C7: FDs & Normalization

Introduction

- Normalization helps DB designers determine the best relation schemas
  - A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
  - A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree
- It is based on the concept of normal form 1NF, 2NF, 3NF, BCNF, 4NF, 5NF
- It is a process which ensures that the data is structured in such a way that attributes are grouped with the PK. Attributes that do not directly depend on PK may be extracted to form a new relation

There are two important properties of decompositions:

1) non-additive or losslessness of the corresponding join
2) preservation of the functional dependencies

Note that property (1) is extremely important and cannot be sacrificed. Property (2) is less stringent and may be sacrificed

Armstrong's inference rules:

- **IR1. (Reflexive)** If $Y \subseteq X$, then $X \rightarrow Y$
- **IR2. (Augmentation)** If $X \rightarrow Y$, then $XZ \rightarrow YZ$
  (Notation: XZ stands for $X \cup Z$)
- **IR3. (Transitive)** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Decomposition: If X -> YZ, then X -> Y and X -> Z
Union: If X -> Y and X -> Z, then X -> YZ
Psuedotransitivity: If X -> Y and WY -> Z, then WX -> Z

A set of FDs is minimal if it satisfies the following conditions:

- Every dependency in F has a single attribute for its RHS.
- We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.
- We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y proper-subset-of X (Y subset-of X) and still have a set of dependencies that is equivalent to F

Normalization

- **Normalization**: The process of **decomposing** unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- Normal form: Using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The database designers **need not** normalize to the highest possible normal form (3NF, BCNF or 4NF)

First normal form (1NF): there is only one value at the intersection of each row and column of a relation - no set valued attributes in 1NF

→ Disallows composite attributes, multivalued attributes, and **nested relations**

To be part of the formal definition of a relation in the basic (flat) relational model

- Second normal form (2NF) - all attributes must be fully functionally dependent on the primary key
- 2NF solves partial dependency problem in 1NF
- Method: identify primary keys and group attributes that relate to the key together to form separate new relations

- A relation schema R is in **third normal form** (**3NF**) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key

NOTE:

In $X \rightarrow Y$ and $Y \rightarrow Z$, with X as the primary key, we consider this a problem only if Y is not a candidate key. When Y is a candidate key, there is no problem with the transitive dependency .

E.g., Consider EMP (SSN, Emp#, Salary).

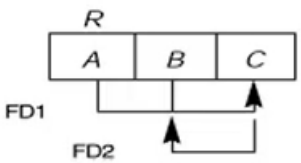
Here, SSN → Emp# →Salary and Emp# is a candidate key

3NF solves indirect (transitive) dependencies problem in 1NF and 2NF

Method: identify all transitive dependencies and each transitive dependency will form a new relation, with non-prime attributes participating in the transitive dependency and the attribute which determines others as the attributes for the new relation

General Normal Form Definitions

- A relation schema R is in **second normal form** (**2NF**) if every non-prime attribute A in R is fully functionally dependent on *every key* of R
- A relation schema R is in **third normal form** (**3NF**) if whenever a FD X -> A holds in R, then either:
  - (a) X is a superkey of R, or
  - (b) A is a prime attribute of R



: vi phạm BCNF

Normalization

- A relation schema R is in **Boyce-Codd Normal Form** (**BCNF**) if whenever an FD X -> A holds in R, then X is a superkey of R

Algorithm 16.5. Relational Decomposition into BCNF with Nonadditive Join Property

Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.

1. Set $D := \{R\}$;

2. While there is a relation schema *Q* in *D* that is not in BCNF do

{

choose a relation schema *Q* in *D* that is not in BCNF;

find a functional dependency $X \rightarrow Y$ in *Q* that violates BCNF;

replace *Q* in *D* by two relation schemas $(Q - Y)$ and $(X \cup Y)$;

} ;

Key-finding algorithm (1)

By Elmasri and Navathe

Input: *A relation R and a set of functional dependencies F on the attributes of R.*

Output: *a key K of R*

1. Set K to contain all attributes in R
2. For each attribute A in K {

   compute $(K - A)^+$ with respect to F;

   if $(K - A)^+$ contains all attributes in R, then

   set K := K – {A}

   };

Key-finding algorithm (2)

By Hossein Saiedian & Thomas Spencer

Input: *A relation R and a set of functional dependencies F on the attributes of R.*

Output: *all candidate keys of R*

Let:

- U contain **all** attributes of R
- U_l contain attributes of R that occur **only** on the **left-hand side** of FDs in F
- U_r contain attributes of R that occur **only** on the **right-hand side** of FDs in F
- U_b contain attributes of R that occur on **both sides** of FDs in F

Note:

- $U_l \cap U_r = \phi$, $U_l \cap U_b = \phi$ and $U_r \cap U_b = \phi$
- $U_l \cup U_r \cup U_b = U$
- For every attribute $A \in U$, if $A \in U_l$, then A **must be** part of every candidate key of **R**.
- For every attribute $A \in U$, if $A \in U_r$, then A will **not** be part of any candidate key of **R.**

Key-finding algorithm (2)

By Hossein Saiedian & Thomas Spencer

Input: *A relation R and a set of functional dependencies F on the attributes of R.*

Output: *all candidate keys of R*

1. Determine U_l, U_r and U_b
2. If $U_l^+ = U$ under F, then U_l forms the only key of R and the algorithm stops here.

   Else: move to step 3 // $U_l^+ \neq U$ under F
3. Consider every subsets U_{bi} of U_b: $U_{bi} \subset U_b$

   For each U_{bi}, if $(U_l \cup U_{bi})^+ = U$ under F, then $K_i = (U_l \cup U_{bi})$ is a candidate key of R (*)

(*) *If $K_i = (U_l \cup U_{bi})$ is a candidate key of R, then we need not to check $U_{bj} \subset U_b$ where $U_{bi} \subset U_{bj}$*

C8: Data storage & Indexing Structures for Files

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries **<field value, pointer to record>**, which is ordered by field value
- The index is called an access path on the field.

The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller.

A *binary search* on the index yields a pointer to the file record.

Indexes can also be characterized as dense or sparse:

- A **dense index** has an index entry for every search key value (and hence every record) in the data file.
- A **sparse (or nondense) index**, on the other hand, has index entries for only some of the search values

Primary Index

- Defined on an **ordered data file.**
  - The data file is ordered on a ***key field***.
- One index entry *for each block* in the data file
  - *First record* in the block, which is called the *block anchor*
- A similar scheme can use the *last record* in a block.

