CS3153/5153 DatabasesSummer 2007
SQL

Orlando Karam okaram@spsu.edu

Summer 2007, September 26, 2007

Introduction

- SQL - Structured Query Language
- Standard language for relational databases
- Every DBMS implements with slight variations
- Main variations:
    - Special data types (auto-increment)
    - Stored procedures
    - Advanced features
    - Kinds of index, file organization

Advantages of SQL Standard

- Reduced training
- Productivity
- App portability, longevity
- Reduced dependency on single vendor
- Cross-System communication

SQL Environment

- Catalog
- Schema
- Data Definition Language (DDL)
  - CREATE TABLE
  - CREATE VIEW
  - CREATE INDEX
  - DROP TABLE/VIEW/INDEX
  - ALTER ...
- Data Manipulation Language (DML)
  - INSERT
  - UPDATE
  - DELETE
  - SELECT
- Data Control Language (DCL): Users, permissions etc

CREATE TABLE Statement

```
CREATE TABLE table_name (
    field1 type constraints,
    field2 type2 constraints,
    CONSTRAINT name ... ,
    more constraints
);
```

```
CREATE TABLE Book (
    ISBN  CHAR(9)     PRIMARY KEY,
    Title VARCHAR(20) UNIQUE NOT NULL,
    Pages Integer
);
```

Common Datatypes

Common Datatypes

- CHAR(n)
- VARCHAR(n)
- NUMERIC(prec,scale) NUMERIC(3,2)= 9.99
- DATE, TIMESTAMP
- Much variation among DBMSs
- BLOB, Float, Text/Long/Memo, Int, Boolean, Serial

Constraints

Constraints

- Syntax/Semantics
  - Column constraints
  - Per table
  - Named or unnamed
  - CREATE CONSTRAINT not widely supported
- Kinds
  - NOT NULL
  - PRIMARY KEY
  - UNIQUE
  - FOREIGN KEY / REFERENCES
  - CHECK
  - DEFAULT (not really a constraint :)

Column Constraints

Column Constraints

- Go at end of each column
- Order is not important
- Can only affect that column
- Not named

```
DROP TABLE Person;
CREATE TABLE Person (
    Id      INTEGER     PRIMARY KEY,
    SSN     CHAR(9)     UNIQUE,
    Name    VARCHAR(20) NOT NULL,
    Age     INTEGER     DEFAULT 18
                          CHECK (Age BETWEEN 10 AND 100),
    Major   CHAR(3) REFERENCES Major(Id)
);
```

Table Constraints

Table Constraints

- Can apply to more than one column
- Can be named (good practice)
- Usually go at end of table

```
CREATE TABLE Person (
    Id      INTEGER     ,
    SSN     CHAR(9)     NOT NULL,
    Name    VARCHAR(20) ,
    Age     INTEGER     DEFAULT 18,
    CONSTRAINT Person_PK PRIMARY KEY (Id),
    UNIQUE(SSN), -- unnamed
    CHECK (Age BETWEEN 10 AND 100)
);
```

Complicated Example I

```
CREATE TABLE Standing (
  deg_code      char(2) REFERENCES Degree(deg_code),
  min_cr        INTEGER DEFAULT 0 NOT NULL,
  max_cr        INTEGER NOT NULL,
  num           INTEGER NOT NULL,
  designation   VARCHAR(20) NOT NULL,
  CONSTRAINT Standing_PK
        PRIMARY KEY (deg_code, num),
  CONSTRAINT Standing_Unique_Designation
        UNIQUE (deg_code, designation),
  CONSTRAINT Standing_min_max
        CHECK (min_cr <= max_cr)
);
```

Complicated Example II

```
CREATE TABLE Enrolls (
    Student INTEGER REFERENCES Person(Id),
    cno     CHAR(7),
    Sec_No  Integer,
    CONSTRAINT Enrolls_PK
        PRIMARY KEY (Student,Cno,Sec_no),
    CONSTRAINT Enrolls_FK FOREIGN KEY
        (cno, Sec_No) REFERENCES Section(cno,Sec_no)
);
```

More on Foreign Keys

More on Foreign Keys

- May defer the check until the end of the transaction
- What to do when referenced table changes ? delete/update

