

H2

The Relational Approach

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

[COMP23111 Handout 02 of 12]

Acknowledgements

- These slides are adaptations (mostly minor, but some major) of material authored and made available to instructors by **A. Silberschatz, H. F. Korth, and S. Sudarshan** to accompany their textbook **Database System Concepts, 6th Edition, McGraw-Hill, 2006, 978-0-07-352332-3** and **H. Garcia-Molina, J.D. Ullman, and J. Widom** to accompany their textbook **Database Systems: The Complete Book, 2nd Edition, Pearson Prentice Hall, 2009, 978-0-13-135428-9**
- Copyright remains with them, whom I thank.
- All errors are my responsibility.

In Previous Handouts

- We learned that data is an enterprise asset and that DBMSs, are crucial to manage it well.
- We learned the importance of adopting different levels of abstraction in designing and implementing databases.
- We learned how data models lead to schemas and instances that enable a logical view of the data.
- We started learning about the relational approach to data modelling, and we saw that conceptual modelling aims to bridge the gap between users and designers whilst still leading to good logical designs.
- We learned about the main components of DBMSs and the various architectures used to deploy them.

In This Handout

- We'll start to have a deeper look at the relational model that underpins the most widely-deployed DBMSs today.
- We'll also begin to look at how SQL implements the relational languages by looking at its DDL and DML facilities.

What are the basic notions in relational data modelling?

What is a Data Model?

- It is formal, often mathematical, representation of data.
- It has one or more structural collection type:
 - ▶ relational model = tables
 - ▶ semistructured model = trees/graphs
- Constraints (e.g., no composite columns, identifying column set per tuple, etc.)
- Operations on the collection elements (e.g., fetching only those tuples from a set that satisfy a given predicate)
- We'll only study the relational model in this course unit

Why Relations?

- Single collection type, so a very simple model
- Lead to simple, but surprisingly expressive, algebra (and calculi)
- Effective, since they often match how we think about data
- Lead to efficient implementations: they underlie the most important commercial DBMSs today
- Underlie SQL, the most widely-used language for interacting with databases today

The Relational Model: Tables, Rows, Columns

8

- Assume a university database, for one example.
- The main abstraction is regular, rectangular tables with typed, atomic columns
 - ▶ Think of the table name and the name/type of the columns as the schema
 - ▶ Think of the table (at a point in time) as an instance
- Each row represents an entity (e.g., a particular instructor)
 - ▶ The one with ID 15151 has name Mozart, belongs to the Music Department and earns 40K
- Each column stores the extension of an attribute of all the entities
 - ▶ The instructors' names are {Einstein, Wu, El Said, Katz, ...}

| <i>ID</i> | <i>name</i> | <i>dept_name</i> | <i>salary</i> |
|-----------|-------------|------------------|---------------|
| 22222 | Einstein | Physics | 95000 |
| 12121 | Wu | Finance | 90000 |
| 32343 | El Said | History | 60000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 76766 | Crick | Biology | 72000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 58583 | Califieri | History | 62000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 76543 | Singh | Finance | 80000 |

(a) The *instructor* table

The Relational Model: Relationships

9

- When columns in different tables represent the same concept (e.g., the name of a department), this captures relationships between entities.
- For example, between instructors and buildings:
 - ▶ In which building does Crick work?
- For another, between salaries and budgets:
 - ▶ Is Computer Science broke?!?
 - ▶ Indeed, which departments aren't?!?

| <i>ID</i> | <i>name</i> | <i>dept_name</i> | <i>salary</i> |
|-----------|-------------|------------------|---------------|
| 22222 | Einstein | Physics | 95000 |
| 12121 | Wu | Finance | 90000 |
| 32343 | El Said | History | 60000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 76766 | Crick | Biology | 72000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 58583 | Califieri | History | 62000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 76543 | Singh | Finance | 80000 |

(a) The *instructor* table

| <i>dept_name</i> | <i>building</i> | <i>budget</i> |
|------------------|-----------------|---------------|
| Comp. Sci. | Taylor | 100000 |
| Biology | Watson | 90000 |
| Elec. Eng. | Taylor | 85000 |
| Music | Packard | 80000 |
| Finance | Painter | 120000 |
| History | Painter | 50000 |
| Physics | Watson | 70000 |

