浙江大学
ZHEJIANG UNIVERSITY

第十章 数据库安全性与完整性

陶煜波
计算机科学与技术学院

空间数据库回顾

- 空间数据库 = 对象关系数据库 + 空间扩展
- 对象关系数据库与关系数据库相比
  - 扩充了系统类型 (结构**类型**，数组类型，多重集合类型和参照类型，支持继承，特别是**方法**)
  - 关系不仅是**元组**的集合，也可以是**对象**的集合
- 空间数据库 (矢量)
  - 空间数据类型 (几何对象模型的Geometry)
    - geom geometry (Point/LineString/Polygon/Multixxx, 空间参考系)
    - ST_GeomFromText ('WKT表示', 空间参考系)
  - 空间数据分析 (几何对象模型的30个查询方法)
    - ST_XXX: ST_Distance, ST_Intersects, ST_DWithin, …
  - 空间数据索引 (PostGIS的GiST索引)

空间数据库回顾

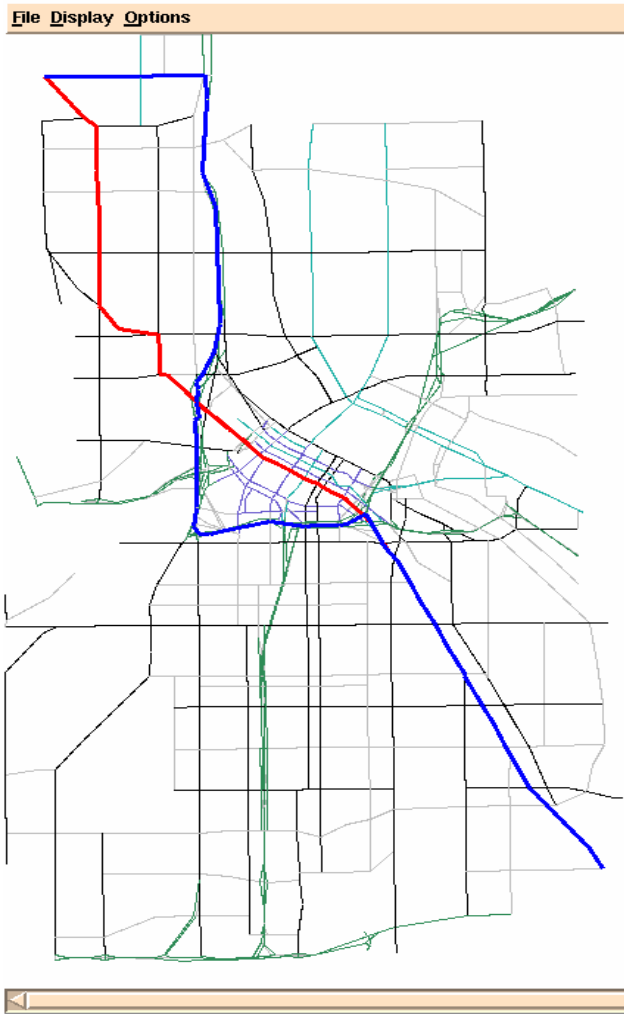
- 空间数据模型
  - 几何对象模型
    - 基于预定义数据类型的实现 (numeric或BLOB)
    - 基于扩展几何类型的实现 (Geometry类)
  - 几何拓扑模型
  - 网络模型
  - 栅格模型
  - 注记文本模型
    - 基于预定义和扩展Geometry的实现
- PostGIS的空间数据类型、GIS分析和索引

第九章 空间网络模型回顾

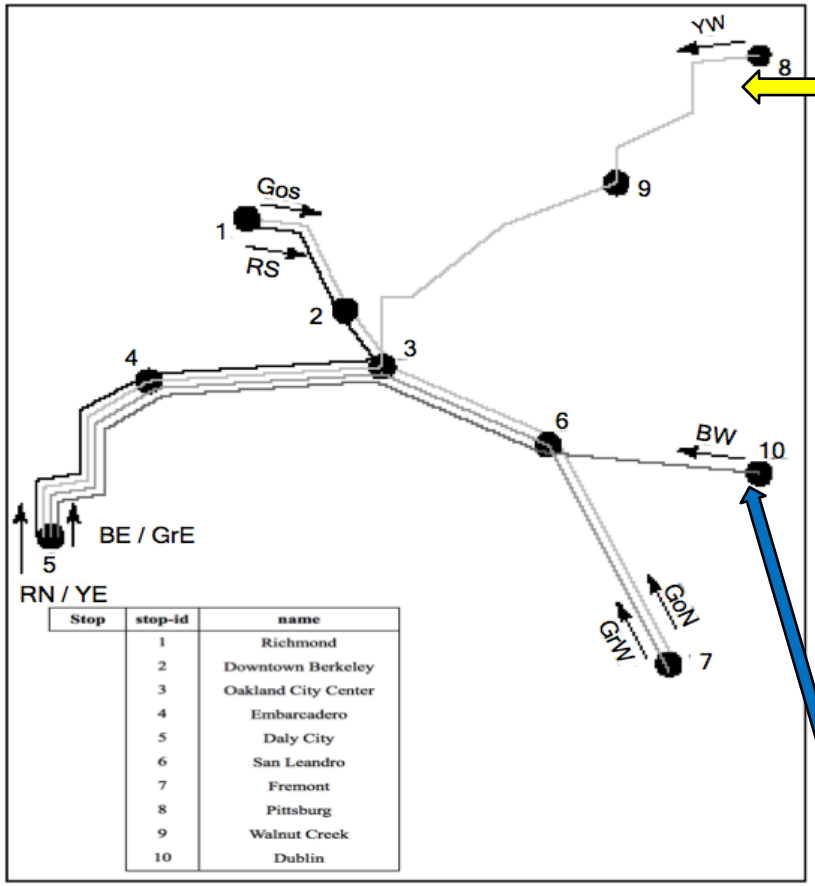
- Location Based Services
  - Location: Where am I ?
  - Directory: What is around me?
  - Routes: How do I get there?
- Spatial Network Analysis
  - Route (A start-point, Destination(s))
  - Allocation (A set of service centers, A set of customers)
  - Site Selection (A set of customers, Number of new service centers)
- Spatial Network Examples
  - Road, Railway, River

Spatial Network Query Example

- Find shortest path from a start-point to a destination
- Find nearest hospital by driving distance
- Find shortest route to deliver packages to a set of homes
- Allocate customers to nearest service center



Railway Network & Queries



Stop	stop-id	name
	1	Richmond
	2	Downtown Berkeley
	3	Oakland City Center
	4	Embarcadero
	5	Daly City
	6	San Leandro
	7	Fremont
	8	Pittsburg
	9	Walnut Creek
	10	Dublin

- Find the number of stops on the Yellow West (YW) route
- List all stops which can be reached from Downtown Berkeley (2)
- List the routes numbers that connect Downtown Berkeley (2) & Daly City (5)
- Find the last stop on the Blue West (BW) route

River Network & Queries

- List the names of all direct and indirect tributaries (支流) of Mississippi river
- List the direct tributaries of Colorado
- Which rivers could be affected if there is a spill in North Platte river

