Overview of core undergraduate material
Outline

Overview of

- Levels of abstraction
- notions of independence
- E-R model
- Relational model
- SQL
- structure of Database Management Systems
Levels of Abstraction

- **Physical level**
  - describes how a record is stored.

- **Logical level**
  - describes the data stored in the database, and the relationships among the data.

- **View level**
  - hide and/or modify aspects of data types as seen by various users.
Models, Instances, Schemas, and Independence

- A data model is a collection of tools for describing data, relationships among them, their semantics, and constraints they must satisfy.

- Instances vs schemas: similar to types and variables in programming languages.

- **Schema** – the logical structure of the database
  - Analogous to type information of a variable in a program
  - Physical schema: database design at the physical level
  - Logical schema: database design at the logical level
  - View schema: database design at the view level

- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable

- **Physical Data Independence**: the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

- **Logical Data Independence**: the ability to modify the logical schema without changing the view schema of a database
Overview of the E-R model

A database can be modeled as a collection of entities, and relationships among those entities.

- An entity is an object that exists and is distinguishable from other objects.
  - An entity set is a set of entities of the same type that share the same properties.
- An entity is represented by a set of attributes, i.e. descriptive properties possessed by all members of an entity set.
  - Domain – the set of permitted values for each attribute

Attribute types:

- Simple and composite attributes (that can be further divided in component attributes).
- Single-valued and multi-valued attributes
- Derived attributes
Overview of the E-R model

- A relationship is an association among several entities.
- A relationship set is a mathematical relation among \( n \geq 2 \) entities, each taken from entity sets:
  \[
  \{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}
  \]
  where \((e_1, e_2, \ldots, e_n)\) is a relationship.
- A relationship set can have associated attributes.
- Degree of relationship:
  - Degree = the number of entity sets that participate in a relationship set.
  - Relationship sets that involve two entity sets are binary (or degree two).
- Mapping cardinalities express the number of entities to which another entity can be associated via a relationship set.
  - Most useful in describing binary relationship sets.
  - One-to-one, one-to-many-, many-to-many.
Overview of the E-R model

- **Rectangles** represent entity sets
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Ellipses** represent attributes
  - **Double ellipses** represent multivalued attributes
  - **Dashed ellipses** denote derived attributes.
- **Underline** indicates primary key attributes
- We express cardinality constraints by drawing either a directed line (→), signifying “one,” or an undirected line (—), signifying “many,” between the relationship set and the entity set.
Overview of the E-R model

- The participation of an entity in a relationship set can be:
  - **total** (each entity is in at least one relationship in the relationship set) indicated by double line or **partial**
- Entity sets of a relationship need not be distinct
  - Use **roles** to specify the purpose of each entity in the relationship - indicate roles by labeling the lines that connect diamonds to rectangles.

- **Keys**
  - A **super key** of a set is a collection of attributes whose values uniquely determine each element of the set
  - **candidate key** = a minimal super key
  - **primary key** = one of the candidate keys
- **Aggregation** = grouping a relationship set and its associated entity sets into a new “virtual” entity set
Overview of the E-R model

- An entity set without a primary key is referred to as a **weak entity set**.
  - weak entity sets existentially depend on the existence of a **identifying entity set**
    - it must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
    - Identifying relationship depicted using a double diamond
  - The **discriminator (or partial key)** of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
  - The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set’s discriminator.
- A weak entity is depicted by a double rectangle with its discriminator underlined by a dashed line
Overview of the E-R model

- **ISA relationship** captures generalization/specialization (superclass/subclass) relationships between entity sets
  - Depicted by a *triangle* component labeled **ISA**

- **Attribute and relationships inheritance**
  - A lower-level entity set inherits **all** the attributes and relationship participations of the higher-level entity set to which it is linked.

- Membership in a subclass can be
  - Indicated explicitly by the user or defined by a condition on the entity’s attributes
  - Disjoint (entity belongs to at most one lower-level subclass) or overlapping
  - Total (must belong to at least one subclass) or partial

- Can have multiple specializations of an entity set based on different features.
Overview of the E-R model
Overview of the Relational model

- A relation $r$ is a set of $n$-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$, i.e. a subset of $D_1 \times D_2 \times \ldots \times D_n$.

- Each attribute of a relation:
  - Has a name and a domain (set of allowed values).
  - Attribute values are (normally) required to be atomic.
  - Multivalued and composite attribute values are not atomic.
  - The special value `null` is a member of every domain.

- If $A_1, A_2, \ldots, A_n$ are attributes then $R = (A_1, A_2, \ldots, A_n)$ is a relation schema, and $r(R)$ is a relation on the schema $R$.

