Data Definition Language

Material covered from Chapter 2, A First Course in Database Systems by Jenifer and Garcia &

SQL is a…

- Data Manipulation Language (DML)
  - Query one or more tables
  - Insert/delete/modify tuples in tables
- Data Definition Language (DDL)
  - Define relational schemata
  - Create/alter/delete tables and their attributes

Relations in SQL

- Stored relations (Defined by user and exists in database)
- Views (Defined by computation, not stored usually)
- Temporary relations

SQL CREATE TABLE

- Attributes
- Constraints
- Indexes (data structures that speed up many operations)

Data Types in SQL

Atomic types:

- Characters: CHAR(20), VARCHAR(50)
- Numbers: INT, BIGINT, SMALLINT, FLOAT
- Others: MONEY, DATETIME…

Every attribute must have an atomic type

Declaring Schema

Students(sid: *string,* name: *string,* gpa: *float*)

```
CREATE TABLE Students (
  sid CHAR(20),
  name  VARCHAR(50),
  gpa  float,
  PRIMARY KEY (sid),
)
```

Declaring Schema

Students(sid: *string,* name: *string,* gpa: *float*)

```
CREATE TABLE Students (
  sid CHAR(20) PRIMARY KEY,
  name  VARCHAR(50),
  gpa  float
)
```

NULL and NOT NULL

- To say “don’t know the value” we use NULL
  NULL has (sometimes painful) semantics, more detail later

Students(sid:string, name:string, gpa: float)

sid	name	gpa
123	Bob	3.9
143	Jim	NULL

Say, Jim just enrolled in his first class.

In SQL, we may constrain a column to be NOT NULL,
e.g., “name” in this table

Deleting Schema

```
DROP TABLE Students;
```

ALTER Schema

ALTER TABLE Students ADD phone CHAR(16);

ALTER TABLE Students DROP phone;

How to change datatype of a column?

DEFAULT Schema

What happens if you add/delete/modify an attribute after certain time period?

DEFAULT Schema

```
ALTER TABLE Students ADD birthdate DATE DEFAULT DATE ‘`0000-00-00’
ALTER TABLE Students ADD gender CHAR(1) DEFAULT ‘?’;


CREATE TABLE Students (
  sid CHAR(20),
  name  VARCHAR(50),
  gpa  float,
  gender CHAR(1) DEFAULT ‘?’,
  PRIMARY KEY (sid),
)
```

Constraints and Relational Algebra

General Constraints

- We can actually specify arbitrary assertions

  E.g. *"There cannot be 25 people in the DB class"*

- In practice, we don't specify many such constraints. Why?

Whenever we do something ugly (or avoid doing something convenient) it's for the sake of performance

KEYS

- PRIMARY KEY OR UNIQUE KEY – Difference

KEYS

- REFERENTIAL INTEGRITY

Relational algebra as a Constraint Language

- The value of R must be empty or there are no tuples in the result of R

- Every tuple in the result of R must also be in the result of S

Example – Referential constraint

Key constraints

- PK: no two tuples must have same values in key attributes

Additional constraints

- One must have net worth of at least $10,000,000 to be the president of a movie studio.
- MoveExec(name, address, cert#, netWorth)
- Studio(name, address, presC#)

$$\sigma_{netWorth<10000000}(\texttt{Studio} \bowtie_{presC\#=cert\#} \texttt{MovieExec}) = \emptyset$$

$$\pi_{presC\#}(\texttt{Studio}) \subseteq \pi_{cert\#}\left(\sigma_{netWorth\geq 10000000}(\texttt{MovieExec})\right)$$

Summary of Schema Information

- Schema and Constraints are how databases understand the semantics (meaning) of data

- SQL supports general constraints:
  - Keys and foreign keys are most important
  - We'll give you a chance to write the others

Foreign Key Constraint

Definition

- A constraint is a relationship among data elements that the DBMS is required to enforce.
  - Example: key constraints.

Types of Constraints

- Keys.
- Foreign-key, or referential-integrity.
- Value-based constraints.
  - Constraint values of an attribute.
- Tuple-based constraints.
  - Relationship among components.
- Assertions: any SQL Boolean expression.