(b) River Network

Data Models of Spatial Networks

- Conceptual Model
  - Information Model: Entity Relationship Diagrams
  - Mathematical Model: Graphs
    - Road (turns), River, Railway
- Logical Data Model
  - Abstract Data Types - Transitive Closure
  - Custom Statements in SQL
    - SQL2 - CONNECT clause in SELECT statement
    - SQL3 - WITH RECURSIVE statement
- Physical Data Model
  - Storage: Data-Structures, File-Structures
    - Adjacency matrix, adjacency list, normalized/denormalized tables
    - Minimize CRR
  - Algorithms for Common Operations

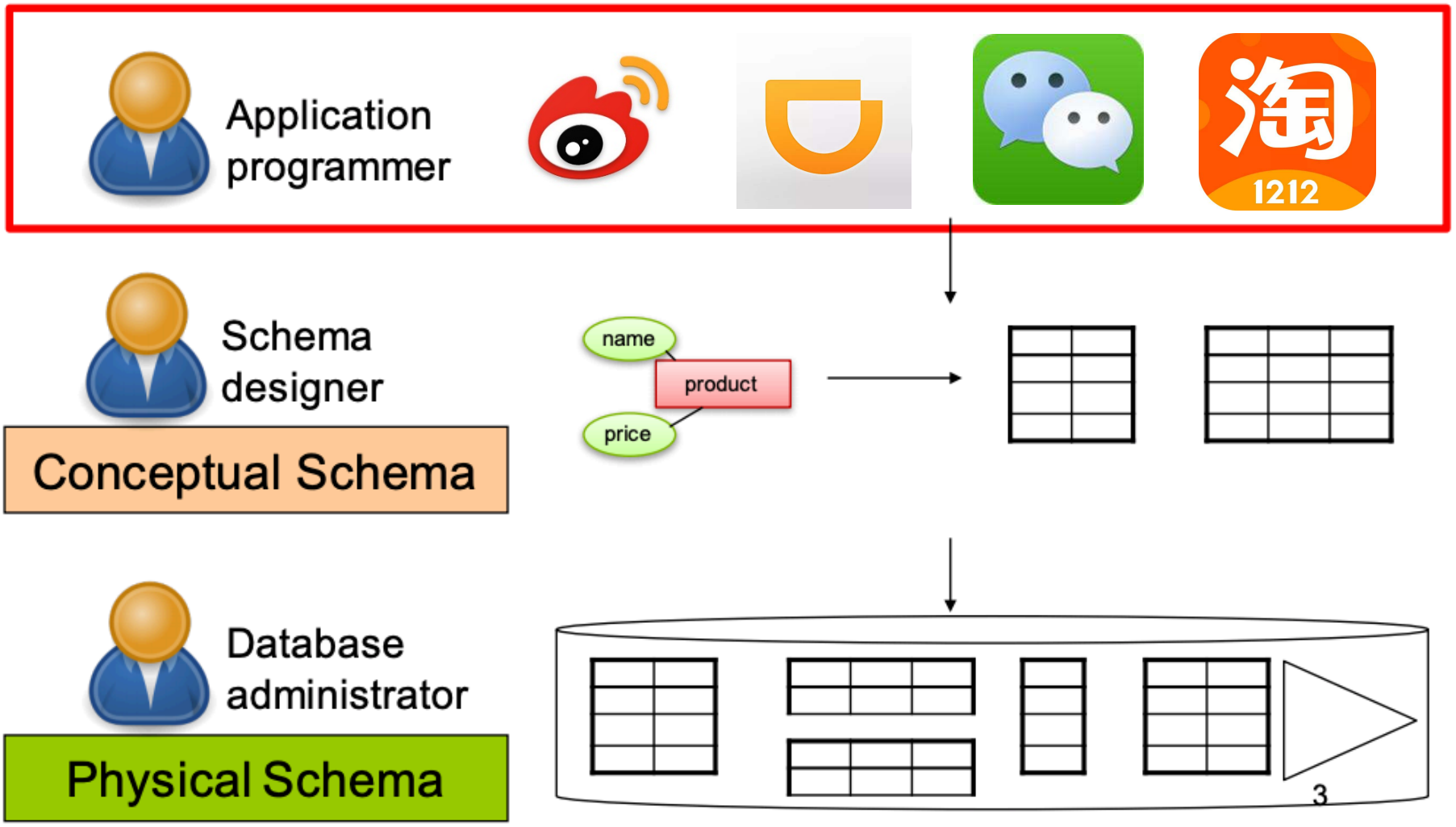
Query Processing for Spatial Networks

- Main memory
  - Connectivity: Breadth first search, depth first search
  - Shortest path: Dijkstra's algorithm, A*
- Disk-based
  - Connectivity strategies are in SQL3
  - Shortest path - Hierarchical routing algorithm
- pgRouting
  - pgr_createTopoloty
  - pgr_analyzeGraph
  - pgr_nodeNetwork
  - pgr_dijkstra(text edges_sql, bigint start_vid, bigint end_vid, boolean directed:=true)

空间数据库回顾

- 地理空间数据库概念，关系代数和SQL — 第1-4周
- 几何对象模型与查询 — 第4-5周
- 空间网络模型与查询 — 第11-12周
- 空间数据库设计
  - 需求分析 (几何，属性，行为)
  - 概念设计 (扩展的E/R图)
  - 逻辑设计 (将扩展E/R图转换为关系，关系优化) — 第6-7周
  - 物理设计 (空间数据库，存储方式，建立索引) — 第8-10周
  - 实施 (建表，导入数据，SQL编程，试运行)
  - 运行维护 (转储和恢复，安全性和完整性控制)
    (性能监督、分析改进、重组与重构) — 第10,12-14周
- OLAP / NoSQL — 第14-15周

空间数据库回顾


Application programmer
Schema designer
Conceptual Schema
name
product
price
Database administrator
Physical Schema
3

第十章 数据库安全性与完整性

- 10.1 数据库的安全性
  - 10.1.1 数据库安全性
  - 10.1.2 存取控制
- 10.2 数据库的完整性
  - 10.2.1 完整性约束条件的定义
  - 10.2.2 触发器
  - 10.2.3 视图与触发器
  - 10.2.4 完整性约束条件的修改
  - 10.2.5 完整性约束条件的检查和违约处理

参考教材：
数据库系统概念 4.2，4.4，4.6，5.3，13.5

10.1.1 数据库安全性

- 数据库安全性
  - 系统运行安全性
  - 系统信息安全性
    - 数据访问(data access)安全性
    - 系统或编程(system or programming)安全性
- 数据库系统的安全特性
  - 数据独立性
  - 数据安全性
  - 数据完整性
  - 并发控制
  - 故障恢复

思考：如何通过数据独立性保障数据库安全性？

https://baike.baidu.com/item/数据库安全/9983370

10.1.1 数据库安全性

- 信息泄露
  - 黑客通过B/S应用，以Web服务器为跳板，窃取数据库中数据；传统解决方案对应用访问和数据库访问协议没有任何控制能力
  - 数据泄露常常发生在内部，大量的运维人员直接接触敏感数据，传统以防外为主的网络安全解决方案失去了用武之地
  - 例如：2013年10月，慧达驿站软件漏洞导致连锁酒店数据库拖库事件：2000万条开房记录泄露

