Concepts To Remember and Understand

Chapter 1

- **Data** – known facts that can be recorded and that have implicit meaning
- **Database** – collection of related data
- **Mini world** – the aspect of the real word the database represents
- **DBMS** – database mgmt system is a collection of programs that enables users to create and maintain a database.
- **Defining** – a database involves specifying the data types, structures, and constraints.
- **Meta-Data** – the database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary. Describes structure of the primary database.
- **Query** – typically causes some data to be retrieve
- **Transaction** – may cause some data to be read and some data to be written in the database
- **Database System** – Database and DBMS
- **Structured vs Unstructured -**
- **Data Catalog/Data Dictionary** -
- **Program-data Independence** – The structure of data files stored in the DBMS catalog separately from the access programs
- **Integrity Constraints (Business Rules)** – things that must hold for the data
    - **Referential Integrity** – When we specify, for example, that every section record must be related to a course record.
    - **Key/Uniqueness** – every item must have a unique key to distinguish it from others
    - **Domain Constraint** – where each column in a database contains the same type of data
    - **Entity Integrity Constraint** – no primary key can be null
    - **Semantic Integrity Constraint** – ensures data entered for a row is an allowable value for that row
- **Data Independence** – The capacity to change the schema at one level without changing it at another
    - **Logical** – Capacity to change the conceptual schema without have to change external schemas or application programs
    - **Physical** – capacity to change the internal schema without having to change the conceptual schema.
- **DDL** – data definition language – used by the DBA and by database designers to define the conceptual and internal schemas
- **DML** – data manipulation language – language to manipulate the database
    - **High Level / non-procedural** – can be used on its own to specify complex database operations concisely. Retrieves records more than one at time.
    - **set oriented** – processing and retrieval of records in sets
    - **embedded vs standalone** – embedded means the dml is contained in

another language while standalone means the dml can be run on its own.
    - **Low Level / procedural** – must be embedded in a general purpose programming language. Retrieves individual records or object from the database processing them one at a time.
    - **record-level** – processing and retrieval of records one at time.

Chapter 2

- **Three-Schema DBMS Architecture** – database with three characteristics. (1)use of a catalog to store the database description (schema) so as to maek it self describing (2) insulation of programs and data (3) support of multiple user views
    - **Internal Level** – describes the physical storage structures
    - **Conceptual Level** – describes the structure of the whole database for a community of users
    - **External View** – describes part of the database that a user can see

Chapter 7

- **Database Design Phases** – phases in which a database is designed. Starts with requirements of and analysis of design. Moves to conceptual design, then logical design, lastly physical design.
- **Entities –** basic object that the ER diagram has which is a thing in the real world with an independent existence.
    - **Type vs set** – a type defines a collection (or set) of entities that have the same attributes. A set is the collection of all entities of a particular entity type in the database at any point in time. Strong has a square box and weak has a double squared box.
    - **Strong vs Weak** – weak entities do not have key attributes of their own where as strong entities do have a key attribute. Weak can't exist on its own.
- **Attribute** – the particular properties that describe an entity.
    - **stored vs derived** – Stored is where the value actually exists in the database, derived is something calculated based on what is stored. Stored is a solid oval and derived is a dotted oval.
    - **multivalued vs single-valued** – Single Valued is where only one value can exist, like a SSN. Multivalued is where you can have multiple values, like phone numbers. Single valued is a single oval, double valued is a double oval.
    - **composite vs atomic** – composite is where an attribute can be broken up into smaller parts, like an address. Atomic is just a single value
    - **key vs non-key** - a key is when an attribute whose values are distinct for each individual entity in the entity set. Like SSN. Non Key is where it's non unique, like name. Partial Key uniquely identifies entity related to the owner. A key is underlined and a partial key is dotted underlined.
- **Relationship** – references between different entities. Must have unique name.
    - **Degree** – is the number of participating entity types, IE, binary or