Foreign Key Constraint

- Product (**id**, name, price, **categoryId**)
- Category(**id**, name, description)

Exp: A constraint that requires **categoryId** in Product table to be a valid **id** in Category table is called a foreign – key constraint.

Category:

Id	Name	Description
1	Hardware	Electronics,..
3	Bakery	Baked items, Eggs,..
5	Books	Novel, Fiction

Product:

Id	Name	Price	CategoryId
A1	Bread	25	3
B4	Laptop	40000	1
C5	Peer-e-Kamil	1500	5

Definition

*In the context of relational databases, a **foreign key** is a set of attributes subject to a certain kind of inclusion dependency constraint, specifically a constraint that the tuples consisting of the foreign key attributes in one relation, R, must also exist in some other (not necessarily distinct) relation, S, and furthermore that those attributes must also be a candidate key in S.*

Or

*A **foreign key** is a set of attributes that references a **candidate key**.*

Example:

Child table (Product) and Parent table (Category, PK and CK: Id)

Definition taken from https://en.wikipedia.org/wiki/Foreign_key

Foreign Key – SQL Syntax

- Use the keyword **REFERENCES**, either:
  - Within the declaration of an attribute (only for one-attribute keys).
  - As an element of the schema:

FOREIGN KEY (<list of attributes>)

REFERENCES <relation> (<attributes>)

- Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Foreign Key Syntax – (1) With Attribute

- Product (**id**, name, price, **categoryId**)
- Category(**id**, name, description)

```
CREATE TABLE Product (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20),
        price REAL,
        categoryId CHAR (20) REFERENCES
                                Category(id)
);
```

```
CREATE TABLE Category (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20),
        description VARCHAR(255)
);
```

Foreign Key Syntax – (1) As Element

- Product (**id**, name, price, **categoryId**)
- Category(**id**, name, description)

```
CREATE TABLE Product (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20),
        price REAL,
        categoryId CHAR (20),
        FOREIGN KEY (categoryId) REFERENCES
                                        Category(id)
);
```

Points to Ponder

- Can attribute(s) (upon which foreign key is added) involved in first relation (where foreign key is being added as constraint)/Child table, have NULL values?
- Can attribute(s) (which are linked as Foreign key) involved in referenced relation i.e. second relation (Parent table) have NULL values?
- What is the impact of having NULL in the value of the FK attribute?

Referential Integrity – Maintenance
What will happen?

- **First relation (R):**
- Insert a new tuple in first relation with non-NULL FK value and this value does not exist in referenced relation. REJECTED
- Update a tuple to change FK attribute in first relation to a non-NULL value and this value does not exist in referenced relation
- **Referenced relation (S):**
- Delete a tuple from referenced relation and it's FK attribute(s) appears as the component of the first relation
- Update a tuple in referenced relation, in such a way that it's FK attribute is changed and old value of the FK's attribute(s) is the value of the first relation.

  = > Dangling tuples in R

Referential Integrity – Maintenance Solution?

- **First relation (R)**

REJECTED

- **Referenced relation (S)**

DEFAULT - REJECT

CASCADE: (1) DELETE
(2) UPDATE

Set NULL: NULL

Product:

Id	Name	Price	CategoryId
A1	Bread	25	3
B4	Laptop	40000	1
C5	Peer-e-Kamil	1500	5

Category:

Id	Name	Description
1	Hardware	Electronics,..
3	Bakery	Baked items, Eggs,..
5	Books	Novel, Fiction

Referential Integrity – Maintenance Policy implementation - Syntax

- Product (**id**, name, price, **categoryId**)
- Category(**id**, name, description)

```
CREATE TABLE Product (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20),
        price REAL,
        categoryId CHAR (20) REFERENCES Category(id)
                ON DELETE SET NULL
                ON UPDATE CASCADE
);
```

Attribute & Tuple-level Constraints

Types of Constraints

- Keys.
- Foreign-key, or referential-integrity.
- Value-based constraints.
  - Constraint values of an attribute.
- Tuple-based constraints.
  - Relationship among components.
- Assertions: any SQL Boolean expression.