思考：上述两类分别属于数据访问安全性，还是系统或编程安全性？

SQL注入

危险动作
仅供教学
请勿模仿

- 网站用户登录

- 除了帐号密码外，还有一个公司名输入框，根据输入框，其服务器端的SQL写法大概：

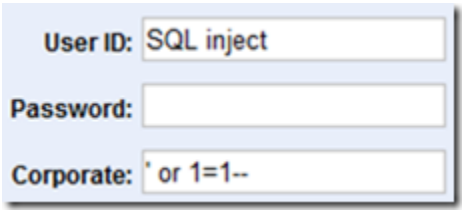
  Select * From Table Where Name= ‘XX’ and Password = ‘YY’ and Corp = ‘ZZ’

- User ID和Password在客户端(网页)做了检查，但Corporate框疏忽了

思考：如何在客户端做输入检查？

SQL注入 – 匿名登录

- Select * From Table Where Name= ‘XX’ and Password = ‘YY’ and Corp = ‘ZZ’

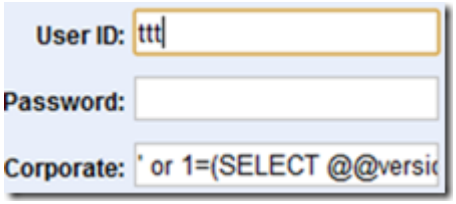


- SELECT * From Table WHERE Name='SQL inject' and Password='' and Corp='' or 1=1--'

思考：在服务器数据库中保存密码，即Password = ‘YY’，是安全的设计？

SQL注入 – 借助异常获取信息

- Select * From Table Where Name= ‘XX’ and Password = ‘YY’ and Corp = ‘ZZ’



- SELECT * From Table WHERE Name=‘ttt' and Password='' and Corp='' or 1=(SELECT @@VERSION)--'
  - Conversion failed when converting the nvarchar value 'Microsoft SQL Server 2008 (SP3) - 10.0.5500.0 (X64) Sep 21 2011 22:45:45 Copyright (c) 1988-2008 Microsoft Corporation Developer Edition (64-bit) on Windows NT 6.1 <X64> (Build 7601: Service Pack 1) ' to data type int.

SQL注入 – 获取服务器所有的库名、表名、字段名

- 输入：t' or 1=(SELECT top 1 name FROM master..sysdatabases where name not in (SELECT top 0 name FROM master..sysdatabases))--
- Conversion failed when converting the nvarchar value 'master' to data type int.
  - 数据库名"master"

SQL注入 – 获取服务器所有的库名、表名、字段名

- 输入：b' or 1=(SELECT top 1 name FROM master..sysobjects where xtype='U' and name not in (SELECT top 1 name FROM master..sysobjects where xtype='U'))--
- Conversion failed when converting the nvarchar value 'spt_fallback_db' to data type int.
  - 表名spt_fallback_db

思考：PosgreSQL数据库存在哪些系统表？

SQL注入 – 获取服务器所有的库名、表名、字段名

- 输入：b' or 1=(SELECT top 1 master..syscolumns.name FROM master..syscolumns, master..sysobjects WHERE master..syscolumns.id=master..sysobjects.id AND master..sysobjects.name='spt_fallback_db');
- Conversion failed when converting the nvarchar value 'xserver_name' to data type int.
  - spt_fallback_db的第一个字段xserver_name
- 获取数据库中的数据

SQL注入

- 服务器端代码

```
advisorName = params[:form][:advisor]
students = Student.find_by_sql(
    "SELECT students.* " +
    "FROM students, advisors " +
    "WHERE student.advisor_id = advisor.id " +
    "AND advisor.name = '" + advisorName + "'");
```

- 正常查询 (从表格中输入参数是'Jones')

```
SELECT students.* FROM students, advisors
    WHERE student.advisor_id = advisor.id
    AND advisor.name = 'Jones'
```

SQL注入

- 在"Advisor Name"框中输入

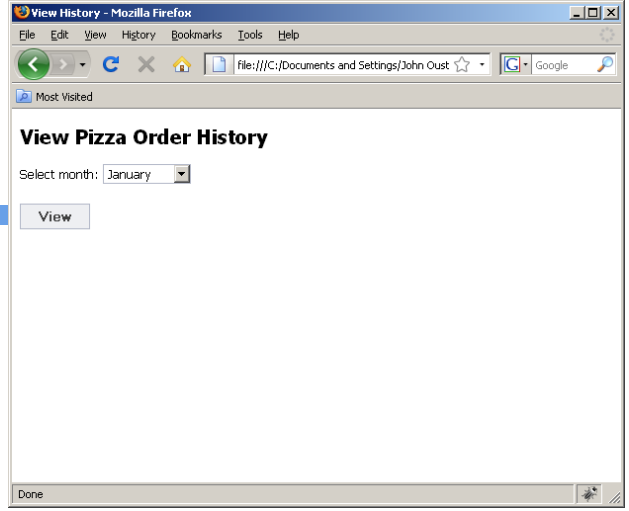
```
Jones';
UPDATE grades
    SET g.grade = 4.0
    FROM grades g, students s
    WHERE g.student_id = s.id
    AND s.name = 'Smith
```

最终查询变为：

```
SELECT students.* FROM students, advisors
    WHERE student.advisor_id = advisor.id
    AND advisor.name = 'Jones';
UPDATE grades
    SET g.grade = 4.0
    FROM grades g, students s
    WHERE g.student_id = s.id
    AND s.name = 'Smith'
```

思考：如何在客户端做输入检查避免此类漏洞，或者在数据访问上做什么限制？

SQL注入



- 服务器端代码

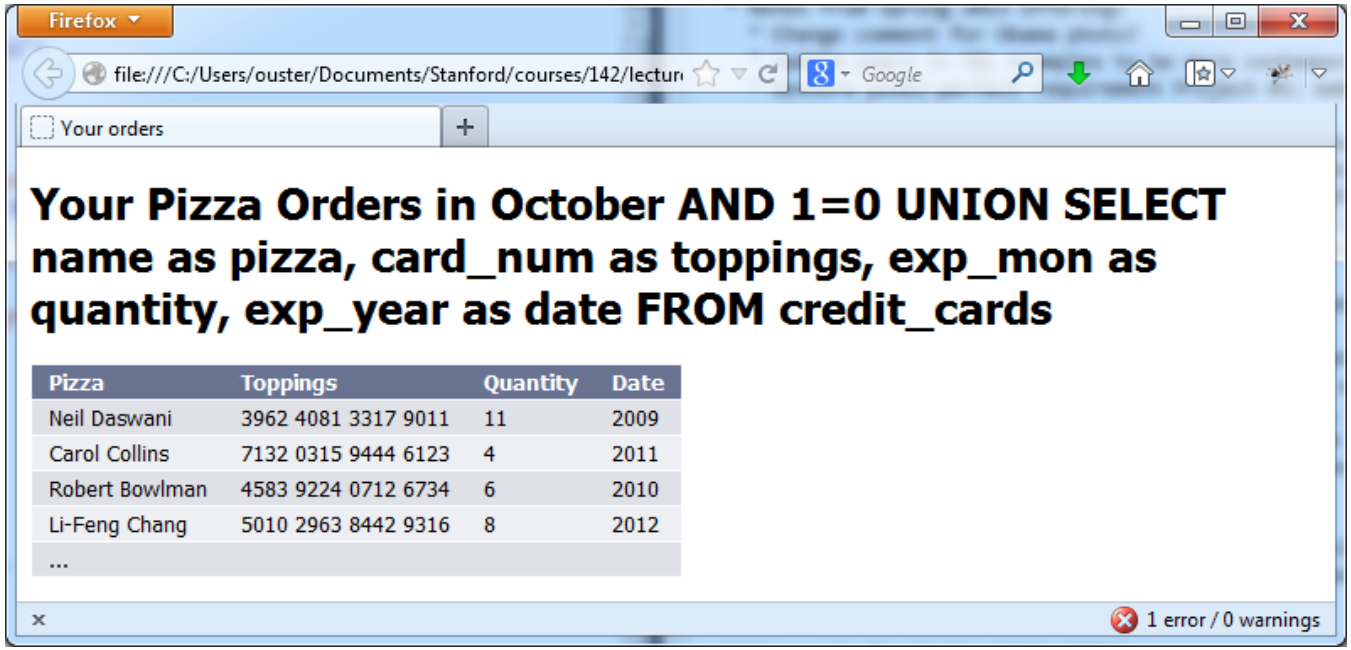
```
month = params[:form][:month]
orders = Orders.find_by_sql(
    "SELECT pizza, toppings, quantity, date " +
        "FROM orders " +
        "WHERE user_id=" + user_id +
        "AND order_month=" + month);
```