- Number of index entries?
  - Number of blocks in data file.
- Dense or Nondense?
  - Nondense

Clustering Index

- Number of index entries?
  - Number of distinct indexing field values in data file.
- Dense or Nondense?
  - Nondense
- Search/ Insert/ Update/ Delete?
- At most **one primary index or one clustering index but not both.**

Clustering Index

- Defined on an **ordered data file.**
  - The data file is ordered on a *non-key field*.
- One index entry *each distinct value* of the field.
  - The index entry points to the *first data block* that contains records with that field value

Secondary index

- A secondary index provides a secondary means of accessing a file.
  - The data file is unordered on indexing field.
- Indexing field:
  - secondary key (unique value)
  - nonkey (duplicate values)
- The index is an ordered file with two fields.
  - The first field: *indexing field.*
  - The second field: *block* pointer or *record* pointer.
- There can be ***many*** secondary indexes for the same file.

Secondary index on key field

- Number of index entries?
  - Number of record in data file
- Dense or Nondense?
  - Dense
- Search/ Insert/ Update/ Delete?

Given the following data file: **EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)**

Suppose that:

- record size **R** = 150 bytes
- block size **B** = 512 bytes
- Number of records **r** = 30.000 records
- SSN Field size $\mathbf{V_{SSN}}$ = 9 bytes,
- record pointer size $\mathbf{P_R}$ = 7 bytes
- block pointer size $\mathbf{P_B}$ = 6 bytes

Then, we get:

- blocking factor: ***bfr*** = $\lfloor B/R \rfloor = \lfloor 512/150 \rfloor$ = 3 records/block
- number of blocks needed for the file: **b** = $\lceil r/bfr \rceil = \lceil 30.000/3 \rceil$ = 10.000 blocks

For a dense index on the SSN field:

- index entry size: $\mathbf{R_i} = (V_{SSN} + P_R) = (9+7) = 16$ bytes
- index blocking factor $\boldsymbol{bfr_i} = \lfloor B/R_I \rfloor = \lfloor 512/16 \rfloor$ = 32 entries/block
- number of blocks for index file: $\mathbf{b_i} = \lceil r/bfr_i \rceil = (30000/32) = 938$ blocks
- binary search to retrieve a record in index file needs: $\lceil \log_2 b_i \rceil = \lceil \log_2 938 \rceil$ = 10 block accesses
- To search for a record using the index, we need one additional block access to the data file for a total of 10 + 1 = **11** block accesses

This is compared to an average linear search cost of:

(b/2)= 10000/2 = **5000** block accesses

If the file records are ordered, the binary search cost would be:

$\lceil \log_2 b \rceil = \lceil \log_2 10000 \rceil$ = **14** block accesses

For a primary index on the **ordering key field** SSN : (using block pointer)

- index entry size: $\mathbf{R_i} = (V_{SSN} + P_B) = (9+6) = 15$ bytes
- index blocking factor $\boldsymbol{bfr_i} = \lfloor B/R_I \rfloor = \lfloor 512/15 \rfloor$ = 34 entries/block
- number of blocks for index file: $\mathbf{b_i} = \lceil r/bfr_i \rceil = (10000/34) = 295$ blocks
- binary search to retrieve a record in index file needs: $\lceil \log_2 b_i \rceil = \lceil \log_2 295 \rceil$ = 9 block accesses
- To search for a record using the index, we need one additional block access to the data file for a total of 9 + 1 = **10** block accesses

This is compared to an average linear search cost of:

(b/2)= 10000/2 = **5000** block accesses

If the file records are ordered, the binary search cost would be:

$\lceil \log_2 b \rceil = \lceil \log_2 10000 \rceil$ = **14** block accesses

Secondary index on non-key field

- **Discussion:** Structure of Secondary index on non-key field?
- Option 1: include **duplicate index entries** with the same $K(i)$ value - one for each record.
- Option 2: keep a **list of pointers** <$P(i, 1)$, ..., $P(i, k)$> in the index entry for $K(i)$.
- Option 3:
  - more commonly used.
  - one entry for each *distinct index field value* + *an* **extra level of indirection** to handle the multiple pointers.

Summary of Single-level indexes

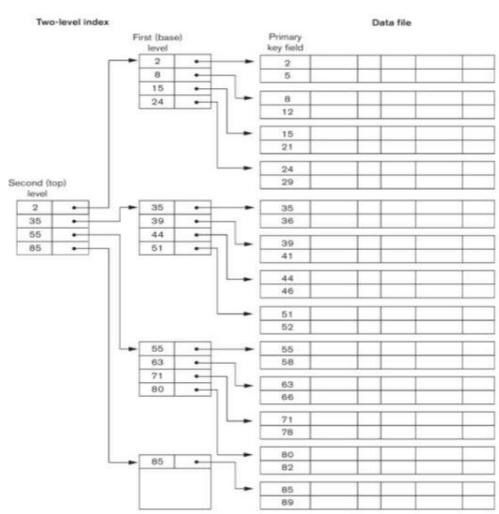
Table 18.2 Properties of Index Types

Type of Index	Number of (First-level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

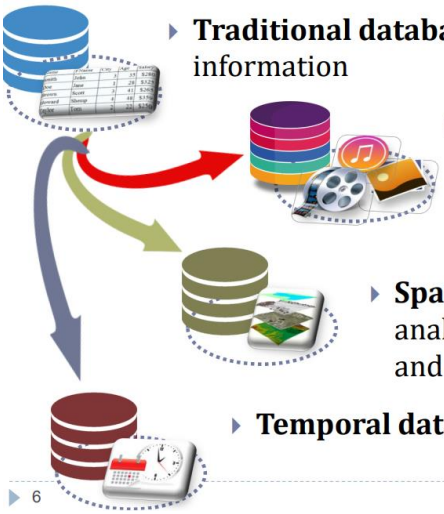
Multi-Level Indexes

- Because a single-level index is an ordered file, we can **create a primary index *to the index itself*.**
  - The original index file is called the *first-level index* and the index to the index is called the *second-level index.*
- We can repeat the process, creating a third, fourth, ..., top level **until all entries of the *top level* fit in one disk block.**
- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block.



A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.

