数据库原理

1. 介绍

1.1. DBMS功能

1.1.1. 设计

DDL: data definition language

1.1.2. 操作

DML: data manipulation language

1.1.3. 数据管理

1.2. DBMS组成

1.2.1. 查询处理

1.2.2. 事务管理

1.2.3. 存储管理

1.3. 数据库结构

1.3.1. Centralized databases

1.3.2. Client-server

1.3.3. Parallel databases

1.3.4. Distributed databases

2. 关系数据模型

2.1. 数据模型

2.1.1. 组成：数据结构、数据操作、约束

2.1.2. 两种重要的数据模型

关系模型

半结构化模型：对数据自带结构化描述

2.2. 关系

2.2.1. 关系是一个表格

2.2.2. Schemas (关系模式)

Relation schema

Database

关系的集合

Database schema

数据库中所有关系对应模式的集合

2.2.3. Relation Instances (关系实例)

关系模式对应的一行数据

2.2.4. Key of Relations（键）

2.3. SQL中的关系模型

2.3.1. SQL操作

创建关系

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

删除关系

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

修改关系

增加属性

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

删除属性

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

2.3.2. 三种表

Stored relations

Views

Temporary tables

2.4. 关系代数

2.4.1. 选择

2.4.2. 投影

2.4.3. 拓展投影

可以加入新的属性

2.4.4. 积（cross join）

2.4.5. Theta-Join

先对表乘积，然后应用选择

连接满足条件的关系，两个关系中所有属性都在结果中

2.4.6. Natural Join

同名属性的值相等

每对同名属性只保留一个在结果中

2.4.7. Out join

如果关系在另一张表中找不到对应关系，依然保留到结果中，无法匹配的属性用 null 代替

变种

theta-outerjoin

不再合并同名属性

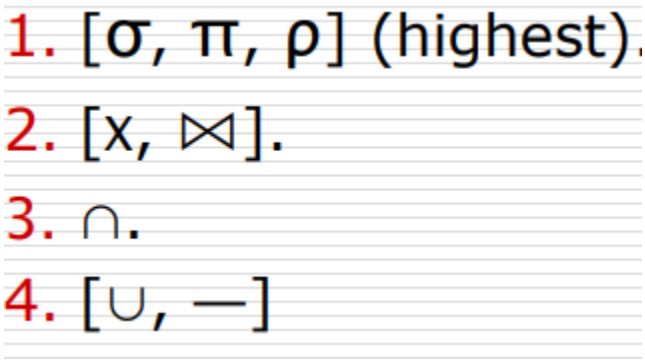
left-outerjoin

right-outerjoin

2.4.8. 重命名

2.5. 关系运算表达式

2.5.1. 运算优先级



2.5.2. 三种表示

赋值表达式序列

单个赋值表达式

表达式树

2.6. 包

2.6.1. 包类似于集合，但允许值重复出现

2.6.2. SQL是包语言

包上的操作效率更高

2.6.3. 包的操作

投影操作不对结果去重

并集操作对出现次数取和

交集操作对出现次数取最小值

差集操作对出现次数作减法

选择、乘积、连接操作无影响，把每一行都视作不同项即可

2.7. 约束

2.7.1. R = 0

2.7.2. R ⊆ S

2.8. 拓展关系代数

2.8.1. 去重（δ）

2.8.2. 排序（τ）

2.8.3. 拓展投影

2.8.4. 分组（γ）

3. 关系数据库设计理论

3.1. 函数依赖

3.1.1. 定义

X->Y：任意两个关系，如果属性**X**相同，那么属性**Y**也一定相同

3.1.2. 右侧拆分

X->AB等价于**X->A+X->B**

3.1.3. trivial

如果函数依赖右侧是左侧的子集则**trivial**

3.1.4. 键（keys）

超键（**superkey**）

若**K**可以函数决定**R**中的所有属性，则**K**是**R**的超键

键（key）

K是一个超键，且任何真子集不是超键

键的来源

定义关系时限定的键

从函数依赖推出

从物理意义推出

3.1.5. 函数依赖的推导

使用函数依赖推导

闭包

3.2. 关系模式设计

3.2.1. 异常

Update anomaly

修改某个函数依赖时，可能忘记修改所有出现处

Deletion anomaly

删除某个关系时，可能丢失一些有效信息

3.2.2. 优秀的设计

没有冗余

没有异常

3.2.3. 拆分原则

连接后可以恢复原来的关系

3.3. 多值依赖

参见: 4NF

3.3.1. 定义

X->->Y: 如果两个元组的**X**相同，那么交换**Y**之后的新元组也全在关系中

3.3.2. MVD规则

每个**FD**都是**MVD**

互补：如果**X->->Y**，**Z**是其他所有属性，那么**X->->Z**

交集：如果**X->->Y**且**X->->Z**，那么**X->->Y**交**Z**

传递性：如果**X->->Y**且**Y->->Z**，那么**X->->Z-Y**

3.3.3. trivial

X->->Y，**Y**不是**X**的子集且**X+Y**不包含全部属性

3.4. 范式

3.4.1. 4NF

参见: 多值依赖

条件

任意多值依赖**X->->Y**是 **nontrivial MVD**，且**X**是超键

由于**FD**都是**MVD**，所以**4NF**一定满足**BCNF**

分解

1、 计算所有的key

2、 找到所有违反4NF的MVD并拆分

3.4.2. BCNF

条件

关系R中的任意nontrivial函数依赖X->Y都必须满足X是R的超键

只要找到一个函数依赖的左边不是超键，关系就不满足BCNF

分解

1、 找到一个不满足BCNF的函数依赖 X->Y

2、 计算X的闭包

3、 拆分为两部分，分别为X的闭包以及剩下部分+X

4、 投影，将函数依赖投影到新关系

无法保留所有的函数依赖

3.4.3. 3NF

prime

如果属性属于某个key，则该属性是一个prime

条件

nontrivial函数依赖X->A违反3NF当且仅当X不是超键且A不是prime

分解

要求

连接无损失

函数依赖关系保留

3NF Synthesis Algorithm

构造Minimal Basis

将所有FD右侧拆分为单属性

不断尝试移除FD

不断尝试移除FD的左侧属性

为每个Minimal Basis中的FD创建一个关系

如果上述的FD中不包含任意一个key，额外创建一个关系，属性为一个key所包括的属性

原理

Minimal Basis中每个FD至少存在于一个关系中，因此所有FD可以被还原

键被至少一个关系保留，原关系可以被还原

3.4.4. 2NF

条件

所有的非键属性必须函数依赖于全部的主键属性

3.4.5. 1NF

条件

列不可再拆分

3.4.6. 总结

Property	3NF	BCNF	4NF
Eliminates Redundancy due to FDs	Most	Yes	Yes
Eliminates redundancy Due to MVDs	No	No	Yes
Preserves FDs	Yes	maybe	maybe
Preserves MVD	maybe	maybe	maybe
Equal to original relation	Yes	Yes	Yes