- month输入

```
October AND 1=0
UNION SELECT name as pizza, card_num as
toppings,
    exp_mon as quantity, exp_year as date
FROM credit_cards '
```

SQL注入

```
SELECT pizza, toppings, quantity, date
    FROM orders
    WHERE user_id=94412
    AND order_month= October AND 1=0
UNION SELECT name as pizza, card_num as toppings,
    exp_mon as quantity, exp_year as date
    FROM credit_cards
```



SQL注入

- CardSystems
  - Credit card payment processing company
  - SQL injection attack in June 2005

- The Attack
  - Credit card #s stored unencrypted
  - 263,000 credit card #s stolen from database
  - 43 million credit card #s exposed

- 服务器端解决方法之一
  - Prepared Statements

SQL注入

- 安全性
  - 对用户输入的内容要时刻保持警惕
  - 只有客户端的验证等于没有验证
  - 永远不要把服务器错误信息暴露给用户
- 此外
  - SQL注入不仅能通过输入框，还能通过URL达到目的
  - 除了服务器错误页面，还有其他办法获取到数据库信息
  - 可通过软件模拟注入行为，这种方式盗取信息的速度要比你想象中快的多

http://www.cnblogs.com/hkncd/archive/2012/03/31/2426274.html

10.1.2 存取控制

- 存取控制
  - 指授予某个用户某种特权，利用该特权可以以某种方式(如读取、修改等)访问数据库中的某些数据对象
- 数据库创建者拥有所有权限
- SQL授权语句：GRANT/REVOKE
  - GRANT将某种权限授予某个用户
  - REVOKE则收回某个用户所授权限

10.1.2 存取控制

- 数据库授权
  - Make sure users see only the data they're supposed to see [类似View的功能]
  - Guard the database against modifications by malicious users [Grant/Revoke的功能]
- 用户仅能在授权的数据上进行操作
  - Select on R or Select (A_1, …, A_n) on R
  - Insert on R or Insert (A_1, …, A_n) on R
  - Update on R or Update (A_1, …, A_n) on R
  - Delete on R

10.1.2 存取控制

- 关系R(a, b), S(b, c)和T(a, c)需要哪些权限才能执行下列SQL语句

Update R

Set a = 10

Where b in (select c from S)

and not exists (select a from T where T.a = R.a)

Update on R(a), Select on R(a, b), Select on S(c), Select on T(a)

10.1.2 存取控制

- GRANT一般格式为：
  - GRANT <权限>[，<权限>]…
  - [ON <对象类型> <对象名>]
  - TO <用户>[，<用户>]…
  - [WITH GRANT OPTION]；
- Grant privs On R to users [With Grant Option]
  - Privileges: Select(R), Insert(R), Update(R), Delete(R)
- 其语义为：将对指定数据对象的指定操作权限授予指定的用户
  - 基本表或视图

10.1.2 存取控制

- 举例
  - 假设用户王平创建了基本表S、C和SC，则他自动获得对这些表的所有权限(包括将这些权限传播给其他用户的权力)

    GRANT INSERT,DELETE ON SC TO 李霞

    WITH GRANT OPTION
  - 执行此SQL语句后，用户李霞不仅拥有了对表SC的INSERT和DELETE权限，还可以传播这些权限，即由用户李霞发上述GRANT命令给其他用户

10.1.2 存取控制

例如：李霞可以将此权限授予张建：

```
GRANT INSERT,DELETE ON SC
TO 张建
WITH GRANT OPTION;
```

同样，张建还可以将此权限授予李丽：

```
GRANT INSERT,DELETE ON SC
TO 李丽;
```

张建未给李丽传播权限，因此李丽不能再传播此权限

10.1.2 存取控制

- REVOKE一般格式为：

  REVOKE <权限>[，<权限>]…

  [ON <对象类型> <对象名>]

  FROM <用户>[，<用户>]…；

- Revoke privs On R From users [Cascade | Restrict]
  - Cascade: also revoke privileges granted from privileges being revoked (transitively), unless also granted from another source
  - Restrict: Disallow if Cascade would revoke any other privileges [Default]
  - Grant diagram

10.1.2 存取控制

把用户王芳修改学生年龄的权限收回

REVOKE UPDATE(Sage) ON S FROM 王芳;

把用户李霞对SC表的INSERT权限收回

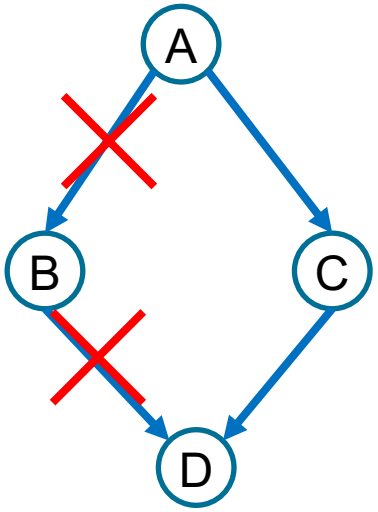
REVOKE INSERT ON SC FROM 李霞 Cascade;

DBMS在收回李霞对SC表的INSERT权限的同时，还会自动收回张建和李丽对SC表的INSERT权限，即收回权限的操作会级联下去的

10.1.2 存取控制

- 用户A创建了表R，接着依次执行以下操作
  - A grant read(R) to B with grant option
  - A grant read(R) to C with grant option
  - B grant read(R) to D
  - C grant read(R) to D
  - A revokes read(R) from B restrict

最后一句会报错吗？ NO



grant diagram

10.1.2 存取控制

- 用户U创建了表T，接着依次执行以下操作
  - User U: grant select on T to V, W with grant option
  - User V: grant select on T to W
  - User W: grant select on T to X,Y
  - User U: grant select on T to Y
  - User U: revoke select on T from V restrict
    - 此时，用户V，W，X和Y各有什么权限？
  - User U: revoke select on T from W cascade
    - 此时，用户V，W，X和Y各有什么权限？