(b) The *department* table

Relation as Tables

10

- Assume another database whose application domain (or universe of discourse, or mini-world) is beers from manufacturers, people who drink them and bars where they do so.
- A relation is a named table, e.g., in this application domain, we'll have one named **Beers**.
- A row is referred to as a tuple.
- Each row models an entity in the application domain, e.g., a particular beer, such as Bud Lite, manufactured by Anheuser-Busch.

| Beers | |
|------------|----------------|
| name | manf |
| Winterbrew | Pete's |
| Bud Lite | Anheuser-Busch |

Tuples as Rows, Attributes as Columns

11

- The columns headers are attribute names, e.g., **name** and **manf** (for manufacturer).
- Each column models the type (or domain) from which the corresponding property of an entity in the application domain can draw values, e.g., the names of beers in this case are strings such as '**Winterbrew**' and '**Bud Lite**'
- Each cell denotes the value taken for a particular attribute of a particular tuple, e.g., Bud Lite's manufacturer is Anheuser-Busch.

| Beers | |
|------------|----------------|
| name | manf |
| Winterbrew | Pete's |
| Bud Lite | Anheuser-Busch |

Relation Schemas and Database Schemas

12

- A relation schema is a relation name and attribute list, e.g.

`Beers(name, manf)`

- We can, and prefer to, declare the types of attributes:

`Beers(name:string, manf:string)`

- Since a database is a set of relations, the database schema is the set of all relation schemas in the database.
- We'll see later on what constraints are also part of the formal construction of the relational model.

What is SQL?

SQL :: Structured Query Language

14

- SQL stands for Structured Query Language
- It is pronounced, interchangeably:
 - ▶ 'sequel'
 - ▶ 'ess queue ell'
- The most widely-used QL +DML language
- An ANSI/ISO standard, but many incompatibilities in practice

SQL :: Structured Query Language

15

- SQL has an expressive QL subset
 - ▶ Based on the well-known **SELECT-FROM-WHERE-GROUP BY** idiom
- SQL has expressive DML capabilities:
 - ▶ Set-at-a-time (using query idioms)
 - ▶ Reactive triggers
 - ▶ Procedural constructs (e.g, loops, assignments, etc.)

SQL as a Language for Application Programming

16

- Application programs generally access databases through one of:
 - ▶ Extensions to general-purpose programming languages to allow embedded SQL statements
 - ▶ APIs (e.g., ODBC/JDBC) that allow SQL queries to be sent to a DBMS from within an application program

Example SQL Queries on the University DB

17

- Find the name of the instructor with ID 22222:

```
SELECT i.name  
FROM instructor i  
WHERE i.ID = '22222';
```

- Find the ID and building of instructors in the Physics department:

```
SELECT i.ID, d.building  
FROM instructor i, department d  
WHERE i.dept_name = d.dept_name  
AND d.dept_name = 'Physics';
```

Answers Through SQL

18

- In which building does Crick work?

```
SELECT d.building
FROM instructor i, department d
WHERE i.dept_name = d.dept_name
AND i.name = 'Crick';
```

- Is Computer Science broke?!?
- It is a simple question in English, but takes some work in SQL! (Can you do better? Try!)

```
SELECT DISTINCT 'BROKE!' AS answer
FROM department d
WHERE (SELECT SUM(i.salary)
      FROM instructor i, department d
      WHERE i.dept_name = d.dept_name
      GROUP BY d.dept_name
      HAVING d.dept_name = 'Comp. Sci.')
      >
      (SELECT d.budget
      FROM department d
      WHERE d.dept_name = 'Comp. Sci.');
```

the first subquery sums the salaries of all Computer Science instructors

the second subquery retrieves the value of the Computer Science budget

the outer query prints BROKE! if the sum of salaries is larger than the budget

SQL as a Data Definition Language: Implementing Relation Schemas in SQL

BBD DB :: A Running Example

20

- Assume again the Beers, Bars and Drinkers (BBD) DB.
- Tuples are constrained not to have the same value in the set of attribute values that identify each tuple.
- These attribute names form what is known as the primary key of the relation.
- The primary key can be, conventionally, underlined with a solid line.

