1 Introduction

1.1 Basic Terms

- *Database:* Collection of information
- *Database (Computing science):* Collection of data
  - Composed of:
    - *Database Management System (DBMS):* Software which stores, manages, and retrieves information in a database
      - Uses a relational model to represent data
      - Advantages of using a DBMS to record data:
        - Efficient access:
          - Index structure maps the attribute values to the address of the record
          - Records can be retrieved without scanning all the data
        - Transactions (a set of actions) are processed so the integrity of the database is maintained, regardless of the number of concurrent users or a system crash
      - Diskspace manager (storage): Interacts with the OS file system and data
        - Allows the DBMS to consider the data as a collection of pages
      - Buffer manager: Brings pages from the disk into main manager
        - Manages policy when the main memory is full
- Databases control all interaction between the data and the application(s)
- Advantages of a database:
  - Allows synchronization of access/management/modification of data
  - Logical and physical data independence
  - Reduced application development time
  - Efficient access
  - Concurrent access/control
  - Data integrity and security
  - Crash recovery

1.2 History

- Originally used by large organizations to store textual data
- 1975: 300 databases containing about 50 million records
- 1998: 11,000 databases holding 12 billion records
- Today:
  - Huge amounts of data processed
    - E.g. Walmart handles more than 1 million transactions per hour
  - 2.7 zettabytes of data in the world

        * Facebook stores over 30 petabytes of user-generated data

1.3 Data Abstraction

- *Data model:* Formal language for describing data
  - *Schema:* Description of a particular collection of data using a particular data model
  - *Relational data model:*
    * See section 5
- 3 levels of abstraction (from lowest-level to highest-level):
  - *Physical schema:* Description of how the data is stored and indexed
    * Connects the database with the contextual/logical schema
    * E.g. Relational databases, document-store databases
  - *Contextual/logical schema:* Description of what type of data is stored in terms of the data model
    * Connects the physical schema and the external/view schema
    * E.g. Attributes of a table
  - *External/view schema:* Description of how the data is displayed and accessed
    * Displays select information from the contextual/logical schema for the client
    * *View:* A selective display of data
      · Different users may see different views of the same data
  - Allows:
    * Modification of one level without affecting the other levels
    * Data independence:
      · *Physical data independence:* Ability to modify the physical schema without rewriting the application programs
        - Useful for performance improvements (e.g. moving a file, or adding or removing an index)
      · *Logical data independence:* Ability to modify the logical schema without rewriting the application programs or modifying the views
        - E.g. Adding an attribute to a relation
- *Transaction:* Single execution of a user program in a DBMS
  - E.g. Transferring money between accounts, looking up balance
  - May result in multiple modifications/actions
    * May be interleaved with actions from another transaction
  - Properties:
    * (Being) *Atomic:* Either none or all of a transaction's actions are executed
      · Transactions are considered in their entirety
      · Avoids leaving the database in an inconsistent state due to interrupted transactions
    * (Having) *Consistency:* Transactions should preserve the rules of the database

· E.g. Entering a patient record with a duplicate number would leave the database inconsistent, so the database should reject the transaction

· E.g. Customers require an account; accounts require a customer - database cannot maintain the circular logic, so an application program must maintain this

· E.g. Employers cannot be customers

· Databases remain consistent if transactions are:

- Consistent
- Processed equivalently to some serial order
- Atomic

· Must satisfy all constraints of the database (e.g. example key constraints, foreign key constraints)

· Generally cannot be detected by the DBMS; inconsistencies may relate to domain-specific information

· Users and programs which access the database are responsible for transaction consistency

- Casual users should only use programs which preserve consistency in the database

∗ (Makes sense in) *Isolation:* Transactions should stand on their own and be protected from the effects of other concurrent transactions

· End result should be equivalent to processing the transactions in serial order

∗ (Being) *Durable:* Effects of a transaction should persist afterwards even in the event of a crash

· Transactions are processed in main memory and later written to the disk

· Log is maintained which records information about database writes

- Used to restore transactions not written to disk if a system crashes

· Backups must be maintained

1.4 Languages

- Database languages are divided into two parts:

– *Data definition language (DDL):* Programming language used to create a database and its integrity constraints, and modify external/conceptual schema

∗ Used to create:

· Databases

· Specification of integrity constraints

- E.g. Domain constraints, assertions, authorizations

– *Data manipulation language (DML):* Programming language used to access/change data in a database

∗ Modifies database schema such as:

· Adding new tables

· Deleting tables

        - Adding attributes
    - *Procedural:* Data manipulation language in which users specify what data is required and its method of retrieval
    - *Declarative (nonprocedural):* Data manipulation language in which users specify what data is required, but not its method of retrieval

1.5 Types of Databases

- Relational database management systems:
  - Expresses relationships using foreign keys
  - Advantages:
    - Access to persistent data
    - Concurrent access
    -
  - Disadvantages:
    - Mismatch with object-oriented programming (partially mitigated by object-relational mapping frameworks)
    - Not designed to run on clusters (see
- Alternatives to RDBMSes which do not use SQL: Amazon Dynamo, Google BigTable
- Separate application databases:
  - Integration with web services
  - See section 11
  - Advantages:
    - Flexible
      - Data can be added as required
      - Un-needed columns do not have to be deleted
    - Ease of manipulating non-uniform data
  - Disadvantages
    - Most programs rely on schemata
      - Implicit schema must be deduced
      - Essentially, the schema is moved from the database to the application

2 Entity-Relationship Model

- *Entity-relationship model:* Conceptual visualization of a database' structure through the use of entities and their relationships
  - Used to create a high-level description of the data
- *Entity-relationship diagram:* Visual representation of an entity-relationship model
  - Many alternative ways to be designed

2.1 Components

- *Entity:* Distinguishable, distinct object
  - Concrete or abstract
  - Described by a set of attributes
  - *Attribute:* Information about an entity
    - *Composite:* Attribute composed of multiple different sub-attributes
      - E.g. An address is composed of a number, street name, and city
    - *Multi-valued:* Attribute with multiple sub-attributes of the same type
      - E.g. Phone numbers for home, work, cell
      - Should be replaced with an entity set, with relationships used to connect with its entity
    - *Derived:* Attribute which contains information from related attribute(s)
      - E.g. Total number of employees
      - Do not need to be stored in the database; can be immediately derived when necessary
    - *Foreign key:* Attribute which references a (primary) attribute from another table
      - Can be explicitly declared as non-null
    - May have a null value
    - Has a domain (set of possible attribute values)
  - *Entity set:* Collection of entities under some categorization
    - Not necessarily disjoint (multiple entity sets may contain the same entity)
    - Cannot contain multiple of the same entity
- *Relationship:* Association between entities
  - E.g. Student *enrolled* in a course
  - Constraint: Rule or policy which restricts a relationship
  - *Relationship set:* Set of related relationships
    - Often represented with a verb/action
    - Attributes:
      - Primary key depends on the key constraints of the relationship and the primary keys of its connected entities (not shown in an entity-relationship diagram)
      - *Descriptive attribute:* Detail of a relationship set