C1: Concepts and Architecture



- **Traditional database:** textual or numeric information
- **Multimedia database:** images, audio clips, and video streams
- **Spatial database:** Store and analyze maps, weather data, and satellite images
- **Temporal database:** historical data

File-based Approach

- **Problems:**
  - Data Redundancy
    - The same information being kept in several different places (files)
    - Wastes storage space and duplicates effort
  - Data Inconsistency
    - Various copies of the same data are conflicting
    - Inconsistency in data format

Shared File Approach

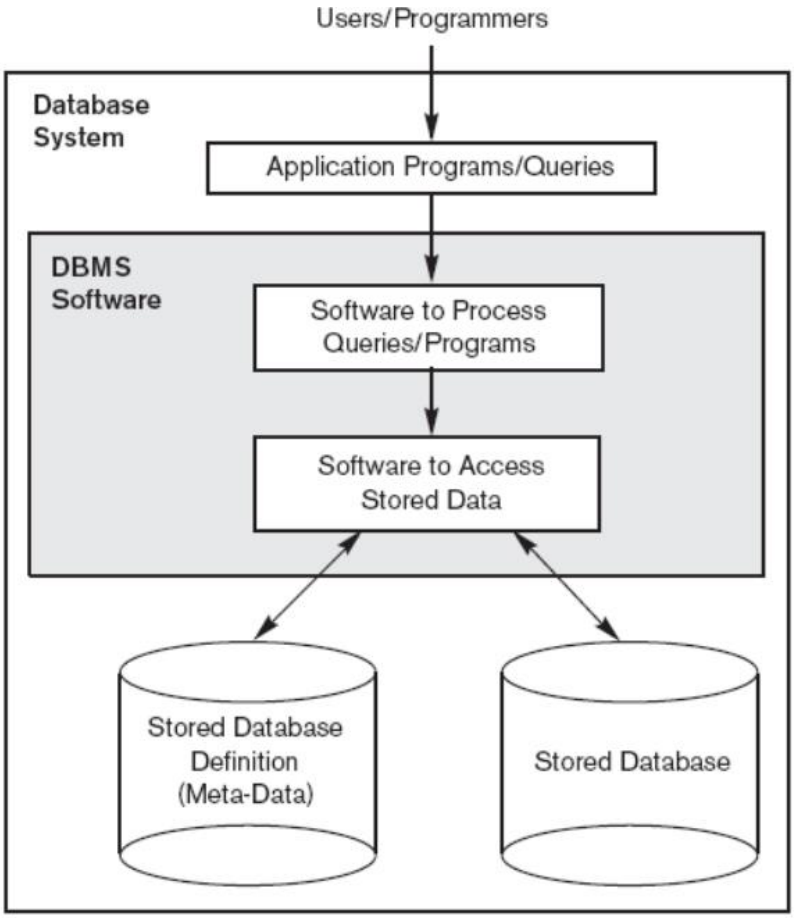
- Other problems:
  - Rigid data structure: If applications have to share files, the file structure that suits one application might not suit another
  - Physical data dependency: If the structure of the data file needs to be changed in some way, this alteration will need to be reflected in all application programs that use that data file
  - No support of concurrency control: While a data file is being processed by one application, the file will not be available for other applications or for ad hoc queries

Arose because:

- Definition of data was embedded in application programs, rather than being stored separately and independently
- No control over access and manipulation of data beyond that imposed by application programs

Result:

- The Database and Database Management System (DBMS).



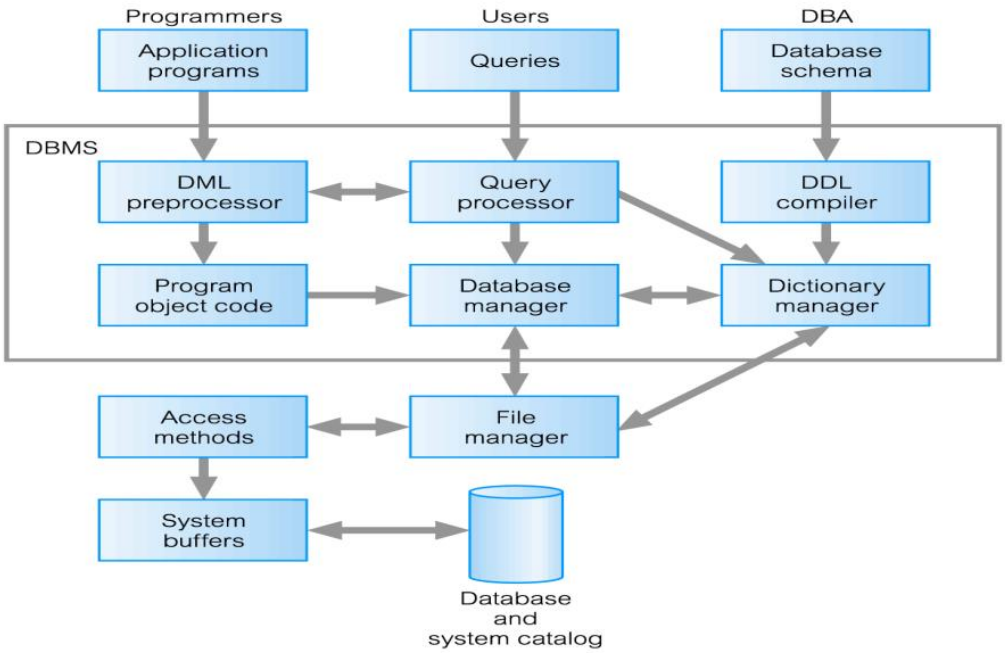
Database System = Database + DBMS software

Database approach allows user:

- Specify data types, structures and any data constraints to be stored in the database. All specifications are stored in the database
- Query data: retrieve (query), update (insert, delete, modify)
- Control access to database:
  - a security system
  - an integrity system
  - a concurrency control system
  - a recovery control system
  - a user-accessible catalog

- **Database administrators (DBA)** are responsible for:
  - Authorizing access to the database
  - Coordinating and monitoring its use
  - Acquiring software and hardware resources
- **Database designers** are responsible for:
  - Identifying the data to be stored
  - Choosing appropriate structures to represent and store this data
- **Application programmers**
  - Implement these specifications as programs
- **End users**
  - People whose jobs require access to the database

DBMS components:



Characteristics of the Database Approach

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
  - Program-data independence + Program-operation independence = Data abstraction
  - A data model is a type of data abstraction
- Support of multiple views of the data
- Sharing of data and multi-user transaction processing

More desirable to use regular files for:

- Simple, well-defined database applications not expected to change at all
- Stringent, real-time requirements that may not be met because of DBMS overhead
- Embedded systems with limited storage capacity
- No multiple-user access to data

