Manipulating Databases

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

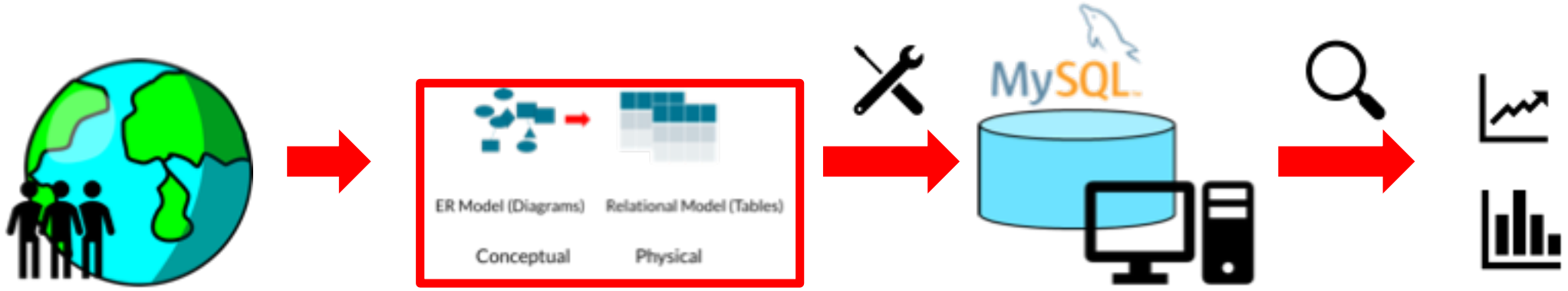
- Explain what manipulating databases means and why.
- Identify the basic “CRUD” functions of DBMS.
- State the general purposes of database manipulation.
- Describe the tools and languages for manipulating databases.

Why Do We Build a Database?

- *So we can capture the world.*
- Then why do we capture the world?
- *So we can support our applications.*
- Then how does it support our applications?
- *It lets us **ask questions** (that our applications need).*
- Then how can a database answer our questions?

- Because we carefully **manipulate** it to make it **consistent** and **up to date** with the real world.

Data Management Process



Real World

Data Modeling

Manipulating Database

Querying Database

Basic Functions of DBMS: CRUD

- **Create**
  - Creating new data elements.
- **Read**
  - Reading data by asking questions.
- **Update**
  - Updating data elements.
- **Delete**
  - Deleting data elements.

- Data elements: Database, table, schema, tuple, attribute.
- Our view: **CRUD** = **Manipulation** (Create, Update, and Delete) + **Querying** (Read).

Manipulating Databases: By Purposes

- **Organizing** Databases
  - Create and delete data “spaces”– database, schema, table.
- **Modifying** Databases
  - Insert, delete, update tuples to a table.
- **Augmenting** Databases
  - Create supporting (redundant) data elements: Indexes and views.
- **Regulating** Databases
  - Creating constraints (rules) that must maintain true all the time.

How Do We Perform Manipulations?

- System:
  - Console software: DBMS provides native interface via shell programs
    - MySQL Command-Line Tool “mysql”.
    - PostgreSQL interactive terminal “psql”.
  - SQL clients: Allowing users to manipulate/query databases using SQL.
- Language:
  - Every DBMS provides languages for manipulating/querying it.
  - For relational databases– the standard SQL includes
    - Querying commands: SELECT-FROM-WHERE, GROUP BY, …
    - Manipulating commands: CREATE, INSERT, UPDATE, DROP, DELETE, …

```
kevinccchang=# UPDATE Sells s
kevinccchang-# SET price = 0
kevinccchang-# WHERE price = (SELECT MAX(price) FROM Sells WHERE beer = s.beer)
kevinccchang-# ;
UPDATE 3
kevinccchang=# SELECT * FROM Sells;
          bar          |          beer          | price | discount
-----------------------+------------------------+-------+----------
 Green Bar             | Bud                    |     2 |        0
 Green Bar             | Sam Adams              |   4.5 |      0.2
 Purple Bar            | Bud                    |   4.5 |      0.2
 Purple Bar            | Sam Adams              |     5 |      0.3
 Sober Bar             | Bud                    |     0 |     0.05
 Sober Bar             | Bud Lite               |     0 |        0
 Sober Bar             | Sam Adams              |     0 |      0.1
(7 rows)
```

Screenshot of MySQL Command Line Tool

Screenshot of MySQL Workbench, 2018. Retrieved from https://www.mysql.com

Screenshot of Dbeaver, 2018. Retrieved from https://dbeaver.jkiss.org/

Organizing Databases

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

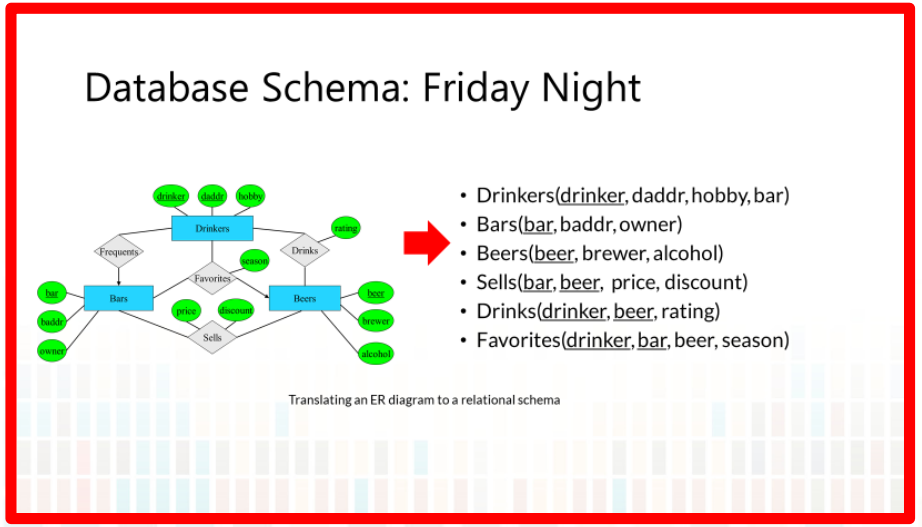
Learning Objectives

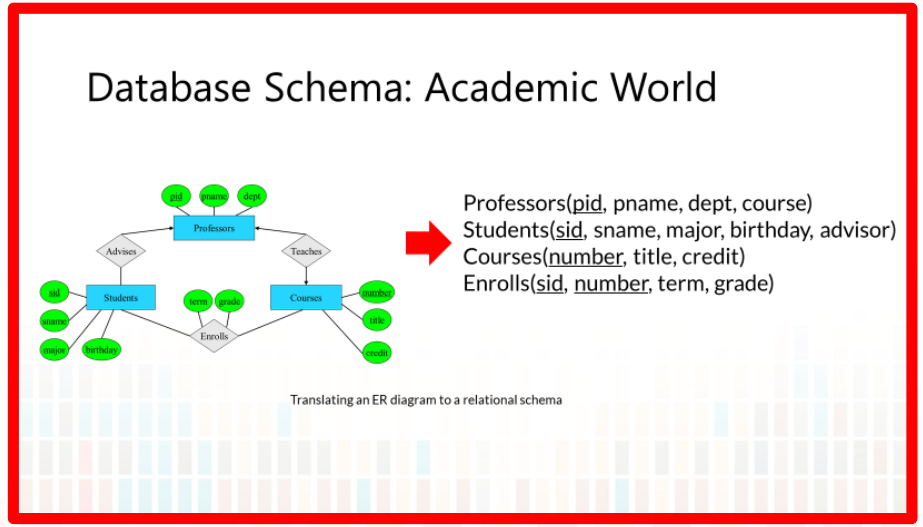
By the end of this video, you will be able to:

- Explain how we organize databases.
- Describe how to manipulate databases and tables.
- Create databases and tables in a relational database.

Organize Our Space of Data: Database

- A space of data that we manage for our applications.
- A set of tables that capture the data in your application domain.
- Different names or non-standard meanings at different systems.





Example databases

Database Organization:
Database (→ *Schema*)→ *Table*

- Most databases support three levels:
  - *Database* → *Schema* → *Table*
  - E.g., Oracle, PostgreSQL

  - The term “schema” is overloaded.

A database contains one or more named *schemas*, which in turn contain tables. Schemas also contain other kinds of named objects, including data types, functions, and operators. The same object name can be used in different schemas without conflict; for example, both `schema1` and `myschema` can contain tables named `mytable`. Unlike databases, schemas are not rigidly separated: a user can access objects in any of the schemas in the database they are connected to, if they have privileges to do so.