ALTER TABLE

- Modifies the structure of a table
- Complicated, Avoid if you can

```
ALTER TABLE xx
actions
    DROP COLUMN yy
    ADD COLUMN xx
    RENAME COLUMN xx TO yy
etc
```

```
ALTER TABLE Person
DROP COLUMN Age,
ADD COLUMN A2 Integer
;
ALTER TABLE Person
RENAME COLUMN a2 TO Age
```

Views

- Basically act as Named Query
- Can also serve for permissions
- Remember? External Schemas
- Materialized Views
- Problem: Updates

```
CREATE VIEW view_name
AS
SELECT ...
```

```
CREATE VIEW Young_People
AS
SELECT *
FROM Person
WHERE Age<99
```

Indexes

- Mainly for performance
- Unique indexes also enforce constraints
- Different DBMSs support different kinds of indexes

```
CREATE [type] INDEX index_name
ON table(fields)
```

```
CREATE UNIQUE INDEX pname_idx
ON Person(name)
```

INSERT INTO

- If no fields named, values correspond to ALL fields, in the 'right' order
- If no value provided (field not named, or ,,) then default value or NULL if no default

```
INSERT INTO table ( field1, field2,...)
    VALUES (1,'John')
```

```
INSERT INTO Person (Id, Name) VALUES(1, 'Orlando')

```

DELETE FROM

- If no condition, then deletes all rows
- Notice table still there, but no data in it

```
DELETE FROM table

DELETE FROM table
WHERE conditions
```

```
DELETE FROM Person
WHERE age>35
```

UPDATE xx SET ...

- Updates done AFTER evaluating conditions

```
UPDATE table
SET f1=v1, f2=v2 ...
WHERE conditions
```

```
UPDATE Person
SET age=age+1
WHERE age>10
```

Simple Select

- Can use expressions rather than fields
- Can use AS to rename fields
- Can rename tables
- Notice original table is NOT modified in any way

```
SELECT fields
FROM table
WHERE conditions
```

```
SELECT *
FROM Person
WHERE Name like ’%X%
```

More operators

- LIKE
- BETWEEN
- IN

Example DB: Student, Program, School

```
CREATE TABLE Student (
    Id        INTEGER PRIMARY KEY,
    Name    VarChar(20),
    Age    Integer,
    Gender CHAR(1) CHECK(Gender IN (’M’,’F’))
);
```

```
CREATE TABLE School (
    Id CHAR(3) PRIMARY KEY,
    Name VarChar(20) UNIQUE
);
```

```
CREATE TABLE Program (
    Id     CHAR(2)         PRIMARY KEY,
    Name    VarChar(20) UNIQUE,
    School CHAR(3) References School(ID)
);
```

Example DB: Majors, Minors

```
CREATE TABLE Majors (
    Student Integer REFERENCES Student(Id),
    Program CHAR(2) REFERENCES Program(Id),
    CONSTRAINT Majors_PK
        PRIMARY KEY (Student,Program)
);
```

```
CREATE TABLE Minors (
    Student Integer REFERENCES Student(Id),
    Program CHAR(2) REFERENCES Program(Id),
    CONSTRAINT Minors_PK
        PRIMARY KEY (Student,Program)
);
```

Basics

Basics

- Aggregates allow you to combine the values from a set of rows
- Useful for counting and summarizing
- Standard aggregates:
  - COUNT
  - MAX
  - MIN
  - AVG
- If you include a field(or expression), only uses the rows for which that field is not null
- Count uses * for counting all rows
- Can use DISTINCT, useful mostly for count

Simple aggregation

Simple aggregation

- Apply to the whole table
- Always return one row (but still a table, not a scalar)
- If you include a field(or expression), only uses the rows for which that field is not null

Number of Students

```
SELECT COUNT(*) as num_students
FROM Student;
```

Average age of students

```
SELECT AVG(Age) as "Average Age"
FROM Student S;
```

Number of different values of Age

```
SELECT COUNT(DISTINCT Age) as num_students
FROM Student;
```

Aggregates and Grouping

Aggregates and Grouping

- Get one value per group
- Groups specified in GROUP BY
- Any non-aggregate mentioned needs to be mentioned in GROUP BY

```
SELECT Gender, COUNT(*) as num_students
FROM Student
GROUP BY Gender;
```

```
SELECT Gender, AVG(Age) as "Average Age"
FROM Student S
GROUP BY Gender;
```