4. 高层数据库模型

4.1. ER模型

4.1.1. 图形

实体：矩形

数学：椭圆

关系：菱形

4.1.2. 关系

Binary

Multiway

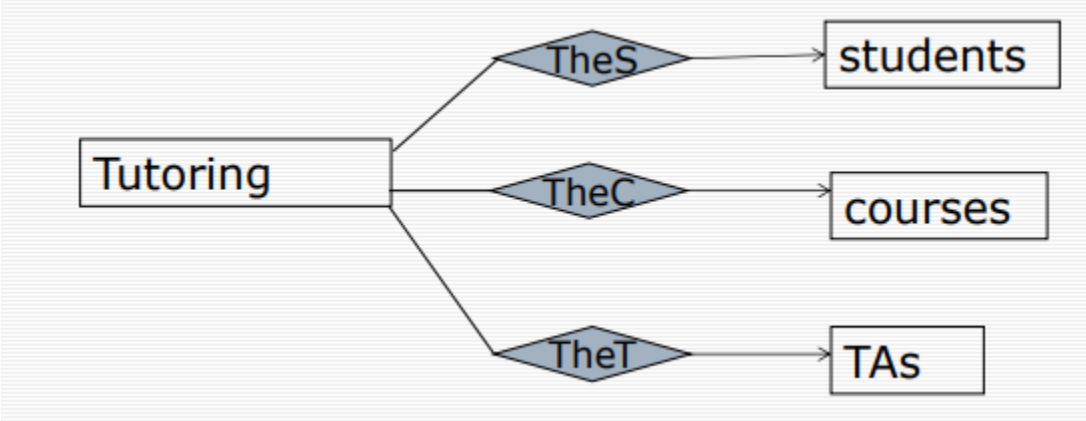
一个关系可能涉及多个实体

Multiplicity

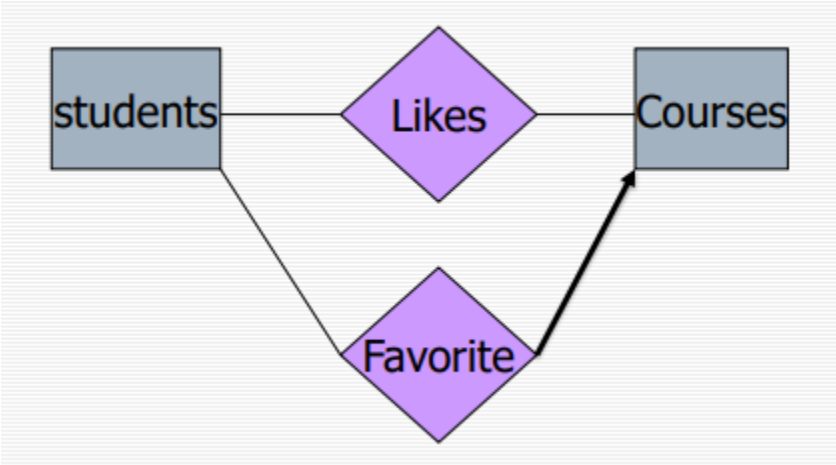
一个实体可以通过关系关联多个实体

4.1.3. 关系表示

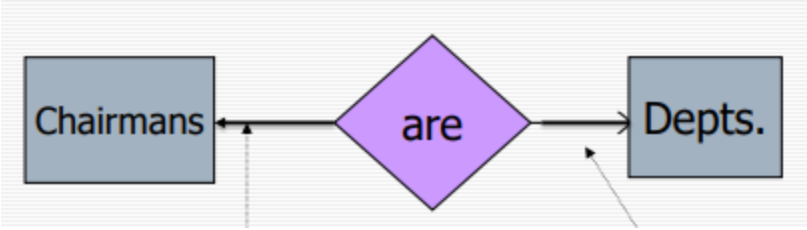
Multi-way relationship



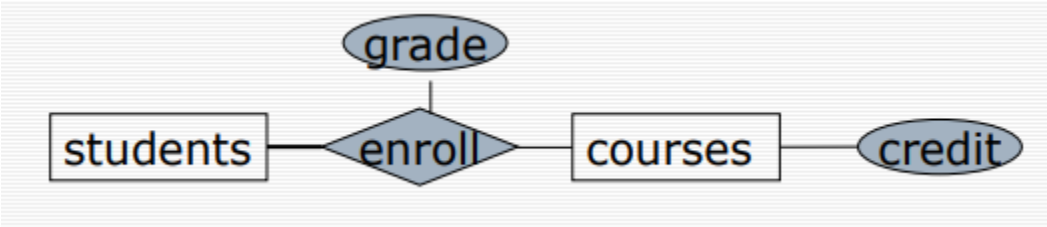
Many-Many and Many-One Relationship



One-One Relationship



Attributes on Relationships



4.1.4. key

每个实体集合至少有一个key，且刚好有一个指定的key

Weak Entity Sets

某个实体集合的key（不完全）来自自己的属性，而是来自于链接的其他实体

在ER图中，weak entity set用双框矩阵表示，对应关系由双框菱形表示

4.1.5. Aggregation

一个实体可能和某个关系存在关联，但并非所有该关系都对应一个实体（因此将实体设计为关系的属性会导致冗余）

将关系看成一个抽象实体

4.1.6. Subclasses

参见: Subclasses

完全/部分

不相交/重叠

子类拆分

Object-oriented

Use nulls

E/R style

4.1.7. 设计技巧

避免冗余

浪费空间且可能导致不一致性

避免相同属性在不同实体中

避免相同关系被重复保存（FD）

对于只有一个属性的实体，可以作为属性合并到其他实体

限制弱实体集

4.1.8. 约束

Key constraints

Single-value constraint

属性有一个值，不能为空

多对一关系右侧的实体集只能关联一个实体

Referential integrity constraints

一个属性必须出现在另一个实体集中（外键）

Domain constraints

General constrains

4.2. UML

4.2.1. Classes

组成

属性

方法：需要标注参数与返回类型

主键：指示某属性为对象主键

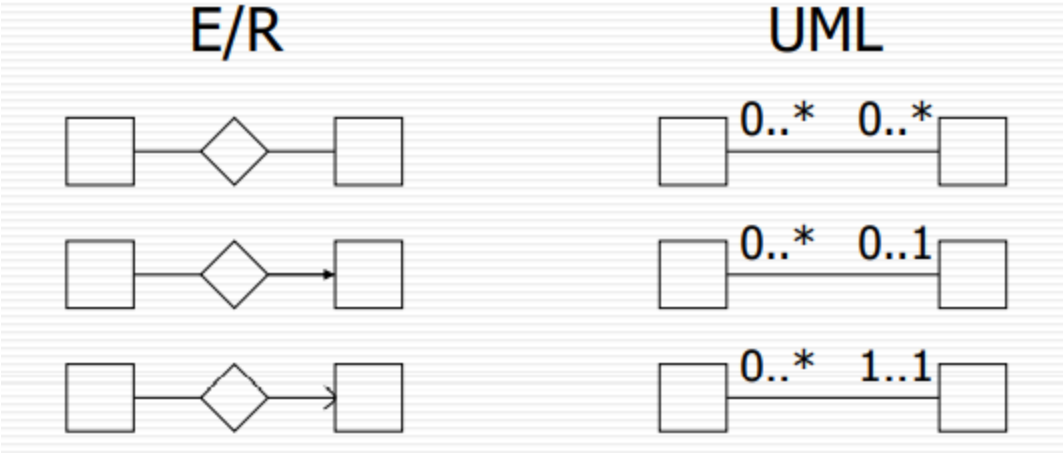
4.2.2. 关联

用带名字的线表示

用数字表示双方关联的数量

关联上的属性用**Association Class**表示

与**ER**对比



4.2.3. Subclasses

参见: Subclasses

传统OO中不允许重叠

UML允许重叠

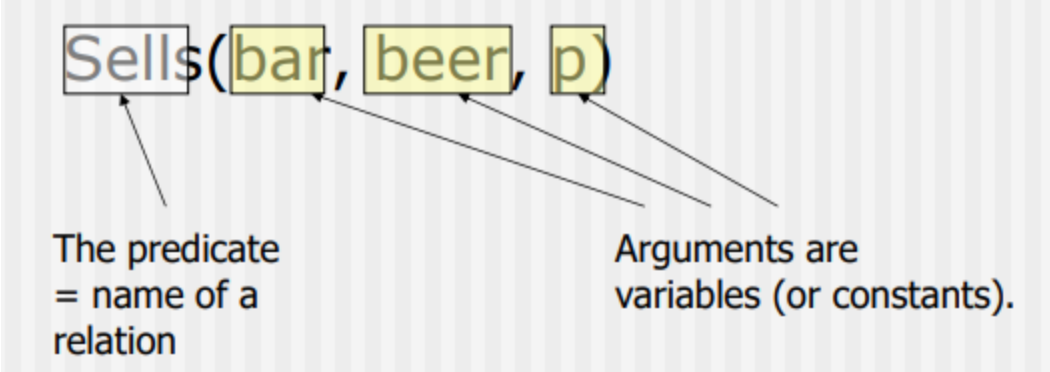
完全/部分

5. 代数逻辑查询语言

5.1. 预测与原子

5.1.1. 预测与其参数的组合被称为原子

5.1.2. Atom：预测+参数



5.1.3. 预测：关系名 或 代数逻辑判断（例如大小比较）

5.1.4. 参数：变量 或 约束

5.2. 语法规则

5.2.1. Head <- Body

5.2.2. Head为一个atom

5.2.3. Body为一个或者多个被AND连接的atom

5.2.4. 命名习惯：预测以大写字母开始，变量以小写字母开始

5.3. 应用规则

5.3.1. 方法**1**

判断所有变量可能取值的全部组合

example

```
Happy(d) <- Frequents(d,bar) AND
      Likes(d,beer) AND Sells(bar,beer,p)
FOR (each d, bar, beer, p)
  IF (Frequents(d,bar), Likes(d,beer), and
  Sells(bar,beer,p) are all true)
      add Happy(d) to the result
```

