Fundamentals of Database Systems
COMPSCI 351

Instructor: Sebastian Link

The University of Auckland

SQL

Data Definition and Manipulation Language

A Standard Language for Relational Databases

Need a language that implements the relational model in a real DBMS

An Industry Standard for Relational Database Systems

- We will consider a subset of SQL-2008
- Newest features out of scope and mostly syntactic sugar
- Although standard, different implementations do differ

SEQUEL = **S**tructured **E**nglish **QUE**ry **L**anguage

SQL - DDL, DML, and QL

SQL is the industry standard for...

- Giving data definitions (defining schemata and instances)
- Manipulating data definitions (updates)
- Querying data

SQL provides many more constructs, for example for

- external schema definitions (views),
- (limited) physical access path definitions (indices),
- coupling with programming languages (embedded SQL,dynamic SQL) and
- authorization (access rights)

Data Definition Language

`CREATE` statement to define relation schemata

`CREATE TABLE` ⟨table_name⟩ ⟨attribute_specification⟩ ;

The attribute specification is

A (comma separated) list of items:

⟨attribute_name⟩ ⟨domain⟩

each of which may be followed by an optional `NOT NULL`

In SQL names are not case-sensitive

More Constructs for DDL

`DROP TABLE`

removes relation schemata from a database schema

`ALTER TABLE`

used to change existing relation schemata

Some Domains (1)

CHARACTER and VARCHAR

- character strings of fixed or variable length
- maximum length given as parameter, e.g. VARCHAR(32)

NATIONAL CHARACTER

handles character strings using another (national) font, e.g.,

- German
- Cyrillic
- Mandarin
- Hindi
- Kanji

and the corresponding encoding

Some Domains (2)

INTEGER and SMALLINT

- handles domains of integers
- integers between -2^n und $2^n - 1$, where n depends on the actual implementation

NUMERIC and DECIMAL

- handle fixed point numbers with parameters
  - *precision* for the total number of digits and
  - *scale* for the number of digits following the decimal point
- for DECIMAL there is a system-dependent maximum number of digits

Some Domains (3)

FLOAT, REAL and DOUBLE PRECISION

- handle floating point numbers
- for FLOAT there is a parameter for the desired precision

BIT and BIT VARYING

handle bit strings analogously to CHARACTER and VARCHAR

Some Domains (4)

DATE

handles date values (year, month, day)

TIME

- handles time values (hour, minute, second)
- the number of digits for seconds can be put in as a parameter

For other domains or variants refer to system manuals

Representing Missing Information

To insert some tuple into a relation, we must know all values of the tuple

That would be a severe restriction in practice

- values may not exist, or
- values may exist, but are currently unknown

SQL permits occurrences of a single kind of *null marker*

- We extend domains by a null marker symbol `null`
- Different `null` occurrences may have different meanings
- A tuple t is called X-total, if $t(A) \neq$ `null` for all $A \in X$

Example: Null Marker Occurrences

id	*first_name*	*last_name*	*year_born*
001	`null`	Plato	427 BC
002	`null`	Timberlake	1981

Null marker interpretation

- First row: *value does not exist*
- Second row: *value exists, but unknown*

Duplicate Tuples

Duplicate tuples have matching values on every attribute

Relations

- Every relation is a set, i.e., a collection of well-distinguished elements
- Hence, relations cannot contain any duplicate tuples (by definition)

In practice, database instances and results of queries are large

- Duplicate removal is an expensive operation
- Hence, duplicates are tolerated (but can be avoided if needed)

Example: Duplicate Tuples

id	*first_name*	*last_name*	*year_born*
002	Justin	Timberlake	1981
002	Justin	Timberlake	1981

A relation?

No, duplicates!

id	*first_name*	*last_name*	*year_born*
002	`null`	Timberlake	1981
002	`null`	Timberlake	1981

A relation?

No, nulls! duplicates?

id	*first_name*	*last_name*	*year_born*
002	`null`	Timberlake	1981
003	`null`	Timberlake	1981

A relation?

No, nulls!

