These lecture notes include some material from Professors Guagliardo, Bertossi, Kolaitis, Libkin, Vardi, Barland, McMahan

Key Constraints

Dr Eugenia Ternovska

Simon Fraser University

Keys: superkeys

We need to distinguish tuples within each relation

> *The attribute values of a tuple must be such that they must* **uniquely identify** *the tuple*

No two tuples of a relation are allowed to have the same values for all attributes[1]

A superkey is a set of one or more attributes that, taken collectively, allow us to identify a tuple uniquely

- {ID} in table Instructor is a superkey
- {Name} in table Instructor is NOT a superkey

[1] Some DBMSs relax this requirement

Keys: superkeys and candidate keys

Let R denotes the set of all attributes of relation r and $K \subseteq R$

Formally, K is a superkey of r if

for any tuples $t_1, t_2 \in r$, if $t_1 \neq t_2$ then $t_1.K \neq t_2.K$

(no two distinct tuples have the same values in all attributes of K)

A superkey may contain extraneous (unnecessary) attributes:

- {ID,Name} in table Instructor is a superkey

If K is a superkey, then so is any superset of K

Keys: candidate keys are "**minimal**" superkeys

A **candidate key** is a minimal superkey, i.e., no proper subset of it is a superkey

There could be several candidate keys

- Suppose that {Name,DepName} are sufficent to distinguish among members of the Instructor relation

- {ID} in relation (table) Instructor is a candidate key
- {Name,DepName} in relation Instructor is a candidate key

But:

- {ID,Name} in table Instructor is a NOT a candidate key because it is not minimal (includes ID)

Keys: primary keys

A **primary key** is a candidate key (i.e., a minimal set of attributes) **chosen by the database designer** as the main way to identify tuples within a relation

It is customary to underline primary keys of a relation schema:

STUDENT(student-id, student-name, major, status)

In database design, primary keys must be chosen with care:
their attributes should never change (e.g., SIN),
or change extremely rarely (e.g., internal company IDs may change when two companies merge)

Key constraints on a database

A key (primary, candidate or super) is the **property of the entire relation** rather than of the individual tuples

Any two individual tuples in a relation are prohibited from having the same values on the key attributes

Therefore, the designation of a key represents a **constraint** on the real-world enterprise being modelled

Keys are probably the **most basic** and **very essential** database constraints

Any update that effects the values of a key, or violate a key integrity constraint, will result in an error state for the DBMS

Key constraints later in the course

We will come back to key constraints when we talk about Functional Dependencies in databases

Foreign-key constraints: Motivation

Consider the attribute DepName in Instructor relation

Suppose we also have Department relation

It would not make sense for a tuple in Instructor to have a value for the attribute DepName that does not correspond to a department in the Department relation

It makes sense to have that

Instructor.DepName is a subset of Department.DepName

Foreign-key constraints

A **Foreign-key constraint** from attribute A of relation r_1 to the primary key B of relation r_2 states that,

for any database instance, the value of A for each tuple in r_1 must also be the value of B for some tuple in r_2. In symbols,

$$r_1.A \subseteq r_2.\underline{B}$$

Attribute set A is called **foreign key** from r_1 referencing r_2

- Attribute A (DepName) in relation r_1 (Instructor) is a foreign key from r_1 referencing relation r_2 (Department)
- Note: DepName must be the primary key in Department

As a result of this constraint,

Instructor.DepName is a subset of Department.DepName

Foreign-key constraints later in the course

We will come back to foreign-key constraints when we talk about Inclusion Dependencies in databases

Declaring Keys in SQL: Basic constraints

Keywords:

- **UNIQUE** to declare keys
- **NOT NULL** to disallow null values
- **PRIMARY KEY** key + **not NULL**
- **FOREIGN KEY** to reference attributes in other tables

NULL values are, generally, ignored when checking constraints except for **NOT NULL** and **PRIMARY KEY**

Declaring Keys in SQL: Example

```
CREATE TABLE Account (
accnum  VARCHAR(12) UNIQUE,
branch  VARCHAR(30),
custid  VARCHAR(10),
balance NUMERIC(14,2)
);
```

