Databases for Data Science

Lecture 05 · 2022-09-13

Today

- Keys
- Table constraints
- Normal forms

Keys

How can we uniquely identify entities in our database?

Keys

A **superkey** is a subset of attributes (columns) that is distinct for each tuple (row).

A **key** is a superkey that is *minimal*: removing any attribute will produce something that is not a superkey.

- These are properties of the schema, *not* of a particular state of the database.
  They must hold for every possible dataset.

Keys

A **superkey** is a subset of attributes (columns) that is distinct for each tuple (row).

A **key** is a *minimal* superkey: if any column is removed, it will no longer be a superkey.

A table may have more than one key; we call each one a **candidate key**.

We may choose to designate one of the candidates as a **primary key**.

Keys

Other vocab:

- When a key consists of more than one column, it is a **composite key**.
- A key that consists of columns with real-world meaning is a **natural key**.
  - Examples?
- By contrast, a **surrogate key** has no real-world meaning and is purely internal to the database.

Why use surrogate vs natural?

Keys: recap

A **superkey** is a set of one or more columns that are unique in each row.

A **key** is a minimal superkey.

A table may have more than one key; we call each one a **candidate key**.

We may choose to designate one of the candidates as a **primary key**.

A **composite key** consists of two or more columns.

If a key has real-world meaning, it is a **natural key**.

If not, it's a **surrogate key**.

Foreign keys

A **foreign key** is a (set of) attributes that refer to the primary key of some other record.

- e.g., if `student_id` is a PK for `student`, we might have it as an FK in the `financial_aid` table.

Where are these useful?

Foreign keys

Referential integrity: each instance of a foreign key *must* refer to an entity that exists in the key's home table.

- If each row in `financial_aid` contains a foreign key `student_id` to the `student` table, then there must be a `student` with that ID.

A foreign key can refer to the same table.

- If each student is assigned a roommate, `roommate_student_id` could be a foreign key in `student`.

Keys: recap

A **superkey** is a set of one or more columns that are unique in each row.

A **key** is a minimal superkey.

A table may have more than one key; we call each one a **candidate key**.

We may choose to designate one of the candidates as a **primary key**.

A **composite key** consists of two or more columns.

If a key has real-world meaning, it is a **natural key**.

If not, it's a **surrogate key**.

A **foreign key** refers to the primary key of another record.

Exercise: identify candidate keys and foreign keys in `bookings`, `booking_dates`, `charges`.

Entity-Relation (ER) Diagrams

[blackboard]

Many-to-Many Relationships

[blackboard]

Integrity and Constraints

Referential integrity is one form of *integrity*: a condition that must hold for the database to be in a valid state.

We might also want to ensure other conditions hold.

Examples?

Integrity and Constraints

Referential integrity is one form of *integrity*: a condition that must hold for the database to be in a valid state.

We might also want to ensure other conditions hold.

Examples?

- A column is not null
- An `end_date` comes after a `start_date`
- Every item in a column is unique

Column constraints

Set per-column when a table is created (or altered).

```
CREATE TABLE student (
  -- both unique and not null
  student_id INT UNIQUE NOT NULL,

  -- not null
  name TEXT NOT NULL,

  -- custom logical constraint
  gpa NUMERIC CHECK (gpa >= 0.0 AND gpa <= 4.0)
);
```

Defaults

Similarly, we can have default values:

```
CREATE TABLE location (
  -- Provide a default country (required)
  country text NOT NULL DEFAULT 'United States',

  -- Provide a default state (optional)
  state text DEFAULT 'Florida',

  -- No default for city
  city text NOT NULL
);
```

Defaults

```sql
CREATE TABLE location (
  country text NOT NULL DEFAULT 'United States',
  state text DEFAULT 'Florida',
  city text NOT NULL
);

-- Note that we have to say which columns we're inserting into
-- Otherwise, we would be inserting into the first column (country)
INSERT INTO location (city) VALUES ('Sarasota'),('Tampa');

-- We can manually insert a `NULL` for state, if we want to
INSERT INTO location (state, city) VALUES (NULL, 'Washington D.C.');

-- Or we can override all of the defaults
INSERT INTO location VALUES ('Canada','Ontario','Toronto');
```

Key constraints

`PRIMARY KEY` = `UNIQUE NOT NULL`

Often, you want to have an auto-incrementing primary key:

```
CREATE TABLE student (
  -- Generate a new student_id for each row inserted
  student_id SERIAL PRIMARY KEY,

  name TEXT NOT NULL
);

-- We don't need to provide a student_id when inserting
INSERT INTO student (name) VALUES ('Alex'),('Blake');
```

(`SERIAL` is shorthand for `integer DEFAULT NEXTVAL('tablename')`)

Key constraints

`REFERENCES` (foreign key): must refer to an existing row in the target table.

```
CREATE TABLE scholarship (
  -- primary key
  scholarship_id SERIAL PRIMARY KEY,

  -- foreign key to the student table (required)
  student_id INT NOT NULL REFERENCES student(student_id),

  -- foreign key to this table (optional)
  previous_scholarship_id INT REFERENCES scholarship(scholarship_id),

  -- some non-key attributes
  dollar_amount MONEY NOT NULL,
  title TEXT
);
```

Key constraints

What happens if we try to delete a row from `student` that is referenced in `scholarship`?

- What are some things we might *want* to have happen?

Deletion behavior

When the referenced row is being deleted...

- `ON DELETE RESTRICT`: throw an error, preventing deletion
  - This is the default behavior
- `ON DELETE CASCADE`: delete *this* row, too
- `SET NULL`: set the FK column to `NULL`
- `SET DEFAULT`: set the FK column to its `DEFAULT` value
  - which must still be a valid key!