There are several reasons why one might want to use schemas:

- To allow many users to use one database without interfering with each other.
- To organize database objects into logical groups to make them more manageable.
- Third-party applications can be put into separate schemas so they do not collide with the names of other objects.

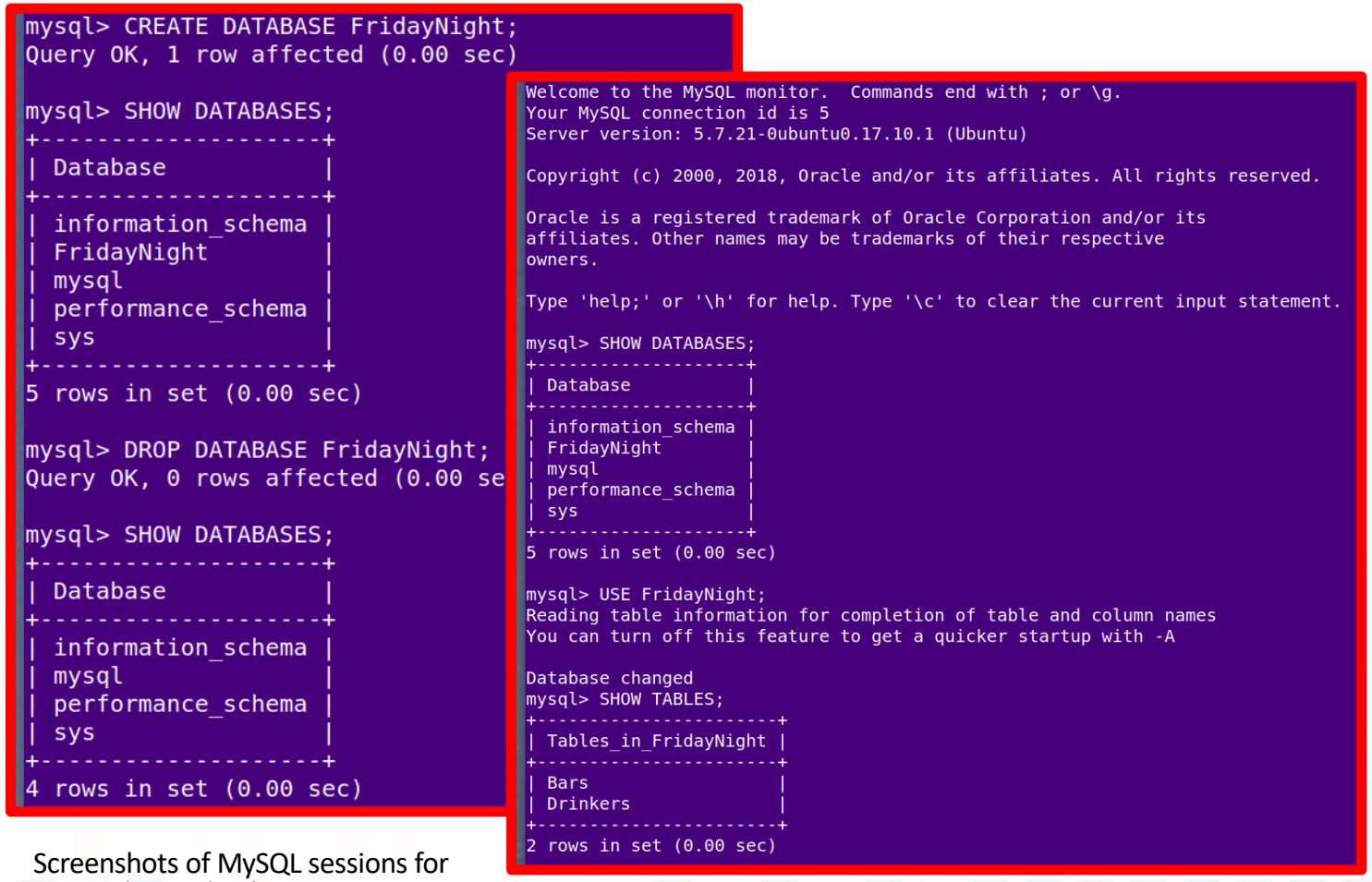
Schemas are analogous to directories at the operating system level, except that schemas cannot be nested.

Schemas – Data Definitions, PostgreSQL, 2018.
Retrieved from https://www.postgresql.org/docs/10/static/ddl-schemas.html

- Some support two levels.
  - *Database* → *Table*
  - E.g., MySQL

Organizing Databases

- Use MySQL as an example.
- Create a database
  CREATE DATABASE <name>;
- Show all databases
  SHOW DATABASES;
- Delete a database
  DROP DATABASE <name>;
- Select a database
  USE <name>;



Screenshots of MySQL sessions for manipulating databases

Organizing Tables

- Create a table

  CREATE TABLE <name>

  (<elements>);
- Show all tables

  SHOW TABLES;
- Show the schema of a table

  DESCRIBE TABLE <name>;
- Change the schema

  ALTER TABLE <name>

  ADD/DROP <attribute>;
- Delete a table

  DROP TABLE <name>;

```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_FridayNight |
+----------------------+
| Bars                 |
| Drinkers             |
+----------------------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE Beers (
    ->        name char(30) PRIMARY KEY,
    ->        brewer char(30),
    ->        alcohol float
    ->        );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE Sells (
    ->        bar char(30),
    ->        beer char(30),
    ->        price float,
    ->        discount float,
    ->        PRIMARY KEY (bar, beer)
    ->        );
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW TABLES;
+----------------------+
| Tables_in_FridayNight |
+----------------------+
| Bars                 |
| Beers                |
| Drinkers             |
| Sells                |
+----------------------+
4 rows in set (0.00 sec)

mysql> DROP TABLE Sells;
Query OK, 0 rows affected (0.01

mysql> SHOW TABLES;
+----------------------+
| Tables_in_FridayNight |
+----------------------+
| Bars                 |
| Beers                |
| Drinkers             |
+----------------------+
3 rows in set (0.00 sec)
```

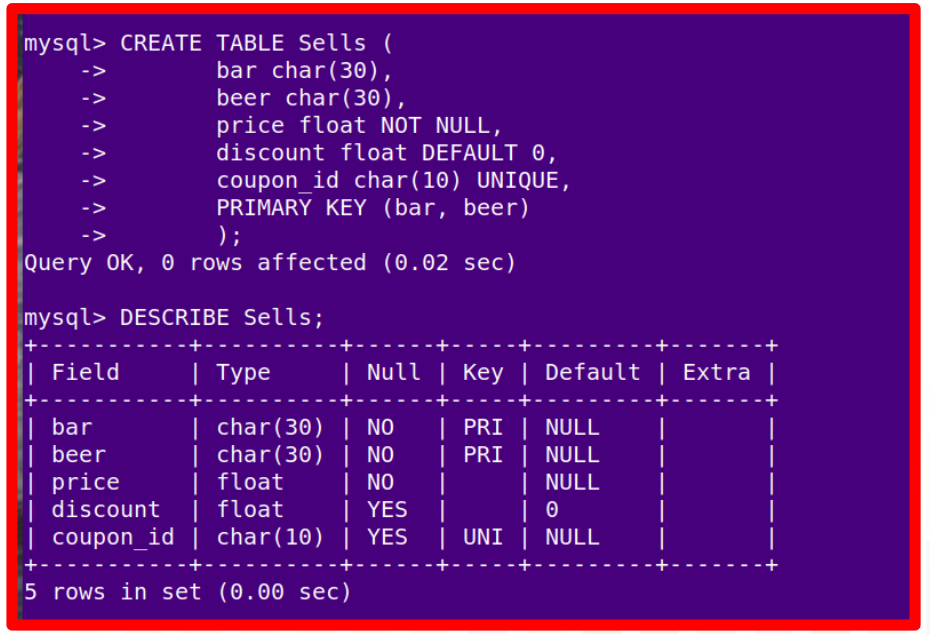
```
mysql> DESCRIBE Beers;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| name    | char(30) | NO   | PRI | NULL    |       |
| brewer  | char(30) | YES  |     | NULL    |       |
| alcohol | float    | YES  |     | NULL    |       |
+---------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> DESCRIBE Sells;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| bar      | char(30) | NO   | PRI | NULL    |       |
| beer     | char(30) | NO   | PRI | NULL    |       |
| price    | float    | YES  |     | NULL    |       |
| discount | float    | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Screenshots of MySQL sessions for manipulating tables

Creating a Table

- We create a table by declaring its schema.
  CREATE TABLE <name> (<elements>);
- Element: Attribute + Constraint:
  - Attribute: Name, type
    - E.g.: bar char(30).
  - Constraint: Optional, including
    - PRIMARY KEY, UNIQUE key, FOREIGN KEY
    - NOT NULL
    - DEFAULT <value>
  - Key constraints can be separate elements
    - Necessary for multi-attribute keys.
      - E.g.: PRIMARY KEY (bar, beer).



Screenshot of a MySQL session for creating a table

Constraints Are Part of Database Creation

- A **constraint** is a relationship among data elements that the DBMS is required to enforce.

- Types of constraints:
  - **Key constraints**: Constrain attributes to serve as some “keys” (identifiers)-- primary key, unique key, foreign key.
  - **Attribute-based constraints**.
    - Constrain values of an attribute to meet some criteria.
  - **Tuple-based constraints**.
    - Constrain relationship among components in a tuple.
  - **Database-based constraints**: Constrain relationship among elements any in a database.

Changing Schema of Table

- Add attributes
  ALTER TABLE <name>
  ADD <attribute declaration>;

- Delete attributes
  ALTER TABLE <name>
  DROP COLUMN <attribute>;

- Modify attributes
  ALTER TABLE <name>
  MODIFY COLUMN <attribute declaration>;