ternary
    - **Binary** – When two entities participating.
    - **Ternary** – When three entities are participating.
- **Cardinality Ratio** – specifies the maximum number of relationships instances that an entity can participate in. IE, 1:1, 1:M, M:N. Another example: Employee 1 Manages N Departments. So, 1:N an employee may manage many depts and each dept has 1 manager. N:1 an employee manages 1 dept and depts have many managers.
- **Participation Constraints** – specifies whether the existence of an entity depends on its being related to another entity via the relationship type. Specifies the minimum number of relationship instances that each entity can participate in.
    - **Total/Mandatory** – where an entity has to participate in a relationship in order to exist. IE, if an employee is required to be a member of a department, then an employee entity cant exist with a relationship with a department entity. This is represented by a double line between entity and relationship.
    - **Partial/Optional** – Means where some or part of the entities participate in the relationship. IE, non all employees have a relationship with the manages role. This is represented by a single line between entity and relationship.
- **Structural Constraints** – is the cardinal ratio and participation constraints together. Min comes from the participation. If partial, min = 0. If total, min = 1 (or true minimum number). Max comes from cardinality from opposite side.
    - **min/max** – the min and max values associated with a constraint.
- **recursive** – where a role points to the same entity. Like employee and supervisor. Supervisor is a type of employee.
- **roles** – signifies the role that a participating entity from the entity type plays in each relationship instance and helps to explain what the relationship means

Chapter 8

- **ER Diagram Notation** – particular notation for entity relationship diagrams.
- **Specialization** – is the process of defining a set of sub classes of an entity type. The set of sub classes that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.
- **Generalization** – process of defining a generalized entity type from the given entity types. The new entity created will be the superclass of the children entities that helped define it.
- **Subclass vs superclass** – superclass is a parent entity type that contains children entities. Subclass is the children entities of a parent entity.
- **Inheritance** – the process of the sub class inheriting the attributes and relationships of the super class.
    - **attribute** – the attributes of the superclass now apply to the sub class.
    - **relationship** – the relationships of the superclass now apply to the sub class.
- **specialization/generalization constraints** -

    - **total vs partial** – total specifies that every entity in the superclass must be a member of at least one subclass in the specialization. Partial specifies you don't have to belong to at least one subclass of the specialization. IE, for total, if every employee must be hourly or salary, then it's a total constraint. For partial, employee entities do not belong to any of the following sub classes (Secretary, engineer, technician).
    - **disjoint vs overlapping** – disjoint is where an entity can be a member of at most one of the sub classes of the specialization where as overlapping is where an entity can be a member of more than one of the sub classes of the specialization. IE, an employee entity has two children, salaried and hourly. Disjoint means you can be salary or hourly, but not both. Overlapping means you can be both.
- **category** – where an entity can have multiple super classes. IE, a person, bank, or company can all be an owner of a vehicle. We create a collection of entities, category, that includes entities of all three types to play the role of the vehicle owner. The vehicle owner will be sub class of the union of the three distinct entities.
    - **total** – holds the union of all entities in its superclass
    - **partial** – can hold a subset of the union.
- **union** -
- **aggregation** – an abstraction concept for building composite objects from their component objects.
- **association** – is used to associate objects from several independent classes.

Chapter 3

- **Domain** – is a set of atomic values.
- **Attribute** – name of the role played by some domain in the relation schema.
- **Tuples** – can be considered as a set of (<attribute>,<value>) pairs, where each pair gives the value of the mapping.
- **Relation Schema vs Relation or Relation State** – Relation schema is made up of a relation and a list of attributes. A relation of the relation schema is a set of n-tuples.
- **Relational DB Schema vs Relation DB State** -
- **NULL Values** – Value that may not be known or may not apply to a tuple.
    - **Missing** – where there should be a value but it's gone.
    - **Unknown** – Where there could be a value or not.
- **Degree of Relation** – Number of attributes n of its relation schema.
- **DB Constraints** – Rules derived from the mini world that it represents.
    - **Schema Based Constraints** – Constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL.
        - **Domain = Atomic Value of Attribute** – specify that within each tupel, the value of each attribute must be an atomic value from the domain.
        - **Key** – a unique identifier for a tuple. Is a property of the relation

schema and should hold on EVERY valid relation state of the schema. A key is a superkey but a superkey is not a key. A key is determined from the meaning of the attributes and is time invariant.
    - **Superkey –** specifies uniqueness constraint that no two distinct tuples in any state can have the same value. It is a unique value representing the tuple.
    - **Candidate Key –** Possible choices for primary key.
    - **Primary Key –** One of the candidate keys that is used to identify the tuples in the relation. Can't ever be null.
    - **Unique Key –** a candidate key not chosen as a primary key.
    - **Foreign Key –** primary key from another relation in your relation.
  - **Entity Integrity Constraint –** states that no primary key value can be null.
  - **Referential Integrity (Foreign Key) –** is specified between two relations and is used to maintain the consistency among tuples in the two relations.
    - **Referencing Relation –** Contains the foreign key
    - **Referenced Relation –** Foreign key points to
  - **Semantic Integrity Constraints (Business Rules) –** general constraints, IE, salary of employee should not exceed salary of supervisor. They enforced using triggers and asserts that will check for these types of constraints.
- **Data Dependencies -**
  - **Functional Dependencies -**
  - **Multivalued Dependencies -**