Deletion behavior

```
CREATE TABLE scholarship (
  -- primary key
  scholarship_id SERIAL PRIMARY KEY,

  -- delete this scholarship if its student is deleted
  student_id INT NOT NULL
    REFERENCES student(student_id)
    ON DELETE CASCADE,

  -- blank out this field if the referenced scholarship is deleted
  previous_scholarship_id INT
    REFERENCES scholarship(scholarship_id)
    ON DELETE SET NULL,

  dollar_amount MONEY NOT NULL,
  title TEXT
);
```

Constraints

Constraints

`SERIAL PRIMARY KEY`
`REFERENCES thing(thing_id)`
`DEFAULT`
`NOT NULL`
`CHECK (...formula...)`

Deletion behavior

`ON DELETE` ...

- `RESTRICT`
- `CASCADE`
- `SET NULL`
- `SET DEFAULT`

Group exercise: design an employee database with an `employee` and `contact_info` table. Identify primary/foreign kys and suggest some appropriate constraints.

Table constraints

Like column constraints, but can apply to more than one column.

```
CREATE TABLE registration (
  student_id INTEGER
    REFERENCES student(student_id),

  course_id INTEGER
    REFERENCES course(course_id),

  distance_learning BOOLEAN DEFAULT FALSE,

  -- Make these two columns the primary key
  PRIMARY KEY(student_id, course_id)
);
```

```
CREATE TABLE style (
  style_id SERIAL PRIMARY KEY,
  color TEXT,
  pattern TEXT,

  -- Require every pair of these columns to be unique
  UNIQUE(color, pattern)
);
```

Normal forms

The design of a database may influence its efficiency and ease of use.

What are some issues we've seen in `homelessness`?

Normal forms

There are many - today we will cover First, Second, and Third Normal Form.

- 1NF, 2NF, 3NF

First normal form

Every attribute is atomic.

- No multi-valued attributes
  - e.g. a list of phone numbers in a single row
- No composite attributes
  - e.g. a filed containing both a phone number and a name

Every entity is distinct.

- e.g., shouldn't insert the same row twice to list multiple instances of something.

First normal form

- Every attribute is atomic.
- Every row is distinct.

Exercise: Does `charges` violate 1NF?

First normal form

How to normalize?

- Composite attributes: _____
- Lists: _____
- Duplicates: _____

First normal form

How to normalize?

- Composite attributes: split into multiple columns
- Lists:
  - Create a new table to store each item, or
  - Split into multiple rows
- Duplicates:
  - Add a surrogate key that is unique to each row, or
  - Merge the rows and add a new "count" column

First normal form

How to normalize?

- Composite attributes: split into multiple columns
- Lists:
  - Create a new table to store each item, or
  - Split into multiple rows
- Duplicates:
  - Add a surrogate key that is unique to each row, or
  - Merge the rows and add a new "count" column

Group exercise: Normalize the `orders` table in your personal DB to 1NF.

Functional dependencies

What is a *function*, in math terms?

Functional dependencies

What is a *function*, in math terms?

First - a **relation** is a set of values that are related to each other.

$$Person(ssn, firstname, lastname)$$

We say the tuple (row) $(123-45-6789, Alex, Ample)$ is in the relation (table) $Person$ if such a person exists.

Functional dependencies

A **function** is a relation $F(x, y)$ in which every value of x is related to exactly one value of y.

- We can then write $f(x) = y$.

Similarly: every ssn maps to exactly one $(firstname, lastname)$.

Functional dependency

A column (or group of columns) x
is **functionally dependent**
on a column (or group) y
if x uniquely determines y.

- So, you can't have a different y for the same x.
- So, you can't have a different $(firstname, lastname)$ for the same ssn.

(You could imagine a function `get_name(ssn)`.)

Second normal form

1NF, and:

There is no non-key attribute which functionally depends on *part* of a candidate key.

- e.g., suppose we have `(street_address, county, tax_rate, ...)` and `(street_address, county)` is a candidate key.
- Then `tax_rate` depends only on `county`, violating 2NF.

This can be tricky!

Second normal form

How to normalize?

Second normal form

How to normalize?

Move the dependent column to a new table along with a copy of its key

- e.g., create a `county_tax_rate` table.

Second normal form

There is no non-key attribute which functionally depends on *part* of a candidate key.

- e.g., suppose we have `(street_address, county, tax_rate, ...)` and `(street_address, county)` is a candidate key.
- Then `tax_rate` depends only on `county`, violating 2NF.
- Fix by creating a `county_tax_rate` table.

Group exercise: is your version of `orders` in 2NF? If not, then normalize it.

Third normal form

For any functional dependency $x \rightarrow y$, if y is a non-key then x is a superkey.

- i.e., every non-key attribute is fully and directly dependent on every key.
- i.e., there are no functional dependencies *from* something that is not a key.
- e.g., given `(property_id, county, tax_rate, ...)` where `property_id` is the PK, `county` is not a key and `county -> tax rate`.

Third normal form

For any functional dependency $x \rightarrow y$, if y is a non-key then x is a superkey.

- i.e., every non-key attribute is fully and directly dependent on every key.
- i.e., there are no functional dependencies *from* something that is not a key.
- e.g., given `(property_id, county, tax_rate, ...)` where `property_id` is the PK, `county` is not a key and `county -> tax rate`.

Exercise: does `bookings` violate 3NF? If so, how?

Third normal form

For any functional dependency $x \rightarrow y$, if y is a non-key then x is a superkey.

- i.e., every non-key attribute is fully and directly dependent on every key.
- i.e., there are no functional dependencies *from* something that is not a key.
- e.g., given `(property_id, county, tax_rate, ...)` where `property_id` is the PK, `county` is not a key and `county -> tax rate`.

Group exercise: are your `orders` tables in 3NF? If not, normalize them.