        - Not usable to identify the relationship set with
        - Cannot be part of the relationship set's primary key

* Used for properties which cannot be associated with any of the entity sets

* An n-ary relationship set R is an association between n entity sets $E_1 \cdots E_n$

    - R is a subset of $\{(e_1, \ldots, e_n) | e_1 \in E_1, \ldots, e_n \in E_n\}$

* Role of an entity is the function it plays in the relationship

* E.g. Employee (entity) works (relationship) on a project (entity).
'Working on' is a relationship set, as it can be applied to to many employees working on projects. The start date of the employee working on the project is a descriptive attribute.

* E.g. Manager (entity) manages (relationship) an employee.
'Manages' is a relationship set, as it can be applied to many managers managing employees. The superiority of the manager over the employee is a descriptive attribute.

* Mapping cardinalities:

    - Specify how many of each entity may be related to another entity

    - Types of mapping between sets:

        - *One-to-one:* Each entity in A maps to at most one entity in B; each entity in B maps to at most one entity in A

            – E.g. One employee is only assigned one office; one office can only hold one employee

        - *One-to-many:* Each entity in A may map to any number of entities in B; each entity in B maps to at most one entity in A

            – E.g. A person can own many bank accounts; a bank account can only be owned by one person

        - *Many-to-one:* Each entity in A maps to at most one entity in B; each entity in B may map to any number of entities in A

            – E.g. A player can be in only one soccer team; a soccer team can contain many players

        - *Many-to-many:* Each entity in A may map to any number of entities in B; each entity in B may map to any number of entities in A

            – E.g. A person can be enrolled in many courses; a course can contain many students

    - In an entity-relationship diagram: directed lines

        - A directed line from an entity set to a relationship set indicates that the entities in the set can only be involved in one relationship in the set

* *Participation constraint:* Rule that any entity in some entity set must be involved in a relationship(s)

    - Examples:

        - An account must be owned by at least one customer; a customer does not have to have an account

        - A child must attend a single pre-school, which must have at least one child

        - A branch of a bank must contain at least one account; each account must be at only one branch
      - *Total participation:* A participation constraint exists
      - *Partial participation:* A participation constraint does not exist
      - In an entity-relationship diagram: Double line between the relationship and entity
    - *Key constraint (entity-relationship model):* An entity can participate in only one relationship
  - Does not have to be binary
    - Ternary relationship example: A branch purchases a part from a supplier

2.2 Additional Features

- *Weak entity set:* Entity set containing entities which cannot be identified using only their own attributes
  - Identification: Compound key of its partial key with the primary key of another entity set
  - Total participation and key constraint in the identifying relationship
  - *Owner entity set:* Entity set containing a primary key which is combined with a partial key of a weak entity set to identify a weak entity
  - In an entity-relationship diagram: Double-line from the weak entity set to the relationship set; double-border around the relationship set
- *Subclass/Superclass:* Hierarchy of entity sets which are contained within other entity sets, and vice versa
  - Used to:
    - Avoid redefining common attributes
    - Add additional descriptive attributes
    - Identify the set of entities which participate in a particular relationship
  - Subclasses are defined by its attributes and the attributes of its superclass(es)
  - May have its own additional attributes
  - 'is a' relationship
    - E.g. A (subclass) is a B (superclass); A specializes/generalizes B
  - *Specialization:* Identification of subsets with additional attributes from an existing entity set
  - *Generalization:* Identification of common attributes of entity sets and creating a new entity set with the attributes
  - *Condition-defined participation:* A given entity is in a subclass if it meets a specific condition
  - *User-defined participation:* A given entity is in a subclass if it is assigned as such by the user
    - E.g. Employees must be manually promoted to manager
  - *Disjoint:* Subclass whose entities cannot appear in another subclass
    - E.g. Birds and mammals
    - Assumed if unlabeled
  - *Overlapping:* Subclass whose entities may appear in another subclass
    - E.g. Customer and employee

        * Must be labeled
    - *Coverage constraint:* Each superclass entity must belong to a subclass
        * E.g. Animals must belong to a kingdom
- *Aggregation:* Using a relationship set as an entity set to assign it to another relationship
    - Used when there is a relationship between an entity set and another relationship
    - E.g. A branch can only buy the same part from the same supplier; the aggregate entity is the branch buying the part, and there is the relationship where the aggregate entity buys from the supplier

2.3 Design Principles

- Faithfulness:
    - Design must be accurate to the specification and the enterprise
    - All relevant aspects must be represented
- Simplicity:
    - Redundancy confuses understanding, wastes storage, creates inconsistency
    - Use attributes over entity sets or relationships
    - Specify as many constraints as possible
- Element choices:
    - Entity set vs attribute
    - Entity set vs relationship set
    - Type of relationship:
        * Multiple binary relationships
        * n-ary relationship
        * Aggregation
- Relationship sets' attributes must be descriptive (i.e. cannot be a part of the primary key)

2.4 Converting Entity-Relationship Diagrams to a Database

- Each entity and relationship set is represented by a unique relation
- Constraints in the database should be created to match constraints in the model
- Primary keys are implicitly NOT NULL
- Table representing an entity set:
    - A column for each attribute
    - The domain of each attribute should be specified
    - Primary key should be specified
- Relationship sets:
    - Attributes:
        * Primary keys of the entity sets (declared as foreign keys)
        * Descriptive attributes

- Mapping cardinalities determine the primary key of the relationship, as well as whether the relationship set requires a separate table or just foreign keys with constraints
  - No cardinality constraint (many-to-many) (n-ary relationship set):
    - Requires a separate table
    - Primary key: Compound key; union of the attributes from the entity sets (declared as foreign keys)
    - Descriptive attributes declared as non-foreign keys
  - Many-to-one constraint (n-ary relationship set):
    - Represented with either a table or an attribute of an entity set
    - Primary key: Primary key of (one of) the entity from the 'many' entity set(s)
  - One-to-one constraint:
    - Primary key: One and only one primary key of the entity sets
- Participation constraint:
  - Declare the attribute on which a foreign key exists to be NOT NULL
    - Does not work if a relationship set is not represented in a separate table
  - May require assertions/triggers