Attribute Level Constraint – (1)

- Attribute is NOT NULL

```
CREATE TABLE Product (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20) NOT NULL,
        price REAL,
        categoryId CHAR (20) REFERENCES
                                Category(id)
);
```

```
CREATE TABLE Category (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20),
        description VARCHAR(255)
);
```

Attribute Level Constraint – (2)

- Attribute is NOT NULL

```
CREATE TABLE Product (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20) NOT NULL,
        price REAL,
        categoryId CHAR (20) REFERENCES Category(id) NOT NULL
);
```

IMPLICATIONS?

Attribute Level Constraint – (2)

```
CREATE TABLE Product (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20) NOT NULL,
        price REAL,
        categoryId CHAR (20) REFERENCES Category(id) NOT NULL
);
```

IMPLICATIONS?

- Without categoryId, no product inserts allowed.
- We cannot use SET-NULL POLICY i.e. in case of deletion of tuple in Category table, if we had used policy ON DELETE SET NULL (in Product table –> attribute - categoryId) then with NOT NULL (Attribute level constraint), we cannot set category id in Product table to be NULL

Attribute based **CHECK** Constraint

- An attribute based CHECK constraint is checked whenever any tuple gets a new value for this attribute
  - Insert
  - Update (Checked on new value)
- Violation: Rejects

```
CREATE TABLE Student (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20),
        gpa REAL CHECK(gpa >= 2.0 AND gpa <= 4.0),
        gender CHAR(1) CHECK(gender IN ('F', 'M'))
);
```

```
INSERT INTO Student
VALUES ('1', 'Ali', 1.0, 'M')
```

```
INSERT INTO Student
VALUES ('2', 'Akram', 3.0, 'G')
```

The INSERT statement conflicted with the CHECK constraint

Points to Ponder

- Can referential integrity constraint be implemented by the attribute-based CHECK constraint?

```
CREATE TABLE Product (
        id CHAR(20) PRIMARY KEY,
        name CHAR(20) NOT NULL,
        price REAL,
        categoryId CHAR (20) CHECK
                (categoryId IN (SELECT id FROM Category))
);
```

Issues?

* Inserts with NULL categoryId may/maynot be allowed

* Deleting a tuple in Category will make respective categoryId in Product invalid

Tuple-Based CHECK Constraint

- The condition is checked every time a tuple is inserted into relation R or updated.
- Evaluated for new/updated tuple
- Violation: Reject
- Attribute based CHECK and tuple based CHECK is invisible to other relations!

Be cautious:

- If condition mentions some other relation in a subquery, and a change to that relation causes the condition to become false for some tuple of R, the check does not inhibit/stop this change.

No subqueries in tuple-based CHECK => reliable

Tuple-based CHECK Constraint Example

```
CREATE TABLE SAMPLE (
        col1 varchar(2),
        col2 int,
        CHECK(col2 < case when col1 = 'A' then 50
                          when col1 = 'B' then 100
                `         when col1 = 'C' then 150
                          else col2 + 1
                     end));
```

Results:

1. Inserted successfully
2. Rejected
3. Rejected
4. Inserted successfully
5. Inserted successfully

```
INSERT INTO SAMPLE VALUES ('A', 49);
INSERT INTO SAMPLE VALUES ('A', 50);
INSERT INTO SAMPLE VALUES ('B', 100);
INSERT INTO SAMPLE VALUES ('C', 149);
INSERT INTO SAMPLE VALUES ('D', 5000);
```

Reference:

- **Example:** https://stackoverflow.com/questions/28679208/add-multiple-check-constraints-on-one-column-depending-on-the-values-of-another

Comparison of Tuple and Attribute-based Constraints

Preference:

- **Single attribute** – Attribute level or tuple level
- **More than one attribute** – Tuple level

Deferred Constraints

Foreign Key Constraint

- Product (**id**, name, price, **categoryId**)
- Category(**id**, name, description)

Exp: A constraint that requires **categoryId** in Product table to be a valid **id** in Category table is called a foreign – key constraint.