```
mysql> DESCRIBE Sells;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| bar       | char(30) | NO   | PRI | NULL    |       |
| beer      | char(30) | NO   | PRI | NULL    |       |
| price     | float    | NO   |     | NULL    |       |
| discount  | float    | YES  |     | 0       |       |
| coupon_id | char(10) | YES  | UNI | NULL    |       |
+-----------+----------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> ALTER TABLE Sells ADD size int NOT NULL;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Sells DROP COLUMN discount;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Sells MODIFY COLUMN size float DEFAULT 6.0;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE Sells;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| bar       | char(30) | NO   | PRI | NULL    |       |
| beer      | char(30) | NO   | PRI | NULL    |       |
| price     | float    | NO   |     | NULL    |       |
| coupon_id | char(10) | YES  | UNI | NULL    |       |
| size      | float    | YES  |     | 6       |       |
+-----------+----------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Screenshot of a MySQL session for changing table schema

Modifying Databases

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Identify different actions for modifying a table.
- Perform insertion, deletion, and update to a table.
- Describe the semantics of updates in SQL.
- Observe that different databases implement the semantics non-uniformly.

Modifying a Table

- Modify a database to keep it up to date with the real world.
- We modify a database by
  - Insertion: Add a tuple.
  - Deletion: Delete a tuple.
  - Update : Modify a tuple.

Updates: Changing Instances

- The database maintains a current database state.
- Updates to instance: Frequent.
  - Add a tuple.
  - Delete a tuple.
  - Modify a tuple.

```
mysql> INSERT INTO Students VALUES ('1', 'Bugs Bunny', 'CS', '2004-11-06');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('2', 'Donald Duck', 'Bio', '1997-02-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('3', 'Peter Pan', 'Econ', '1998-10-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('4', 'Mickey Mouse', 'CS', '1995-04-01');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM Students;
+------+-------------+-------+------------+
| id   | name        | major | birthday   |
+------+-------------+-------+------------+
| 1    | Bugs Bunny  | CS    | 2004-11-06 |
| 2    | Donald Duck | Bio   | 1997-02-01 |
| 3    | Peter Pan   | Econ  | 1998-10-01 |
| 4    | Mickey Mouse| CS    | 1995-04-01 |
+------+-------------+-------+------------+
4 rows in set (0.00 sec)
```

Screenshot of a MySQL session for updating database instances

Insertion with Values or Query Results

- Insert tuples with some or all attribute values

  INSERT $R\ (A_1, \ldots, A_n)$ ← Relation and attributes to insert into.

  VALUES $(v_{11}, \ldots, v_{1n}), \ldots, (v_{m1}, \ldots, v_{mn})$; ← Values to insert.
- Can omit attribute names if the same as in schema definition.

- Insert the results of queries

  INSERT $R\ (A_1, \ldots, A_n)$ ← Relation and attributes to insert into.

  (SUBQUERY); ← Query results to insert.

Insertion Examples

- *Q1: Add Goose Island IPA by Goose Island to Beers.*
- *Q2: Add Eileen, address Purple St, hobby Running, and frequent Purple Bar to Drinkers.*

```
mysql> SELECT * FROM Beers;
+-----------+-------------+---------+
| name      | brewer      | alcohol |
+-----------+-------------+---------+
| Bud       | AB InBev    |    0.05 |
| Bud Lite  | AB InBev    |   0.042 |
| Coors     | Coors       |    0.05 |
| Sam Adams | Boston Beer |   0.049 |
+-----------+-------------+---------+
4 rows in set (0.00 sec)

mysql> INSERT INTO Beers(name, brewer) VALUES ("Goose Island IPA", "Goose Island");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Beers;
+------------------+--------------+---------+
| name             | brewer       | alcohol |
+------------------+--------------+---------+
| Bud              | AB InBev     |    0.05 |
| Bud Lite         | AB InBev     |   0.042 |
| Coors            | Coors        |    0.05 |
| Goose Island IPA | Goose Island |    NULL |
| Sam Adams        | Boston Beer  |   0.049 |
+------------------+--------------+---------+
5 rows in set (0.00 sec)
```

```
mysql>
mysql> SELECT * FROM Drinkers;
+-------+----------+---------+-----------+
| name  | addr     | hobby   | frequent  |
+-------+----------+---------+-----------+
| Alex  | Green St | Reading | Sober Bar |
| Betty | King St  | Singing | Green Bar |
| Cindy | Green St | Hiking  | Green Bar |
| Dan   | Queen St | NULL    | NULL      |
+-------+----------+---------+-----------+
4 rows in set (0.00 sec)

mysql> INSERT INTO Drinkers VALUES ("Eileen", "Purple St", "Running", "Purple Bar");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Drinkers;
+--------+-----------+---------+------------+
| name   | addr      | hobby   | frequent   |
+--------+-----------+---------+------------+
| Alex   | Green St  | Reading | Sober Bar  |
| Betty  | King St   | Singing | Green Bar  |
| Cindy  | Green St  | Hiking  | Green Bar  |
| Dan    | Queen St  | NULL    | NULL       |
| Eileen | Purple St | Running | Purple Bar |
+--------+-----------+---------+------------+
5 rows in set (0.00 sec)
```

Screenshots of insertion examples in MySQL

Insertion Examples

- *Q3: Add Purple Bar, Bud Lite to Sells.*
- *Q4: Add Purple Bar, Bud Lite, $3.5 to Sells.*
- *Q5: Add to Beers all beers on the market but not in Beers.*

```
mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)

mysql> INSERT INTO Sells(bar, beer) VALUES ("Purple Bar", "Bud Heavy");
ERROR 1364 (HY000): Field 'price' doesn't have a default value
mysql> INSERT INTO Sells(bar, beer, price) VALUES ("Purple Bar", "Bud Heavy", 3.
5);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Bud Heavy |   3.5 |        0 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
8 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Beers;
+------------------+--------------+---------+
| name             | brewer       | alcohol |
+------------------+--------------+---------+
| Bud              | AB InBev     |    0.05 |
| Bud Lite         | AB InBev     |   0.042 |
| Coors            | Coors        |    0.05 |
| Goose Island IPA | Goose Island |    NULL |
| Sam Adams        | Boston Beer  |   0.049 |
+------------------+--------------+---------+
5 rows in set (0.00 sec)

mysql> INSERT INTO Beers(name) SELECT beer FROM Sells s WHERE NOT EXISTS (SELECT * FROM Beers WH
ERE name = s.beer);
Query OK, 1 row affected (0.00 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Beers;
+------------------+--------------+---------+
| name             | brewer       | alcohol |
+------------------+--------------+---------+
| Bud              | AB InBev     |    0.05 |
| Bud Heavy        | NULL         |    NULL |
| Bud Lite         | AB InBev     |   0.042 |
| Coors            | Coors        |    0.05 |
| Goose Island IPA | Goose Island |    NULL |
| Sam Adams        | Boston Beer  |   0.049 |
+------------------+--------------+---------+
6 rows in set (0.00 sec)
```

Screenshots of insertion examples in MySQL

Deletion

- Delete tuples that match some condition

  DELETE FROM <name>

  WHERE <condition>;

- If WHERE is omitted: Delete all!

