Summary on Chapter 3
Relational Data Model and Relational Database Constraints

1. Relational Model Concepts

- The relational Model of Data is based on the concept of aRelation
  - Has a formal mathematical foundation provided by set theory and first order predicate logic
- We review the essentials of theformal relational model in this chapter
- Inpractice, there is astandard model based on SQL (Structured Query Language) – described in Chapters 4 and 5
- Note: There are several important differences between theformal model and thepractical model.
- The model was first proposed by Dr. E.F. Codd of IBM Research in 1970 in the following paper:
  - "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970
- The above paper caused a major revolution in the field of database management
- Dr. Codd earned the ACM Turing Award in 1981

1.1 Domains, Attributes, Tuples, and Relations

Informal Definitions

- Informally, a relation looks like a table of values (see Figure 3.1).
- A relation contains a set of rows.
- The data elements in each row represent certain facts that correspond to a real-world entity or relationship
  - In the formal model, rows are called tuples
- Each column has a column header that gives an indication of the meaning of the data items in that column
  - In the formal model, the column header is called an attribute name (or just attribute)

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT</td>
<td>Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tuples</th>
<th>Name</th>
<th>Ssn</th>
<th>Home_phone</th>
<th>Address</th>
<th>Office_phone</th>
<th>Age</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Benjamin Bayar</td>
<td>305-61-2435</td>
<td>(817)973-1616</td>
<td>2918 Bluebonnet Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.21</td>
</tr>
<tr>
<td></td>
<td>Chung-cha Kim</td>
<td>381-62-1245</td>
<td>(817)375-4409</td>
<td>125 Kirby Road</td>
<td>NULL</td>
<td>16</td>
<td>2.89</td>
</tr>
<tr>
<td></td>
<td>Dick Davidson</td>
<td>422-11-2320</td>
<td>NULL</td>
<td>3452 Elgin Road</td>
<td>(817)749-1253</td>
<td>25</td>
<td>3.53</td>
</tr>
<tr>
<td></td>
<td>Rohan Panchal</td>
<td>489-22-1100</td>
<td>(817)376-9821</td>
<td>265 Lark Lane</td>
<td>(817)749-6492</td>
<td>28</td>
<td>3.93</td>
</tr>
<tr>
<td></td>
<td>Barbara Benson</td>
<td>533-69-1238</td>
<td>(817)839-8461</td>
<td>7384 Fontana Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.25</td>
</tr>
</tbody>
</table>

Figure 3.1
The attributes and tuples of a relation STUDENT.

- Key of a Relation:
  - Each row (tuple) in the table is uniquely identified by the value of a particular attribute (or several attributes together)
  - Called the key of the relation
In the STUDENT relation, SSN is the key

If no attributes possess this uniqueness property, a new attribute can be added to the relation to assign unique row-id values (e.g., unique sequential numbers) to identify the rows in a relation
- Called artificial key or surrogate key

Formal Definitions (PP. 62)

