Constraints and Integrity Constraints

CMPSC 305 – Database Systems

ALLEGHENY COLLEGE

Remember these shape fitting puzzles?!

Figure: How do constraints guide the completion of these puzzles?

Integrity Constraints

- The CONSTRAINTS enforce conditions to restrict attributes to contain a correct type of data while inserting or updating or deleting.
- Integrity constraints provide a mechanism for ensuring that data conforms to guidelines specified by the database administrator.

Adding Constraints to CREATE TABLE

```
CREATE TABLE relationshipTable (
  Attribute1 D1, (integrity-constraint 1),
  Attribute2 D2, (integrity-constraint 2),
  …,
  Attributen Dn , (integrity-constraint n),,
```

- relationshipTable is the name of the table
- Each Ai is an attribute name in the schema of relation relationshipTable
- Di is the data type of values in the domain of attribute Ai
  - The Di constrains the particular type of entry
- The integrity-constraint defines attribute application

Types of Affinity Constraints and Rules

The affinity of a column is determined by the declared type of the column, according to the following rules in the below order (source: https://www.sqlite.org/datatype3.html

1. If the declared type contains the string "INT" then it is assigned INTEGER affinity.
2. If the declared type of the column contains any of the strings "CHAR", "CLOB", or "TEXT" then that column has TEXT affinity. Notice that the type VARCHAR contains the string "CHAR" and is thus assigned TEXT affinity.
3. If the declared type for a column contains the string "BLOB" or if no type is specified then the column has affinity BLOB.
4. If the declared type for a column contains any of the strings "REAL", "FLOAT", or "DOUBLE" then the column has REAL affinity.
5. Otherwise, the affinity is NUMERIC.

Particular Constraints

Keyword	Type	Applies to Rule
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2

Particular Constraints

Keyword	Type	Applies to Rule
BLOB no datatype specified	BLOB	3
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

Constraints

- Note: In some varieties of SQL, the n may need to be defined
- In Sqlite3, we do not worry about defining n.

- **char(n):** Fixed length character string, with user-specified length n.
  - Used to store character string value of fixed length
  - The maximum num of chars (not important to SQLite3)
  - About 50 percent faster than VARCHAR
- **varchar(n):** Variable length character strings, with user specified maximum length n.
  - Used to store variable length alphanumeric data
  - The maximum num of chars (not important to SQLite3)
  - Slower than CHAR

Integrity Constraints

NOT NULL: To ensure that no NULL values are allowed
DEFAULT: When none is specified, this constraint provides a default value for a column.
CHECK: Ensures that all attribute values satisfy specified conditions
UNIQUE: To ensure that all values of an attribute are different
PRIMARY KEY: Uniquely identifies each row/record in a database table.

- Also, ensures potential links exist (as designed) between two tables

Simple NULL constraint demo

Spot the integrity constraint's influence

```
DROP TABLE IF EXISTS company;
CREATE TABLE company(
        Id text NOT NULL,
        Name text NOT NULL);
```

NOT NULL

Simple NULL constraint demo

```
INSERT INTO company VALUES("COM1","TS-LTD.");
```

```
INSERT INTO company VALUES("COM1","");
```

```
sqlite> SELECT * from company;
COM1|TS-LTD.
COM1|
```

```
/*Bad insert command: NULL is not allowed*/
INSERT INTO company VALUES("COM1",NULL);
```

Simple DEFAULT constraint demo

Spot the integrity constraint's influence

```
DROP TABLE IF EXISTS company;
CREATE TABLE COMPANY(
        ID INT PRIMARY KEY NOT NULL,
        NAME TEXT NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR,
        SALARY REAL DEFAULT 50000.00);
```

DEFAULT

Simple DEFAULT constraint demo

```
INSERT INTO COMPANY
        VALUES (70, "JAMES", 25, "10, Rue du fleur",100000);
```

```
/* Missing entry for SALARY*/
INSERT INTO COMPANY (ID, Name, AGE, ADDRESS)
        VALUES (221, "Sherlock", 25, "10, Rue du fleur");
```

```
sqlite> select * from company;
70|JAMES|25|10, Rue du fleur|100000.0
221|Sherlock|25|10, Rue du fleur|50000.0
```