  DELETE FROM <name>;

Deletion Examples

- *Q1: Delete Eileen from Drinkers.*
- *Q2: Delete any beers in Beers that are not sold in any bars.*
- *Q3: No beers allowed: Delete all beers from Sells.*

```
mysql> SELECT * FROM Drinkers;
+--------+-----------+---------+------------+
| name   | addr      | hobby   | frequent   |
+--------+-----------+---------+------------+
| Alex   | Green St  | Reading | Sober Bar  |
| Betty  | King St   | Singing | Green Bar  |
| Cindy  | Green St  | Hiking  | Green Bar  |
| Dan    | Queen St  | NULL    | NULL       |
| Eileen | Purple St | Running | Purple Bar |
+--------+-----------+---------+------------+
5 rows in set (0.00 sec)

mysql> DELETE FROM Drinkers WHERE name = "Eileen";
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Drinkers;
+-------+----------+---------+----------+
| name  | addr     | hobby   | frequent |
+-------+----------+---------+----------+
| Alex  | Green St | Reading | Sober Bar |
| Betty | King St  | Singing | Green Bar |
| Cindy | Green St | Hiking  | Green Bar |
| Dan   | Queen St | NULL    | NULL      |
+-------+----------+---------+----------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Beers;
+-----------------+-------------+---------+
| name            | brewer      | alcohol |
+-----------------+-------------+---------+
| Bud             | AB InBev    |    0.05 |
| Bud Heavy       | NULL        |    NULL |
| Bud Lite        | AB InBev    |   0.042 |
| Coors           | Coors       |    0.05 |
| Goose Island IPA | Goose Island |    NULL |
| Sam Adams       | Boston Beer |   0.049 |
+-----------------+-------------+---------+
6 rows in set (0.00 sec)

mysql> DELETE FROM Beers WHERE NOT EXISTS (SELECT * FROM Sells WHERE beer = name);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM Beers;
+----------+------------+---------+
| name     | brewer     | alcohol |
+----------+------------+---------+
| Bud      | AB InBev   |    0.05 |
| Bud Heavy | NULL       |    NULL |
| Bud Lite | AB InBev   |   0.042 |
| Sam Adams | Boston Beer |   0.049 |
+----------+------------+---------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Bud Heavy |   3.5 |        0 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
8 rows in set (0.00 sec)

mysql> DELETE FROM Sells;
Query OK, 8 rows affected (0.00 sec)

mysql> SELECT * FROM Sells;
Empty set (0.00 sec)
```

Screenshots of deletion examples in MySQL

Updates

- Change certain attributes in certain tuples of a relation:

  UPDATE <name>

  SET <list of attribute assignments>

  WHERE <condition>;

Update Examples

- *Q1: Set Dan's hobby to Drinking.*
- *Q2: Raise price by 30% for beers less then $3.*
- *Q3: If a bar is selling a beer with the highest price of that beer on the market, make the the bar sell it for free (price = $0).*

```
mysql> SELECT * FROM Drinkers;
+-------+----------+---------+-----------+
| name  | addr     | hobby   | frequent  |
+-------+----------+---------+-----------+
| Alex  | Green St | Reading | Sober Bar |
| Betty | King St  | Singing | Green Bar |
| Cindy | Green St | Hiking  | Green Bar |
| Dan   | Queen St | NULL    | NULL      |
+-------+----------+---------+-----------+
4 rows in set (0.00 sec)

mysql> UPDATE Drinkers SET hobby = "Drinking" WHERE name = "Dan";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Drinkers;
+-------+----------+----------+-----------+
| name  | addr     | hobby    | frequent  |
+-------+----------+----------+-----------+
| Alex  | Green St | Reading  | Sober Bar |
| Betty | King St  | Singing  | Green Bar |
| Cindy | Green St | Hiking   | Green Bar |
| Dan   | Queen St | Drinking | NULL      |
+-------+----------+----------+-----------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)

mysql> UPDATE Sells SET price = price*1.3 WHERE price < 3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |   2.6 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)

mysql> UPDATE Sells s
    -> SET price = 0
    -> WHERE price = (SELECT MAX(price) FROM Sells WHERE beer = s.beer)
    -> ;
ERROR 1093 (HY000): You can't specify target table 's' for update in FROM clause
mysql>
```

Screenshots of update examples in MySQL

Semantics of Modification:
*What **SHOULD** This Query Do?*

UPDATE Sells s

SET price = 0

WHERE price =

(SELECT MAX(price) FROM Sells WHERE beer = s.beer)

```
kevinccchang=# SELECT * FROM Sells;
              bar               |              beer               | price | discount
--------------------------------+---------------------------------+-------+----------
 Sober Bar                      | Bud                             |     5 |     0.05
 Sober Bar                      | Bud Lite                        |     3 |        0
 Sober Bar                      | Sam Adams                       |     6 |      0.1
 Green Bar                      | Bud                             |     2 |        0
 Green Bar                      | Sam Adams                       |   4.5 |      0.2
 Purple Bar                     | Bud                             |   4.5 |      0.2
 Purple Bar                     | Sam Adams                       |     5 |      0.3
(7 rows)
```

Table Sells, executed in PostgreSQL

Standard Semantics of Modification: *What **SHOULD** This Query Do?*

1. Set price=0 for Sam Adams at Sober Bar, since it's highest.
2. Set price = 0 for Sam Adams at Purple Bar, since it becomes new highest.
3. The "cascading updates" can go on and on…

```
UPDATE Sells s
SET price = 0
WHERE price =
   (SELECT MAX(price) FROM Sells WHERE beer = s.beer)
```

```
kevinccchang=# SELECT * FROM Sells;
             bar              |              beer              | price | discount
------------------------------+--------------------------------+-------+----------
 Sober Bar                    | Bud                            |     5 |     0.05
 Sober Bar                    | Bud Lite                       |     3 |        0
 Sober Bar                    | Sam Adams                      |     6 |      0.1
 Green Bar                    | Bud                            |     2 |        0
 Green Bar                    | Sam Adams                      |   4.5 |      0.2
 Purple Bar                   | Bud                            |   4.5 |      0.2
 Purple Bar                   | Sam Adams                      |     5 |      0.3
(7 rows)
```

Table Sells, executed in PostgreSQL

Executing an update query over the Sells table in PostgreSQL

Standard Semantics of Modification: *"Deferred Updates"*

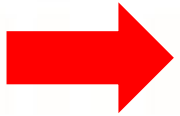
1. Evaluate WHERE expression and identify every tuple that needs to be updated.
2. Update the tuples identified.

15) The <value expression>s are effectively evaluated before updating the object row. If a <value expression> contains a reference to a column of *T* or to *T* itself, then the reference is to the value of that column or table in the object row before any value of the object row is updated.

590 (ISO-ANSI working draft) Database Language SQL (SQL/Foundation [SQL3])

SQL Standard draft, X3H2-94-329 DBL:RIO-004 August, 1994.

```
kevinccchang=# SELECT * FROM Sells;
             bar              |              beer              | price | discount
------------------------------+--------------------------------+-------+----------
 Sober Bar                    | Bud                            |     5 |     0.05
 Sober Bar                    | Bud Lite                       |     3 |        0
 Sober Bar                    | Sam Adams                      |     6 |      0.1
 Green Bar                    | Bud                            |     2 |        0
 Green Bar                    | Sam Adams                      |   4.5 |      0.2
 Purple Bar                   | Bud                            |   4.5 |      0.2
 Purple Bar                   | Sam Adams                      |     5 |      0.3
(7 rows)
```



```
kevinccchang=# UPDATE Sells s
kevinccchang-# SET price = 0
kevinccchang-# WHERE price = (SELECT MAX(price) FROM Sells WHERE beer = s.beer)
kevinccchang-# ;
UPDATE 3
kevinccchang=# SELECT * FROM Sells;
             bar              |              beer              | price | discount
------------------------------+--------------------------------+-------+----------
 Green Bar                    | Bud                            |     2 |        0
 Green Bar                    | Sam Adams                      |   4.5 |      0.2
 Purple Bar                   | Bud                            |   4.5 |      0.2
 Purple Bar                   | Sam Adams                      |     5 |      0.3
 Sober Bar                    | Bud                            |     0 |     0.05
 Sober Bar                    | Bud Lite                       |     0 |        0
 Sober Bar                    | Sam Adams                      |     0 |      0.1
(7 rows)
```

Executing an update query over the Sells table in PostgreSQL

Update-Semantics Examples

- *Run the following query at PostgreSQL and MySQL:*

```
UPDATE Sells s
SET price = 0
WHERE price =
    (SELECT MAX(price) FROM Sells WHERE beer = s.beer)
```

Food for Thought

The standard "deferred update" semantics is not uniformly implemented. At MySQL, we got an error message. Do you think it's a reasonable semantics?

```
mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)

mysql> UPDATE Sells s
    -> SET price = 0
    -> WHERE price = (SELECT MAX(price) FROM Sells WHERE beer = s.beer)
    -> ;
ERROR 1093 (HY000): You can't specify target table 's' for update in FROM clause
mysql>
```