学在浙大APP答题：课中互动 – 随堂测试 – 存取控制

10.1.2 存取控制

- 特权是执行一种特殊类型的SQL语句或存取另一用户的对象的权力
- 有两类特权
  - 系统特权
  - 对象特权

10.1.2 存取控制

- 系统特权
  - 执行一种特殊动作或者在特定对象类型上执行一种特殊动作的权利
  - 系统特权可授权给用户或角色
  - 一般情况下，系统特权授给系统管理人员和应用开发人员

10.1.2 存取控制

- 对象特权
  - 在指定的表、视图、存储过程或函数上执行特殊动作的权利。对于不同类型的对象,有不同类型的对象特权
  - 对于有些对象，如索引、触发器等，没有相关的对象特权，它们由系统特权控制
  - 数据库表的创建者对这些表上的对象具有全部对象特权
  - 对象的创建者可将这些对象上的任何对象特权授权给其他用户
  - 如果被授权者包含有GRANT OPTION 特权，那么他也可将其权利再授权给其他用户

10.1.2 存取控制

- 角色是一组权限的集合。有了角色的概念，安全管理机制可以把表或其他数据库对象上的一些权限进行组合，将它们赋予一个角色。需要时只需将该角色授予一个用户或一组用户，这样可以降低安全机制的负担和成本
- 优点：分组，便于管理

存取控制小结

- Make sure users see only the data they're supposed to see
- Guard the database against modifications by malicious users
- Users have privileges; can only operate on data for which they authorized
- Grant and Revoke statements
- Beyond simple table-level privileges: use views
- 系统特权和对象特权，角色

第十章 数据库安全性与完整性

- 10.1 数据库的安全性
  - 10.1.1 数据库安全性
  - 10.1.2 存取控制
- 10.2 数据库的完整性
  - 10.2.1 完整性约束条件的定义
  - 10.2.2 触发器
  - 10.2.3 视图与触发器
  - 10.2.4 完整性约束条件的修改
  - 10.2.5 完整性约束条件的检查和违约处理

参考教材：
数据库系统概念 4.2，4.4，4.6，5.3，13.5

10.2 数据完整性

- 数据库的完整性是指保证数据库中数据的正确性、有效性和相容性，防止错误的数据进入数据库
- 例如，学生的学号必须唯一；性别只能是男或女；学生所在的系必须是学校开设的系；用户需求要求的约束等

10.2 数据完整性

- 为维护数据库的完整性，DBMS必须提供一种机制来检查数据库中的数据，看其是否满足语义规定的条件
- 这些加在数据库数据之上的语义约束条件称为数据库完整性约束条件，它们作为模式的一部分存入数据库中
- DBMS中检查数据是否满足完整性约束条件的机制称为完整性检查

10.2 数据完整性

- 完整性约束(Integrity constraints) – static
  - Constrain allowable database states, beyond those imposed by structure and types
  - Why use them?
    - Data-entry errors (inserts)
    - Correctness criteria (updates)
    - Enforce consistency
    - Tell system about data (store, query processing)
- 触发器(Triggers) – dynamic
  - Monitor database changes
  - Check conditions and initiate actions

10.2 数据完整性

- 完整性约束(Integrity constraints)分类
  - 实体完整性
  - 参考完整性
  - 用户定义完整性
- 完整性约束(Integrity constraints)分类
  - Not Null
  - Key
  - Referential integrity (foreign key)
  - Attribute-based
  - Tuple-based
  - General assertion

思考：在实际应用中，我们并不定义太多完整性约束，为什么？

10.2 数据完整性

- SQL标准使用了一系列的技术来表达完整性，包括关系模型的实体完整性、参照完整性和用户定义的完整性
  - 不同数据库系统在实现上存在差异

- 这些完整性约束条件是用DDL语句定义的，主要体现在CREATE TABLE语句中，也可以通过Alter Table添加

10.2 数据完整性

- Declaring and enforcing constraints
- Declaration
  - With original scheme: checked after bulk loading
  - Or later: checked on current DB
- Enforcement
  - Check after every “dangerous” transaction
  - Deferred constraint checking

10.2 数据完整性

```
Create table T (A int primary key);
Insert into T values(123);
Insert into T values(234);
Update T set A = A – 111;
Update T set A = A + 111;
```

- 以上语句在各数据库系统中的执行结果？

- 用PRIMARY KEY语句定义了关系的主码后，每个用户程序对主属性进行操作时，系统将自动进行完整性检查，如果操作使主属性为空或使主码不唯一，则系统拒绝此操作，从而保证实体完整性

10.2 数据完整性

- 触发器Trigger
  - “Event-Condition-Action Rules”

  When *event* occurs, check *condition*; if true, do *action*
- When use them?
  - Move logic from applications to DBMS
  - To enforce constraints
    - Expressiveness
    - Constraint “repair” logic

10.2 数据完整性

- 触发器Trigger SQL标准写法
  - “Event-Condition-Action Rules”
  - 不同数据库系统存在较大差异

Create Trigger name
Before | After | Instead Of events
[referencing-variables]
[For Each Row]
When (condition)
action

10.2.1 完整性约束条件的定义

- 类型
  - Key
  - Referential integrity (foreign key)
  - Unique
  - Not Null
  - Attribute-based (Check)
  - Tuple-based (Check)
  - General assertion (大部分数据库系统没有实现)
- 不同数据库系统中不同实现方式

10.2.2 触发器

- 触发器是一类特殊的过程。触发器中规定用户在对数据库表(关系)执行INSERT、UPDATE、DELETE等操作时，数据库系统应该执行什么相关的操作以保证数据的完整性
  - 不同数据库系统实现存在较大差异
- Event-Condition-Action Rules

When event occurs, check condition; if true, do action

10.2.2 触发器

- Events
  - Insert on T [new]
  - Delete on T [old]
  - Update [of C_1, …, C_n] on T [old, new]
- For Each Row
  - Once for each modified tuple
  - Tuple level / Statement (transaction) level
- Referencing-variables
  - Old row as var
  - New row as var
  - Old table as var
  - New table as var

SQL标准
Create Trigger name
Before | After | Instead Of events
[referencing-variables]
[For Each Row]
When (condition)
action

10.2.2 触发器

- Condition
  - Like SQL where
- Action
  - SQL statement

SQL标准
Create Trigger name
Before | After | Instead Of events
[referencing-variables]
[For Each Row]
When (condition)
action

10.2.2 触发器

- 举例：用触发器实现参考完整性
  - R.A references S.B, cascaded delete

```
Create Trigger Trigger1
After Delete On S
Referencing Old Row As O
For Each Row
[No condition]
Delete From R where A = O.B
```

思考：这是tuple level，还是statement level?

SQL标准
Create Trigger name
Before | After | Instead Of events
[referencing-variables]
[For Each Row]
When (condition)
action

10.2.2 触发器

- 举例：用触发器实现参考完整性
  - R.A references S.B, cascaded delete
  - 如何用statement level实现？

```
Create Trigger Trigger2
After Delete On S
Referencing Old Table As OT
[For Each Row]
[No condition]
Delete From R where A in (select B from OT)
```

10.2.2 触发器

- 举例：用触发器实现参考完整性
  - R.A references S.B, cascaded update

```
Create Trigger Trigger3
After Update of B On S
Referencing Old Row As O, New Row As N
For Each Row
[No condition]
Update R Set A = N.B Where A = O.B;
```

思考：如何用触发器实现外码修改时的其他处理方式?