- **Relation Schema** (or description) of a Relation:
  - Denoted by \( R(A_1, A_2, ..., A_n) \)
  - \( R \) is the name of the relation
  - The attributes of the relation are \( A_1, A_2, ..., A_n \)
  - \( n \) is the cardinality (degree) of the relation
- Example:
  - CUSTOMER (Cust-id, Cust-name, Address, PhoneNo)
  - CUSTOMER is the relation name
  - The CUSTOMER relation schema (or just relation) has four attributes: Cust-id, Cust-name, Address, Phone#
- Each attribute has a domain (denoted by \( D = \text{dom}(A_i) \)) or a set of valid values.
  - For example, the domain of Cust-id can be 6 digit numbers.
- A tuple is an ordered set of values (enclosed in angled brackets ‘< … >’)
- Each value is derived from an appropriate domain.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - \(<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">\)
  - Called a 4-tuple because it has 4 values
  - In general, a particular relation will have \( n \)-tuples, where \( n \) is the number of attributes for the relation
- A relation is a set of such tuples (rows)
- A domain of values can have a logical definition:
  - Example: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain also has a data-type or a format defined for it.
  - The USA_phone_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
  - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd:mm:yyyy etc.
- The attribute name designates the role played by a domain in a relation:
  - Used to interpret the meaning of the data elements corresponding to that attribute
  - Example: The domain Date may be used to define two attributes “Invoice-date” and “Payment-date” with different meanings (roles)
- Formally, a relation state \( r(R) \) is a subset of the Cartesian product of the domains of its attributes
  - Each domain contains the set of all possible values the attribute can take.
  - The Cartesian product contains all possible tuples from the attribute domains
  - The relations state \( r(R) \) is the subset of tuples that represent valid information in the mini-world at a particular time
- Formally (see Figure 3.1),
  - Given relation schema \( R(A_1, A_2, ..., A_n) \)
  - Relation state \( r(R) \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times ... \times \text{dom}(A_n) \)
  - \( R(A_1, A_2, ..., A_n) \) is the schema of the relation
- \( R \) is the **name** of the relation
- \( A_1, A_2, \ldots, A_n \) are the **attributes** of the relation
- \( r(R) \): is a specific **state** (or "instance" or "population") of relation \( R \) – this is a set of tuples (rows) in the relation at a particular moment in time
  - \( r(R) = \{ t_1, t_2, \ldots, t_n \} \) where each \( t_i \) is an \( n \)-tuple
  - \( t_i = \langle v_{i1}, v_{i2}, \ldots, v_{in} \rangle \) where each \( v_j \) element-of \( \text{dom}(A_j) \)

**Example**

- Let \( R(A_1, A_2) \) be a relation schema:
  - Let \( \text{dom}(A_1) = \{0,1\} \)
  - Let \( \text{dom}(A_2) = \{a,b,c\} \)
- Then: The **Cartesian product** \( \text{dom}(A_1) \times \text{dom}(A_2) \) contains all possible tuples from these domains:
  - \( \{ \langle 0,a \rangle, \langle 0,b \rangle, \langle 0,c \rangle, \langle 1,a \rangle, \langle 1,b \rangle, \langle 1,c \rangle \} \)
- The relation state \( r(R) \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \)
- For example: One possible state \( r(R) \) could be \( \{ \langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle \} \)
  - This state has three 2-tuples: \( \langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle \)

**Relation Definitions Summary**

<table>
<thead>
<tr>
<th>Informal Terms</th>
<th>Formal Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Relation</td>
</tr>
<tr>
<td>Column Header</td>
<td>Attribute</td>
</tr>
<tr>
<td>All possible Column Values or Data Type</td>
<td>Domain</td>
</tr>
<tr>
<td>Row</td>
<td>Tuple</td>
</tr>
<tr>
<td>Table Definition</td>
<td>Schema of a Relation</td>
</tr>
<tr>
<td>Populated Table</td>
<td>State of the Relation</td>
</tr>
</tbody>
</table>

**1.2. Characteristics of a Relation**

- Ordering of tuples in a relation \( r(R) \):
  - The tuples are **not considered to be ordered**, because a relation is a set of tuples (elements of a set are unordered) – see Figure 3.2
- Ordering of attributes in a relation schema \( R \) (and of values within each tuple):
  - We will consider the attributes in \( R(A_1, A_2, \ldots, A_n) \) and the values in each \( t = \langle v_1, v_2, \ldots, v_n \rangle \) to be **ordered**
  - However, a more general definition of relation (which we will not use) does not require attribute ordering
  - In this case, a tuple \( t = \{ \langle a_1, v_1 \rangle, \ldots, \langle a_p, v_p \rangle \} \) is an unordered set of \( n \) \( \langle \text{attribute}, \text{value} \rangle \) pairs – one pair for each of the relation attributes (see Figure 3.3)
Figure 3.2
The relation STUDENT from Figure 3.1 with a different order of tuples.