SQL Semantics

Definition (Table Schema)

A finite non-empty set of attributes

- denoted by $R = \{A_1, \ldots, A_n\}$, same as a relation schema
- but domains of an attribute extended by null marker `null`

Definition (Table)

A finite multi-set r of partial tuples

- partiality of tuple t means that $t(A)$ may be `null` for some attribute A
- multi-set means that duplicate (partial) tuples may occur
- table is said to be X-total, if all its tuples are X-total

Example: A Table that is not a Relation

id	*first_name*	*last_name*	*year_born*
002	`null`	Timberlake	1981
002	`null`	Timberlake	1981

Properties

- is defined over table schema $\{id, first\_name, last\_name, year\_born\}$
- is $\{id, last\_name, year\_born\}$-total
- is not a relation as it contains two duplicate tuples

Relations are idealized special cases of tables where neither duplicate tuples nor partial tuples are permitted to occur

NOT NULL Constraints

We may add constraints such as NOT NULL, uniqueness constraints, primary keys and foreign keys to a table definition

NOT NULL constraints can be attached to an attribute

- disallow any occurrences of the null maker on that attribute
- a table satisfies the NOT NULL constraint on A, if it is A-total

UNIQUE Constraints

Uniqueness constraints are specified using the keyword UNIQUE, that is,

UNIQUE (⟨attribute-list⟩)

Semantics

- different tuples must not have total matching values on all attributes in the list
- A table satisfies UNIQUE(X) (shorter: $u(X)$), if there are no different X-total tuples t, t' in the table such that $t(X) = t'(X)$

PRIMARY KEY Constraints

Primary keys can be defined by

`PRIMARY KEY` (⟨attribute-list⟩)

Semantics

- satisfied by tables that satisfy `UNIQUE`(⟨attribute-list⟩), and
- `NOT NULL` for every attribute from the list

Alternatively, the keyword `PRIMARY KEY` can simply be attached to an attribute, if there is just one attribute in the uniqueness constraint

Example for UNIQUE and PRIMARY KEY constraints

title	*author*	*journal*
The uRNA database	Zwieb C	Nucleic Acids 1997
The uRNA database	Zwieb C	Nucleic Acids 1996
Genome wide detect.	Ragh. R	`null`

UNIQUE?

- only *journal*

PRIMARY KEYs?

none!

Genius Question

What is remarkable about this example?

Syntax of Referential Constraints

Foreign keys can be defined by

`FOREIGN KEY` (⟨attribute-list⟩) `REFERENCES` ⟨table-name⟩ (⟨attribute-list⟩);

Naming convention for foreign keys

- child schema: table schema on which foreign key is defined
- parent schema: table schema which foreign key references
- `FOREIGN KEY` $(A_1, \ldots, A_n)$ `REFERENCES` $P(B_1, \ldots, B_n)$ on child schema C

- Attribute-list $(B_1, \ldots, B_n)$ can be omitted, if $A_1 = B_1, \ldots, A_n = B_n$
- `UNIQUE`$(B_1, \ldots, B_n)$ must be enforced on P

Semantics of Referential Constraints

`FOREIGN KEY` $(A_1, \ldots, A_n)$ `REFERENCES` $P(B_1, \ldots, B_n)$

Foreign key enforces that...

- for every $\{A_1, \ldots, A_n\}$-total tuple t_c in child table there is
- some tuple t_p in parent table such that
- $t_c(A_1) = t_p(B_1), \ldots, t_c(A_n) = t_p(B_n)$

Example of a Relational Database Schema

Movie=
{title, year, country, run_time, genre}

with key {title, year}

Person=
{id, first_name, last_name, year_born}

with key {id}

Actor={id, title, year, role}

- with key {id, title, year, role}
- and foreign keys:
  - [id] $\subseteq$ Person[id]
  - [title, year] $\subseteq$ Movie[title, year]

Director={id, title, year}

- with key {title, year}
- foreign keys:
  - [id] $\subseteq$ Person[id]
  - [title, year] $\subseteq$ Movie[title, year]