10.2.2 触发器

- 举例：用触发器实现实体完整性
  - 关系R的主码为属性A

```
Create Trigger Trigger4
Before Insert On R
Referencing New Row As N
For Each Row
When exists (select * from R where A = N.A)
select raise(ignore);                    -- SQLite raise error
```

10.2.2 触发器

- 举例：Self-triggering: T(A)

```
Create Trigger Trigger5
After Insert On T
Referencing New Row As N
For Each Row
When (select count(*) from T) < 100
insert into T values(N.A + 1);

Insert into T values (1)
```

思考：在哪些数据库系统中会产生自我触发?

10.2.2 触发器

- 举例：Row-Level Immediate Activation
  - T1(A), T2(A)

Insert into T1 values (1);

Insert into T1 values (1);

Insert into T1 values (1);

Insert into T1 values (1)

Create trigger Trigger6

After insert on T1

Referencing New Row as N

For Each Row

insert into T2 select avg(A) from T1;

思考：哪些数据库系统是Row-Level Immediate Activation?

Insert into T1 select A + 1 from T1;

10.2.2 触发器

- 举例：创建一个名为RaiseTrig的触发器，当Employee表中某个雇员的工资涨幅大于10%时，则将此次操作记录到另一个表Salarylog中

Employee表定义：

```
CREATE TABLE Employee
(Name           CHAR(15),
Deptid          INTEGER,
Salary          DECIMAL(10,2),
Job_Title       CHAR(15))
PRIMARY KEY     (Name);
```

Salarylog表定义：

```
CREATE TABLE Salarylog
(UserName       CHAR(30),
EmpName         CHAR(30),
OldSalary       DECIMAL(10,2),
NewSalary       DECIMAL(10,2))
PRIMARY KEY     (UserName);
```

10.2.2 触发器

- 触发器的定义如下

```
CREATE TRIGGER  RaiseTrig
AFTER UPDATE OF (Salary) ON Employee /*触发事件*/
REFERENCING  OLD Row AS OldRow, NEW Row AS NewRow
FOR EACH ROW
WHEN((NewRow.Salary - OldRow.Salary)/OldRow.Salary > 0.1)
INSERT INTO SalaryLog
VALUES (USER，NewRow.Name，OldRow.Salary, NewRow.Salary);
```

10.2.2 触发器

- Row-level vs. Statement-level
  - New/Old Row and New/Old Table
  - Before, Instead Of
- Multiple triggers activated at same time
- Trigger actions activating other triggers (chaining)
  - Self-triggering, cycles, nested invocations
- Conditions in When vs. as part of action

10.2.2 触发器

- T(K, V) – K key, V value

Create Trigger IncreaseInserts

After Insert on T

Referencing New Row as NR, New Table As NT

For Each Row

When (Select avg(V) from T) < (Select avg(V) from NT)

Update T set V = V + 10 where K = NR.K

- No statement-level equivalent
- Nondeterministic final state

10.2.2 触发器

- 关系R(a, b)具有以下触发器

```
CREATE TRIGGER Rins
AFTER INSERT ON R
REFERENCING NEW ROW AS new
FOR EACH ROW
WHEN (new.a * new.b > 10)
INSERT INTO R VALUES (new.a - 1, new.b + 1);
```

初始时R为空集，当insert下列哪个元组时，关系R中正好包含3个元组？

A. (2, 10)　B. (3, 9)　C. (11, 1)　D. (5, 4)

学在浙大APP答题：课中互动 – 随堂测试 – 触发器

10.2.2 触发器

- PostgreSQL
  - Expressiveness/behavior = full standard row-level + statement-level, old/new row & table
  - Cumbersome & awkward syntax
  - http://www.postgresql.org/docs/current/static/plpgsql-trigger.html
- SQLite
  - Row-level only, immediate activation ➔No old/new table
- MySQL
  - Row-level only, immediate activation ➔No old/new table
  - Only one trigger per event type
  - Limited trigger chaining
- SQL Server
  - http://technet.microsoft.com/zh-cn/library/ms178110.aspx

触发器小结

- Before and After; Insert, Delete, and Update
- New and Old
- Conditions and actions
- Triggers enforcing constraints
- Trigger chaining
- Self-triggering, cycles
- Conflicts
- Nested trigger invocations

10.2.3 视图与触发器

- 视图
  - 内模式—模式—外模式
- Why use views?
  - Hide some data from some users
  - Make some query easier / more natural
  - Modularity of database access
- Real applications tend to use lots and lots (and lost and lost!) of views

10.2.3 视图与触发器

- 定义和使用视图
  - View V = ViewQuery(R_1, R_2, …, R_n)
  - Schema of V is schema of query result
  - Query Q involving V, conceptually:

  V := ViewQuery(R_1, R_2, …, R_n)

  Evaluate(Q)
  - In reality, Q rewritten to use R_1, …, R_n instead of V
    - R_i could itself be a view

10.2.3 视图与触发器

- 视图的特点
  - 虚表，是从一个或几个基本表(或视图)导出的表
  - 只存放视图的定义，不会出现数据冗余
  - 基表中的数据发生变化，从视图中查询出的数据也随之改变

10.2.3 视图与触发器

- 视图的作用
  - 视图能够简化用户的操作
  - 视图使用户能以多种角度看待同一数据
  - 视图对重构数据库提供了一定程度的逻辑独立性
  - 视图能够对机密数据提供安全保护

10.2.3 视图与触发器

- 语句格式

  CREATE VIEW

  <视图名> [(<列名> [, <列名>]…)]

  AS <子查询>

  [WITH CHECK OPTION];

- 其中子查询可以是任意复杂的SELECT语句，但通常不允许含有ORDER BY子句和DISTINCT短语
- WITH CHECK OPTION表示对视图进行UPDATE，INSERT和DELETE操作时要保证更新、插入或删除的元组满足视图定义中子查询的WHERE子句中的条件表达式

10.2.3 视图与触发器

- 组成视图的属性列名或者全部省略或者全部指定，没有第三种选择。如果省略了视图的各个属性列名，则隐含该视图由子查询中SELECT子句目标列中的诸字段组成。下列三种情况下必须明确指定组成视图的所有列名
  - 某个目标列不是单纯的属性名，而是聚集函数或列表达式
  - 多表连接时选出了几个同名列作为视图的列
  - 需要在视图中为某个列启用更合适的名字

10.2.3 视图与触发器

- 若一个视图是从单个表导出的，并且只是去掉了表的某些行和某些列，但保留了主码，称这类视图为行列子集视图

- 例：建立计算机系学生的视图

```
CREATE VIEW S_CS
    AS
    SELECT Sno, Sname, Sage
    FROM    S
    WHERE  Sdept= ‘CS’;
```

10.2.3 视图与触发器

- WITH CHECK OPTION
  - 通过视图进行增删改操作时，不得破坏视图定义中的谓词条件(即子查询中的条件表达式)

```
CREATE VIEW CS_S
        AS
        SELECT Sno, Sname, Sage
        FROM  S
        WHERE  Sdept= ‘CS'
        WITH CHECK OPTION;
```