5.3.2. 方法2

选择满足每个子目标的元组，判断一组选择是否为每个变量定义了一个单值

5.4. Safe Rules

5.4.1. 条件

所有Head、代数子目标、否定子目标中的变量至少出现在一个非否定、关系子目标中

5.4.2. Unsafe Rules可能有无限种取值满足条件，即使右侧的关系全是有限的

5.4.3. 意义

保证所有变量的取值在一个有限域内

5.5. Datalog Programs

5.5.1. Datalog program：规则集合

5.5.2. 预测

EDB = Extensional Database = stored table

IDB = Intensional Database = relation defined by rules

预测不能同时属于EDB与IDB

EDB不能在head

6. SQL

6.1. Components

6.1.1. Schema definition, Data retrieval, Data modification, Indexes, Constraints, Views, Triggers, Transactions, authorization,etc

6.2. Queries

6.2.1.

```
SELECT <desired attributes>
FROM <tuple variables or relation name>
WHERE <conditions>
GROUP BY <attributes>
HAVING <conditions>
ORDER BY < list of attributes>
```

6.2.2. Star as List of All Attributes

6.2.3. Renaming columns

SELECT <column> AS <new-name> FROM ...

6.2.4. Expressions as Values in Columns

SELECT <expressions/constant> AS <new-name> FROM ...

6.2.5. Patterns

.... WHERE <column> LIKE/NOT LIKE <pattern> [ESCAPE <char>]

操作符

%：任意长度字符串

_：任意单个字符

转义

在LIKE语句后使用ESCAPE指定转义字符

6.3. NULLs

6.3.1. 意义

缺少值：元组存在某个属性，但是不知道具体值

不适用：元组对应的对象不存在这个属性

6.3.2. 操作

任何值（包括NULL）与NULL比较结果均为UNKONWN

任何值与NULL进行算术操作，结果为NULL

6.3.3. 3-Valued Laws

TRUE=1，FALSE=0，UNKONWN=1/2

AND=MIN，OR=MAX，NOT(X)=1-X

对比2-Valued Laws

对于AND操作无较大差异

OR：p OR NOT p 结果可能为UNKNOWN

6.3.4. 检测NULL

IS NULL / IS NOT NULL

6.4. Multi-relation Queries

6.4.1. 在FROM后标注所有引用的关系

6.4.2. Formal Semantics

先对所有关系做积再应用WHERE

6.4.3. Operational Semantics

每一行与其他行依次组合，符合WHERE则加入结果

6.5. Subquery

6.5.1. 场合

From clause

子查询结果作为被查询的表

Where clause

子查询结果作为条件

6.5.2. scoping rule

属性引用自最近的有该属性的关系

6.5.3. Operator

IN

元组在关系中时为TRUE

Exists

关系不为空时为TRUE

ANY

至少一个元组满足时为**TRUE**

ALL

至少一个元组不满足时为**FALSE**

6.6. Aggregations

6.6.1. 包括

sum, avg, min, max, count

默认对包进行操作，使用**distinct** 去重

6.6.2. NULL在聚合操作中被忽略

NULL被**sum, avg, count**忽略

NULL不作为**max,min**的结果

如果一列的值全为**NULL**，聚合结果也为**NULL**（**count**结果为**0**）

6.6.3. count的区分

count(*)：全部元组数量

count(bar)：**bar**不为**NULL**的元组数量

count (distinct bar)：**bar**不同取值的数量（不含**NULL**）

6.6.4. Grouping

语法

在**SELECT-FROM-WHERE**后

GROUP BY <attribute list>

作用

FROM-WHERE的结果被按照给的属性分组

聚合操作只在组内进行

NULL在grouping中被当做普通值对待

6.6.5. 选择属性限制

参见: 属性限制

使用聚合之后，选择的值只能被聚合或者属于分类属性

（SQLite）对于没有聚合的属性输出每组第一个结果

6.6.6. HAVING

语法

在GROUP BY后

HAVING <condition>

作用

对每个分组进行评估，不满足条件则不会出现在结果中

可以认为是在GROUP BY的结果之后进行了一次选择

属性限制

参见: 选择属性限制

6.7. Order

6.7.1. 语法

在HAVING之后

ORDER BY < list of attributes>

每个 attributes 后可以接 ASC或DESC

6.7.2. 作用

按照属性顺序排序

默认为升序

每个属性需要单独制定升降序（否则该属性使用默认升序）

6.8. Union, Intersection and Difference

6.8.1. 语法

(subquery) UNION (subquery)

(subquery) INTERSECT (subquery)

(subquery) EXCEPT (subquery)

6.8.2. set/bag

默认情况下视为集合处理

在操作符之后使用ALL可转为包

部分DBMS没有实现ALL

6.9. Join

6.9.1. 语法

自然连接

R NATURAL JOIN S

乘积

R CROSS JOIN S

SQLite中，不加ON和乘积等价

Theta Join

R JOIN S ON <condition>

外连接

R [NATURAL] [LEFT/RIGHT/FULL] JOIN S [ON <condition>]

6.10. Modifications

6.10.1. Insertion

语法

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

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

INSERT INTO <relation> (<subquery>);

插入会在子查询全部完成后进行，子查询看不到新插入的值

6.10.2. Deletion

语法

DELETE FROM <relation> WHERE <condition>;

全部删除： **DELETE FROM <relation>;**

6.10.3. Updates

语法

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

更新**WHERE**匹配到的所有元组

6.10.4. Adding Attributes

ALTER TABLE <name> ADD
<attribute declaration>;

6.10.5. Deleting Attributes

ALTER TABLE <name>
DROP <attribute>;