SQL Definition of Example

```
CREATE  TABLE MOVIE (
        title varchar(40) NOT NULL,
        year number(4) NOT NULL,
        country varchar(20) NOT NULL,
        run_time number(4) NOT NULL,
        genre varchar(10) NOT NULL,
        CONSTRAINT pk_movie PRIMARY KEY(title,year) );
```

```
CREATE  TABLE PERSON(
        id char(8) PRIMARY KEY,
        first_name varchar(15) NOT NULL,
        last_name varchar(15) NOT NULL,
        year_born number(4),
        PRIMARY KEY(id) );
```

SQL Definition of Example Continued

```
CREATE  TABLE ACTOR(
        id char(8) NOT NULL,
        title varchar(40) NOT NULL,
        year number(4) NOT NULL,
        role varchar(40) NOT NULL,
        PRIMARY KEY(id, title, year, role),
        FOREIGN KEY(title, year) REFERENCES MOVIE(title, year),
        FOREIGN KEY(id) REFERENCES PERSON );
```

```
CREATE  TABLE DIRECTOR(
        id char(8) NOT NULL,
        title varchar(40) NOT NULL,
        year number(4) NOT NULL,
        PRIMARY KEY(title, year),
        FOREIGN KEY(title, year) REFERENCES MOVIE(title, year),
        FOREIGN KEY(id) REFERENCES PERSON(id) );
```

Pop-quiz

Four different SQL table definitions (without domains)

- `CREATE TABLE` S1(id `PRIMARY KEY`, name `NOT NULL`);
- `CREATE TABLE` S2(id `NOT NULL`, name `NOT NULL`);
- `CREATE TABLE` S3(id `UNIQUE`, name `NOT NULL`);
- `CREATE TABLE` S4(id, name `NOT NULL`);

t_1

id	*name*
1	Rihanna
2	Gaga

t_2

id	*name*
1	Rihanna
1	Gaga

t_3

id	*name*
`null`	Eminem
3	Eminem

t_4

id	*name*
`null`	Eminem
1	Rihanna
1	Gaga

Which tables are instances of which table schema?

Pop-quiz

Four different SQL table definitions (without domains)

- `CREATE TABLE` S1(id `PRIMARY KEY`, name `NOT NULL`);
- `CREATE TABLE` S2(id `NOT NULL`, name `NOT NULL`);
- `CREATE TABLE` S3(id `UNIQUE`, name `NOT NULL`);
- `CREATE TABLE` S4(id, name `NOT NULL`);

t_1

id	*name*
1	Rihanna
2	Gaga

over S1-S4

t_2

id	*name*
1	Rihanna
1	Gaga

over S2 and S4

t_3

id	*name*
`null`	Eminem
3	Eminem

over S3 and S4

t_4

id	*name*
`null`	Eminem
1	Rihanna
1	Gaga

over S4

Foreign Key Example

Consider the following version of our SQL schema

```
CREATE TABLE MOVIE' (
title varchar(40),
year number(4),
country varchar(20),
run_time number(4),
genre varchar(10),
PRIMARY KEY(title,year) );
```

```
CREATE TABLE DIRECTOR'(
id char(8) NOT NULL,
title varchar(40),
year number(4),
UNIQUE(title, year),
FOREIGN KEY(title, year) REFERENCES MOVIE'(title, year),
FOREIGN KEY(id) REFERENCES PERSON(id) );
```

Does the following database satisfy the foreign key

FOREIGN KEY(title, year) REFERENCES MOVIE'(title, year)?

DIRECTOR'

id	*title*	*year*
1	null	2012
8	Nosferatu	1922

MOVIE'

title	*year*	*country*	*run_time*	*genre*
Nosferatu	1922	Germany	80	Horror
Mana Waka	1937	New Zealand	85	History

Referential Actions

Specify what happens in case of deleting or updating referenced tuples

SQL offers four possibilities

- `CASCADE` will force the referencing tuples to be deleted
- `SET NULL` will force the corresponding values in the referencing tuples to be set to a null marker (i.e., `null`)
- `SET DEFAULT` will force the corresponding values in the referencing tuples to be set to a specified default value
- `NO ACTION` does nothing, i.e., the foreign key constraint will be violated