HAVING

HAVING clause

- Allows us to add conditions after the grouping
- You need to repeat the expression, even if used as a field and renamed
- WHERE still works as usual (but is applied before the grouping

```
SELECT Age, COUNT(*) as num_students
FROM Student S
WHERE Gender = 'F'
GROUP BY Age
HAVING COUNT(*)>=2;
```

Basics

Basics

- Allow you to combine information from two (or more) different tables
- the join clause syntactically acts as anothe table
- table1 JOIN table2 ON (condition)
- table1 NATURAL JOIN table2
  condition is equality of all fields with same name
- table1 JOIN table2 USING (fields)
  condition is equality of mentioned fields
- OUTER JOIN makes sure ALL tuples from one (or both) tables are included in the result

Simple Join Examples

Examples

```
SELECT *
FROM Majors M JOIN Program P ON(M.Program=P.Id)
```

```
SELECT P.Name, COUNT(*)
FROM Majors M JOIN Program P ON(M.Program=P.Id)
GROUP BY P.Id, P.Name
```

Examples - ON vs NATURAL vs USING

Examples - ON vs NATURAL vs USING

```
SELECT *
FROM Majors Ma JOIN Minors Mi ON (
    Ma.Program=Mi.Program AND
    Ma.Student=Mi.Student
)
```

```
SELECT *
FROM Majors Ma NATURAL JOIN Minors Mi
```

```
SELECT *
FROM Majors Ma JOIN Minors Mi
```

```
SELECT *
FROM Majors Ma JOIN Minors Mi USING(Program)
```

OUTER JOIN

OUTER JOIN

```
SELECT P.Name, COUNT(*)
FROM Majors M JOIN Program P ON(M.Program=P.Id)
GROUP BY P.Id, P.Name
```

```
SELECT P.Name, COUNT(*)
FROM Majors M LEFT OUTER JOIN
    Program P ON(M.Program=P.Id)
GROUP BY P.Id, P.Name
```

Joining several tables

Joining several tables

```
SELECT DISTINCT St.Name, Sc.Name
FROM Student St JOIN
    Majors M ON (St.Id=M.Student) JOIN
    Program P ON (P.Id=M.Program) JOIN
    School Sc ON (Sc.Id=P.School)
```

```
SELECT Sc.Id, Sc.Name, COUNT(M.Student)
FROM Majors M JOIN
    Program P ON (P.Id=M.Program)
        RIGHT OUTER JOIN
    School Sc ON (Sc.Id=P.School)
GROUP BY Sc.Id, Sc.Name
```

Basics

Subqueries- Basics

- sub-query – another SELECT statement within a SELECT statement
- Can use a sub-query instead of a table in FROM clause
- Can use with IN, EXISTS, >ANY, >ALL
- Nesting to any level of complexity

Example: Subquery in FROM

Example: Subquery in FROM

```
SELECT P.Name, COUNT(*)
FROM (SELECT Program as Id FROM Majors) M
    NATURAL JOIN Program P
GROUP BY P.Id, P.Name
```

Examples: IN

Examples: IN

Students majoring in CS

```sql
SELECT *
FROM Student
WHERE Id IN (
    SELECT Student
    FROM Majors
    WHERE Program='CS'
)
```

Students NOT majoring in CS

```sql
SELECT *
FROM Student
WHERE Id NOT IN (
    SELECT Student
    FROM Majors
    WHERE Program='CS'
)
```

Example: EXISTS

Example: EXISTS

Students majoring in CS

```
SELECT *
FROM Student
WHERE EXISTS (
    SELECT Student
    FROM Majors
    WHERE Program=’CS’ AND Student=Student.Id
)
```

Students NOT majoring in CS

```
SELECT *
FROM Student
WHERE NOT EXISTS (
    SELECT Student
    FROM Majors
    WHERE Program=’CS’ AND Student=Student.Id
)
```

Example: >=ALL

Example:

Oldest Student(s)

```
SELECT *
FROM Student
WHERE Age >=ALL (
    SELECT Age
    FROM Student
)
```