- Weak entity set:
  - Must have a foreign key referencing its owner's primary key
- Class hierarchy: (two ways)
  - Create separate tables for the superclass and subclasses; make both a superclass and subclass tuple for a given subclass instances
    - Subclass tables only have the attributes which the superclasses do not have
    - Subclass tables have a foreign key referencing the superclass entity
    - Deletion should cascade from superclass records to subclass records
  - Create tables for subclasses only
    - Coverage constraint must exist (all superclass entities must be also subclass entities)
    - Subclass tables contain all attributes of the superclass
- Aggregation:
  - Defined by the relationship set connecting the aggregate entities, as well as another relationship set connected to another entity which references the aggregate entities
  - Relationship between the aggregate entities contains:
    - Primary key of the participating entity set
    - Primary key of the relationship defining the aggregation
    - Descriptive attributes
  - If there is total participation between the aggregate entity and its relationship, and there are no descriptive attributes:
    Move the aggregate entities' relationship attributes into the table representing the relationship with the aggregate entity

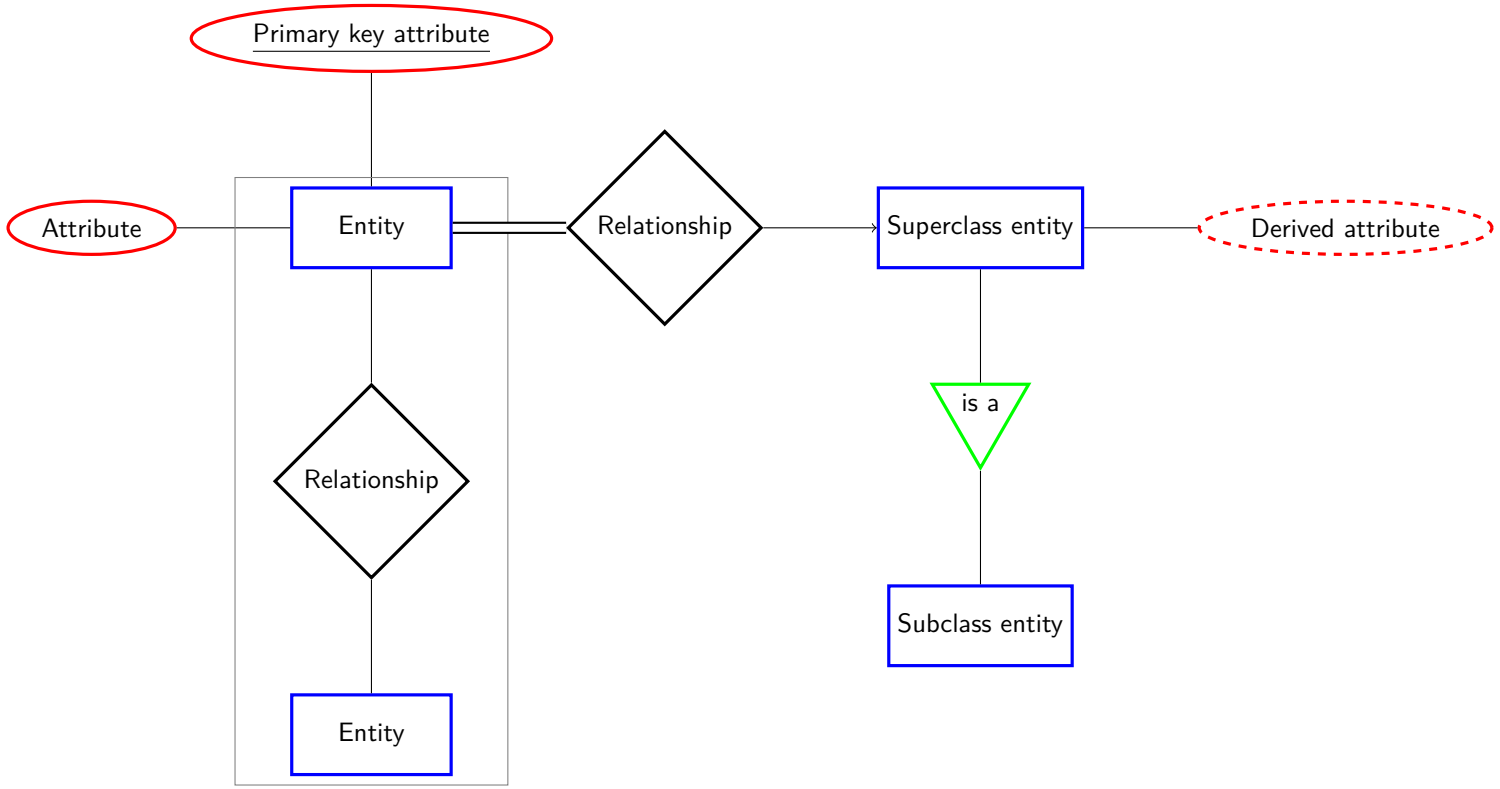
2.5 Entity-Relationship Diagrams in CMPT 354

- Entity: Rectangle
  - Attribute: Oval
    - Linked to their entity with a line
    - Derived attribute: Oval with a broken outline
    - Attribute of a primary key is underlined
- Relationship: Line
  - Relationship set: Line with a description in a diamond
    - X-to-one relationship: Directed line pointing from the entity with restrictions to the relationship set
  - Participation constraint: Double/thick line
  - Subclass/superclass: Triangle with the words 'is a', the superclass connected above, and the subclass(es) connected below
    - Disjoint: Default, unmarked
    - Overlapping: Labeled as 'OVERLAP'
    - Coverage constraint: Labeled as 'COVER'
- Aggregation: Rectangle around the entity sets and relationship set; rectangle as a whole is treated as an entity set

2.6 Example Diagram

See figure 1.