Executing an update query over the Sells table in MySQL

Food for Thought

We want to make sure each beer is only sold at one bar.
So, what will this query do?

```
DELETE FROM Sells  s1
WHERE EXISTS (
        SELECT bar FROM Sells s2
        WHERE s2.beer = s1.beer and s2.bar <> s1.bar);
```

```
kevinccchang=# SELECT * FROM Sells;
          bar          |             beer             | price | discount
-----------------------+------------------------------+-------+----------
 Sober Bar             | Bud                          |     5 |     0.05
 Sober Bar             | Bud Lite                     |     3 |        0
 Sober Bar             | Sam Adams                    |     6 |      0.1
 Green Bar             | Bud                          |     2 |        0
 Green Bar             | Sam Adams                    |   4.5 |      0.2
 Purple Bar            | Bud                          |   4.5 |      0.2
 Purple Bar            | Sam Adams                    |     5 |      0.3
(7 rows)
```

Table Sells, executed in PostgreSQL

Results of query

Augmenting Databases: Indexes

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Explain what an index is, and why we need it.
- Manipulate indexes by creating or deleting them.
- Observe how indexes speed up queries.

Indexes: What and Why?

- An index is a “map” of a table, telling DBMS where different values are stored.
  - E.g., SELECT sid, grade FROM Enrolls WHERE number = “CS411” AND grade = “A”;
  - How to locate tuples with number “CS411”? Where are those “A” tuples?
- We need indexes for those attributes that are frequently used in queries, so that DBMS can process queries quickly.
  - But DBMS can still execute (maybe slower) without indexes.
- Users do not view or access indexes directly.
  - They are used by “query processor” in DBMS internally.

Creating an Index

- DBMS may create an index automatically for important attributes.
  - MySQL: The “PRIMARY” index, auto created from PRIMARY KEY attributes.
- Users can tell DBMS what attributes to create index for, since users anticipate what queries will be asked.

Manipulating Indexes

- Most DBMS provides ways for users to create indexes.
  - But syntax may be slightly different. We use MySQL here.
- Create an index: **CREATE INDEX** <name> on <table>(<attributes>);
- Show indexes:

  SHOW INDEX FROM <table>;
- Delete an index

  ALTER TABLE <table>

  DROP INDEX < name>;

```
mysql> SHOW INDEX FROM Sells;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Sells |          0 | PRIMARY  |            1 | bar         | A         |           3 |     NULL | NULL   |      | BTREE      |         |               |
| Sells |          0 | PRIMARY  |            2 | beer        | A         |           7 |     NULL | NULL   |      | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
2 rows in set (0.01 sec)

mysql> CREATE INDEX price_idx ON Sells(price);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW INDEX FROM Sells;
+-------+------------+-----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name  | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+-----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Sells |          0 | PRIMARY   |            1 | bar         | A         |           3 |     NULL | NULL   |      | BTREE      |         |               |
| Sells |          0 | PRIMARY   |            2 | beer        | A         |           7 |     NULL | NULL   |      | BTREE      |         |               |
| Sells |          1 | price_idx |            1 | price       | A         |           5 |     NULL | NULL   |      | BTREE      |         |               |
+-------+------------+-----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE Sells DROP INDEX price_idx;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW INDEX FROM Sells;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Sells |          0 | PRIMARY  |            1 | bar         | A         |           3 |     NULL | NULL   |      | BTREE      |         |               |
| Sells |          0 | PRIMARY  |            2 | beer        | A         |           7 |     NULL | NULL   |      | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
2 rows in set (0.00 sec)
```

Indexing examples in MySQL

How Will DBMS Use Indexes?

- DBMS will use indexes to construct more efficient “query plans”.

```
mysql> SHOW INDEX FROM Sells;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Sells |          0 | PRIMARY  |            1 | bar         | A         |     1097401 |     NULL | NULL   |      | BTREE      |         |               |
| Sells |          0 | PRIMARY  |            2 | beer        | A         |     1023765 |     NULL | NULL   |      | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
2 rows in set (0.00 sec)

mysql> EXPLAIN SELECT beer, bar FROM Sells WHERE price < 3;
+----+-------------+-------+------------+------+---------------+------+---------+------+---------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows    | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+---------+----------+-------------+
|  1 | SIMPLE      | Sells | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 1097401 |    33.33 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+---------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> SELECT beer, bar FROM Sells WHERE price < 3;
+------+-----------+
| beer | bar       |
+------+-----------+
| Bud  | Green Bar |
+------+-----------+
1 row in set (0.22 sec)
```

Running a query on “price” without an index in MySQL

```
mysql> SHOW INDEX FROM Sells;
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Sells |          0 | PRIMARY    |            1 | bar         | A         |     1097401 |     NULL | NULL   |      | BTREE      |         |               |
| Sells |          0 | PRIMARY    |            2 | beer        | A         |     1023765 |     NULL | NULL   |      | BTREE      |         |               |
| Sells |          1 | price_idx  |            1 | price       | A         |          18 |     NULL | NULL   |      | BTREE      |         |               |
+-------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
3 rows in set (0.00 sec)

mysql> EXPLAIN SELECT beer, bar FROM Sells WHERE price < 3;
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+--------------------------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered | Extra                    |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+--------------------------+
|  1 | SIMPLE      | Sells | NULL       | range | price_idx     | price_idx | 4       | NULL |    1 |   100.00 | Using where; Using index |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SELECT beer, bar FROM Sells WHERE price < 3;
+------+-----------+
| beer | bar       |
+------+-----------+
| Bud  | Green Bar |
+------+-----------+
1 row in set (0.00 sec)
```

Running a query on “price” with an index in MySQL

Food for Thought

Indexing seems so powerful to make queries run fast.
*Why not just automatically create an index for **every** attribute?*

Augmenting Databases: Views

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Define what views are and why we need them.
- Declare and use views in relational databases.
- Explain why views are not always updatable.

Views: What and Why?

- A view is a “virtual” table, a relation that is defined in terms of the contents of other tables (and views).
- We created tables via our data modeling process, to capture the essential needs of the applications.
- However, different users may have different “perspectives”, and they may want a particular subset organized in particular ways.
  - AcademicWorld:
    - Roles: Students vs. instructors vs. administrators.
    - Levels of concerns: CS department vs. EE department vs. College of Engineering.
- Views help to provide such different perspectives.

Creating and Querying a View

- Declare by:

  CREATE VIEW <name> **AS** <query>;

- To contrast, a relation whose value is really stored in the database is called a **base table**.

- You can “query” a view as if it were a base table.
  - But a view may not be “updatable”.

View Examples

- *Q1: Create a view GreenStDrinkers – those drinkers who live on Green St.*

```
mysql> CREATE VIEW GreenStDrinkers AS
    -> SELECT * FROM Drinkers WHERE addr = "Green St";
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM GreenStDrinkers;
+-------+----------+---------+-----------+
| name  | addr     | hobby   | frequent  |
+-------+----------+---------+-----------+
| Alex  | Green St | Reading | Sober Bar |
| Cindy | Green St | Hiking  | Green Bar |
+-------+----------+---------+-----------+
2 rows in set (0.00 sec)
```

CREAE VIEW query example in MySQL

View Examples

- *Q2: Create a view HappyDrinkers -- drinker and beer pairs for those drinkers who have bars on the same street they live and the bar sells those beers that they drink.*

```
mysql> CREATE VIEW HappyDrinkers AS
    ->   SELECT dr.name AS drinker, dk.beer
    ->   FROM Drinkers dr, Bars b, Sells s, Drinks dk
    ->   WHERE dr.addr = b.addr AND b.name = s.bar AND
    ->         s.beer = dk.beer AND dk.drinker = dr.name;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM HappyDrinkers;
+---------+-----------+
| drinker | beer      |
+---------+-----------+
| Alex    | Bud       |
| Alex    | Sam Adams |
+---------+-----------+
2 rows in set (0.00 sec)
```

CREAE VIEW query example in MySQL

View-Querying Examples

- *Q1: Find GreetStDrinkers who frequent Green Bar.*

```
mysql> SELECT name FROM GreenStDrinkers g WHERE g.frequent = "Green Bar";
+-------+
| name  |
+-------+
| Cindy |
+-------+
1 row in set (0.00 sec)
```

View-querying example in MySQL

- *Q2: Find HappyDrinkers whose beers are made by Boston Beer.*

```
mysql> SELECT h.drinker FROM HappyDrinkers h, Beers b WHERE h.beer = b.name AND b.brewer = "Boston Beer";
+---------+
| drinker |
+---------+
| Alex    |
+---------+
1 row in set (0.00 sec)
```

View-querying example in MySQL

Querying a View: What Happens When a View Is Used?