- An instance of a relation corresponds to a table $T$:
  - Each element $t$ of $r$ is a tuple, and corresponds to a row in table $T$.
  - The attributes of $r$ correspond to the columns of the table $T$. 
Overview of the Relational model

Relation keys
- Let $K \subseteq R$
- $K$ is a **superkey** of $R$ if values for $K$ are sufficient to identify a unique tuple of each possible relation $r(R)$
  - by “possible $r$” we mean a relation $r$ that could exist in the enterprise we are modeling.
- $K$ is a **candidate key** if $K$ is minimal
  - **Primary key** one of the candidate keys
  - Often, we determine primary keys of relations from E-R models

Database
- a collection of relations
- Storing all information as a single (universal) relation results in
  - repetition of information, the need for null values, and certain anomalies (insert, delete, and update anomalies)
  - Normalization theory deals with how to design relational schemas
Overview of Relational algebra

A procedural query language based on the mathematical theory of sets that is the foundation of commercial DBMS query languages

Six basic operators

- select
- project
- union
- set difference
- Cartesian product
- rename

The operators take two or more relations as inputs and give a new relation as a result.

Therefore we can build complex expressions using multiple relational algebra operations
Overview of Relational algebra: operators

**select**

\[ \sigma_p(r) = \{ x : x \in r \text{ and predicate } p \text{ is true for tuple } x \} \]

Predicate \( p \) is a propositional calculus formula with terms that attributes or constants with comparison operators =, <, >, \( \neq \), connected with the logical operators \( \land, \lor, \neg \).

**project**

\[ \Pi_{A_1, A_2, A_3, \ldots, A_k}(r) = \left\{ x : x \in D_{A_1} \times D_{A_2} \times \cdots \times D_{A_k} \text{ and } \exists y \in r \text{ such that } x[A_1, A_2, \ldots, A_k] = y[A_1, A_2, \ldots, A_k] \right\} \]
Overview of Relational algebra: operators

- **union compatible** relations have the same arity (same number of attributes) and the domains of the corresponding attributes are compatible

- **Union**
  \[ r \cup s = \{ x : x \in r \text{ or } x \in s, \text{ and } r \text{ and } s \text{ are union compatible} \} \]

- **Set difference**
  \[ r - s = \{ x : x \in r \text{ and } x \not\in s, \text{ and } r \text{ and } s \text{ are union compatible} \} \]

- **Cartesian product**
  \[ r \times s = \{ xy : x \in r \text{ and } y \in s \} \]
  assumes that the schemas of r and s are disjoint. If not, renaming must be used.

- **Rename**
  \[ \rho_{x(A_1, A_2, \ldots, A_k)}(E) \text{, where } E \text{ is a relational algebra expression} \]
  whose resulting relation has arity k
Overview of Relational algebra: extra operators

- **Intersection**
  \[ r \cap s = \{ t : t \in r \text{ and } t \in s, \text{ and } r \text{ and } s \text{ are union compatible} \} = r - (r - s) \]

- **Natural join**
  \[ r \bowtie s = \left\{ z : z[a] = x[a] \text{ and } z[b] = y[b] \text{ for all attributes } a \in R \text{ and } b \in S, \right. \]
  \[ \left. \text{and all tuples } x \in r \text{ and } y \in s \right\} \]

- **Division**
  \[ r \div s = \{ x : x \in \Pi_{R-S} (r) \text{ and for all } y \in s, xy \in r \} \]
  Note that \[ r \div s = \Pi_{R-S} (r) - \Pi_{R-S} (\Pi_{R-S} (r) \times s) - \Pi_{R-S,S} (r) \]
  the quotient relation \( q = r \div s \) is the largest relation satisfying \( q \times s \subseteq r \)

- **Assignment**
  \[ r \leftarrow \langle \text{relational algebra expression } E \rangle \]
Overview of Relational algebra: extra operators

- **Grouping and aggregation operator**
  \[ F_1(A_1, A_2, \ldots, A_k), F_2(A_1, A_2, \ldots, A_k), \ldots, F_m(A_1, A_2, \ldots, A_k)(r) \]

  where \( A_i \) are the grouping attributes and \( F_j \) are the aggregation functions (\( \text{min, max, sum, count, avg, etc} \))

- **Outer joins**
  - An extension of the join operation that avoids loss of information.
  - Includes all tuples in the natural inner join
    - Adds tuples form one of the operand relations that do not match tuples in the other operand relation to the result of the above join
    - Uses *null* values:
      - *null* signifies that the value is unknown or does not exist
      - All comparisons involving *null* are (roughly speaking) **false** by definition.
  - Inner and the left, right, and full outer join operators

Overview of Relational algebra: nulls

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes.
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*.
- Aggregate functions simply ignore null values.
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same.
Overview of Relational algebra: nulls

- Comparisons with null values return the special truth value *unknown*
  - If *false* was used instead of *unknown*, then \( \text{not} (A < 5) \)
    would not be equivalent to \( A \geq 5 \)

- Three-valued logic using the truth value *unknown*:
  - OR: \((\text{unknown or true}) = \text{true},\)
    \((\text{unknown or false}) = \text{unknown}\)
    \((\text{unknown or unknown}) = \text{unknown}\)
  - AND: \((\text{true and unknown}) = \text{unknown},\)
    \((\text{false and unknown}) = \text{false},\)
    \((\text{unknown and unknown}) = \text{unknown}\)
  - NOT: \((\text{not unknown}) = \text{unknown}\)

- In SQL “\(P\) is unknown” evaluates to true if predicate \(P\) evaluates to unknown

- Result of select predicate is treated as *false* if it evaluates to unknown
Overview of SQL

- SQL is based on set and relational operations with certain modifications and enhancements.