<table>
<thead>
<tr>
<th>Name</th>
<th>Ssn</th>
<th>Home_phone</th>
<th>Address</th>
<th>Office_phone</th>
<th>Age</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dick Davidson</td>
<td>422-11-2320</td>
<td>NULL</td>
<td>3452 Elgin Road</td>
<td>(817)749-1253</td>
<td>25</td>
<td>3.53</td>
</tr>
<tr>
<td>Barbara Benson</td>
<td>533-69-1238</td>
<td>(817)839-8461</td>
<td>7384 Fontana Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.25</td>
</tr>
<tr>
<td>Rohan Panchal</td>
<td>489-22-1100</td>
<td>(817)376-9821</td>
<td>265 Lark Lane</td>
<td>(817)749-6492</td>
<td>28</td>
<td>3.93</td>
</tr>
<tr>
<td>Chung-cha Kim</td>
<td>381-62-1245</td>
<td>(817)375-4409</td>
<td>125 Kirby Road</td>
<td>NULL</td>
<td>18</td>
<td>2.89</td>
</tr>
<tr>
<td>Benjamin Bayer</td>
<td>305-61-2435</td>
<td>(817)373-1616</td>
<td>2918 Bluebonnet Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.21</td>
</tr>
</tbody>
</table>

Figure 3.3
Two identical tuples when the order of attributes and values is not part of relation definition.

\[ t = (\text{Name}, \text{Dick Davidson}), (\text{Ssn}, 422-11-2320), (\text{Home_phone}, \text{NULL}), (\text{Address}, 3452 Elgin Road), (\text{Office_phone}, (817)749-1253), (\text{Age}, 25), (\text{Gpa}, 3.53) \]

\[ t' = (\text{Address}, 3452 Elgin Road), (\text{Name}, \text{Dick Davidson}), (\text{Ssn}, 422-11-2320), (\text{Age}, 25), (\text{Office_phone}, (817)749-1253), (\text{Gpa}, 3.53), (\text{Home_phone}, \text{NULL}) \]

- Values in a tuple:
  - All values are considered atomic (indivisible).
  - Each value must be from the domain of the attribute for that column
    - If tuple \( t = \langle v_1, v_2, \ldots, v_n \rangle \) is a tuple (row) in the relation state \( r \) of \( R(A_1, A_2, \ldots, A_n) \)
    - Then each \( v_i \) must be a value from \( \text{dom}(A_i) \)
  - A special \text{null} value is used to represent values that are unknown or inapplicable to certain tuples.

1.3. Relational Model Notation

- Notation:
  - We refer to \textit{component values} of a tuple \( t \) by:
    - \( t[A_i] \) or \( t.A_i \)
    - This is the value \( v_i \) of attribute \( A_i \) for tuple \( t \)
  - Similarly, \( t[A_u, A_v, \ldots, A_w] \) refers to the subtuple of \( t \) containing the values of attributes \( A_u, A_v, \ldots, A_w \), respectively in \( t \)

2. Relational Model Constraints and Relational Database Schemas

- Constraints are \textbf{conditions} that must hold on all \textit{valid} relation states.
- Constraints are \textit{derived} from the mini-world semantics (Universe of Discourse: UoD)
- Generally divided into three main categories
  - Inherent model-based constraints (implicit constraints)
  - Schema-based constraints (explicit constraints)
  - Application based (or semantic constraints or business rules)
- There are three main types of schema based constraints in the relational model:
  - \textbf{Key} constraints
  - \textbf{Entity integrity} constraints
2.1. Domain Constraints

- Implicit constraint is the domain constraint
  - Every value in a tuple must be from the domain of its attribute (or it could be null, if allowed for that attribute)

2.2. Key Constraints

- Superkey SK of R:
  - Is a set of attributes SK of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - That is, for any distinct tuples t1 and t2 in r(R), t1.SK ≠ t2.SK
    - This condition must hold in any valid state r(R)

- Key K of R:
  - Is a "minimal" superkey
  - Formally, a key K is a superkey such that removal of any attribute from K results in a set of attributes that is not a superkey (or key) any more (does not possess the superkey uniqueness property)
  - Hence, a superkey with one attribute is always a key

- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - CAR has two keys (determined from the mini-world constraints):
    - Key1 = {State, Reg#}
    - Key2 = {SerialNo}
  - Both are also superkeys of CAR
  - However, {SerialNo, Make} is a superkey but not a key.