Figure 1: Example Entity-Relationship Diagram



3 Design

3.1 Structure

- Conceptual database design:
  - Identify the entities and relationships (see subsection 2)
  - Identify the entity and relationship information
  - Identify the integrity constraints
  - Discuss the conceptual model with the client
- Logical database design:
  - Decide on a data model to use
  - Choose a DBMS to use
  - Create a database schema using the conceptual schema
    - Avoid redundant information or inability to record information

3.2 Identification of Tables

- *Key/Superkey:* Set of attributes with values which uniquely identify an entity in an entity set
  - Mathematical definition: A subset K of a relation R is a superkey of R if, for all pairs of tuples t_1 and t_2, $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$
    - I.e. K, a set of attributes of relation R, is a superkey if, for all distinct pairs of tuples, there are no two tuples with the same values for K
  - *Candidate key:* Superkey with no extraneous attributes
    - E.g. Class number, teacher + term + meeting times
    - *Primary key:* Candidate key chosen to represent a tuple in the relation (rows in the table)
      - Should be chosen from an attribute(s) which never changes (e.g. Social Insurance Number)
      - Can be arbitrarily generated for easier classification
      - In an entity-relationship diagram: Attributes underlined

3.3 Constraints

- *Integrity constraint:* Rule which restricts the data in a database
  - *Domain constraint:* Integrity constraint on a given data column of the type of data
  - *Key constraint (integrity constraint):* Integrity constraint which identifies primary keys and candidate keys
  - *Foreign key constraint:* Integrity constraint which references a primary key from another tables
    - *Foreign key:* Attribute(s) which reference a primary key of another entities
      - References the entire primary key
      - Number of attributes and attribute types must be consistent; attribute names may be different

3.4 Normalization

- *Normalization:* Minimizing redundancy of related information
  - Goals:
    - Efficient space use
    - Efficient processing of data
    - Minimal possibility of inconsistent data and therefore data integrity
- *Repetition:* Data which is repeated for no use, and which obscures updating
  - E.g. A customer can own many accounts; accounts can have many customers.
    Account = {customerID, accNumber, balance, type}
    There will be multiple rows for each account with multiple customers, so updating the information of one customer's account will require updating multiple records.
- OTHER:
  - *Lossless join:* A join where only and all of the appropriate data is returned
  - *Lossy join:* A join where extraneous information/records is created
  - E.g. Decomposing a table into two tables, one of which has no primary key, and joining them together will create multiple records with useless information
- Normalized database design:
  - Creating the relational database schema:
    - Do not use composite attributes or set valued attributes
    - Do not assign attributes to relationship sets which are not descriptive
  - Decompose the tables and make sure they satisfy at least one normal form (see subsection 3.5)

3.5 Normal Forms

- *First Normal Form:* Placing all data in one table while removing repeating groups
  - Adds rows for repeated groups
  - Uses a compound primary key
  - Creates fixed-length records
  - Complies with the relational model
  - Disadvantages:
    - Redundant data due to repeated information in groups of rows
    - An instance of one attribute of the compound primary key cannot be inserted without creating an instance of the other attribute(s) of the compound primary key
    - Deleting the last instance of an attribute in a primary key also deletes...
    - Update...
- *Second Normal Form:* Removing partial key dependencies from a First Normal Form...
  -
  - Any database in 2NF is also in 1NF
  - Disadvantages:

    - Only considers partial key dependencies; ignores non-key dependencies
    - Redundant data
    - Insert/delete/update anomalies
- *Third Normal Form:*
- See *functional dependencies*, subsection
  - Goals of decomposition:
    - Should not result in a lossy join

      .

    - *Dependency preservation:* If a set of attributes depends on an attribute, the dependency should be maintained in one table
    - Minimal redundancy
- *Boyce-Codd Normal Form (BCNF):* Obtained by simplifying functional dependencies from Third Normal Form
  - Ignores multi-valued dependencies
  - The only functional dependencies are those where the key of the table determines attributes (excluding trivial dependencies)
  - Database is in BCNF if each table is in BCNF
  - Strict definition: A relational schema R is in BCNF with respect to a set of dependencies F if, for all...

3.6 Views

- *View:* External schema which displays a particular set of data
  - Convenient for users to access data without referring to multiple tables
  - Ensures that users can only access specific data
  - Masks changes in the conceptual schema
  - Updates may cause problems when derived from multiple tables, and does not include the primary keys of all tables, so they are generally only allowed on views derived from a single table

4 Query Optimization

- *Query optmization:* Process of finding an equivalent and more efficient query
  - Process:
    - Convert the SQL query to relational algebra
    - Find equivalent queries and their estimated costs
    - Choose the most efficient query
  - Modern DBMSes automatically optimize queries
- Process of an unoptimized query:
  - Read as much of the first table into main memory as possible
  - Scan the second table in the Cartesian product for each block of records in main memory
  - Output the resulting records
- Process of an optimized query:
  - Apply other operations when/before each Cartesian product/natural join is computed
    - Selection and projection do not require additional reads/writes
- Example of a slow query:

```
SELECT C.customerID, C.lastName, A.balance
FROM Customer C, Owns O, Account A
WHERE A.accNumber = O.accNumber AND
          C.customerID = O.customerID AND
          A.branchName = 'London' AND
        C.firstName = 'Bob'
```

  - Process:
    - Computes the Cartesian product of the three tables
    - Selects any records which match the given conditions
    - Projects all columns in the SELECT list (in the final table)
- Example of a faster query:

```
SELECT C.customerID, C.lastName, A.balance
FROM (SELECT accNumber, balance
          FROM Account
          WHERE branchName = 'London' ) AS A
        NATURAL INNER JOIN Owns
        NATURAL INNER JOIN
        (SELECT customerID, lastname
        FROM Customer
        WHERE C.firstName = 'Bob') AS C
```

  - Process:
  - Selects all records which match the given conditions
  - Projects all columns in the SELECT list (in each separate table)
  - Computes the natural join of the three resulting tables

- Indexing may reduce the cost of a selection (see

5 Relational Model

- *Relational database:* Collection of tables (mathematical concept of a relation
  - Tables have unique names; rows represent an entity or relationship
- *Tuple:* Record/row of a relational database
- *Relation:* Set of unique tuples
  - *Relation instance:* Actual table with a particular set of rows
    - *Cardinality (relation instance):* Number of tuples in a given relation instance
  - *Relation schema:* Column headings of a table
    - Consists of the names of the relation, the names of the columns, and the domain of each field
    - *Domain:* Set of possible values
      - *Relation instance:* Subset of the Cartesian product of the domains; set of distinct, valid tuples/records
        - *Cartesian product:* All elements in a set paired with all elements in another set
        - I.e. One possible row
    - E.g. Customer relation:
      Customer = { sin, firstName, lastName, age, income }
      Domain: integer(9), char(20), char(20), integer, realNumber
  - Degree (arity): Number of fields of a relation