Beers(name, manf)
Bars(name, addr, license)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

BBD DB :: A Running Example

21

- One relation may use the primary key of another relation to model a relationship.
- In this case, we call it a foreign key.
- Foreign keys can be, conventionally, underlined with a dashed line (unless it is also part of the primary key, in which case the underline remains solid).
- The foreign-key values in a relation S that refer to the primary key of a relation R must always occur in R (i.e., in every instance of R, i.e., no update on R can remove that value without there being consequences for its occurrence as a foreign key in S)
- Constraints such as these are another defining characteristic of a data model.

```
Beers(name, manf)
Bars(name, addr, license)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)
```

Database Schemas in SQL

22

- SQL is primarily a query language, for getting information from a database.
- But SQL also includes
 - ▶ a data definition component for describing database schemas
 - ▶ a data manipulation component for inserting, deleting and updating the current database (instance)

Database Schemas in SQL

- We'll focus first on the DDL (data definition language) subset for now, and consider the DML (data manipulation) subset later.
- DDL statements manipulate metadata (which is stored in the data dictionary).

SQL DDL :: Adding/Removing Relations

24

- To add a relation, we create a table
- The simplest form is:

```
CREATE TABLE <name>  
    (<list of elements>);
```

- To remove a relation, we drop the table:

```
DROP TABLE <name>;
```


SQL DDL :: Table Creation Elements

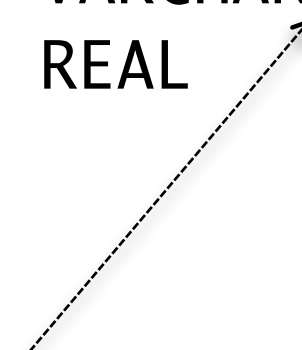
25

- The most basic element declares an attribute by asserting its name and type.
- The most common types are:

INTEGER
FLOAT
CHAR(n)
VARCHAR(n)

```
CREATE TABLE Sells (  
    bar    CHAR(20),  
    beer   VARCHAR2(20),  
    price  REAL  
);
```

This '2' here is an Oracle quirk
we must learn to live with



SQL DDL :: Table Creation Elements

26

- There are synonyms in some cases (e.g. **INT** for **INTEGER**, **REAL** for **FLOAT**).
- **CHAR(*n*)** declares a fixed-length string of *n* characters.
- **VARCHAR(*n*)** declares a variable-length string of up to *n* characters.
- For historical reasons, Oracle SQL only uses **VARCHAR2(*n*)**, which is not a standard synonym.

```
CREATE TABLE Sells (  
    bar    CHAR(20),  
    beer   VARCHAR2(20),  
    price  REAL  
);
```

SQL DDL :: Representing Values

27

- Integers and reals are represented as you would expect.
- So are strings, except that they require single quotes.
- If you do want a quote, use two single quotes, e.g.,

'Joe' 's Bar'

- Any value can be **NULL**.

SQL DDL :: Date and Time Types

28

- **DATE** and **TIME** are primitive types in SQL
- The form of a date value is:

`DATE 'yyyy-mm-dd'`

- For example:

`DATE '2012-10-02'`

denotes 2nd October 2012

SQL DDL :: Date and Time Types

29

- The form of a time value is:

`TIME 'hh:mm:ss'`

with an optional decimal point and fractions of a second following.

- For example, two and a half seconds after 3:30PM

`TIME '15:30:02.5'`

denotes two and a half seconds after 3:30 PM.

SQL DDL :: Declaring Keys

30

- Place **PRIMARY KEY** after the type in the declaration of the attribute.
- A key declaration can also be another element in the list of elements of a **CREATE TABLE** statement.
- This form is preferred because whilst it is the only form that works if the key consists of more than one attribute, it can also be used for one-attribute keys.
- For example, the primary key of **SELLS** is the concatenation of **bar** and **beer**.

```
CREATE TABLE Beers (  
    name  CHAR(20) PRIMARY KEY,  
    manf  CHAR(20)  
);
```

```
CREATE TABLE Sells (  
    bar      CHAR(20),  
    beer     VARCHAR2(20),  
    price    REAL,  
    PRIMARY KEY (bar, beer)  
);
```

SQL DDL :: Primary Key v. Unique

31

- A primary key takes unique values (i.e., no two tuples can have the same value for the primary key).
- There can be only one **PRIMARY KEY** declaration for a relation, but there can be several **UNIQUE** attributes.
- Note that **PRIMARY KEY** and **UNIQUE** are not semantically equivalent:
 - ▶ No attribute of a **PRIMARY KEY** can ever be **NULL** in any tuple.
 - ▶ Attributes declared **UNIQUE** may be **NULL**, and there may be several tuples with **NULL** in an attribute declared to be **UNIQUE**.

