Introduction to Database Systems: CS312 Constraints and Integrity Constraints

Oliver Bonham-Carter

15 March 2022

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.

Integrity Constraints

Common 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.
- **UNIQUE**: To ensure that all values of an attribute are different
- **PRIMARY KEY**: Uniquely identifies each row/record in a database table.
  - Ensure that a link exists between two tables.
- **CHECK**: Ensures that all attribute values satisfy specified conditions

Simple NULL constraint demo

Spot the integrity constraint's influence

```
drop table company;
create table company(
   Id text NOT NULL,
   Name text NOT NULL);
```

```
/*Good insert command: complete tuple allowed*/
INSERT INTO company VALUES("COM1","TS-LTD.");
SELECT * FROM company;
```

```
/*Good insert command: Empty spaces are allowed*/
INSERT INTO company VALUES("COM1","");
```

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

Simple DEFAULT constraint demo

Place predetermined value to a column when no value given

Spot the integrity constraint's influence

```
drop table 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);
```

```
/*Good insert command: complete tuple allowed*/
INSERT INTO COMPANY  VALUES (12, "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");
```

Spot the error!

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

Simple UNIQUE constraint demo

Prevents two records from having identical values in columns

Spot the integrity constraint's influence

```
/*Create table*/
drop table COMPANY;
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY NOT NULL,
   NAME TEXT NOT NULL,
   AGE INT NOT NULL UNIQUE,
   ADDRESS CHAR,
   SALARY REAL DEFAULT 50000.00 );
```

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

Age is not unique

```
SELECT * FROM company;
 /* Try to reinsert same values again. */
 /* Now Fix AGE. Find any other errors?*/
```

Simple PRIMARY KEY constraint demo

Only one primary key in a table; UNIQUE Identifiers

Spot the integrity constraint's influence

```
/*Create table*/
drop table 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 );
```

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

Key not unique failure

```
SELECT * FROM company;
 /* Try to reinsert same values again.*/
```

Simple CHECK constraint demo

Only one primary key in a table; UNIQUE Identifiers

Spot the integrity constraint's influence

```
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY NOT NULL,
   NAME TEXT NOT NULL,
   AGE INT NOT NULL,
   ADDRESS CHAR,
   SALARY REAL CHECK(SALARY > 0));
```

```
/*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);
```

Types of Constraints

Source: https://www.sqlite.org/datatype3.html

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

Types of Constraints :: Rules

Source: `https://www.sqlite.org/datatype3.html`

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

Types of Constraints :: Rules

Rules to determine Column Affinities

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.

Constraints
General SQL: Data types

- 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 per cent 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

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 A_i is an attribute name in the schema of relation relationshipTable
- D_i is the data type of values in the domain of attribute A_i
  - The D_i constrains the particular type of entry
- The `integrity-constraint` defines attribute *application*

Unique Constraint in the Code

```
/*Two constraints?*/
DROP TABLE instructor;
CREATE TABLE instructor (
    ID CHAR UNIQUE,
    name VARCHAR NOT NULL,
    dept_name VARCHAR,
    salary VARCHAR
 );
```

```
/******PSSST! Now Add some department information ****************************/
insert into instructor  values ('10211', 'Smith', 'Biology', 66000);

/*Any trouble inserting this next line?*/
insert into instructor  values ('10212', null, 'Biology', 66000);
insert into instructor  values ('10211', 'Franklin', 'Biology', 66000);
```

- 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 Table with Primary a Key?

ID is unique, Salary bound by numbers

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

```
/******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?! */
```

Primary versus Unique Keys

- **Primary Keys**: A unique or `NON NULL` key that uniquely identifies every record in that table or relation.
  - User cannot specify more than one primary key in any relationship (i.e., an attribute, or the column of the table)

- **Unique Keys**: Creates a single column or combination of columns in a table to uniquely identify database records.
  - A unique key prevents from storing duplicate values in the column. A table can contain multiple unique key columns, unlike a primary key column

- **General Difference**: The primary key identifies each record in the table, and the unique key prevents duplicate entries in a column except for a `NULL` value.