6.11. Transactions

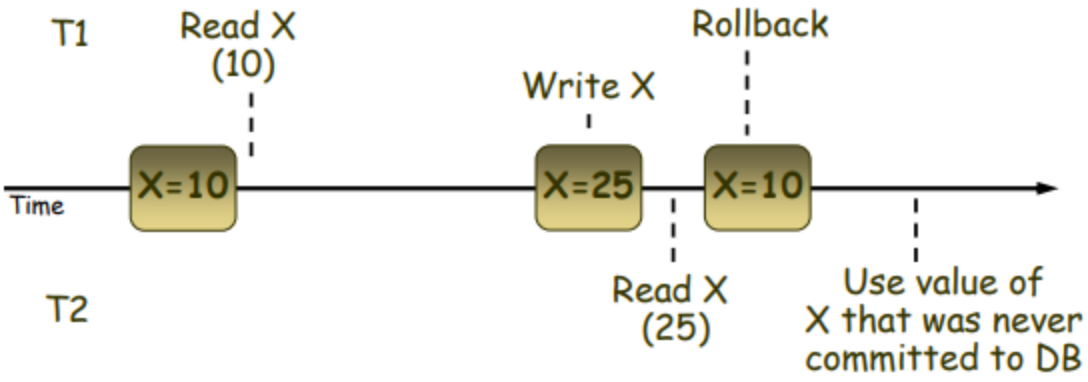
6.11.1. 问题

连续查询之间看到的数据不同

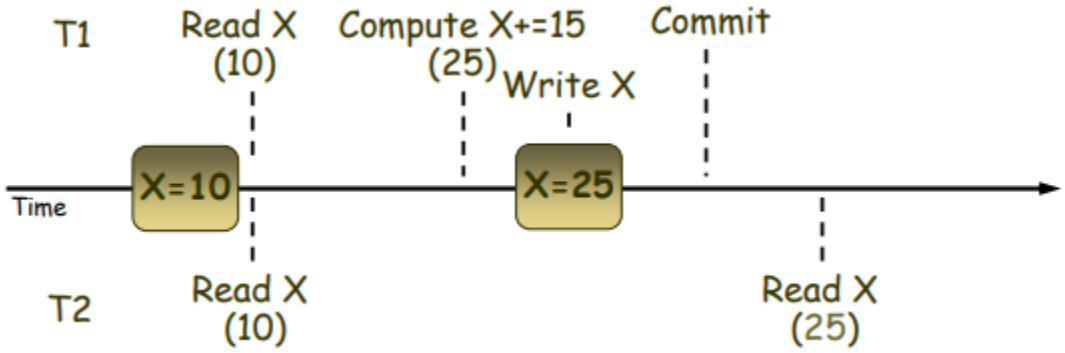
查询得到已被回滚的数据

总结

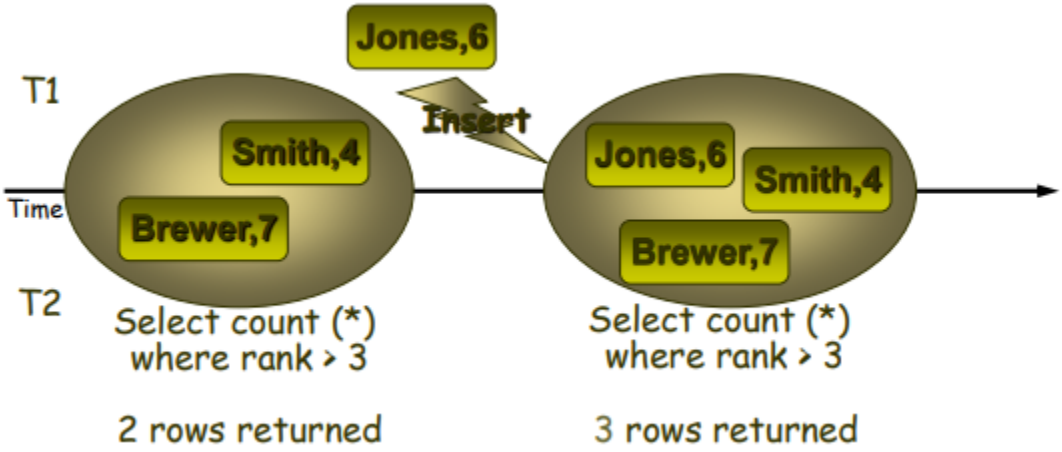
Dirty read



Non-Repeatable Read



The "Phantom" Problem



6.11.2. 定义

事务指将一个或多个SQL操作看做一个单元对待

6.11.3. 要求

参见: ACID

并发数据库访问

每个SQL执行环境都看上去是独立的

系统失效忍耐

保证要么整体执行，要么完全不执行

6.11.4. SQL标准

commit 结束一个事务并开始新事物

会话结束时完成当前事务

Autocommit 让每个语句变为一个事务

显式控制

```
Begin Transaction
…
End Transaction
```

6.11.5. 事务的结果

COMMIT

修改被持久化保存

ROLLBACK

放弃全部修改

数据库返回至事务开始状态

6.11.6. ACID

参见: 要求

Atomic

Consistent

数据库约束被保留

Isolated

Durable

事务结果可以在崩溃后保留

6.11.7. Isolation Levels

4 levels

SERIALIZABLE

看到的数据要么是其他事务开始前的，要么是其他事务结束后的

REPEATABLE READ

事务开始之后，即使过程中被其他事务删除的数据也能再次看到

事务运行过程中可能看到新COMMIT的数据

READ COMMITTED

保证看到的数据被COMMIT，但不保证每次看到的数据一致

事务运行过程中可能看到新COMMIT的数据

READ UNCOMMITTED

语法

SET TRANSACTION ISOLATION LEVEL X

Set transaction read only;

总结

isolation level	dirty reads	nonrepeatable reads	phantoms
READ UNCOMMITTED	Y	Y	Y
READ COMMITTED	N	Y	Y
REPEATABLE READ	N	N	Y
SERIALIZABLE	N	N	N

7. 约束与触发器

7.1. Why

7.1.1. 发现数据错误

7.1.2. 作为数据更新时的正确性检查

7.1.3. 保证数据一致性

7.1.4. 向DBMS描述数据特征

7.2. 定义

7.2.1. 约束是DBMS需要保证的数据元素关系

7.2.2. 触发器是特殊事件发生时需要执行的事情

7.3. Constraints

7.3.1. Non-null

7.3.2. unique

数据可能为NULL

但是非NULL的数据不能重复

7.3.3. Key

不能为NULL且唯一

只能有一个Key

7.3.4. Foreign Keys

任意一个非NULL值必须在被引用的列中出现

被引用的列必须PRIMARY KEY或 UNIQUE

语法

作为单独元素

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

在属性定义之后

```
REFERENCES <relation> ( <attributes> )
```

被违背后的行为

```
ON DELETE SET NULL ON UPDATE CASCADE
```

被违背时

插入或更新一个不存在与被引用列的值：拒绝

在被引用列插入或删除

Default：拒绝

Cascade：扩散修改到引用的元组

set NULL：引用的元组对应外键列被设为**NULL**

7.3.5. Attribute-based Checks

Attribute-based Check

语法

在属性定义后 **CHECK (condition)**

条件仅在相关属性变化后检查

Tuple- Based Checks

语法

作为元素添加 **CHECK (condition)**

条件在关系变化后检查

CHECK with and without subquery

子查询的关系更新不会触发检查，因此不能保证约束

7.3.6. Assertions

语法

CREATE ASSERTION <name>

CHECK (< condition>);

断言基于数据库设定

断言检查的时机

理论上只在相关关系被修改时触发

实际上可能在任何修改发生时触发

7.3.7. 比较

Type of constraints	Where Declared	When Activated	Guarranteed to Hold?
Attribute-based Check	With attribute	On insertion to relation or attribute update	Not if subqueries
Tuple-based Check	Element of relation schema	On insertion to relation or tuple update	Not if subqueries
Assertion	Element of database schema	On change to any mentioned relation	Yes

7.3.8. 约束的修改

为约束命名

语法

Gender Char(1) CONSTRAINT NoAndro CHECK
(gender in ('F','M'))

Name Char(30) CONSTRAINT NameIsKey
PRIMARY KEY

修改

语法

ALTER TABLE Student DROP CONSTRAINT
NoAndro;

ALTER TABLE Student ADD CONSTRAINT
NameIskey PRIMARY KEY(name);

