference columns via their component number), then we need to have a special symbol, say $, in front of a component number. Thus,
  - $4 > 100$ is a meaningful basic clause
  - $1 = “Aptos”$ is a meaningful basic clause, and so on.
Algebraic Laws for the Selection Operation

- \( \sigma_{\theta_1} (\sigma_{\theta_2} (R)) = \sigma_{\theta_2} (\sigma_{\theta_1} (R)) \)
- \( \sigma_{\theta_1} (\sigma_{\theta_2} (R)) = \sigma_{\theta_1 \land \theta_2} (R) \)
- \( \sigma_{\theta} (R \times S) = \sigma_{\theta} (R) \times S, \)
  provided \( \theta \) mentions only attributes of \( R \).

Note: These are very useful laws in query optimization.
Relational Algebra

- **Definition:** A *relational algebra expression* is a string obtained from relation schemas using union, difference, cartesian product, projection, and selection.

- Context-free grammar for relational algebra expressions:

  \[ E := R, S, ... \mid (E_1 \cup E_2) \mid (E_1 - E_2) \mid (E_1 \times E_2) \mid \pi_X(E) \mid \sigma_\Theta(E), \]

  where

  - \( R, S, ... \) are relation schemas
  - \( X \) is a list of attributes
  - \( \Theta \) is a condition.
Strength from Unity and Combination

- By itself, each basic relational algebra operation has limited expressive power, as it carries out a specific and rather simple task.

- When used in combination, however, the five relational algebra operations can express interesting and, quite often, rather complex queries.

- **Derived relational algebra operations** are operations on relations that are expressible via a relational algebra expression (built from the five basic operators).
Intersection

- **Input:** Two k-ary relations R and S, for some k.
- **Output:** The k-ary relation $R \cap S$, where

$$R \cap S = \{(a_1, \ldots, a_k) : (a_1, \ldots, a_k) \text{ is in } R \text{ and } (a_1, \ldots, a_k) \text{ is in } S\}$$

- **Fact:** $R \cap S = R - (R - S) = S - (S - R)$

Thus, intersection is a derived relational algebra operation.