Objectives of Three-Schema Architecture

- All users should be able to access same data
- Users should not need to know physical database storage details
- DBA should be able to change database storage structures without affecting the users' views
- Internal structure of database should be unaffected by changes to physical aspects of storage
- DBA should be able to change conceptual structure of database without affecting all users

External Level

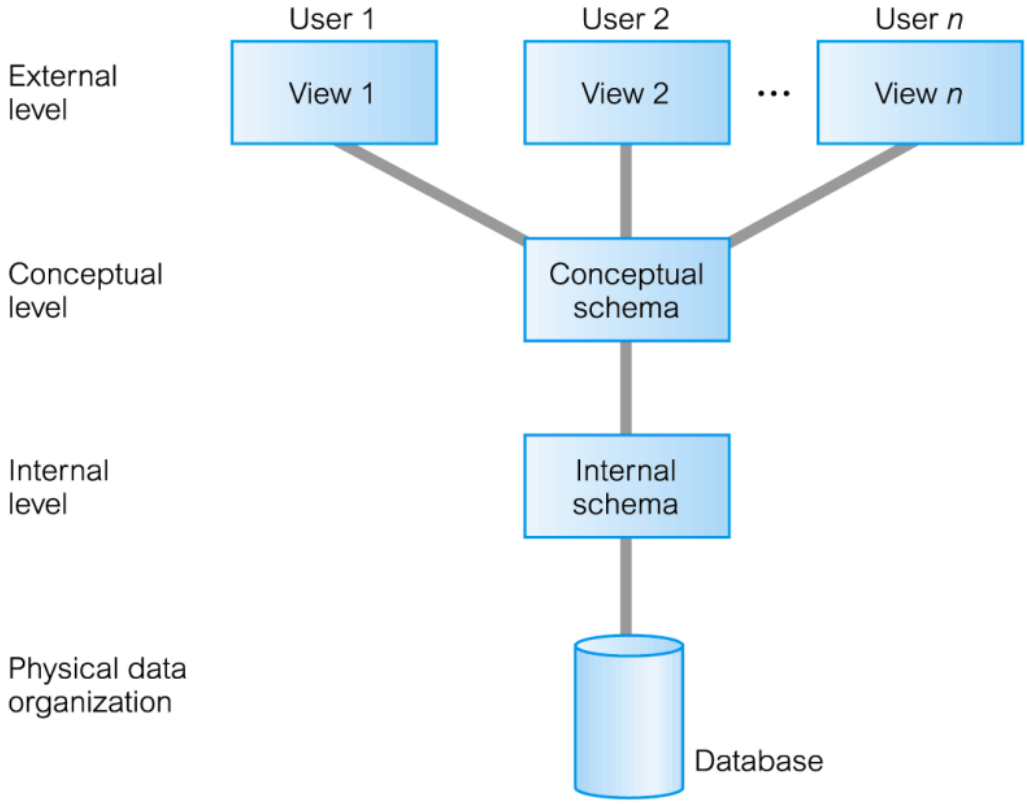
- Users' view of the database
- Describes that part of database that is relevant to a particular user

Conceptual Level

- Community view of the database
- Describes what data is stored in database and relationships among the data

Internal Level

- Physical representation of the database on the computer.
- Describes how the data is stored in the database



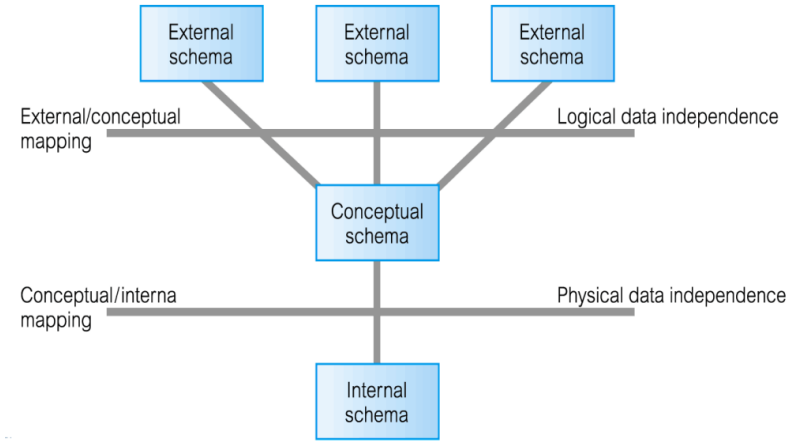
Data Independence: is the capacity to change the schema at one level of a database system without having to change the schema at the next higher levels

Logical Data Independence:

- Conceptual schema changes (e.g. addition/removal of entities) should not require changes to external schema or rewrites of application programs

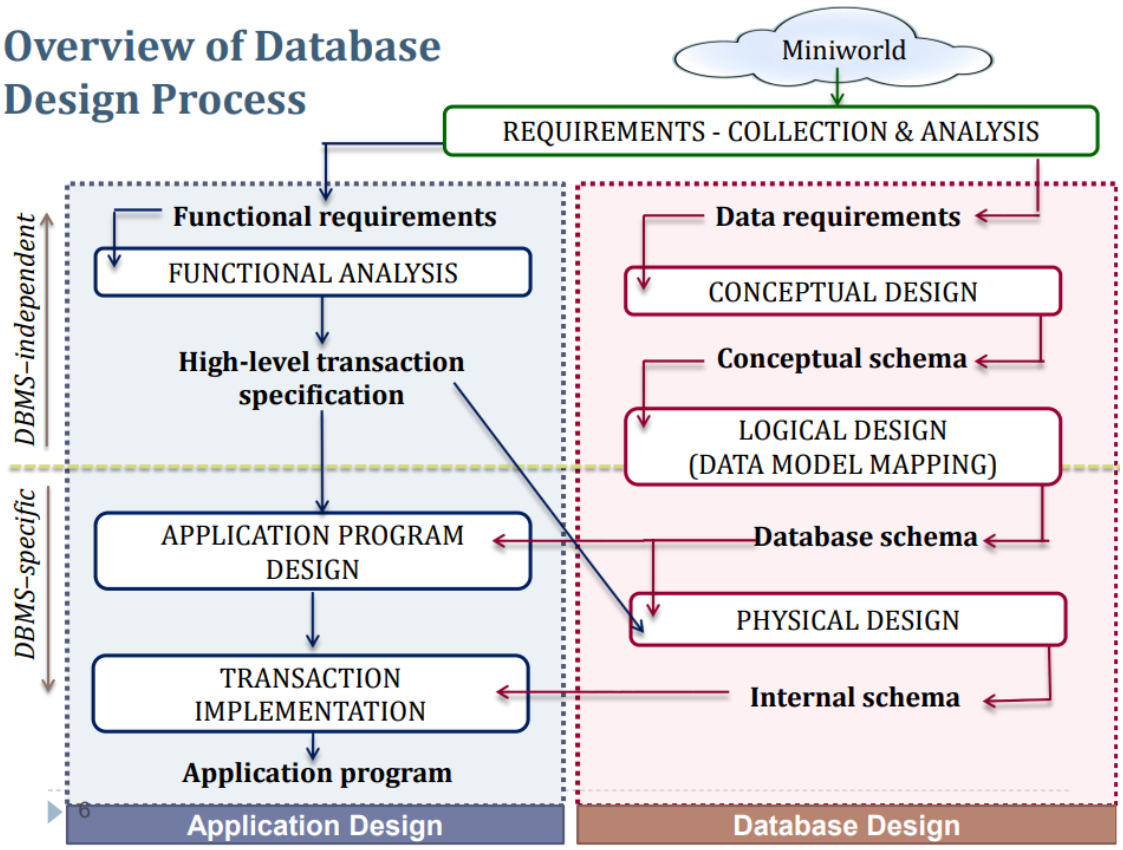
Physical Data Independence:

- Internal schema changes (e.g. using different file organizations, storage structures/devices) should not require changes to conceptual or external schemas



C2: ERD

Overview of Database Design Process



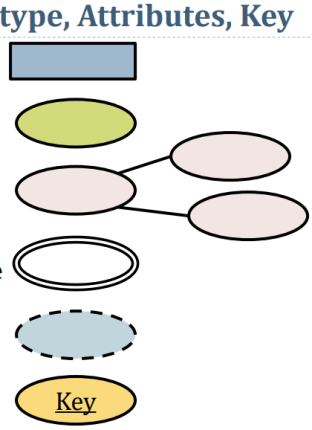
Types of Attributes

- *Simple attribute* has a single atomic value.
  - SSN, Sex
- *Composite attribute* may be composed of several components.
  - Name (First name, Middle name, Last name)
- *Multi-valued attribute* has multiple values.
  - Colors of a Car {Color}, Phones of a Person {Phone}
- *Derived attribute* has a value that is derivable from values of related attributes.
  - Number of students in a class
- *Complex attribute* is a combination of composite and multivalued attributes.

- **Key or uniqueness constraint**
  - Attributes whose values are **distinct** for each individual entity in entity set
  - Uniqueness property must hold for every entity set of the entity type
  - Ex: SSN of EMPLOYEE
- An entity type may have **more than one key**.
  - Ex: the STUDENT entity type may have two keys (in university context):
    - Citizen ID and
    - Student ID

Notations of Entity type, Attributes, Key

- Entity type
- Simple attribute
- Composite attribute
- Multi-valued attribute
- Derived attribute
- Key



- **Relationship type *R*** among n entity types $E_1, E_2, ..., E_n$
  - Defines a set of associations among entities from these entity types
  - Ex: Relationship type WORKS_FOR between EMPLOYEEs and DEPARTMENTs
- **Relationship instances r_i**
  - Each r_i associates n individual entities $(e_1, e_2, ..., e_n)$. Each entity e_j in r_i is a member of entity set E_j
  - Ex: EMPLOYEE John Smith <u>*works on*</u> the PROJECT ProductX

- **Degree** of a relationship type
  - Number of participating entity types
  - Binary (degree 2), ternary (degree 3), and n-ary (degree n)
- More than one relationship type can exist with the same participating entity types.
  - EMPLOYEE – *Works-for* – DEPARTMENT
  - EMPLOYEE - *Manages* – DEPARTMENT

Recursive relationships

- Same entity type participates more than once in a relationship type in different roles
- Must specify *role* that a participating entity plays in each relationship instance
- Ex: SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker)

Structural constraints: one way to express semantics of relationship: *cardinality ratio* and *participation constraint.*

Cardinality ratio: specifies maximum number of relationship instances that entity can participate in a binary relationship.

- one-to-one (1:1)
- one-to-many (1:M) or many-to-one (M:1)
- many-to-many (M:N)

Participation constraint (or membership class): specifies whether existence of entity depends on its being related to another entity

- *Mandatory* (total participation) - every instance of a participating entity type must participate in the relationship. (double line)
- *Optional* (partial participation) - not every instance of a participating entity type must participate in the relationship. (single line)

Do not have key attributes of their own

- Identified by being related to specific entities from another entity type

Identifying relationship: Relates a weak entity type to its owner

Always has a total participation constraint

Entities are identified by the combination of:

- **A partial key** of the weak entity type
- The particular entity they are related to in the identifying entity type

- **(Min-max) notation for relationships**
  - Specify structural constraints on relationships
  - Replaces cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints
  - Associate a pair of integer numbers (min, max) with each participation of an entity type E in a relationship type R, where $0 \leq \min \leq \max$ and $\max \geq 1$

- Constraints that cannot be directly expressed in the ER diagram
- Must be expressed and enforced by the trigger mechanism, or application programs or in some other ways.
- Examples:
  - The age of an employee must be greater than 18 years old
  - The salary of a department manager must be higher than the other employees works for that department.
  - When increasing salary of employee, the increasing amount must not more than 20% of current salary.

Connection traps

- **Fan Trap**
  - Where a model represents a relationship between entity types, but pathway between certain entity occurrences is ambiguous
  - Usually: two or more 1:N relationships fan out from the same entity
- **Chasm Trap**
  - Where a model suggests the existence of a relationship between entity types, but pathway does not exist between certain entity occurrences
  - Usually: optional participation

C3: EERD

Subclasses, Superclasses & Inheritance

- **Subtype or subclass** of an entity type
  - Subgroupings of entities that are meaningful
  - Represented explicitly because of their significance to the database application
- Terms for relationship between a superclass and any one of its subclasses
  - Superclass/subclass
  - Supertype/subtype
  - Class/subclass
- A class/subclass relationship is often called an **IS-A** (IS-AN) relationship.

Subclasses, Superclasses & Inheritance

- A subclass member is the same as the entity in the superclass, but in a distinct specific role
- An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass
- A member of a superclass can be optionally included as a member of some of its subclasses
- It is not necessary that every entity in a superclass be a member of some subclass
- Superclass/subclass relationship is one-to-one (1:1)

Subclasses, Superclasses & Inheritance

- Subclass entity **inherits all attributes and relationships** of superclass
- Notice that a subclass, with its own specific (or local) attributes and relationships together with all the attributes and relationships it inherits from the superclass, can be considered an entity type in its own right.