- In general:
  - Any key is a superkey (but not vice versa)
  - Any set of attributes that includes a key is a superkey
  - A minimal superkey is also a key

- If a relation has several keys, they are called candidate keys; one is chosen to be the primary key; the others are called unique (or secondary) keys
  - The primary key attributes are underlined.

- Example: Consider the CAR relation schema:
  - CAR(State, RegNo, SerialNo, Make, Model, Year)
  - We choose License_number (which contains (State, RegNo) together) as the primary key – see Figure 3.4. Engine serial number is a candidate key.

- The primary key value is used to uniquely identify each tuple in a relation
  - Provides the tuple identity
  - Also used to reference the tuple from other tuples

- General rule: Choose the smallest-sized candidate key (in bytes) as primary key
Not always applicable – choice is sometimes subjective (as in Figure 3.4)

<table>
<thead>
<tr>
<th>License_number</th>
<th>Engine_serial_number</th>
<th>Make</th>
<th>Model</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas ABC-739</td>
<td>A69352</td>
<td>Ford</td>
<td>Mustang</td>
<td>02</td>
</tr>
<tr>
<td>Florida TVP-347</td>
<td>B43696</td>
<td>Oldsmobile</td>
<td>Cutlass</td>
<td>05</td>
</tr>
<tr>
<td>New York MPO-22</td>
<td>X83554</td>
<td>Oldsmobile</td>
<td>Delta</td>
<td>01</td>
</tr>
<tr>
<td>California 432-TFY</td>
<td>C43742</td>
<td>Mercedes</td>
<td>190-D</td>
<td>99</td>
</tr>
<tr>
<td>California RSK-629</td>
<td>Y82935</td>
<td>Toyota</td>
<td>Camry</td>
<td>04</td>
</tr>
<tr>
<td>Texas RSK-629</td>
<td>U028365</td>
<td>Jaguar</td>
<td>XJS</td>
<td>04</td>
</tr>
</tbody>
</table>

**Figure 3.4**  
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

### 2.3. Relational Databases and Schemas

**Relational Database Schema:**
- A set $S$ of relation schemas that belong to the same database.
- And a set of integrity constraints (part of relation schemas)
- $S$ is the name of the whole **database schema**
- $S = \{R_1, R_2, \ldots, R_n\}$
- $R_1, R_2, \ldots, R_n$ are the names of the individual **relation schemas** within the database $S$
- Figure 3.5 shows a COMPANY database schema with 6 relation schemas

**EMPLOYEE**

<table>
<thead>
<tr>
<th>Fname</th>
<th>Minit</th>
<th>Lname</th>
<th>Ssn</th>
<th>Bdate</th>
<th>Address</th>
<th>Sex</th>
<th>Salary</th>
<th>Super_ssn</th>
<th>Dno</th>
</tr>
</thead>
</table>

**DEPARTMENT**

<table>
<thead>
<tr>
<th>Dname</th>
<th>Dnumber</th>
<th>Mgr_ssn</th>
<th>Mgr_start_date</th>
</tr>
</thead>
</table>

**DEPT_LOCATIONS**

<table>
<thead>
<tr>
<th>Dnumber</th>
<th>Dlocation</th>
</tr>
</thead>
</table>

**PROJECT**

<table>
<thead>
<tr>
<th>Pname</th>
<th>Pnumber</th>
<th>Plocation</th>
<th>Dnum</th>
</tr>
</thead>
</table>

**WORKS_ON**

<table>
<thead>
<tr>
<th>Essn</th>
<th>Phno</th>
<th>Hours</th>
</tr>
</thead>
</table>

**DEPENDENT**

<table>
<thead>
<tr>
<th>Essn</th>
<th>Dependent_name</th>
<th>Sex</th>
<th>Bdate</th>
<th>Relationship</th>
</tr>
</thead>
</table>