Chapter 9

- **Mapping Regular Entity Type** – Look for all strong entities of the ER diagram and make a relation for each one. The relation will include all simple attributes of the entity. For composite attributes, only list the components of it. Choose one key as the primary key and underline it. If more than one key, underline them both.
- **Mapping Weak Entity Type** – Look for all weak entities of the ER diagram and make a relation for each one. The relation will include all simple attributes of the entity. For composite attributes, only list the components of it. Choose one key as the partial key and underline it. If more than one key, underline them both. Also include as a foreign key the primary key from the owning entity and underline it.
- **Mapping 1:1 Relationship** -
  - **Foreign Key Approach –** Choose one of the relations and include as a foreign key the primary key of the other relation. It is better to choose an entity type with total participation to have the foreign key. Include all simple attributes of the 1:1 relationship type as attributes of relation with the foreign key.
  - **Merged Relation Approach –** An alternative is to merge the two entity

types and the relationship into a single relation. This is possible when both participations are total.
    - **Cross Reference or Relationship Relation Approach –** Setup a third relation for the purpose of cross-referencing the primary keys of the two relations representing the entity types. This is required for M:N relationships.
- **Mapping 1:N Relationship** – On the side of the relationship that has the N, add as a foreign key the primary key from the other entity involved.
- **Mapping M:N Relationship** – Create a new relation that includes as foreign keys the primary keys of the two relations involved in the relationship. Also, add any simple attributes of the relationship.
- **Mapping Multivalued Attribute** – Create a new relation that includes an attribute corresponding to the multivalued attribute. Also add as a foreign key the primary key of the relation that contained the multivalued attribute in the first place.
- **Mapping Composite Attribute** – To map composite attributes into a relation, add just the components of the composite as attributes in the relation.
- **Mapping N-ary Relationship** – Create a new relation with foreign keys from the primary keys of all types of relations that are involved in the relationship. Also add any attributes of the relationship to the newly created relation.
- **Mapping Specialization/Generalization** -
    - **Option A – Multiple Relations – Superclass & subclass –** Create the superclass relation like usual, then create new relations with the specific attributes for each subclass. Also, add as a foreign key the primary key of the superclass.
    - **Option B – Multiple Relations – Subclass Relations Only –** Create just the subclass relations but add as attributes the attributes that were there in the superclass, including it's primary key.
    - **Option C – Single Relation with one Type Attribute –** Create one relation that has all the attributes of the superclass and also all the attributes of each subclass to make one giant relation.
    - **Option D – Single Relation with Multiple Type attributes –** Create one relation that has all the attributes of the superclass and also all the attributes of each subclass to make one giant relation. Now add a boolean flag for each subclass type that will tell you which of the subclasses are selected.
- **Correspondence between ER and Relational Models**
    - Entity Type = Entity Relational
    - 1:1 or 1:N Relationship Type = Foreign Key (or relationship relation)
    - M:N relationship Type = Relationship relation and two foreign keys
    - n-ary Relationship Type = Relationship relation and n foreign keys
    - Simple Attribute = Attributes
    - Composite Atrribute = set of simple component attributes
    - Multivalued Attribute = relation and foreign key
    - value set = domain
    - key attribute = primary (or secondary) key

Chapter 4

- **SQL Commands**
  - **Create Schema –** CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
  - **Create Table -** CREATE TABLE COMPANY.EMPLOYEE;
    - Example for Department Table

```
CREATE TABLE COMPANY.DEPARTMENT
( Dname          VARCHAR(15)          NOT NULL,
  Dnumber        INT                  NOT NULL,
  Mgr_ssn        CHAR(9)              NOT NULL,
  Mgr_start_date DATE ,
  PRIMARY KEY (Dnumber),
  UNIQUE (Dname)
  FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn));
```

- **Data Types -**
  - **Numeric -** INTEGER, INT, SMALLINT, FLOAT, REAL, DECIMAL
  - **Character-String –**
    - **CHAR(n)** – fixed length
    - **VARCHAR(n)** – varying with n maximum
  - **Bit-string -**
    - **BIT(n) –** fit bit length of n
    - **BIT VARYING(n) -** bit length max n
  - **Boolean –** TRUE or FALSE
  - **DATE –** YYYY-MM-DD
  - **TIME –** HH:MM:SS
  - **timestamp –** Includes data and time YYYY-MM-DD HH:MM:SS
- **Attribute Definitions -**
  - **DEFAULT <value> -** default value in tuple if none provided
  - **CHECK –** CHECK (Dnumber > 0 AND Dnumber < 21)
  - **Unique –** secondary key
- **SELECT –** (attribute list)
- **FROM –** (table list)
- **WHERE –** (condition)

Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname='John' AND Minit='B' AND Lname='Smith';
```

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT Fname, Lname, Address
```

```
FROM EMPLOYEE, DEPARTMENT
WHERE Dname='Research' AND Dnumber=Dno;
```

Q1A:

```
SELECT Fname, EMPLOYEE.Name, Address
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.Name='Research' AND
        DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Q1':

```
SELECT EMPLOYEE.Fname, EMPLOYEE.LName, EMPLOYEE.Address
FROM EMPLOYEE, DEPARTMENT
WHERE  DEPARTMENT.DName='Research' AND
DEPARTMENT.Dnumber=EMPLOYEE.Dno;
```

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM EMPLOYEE AS E, EMPLOYEE AS
WHERE E.Super_ssn=S.Ssn;
```

Q1B:

```
SELECT E.Fname, E.LName, E.Address
FROM EMPLOYEE E, DEPARTMENT D
WHERE D.DName='Research' AND D.Dnumber=E.Dno;
```

Queries 9 and 10. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

```
SELECT Ssn
FROM EMPLOYEE;

SELECT Ssn, Dname
FROM EMPLOYEE, DEPARTMENT;
```

Q1C:

```
SELECT *
 FROM EMPLOYEE
 WHERE Dno=5;
```

Q1D:

```
SELECT *
FROM EMPLOYEE, DEPARTMENT
WHERE Dname='Research' AND Dno=Dnumber;
```

Q10A:
```
SELECT *
FROM EMPLOYEE, DEPARTMENT;
```

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

Q11:
```
SELECT ALL Salary
FROM EMPLOYEE;
```

Q11A:
```
SELECT SELECT DISTINCT Salary
FROM FROM EMPLOYEE;
```

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
( SELECT DISTINCT Pnumber
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Lname='Smith' )

UNION

( SELECT DISTINCT Pnumber
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber=Pno AND Essn=Ssn AND Lname='Smith' );
```

Query 12. Retrieve all employees whose address is in Houston, Texas.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Address LIKE '%Houston,TX%';
```

Query 12A. Find all employees who were born during the 1950s.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';
```

Query 13. Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
SELECT E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname='ProductX';
```

Query 14. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
SELECT *
FROM EMPLOYEE
WHERE (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

Q15:

```
SELECT D.Dname, E.Lname, E.Fname, P.Pname
WHERE DEPARTMENT D, EMPLOYEE E, WORKS_ON W, PROJECT P
FROM D.Dnumber= E.Dno AND E.Ssn= W.Essn AND W.Pno= P.Pnumber
ORDER BY D.Dname, E.Lname, E.Fname;
```

Insert1:

```
INSERT INTO EMPLOYEE
VALUES ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

Insert1A:

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)
VALUES ('Richard', 'Marini', 4, '653298653');
```

Insert3:

```
INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno)
VALUES  ('Robert', 'Hatcher', '980760540', 2);
```

Insert2A:

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno)
VALUES ('Robert', 'Hatcher', 5);
```

Insert3A:

```
CREATE TABLE WORKS_ON_INFO
( Emp_name VARCHAR(15),
Proj_name VARCHAR(15),
Hours_per_week DECIMAL(3,1) );
```

Insert3B:

```
INSERT INTO WORKS_ON_INFO ( Emp_name, Proj_name, Hours_per_week )
SELECT E.Lname, P.Pname, W.Hours
FROM PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE P.Pnumber=W.Pno AND W.Essn=E.Ssn;
```

Delete4A:

```
DELETE FROM EMPLOYEE
WHERE Lname='Brown';
```

Delete4B:

```
DELETE FROM EMPLOYEE
```

WHERE Ssn='123456789';

Delete4C:
DELETE FROM EMPLOYEE
WHERE Dno=5;

Delete4D:
DELETE FROM EMPLOYEE;

Update5:
UPDATE PROJECT
SET Plocation = 'Bellaire', Dnum = 5
WHERE Pnumber=10;

Update6:
UPDATE EMPLOYEE
SET Salary = Salary * 1.1
WHERE Dno = 5;