SQL DDL :: Altering a Relation Schema

32

- To add to or to remove existing attributes:

```
ALTER TABLE <name>  
ADD  
    <attribute name>  
    <attribute type>;
```

```
ALTER TABLE <name>  
DROP <attribute name>;
```


SQL DDL :: Altering a Relation Schema

33

- We can assign a **DEFAULT** value whenever we are allowed to declare

<attribute name> <attribute type>

- For example,

```
ALTER TABLE Beers  
ADD kind CHAR(5)  
    DEFAULT 'ale';
```

- If we add an attribute and do not specify a default value, it is set to **NULL** in all rows.

SQL as a Data Definition Language: Declaring and Enforcing Constraints

SQL DDL :: Declaring Constraints

35

- A constraint is a relationship among data elements that the DBMS is required to enforce.
- Some are application-independent, i.e., follow from the semantics of the data model.
 - ▶ We have seen three of those already:
 - the domain constraint (i.e., values are typed)
 - the primary key constraint
 - the foreign-key constraint
- Some are application-specific, e.g., that a manager must not earn less than a subordinate employee.

SQL DDL :: Types of Constraints

36

- Keys
 - ▶ a form of identity constraint
- Foreign keys
 - ▶ a form of referential-integrity constraint
- Domain
 - ▶ a form of typing constraint

SQL DDL :: Types of Constraints

37

- Tuple-based constraints
 - ▶ constrain one or more attributes to jointly stand in a given relationship
- Assertions
 - ▶ any SQL Boolean expression

SQL DDL :: Constraint Enforcement

38

- Data model constraints (i.e., application-independent ones) are enforced by the DBMS with no need for user involvement.
- Application-specific constraints can be left for the DBMS to enforce.
- However, it is often easier for the database designer simply to write them as triggers.
- Triggers are only executed when a specified event takes place, e.g., upon insertion of a tuple.
- We'll study them later in this course unit.

SQL DDL :: Declaring Foreign Keys

39

- Values appearing in attributes of one relation must appear together in certain attributes of another relation.

- For example, in

`Sells(bar, beer, price)`

we expect that a **beer** value that appears here must also appear as a value in **Beers.name**

- Because this is a composite key, we have the same expectation for **bar** and **Bars.name**

SQL DDL :: Declaring Foreign Keys

40

```
CREATE TABLE Sells (  
    bar      CHAR(20)      REFERENCES Bars(name),  
    beer     VARCHAR2(20)  REFERENCES Beers(name),  
    price    REAL,  
    PRIMARY KEY (bar, beer)  
);
```

```
CREATE TABLE Sells (  
    bar      CHAR(20),  
    beer     VARCHAR2(20),  
    price    REAL,  
    PRIMARY KEY (bar, beer)  
    FOREIGN KEY (bar) REFERENCES Bars(name),  
    FOREIGN KEY (beer) REFERENCES Beers(name)  
);
```

```
CREATE TABLE Sells (  
    bar      CHAR(20),  
    beer     VARCHAR2(20),  
    price    REAL,  
    CONSTRAINT s_pk PRIMARY KEY (bar, beer)  
    CONSTRAINT s_fk_bar FOREIGN KEY (bar)  
        REFERENCES Bars(name),  
    CONSTRAINT s_fk_beer FOREIGN KEY (beer)  
        REFERENCES Beers(name)  
);
```

- We use the keyword **REFERENCES**:
 - ▶ Either after an attribute (for one-attribute keys)

- ▶ Or as another element of the schema:

```
FOREIGN KEY (<list of attributes>)  
REFERENCES <relation> (<attributes>)
```

- ▶ Or, as preferred, by additionally giving the constraint a name prefixed by the keyword **CONSTRAINT**
- The referenced attributes must have been declared as **PRIMARY KEY** (or **UNIQUE**).