**Figure 3.5**  
Schema diagram for the COMPANY relational database schema.

### Relational Database State

- **See the COMPANY database state (Figure 3.6)**
  - Each relation has a set of tuples
    - The tuples in each table satisfy key and other constraints
    - If all constraints are satisfied by a database state, it is called a **valid state**
  - The database state changes to another state whenever the tuples in any relation are changed via insertions, deletions, or updates
2.4. Integrity, Referential Integrity, and Foreign Keys

Entity Integrity:
- The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R).
  - This is because primary key values are used to identify the individual tuples.
  - If PK ≠ null for any tuple t in r(R)
  - If PK has several attributes, null is not allowed in any of these attributes
- Note: Other attributes of R may be also be constrained to disallow null values (called NOT NULL constraint), even though they are not members of the primary key.

Referential Integrity Constraint
A constraint involving two relations
- The previous constraints (key, entity integrity) involve a single relation.
- Used to specify a relationship among tuples in two relations:
  - The referencing relation and the referenced relation.

Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2.
- A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1.FK = t2.PK
- Referential integrity can be displayed as a directed arc from R1.FK to R2.PK – see Figure 3.7

Statement of the constraint
- For a particular database state, the value of the foreign key attribute (or attributes) FK in each tuple of the referencing relation R1 can be either:
  - (1) An existing primary key (PK) value of a tuple in the referenced relation R2, or
  - (2) null
- In case (2), the FK in R1 should not be a part of its own primary key, and cannot have the NOT NULL constraint.

2.5. Other Types of Constraints

- Semantic Integrity Constraints:
  - cannot be expressed by the built-in model constraints
  - Example: “the max. no. of hours per employee for all projects he or she works on is 56 hours per week”
- A constraint specification language can be used to express these
- SQL has TRIGGERS and ASSERTIONS to express some of these constraints (see Section 5.2 CREATE ASSERTION, CREATE TRIGGER)
3. Update Operations, Transactions, and Dealing with Constraint Violations

- Each relation will have many tuples in its current relation state
- The relational database state is a union of all the individual relation states at a particular time
- Whenever the database is changed, a new state arises

Operations to Modify Relations

- Basic operations for changing the database:
  - INSERT new tuples in a relation
  - DELETE existing tuples from a relation
  - UPDATE attribute values of existing tuples
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together into a transaction.
- Updates may propagate to cause other updates automatically. This may be necessary to maintain integrity constraints.

Update Operations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

3.1. INSERT Operation

- INSERT one or more new tuples into a relation

Example:

```
```

See examples on pp. 76 & 77

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for a new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in a new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in a new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
if the primary key value is null in a new tuple

3.2. DELETE Operation

- DELETE one or more existing tuples from a relation
- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 4 for more details)
      - RESTRICT option: reject the deletion
      - CASCADE option: propagate the deletion by automatically deleting the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL (the foreign keys cannot have NOT NULL constraint)
  - One of the above options must be specified during database design for each referential integrity (foreign key) constraint

3.3. UPDATE Operation

- UPDATE modifies the values of attributes in one or more existing tuples in a relation
- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- Other constraints may also be violated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
    - The CASCADE option propagates the new value of PK to the foreign keys of the referencing tuples automatically
  - Updating a foreign key (FK) may violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain or NOT NULL constraints

3.4. Transaction Concept

- A transaction is an executing program that includes some database operations, such as reading from DB, or applying insertions, deletions, or updates to the DB.
- At the end of the transaction, it must leave the DB in a valid or consistent state (satisfies all the constraints specified in the DB schema).
- A single transaction may involve any number of retrieval operations and any number of update operations.
- A transaction form an atomic unit of work against the DB
  - Example: Bank withdrawal will typically read the user account record, check if there is sufficient balance, and then update the record by the withdrawal amount.
  - OLTP (online transaction processing) systems are executing transactions at rates that reach several hundred per second → issue: atomic processing, concurrent execution/synchronization, recovery from failures