7.4. Triggers

7.4.1. 动机

check能力有限

assertion难以保证效率

7.4.2. 组成

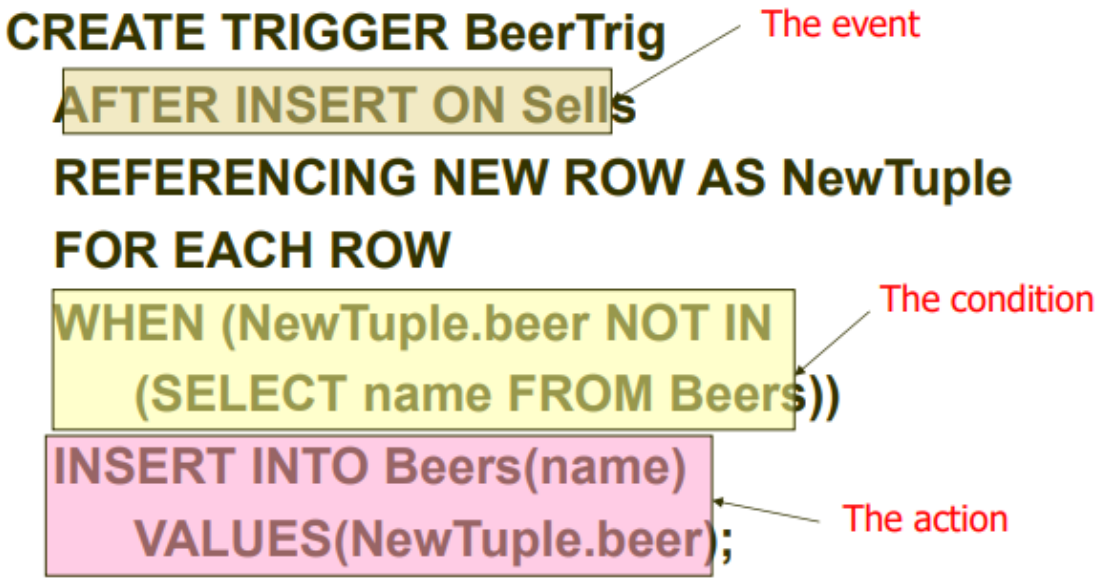
事件

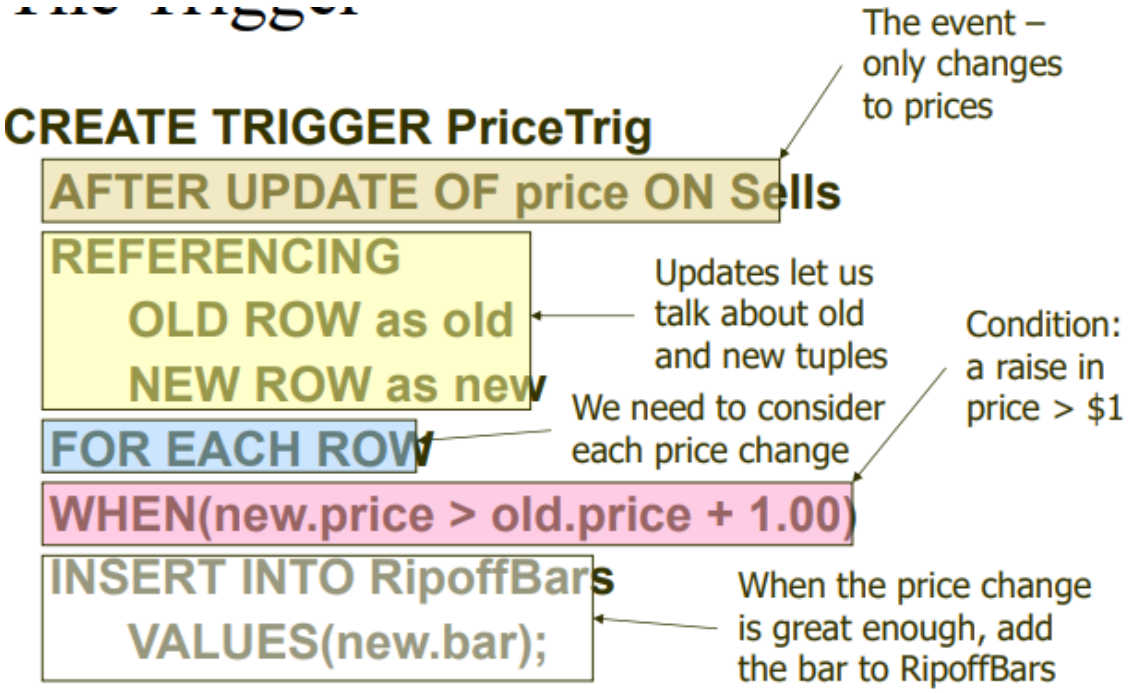
条件

行动

7.4.3. 语法

总览





CREATE TRIGGER

CREATE TRIGGER <name>

The event

AFTER/BEFORE/INSTEAD OF

INSERT/DELETE/UPDATE

UPDATE可以为UPDATE . . . ON特定属性

REFERENCING

[NEW OLD][ROW TABLE] AS <name>

对于**INSERT**，代指插入的行（可能多行）

对于**DELETE**，代指旧行

对于**UPDATE**，新旧行都可能

FOR EACH ROW

FOR EACH ROW代表每行触发一次

否则为每个语句执行一次

The Condition

WHEN <condition>

基于**BEFORE/AFTER**确定评估时间

通过**REFERENCING**语句中声明访问

The Action

对于多余一条语句的情况，使用 **BEGIN . . . END**包裹

查询在**action**中不起作用

7.4.4. Event vs. Triggers

AFTER

事件发生->测试条件

BEFOR

测试条件->事件发生

INSTEAD OF

测试条件

如果为真则执行

不执行原事件

8. 视图与索引

8.1. Views

8.1.1. 视图是一个根据存储表（base table）与其他视图定义的关系

8.1.2. Two kinds

Virtual

没有真正保存

Materialized

8.1.3. 语法

CREATE [MATERIALIZED] VIEW <name> AS <query>;

DROP VIEW <name>;

视图的查询和普通表一样

复杂视图不允许修改，部分SQL实现允许修改简单视图（定义在单个关系上，没有聚合）

8.1.4. 实现

解析语法时将视图当做base table看待

执行时将语法树展开代替视图名

8.1.5. 使用触发器修改视图

INSTEAD OF

8.1.6. Materialized Views

优缺点

缺点：base table修改时，视图也需要修改

优点：查询速度更快

增量更新：无需全部重构视图

定期更新

Materialized Views常被用于数据分析，数据可能过时

定期更新是可行的

8.2. Indexes

8.2.1. 用于加快访问特定属性的数据结构

8.2.2. 实现

balanced search tree： B tree, B+ tree

Hash index

8.2.3. 语法

No standard

CREATE INDEX <name> ON <table>(<attributes>);

8.2.4. 用法

index在查询中快速找到指定属性的值对应的元组

8.2.5. 数据库优化

优缺点

有点：加快可以使用索引的查询

缺点：减慢修改，因为需要额外修改索引

根据查询与修改频率决定索引

Tuning Advisors

意义

提供优化建议

手动优化太困难

工作流程

获取工作负载

从历史查询中随机选择

由设计者提供

生成、评估候选索引

8.2.6. 建议

在键上增加索引

在接近于键的属性上添加索引

在成簇状的属性上添加索引

9. 关系数据库进阶

9.1. Authorization

9.1.1. 目标

保证用户看到应该看到的数据

保证数据不被恶意破坏

9.1.2. Authorization ID

PUBLIC：special ID

可以指定特定的权限

9.1.3. 9种

SELECT

INSERT

DELETE

UPDATE

REFERENCES

作为外键引用

USAGE

使用模式

TRIGGER

定义关系上的触发器

EXCUTE

执行代码

UNDER

在类型下创建子类型

9.1.4. 权限获取

拥有者

拥有所有权限，可以授权

授权用户

9.1.5. 权限检查

授权ID来自于： 1、模块授权ID 2、会话授权ID

授权ID检查：1、是数据的拥有者 2、被授权 3、数据被授权给PUBLIC

9.1.6. 授权

拥有者有所有权限

如果用户有带“grant”的权限，可以将权限授权给他人

语法

GRANT<权限列表>ON<数据库元素> TO <用户列表> [WITH GRANT OPTION]

example

1) Sally can query Sells and can change prices, but cannot pass on this power:

GRANT SELECT ON Sells, UPDATE (price) ON Sells TO *sally*;

2) Sally can also pass these privileges to whom she chooses;

GRANT SELECT ON Sells, UPDATE (price) ON Sells TO sally WITH GRANT OPTION;

授权图

跟踪权限与其来源

图例

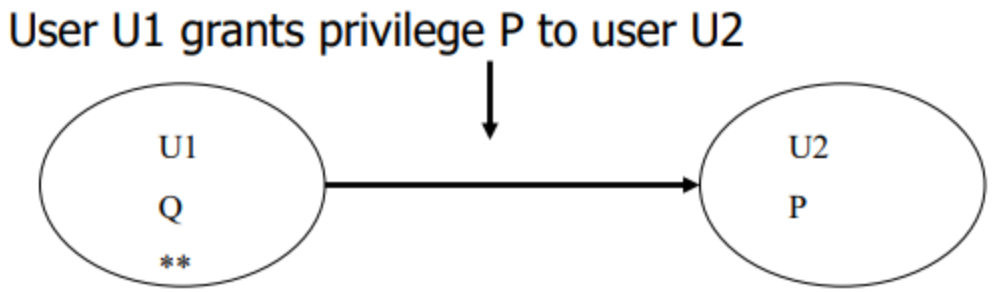
节点：用户与权限

弧：授权关系

*** ： WITH GRANT OPTION**

**** ：权限来自ownership**

example



9.1.7. 撤销授权

语法

REVOKE privileges ON relation FROM users [CASCADE | RESTRICT]

