

Key Constraints

Dr Evgenia 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 attributes¹

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

¹Some 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 attributes:

- ▶ $\{ID, Name\}$ in table **Instructor** is a superkey

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

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

Keys: candidate keys: **minimal** superkeys

There could be several candidate keys

- ▶ Suppose that $\{Name, DepName\}$ are sufficient to are sufficient 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
- ▶ $\{ID, Name\}$ 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 chosen by the database designer as the main way to identify tuples of within a relation

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

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

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 the key, or violate the key integrity will result in an error state for the DBMS

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

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`

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	Account	Number	CustID
	cust1	John		123456	cust1
	cust2	Mary		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 Edition Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom Prentice Hall, 2009

[2] Database System Concepts, Seventh Edition Avi Silberschatz, Henry F. Korth, S. Sudarshan McGraw-Hill, March 2019 www.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.