SQL DDL :: Policies for Enforcing FK Constraints

41

- If there is a foreign-key constraint from relation R to relation S , two violations are possible:
 - ▶ An insert or update to R introduces values not found in S .
 - ▶ A deletion or update to S causes some tuples of R to “dangle”, i.e., lose the value they referred to.

SQL DDL :: Policies for Enforcing FK Constraints

42

- An insert or update to R that introduces an entity that does not already exist in S must be rejected.
- A deletion or update to S that removes a foreign-key value found in some tuples of R can be handled in one of three ways:
 - ▶ rejection
 - ▶ propagation, or
 - ▶ nullification.

SQL DDL :: Policies for Enforcing FK Constraints

43

- Rejection of the modification is the default policy
- Propagation is referred to as cascading.
- There are two cases:
 - ▶ If a tuple in S was deleted, delete the R tuples that referred to it.
 - ▶ If a tuple in S was updated, update the value in the R tuples that refer to it.
- Nullification is implemented by changing the beer name to **NULL**.

SQL DDL:: Enforcing FK Constraints Through Cascading

44

- Suppose $R = \mathbf{Sells}$ and $S = \mathbf{Beers}$.
- If the Bud tuple is deleted from **Beers**, then delete all tuples from **Sells** where **beer** = 'Bud'.
- If the Bud tuple in **Beers** is changed so that the name 'Bud' is now 'Budweiser', then change all tuples in **Sells** where **beer** = 'Bud' to **beer** = 'Budweiser'.

- Again, suppose $R = \mathbf{Sells}$ and $S = \mathbf{Beers}$.
- If the Bud tuple is deleted from **Beers**, then change all tuples in **Sells** where **beer** = 'Bud' to **beer** = NULL.
- If the Bud tuple in **Beers** is changed so that the name 'Bud' is now 'Budweiser', then do the same as above.

SQL DDL :: Declaring a Policy

46

```
CREATE TABLE Sells (  
    bar      CHAR(20),  
    beer     VARCHAR2(20),  
    price    REAL,  
    CONSTRAINT s_pk  
        PRIMARY KEY (bar, beer)  
    CONSTRAINT s_fk_bar  
        FOREIGN KEY (bar)  
            REFERENCES Bars(name)  
            ON DELETE SET NULL  
            ON UPDATE CASCADE,  
    CONSTRAINT s_fk_beer  
        FOREIGN KEY (beer)  
            REFERENCES Beers(name)  
            ON DELETE SET NULL  
            ON UPDATE CASCADE,  
);
```

- When we declare a foreign key, we may override the default rejection and choose between **SET NULL** or **CASCADE** independently for deletions and updates.
- For each case, we add, or not, one **ON** clause to the foreign-key declaration.
- The **ON** clause has the form:

```
ON [UPDATE | DELETE]  
   [SET NULL | CASCADE]
```

SQL as a Data Manipulation Language: Changing the Database State

SQL DML :: Insert, Update and Delete

48

- A manipulation/modification SQL statement does not return a result (as a query does), but changes the database in some way.
- Three kinds of modifications are supported:
 - ▶ Insert a tuple or tuples.
 - ▶ Delete a tuple or tuples.
 - ▶ Update the value(s) of an existing tuple or tuples.

SQL as a Data Manipulation Language: Inserting Tuples

SQL DML :: Positional Form for Single-Tuple Insertion

50

- To insert a single tuple, we use:

```
INSERT INTO <relation>  
VALUES (<list of values>);
```

- For example, to add to **Likes(drinker, beer)** the fact that Sally likes Bud, we use:

```
INSERT INTO Likes  
VALUES ('Sally', 'Bud');
```

SQL DML :: Named Form for Single-Tuple Insertion

51

- We may add to the relation name a list of attributes.
- Two reasons to do so:
 - ▶ We may forget the standard order of attributes for the relation.
 - ▶ We don't have values for all attributes, and we want the system to fill in missing components with **NULL** or a default value.

SQL DML :: Named Form for Single-Tuple Insertion

52

- Another way to add the fact that Sally likes Bud to **Likes(drinker, beer)**:

```
INSERT INTO  
Likes(beer, drinker)  
VALUES ('Bud', 'Sally' );
```

- Note the variation in the schema-specified order

- In a **CREATE TABLE** statement, we can follow an attribute by **DEFAULT** and a value.
- When an inserted tuple has no value for that attribute, the default will be used.