5.1 Relational Models vs. Databases

- For differences in terminology, see table 1

Table 1: Equivalent terms for Relational Models and Databases

Relational models:	Databases:
Relation schema	Table schema
Relation instance / relation	Table
Field	Column/attribute
Tuple	Record/row

6 Functional Dependencies

- *Functional dependency:* For any given value of an attribute(s), there can only be a single value for a set of attributes
  - Similar to the notion of a superkey for a set of attributes
  - x functionally determines y and y is functionally determined by x if there can only be one y value for a given x value
  - Mathematical definition: If R is a relation and $\alpha \subseteq R$ and $\beta \subseteq R$, then $\alpha \rightarrow \beta$
  - E.g. All of the same car models have the same capacity; therefore, capacity is functionally determined by the model.
    model $\rightarrow$ capacity
  - Used to:
    - Specify additional constraints
    - Decompose tables to reduce repeated data
  - Statement about all possible legal instances of a relation
    - Relation R satisfies the set of functional dependencies F if R is legal under F
-
- Primary key is a special case of a functional dependency
  - If X $\rightarrow$ Y on R...
- Set of functional dependencies can be used to identify additional functional dependencies
  - F implies the additional functional dependencies

6.1 Armstrong's Axioms

- Named after William Armstrong
- Axioms:
  - Where X and Y are sets of attributes
  - *Reflexivity (A-Axioms):* If X $\supseteq$ Y, then X $\rightarrow$ Y
    - E.g. sin, firstName $\rightarrow$ sin
    - E.g. sin $\rightarrow$ sin
    - *Trivial functional dependency:* X $\rightarrow$ Y where X $\supseteq$ Y
  - *Augmentation (A-Axioms):* If X $\rightarrow$ Y, then XZ $\rightarrow$ Y, Z for any Z
    - E.g. If sin $\rightarrow$ firstName, then sin, lastName $\rightarrow$ firstName, lastName
  - *Transitivity (A-Axioms):* If X $\rightarrow$ Y and Y $\rightarrow$ Z, then X $\rightarrow$ Z
    - E.g. If sin $\rightarrow$ age and age $\rightarrow$ seniority, then sin $\rightarrow$ seniority
- Derivable axioms:
  - *Union:*
  - *Decomposition (A-Axioms):* If X $\rightarrow$ YZ, then X $\rightarrow$ Y and X $\rightarrow$ Z
    - Proof:

- *Pseudo-transitivity (A-Axioms):* If X $\rightarrow$ Y and WY $\rightarrow$ Z, then XW $\rightarrow$ Z
  - Proof: See...

Assume X $\rightarrow$ Y.

By the axiom of augmentation, XW $\rightarrow$ WY.

Assume WY $\rightarrow$ Z.

By the axiom of transitivity, XW $\rightarrow$ Z.

- *F:* Set of identified functional dependencies
- *F^+:* Closure of *F*; all functional dependencies implied by *F*
- *X^+:* Attribute closure of the set of attributes in X; list of attributes containing all attributes that are functionally dependent on X
- *Sound:* Axioms which do not generate incorrect functional dependencies
- *Complete:* Axioms which allow F^+ to be generated from a given F

6.2 Removing Redundant Functional Dependencies

- *Extraneous attribute:* Attribute
  - Attribute a is extraneous in X if $a \in X$
- Testing if attribute $a \in X$ is extraneous in X, compute $(\{X\} - a)^+$ with...
- *Canonical cover (of F):* Minimal set of functional dependencies equivalent to *F* without redundant or partial dependencies
  - F logically implies all dependencies in F_c;
    F_c logically implies all dependencies in F;
    no functional dependency in F_c contains an extraneous attribute;
    each left side of a dependency in F_c is unique
  - Denoted by F_c

7 SQL

- *SQL (Structured Query Language):* Programming language designed to manage data in a database
  - Data manipulation language and data definition language (see subsection 1.4)
    - DML is used to write queries and insert/delete/modify records
    - DDL is used to create/delete/modify table definitions, and create integrity constraints and views
- Standards:
  - SQL-92: 3rd revision of the standard
  - SQL 1999 / SQL 3: 4th revision of the standard
  -

7.1 Syntax

7.1.1 Introduction

- Reasons to use short, explicit tuple variables:
    - Readability
    - Efficiency
    - Distinguishing between columns with the same name from different tables
    - Referencing the same table twice in the from-list
      - E.g. Finding the greatest income in the table
- LIKE:
  - Pattern matching - 'Text' LIKE 'Text'
  - % symbolizes 0+ arbitrary characters
  - _ symbolizes for exactly one arbitrary character
  - % and _ characters must be escaped with \
  - Does not ignore whitespace
- SIMILAR:
  -
- Set operations:
  - Tables must be union-compatible (see
  - WIll remove duplicate
  - Set union: UNION
    - Difference between the ONE OF TWO BRANCHES UNION example:
      - Someone with an account in both Robson and Lonsdale will be returned twice in the first one (no default duplicates removed), and once in the second (set union) one
  - Set intersection: INTERSECT
  - Set difference: EXCEPT

  -
- Strings:
  - Enclosed in single quotes
  - Single quote can be specified with two single quote characters
  - Comparable lexicographically using comparison operators
    - Whitespace is ignored
- Date:
- Time:
- NULL:
  - Databases may contain NULL when data does not exist or is unknown
  - Using a non-special operator to test for NULL values evaluates to UNKNOWN
  - Special operators:
    - IS NULL:
    - IS NOT NULL:
  - Examples:
    - $23 <$ NULL $=$ UNKNOWN
    -
  - UNKNOWN is interpreted as FALSE
- Nested queries:
  - Uncorrelated/independent query: Sub-query does not contain...
  - Correlated query: Sub-query must be evaluated for each row in the super-query
    - Inefficient
- Division in SQL:
  - Can be computed using NOT EXISTS and EXCEPT by SELECTing the target values WHERE NOT EXISTS (all attributes to be contained EXCEPT all attributes of a given target)
- *UNIQUE:* Returns TRUE if no two rows contain the same attributes, and FALSE otherwise
  - If the table contains no rows or only one row, TRUE is returned
- *NOT UNIQUE:* Returns TRUE if there exist at least one pair of rows with the same attributes, and FALSE otherwise
- Aggregate operators:
  - *COUNT(column_ name):* Returns the number of values in a column
  - *SUM(column_ name):* Returns the sum of the values in a column
  - *AVG(column_ name):* Returns the average of the values in a column
  - *MIN(column_ name):* Returns the minimum of the values in a column
  - *MAX(column_ name):* Returns the maximum of the values in a column
  - May be used with DISTINCT to choose only unique values
    - E.g. COUNT(DISTINCT column)