Category:

Id	Name	Description
1	Hardware	Electronics,..
3	Bakery	Baked items, Eggs,..
5	Books	Novel, Fiction

Product:

Id	Name	Price	CategoryId
A1	Bread	25	3
B4	Laptop	40000	1
C5	Peer-e-Kamil	1500	5

```
INSERT INTO Product
VALUES ('Face Powder', 600, 8)
```

Should it be inserted into Product's table?

DEFERRED Constraint

- Person (**id**, name, **marriedTopid**, gender, age)

marriedTopid is FK to Person table.

Person:

Id	Name	marriedTopid	Gender	Age
1	Amir	2	M	25
2	Sadia	1	F	21

```
INSERT INTO Person
VALUES (10,'Ali', ????, 'M', 28)
```

```
INSERT INTO Person
VALUES (5, 'Rehana', 10,  'F', 24)
```

- **References:**.
- Book: Database system concepts by Silberchatz

Available Options for Deferring Constraints

- DEFERRABLE
  - Constraint is checked immediately by default, but can be deferred when needed.
- **Available options after DEFERRABLE:**
  - INITIALLY DEFERRED
    - Constraint will be checked at the end of transaction (set of queries)
  - INITIALLY IMMEDIATE
    - Constraint checking will be made immediately after each statement

Syntax

- Person (**id**, name, **marriedTopid**, gender, age)

marriedTopid is FK to Person table.

SQL:

```
CREATE TABLE Person (
        id INT PRIMARY KEY,
        name VARCHAR(100),
        gender CHAR(1),
        age INT,
        marriedTopid INT UNIQUE
                REFERENCES Person (id)
                DEFERRABLE INITIALLY DEFERRED
);
```

Syntax (Update constraint)

- Person (**id**, name, **marriedTopid**, gender, age)

marriedTopid is FK to Person table.

```
SET CONSTRAINT MyNamedConstraint DEFERRED
```

```
SET CONSTRAINT MyNamedConstraint IMMEDIATE
```

Editing Constraints

Giving Names to Constraints

- Why naming constraints?
  - Add, delete or modify constraints

```
CREATE TABLE Student (
        id CHAR(20) CONSTRAINT IdIsKey PRIMARY KEY,
        name CHAR(20) NOT NULL,
        gpa REAL CONSTRAINT ValidGPA
                        CHECK(gpa >= 2.0 AND gpa <= 4.0),
        gender CHAR(1) CHECK(gender IN ('F', 'M', 'U')),
        weight REAL,
        CONSTRAINT RightFit
                CHECK (weight < case when gender = 'F' then 64
                                     when gender = 'M' then 70
                        end)
);
```

Altering Constraints on Tables

- We can set our named constraints to deferred or immediate using SET command.
- Other changes can be made using **ALTER TABLE** statement.

Note: One cannot add a constraint to a table unless it holds at that for every tuple in the table

ADDING and DROPPING Constraints

DROP:

- ALTER TABLE Student DROP CONSTRAINT IdIsKey
- ALTER TABLE Student DROP CONSTRAINT ValidGPA
- ALTER TABLE Student DROP CONSTRAINT RightFit

ADD:

- ALTER TABLE Student ADD CONSTRAINT IdIsKey PRIMARY KEY (id)
- ALTER TABLE Student ADD CONSTRAINT ValidGPA
  CHECK (gpa >= 2.0 AND gpa <= 4.0)

All of these constraints are now tuple-based!