SQL DML :: Using Default Values

54

- If **Drinkers** is specified as:

```
CREATE TABLE Drinkers (  
    name CHAR(30) PRIMARY KEY,  
    addr CHAR(50)  
        DEFAULT '123 Sesame St.',  
    phone CHAR(16));
```

- then, the insertion:

```
INSERT INTO Drinkers(name)  
VALUES ('Sally');
```

- inserts the tuple:

```
('Sally', '123 Sesame St', NULL)
```

- We may insert the entire result of a query into a relation, using the form:

```
INSERT INTO <relation>  
    ( <subquery> );
```

SQL DML :: Tuple-Set Insertion

56

- Using `Frequents(drinker, bar)`, enter into the new relation `PotBuddies(name)` all of Sally's potential buddies, i.e., those drinkers who frequent at least one bar that Sally also frequents.
- This is an example of a self-join (i.e., relating a table with itself, typically picking some rows in one input and other rows in the other input).
- We'll see more of this later.
- Note how the use of range variables **d1** and **d2** is crucial here.

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Sally' AND
        d2.drinker <> 'Sally' AND
        d1.bar = d2.bar
);
```

d1 is made to point to Sally

d2 is made to point to all the other drinkers

This ensures that Sally (d1) and the other drinker (d2) frequent the same bar.

SQL as a Data Manipulation Language: Deleting Tuples

SQL DML :: Deleting Some of the Tuples

58

- To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>  
        WHERE <condition>;
```

SQL DML :: Deleting All of the Tuples

59

- To make the relation **Likes** the empty set of tuples:

```
DELETE FROM Likes;
```

- Note no **WHERE** clause used, so all tuples satisfy the deletion condition.

SQL DML :: Deleting Some of the Tuples

60

- Delete from **Beers(name, manf)**
all beers for which there is another beer by the same manufacturer.
- Beers with the same manufacturer and a different name from the name of the beer, represented by tuple **b1**, are deleted.
- Again the range variables **b1** and **b2** are essential.
- This is called a correlated (or synchronized) sub-query, i.e., one that uses values (from tuple **b1**) from the outer query in the **WHERE** clause of an inner query.

```
DELETE FROM Beers b1
WHERE EXISTS (SELECT name
               FROM Beers b2
               WHERE b2.manf=b1.manf
                  AND b2.name<>b1.name);
```

SQL DML :: Semantics of Deletion

61

- Suppose Anheuser-Busch makes only Bud and Bud Lite.
- Suppose we come to the tuple **b1** for Bud first.
- The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- Now, when **b1** is the tuple for Bud Lite, do we delete that tuple too?

- The answer is yes, we do delete Bud Lite as well, because deletion proceeds in two stages:
 1. Mark all tuples for which the **WHERE** condition is satisfied.
 2. Delete the marked tuples.

SQL as a Data Manipulation Language: Modifying Tuples

SQL DML :: Changing Values in Tuples

64

- To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>  
    SET <list of attribute assignments>  
    WHERE <condition on tuples>;
```


SQL DML :: Changing Values in Tuples

65

- Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers  
    SET phone = '555-1212'  
    WHERE name = 'Fred';
```

- Make \$4 the maximum price for beer:

```
UPDATE Sells  
    SET price = 4.00  
    WHERE price > 4.00;
```

Summary

The Relational Model

67

- Simply defined, the relational model represents data as tables, each of which corresponds to a class (e.g., beers, bars) described by properties that are represented as columns (e.g., the price of a beer).
- We refer to naming of a table along with the definition of its columns and their type as a relation schema. A relational database is a collection of relation schemas.
- Each row in a table is an instance of the relation schema.
- The relational model supports referential constraints by means of primary and foreign keys.
- A referential constraint semantically links two tables. For example, if a beer has stopped being manufactured, there should be no bar that claims to sell it.
- SQL is a query language, with a definition and a manipulation subset.
- SQL supports relational integrity maintenance rejecting or propagating changes.

In The Next Handout

- We'll delve into the relational algebra.
- We'll look into how SQL can be mapped into the relational algebra.
- We'll explore how this declarative-to-procedural transformation is a key one in DBMSs.