Primary versus Unique Keys

Part 1

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

Part 2

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! :-)

Building your Database

Two Tables, One with a Primary Key and One Without

- To try out the following example, you can build a populated database or one without data for manual populating.

Build an empty table: paste on one line in terminal

```
cat builder_agentsDB_notPopulating.txt |
  sqlite3 agentsDB_nonPopulating.sqlite3
```

Build a populated table

```
cat builder_agentsDB.txt | sqlite3 agentsDB.sqlite3
```

Two Tables, One with Primary Key

Super Keys: Using multiple attributes to enforce unique-ness

Accepts no redundancy

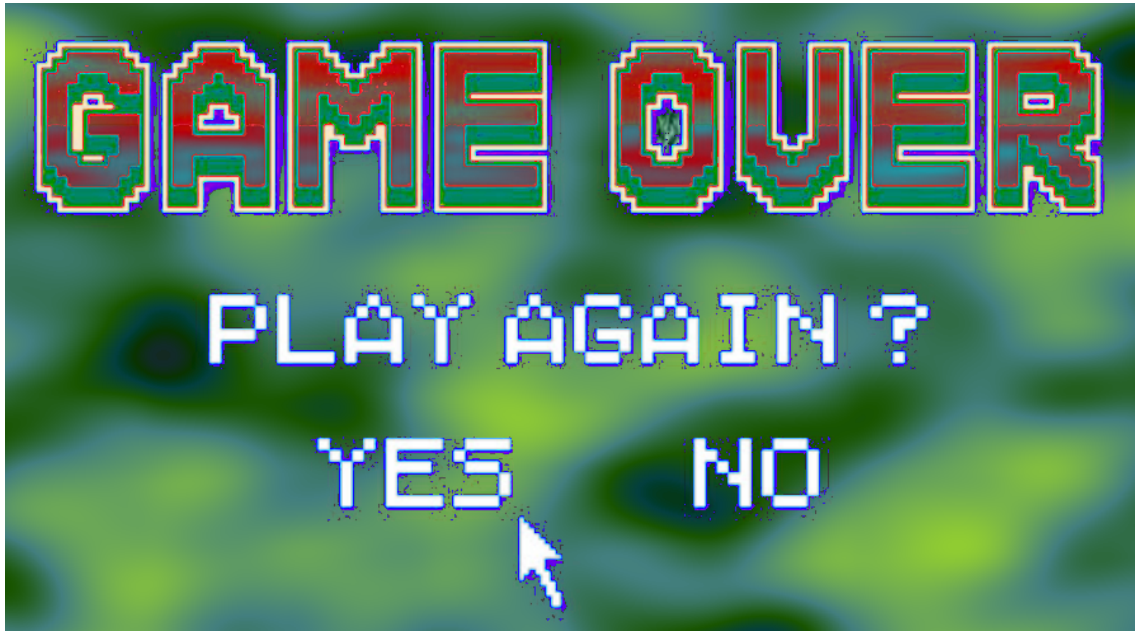
```
/* Accepts no redundancy */
DROP TABLE Agents1;
CREATE TABLE Agents1
( last_name VARCHAR NOT NULL,
  first_name VARCHAR NOT NULL,
  address VARCHAR,
  CONSTRAINT agents_pk
   PRIMARY KEY (last_name, first_name)
);
```

Accepts redundancy

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

Try Your Insert Twice

Let's try



- Insert agent names into both tables.
- Try the same `INSERT` commands again!
- Which commands work?

```
INSERT INTO Agents1 VALUES ("Bond","James","123 abc street");
INSERT INTO Agents2 VALUES ("Bond","James","123 abc street");
```

Forget-Me-Nots

How is the database set up?

- .tables (The tables are of the DB)
- .schema (How the data is stored in the tables)

What data is stored in each table?

- select * from agentsConst;
- select * from agents;
- (note the '*' for the column wildcard)

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)