10.2.3 视图与触发器

- 修改操作：DBMS自动加上Sdept= ‘CS'的条件
- 删除操作：DBMS自动加上Sdept= ‘CS'的条件
- 插入操作：DBMS自动检查Sdept属性值是否为‘CS‘

- 如果不是，则拒绝该插入操作
- 如果没有提供Sdept属性值，则自动定义Sdept为‘CS'

10.2.3 视图与触发器

- 视图可以建立在
  - 单个表
  - 多个表 (基于多个基表的视图)
  - 一个或多个视图 (基于视图的视图)
  - 表和视图

10.2.3 视图与触发器

- 定义基本表时，为了减少数据库中的冗余数据，表中只存放基本数据，由基本数据经过各种计算派生出的数据一般不存储
- 由于视图中的数据并不实际存储，所以定义视图时可以根据应用的需要，设置一些派生属性列。这些派生属性由于在表中并不实际存在也称它们为虚拟列。带虚拟列的视图也称为带表达式的视图

10.2.3 视图与触发器

- 例：定义一个反映学生出生年份的视图

```
CREATE  VIEW BT_S(Sno, Sname, Sbirth)
    AS
SELECT Sno, Sname, 2020 - Sage
FROM  S
```

10.2.3 视图与触发器

- 用带有聚集函数和GROUP BY子句的查询来定义视图，这种视图称为分组视图
- 例：将学生的学号及他的平均成绩定义为一个视图，假设SC表中“成绩”列Grade为数字型

```
CREAT  VIEW S_G(Sno, Gavg)
    AS
      SELECT Sno, AVG(Grade)
      FROM  SC
      GROUP BY Sno;
```

10.2.3 视图与触发器

- 一类不易扩充的视图
  - 以 SELECT * 方式创建的视图可扩充性差，应尽可能避免
  - 例：将S表中所有女生记录定义为一个视图

```
CREATE VIEW
        F_S1(stdnum, name, sex, age, dept)
 AS  SELECT *
 FROM  S
 WHERE Ssex='女';
```

  - 缺点：修改基表S的结构后，S表与F_S1视图的映象关系被破坏，导致该视图不能正确工作

思考：这类视图违背了数据的什么独立性？

10.2.3 视图与触发器

- 删除视图

  DROP VIEW <视图名>;

  - 该语句从数据字典中删除指定的视图定义
  - 由该视图导出的其他视图定义仍在数据字典中，但已不能使用，必须显式删除
  - 删除基表时，由该基表导出的所有视图定义都必须显式删除
  - 例：删除视图S_CS

    DROP VIEW S_CS;

10.2.3 视图与触发器

- 从用户角度：查询视图与查询基本表相同
  - [例]在计算机系学生的视图中找出年龄小于19岁的学生的学号和年龄

    SELECT Sno, Sage FROM S_CS WHERE Sage<19

  - DBMS执行对视图的查询时，首先将其转化成基本表，然后再在基本表上执行查询操作。本例转换后的查询语句为

    SELECT Sno, Sage
    FROM S
    WHERE Sdept = 'CS' AND Sage < 19;

10.2.3 视图与触发器

- 在一般情况下，视图查询的转换是直接了当的。但有些情况下，这种转换不能直接进行，查询时就会出现问题
  - [例]在S_G视图中查询平均成绩在90分以上的学生学号和平均成绩

```
SELECT *
FROM   S_G
WHERE  Gavg >= 90;
```

```
SELECT Sno, AVG(Grade)
FROM     SC
WHERE  AVG(Grade)>=90
GROUP BY Sno;
```

  - S_G视图定义：

```
CREATE VIEW S_G (Sno, Gavg)
AS
   SELECT  Sno, AVG(Grade)
    FROM  SC
    GROUP BY Sno;
```

```
SELECT  Sno，AVG(Grade)
FROM  SC
GROUP BY Sno
HAVING AVG(Grade)>=90;
```

10.2.3 视图与触发器

- 更新视图是指通过视图来插入(INSERT)、删除(DELETE)和修改(UPDATE)数据
- 由于视图是不实际存储数据的虚表，因此对视图的更新，最终要转换为对基本表的更新

10.2.3 视图与触发器

- [例]在计算机系学生视图中，将学号为2000012的学生的姓名改为李大勇

```
UPDATE  S_CS
SET  Sname= '李大勇'
WHERE  Sno= ‘2000012’;
```

- 转换后的更新语句为：

```
UPDATE S
SET Sname= '李大勇'
WHERE Sno= '2000012' AND Sdept= ‘CS‘;
```

10.2.3 视图与触发器

- 一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新

```
UPDATE  S_G
SET Gavg=90
WHERE  Sno= '2000012';
```

- 无法将其转换成对基本表SC的更新

10.2.3 视图与触发器

- 视图修改
  - 视图*V*可以像基本表一样修改吗？
  - Doesn’t make sense: *V* is not stored
  - Has to make sense: views are some users’ entire “view” of the database
  - Solution: Modifications to *V* rewritten to modify base tables
  - One-to-Many translations

10.2.3 视图与触发器

- Modifications to *V* rewritten to modify base tables
- (1) Rewriting process specified explicitly by view creator [Instead-of triggers]
  - \+ Can handle all modifications
  - \- No guarantee of correctness (or meaningful)
- (2) Restrict views + modifications so that translation to base table modification is meaningful and unambiguous [SQL standard]
  - \+ No user intervention
  - \- Restrictions are significant

10.2.3 视图与触发器

```
Create view MathGrade (Sno, Sname, Grade)
AS SELECT S.Sno, Sname, Grade
FROM  S, SC
WHERE  S.Sno=SC.Sno AND SC.Cno= 'C02';
```

10.2.3 视图与触发器

- MathGrade (Sno, Sname, Grade)

Create trigger MathGradeDelete

Instead of delete on MathGrade

Referencing Old Row As O

For Each Row

Delete from SC where Sno = O.Sno and SC.Cno = 'C02';

10.2.3 视图与触发器

- MathGrade (Sno, Sname, Grade)

```
Create trigger MathGradeUpdate
Instead of update of Sname on MathGrade
Referencing New Row As N
For Each Row
Update S
Set Sname = N.Sname
Where Sno = N.Sno
```

10.2.3 视图与触发器

- MathGrade (Sno, Sname, Grade)

Create trigger MathGradeInsert1
Instead of insert on MathGrade
Referencing New Row As N
For Each Row
Insert Into SC(Sno, Cno, Grade)
Values(N.Sno, ‘C02’, N.Grade)

10.2.3 视图与触发器

- MathGrade (Sno, Sname, Grade)
- 视图修改语义取决于用户

Create trigger MathGradeInsert2

Instead of insert on MathGrade

Referencing New Row As N

For Each Row

Insert Into S(Sno, Sname) Values(N.Sno, N.Sname)

Insert Into SC(Sno, Cno, Grade) Values(N.Sno, 'C02', N.Grade)

10.2.3 视图与触发器

- Instead of Trigger在不同数据库系统上的实现
  - Works for SQLite
  - PostgreSQL uses different rules/trigger syntax
  - MySQL doesn’t support rule-based view modifications
  - SQL Server?