Example for Referential Action

Consider a foreign key on DIRECTOR again

`FOREIGN KEY`(title,year) `REFERENCES` MOVIE(title,year) `ON DELETE CASCADE`

the deletion of a movie would trigger the deletion of its director

Another Example for Referential Action

Consider a different foreign key on DIRECTOR

`FOREIGN KEY`(title,year) `REFERENCES` MOVIE(title,year) `ON DELETE SET NULL`

the deletion of a movie would trigger that the information of the person who is the director of that movie is kept, but we loose the information which movie was directed

Extensions of the CREATE Statement

So far, we only considered the definition of relation schemata (CREATE TABLE)

Define integrity constraints outside the definition of relation schemata

CREATE ASSERTION ⟨name⟩ CHECK ...

Define new domains together with a check-clause

CREATE DOMAIN ...

More Extensions of the CREATE Statement

Define views (external schemata)

```
CREATE VIEW ⟨name⟩ AS ⟨query⟩
```

Define table schema only for duration of the current SQL session

```
CREATE GLOBAL TEMPORARY TABLE ...
```

Define temporary table schema only to be accessed in one module or ESQL program

```
CREATE LOCAL TEMPORARY TABLE ...
```

Example for Views and Temporary Tables

```
CREATE VIEW DIRECTED_ROLES AS
        SELECT d.id, a.role
        FROM DIRECTOR d, ACTOR a
        WHERE d.title = a.title
        AND d.year = a.year ;
```

```
CREATE GLOBAL TEMPORARY TABLE ACTORS_ROLES(
        id char(8) NOT NULL,
        first_name varchar(15) NOT NULL,
        last_name varchar(15) NOT NULL,
        role varchar(30) NOT NULL,
        PRIMARY KEY (id, role) );
```

Data Manipulation Language

The DML of SQL contains statements to insert, delete or update data in relations

For insertions we use one of

- `INSERT INTO` ⟨name⟩ `VALUES (` ⟨value-list⟩ `) ;`
- `INSERT INTO` ⟨name⟩ `(` ⟨attribute-list⟩ `) VALUES (` ⟨value-list⟩ `) ;`
- `INSERT INTO` ⟨name⟩ ⟨query⟩ `;`

For deletions we use a form similar to queries

`DELETE FROM` ⟨name⟩ `WHERE` ⟨condition⟩ `;`

For updates we use value assignments ⟨attribute⟩ = ⟨value⟩ in

`UPDATE` ⟨name⟩ `SET` ⟨value-assignment-list⟩ `WHERE` ⟨condition⟩ `;`

Example for Data Manipulation

- `INSERT INTO PERSON VALUES (10, 'Taylor', 'Lautner', 1992);`
- `INSERT INTO PERSON VALUES (11, 'Ziyi', 'Zhang', 1979);`
- `INSERT INTO PERSON (id, first_name, last_name) VALUES (12, 'Benicio', 'Del Toro');`
- `INSERT INTO PERSON (id, first_name, last_name) VALUES (13, 'Sophie', 'Marceau');`
- `UPDATE PERSON SET year_born = '1966' WHERE id = 13;`
- `DELETE FROM PERSON WHERE last_name = 'Del Toro' OR first_name = 'Taylor';`
- `DELETE FROM PERSON WHERE id >= 11;`

PERSON

id	*f_name*	*l_name*	*year_born*
10	Taylor	Lautner	1992
11	Ziyi	Zhang	1979
12	Benicio	Del Toro	null
13	Sophie	Marceau	1966

Summary of SQL as a DDL and DML

- SQL is the industry standard for defining and querying data
- SQL is founded on the relational model of data
- SQL allows us to specify relational database schemata
- SQL allows us to specify integrity constraints to restrict the data to those considered meaningful to the application at hand
  - domain constraints
  - uniqueness constraints, keys and foreign keys
  - triggers can enforce more expressive integrity constraints
- SQL allows us to manipulate data: insert, delete, update
- SQL allows us to use `null` markers to ease data entry
- SQL's default is to preserve duplicate tuples to facilitate data processing
- SQL's features as a query language are discussed later on