- Keywords are not case-sensitive

7.1.2 Tables

- Creating a table:

```
CREATE TABLE TableName
(
        attribute       DOMAIN,
        attribute       DOMAIN NOT NULL,
        CONSTRAINT unique_attributename
                UNIQUE(attribute),
        CONSTRAINT unique_attributescombinedname
                UNIQUE(attribute, attribute),
        CONSTRAINT fk_tablename_references_tablename
                FOREIGN KEY(attribute)
                REFERENCES ReferencedTableName,
        PRIMARY KEY(attribute)
)
```

  - UNIQUE identifies a candidate key (see *candidate key*, subsection 3.2)
    - Records which have an attribute specified as UNIQUE cannot have the same value for that attribute
    - Having multiple attributes in a UNIQUE statement creates a single candidate key consisting of all the attributes
  - PRIMARY KEY identifies a primary key (see *primary key*, subsection 3.2)

- Deleting a table:

```
DROP TABLE TableName
```

7.1.3 Modifying Schema

- Adding a column:

```
ALTER TABLE TableName
ADD attribute TYPE
```

- Deleting a column:

```
ALTER TABLE TableName
DROP attribute
```

7.1.4 Modifying Data

- Modifying a record:

```
UPDATE TableName
SET attribute=value
WHERE attribute=value  -- Primary key
```

- Can affect more than one record
- Though the SET statement comes before the WHERE statement, the WHERE statement is evaluated first

- Inserting a record into a table:

```
INSERT
INTO TableName (attribute, attribute)
VALUES (value, 'string')
```

- List of column names is optional
- Values must be in the same order as the columns

- Deleting a record from a table:

```
DELETE
FROM TableName
WHERE attribute=value
```

- Deleting all records from a table:

```
DELETE
FROM TableName
```

7.1.5 Accessing Data

- Query:

```
SELECT select-list
FROM from-list
WHERE condition
```

- Equivalent to $\pi_{select-list}(\sigma_{condition}(from\ list_1 \times list_2 \times \ldots \times list_n))$ (see)
  - SELECT performs a projection; WHERE performs a selection
  - *
- Returns a multiset/bag of rows
  - *Bag:* Collection of objects, with duplicates allowed
  - The keyword DISTINCT will remove duplicates
  - E.g. SELECT DISTINCT select-list FROM ...
- select-list: Column names belonging to tables in the from-list
  - Must be a candidate key
  - To select all columns, use *
    - E.g. SELECT * FROM ...
- from-list: List of table names
- condition: Boolean expression
  - Must also specify the equal fields
- Does not automatically remove duplicates
- Select and condition statements can utilize arithmetic operations

- AS keyword renames a column
  - E.g. SELECT col1, col1 AS renamedcol FROM Table
- E.g. Rupert Giles
- ORDER BY: ASC/DESC
  - Default is ascending order

7.1.6 Views

- Creating a view:

```
CREATE VIEW ViewName
(
        Specifier.attribute,
        Specifier.attribute
)
AS
(
        SELECT Specifier.attribute, Specifier.attribute
        FROM Table1Name Specifier1, Table2Name Specifier2
        WHERE Specifier1.attribute = Specifier2.attribute
)
```

7.1.7 Invalid Transactions

- SQL will reject any operation which violates a constraint by deleting, modifying, or creating invalid data
- Responses:
  - Can be specified as ON DELETE or ON UPDATE in the constraint declaration
  - Possible responses:
    - Reject the transaction (NO ACTION)
    - Delete/update the referencing record (CASCADE)
    - Set the referencing record's foreign key attributes to null (SET NULL)
      - Only upon deletion
      - Cannot be used on an attribute with a participation constraint
    - Set the referencing record's foreign key attributes to a default value (SET DEFAULT)
      - Only upon deletion
      - Default value must be specified in the foreign key
  - E.g.

```
CREATE TABLE table_name
(
        attribute INTEGER,
        attribute_foreign_key INTEGER,
        FOREIGN KEY (attribute_foreign_key)
                REFERENCES other_table_name
```

```
                ON DELETE CASCADE
                ON UPDATE NO ACTION,
        PRIMARY KEY (attribute)
)
```

7.2 Hierarchies and Aggregation

- Subclasses can either be represented using:
  - An entity with all the attributes in the superclass entity as well as all the attributes in the subclass entity
  - An entity with the attributes not in the superclass entity, and a foreign key referencing a superclass entity with the remaining attributes
- Aggregation
  - Represented by a connection between the relationship set of the aggregated entities and the other entity

7.3 Constraints

- Participation constraints:
  - Declare foreign keys as NOT NULL (if the relationship set is not represented in its own table)
  - May require assertions/triggers
- Integrity constraints:
  - Specified when the database schema is specified
  - Checked with every change to ensure that no constraints are violated
- Primary key:
  - Each table has one and only one primary key
  - Attribute(s) must be unique and non-null
- Table constraints are within a single table
  - E.g. A population greater than 0
- Assertions may cover multiple tables
  - Checked whenever any of the related tables are modified
- Weak entity sets' foreign keys referencing the owner's primary key should be specified as ON DELETE CASCADE (deletes the weak entity when the owner is deleted)
- Primary key constraint:
  - Can be specified in:
    - CREATE TABLE statement
    - ALTER TABLE statement
  - Clustered index is created by default (see)
- Foreign key constraint:
  - If table R contains a foreign key on attributes $\{r\}$ which references table S, $\{r\}$ must...