Specialization and Generalization

- **Specialization**
  - Process of defining a set of subclasses of an entity type, called superclass
  - Defined on the basis of some distinguishing characteristic of the entities in the superclass
  - May have several specializations of the same entity type based on different distinguishing characteristics
- Subclass can have its own:
  - Specific attributes (local attributes)
  - Specific relationship types

Generalization

- Reverse process of Specialization
- **Generalize** several entity types which have some common features into a single **superclass**
  - Original entity types are special subclasses
- **Generalization**
  - Process of defining a generalized entity type from the given entity types

Specialization and Generalization

- Diagrammatic notation sometimes used to distinguish between generalization and specialization
  - Arrow pointing to the generalized superclass represents a generalization
  - Arrows pointing to the specialized subclasses represent a specialization
  - We do not use this notation because it is often subjective as to which process is more appropriate for a particular situation
  - We advocate not drawing any arrows in these situations

Constraints and Characteristics of Specialization and Generalization Hierarchies

- Constraints that apply to a single specialization or a single generalization
- Differences between specialization/ generalization lattices and hierarchies

Constraints on Specialization & Generalization

- Determine subclass:
  - Predicate-defined (or condition-defined) subclasses
  - Attribute-defined specialization
  - User-defined

- Two basic conditions apply to a specialization/ generalization: **disjointness** and **completeness** constraints
- **Disjointness constraint**
  - May be **disjointed** or **overlap**
- **Completeness** (or **totalness**) **constraint**
  - May be **total** or **partial**
- Disjointness and completeness constraints are independent

Disjointness Constraint:

- Specifies that the subclasses of the specialization must be disjointed (an entity can be a member of at most one of the subclasses of the specialization)
- Specified by **d** in EER diagram
- If not disjointed, overlap; that is the same entity may be a member of more than one subclass of the specialization
- Specified by **o** in EER diagram

Completeness Constraint:

- Total specifies that every entity in the superclass must be a member of some subclass in the specialization/ generalization: Shown in EER diagrams by a **double line**
- Partial allows an entity not to belong to any of the subclasses: Shown in EER diagrams by a **single line**

Hence, we have four types of specialization / generalization:

- Disjoint, total
- Disjoint, partial
- Overlapping, total
- Overlapping, partial

Note: Generalization is usually total because the superclass is derived from the subclasses

- **Category** or **Union type**
  - Represents a single superclass/subclass relationship with more than one superclass
  - Subclass represents a collection of objects that is a subset of the UNION of distinct entity types
  - Attribute inheritance works more selectively
  - Category can be **total** or **partial**
- Some modeling methodologies do not have union types

- A category can be total or partial
  - **Total**
    - Hold the union of all entities in its superclasses.
    - Represented diagrammatically by a *double line* connecting the category and the circle.
  - **Partial**
    - Can hold a subset of the union.
    - Represented diagrammatically by a *single line* connecting the category and the circle.
- The superclasses of a category may have different key attributes or the same key attribute.
  - E.g.:
    - Different key attributes: OWNER category.
    - The same key attribute: REGISTERED_VEHICLE category.

Category

A member entity of a category must exist in only one of its superclasses (**OR**).

- E.g.: An OWNER may be a COMPANY, a BANK, or a PERSON.

A category is a subset of the union of its superclasses.

- E.g.: OWNER is a subset of the union of the three superclasses.

Shared Subclass

An member entity of a shared subclass must exist in all its superclasses (**AND**).

- E.g.: An engineering manager must be an ENGINEER, a MANAGER, and a SALARIED_EMPLOYEE.

A shared subclass is a subset of the intersection of its superclasses.

- E.g.: ENGINEERING_MANAGER is a subset of the intersection of the three superclasses.

Category

A member of a category inherits attributes and relationships of only one of its superclasses.

- E.g.: Each OWNER entity inherits the attributes of a COMPANY, a PERSON, or a BANK, depending on the superclass to which the entity belongs.

Shared Subclass

A member of a shared subclass inherits all the attributes and relationships of all its superclasses.

- E.g.: ENGINEERING_MANAGER inherits all the attributes of its superclasses SALARIED_EMPLOYEE, ENGINEER, and MANAGER.

C4: Relational Data Model and ER-EER Mapping

- **Step 1: Mapping of Regular (strong) Entity Types**
  - Entity --> Relation
  - Attribute of entity --> Attribute of relation
  - Primary key of entity --> Primary key of relation
  - **Example:** We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram. SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown

- **Step 2: Mapping of Weak Entity Types**
  - For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R
  - In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s)
  - The primary key of R is the *combination of* the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any
  - **Example:** Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN)

    The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT
  - Note: CASCADE option as implemented

- **Mandatory** membership class
  - For two entity types E1 and E2: If E2 is a mandatory member of an N:1 (or 1:1) relationship with E1, then the relation for E2 will include the prime attributes of E1 as a foreign key to represent the relationship
  - 1:1 relationship: If the membership class for E1 and E2 are both mandatory, a foreign key can be used in either relation
  - N:1 relationship: If the membership class of E2, which is at the N-side of the relationship, is *optional* (i.e. partial), then the above guideline is not applicable

Optional membership classes

- If entity type E2 is an optional member of the N:1 relationship with entity type E1 (i.e. E2 is at the N-side of the relationship), then the relationship is **usually** represented by a new relation containing the prime attributes of E1 and E2, together with any attributes of the relationship. The key of the entity type at the N-side (i.e. E2) will become the key of the new relation
- If both entity types in a 1:1 relationship have the optional membership, a new relation is created which contains the prime attributes of both entity types, together with any attributes of the relationship. The prime attribute(s) of either entity type will be the key of the new relation

ER-to-Relational Mapping

- N:M binary relationships:
  - An N:M relationship is always represented by a new relation which consists of the prime attributes of both participating entity types together with any attributes of the relationship
  - The combination of the prime attributes will form the primary key of the new relation

ER-to-Relational Mapping

- Transformation of recursive/involuted relationships
  - Relationship among different instances of the same entity
  - The name(s) of the prime attribute(s) needs to be changed to reflect the role each entity plays in the relationship

ER-to-Relational Mapping

- Step 6: Mapping of Multivalued attributes
  - For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type or relationship type that has A as an attribute
  - The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components