- A typical SQL query has the form:

```
SELECT {DISTINCT} <projected attributes list>
FROM <source relations list>
{WHERE <selection predicate>}
{GROUP BY <grouping attributes>}
{HAVING <predicate on groups>}
{ORDER BY <attributes> ASC|DESC}
```
Overview of SQL

- SQL names are case insensitive
- SQL allows duplicates in relations as well as in query results.
  
  Use **DISTINCT**
- The **select** clause corresponds to the generalized project relational algebra operator
  
  An asterisk * denotes “all attributes”
- The **where** clause specifies conditions that the result must satisfy, and it corresponds to the selection predicate of the relational algebra
- The **from** clause lists the relations involved in the query
  
  corresponds to the Cartesian product operation of the relational algebra
- SQL allows renaming relations and attributes using the **AS** clause: *old-name AS new-name*.
  
  Tuple variables are defined in the **from** clause via the use of the **AS** clause.
- SQL includes a string-matching operator for comparisons on character strings.
Overview of SQL

- The predicate **is {not} null** can be used to check for null values.

- The **union, intersect, except** correspond to the $\cup$, $\cap$, $-$ relational algebra operators
  - They automatically eliminate duplicates - use the modifier ALL to retain duplicates

- Supports grouping and aggregates
  - Attributes in the **select** clause outside of aggregate functions must appear in the group by clause
  - The **having** clause filters out unwanted groupings
    - predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups
  - aggregate functions except **count()** ignore tuples with null values on the aggregated attributes.
Overview of SQL

Nesting of SQL queries

A subquery is an SQL query that is nested within another query
- used to perform tests for set membership, set comparisons, and set cardinality.
- Appear in the where clause
  - as an operand in comparisons
    - `<operand> <comparison> ALL (sub-query) or SOME (sub-query)`
    - `<tuple> {not} IN (sub-query)`
  - Operand in logical connectors
    - `{not} exists (sub-query) and {not} unique (sub-query)`
- Can be correlated (dependent on tuples of the host/outer query)
Overview of SQL

Update SQL statements

- DELETE FROM <relation> {WHERE <clause>}
- INSERT INTO <relation> VALUES(<value-list>)
- INSERT INTO <relation> <SELECT query>
- UPDATE <relation> <list of attribute=value pairs> {where-clause}

Create SQL statements

- CREATE VIEW <view-name> AS <SELECT query>
- CREATE TABLE <relation> (<list of attribute name/type defs>)
- CREATE INDEX <idx-name> ON <relation>(<attribute names>)

SQL constructs for security, tuning, maintenance, etc
Overview of SQL

CREATE TABLE <table-name>

(<attribute-name> <attribute-domain>,)*,
(<integrity-constraint>)*

- **Attribute domains**
  - char(n) and varchar(n).
  - Int, smallint, real, double, float(n), numeric(p,d)
  - Date, time, timestamp, interval
  - User-defined

- **Integrity constraints are**
  - not null
  - primary key \( A_1, \ldots, A_n \)
  - check \( P \), where \( P \) is a predicate
  - Foreign key(<attributes>) references <relation>{(<attributes>)} {on update/delete cascade/set null}
Overview of SQL

A transaction is a sequence of queries and update statements executed as a single unit
- AtomicConsistencyIsolationDurability (ACID) properties
- Transaction specification
  - Begin atomic/end
  - Commit/rollback work
- Isolation levels
  - Transaction and concurrency control managers ensure the consistency of the database due to system/application failures and interaction among the concurrent transactions

A trigger is an Event-Condition-Action SQL statement that is executed as a side effect of another SQL statement
- Events can be insert, delete or update of tuples
- Can access both the old and new values of those tuples to evaluate the trigeg condition
- Action can be taken before or after triggering statement
- Action could for each affected tuple or the triggering statement as a whole
Overview of SQL: trigger Example

```sql
create trigger overdraftTrigger after update on account 
referencing new row as nrow 
for each row 
when nrow.balance < 0 
begin atomic 
  insert into borrower
    (select customerName, accountNumber from depositor 
     where nrow.accountNumber = depositor.accountNumber); 
  insert into loan values(n.row.accountNumber, nrow.branchName, – nrow.balance); 
  update account set balance = 0 
    where account.accountNumber = nrow.accountNumber 
end;
```
Overall DBMS structure