10.2.3 视图与触发器

- Automatic View Modifications
- SQL standard for “updatable views”
  - Select (no Distinct) on single table T
  - Attributes not in view can be ‘NULL’ or have default value
  - Subqueries must not refer to T
  - No Group by or aggregation
- With Check Option

10.2.3 视图与触发器

- SQL Server可更新视图标准
  - 任何修改(包括 UPDATE、INSERT 和 DELETE 语句)都只能引用一个基表的列
  - 被修改的列不受 GROUP BY、HAVING 或 DISTINCT 子句的影响
  - TOP 在视图的 select_statement 中的任何位置都不会与 WITH CHECK OPTION 子句一起使用

http://technet.microsoft.com/zh-cn/library/ms187956.aspx

10.2.3 视图与触发器

- SQL Server可更新视图标准
  - 视图中被修改的列必须直接引用表列中的基础数据。 不能通过任何其他方式对这些列进行派生，如通过以下方式
    - 聚合函数：AVG、COUNT、SUM、MIN、MAX、GROUPING、STDEV、STDEVP、VAR 和 VARP
    - 计算。 不能从使用其他列的表达式中计算该列。 使用集合运算符 UNION、UNION ALL、CROSSJOIN、EXCEPT 和 INTERSECT 形成的列将计入计算结果，且不可更新

http://technet.microsoft.com/zh-cn/library/ms187956.aspx

10.2.3 视图与触发器

- PostgreSQL可更新视图标准
  - The view must have exactly one entry in its FROM list, which must be a table or another updatable view.
  - The view definition must not contain WITH, DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clauses at the top level.
  - The view definition must not contain set operations (UNION, INTERSECT or EXCEPT) at the top level.
  - All columns in the view's select list must be simple references to columns of the underlying relation. They cannot be expressions, literals or functions. System columns cannot be referenced, either.
  - No column of the underlying relation can appear more than once in the view's select list.
  - The view must not have the security_barrier property.

http://www.postgresql.org/docs/current/static/sql-createview.html

10.2.3 视图与触发器

- Virtual Views → Materialized Views (物化视图)
  - Hide some data from some users
  - Make some query easier / more natural
  - Modularity of database access
  - Improve query performance

10.2.3 视图与触发器

- Materialized Views
  - View V = ViewQuery(R_1, R_2, …, R_n)
  - Create table V with schema of query result
  - Execute ViewQuery and put results in V
  - Queries refer to V as if it's a table
- But
  - V could be very large
  - Modifications to R_1, R_2, …, R_n ➔ recompute or modify V

10.2.3 视图与触发器

- Materialized Views
  - Modifications to base data invalidate view

```
create materialized view Student_Course as
select S.Sno as Sno, Sname, C.Cno as Cno, Cname, grade
from S, C, SC
where S.Sno = SC.Sno and C.Cno = SC.Cno
```

10.2.3 视图与触发器

- Modifications on materialized views?
  - Good news: just update the stored table
  - Bad news: base table must stay in synch
    - Same issues as with virtual views
- Materialized views vs. index
  - 提高查询效率
  - 影响数据修改的效率
- SQL Server – indexed views
  - Create a regular view
  - Create a clustered index on that view

http://stackoverflow.com/questions/3986366/how-to-create-materialized-views-in-sql-server
http://www.postgresql.org/docs/current/static/sql-creatematerializedview.html

10.2.3 视图与触发器

- (Efficiency) benefits of a materialized view depend on
  - Size of data
  - Complexity of view
  - Number of queries using view
  - Number of modifications affecting view
    - Also “incremental maintenance” vs. full recomputation
    - Trade off between query and update
- DBMS自动将使用基表的SQL语句转化为使用materialized view的SQL语句
  - 与索引相同，DBMS自动决定是否采用某一索引

视图与触发器小结

- 视图作用
  - Hide some data from some users
  - Make some query easier / more natural
  - Modularity of database access
  - Improve query performance [Materialized Views]
- 视图修改
  - Instead of trigger
  - 遵循SQL标准，通过With Check Option
  - 不同数据库系统，视图是否可修改标准不同
- 现实系统应用中使用大量视图

10.2.4 完整性约束条件的修改

- SQL允许在任何时候增加、删除一个完整性约束条件，为此首先要对完整性约束条件命名
- 要为完整性约束条件命名，需要在完整性约束条件前增加一个关键字CONSTRAINT，后面跟上一个名字

10.2.4 完整性约束条件的修改

将S表的码约束条件命名为SKey

```
CREATE TABLE S
(Sno      CHAR(7) ,
Sname  CHAR(8) NOT NULL,
Ssex      CHAR(2) ,
Sage      SMALLINT,
Sdept   CHAR(20),
CONSTRAINT  SKey  PRIMARY KEY(Sno));
```

10.2.4 完整性约束条件的修改

- 在表中增加一个完整性约束条件
- 使用关键字ADD
- 举例：增加表S中Ssex只能取’男’和’女’的约束
  - ALTER TABLE S ADD CONSTRAINT Gender CHECK (Ssex IN (‘男’, ’女’));

10.2.4 完整性约束条件的修改

- 在表中删除一个完整性约束条件
- 使用关键字DROP
- 举例：去掉表S中的SKey限制
  - ALTER TABLE S DROP CONSTRAINT SKey;

- 删除触发器
- 使用关键字Drop Trigger TriggerName
- 举例：删除触发器R1
  - Drop Trigger R1

10.2.5 完整性约束条件的检查和违约处理

- RDBMS中检查数据是否满足完整性约束条件的机制称为完整性检查
- 完整性约束条件检查的时机通常是在一条语句执行完后立即检查，这类约束称为立即执行的约束
- 但在某些情况下，完整性检查需要延迟到整个事务执行结束后再进行，这类约束称为延迟执行的约束

10.2.5 完整性约束条件的检查和违约处理

- 举例：表S中已经存在一个学号为2000015的学生，则下列插入语句将由于违反了实体完整性(主码惟一性)而被系统拒绝

```
INSERT INTO S (Sid, Sname, Ssex, Sage, Sdept)
    VALUE (2000015, '刘燕', '女', 20, '信息')
```

思考：这是立即执行的约束，还是延迟执行的约束？

10.2.5 完整性约束条件的检查和违约处理

- 举例：下列插入语句将由于违反了主码不能取空值的约束条件而被系统拒绝

```
INSERT INTO S (Sid, Sname, Ssex, Sage, Sdept)
    VALUE (NULL, '刘燕', '女', 20, '信息')
```

10.2.5 完整性约束条件的检查和违约处理

- 参照完整性将两个表中的相应元组联系起来，因此，对被参照表和参照表进行增删改操作时有可能破坏参照完整性
- 当参照完整性被破坏时，系统可以采用以下的策略加以处理
  - 拒绝(reject)
  - 级连(cascade)删除
  - 设置为空值(set-null)

第十章 数据库安全性与完整性

- 10.1 数据库的安全性
  - 10.1.1 数据库安全性
  - 10.1.4 存取控制
- 10.2 数据库的完整性
  - 10.2.1 完整性约束条件的定义
  - 10.2.2 触发器
  - 10.2.3 视图与触发器
  - 10.2.4 完整性约束条件的修改
  - 10.2.5 完整性约束条件的检查和违约处理

参考教材：
数据库系统概念 4.2，4.4，4.6，5.3，13.5