Assertions in SQL

```cpp
#include <iostream>
#include "assert.h"

using namespace std;

int main()
{
    cout << "Program of adding 10 to the input" << endl;

    int a = 0;

    cout << "Please enter any number greater than or equal to 0 =" << endl;
    cin >> a;

    assert(a >= 0);

    int result = a + 10;
    cout << "Result = " << result;

    return 0;

}
```

```
Program of adding 10 to the input
Please enter any number greater than or equal to 0 =
-2
Assertion failed!

Program: G:\UET\PF\assertTest.exe
File: G:\UET\PF\assertTest.cpp, Line 15

Expression: a >= 0
```

Definition

- An assertion is a Boolean-valued SQL expression that must be true at all times.
- Easy to use
- Hard to implement efficiently
  - Will be checked on every modification
- Enforce any condition (expression that can follow **WHERE**)
- Any modifications in database which causes it to become false, is **rejected**. (We can also defer them)
  - All relations inclusive (not like attribute and tuple level constraints)

Syntax (ADD/DELETE)

```
CREATE ASSERTION <assertion-name> CHECK (<condition>)
```

```
DROP ASSERTION <assertion-name>
```

Example 1

- A student with GPA < 3.0 can only apply to campuses with rank > 4

Schema:

Student(ID, name, address, GPA)

Campus(location, numberofEnrolledStudents, rank)

Apply(ID, location, date, major, decision)

Constraints:

1. Apply.ID and Apply.location appear in Student.ID and Campus.location respectively

2. Campus.rank <=10

```
CREATE ASSERTION RestrictApps CHECK
        (NOT EXISTS
                (SELECT *
                 FROM Student, Apply, Campus
                 WHERE Student.ID = Apply.ID
                 AND Apply.location = Campus.location
                 AND Student.GPA < 3.0 AND Campus.rank <=4)
        );
```

Example: http://infolab.stanford.edu/~ullman/fcdb/jw-notes06/constraints.html

Example 2

- Total length of all movies by a given studio shall not exceed 10,000 minutes

Schema:

Movies(title, year, genre, studioName, producer#)

```
CREATE ASSERTION SumLength CHECK
        (10000 >= ALL
                (SELECT SUM(length)
                 FROM Movies
                 GROUP BY studioName)
        );
```

Can same be implemented using tuple-based CHECK constraint?

```
CREATE TABLE Movies(
        title VARCHAR(20),
        ….
        CHECK (10000 >= ALL
                (SELECT SUM(length)
                FROM Movies
                GROUP BY studioName)
                );
);
```

Still valid?

Example taken from Jennifer Ullman book "A first course in database systems"

Difference between Assertion and Attribute and Tuple level constraints

- Tuple level constraints can use name of attributes of same relation in the CHECK. Attribute level cannot use other attributes. It can only use the attribute, where it applied.
- Assertions cannot use any attribute referred in the condition. It must be first introduced in select-fr0m-where expression by mentioning their relation.

Difference between Attribute-based CHECK, Tuple-based CHECK Constraints and Assertions

Type of Constraint	Where Declared	When Activated	Guaranteed 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 any change to any mentioned relation	Yes

Table taken from the book A First Course in Database Systems by Jennifer Ullman

Triggers in SQL

Definition

- A **trigger** is a series of actions that are associated with certain events, such as insertions into a particular relation, and that are performed whenever these events arise
- Awakened by triggering event
- **If** condition is **true**, associated action is performed **else** nothing happens.
- Actions can be any sequence of database operations
- Tells exactly when to fire them (not like assertions)

Provisions in SQL for Triggers

- **Condition** and **Action** may be **executed** on current instances of all the relations (**before triggering event**) or on the state that exists **after the triggering event**.
- **Condition** and **Action** can refer to both **old** and **new** values of **tuples**
- It is possible to **define update events** that are limited to a **particular attribute** or **set of attributes.**

Provisions in SQL for Triggers

- Programmer can specify if trigger executes:
  - Once for each modified tuple (**a row-level trigger**)
  - Once for all tuples that are changed in one SQL (**a statement-level trigger**, remember that one SQL modifications statement can affect many tuples)

Syntax (ADD/DELETE)

```
CREATE TRIGGER <trigger-name>
AFTER/BEFORE/INSTEAD OF INSERT/DELETE/UPDATE OF <attribute-name> ON <table-name>
REFERENCING //optional
        OLD ROW/TABLE AS OldTuple/OldTable //optional
        NEW ROW/TABLE AS NewTuple/NewTable        //optional
FOR EACH ROW/FOR EACH STATEMENT
WHEN (<Condition>)
        BEGIN
                <Action - sequence of SQL statements>
        END
```

```
DROP TRIGGER <trigger-name>
```

Example

Schema:

Student(ID, name, address, GPA, sizeHS)

Campus(location, enrolment, rank)

Apply(ID, location, date, major, decision)

Problem:

If an application tuple is inserted for a student with GPA > 3.9 and sizeHS > 1500 to Berkeley, auto accept it.