  - Can be specified in:
    * Separate foreign key statement
    * REFERENCES statement
  - Action taken if referenced records are updated/deleted can be specified
    * For SQL, see
    * If any cascading deletion/update causes a violation, the entire update is rejected
- *General/Table constraint:* Constraint on a single table (but may reference other tables)
  - Can be specified in the CREATE TABLE statement
  - Keyword: CHECK (boolean_ expression)
    * Transaction is rejected if the given boolean expression evaluates to false
    * SQL can only handle comparisons with scalar expressions (only one value; expressions with SQL queries are not allowed), but can use results of User Defined Functions (see
- *Domain:* Range of possible values
  - Can be specified in:
    * CREATE DOMAIN STATEMENT
  -
  - Can replace a base type in a CREATE TABLE statement
  - Comparisons between two domains can be made in terms of their base types
- *Assertion:* Constraint on multiple tables
  - Can be specified in:
    *
  - Separate statements in the database schema
  - Tested whenever the database is updated (therefore, may have significant overhead)
  -
  - Syntax:

```
CREATE ASSERTION name
CHECK (boolean_expression)
```

- *User Defined Function (UDF)*: SQL server function which can return scalar values or tables
  - Can be written in T-SQL or a .NET language
  - Useful for:
    * Modular programming
    * Faster execution
    * Reduced network traffic
  - Syntax:

```
CREATE FUNCTION funcName()
RETURNS type
AS
BEGIN
```

```
        DECLARE @variable_name type
        IF value = value  -- Can have queries
                SET @variable_name = value
        ELSE
                SET @variable_name = value
        RETURN @variable_name
END
```

8 Triggers

- *Trigger:* Procedure invoked by the DBMS as a response to a specified change/condition
  - Can be used to...
    - E.g. Sending an email
  - Generally will reject the transaction
- Components:
  - *Event (trigger):* Specified modification to a database
    - Insert/deletion/change
  - *Condition (triggger):*
    - IF/ELSE structure
  - *Action (trigger):*
- *Active database:* Set of triggers associated with a database
- Available in most database products
- Syntax:

```
CREATE TRIGGER TR_trigger_name
ON TableName/View
FOR/AFTER/INSTEAD OF OPERATION -- CREATE, INSERT, UPDATE, DELETE, etc.
AS RAISERROR(
```

-
- Order of checks:
  - Check constraints
  - Check foreign key constraints
  - Create inserted/deleted tables
  - Execute a triggering statement
  - Execute the (after) trigger

8.1 Indexes

- *Index:* Data structure which provides efficient access to records by mapping attribute values to record IDs
  - *Record ID:* Disk address of a record

  - Avoids the need to read an entire table from disk into main memory
  - Variation on hash tables (for an arbitrary value(s)) and/or binary search trees (for a range of values)
  - Based on the value of an attribute(s)
  - Conceptually similar to a book index
- *Index search key:* Attribute used to find an index entry
- A table may have multiple indexes and single-attribute and/oror multi-attribute (compound) search keys
- *Hash index:*
- *Tree index:*
  - Allows easier lookup of a range of values
- *Clustered index:* Index where the data is sorted by the index search key
  - Only one clustered index per table
  - Greater support for range queries
- Cost of maintenance:
  - Insertions/deletions always require modification of all table indexes
  - Updates may require modification of all table indexes
  - Indexes should be chosen by analyzing usage and finding the most-used attributes

9 Database Applications

- An application is built on top of the database to store, retrieve, and manipulate data as necessary
  - Created in a general purpose programming language
  - DBMS is a back-end which stores the data on a server
  - May include:
    - Interface and presentation to provide access to data
    - Business logic to ensure data consistency
      - Constraints which cannot be easily implemented in the DBMS
    - Processing and computation of data
- Application architecture:
  - *Single-tier architecture:*
    - DBMS, application logic, business rules, and user interface in the same tier
    - Run on a mainframe; accessed through dumb terminals
      - *Dumb terminal:* Terminal which lacks the computational power to support a GUI
  - *Two-tier architecture:* Used for client-server architecture
    - Consist of client and server computers
      - *Thin client:* Client which implements the UI
        - Commonly used
      - *Thick client:* Client which implements the UI and some of the business logic
        - Uncommonly used
        - More difficult to update and maintain the business logic
        - Do not scale well
        -
      - Server implements the business logic and data management; clients are not responsible for data processing
  - *Three/Multi-tier architecture:* Used for web systems
    - Scalable
    - Presentation layer handles the user interaction with the middle? tier
    - Differnent interfaces...
    - *Style sheet:* Method to format a document in different ways depending on the context (e.g. mobile devices)
      - Display format can be standardized with the same conventions
    - *Business logic layer:* Runs the business logic of the application
      - Controls:
        -
    - *Data management layer:* Contains the database(s)

· Data is exchanged between the middle tier and the database servers if multiple data sources are used

▪ XML is often used as a data exchange format between database servers and the middle tier

* State of a user's interaction must be maintained across different web pages

· Maintained at the middle tier in...

* Advantages:

· *Heterogeneous system:*

▪ Applications can use different platforms and software components at different tiers

▪ Ease of modifying code at any specific tier

·

▪ Database must be connected to the host language (the programming language that the application is written in)

– Database connections allows communiation between a database server and client software

–

– Accessing database data from a host language application:

* *Embedded SQL:* SQL statements identified at compile time

· Does not allow the creation of new SQL queries

· SQL statements are marked so the preprocessor can process them

· Variables used to pass arguments into a SQL command are declared in SQL

▪ Data types may not match

▪ E.g. C:

```
EXEC SQL BEGIN DECLARE SECTION;
long sin;
char[20] fname;
char[20] lname;
int age;
EXEC SQL END DECLARE SECTION;

EXEC SQL
INSERT INTO Customer VALUES
        (:sin, :fname, :lname, :age);
```

· SQLSTATE: Array of 5 characters which connects the host language to...

· *Cursor:* Position in a relation which allows retrieval of a set of records

* *Dynamic SQL:* SQL queries constructed at runtime

· E.g. Open DataBase Connectivity, Java DataBase Connectivity - Protocols allownig a single executable to be run on different databases without...

· New queries can be created

· Query is created as a string, then parsed and executed

* Mapping a database table to a host language construct

- E.g. CRecordSet

*

10 Object Relational Mapping

- *Object relational mapping:* Process of converting data between an object/class and attributes in a database
- *Impedance mismatch:* System where inputs and outputs do not match
- Tables are mapped to classes; columns are mapped to member variables
- Inheritance options:
  - Table for each class
  - Table for each concrete class
  - Table for each class family
- Mapping a subclass to a table:
  - If each class in an inheritance
- Complex data types are composed of composite/multi-valued attributes

10.1 Injection

- *SQL injection:* Code injection attack which places unwanted code into an SQL query
- E.g. Given the following query in response to entered data:

```
SELECT attribute
FROM table
WHERE attribute = '$email' -- Variable from the form
```

  - If the input is:

```
X'; DROP TABLE tablename --
```

    then the entire table would be deleted.