  Example: The relation DEPT_LOCATIONS is created. The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation. The primary key of R is the combination of {DNUMBER, DLOCATION}

- Step 7: Mapping of N-ary Relationship Types
  - For each n-ary relationship type R, where n>2, create a new relationship S to represent R
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types
  - Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S

  Example: The relationship type SUPPY in the ER below. This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}

ER Model	Relational Model
Entity type	"Entity" relation
1:1 or 1:N relationship type	Foreign key (or "relationship" relation)
M:N relationship type	"Relationship" relation & 2 foreign keys
n-ary relationship type	"Relationship" relation & n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

- ER-
  - Step 1: Mapping of Regular Entity Types
  - Step 2: Mapping of Weak Entity Types
  - Step 3: Mapping of Binary 1:1 Relationship Types
  - Step 4: Mapping of Binary 1:N Relationship Types
  - Step 5: Mapping of Binary M:N Relationship Types
  - Step 6: Mapping of Multivalued attributes
  - Step 7: Mapping of N-ary Relationship Types
- EER-
  - Step 8: Options for Mapping Specialization or Generalization.
  - Step 9: Mapping of Union Types (Categories)

C5: Relational Algebra

- Unary Relational Operations:
  - SELECT (symbol: σ (sigma))
  - PROJECT (symbol: π (pi))
  - RENAME (symbol: ρ (rho))
- Relational Algebra Operations from Set Theory:
  - UNION (∪), INTERSECTION (∩), DIFFERENCE (or MINUS, –)
  - CARTESIAN PRODUCT (**x**)
- Binary Relational Operations:
  - JOIN (several variations of JOIN exist)
  - DIVISION
- Additional Relational Operations:
  - OUTER JOINS, OUTER UNION
  - AGGREGATE FUNCTIONS (SUM, COUNT, AVG, MIN, MAX)

- In general, the ***select*** operation is denoted by $\sigma_{<\text{selection condition}>}(R)$ where:
  - σ (sigma) is used to denote the ***select*** operator.
  - <selection condition> is a ***Boolean expression*** specified on the attributes of relation R.
  - Tuples that make the condition **true** appear in the result of the operation, and tuples that make the condition **false** are discarded from the result of the operation.

PROJECT Operation is denoted by **π (pi).**

This operation keeps certain *columns* (attributes) from a relation and discards the other columns.

- PROJECT creates a vertical partitioning: the list of specified columns (attributes) is kept in each tuple, the other attributes in each tuple are discarded.

Example: To list each *employee's first and last name and salary*, the following is used:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

- The **RENAME** operator is denoted by **ρ (rho).**

- In some cases, we may want to *rename* the attributes of a relation or the relation name or both.
  - Useful when a query requires multiple operations.
  - Necessary in some cases (see JOIN operation later).

Type Compatibility of operands is required for the binary set operation UNION ∪, (also for INTERSECTION ∩, SET DIFFERENCE –).

The resulting relation for R1∪R2 (also for R1∩R2, or R1–R2) has the ***same attribute names*** as the first operand relation R1 (by convention).

Operations of Relational Algebra

Table 6.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$

Operations of Relational Algebra

Table 6.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

- **Use of the Functional operator $\mathcal{F}$**
  - $\mathcal{F}_{\text{MAX } Salary}$ **(Employee)** retrieves the maximum salary value from the Employee relation
  - $\mathcal{F}_{\text{MIN } Salary}$ **(Employee)** retrieves the minimum Salary value from the Employee relation
  - $\mathcal{F}_{\text{SUM } Salary}$ **(Employee)** retrieves the sum of the Salary from the Employee relation
  - $_{\text{DNO}}\mathcal{F}_{COUNT\ SSN,\ AVERAGE\ Salary}$ **(Employee)** groups employees by DNO (department number) and computes the count of employees and average salary per department.
  - **Note: count just counts the number of rows, without removing duplicates.**
  - $\mathfrak{I}$

The **OUTER JOIN** Operation

- In NATURAL JOIN and EQUIJOIN, tuples without a *matching* (or *related*) tuple are eliminated from the join result.
  - Tuples with *null* in the join attributes are also eliminated.
  - This amounts to loss of information.
- A set of operations, called OUTER joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.

C6: SQL

Basic Data Types

- **Numeric data types**:
  - Integer numbers: INTEGER, INT, and SMALLINT
  - Floating-point (real) numbers: FLOAT or REAL, and DOUBLE PRECISION
- **Character-string data types**:
  - Fixed length: CHAR(n), CHARACTER(n)
  - Varying length: VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)
- **Bit-string data types**:
  - Fixed length: BIT(n)
  - Varying length: BIT VARYING(n)
- **Boolean data type**:
  - Values of TRUE or FALSE or NULL
- **Date-Time data types**:
  - Date components: YEAR, MONTH, and DAY ('YYYY-MM-DD')
  - Time components: HOUR, MINUTE, and SECOND ('HH:MM:SS')

CREATE DOMAIN *DomainName* AS *DataType*
[CHECK *conditions*];

CREATE DOMAIN SSN_TYPE **AS** CHAR(9);
CREATE DOMAIN D_NUM **AS** INTEGER
CHECK (D_NUM>0 AND D_NUM<21);

- **PRIMARY KEY** clause: specifies one or more attributes that make up the primary key of a relation.

Dnumber INT **PRIMARY KEY**
PRIMARY KEY (Dnumber, DLocation)

- **UNIQUE** clause: Specifies alternate (secondary) keys.

Dname VARCHAR(15) **UNIQUE**;

- **FOREIGN KEY** clause

FOREIGN KEY (*listOfFKColumns*)
REFERENCES *ParentTableName* [(*listOfCKColumns*)]
[ON UPDATE *referentialAction*]
[ON DELETE *referentialAction*]

- Referential triggered actions: RESTRICT (default), SET NULL, CASCADE, and SET DEFAULT

FOREIGN KEY Dno **REFERENCES** Department(Dnumber)
ON DELETE CASCADE
ON UPDATE CASCADE

DROP Command

- Used to drop named schema elements: tables, domains, constraints, and the schema itself
- Drop behavior options:
  - CASCADE and RESTRICT

DROP SCHEMA Company **CASCADE**;
Or
DROP SCHEMA Company **RESTRICT**;

DROP Command

- Drop a table:

DROP TABLE Department **CASCADE**;

  - RESTRICT (default): dropped on if it is not referenced in any constraints or views
  - CASCADE: all such constraints and views that reference the table are dropped automatically from the schema along with the table itself
- Similarly, we can drop constraints & domains

ALTER Command

- ALTER command: change the definition of a base table or of other named schema elements
- Base tables: adding or dropping a column or constraints, changing a column definition.