Constraints:

1. Apply.ID and Apply.location appear in Student.ID and Campus.location respectively

2. Campus.rank <=10

```
CREATE TRIGGER AutoAccept
AFTER INSERT ON Apply
REFERENCING
        NEW ROW AS NewApp
FOR EACH ROW
WHEN (NewApp.location = 'Berkeley' AND
        3.9 < (SELECT GPA FROM Student WHERE ID = NEwApp.ID) AND
        1500 < (SELECT sizeHS FROM Student WHERE ID = NewApp.ID))
        UPDATE Apply
                SET decision = 'Y'
                WHERE ID = NewApp.ID
                AND location = NewApp.location
                AND date = NewApp.date
```

Example taken from http://infolab.stanford.edu/~ullman/fcdb/jw-notes06/constraints.html

Example

Schema:

Student(ID, name, address, GPA, sizeHS)

Campus(location, enrolment, rank)

Apply(ID, location, date, major, decision)

Problem:

If an application tuple is inserted for a student with GPA > 3.9 and sizeHS > 1500, auto accept it.

Constraints:

1. Apply.ID and Apply.location appear in Student.ID and Campus.location respectively

2. Campus.rank <=10

```
CREATE TRIGGER AutoAccept
AFTER INSERT ON Apply
REFERENCING
        NEW TABLE AS NewApp
UPDATE Apply
        SET decision = 'Y'
        WHERE (ID, location, date)  IN (SELECT ID, location, date FROM NewApps)
        AND location = 'Berkeley'
        AND 3.9 < (SELECT GPA FROM Student WHERE ID = Apply.ID)
        AND 1500 > (SELECT sizeHS FROM Student WHERE ID = Apply.ID)
```

Example taken from http://infolab.stanford.edu/~ullman/fcdb/jw-notes06/constraints.html

Example

Schema:

Student(ID, name, address, GPA, sizeHS)

Campus(location, enrollment, rank)

Apply(ID, location, date, major, decision)

Problem:

If campus enrolment increases from below 7,000 to above 7,000 delete all applications to that campus date after 2/15/20 and set all 'Y' decisions for applications before 2/15/20 to "u"

```
CREATE TRIGGER TooMany
AFTER UPDATE OF enrollment ON Campus
REFERENCING
        OLD ROW AS OldVal
        NEW ROW AS NewVal
FOR EACH ROW
WHEN (OldVal.enrollment <= 7,000 AND NewVal.enrollment > 7,000)
DELETE FROM Apply
        WHERE location = NewVal.location AND date > 2/15/20
UPDATE Apply
        SET decision = 'U'
        WHERE location = NewVal.location
                AND decision = 'Y'
```

Example taken from http://infolab.stanford.edu/~ullman/fcdb/jw-notes06/constraints.html

Example

Problem:
We want to check whenever an employee's salary is greater than the salary of his or her direct supervisor in the COMPANY database, then supervisor should be informed about this.

Schema:

Employee(name, ssn, bdate, address, gender, salary, super_Ssn)

Department(name, number, mgr_ssn, mgr_start_date)

DeptLocations(number, location)

Project(name, number, location, dnum)

Works_on(essn, pno, hours)

Dependent(essn, dependent_name, gender, bdate, relationship)

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN ON
Employee
FOR EACH ROW
WHEN (NEW.SALARY > (SELECT SALARY FROM EMPLOYEE
                    WHERE SSN = NEW.SUPERVISOR_SSN))
    INFORM_SUPERVISOR(NEW.Supervisor_ssn, NEW.ssn);
```

Book