仅回收传递授权：

REVOKE GRANT OPTION FOR privilege ON relation FROM users [CASCADE | RESTRICT]

CASCADE：递归撤回

RESTRICT: 如果权限被传递，则执行失败

9.2. Recursion

9.2.1. 语法

WITH

[RECURSIVE] R1 AS <Definition of R1>

[RECURSIVE] R2 AS <Definition of R2>

9.2.2. 非线性

递归表的规则中自己不能出现超过一次

递归不能互相包含

递归带有聚合

9.3. Object-relational model

9.3.1. 特点

支持多种数据类型

支持高级查询

允许处理复杂类型

保证关系数据模型基础

向前兼容：在原有语法上建立

9.3.2. User Defined Types

用途

作为关系的类型

作为属性的类型

语法

CREATE TYPE <typename> AS (
<list of attribute-type pairs>);

方法

方法声明

```
CREATE TYPE BarType AS (
  name    CHAR(20),
  addr    CHAR(20))
   METHOD Telnumber() returns CHAR(10);
```

方法定义

```
CREATE METHOD Telnumber() returns
  CHAR(10)
   FOR BarType
   Begin … End; // method body
```

REF

引用对象（指针）

语法

```
CREATE TYPE MenuType AS (
bar REF BarType,
beer REF BeerType,
price FLOAT);
```

A REF(T) SCOPE R

限制引用对象出现的表（否则可以引用任意表）

UDT作为行类型

语法

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

元素可能为约束、索引

这样的关系是一个一元关系，每一行只有一个对象

生成对象标识

类型

SYSTEM GENERATED

系统在指定行生成一个唯一标识

DERIVED

使用主键

语法

REF IS<属性名><how generated>

获取数据

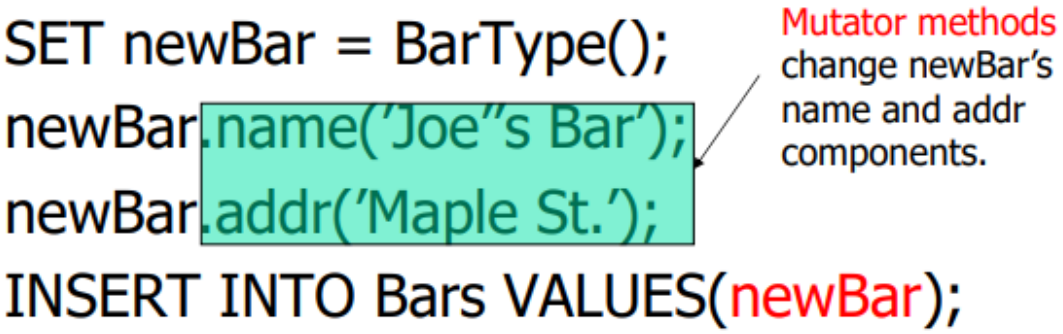
generator

A()获取数据

mutator

A(v) 设置数据

插入数据



获取REF数据

使用 -> 获取引用的属性

使用**DEREF**解引用

排序

定义两种方法

EQUAL

LESSTHAN

语法

比较特定属性

CREATE ORDERING FOR T EQUALS ONLY BY STATE;

使用函数比较

CREATE ORDERING FOR T ORDERING FULL BY RELATIVE WITH F

9.4. Online analytic processing

9.4.1. On-Line Transaction Processing (OLTP)

简单而频繁的数据库操作

9.4.2. On-Line Application Processing（OLAP）

少但是复杂的操作，不一定要最新

9.4.3. The Data Warehouse

将数据复制进单独的数据库

定期重建、更新不频繁

执行复杂查询

9.4.4. Star Schemas

组成

Fact table

一张非常大的表

Dimension attributes：引用某个维表的外键

Dependent attributes：由维表属性决定的值

Dimension tables

9.4.5. 建立数据仓库

ROLAP：使关系数据库支持星形结构

MOLAP：使用特殊**DBMS**（**data cube**）

9.4.6. ROLAP

Bitmap indexes

在事实表上建立，方便快速索引

Materialized views

保存视图

9.4.7. Data Cubes

组成

维表的键是立方体的维度

依赖属性是立方体中的点

Drill-Down

拆分某一个聚合

Roll-Up

聚合某一个分类

切片

指定某一个维度的值

切块

指定某一系列维度的值

SQL操作

语法

Select dimension-attributes,aggregations
From tables
Where conditions
Group by cube(dimension-attributes)

```
Select dimension-attributes,aggregations
From tables
Where conditions
Group by rollup (dimension-attributes)
```

ROLLUP

维度属性从右往左依次上卷

9.4.8. Data Warehouse vs. Big Data

对立：数据仓库需要演变成新的形式来解决大数据问题

和而不同：数据仓库是架构而大数据是解决方案

支撑：数据仓库是大数据的支撑

10. NOSQL

10.1. 大数据

10.1.1. 发展

超大规模数据

海量数据

大数据

10.1.2. 定义

无法在课容忍时间里使用现有技术对其感知、获取、管理、处理、服务的数据集合

PB以上数量级

包括结构化、半结构化、非结构化数据

10.1.3. 特点

巨量

多样

快变

价值

Volume（巨量存储）, Variety（多样性）, Velocity（快速插入）, Variability（可变性、易变性）,Veracity（真实性）

10.1.4. 大数据查询

系统需要非常大规模(牺牲ACID)