- **View Expansion:** DBMS will replace a view with its definition, thus turning the query into one with only base tables.
  - It will first rewrite the query and view definition both into some internal expressions, e.g., in relational algebra.
  - The view definition expression is then "sliced into" the query expression.

- SELECT name FROM GreenStDrinkers g WHERE g.frequent = "Green Bar";
  - **Query:** $Q = \pi_{name}(\sigma_{\text{frequent="Green Bar"}}(\text{GreenStDrinkers}))$
  - **View:** $\text{GreenStDrinkers} = \sigma_{\text{addr="Green St"}}(\text{Drinkers})$
  - **Query with View Expanded:**

$$Q' = \pi_{name}(\sigma_{\text{frequent="Green Bar"}}(\sigma_{\text{addr="Green St"}}(\text{Drinkers})))$$

Updating a View

- Can we insert a tuple into a “virtual” table?
- Not all views are “updatable”.
- GreenStDrinkers?

HappyDrinkers?

```
mysql> CREATE VIEW GreenStDrinkers AS
    -> SELECT * FROM Drinkers WHERE addr = "Green St";
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM GreenStDrinkers;
+-------+----------+---------+-----------+
| name  | addr     | hobby   | frequent  |
+-------+----------+---------+-----------+
| Alex  | Green St | Reading | Sober Bar |
| Cindy | Green St | Hiking  | Green Bar |
+-------+----------+---------+-----------+
2 rows in set (0.00 sec)
```

Example view GreenStDrinkers in MySQL

```
mysql> CREATE VIEW HappyDrinkers AS
    ->   SELECT dr.name AS drinker, dk.beer
    ->   FROM Drinkers dr, Bars b, Sells s, Drinks dk
    ->   WHERE dr.addr = b.addr AND b.name = s.bar AND
    ->         s.beer = dk.beer AND dk.drinker = dr.name;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM HappyDrinkers;
+---------+-----------+
| drinker | beer      |
+---------+-----------+
| Alex    | Bud       |
| Alex    | Sam Adams |
+---------+-----------+
2 rows in set (0.00 sec)
```

Example view HappyDrinkers in MySQL

Why Not All Views Are Updatable?

- Values of a view may not sufficiently determine values of its base tables.
- 1. Values may be missing.
  - CREATE VIEW BarHasBeer AS (SELECT bar, beer FROM Sells)
  - INSERT (“Green Bar”, “Goose Island IPA”)
    - What is price? Note that price must be NOT NULL in Sells.
- 2. Values cannot not be uniquely determined.
  - CREATE VIEW DeptGPA AS (SELECT major, AVERAGE(gpa) FROM Students GROUPBY major).
  - INSERT (“CS”, 3.0)
    - How to map 3.0 average GPA to every student in CS?
- Views from joins are more complicated, involving multiple base tables.
  - Cannot uniquely determine the values for each base table.

View-Updating Examples

- *Q1: Insert ("Gary", "Green St", "Singing", "Purple Bar") into GreenStDrinkers.*

```
mysql> INSERT INTO GreenStDrinkers VALUES ("Gary", "Green St", "Singing", "Purple Bar");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM GreenStDrinkers;
+-------+----------+---------+------------+
| name  | addr     | hobby   | frequent   |
+-------+----------+---------+------------+
| Alex  | Green St | Reading | Sober Bar  |
| Cindy | Green St | Hiking  | Green Bar  |
| Gary  | Green St | Singing | Purple Bar |
+-------+----------+---------+------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Drinkers;
+-------+----------+---------+------------+
| name  | addr     | hobby   | frequent   |
+-------+----------+---------+------------+
| Alex  | Green St | Reading | Sober Bar  |
| Betty | King St  | Singing | Green Bar  |
| Cindy | Green St | Hiking  | Green Bar  |
| Gary  | Green St | Singing | Purple Bar |
+-------+----------+---------+------------+
4 rows in set (0.00 sec)
```

View-updating example in MySQL

View-Updating Examples

- *Q2: Insert (Harry, "Bud Lite") into HappyDrinkers.*

```
mysql> INSERT INTO HappyDrinkers VALUES ("Harry", "Bud Lite");
ERROR 1394 (HY000): Can not insert into join view 'FridayNight.HappyDrinkers' without fields list
mysql>
```

View-updating example in MySQL

Not All Views Are “Updatable”

- GreenStDrinkers: Yes.
- HappyDrinkers: No.

- Complex rules to determine if a view is updatable, but intuitively:

  A view is updatable if it selects some **enough attribute** from **one relation**.
  - A view that joins multiple tables are not updatable.
  - A view with too few attributes is not updatable.

Food for Thought

Even when a view is updatable, its updates can be tricky. Try inserting (“Paul”, “Purple St”, “Riding”, “Green Bar”) to GreenStDrinkers.
Can you?

Regulating Databases: Key Constraints

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

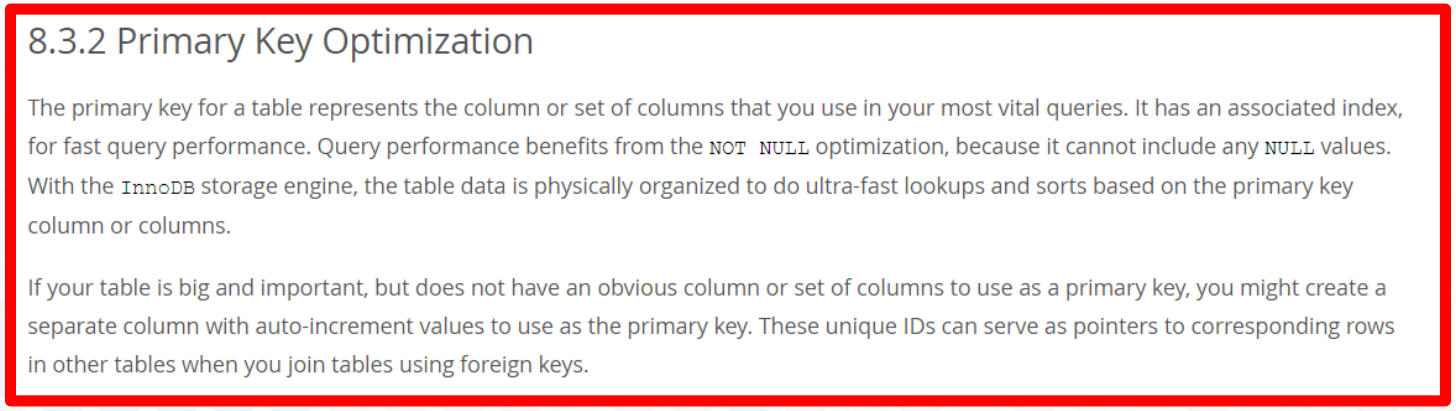
- Define key constraints and identify their different types.
- Explain the distinctions between primary and unique key.
- Declare key constraints in relational databases.
- Describe the different ways foreign-key constraint violation can be handled and how to declare them.

Key Constraints: Why and What?

- Some attributes serve as keys– or identifiers; e.g., SSN, email.
- DB needs to know such attributes, to ensure their correctness, and to enhance performance (e.g., creating indexes).
- Types of keys:
  - **Primary Key**: Attributes (e.g., SSN) as a primary way to identify a tuple.
    - SQL keywords: PRIMARY KEY.
  - **Unique Key**: Attributes (e.g., email) as a way to uniquely identify a tuple.
    - SQL keywords: UNIQUE.
  - **Foreign Key**: Attributes that uniquely identify a tuple in another table.
    - SQL keywords: FOREIGN KEY, REFERENCES.

PRIMARY KEY vs. UNIQUE Constraints

- Both have in common
  - Values are unique.
- Differences
  - PRIMARY KEY: at most one. UNIQUE: can be more than one.
  - PRIMARY KEY: values are also NOT NULL. UNIQUE: can be null.
  - SQL standard allows DBMS implementation to make own distinctions.
    - E.g., MySQL automatically creates an index for PRIMARY KEY.