  - If the input is:

```
anything' OR 'X' = 'X
```

    then the application may display an email in the table (often the first one).

  - If the input is:

```
X'; UPDATE members SET email = 'myemail@gmail.com' WHERE email = '
    someonesemail@gmail.com' --
```

    then the email for an account will be changed to another email and access to the account will be compromised.

- Preventing injection attacks:
  - *Sanitized input:* Data which has no invalid characters

    - Difficult due to exceptions such as apostrophes and hyphens in names
- *Bound input:* Data which is given to the code through placeholders
    - Can be done by the *Prepare* statement
- Limit permissions and segregate users
- Use stored procedures

11 NoSQL

11.1 Introduction

- For a comparison with SQL, see subsection 1.5
- Possible definitions:
  - No SQL
  - Not only SQL
- No schema
  - Key-value store allows any data to be associated with the key
  - No restrictions on what is contained
  - Column family databases allow any type of data in any column
- Common characteristics:
  - Does not use relational model
  - Runs well on clusters
  - Built for modern web applications
  - Lack a schema
  - Open source
- *Aggregate:* Collection of related units which is used like a single unit
  - Allows easier operation in clusters (see
  - Used as a unit for replication and sharding
  - Designed with respect to how the data is manipulated
  - Data related to a single aggregate should be held at a single cluster
  - Examples:
    - A bank Branch object containing a container of associated accounts
    - A bank Account object is independent of a Branch, and contains a reference to it
- Possible properties:
  - BASE:
    - Basically Available
    - Soft state
    - Eventually consistent
      - Allow temporary inconsistency until all nodes sync information
  - Do not follow the ACID properties
  - See Issues

11.2 Distribution of Data

- Stored data must be distributed across the cluster
- *Sharding:* Distribution of data across several nodes

  - Users should be able to retrieve most or all their relevant data from a single server
  - Data can be sharded by:
    * Physical location (e.g. Vancouver servers)
    * Aggregates which are likely to be accessed together, or in sequence from the same location
  - Often automatically done by the database
  - Advantages:
    * Improved read/write performance
  - Disadvantages:
    * Decreased resilience (higher chance of failure due to many machines)
- *Replication:* Maintenance of multiple copies of data
  - Advantages:
    * Improved resilience and availability
  - Disadvantages:
    * Replication schemas handle writes in different ways
    * May create inconsistency
  - *Master-slave replication:* Replication where a single copy is maintained as the 'definitive' copy
    * Updates are performed on the master copy and sent to the slaves
    * Reads are performed on the slaves
    * If the master fails, one of the slaves becomes the new master
    * Advantages:
      · G
      · Master is a single point of failure
    * Disadvantages:
      · Does not improve write readability
  - *Peer-to-peer replication:*
    *
- Combination of sharding and replication:
  - Each shard is replicated
  - Master-slave replication: One master for each shard
  - Peer-to-peer replication:
    * Commonly used for column family databases

11.3 Distributed Concurrency

-
- Concurrency is complex on a distributed system
- When any node can process updates:

  - Conflicts must be detected across notes
  - Updates are processed in the same order
- Aggregate-oriented databases support atomic transactions within aggregates, but not necessarily across aggregates
  - *Inconsistency window:* Period of time when reads may be inconsistent due to updates affecting multiple aggregates
-
- Distributed systems must have partition tolerance

11.4 Issues

- Consistency/availability/:
  - Given a peer-to-peer system

11.5 Categories

- *Key-value store:* Database which stores only one key and one value per row
  - Value is not usable except for direct storage/extraction (stored as binary)
  - Structurally simple
  - Uses:
    * Session information
    * User profiles and preferences
    * Shopping carts
  - Consistency is...
  - Transactions vary between products/implementations
  - Queries are done on the key only
  - Scaling is done by sharding by the key's value
- *Document database:* Database which stores only documents
  - *Document:* Self-describing hierarchical tree structure
    * E.g. XML, JSON
  - Differences compared to a key-value store:
    * Value can only be a document
    * Document can be compared/examined
  - Uses master-slave replication
  - Level of consistency can be specified (i.e. manual specification of the number of nodes to which an update must be propagated before it is considered successful)
  - Availability is improved through duplicated data across nodes
    * Master-slave replication allows
  - Scaling:

        * For reads: Adding more slaves
        * For writes: Sharding
    - Uses:
        * Event logging
        * Content management systems
        * Web analytics
    - E.g. MongoDB, CouchDB
- *Column-family store:* Database which groups column families and is a refined columnar database
    - *Columnar database:* Database which stores each column separately
        * Increased speed of aggregate operations on column data (e.g.

COUNT( )

)
        * Decreased speed of access on an entire row
        * Similar to y how the table is reduced/broken up
    - Multiple related columns:
    - Uses peer-to-peer replication
    - Level of consistency can be specified (i.e. manual specification of the number of nodes which must respond to an update before it is committed)
    -
    - Queries are not as rich as SQL but are similar
    - Uses:
        * Event logging
        * Content management systems
        * Blogging sites
        * Counters
    - E.g. Cassandra, HBase, Amazon SimpleDB
- *Graph database:* Database which stores relationships between objects
    - Nodes map to entities
    - Edges:
        * Map to relationships
        * Can have types
        * Can be bidirectional
    - Traversing relationships is efficient
    - Uses:
        * Connected data such as social networks
        * Location-based services
        * Recommendation engines

- E.g. Neo4j, Titan, OrientDB

12 Data Warehousing and Mining

- Traditional DBMSes are used to main data to record day-to-day operations
  - Online Transaction Processing (OLTP):
- Content and historical data is analyzed to identify patterns
  - Uses:
    - Market research to target products to a particular market segment
    - Decision support for high-level decision-making
  - OLAP databases:

12.1 Data Warehouses

- *Data warehouse*: Consolidation of data from multiple databases in order for analysis of global patterns
  - Allows high-level decision making covering many aspects of an organization
  - Currency of data is not a priority
- Data is:
  - Extracted from databases and other sources (heterogeneous sources)
  - Cleaned to minimize errors and missing information
  - Transformed to match semantics (semantic integration)
  - Loaded into the warehouse (metadata management - data such as load date, source, etc.)
  - Indexed for efficiency
  - Summarized
- Issues:
  -
-