Why will this line not work?

```
INSERT INTO COMPANY (ID, Name, AGE, ADDRESS)
        VALUES (221b, "Sherlock", 25, "10, Rue du fleur");
```

Simple CHECK constraint demo

Spot the integrity constraint's influence

```
DROP TABLE IF EXISTS company;
CREATE TABLE company(
        ID INT UNIQUE NOT NULL,
        NAME TEXT NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR,
        SALARY REAL CHECK(SALARY > 0));
```

CHECK

Simple CHECK constraint demo

```
/*Good insert command: complete tuple allowed*/
INSERT INTO company
        VALUES (221, "Sherlock", 25, "10, Rue du fleur",100000);
```

CHECK failure

```
INSERT INTO company VALUES
        (2211, "Sherlock", 25, "10, Rue du fleur", -10);
```

Simple UNIQUE constraint demo

Spot the integrity constraint's influence

```
DROP TABLE IF EXISTS company;
CREATE TABLE company(
        ID INT UNIQUE NOT NULL,
        NAME TEXT NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR,
        SALARY REAL );
```

UNIQUE

Simple UNIQUE constraint demo

```
INSERT INTO company
        VALUES (221, "Sherlock", 25, "10, Rue du fleur",100000);
```

Try to reinsert same values again

```
INSERT INTO company
        VALUES (221, "Sherlock", 25, "10, Rue du fleur",100000);

INSERT INTO company
        VALUES (NULL, "Sherlock", 25, "10, Rue du fleur",100000);
        /* What errors did you find? */
```

Returning to Unique Constraints …

```
DROP TABLE IF EXISTS company;
CREATE TABLE company(
        ID INT UNIQUE NOT NULL,
        NAME TEXT NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR,
        SALARY REAL );
```

Add some data

```
INSERT INTO company VALUES (10, "Sherlock Holmes", 25, "221b Baker Street", 100000);
INSERT INTO company VALUES (10, "Sherlock Holmes", 25, "221b Baker Street", 100000);
```

- NULL and repeating UNIQUE values are not inserted
- A UNIQUE constraint ensures all values in a column or a group of columns are distinct from one another or unique.

Returning to Unique Constraints …

```
DROP TABLE IF EXISTS company;
CREATE TABLE company(
        ID INT PRIMARY KEY,
        NAME TEXT NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR,
        SALARY REAL );
```

Add some data

```
INSERT INTO company VALUES (10, "Sherlock Holmes", 25, "221b Baker Street", 100000);
INSERT INTO company VALUES (10, "Sherlock Holmes", 25, "221b Baker Street", 100000);
```

- NULL and repeating UNIQUE values are not inserted
- A UNIQUE constraint ensures all values in a column or a group of columns are distinct from one another or unique.

Defining a New Table with a Primary Key

ID is unique, Salary bound by numbers

```
/*Two constraints?*/
DROP TABLE IF EXISTS Employee;
CREATE TABLE Employee (
            ID CHAR PRIMARY KEY,
            name VARCHAR NOT NULL,
            dept_name VARCHAR,
            salary NUMERIC
);
```

Defining a New Table with a Primary Key

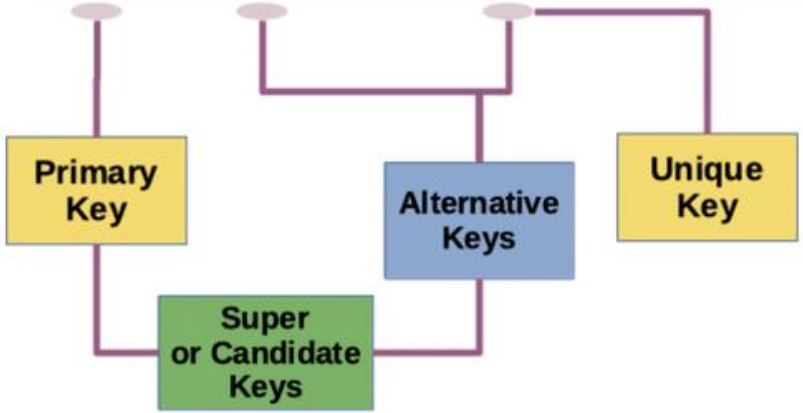
```
/******PSSST! Now Add some secret information **************************/
INSERT INTO Employee VALUES("001","Jimmy", "secretService", 1000000);
INSERT INTO Employee VALUES("002","Stevie", "secretService", 1000000);
INSERT INTO Employee VALUES("003","Frankie", "secretService", 10);
INSERT INTO Employee VALUES("004","Robbie", "secretService", 10A);

/* Oops! Robbie's salary information has a typographical error*/
INSERT INTO Employee VALUES("004","Robbie", "secretService", 100);
INSERT INTO Employee VALUES("004","Jamie", "secretService", 500);

/* Error: UNIQUE constraint failed: Employee.ID */
/* Huh?! */
```