需要支持非关系数据

10.2. 大数据模型

10.2.1. Nosql 的解释

非关系

Not only SQL

10.2.2. 回顾：关系模型

特点

保证ACID

事务控制

多个应用共享数据库

有标准模型

实际应用

阻碍了新应用开发->应用数据库

大数据爆发->集群

需求：分布式、可拓展性、混合存储

10.2.3. 扩展

纵向拓展

横向拓展

增加机器数量

10.2.4. 聚合模型

将相关的对象当做一个单位处理

10.3. NOSQL系统

10.3.1. 定义

牺牲事务与强一致性，获取更好的分布式能力与横向拓展能力

10.3.2. 应用场景

数据模型简单

灵活性需求

性能要求高

一致性要求低

给定key可能映射复杂值环境

10.3.3. NOSQL特点

不使用关系模型

在集群上表现良好

没有模式

灵活数据模型

多语言持久

使用多种方式存储数据

自动分片

自动复制

分布式处理

10.3.4. 分布式数据库

定义

一组数据组成，分布于不同节点，逻辑属于同一系统

每个站点有独立处理能力，可以执行本地应用与全局应用

故障模式

站点故障

消息丢失

通信链路故障

网络分隔

结构

分片模式

水平分片

垂直分片

拆分不同列到不同节点

混合分片

原则

完备性

可重构性

不相交性

分布策略

目的：在系统故障时保证可用

要求：数据复制多份、故障时自动路由

问题：副本一致性

并发访问控制

Primary copy scheme

Majority protocol

Quorum consensus

读写操作时选择指定数量的节点

根据开销单独选择读写操作副本数

要求

行为和单副本模式一样

Auto-sharding

数据库复杂分配、访问数据到正确分片

Replication

Master-Slave replication

中心节点负责保存副本

Peer-to-peer replication

10.3.5. NOSQL四种数据库

键值数据库

列簇数据库

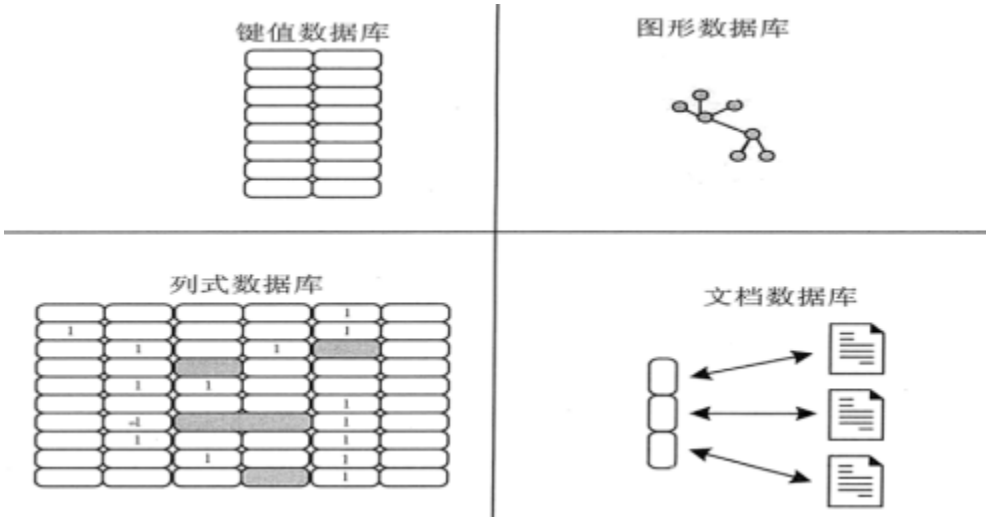
垂直分片

文档数据库

键值数据库升级

允许嵌套、索引

图形数据库



10.3.6. 弱一致性

CAP

组成

可用性

一致性

分区

分区容错性

分布式系统只能满足两个特性

BASE

组成

基本可用：保证核心功能可用

允许副本暂时不一致

副本最终一致

弱化一致性提高可拓展性、可靠性、可用性

Quorums

一致性与可用性间权衡

强一致性： **W+R > N**

无需联系所有节点，只需要有足够多副本数

10.3.7. NOSQL优缺点

优点

高拓展性

分布计算

低成本

半结构化数据、邻国

没有复杂关系

缺点

没有标准

受限的查询能力

一致性不够好

10.3.8. NOSQL系统实现

10.4. NEWSQL模型

10.4.1. 使用SQL语言

10.4.2. 遵循ACID

10.4.3. 使用无锁并发

11. 半结构化数据（XML）

11.1. XML = Extensible Markup Language

11.2. 关键思路：使用tag标记数据

11.3. 动机：数据交换

11.4. 与关系模型的对比

11.4.1.

	Relational	XML
Structure	Table	Tree, graph Non rigid format
Schema	fixed	Flexible Tags self describing Allows nested structures
Queries	Simple, high level language	complex
Ordering	none	implied

11.5. 分类

11.5.1. Well-Formed

在数据中引入自己的tag

11.5.2. Valid

需要DTD或XML schema

11.6. Well-Formed XML

11.6.1. 基本结构要求

单个根节点

闭合的tag

适当的嵌套

元素内属性唯一

11.6.2. 语法

<?xml version = ”1.0” standalone = ”yes” ?>

matched pairs：<FOO> …</FOO>

Unmatched tags：<FOO/>

11.7. Valid XML

11.7.1. 结构

需要标准定义：DTD or XML schema

11.7.2. DTD

语法

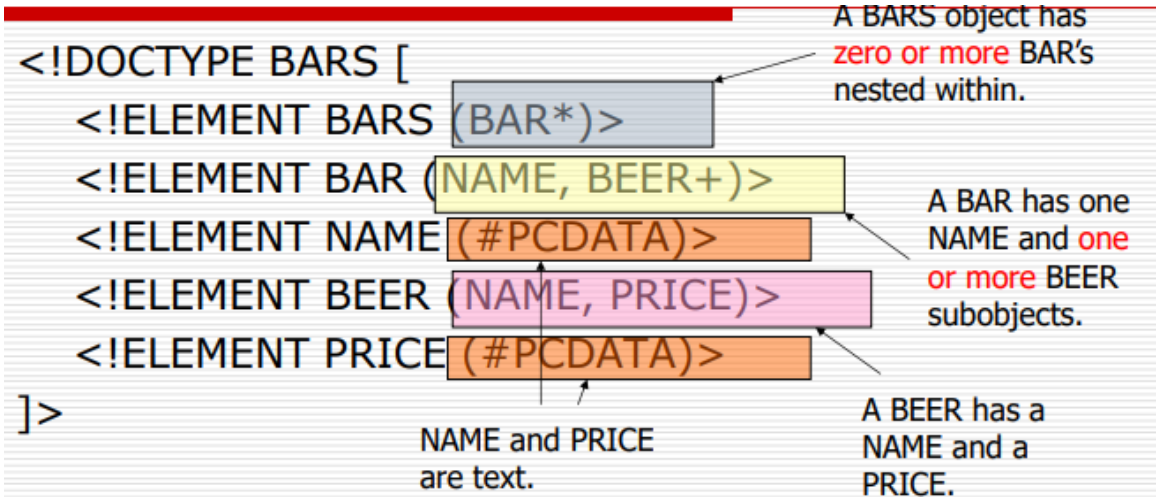
<!DOCTYPE <root tag> [

<!ELEMENT <name>(<components>)>

. . . more elements . . .

]>

example



元素描述

括号内Subtags必须按序出现

数量标记：*+?

选择标记：|

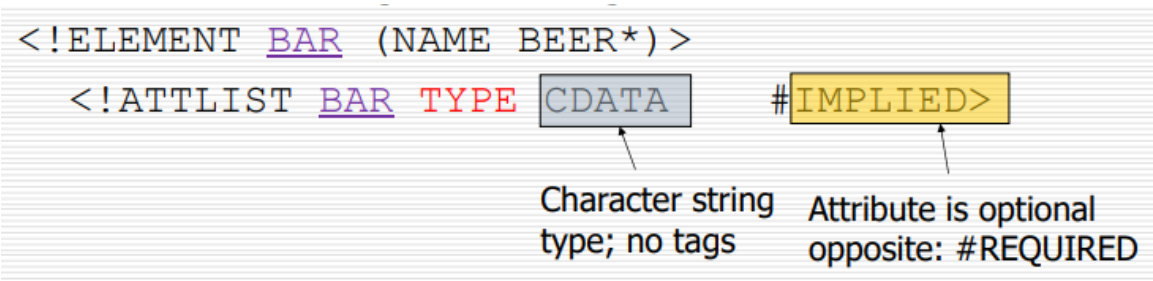
属性定义

<!ATTLIST element-name, attribute-name, type>

Type： CDATA/ID/IDREF/IDREFS/EMPTY

#REQUIRED/#IMPLIED/default value

example

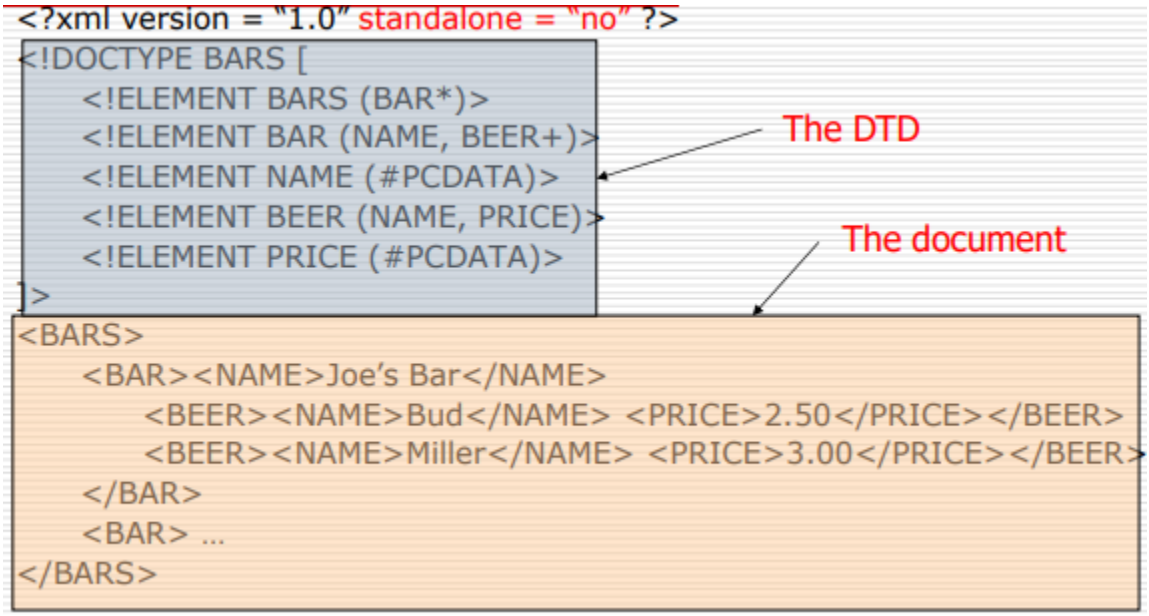


ID/IDREF(S)