ALTER TABLE Employee **ADD** Job VARCHAR(15);

ALTER TABLE Employee
  DROP COLUMN Address CASCADE;

ALTER TABLE Department
  ALTER COLUMN Mgr_ssn SET DEFAULT '333445555';

ALTER TABLE Employee
  DROP CONSTRAINT Empsuperfk CASCADE;

SELECT [DISTINCT | ALL]
  {* | [*columnExpression* [AS *newName*]] [,...] }
FROM *TableName* [*alias*] [, ...]
[WHERE *condition*]
[GROUP BY *columnList*] [HAVING *condition*]
[ORDER BY *columnList*]

SELECT Command

- SELECT : Specifies which columns are to appear in output
- FROM : Specifies table(s) to be used
- WHERE : Filters rows
- GROUP BY : Forms groups of rows with same column value
- HAVING : Filters groups subject to some condition
- ORDER BY : Specifies the order of the output

Insert Command

- Add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

INSERT INTO *TableName (Attribute1, Attribute2, ...)*
VALUES (*value1, value2, ...*);

Delete Command

DELETE FROM *TableName*
WHERE *Condition*;

- Removes tuples from a relation
- Tuples are deleted from only one table at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that all tuples in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

Update Command

UPDATE *TableName*
SET *Set-Clause*
WHERE *Condition;*

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples in the same relation
- Referential integrity should be enforced

Views

- A view is a "virtual" table that is derived from other tables
- Allows for limited update operations (since the table may not physically be stored)
- Allows full query operations
- A convenience for expressing certain operations

VIEWs

- Specify a different WORKS_ON table (view)

```
CREATE VIEW   Works_on_new AS
    SELECT    Fname, Lname, Pname, Hours
    FROM      Employee, Project, Works_on
    WHERE     SSN = ESSN AND Pno = Pnumber;
```

- We can specify SQL queries on a newly create table (view):

```
SELECT Fname, Lname From Works_on_new
WHERE Pname = 'Seena';
```

- When no longer needed, a view can be dropped:

```
DROP VIEW Works_on_new;
```

C9: Database Security

- Types of Security:
  - Legal and ethical issues
  - Policy issues
  - System-related issues
  - The need to identify multiple security levels

Introduction to Database Security Issues (2)

- Three Basic Concepts:
  - Authentication: a mechanism that determines whether a user is who he or she claims to be.
  - Authorization: the granting of a right or privilege, which enables a subject to legitimately have access to a system or a system's objects.
  - Access Control: a security mechanism (of a DBMS) for restricting access to a system's objects (the database) as a whole.

Introduction to Database Security Issues (3)

- Threats:
  - Any situation or event, whether intentional or unintentional, that will adversely affect a system and consequently an organization.
  - Threats to:
    - Computer systems
    - Databases

Introduction to Database Security Issues (4)

- Threats to databases:
  - Loss of **integrity**
  - Loss of **availability**
  - Loss of **confidentiality**
- To protect databases against these types of threats four kinds of countermeasures can be implemented:
  - **Access control**
  - **Inference control**
  - **Flow control**
  - **Encryption**

Introduction to Database Security Issues (5)

- A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access.
- Two types of database security mechanisms:
  - **Discretionary** security mechanisms
  - **Mandatory** security mechanisms

Introduction to Database Security Issues (6)

- The security mechanism of a DBMS must include provisions for restricting access to the database as a whole
  - This function is called **access control** and is handled by creating user accounts and passwords to control login process by the DBMS.

Introduction to Database Security Issues (7)

- The security problem associated with databases is that of controlling the access to a **statistical database**, which is used to provide statistical information or summaries of values based on various criteria.
  - The countermeasures to **statistical database security** problem is called **inference control measures**.

Introduction to Database Security Issues (8)

- Another security is that of **flow control**, which prevents information from flowing in such a way that it reaches unauthorized users.
- Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels**.

Introduction to Database Security Issues (9)

- A final security issue is **data encryption**, which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type communication network.
- The data is **encoded** using some **encoding algorithm**.
  - An unauthorized user who access encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher data.

Database Security and the DBA (1)

- The database administrator (**DBA**) is the central authority for managing a database system.
  - The DBA's responsibilities include:
    - granting privileges to users who need to use the system.
    - classifying users and data in accordance with the policy of the organization.
- The DBA is responsible for the overall security of the database system.

Database Security and the DBA (2)

- The DBA has a DBA account in the DBMS:
  - Sometimes these are called a system or superuser account.
  - These accounts provide powerful capabilities such as:
    - 1. Account creation
    - 2. Privilege granting
    - 3. Privilege revocation
    - 4. Security level assignment
  - Action 1 is access control, whereas 2 and 3 are discretionary and 4 is used to control mandatory authorization.

Types of Discretionary Privileges (1)

- The **account level**:
  - At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- The **relation level** (or **table level**):
  - At this level, the DBA can control the privilege to access each individual relation or view in the database.

Comparing DAC and MAC (1)

- **Discretionary Access Control (DAC)** policies are characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains.
  - The main drawback of **DAC** models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs.

Comparing DAC and MAC (2)

- By contrast, mandatory policies ensure a high degree of protection in a way, they prevent any illegal flow of information.
- Mandatory policies have the drawback of being too rigid and they are only applicable in limited environments.
- In many practical situations, discretionary policies are preferred because they offer a better trade-off between security and applicability.

Types of Discretionary Privileges (5)

- The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model where:
  - The **rows** of a matrix M represents **subjects** (users, accounts, programs)
  - The **columns** represent **objects** (relations, records, columns, views, operations).
  - Each position **M(i,j)** in the matrix represents the types of privileges (read, write, update) that **subject i** holds on **object j**.

Bell-LaPadula Model

- Typical **security classes** are *top secret (TS), secret (S), confidential (C), and unclassified (U)*, where TS is the highest level and U is the lowest one: TS ≥ S ≥ C ≥ U
- Two restrictions are enforced on data access based on the subject/object classifications:
  - A subject S is not allowed read access to an object O unless class(S) ≥ class(O). This is known as the **simple security property.**
  - A subject S is not allowed to write an object O unless class(S) ≤ class(O). This known as the **star property** (or * property).