The following insertion gives an error:

```
INSERT INTO Account VALUES
(1, 'London',    'cust1', 100),
(1, 'Edinburgh', 'cust3', 200);
```

The following insertion succeeds:

```
INSERT INTO Account VALUES
(NULL, 'London',    'cust1', 100),
(NULL, 'Edinburgh', 'cust3', 200);
```

Compound keys

Keys consisting of more than one attribute
must be declared using a different syntax

```
CREATE TABLE Movies (
m_title    VARCHAR(30),
m_director VARCHAR(30),
m_year     SMALLINT,
m_genre    VARCHAR(30),
UNIQUE (m_title,m_year)
);
```

This declares the set {m_title,m_year} as a key for Movies

Primary Keys

Essentially **UNIQUE** + **NOT NULL**

```
CREATE TABLE Account (
accnum  VARCHAR(12) PRIMARY KEY,
branch  VARCHAR(30),
custid  VARCHAR(10),
balance NUMERIC(14,2)
);
```

same as

```
CREATE TABLE Account (
accnum  VARCHAR(12) NOT NULL UNIQUE,
branch  VARCHAR(30),
custid  VARCHAR(10),
balance NUMERIC(14,2)
);
```

Foreign keys in SQL (1)

```
CREATE TABLE Customer (
custid  VARCHAR(10) PRIMARY KEY
name    VARCHAR(20),
city    VARCHAR(30),
address VARCHAR(30)
);

CREATE TABLE Account (
accnum  VARCHAR(12),
branch  VARCHAR(30),
custid  VARCHAR(10) REFERENCES Customer(custid),
balance NUMERIC(14,2)
);
```

Every value for attribute `custid` in Account must appear among the values of the **primary key** `custid` in Customer

Foreign keys in SQL (2)

General syntax (useful for declaring compound foreign keys)

```
CREATE TABLE <table1> (
<attr> <type>,
...
<attr> <type>,
FOREIGN KEY (<list1>)
REFERENCES <table2>(<list2>)
);
```

where

- `<list1>` and `<list2>` are lists with the **same number** of attributes
- attributes in `<list1>` are from table `<table1>`
- attributes in `<list2>` are **unique** in `<table2>`

Referential integrity and database modifications (1)

Deletion can cause problems with foreign keys

Customer	**ID**	**Name**
	cust1	John
	cust2	Mary

Account	**Number**	**CustID**
	123456	cust1
	654321	cust2

where Account.CustID is a **foreign key** for Customer.ID

What happens if one deletes (cust1,John) from Customer?

Three approaches are supported in SQL:

1. Reject the deletion operation
2. Propagate it to Account by deleting also (123456,cust1)
3. "Don't know" approach: keep the tuple in Account, but set CustID value to **NULL**

Referential integrity and database modifications (2)

All three approaches are supported in SQL

```
CREATE TABLE <table1> (
<attr> <type>,
...
FOREIGN KEY <list1> REFERENCES <table2>(<list2>)
<approach>
)
```

where <approach> can be:

1. Empty: Reject deletions from <table2> causing the FK to be violated (this is the default when <approach> is not specified)
2. **ON DELETE CASCADE**: Propagate the deletion to <name> (tuples in <table1> that violate the FK will be deleted)
3. **ON DELETE SET NULL**: "Don't know" approach (the values of the attributes in <list1>, for tuples in <name> that violate the FK, are set to **NULL**)

Acknowledgements

[1] Database Systems: The Complete Book, 2nd EditionHector Garcia-Molina, Jeffrey D. Ullman, Jennifer WidomPrentice Hall, 2009

[2] Database System Concepts, Seventh EditionAvi Silberschatz, Henry F. Korth, S. SudarshanMcGraw-Hill, March 2019www.db-book.com

Additional references and resources used in preparation of this course are listed on https://canvas.sfu.ca/courses/77505/pages/references-and-resources or mentioned in slides.