Where are these keys?
Part 1

RowID	OrderNumber	BuyerName	BuyerID	BuyerAddress	DeptID
1	8000	Holmes	101	221b Baker Street	A
2	9192	Watson	102	221b Baker Street	A
3	1000	Hudson	103	221b Baker Street	A
4	4005	Moriarty	104	London	B
5	5003	Lestrade	105	Scotland yards	A
6	2303	Adler	106	London	B
7	5191	Milverton	107	London	B



Primary versus Unique Keys

Comparison Basis	Primary Key	Unique Key
Basic	The primary key is used as a unique identifier for each record in the table.	The unique key is also a unique identifier for records when the primary key is not present in the table.
NULL	We cannot store NULL values in the primary key column.	We can store NULL value in the unique key column, but only one NULL is allowed.
Purpose	It enforces entity integrity.	It enforces unique data.
Index	The primary key, by default, creates clustered index.	The unique key, by default, creates a non-clustered index.
Number of Key	Each table supports only one primary key.	A table can have more than one unique key.

Source: https://www.javatpoint.com/primary-key-vs-unique-key

Primary versus Unique Keys

Value Modification	We cannot change or delete the primary key values.	We can modify the unique key column values.
Uses	It is used to identify each record in the table.	It prevents storing duplicate entries in a column except for a NULL value.

Syntax	We can create a primary key column in the table using the below syntax:
	`CREATE TABLE Employee`
	`(`
	`Id INT PRIMARY KEY,`
	`name VARCHAR(150),`
	`address VARCHAR(250)`
	`)`
		We can create a unique key column in the table using the below syntax:
		`CREATE TABLE Person`
		`(`
		`Id INT UNIQUE,`
		`name VARCHAR(150),`
		`address VARCHAR(250)`
		`)`

Source: https://www.javatpoint.com/primary-key-vs-unique-key

AgentsDB: Two Tables, One With a Primary Key

Let’s play with code!

Alternative Keys

Accepts no redundancy

```
DROP TABLE Agents1;
CREATE TABLE Agents1
(           last_name VARCHAR NOT NULL,
            first_name VARCHAR NOT NULL,
            address VARCHAR,
            CONSTRAINT agents_pk
            PRIMARY KEY (last_name, first_name)
);
```

The last and first names are used to make a Candidate Key.

```
INSERT INTO Agents1 VALUES("Bond", "James", "London");
```

One with Primary Key

Accepts redundancy

```
/* Accepts redundancy */
DROP TABLE Agents2;
CREATE TABLE Agents2
( last_name VARCHAR NOT NULL,
first_name VARCHAR NOT NULL,
address VARCHAR
);
```

Accepts redundancy

```
INSERT INTO Agents2 VALUES("Bond", "James", "London");
```

Try Your Insert Twice

- Insert agent names again into both tables.
- Now try changing the values to see what can be inserted

```
INSERT INTO Agents1 VALUES("Bond", "James", "London");
INSERT INTO Agents2 VALUES("Bond", "James", "London");
```

Is James the plural form of Jame? Conclusions?

- There can only be one “James Bond”
- The name “James Bond” could not be inserted more than once in our base
- Constraints were in place to ensure distinguishable rows

Consider this ...

- Can you build a new database table with two (or more) types of constraints?
- For instance, try to alter an earlier database for which you have the build file to recreate it (in case anything goes dreadfully wrong)