8.3.2 Primary Key Optimization

The primary key for a table represents the column or set of columns that you use in your most vital queries. It has an associated index, for fast query performance. Query performance benefits from the `NOT NULL` optimization, because it cannot include any `NULL` values. With the `InnoDB` storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

If your table is big and important, but does not have an obvious column or set of columns to use as a primary key, you might create a separate column with auto-increment values to use as the primary key. These unique IDs can serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

Primary Key Optimization, MySQL, 2018. Retrieved from https://dev.mysql.com

Foreign Keys

- **Foreign key** is a set of one or more attributes of a table that refers to a key in another table.
- **Foreign-key (referential integrity) constraint** requires a set of attributes to refer to key values in another table.
- Consider Relation Sells(bar, beer, price).
  - We expect that a beer value is a real beer– so it should appear in Beers.name. Thus, Sells.beer is a **foreign key** of Sells.
  - A constraint that requires a beer in Sells to be some key in Beers is called a **foreign-key** constraint.

Expressing Foreign Keys When Creating Tables

- In an attribute element:

  <att> <type> **REFERENCES** <table> (<attribute>)
- As a separate element:

  FOREIGN KEY (<attributes>)

  REFERENCES <table> (<attributes>)
- Referenced attributes must be declared PRIMARY KEY or UNIQUE in the referenced table.

Foreign-Key Examples

- *Q1: Make Sells.bar and Sells.beer foreign keys to the Bars and Beers tables.*

```
mysql> CREATE TABLE Sells (
    ->        bar char(30) REFERENCES Bars(name),
    ->        beer char(30),
    ->        price float NOT NULL,
    ->        discount float DEFAULT 0,
    ->        PRIMARY KEY (bar, beer),
    ->        FOREIGN KEY (beer) REFERENCES Beers(name)
    ->        );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO Sells VALUES ('Purple Bar', 'New Beer', 4, 0);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint f
ails (`FridayNight`.`Sells`, CONSTRAINT `Sells_ibfk_1` FOREIGN KEY (`beer`) REFE
RENCES `Beers` (`name`))
mysql> SELECT * FROM Beers;
+-----------+-------------+---------+
| name      | brewer      | alcohol |
+-----------+-------------+---------+
| Bud       | AB InBev    |    0.05 |
| Bud Lite  | AB InBev    |   0.042 |
| Coors     | Coors       |    0.05 |
| Sam Adams | Boston Beer |   0.049 |
+-----------+-------------+---------+
4 rows in set (0.00 sec)
```

Foreign-key example in MySQL

How Can a Foreign-Key Constraint Be Violated?

- Consider a foreign-key constraint $S.a$ References $T.k$.
- Violation 1: Source Change.

  An insertion or update to $S.a$ introduces values not found in T.

- Violation 2: Target Change.

  A deletion or update to $T.k$ causes some tuples of S to “dangle.”

Actions for Handling Violations

- Violation 1: Source Change – Must be **rejected**.
  - The change of Source introduces a non-existent key for Target.
  - Such violations can only be removed by rejecting the change.

- Violation 2: Target Change – Three choices.
  - The change of Target removed a key that Source referenced.
    1. **Reject** (default) : Reject the modification.
    2. **Cascade**: Change $S.a$ in the same way (delete or update) as $T.k$.
    3. **Set NULL**: Change $S.a$ to NULL.

Declaring Actions for Foreign-Key Constraints

- Add declaration **ON** <DELETE/UPDATE> <ACTION>
  - E.g., ON DELETE CASCADE
  - E.g., ON UPDATE SET NULL

- If declaration is omitted, the default (reject) is take is taken.

Foreign-Key Violation Examples

- *Q1: Try the default action for the Sells foreign key Sells.bar.*
- *Q2: Set Cascade for the Sells foreign key Sells.beer.*

```
mysql> CREATE TABLE Sells (
    ->        bar char(30),
    ->        beer char(30),
    ->        price float NOT NULL,
    ->        discount float DEFAULT 0,
    ->        PRIMARY KEY (bar, beer),
    ->        FOREIGN KEY (bar) REFERENCES Bars(name),
    ->        FOREIGN KEY (beer) REFERENCES Beers(name)
    ->        ON DELETE CASCADE
    ->        ON UPDATE CASCADE
    ->        );
Query OK, 0 rows affected (0.01 sec)
```

Foreign-key violation example in MySQL

```
mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)

mysql> DELETE FROM Bars WHERE name = "Green Bar";
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`Frida
yNight`.`Sells`, CONSTRAINT `Sells_ibfk_1` FOREIGN KEY (`bar`) REFERENCES `Bars` (`name`))
mysql>
mysql> DELETE FROM Beers WHERE name = "Bud";
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE Beers SET name = "Bud Heavy" WHERE name = "Bud Lite";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud Heavy |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
4 rows in set (0.00 sec)
```

Food for Thought

The foreign-key handling choices may not all be applicable– some may conflict with other constraints. Why would the following not work?

```
mysql> CREATE TABLE Sells (
    ->         bar char(30) REFERENCES Bars(name),
    ->         beer char(30),
    ->         price float NOT NULL,
    ->         discount float DEFAULT 0,
    ->         PRIMARY KEY (bar, beer),
    ->         FOREIGN KEY (beer) REFERENCES Beers(name)
    ->         ON DELETE SET NULL
    -> );
ERROR 1215 (HY000): Cannot add foreign key constraint
```

No!

Foreign-key example in MySQL

Regulating Databases: Checks and Assertions

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Describe what checks and assertions are, and how to declare them in DBMS.
- Explain the differences between different types of checks and assertions.
- Understand that checks and assertions are not fully implemented in most DBMS.

General Constraint Mechanisms

- Checks
  - Attribute-based checks
  - Tuple-based checks
- Assertions
  - Database-based
- Advanced features– not uniformly supported in various RDBMS.
  - MySQL does not support checks and assertions.
  - PostgreSQL supports checks without no subqueries in checks.

Checks 1: Attribute-based

- Constrain the value of a particular attribute in a tuple.

  CREATE TABLE <name> (

  … … ,

  <attr> <type> **CHECK** (<condition>),

  … …);

- The condition refers to the attribute; any other attributes or relations must be in a subquery to compare.
  - E.g., price float CHECK (price > 1.0).
- Timing: Attribute-based check is checked only when the attribute is inserted or updated.

Checks 2: Tuple-based

- Constrain the value of multiple attributes in a tuple.

  CREATE TABLE <name> (

  … … ,

  CHECK (<condition>),

  … …);

- The condition refers to multiple attributes in the table; any other attributes or relations must be in a subquery to compare.
  - E.g.: CHECK (price * (1-discount) > 1.0).
- Timing: Tuple-based check is checked only when a tuple is inserted or updated.

Check Examples

- *Q1: Set a check for price < $10.0 in Sells.*
- *Q2: Set a check that the actual price (after discount) must be more than $1.*

```
psql (9.6.8)
Type "help" for help.

kevinccchang=# CREATE TABLE Sells (
kevinccchang(#        bar char(30),
kevinccchang(#        beer char(30),
kevinccchang(#        price float CHECK (price <= 10.00),
kevinccchang(#        discount float DEFAULT 0,
kevinccchang(#        PRIMARY KEY (bar, beer),
kevinccchang(#        CHECK (price*(1-discount)>1.0)
kevinccchang(#        );
CREATE TABLE
kevinccchang=# INSERT INTO Sells VALUES ('Green Bar', 'Coors', 8, 0);
INSERT 0 1
kevinccchang=# INSERT INTO Sells VALUES ('Green Bar', 'Coors', 12, 0);
ERROR:  new row for relation "sells" violates check constraint "sells_price_chec
k"
DETAIL:  Failing row contains (Green Bar                    , Coors
            , 12, 0).
kevinccchang=# INSERT INTO Sells VALUES ('Green Bar', 'Coors', 10, 0.95);
ERROR:  new row for relation "sells" violates check constraint "sells_check"
DETAIL:  Failing row contains (Green Bar                    , Coors
            , 10, 0.95).
kevinccchang=#
```

Check examples in PostgreSQL

Assertions: Database-based

- We declare an assertion as part of the database schema:
  CREATE ASSERTION <name> **CHECK** (<condition>);

- Thus, a CHECK over the database
  - Not limited to a tuple.
  - Condition may refer to any relation or attribute in the database schema.

```
CREATE ASSERTION NoBoringBars CHECK (
   NOT EXISTS (
      SELECT bar FROM Sells
      GROUP BY bar
      HAVING COUNT(beer) < 3
   ));
```

Food for Thought

Wondering if we really need assertions. Consider, say, the NoBoringBars example assertion: Can we express it using attribute/tuple-based checks?

```
CREATE ASSERTION NoBoringBars CHECK (
    NOT EXISTS (
            SELECT bar FROM Sells
            GROUP BY bar
            HAVING COUNT(beer) < 3
    ));
```

Food for Thought

While assertion has been defined in SQL since SQL-92, it has not been implemented by major DBMS.
(Oracle is considering- see below.)
Can you imagine why?

