CSCI 403
Database Management

8

Constraints, keys, indexes

CS@Mines

Restrictions on tables

CONSTRAINTS

CS@Mines 2

Types of Constraints

- Implicit (model-based)
- Explicit (schema-based)
- Application-based

CS@Mines 3

Implicit Constraints

Implied by model of world represented by DB

- E.g., a column named “instructor” should contain person names
- Not (usually) enforced

CS@Mines 4

Application-Based Constraints

- Not defined in DB
- Enforced by applications using DB
- AKA “business rules”
- Often too complex to reasonably implement on database side
- Example:

“no employee can have a hire date that is not on the first of some month”

CS@Mines 5

Explicit Constraints

- Domain constraints
  - Values in a column must match the type domain
- Primary key constraints (next section)
- Foreign key constraints (next section)

CS@Mines 6

KEYS

Keys (Theory)

Keys have a very specific definition in relational database theory.

We'll revisit this topic when we get to the theory portion of the class.

For now, we'll discuss the practical applications in an SQL database.

Primary Key

- Each table can have zero or one *primary key*
- Primary key applies to a subset of the columns
  - i.e., one or more identified columns form the key
  - Can be all columns!
- Does three things:
  - Constrains data in rows to be unique for the combination of columns in the primary key (example next page)
  - Constraints data in primary key columns to be not null
  - Creates an *index* on the combination of columns

Primary Key Example

Consider a **person** table:

ssn	first	last	birthdate	…
123-45-6789	Wood	Carpenter	4/17/1975	
111-22-3333	Opal	Miner	12/1/1982	
454-45-4545	Jane	Doe	6/2/1995	
…	…	…	…	

SSN would make a good primary key.
What about (first, last)?
Or (first, last, birthdate)?

Uniqueness Constraint

If we make (first, last) a primary key: cannot have more than one Jane Doe.

SSN is probably okay, unless someone has a fake identity…

If SSN is primary key, we *cannot* insert a duplicate SSN into the table!

SQL Example

```
CREATE TABLE person (
    ssn text PRIMARY KEY,
    first text,
    last text,
    birthdate date);

INSERT INTO person VALUES ('123-45-6789',
'Wood', 'Carpenter', '4/17/1975'); OK

INSERT INTO person VALUES ('123-45-6789',
'Evil', 'Impostor', ‘6/22/1963'); ERROR!
```

Indexes

- Short explanation:
  - Indexes speed searching by indexed columns
  - E.g., SELECT * FROM person WHERE ssn = '123-45-6789';
  - Speeds up most if *all* indexed columns are searched
    - Can get some speed up by searching first listed column, first two, etc.
- Longer explanation:
  - Without index, must do *linear search* – look at every row
  - With index, search through an optimized data structure called a Btree – look at only a fraction of rows
- We'll study Btrees later in the semester
  - For the curious:
    - Like a binary search tree but with >2 children per node
    - Performance is $\log_b(n)$, where b is typically large (e.g. b=100)

Creating Primary Keys

```
CREATE TABLE person (
    ssn text PRIMARY KEY,
    first text,
    last text,
    birthdate date);
\d person
              Table "public.person"
  Column   | Type | Collation | Nullable | Default
-----------+------+-----------+----------+---------
 ssn       | text |           | not null |
 first     | text |           |          |
 last      | text |           |          |
 birthdate | date |           |          |
Indexes:
    "person_pkey" PRIMARY KEY, btree (ssn)
```

Multiple Column Primary Key

```
CREATE TABLE person2 (
    first text,
    last text,
    birthdate date,
    PRIMARY KEY (first, last));
\d person2
              Table "public.person2"
  Column   | Type | Collation | Nullable | Default
-----------+------+-----------+----------+---------
 first     | text |           | not null |
 last      | text |           | not null |
 birthdate | date |           |          |
Indexes:
    "person2_pkey" PRIMARY KEY, btree (first, last)
```

Foreign Key

A foreign key defines a constraint between two tables:

- An FK applies to a subset of columns of one table
- The FK *references* a subset of columns of second table
- FK constraint:
  - Values in FK column(s) must either:
    - Exist in columns in referenced table OR
    - Be NULL

Example

address

street	city	state	zip
123 Mockingbird Ln	Golden	CO	80402
7575 Elm St	Arvada	CO	80003
9101 Main St	Durham	NC	27709

state

state	code
Colorado	CO
North Carolina	NC
Virginia	VA
…	…

Example in SQL

```
CREATE TABLE state (
  state text, code char(2) UNIQUE);  -- or often PRIMARY KEY

CREATE TABLE address (
  street text,
  city text,
  state char(2) REFERENCES state (code),
  zip numeric(5));

\d address
                 Table "public.address"
 Column |     Type     | Collation | Nullable | Default
--------+--------------+-----------+----------+---------
 street | text         |           |          |
 city   | text         |           |          |
 state  | character(2) |           |          |
 zip    | numeric(5,0) |           |          |
Foreign-key constraints:
    "address_state_fkey" FOREIGN KEY (state) REFERENCES state(code)
```

FK Constraint Example

```
INSERT INTO address
VALUES ('129 West 81st Street', 'New
York', 'NY', 10024) OK

INSERT INTO address
VALUES ('221B Baker St.', London, NULL,
NULL) OK

INSERT INTO address
VALUES ('4222 Clinton Way', 'Los
Angeles', 'XX', 90204) ERROR
```

CS@Mines 19

Another FK Example

```
INSERT INTO address
VALUES ('129 West 81st Street', 'New York',
'NY', 10024) OK

DELETE FROM state WHERE code = 'NY';  ERROR
```

(If we had no addresses in NY, would be fine.)

CS@Mines 20

Multiple Column Foreign Key

```
CREATE TABLE foo (
   x integer,
   y date,
   PRIMARY KEY (x, y));    -- or UNIQUE

CREATE TABLE bar (
   zz text,
   xx integer,
   yy date,
   FOREIGN KEY (xx, yy) REFERENCES foo(x, y));
```

CS@Mines 21

MISCELLANEOUS CONSTRAINTS

CS@Mines 22

Miscellaneous Constraints

- NOT NULL – disallows NULL values in column
- UNIQUE – imposes uniqueness on a column or set of columns (and creates index)
- CHECK constraints – requires meeting a Boolean condition
- Example:

```
CREATE TABLE foo (
  id integer CHECK (id > 0),
  name text NOT NULL,
  x integer UNIQUE );

\d foo

                 Table "public.foo"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           |          |
 name   | text    |           | not null |
 x      | integer |           |          |
Indexes:
    "foo_x_key" UNIQUE CONSTRAINT, btree (x)
Check constraints:
    "foo_id_check" CHECK (id > 0)
```

CS@Mines 23

Multiple Column Uniqueness

```
CREATE TABLE bar (
  a integer,
  b text,
  UNIQUE (b, a));
\d bar
                Table "public.bar"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 a      | integer |           |          |
 b      | text    |           |          |
Indexes:
    "bar_b_a_key" UNIQUE CONSTRAINT, btree (b, a)
```

CS@Mines 24

Multiple Column Uniqueness

```
CREATE TABLE bar (
  a integer,
  b text,
  UNIQUE (b, a));

INSERT INTO bar VALUES (1, 'apple');  OK
INSERT INTO bar VALUES (2, 'pear');  OK
INSERT INTO bar VALUES (1, 'banana');  OK
INSERT INTO bar VALUES (3, 'pear');  OK
INSERT INTO bar VALUES (1, 'apple');  ERROR
```

CS@Mines 25

Notes

- Constraints can be combined, e.g.
  CREATE TABLE foo (x integer UNIQUE NOT NULL …);
- Foreign key constraints also known as *referential integrity* constraints
- Foreign key relationships often mirror a common/likely choice of join

CS@Mines 26

Up Next

- Next lecture:
  Miscellaneous DDL: default column values, sequences, ALTER TABLE, views, indexes, DROP

CS@Mines 27