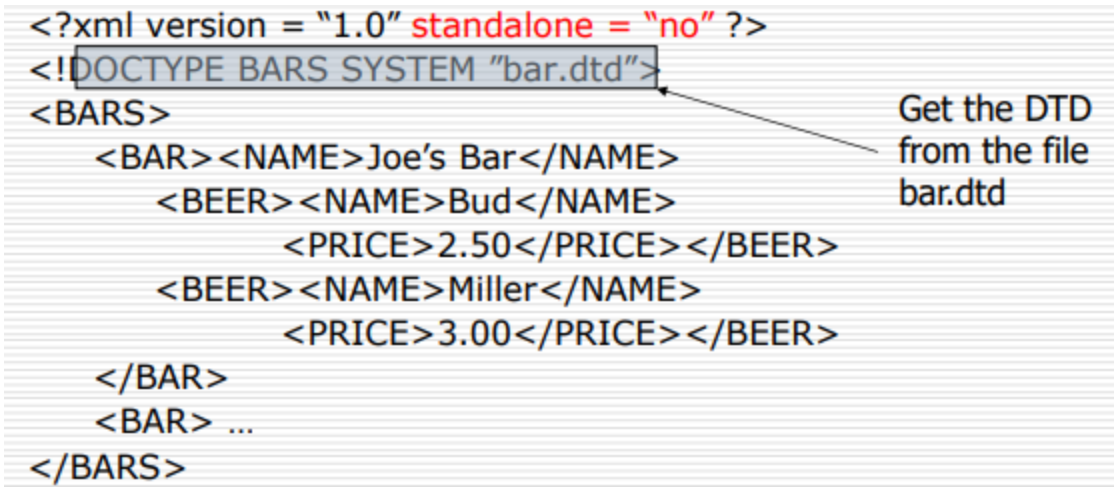
使用**DTD**

standalone = “no”

作为前言



引用外部数据



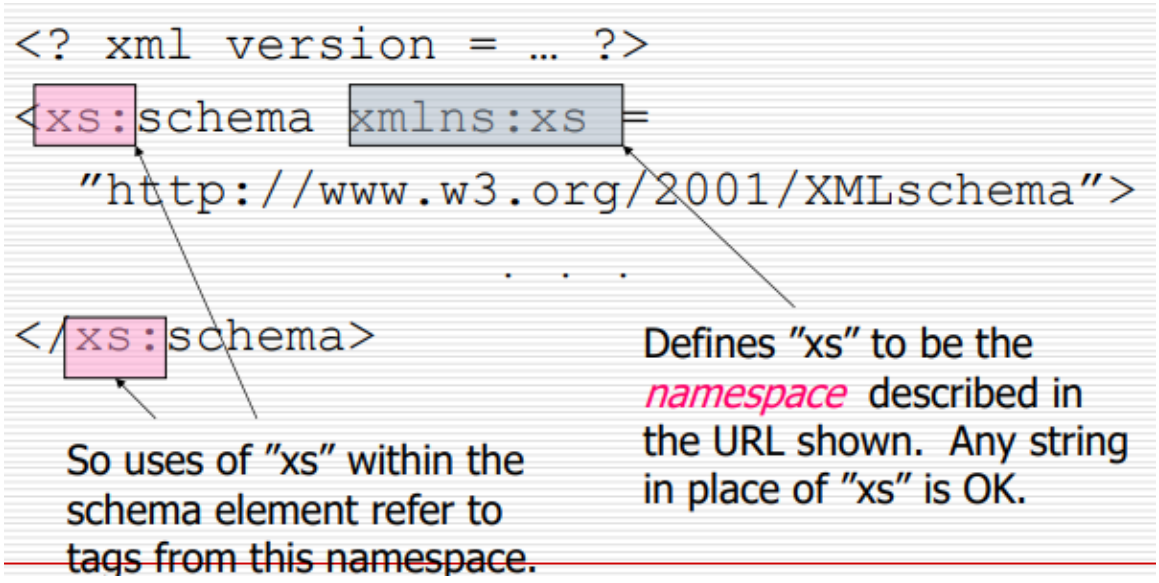
限制

文本元素没有类型

不能表示无序集合

11.7.3. XML Schema

主结构



xs:element

属性

name

type

Simple Types

用于描述枚举或范围受限的基类

xs:restriction

base：基类

xs:{min, max}{Inclusive, Exclusive}

xs:enumeration

value

example

```
<xs:simpleType name = "price">
  <xs:restriction
    base = "xs:float"
    minInclusive = "1.00"
    maxExclusive = "5.00" />
</xs:simpleType>
```

Complex Types

用于描述由子元素组成的元素

xs:sequence

子元素为**xs:element**

子元素属性**minOccurs**、**maxOccurs**标记出现次数

xs:all

xs:choice

xs:attribute

定义复杂元素的属性

属性

name

type

use

Keys

使用**xs:key**作为**xs:element**子元素

xs:selector选择路径

xs:field指定唯一的**filed**

Foreign Key

使用**xs:keyref**作为子元素

name属性：外键名

refers属性：引用键名

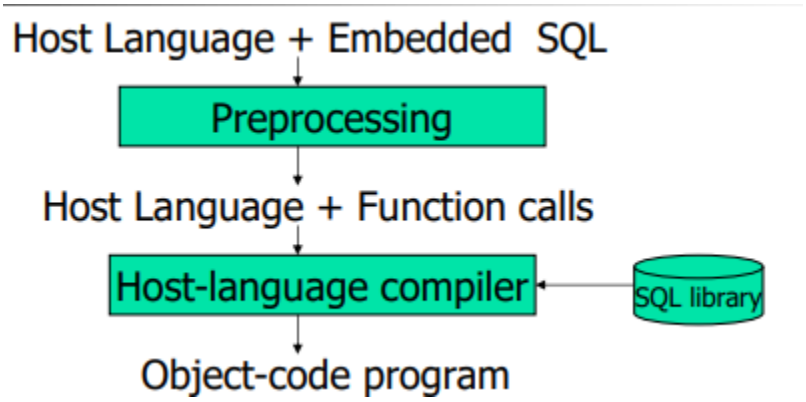
12. SQL environment（Embedded SQL）

12.1. 定义

12.1.1. 将SQL加入传统编程语言

12.2. 实现

12.2.1.



12.3. SQL识别

12.3.1. EXEC SQL标识

12.3.2. 共享变量：使用EXEC SQL BEGIN / END DECLARE SECTION声明

12.4. 游标

12.4.1. 声明

EXEC SQL DECLARE <cursor>

CURSOR FOR <query>

12.4.2. 使用

EXEC SQL OPEN<cursor>

EXEC SQL FETCH FROM < cursor > INTO <list of variables>

EXEC SQL DELETE FROM <table> WHERE CURRENT OF <cursor>

12.4.3. 关闭

EXEC SQL CLOSE <cursor>

12.4.4. 并发保护

EXEC SQL DECLARE c INSENSITIVE CURSOR FOR
SELECT beer,price FROM Sells

12.4.5. 移动

EXEC SQL DECLARE C SCROLL CURSOR FOR Sells

在**FETCH**后加上方向参数

12.4.6. 动态

准备：**EXEC SQL PREPARE <query-name> FROM <text of the query>;**

执行：**EXEC SQL EXECUTE <query-name>**

立即执行：**EXEC SQL EXECUTE IMMEDIATE <text>**