Oracle Developer Community, 2018
Retrieved from
https://community.oracle.com/ideas/13028.

Regulating Databases: Triggers

Manipulating Databases

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Explain the limitations of checks and assertions, and how triggers address these limitations.
- Define the trigger mechanism in terms of event, condition, and action.
- Create triggers in RDBMS.

Checks and Assertions Fall Short

- Limitations:
  - Lacking timing
    - Require checking whenever any updates are made (on the elements involved)– expensive!
  - Lacking flexibility
    - Checks are limited to attributes or tuple-based constraints.
  - Lacking handling
    - We cannot control what to do when constraints are violated.

- What's missing?
  - Timing
  - Flexibility
  - Handing

Trigger: Event-Condition-Action Rules

- Trigger is also called ECA rule, event-condition-action rule.
- **Event**: provide **timing**
  - Change of database state, i.e., modification, e.g., “insert on Sells.”
- **Condition**: provide **flexibility**:
  - Any SQL Boolean-valued expression, with flexible scopes.
- **Action** : provide **handling**.
  - Any SQL (modification) statements.
- SQL standard since SQL:1999
  - Widely implemented
  - However, with slightly non-uniform functionality and syntax.

Trigger: Example #1

```
CREATE TRIGGER PriceReset
    AFTER
        UPDATE OF price ON Sells
    REFERENCING
        OLD ROW AS old
        NEW ROW AS new
    FOR EACH ROW
    WHEN (new.price > old.price + 5.00)
        UPDATE Sells SET price = 0
            WHERE bar = new.bar AND beer = new.beer
```

Event

Condition

Action

Trigger 1): Event

- Describes timing of trigger activation
- Consists of preposition + event

- Preposition:
  - AFTER, BEFORE
  - INSTEAD OF

- Event: A modification to the database
  - INSERT, DELETE of tuples
  - UPDATE on attributes

Trigger 2): Condition

- Describes a condition to check

  REFERENCING …..
  FOR EACH ROW / STATEMENT
  WHEN <condition>

- **Variables**: REEERCEEING declares data elements to refer to.
- **Scope**:
  - row-level: FOR EACH ROW
  - statement-level: FOR EACH STATEMENT
- **Constraint violation**: as expressed in WHEN <codition>

Trigger 2): Condition - REFERENCING

- Declares variables to refer to.

  REFERENCING [NEW / OLD][ROW / TABLE] AS <name>
- Row level
  - INSERT→ NEW ROW
  - DELETE → OLD ROW
  - UPDATE → both OLD ROW and NEW ROW
- Statement level
  - Both NEW TABLE and OLD TABLE

Trigger: Example #2

```
CREATE TRIGGER BeerForeignKey
AFTER
        INSERT ON Sells
REFERENCING
        NEW ROW AS new
FOR EACH ROW
WHEN (new.beer NOT IN (SELECT name FROM Beers))
        INSERT INTO Beers(name) VALUES (new.beer);
```

Event

Condition

Action

Trigger: Example #3

```
CREATE TRIGGER BarShouldNotBeBoring
AFTER
        DELETE OR UPDATE ON Sells
FOR EACH STATEMENT
    WHEN EXISTS (SELECT bar FROM Sells GROUP BY bar
                 HAVING COUNT(beer) < 3)
    INSERT INTO Boring(bar) VALUES
                (SELECT bar FROM Sells GROUP BY bar
                 HAVING COUNT(beer) < 3));
```

Event

Condition

Action

Trigger Examples

- *Q1: Create the trigger PriceReset (Example #1) in MySQL.*
- *Q2: Create the trigger BarShouldNotBeBoring (Example #3) in PostgreSQL.*

```
mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |   4.5 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     5 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |     6 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)

mysql> delimiter //
mysql> CREATE TRIGGER PriceReset BEFORE UPDATE ON Sells
    -> FOR EACH ROW
    -> BEGIN
    -> IF NEW.price > OLD.price + 5 THEN
    -> SET NEW.price = 0;
    -> END IF;
    -> END;//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> UPDATE Sells SET price=11 WHERE beer='Sam Adams';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM Sells;
+------------+-----------+-------+----------+
| bar        | beer      | price | discount |
+------------+-----------+-------+----------+
| Green Bar  | Bud       |     2 |        0 |
| Green Bar  | Sam Adams |     0 |      0.2 |
| Purple Bar | Bud       |   4.5 |      0.2 |
| Purple Bar | Sam Adams |     0 |      0.3 |
| Sober Bar  | Bud       |     5 |     0.05 |
| Sober Bar  | Bud Lite  |     3 |        0 |
| Sober Bar  | Sam Adams |    11 |      0.1 |
+------------+-----------+-------+----------+
7 rows in set (0.00 sec)
```

```
kevinccchang=# SELECT  * FROM Sells;
             bar              |             beer             | price | discount
------------------------------+------------------------------+-------+----------
 Sober Bar                    | Bud                          |     5 |     0.05
 Sober Bar                    | Bud Lite                     |     3 |        0
 Sober Bar                    | Sam Adams                    |     6 |      0.1
 Green Bar                    | Bud                          |     2 |        0
 Green Bar                    | Sam Adams                    |   4.5 |      0.2
 Purple Bar                   | Bud                          |   4.5 |      0.2
 Purple Bar                   | Sam Adams                    |     5 |      0.3
 Purple Bar                   | Coors                        |   6.5 |      0.2
(8 rows)

kevinccchang=# SELECT bar FROM Sells GROUP BY bar
kevinccchang-# HAVING COUNT(beer) < 3;
             bar
--------------------------------
 Green Bar
(1 row)

kevinccchang=# \d+
                        List of relations
 Schema |  Name  | Type  |    Owner     |    Size    | Description
--------+--------+-------+--------------+------------+-------------
 public | boring | table | kevinccchang | 8192 bytes |
 public | sells  | table | kevinccchang | 8192 bytes |
(2 rows)

kevinccchang=# SELECT * FROM Boring;
 bar
-----
(0 rows)
```

```
kevinccchang=# CREATE FUNCTION boringHandling() RETURNS trigger AS
kevinccchang-# $body$
kevinccchang$#     BEGIN
kevinccchang$#     INSERT INTO Boring(bar)
kevinccchang$#       SELECT bar FROM Sells GROUP BY bar
kevinccchang$#        HAVING COUNT(beer) < 3;
kevinccchang$#     RETURN NULL;
kevinccchang$#     END;
kevinccchang$# $body$ LANGUAGE plpgsql;
CREATE FUNCTION
kevinccchang=# CREATE TRIGGER BarShouldNotBeBoring
kevinccchang-# AFTER DELETE OR UPDATE ON Sells
kevinccchang-# FOR EACH STATEMENT
kevinccchang-# EXECUTE PROCEDURE boringHandling();
CREATE TRIGGER
kevinccchang=# DELETE FROM Sells WHERE bar='Purple Bar' AND beer='Sam Adams';
DELETE 1
kevinccchang=# SELECT * FROM Sells;
             bar              |             beer             | price | discount
------------------------------+------------------------------+-------+----------
 Sober Bar                    | Bud                          |     5 |     0.05
 Sober Bar                    | Bud Lite                     |     3 |        0
 Sober Bar                    | Sam Adams                    |     6 |      0.1
 Green Bar                    | Bud                          |     2 |        0
 Green Bar                    | Sam Adams                    |   4.5 |      0.2
 Purple Bar                   | Bud                          |   4.5 |      0.2
 Purple Bar                   | Coors                        |   6.5 |      0.2
(7 rows)

kevinccchang=# SELECT * FROM Boring;
             bar
--------------------------------
 Green Bar
 Purple Bar
(2 rows)
```

Trigger examples in MySQL and PostgreSQL

Food for Thought

We learned that not all views are updatable– because the DBMS may not know how to update the underlying base tables given updates on views. One reason that trigger was invented was to allow users to control views!

Can you imagine how we can use triggers to handle view updates?

Aspects of a Trigger Subsystem in an Integrated
Database System

by

Kapali P. Eswaran
IBM Research Laboratory
San Jose

ABSTPACT. This paper considers the specifications and design of a trigger subsystem in a database management system. The use of triggers as extended assertions and as a means to materialize virtual data objects are discussed. The functional requirements of a trigger subsystem and different implementation issues are studied. We also examine the relationships between a trigger subsystem and the rest of the database system, in particular the authorization and locking subsystems.

Kapali P. Eswaran:
Aspects of a Trigger Subsystem in an Integrated Data Base System. ICSE 1976: 243-250

The End