Gagnasafnsfræði

Venslalíkanið (The Relational Model)

Hallgrímur H. Gunnarsson

Part I: The Relational Model

Relational Database: Definitions

Relational database: a set of *relations*

Relation: made up of two parts: schema and instance

Schema: specifies name of relation, plus name and type of each column

```
Students(sid: string, name: string, login: string)
```

Instance: a table, actual data at a given time

Cardinality: number of rows

Degree/arity: number of fields

Example: Instance of Students Relation

Schema:

```
Students(sid: string, name: string, login: string)
```

Instance:

sid	name	login
50000	Dave	dave@cs
53666	Jones	jones@cs
53688	Smith	smith@eecs
53650	Smith	smith@math

Terms: relation, table, tuple (row/record), attribute (column/field), domain (type)

Example: Instance of Students Relation (2)

Instance:

sid	name	login
50000	Dave	dave@cs
53666	Jones	jones@cs
53688	Smith	smith@eecs
53650	Smith	smith@math

Cardinality = 4, degree = 3, all rows distinct

Do all columns in a relation instance have to be distinct?

SQL

SQL: the standard query language for relational databases

Data Definition Language (DDL):

1. create, modify, delete relations (tables)
2. specify constraints
3. administer users, security, etc.

Data Manipulation Language (DML):

1. Specify queries to find tuples that satisfy criteria
2. insert, update, delete tuples

Creating relations in SQL

Create the Students relation:

```
CREATE TABLE Students
(
    sid INTEGER, -- Or SERIAL in PostgreSQL
    name TEXT,
    login VARCHAR(32)
);
```

Generally (albeit simplified):

```
CREATE TABLE <name> (<field> <domain>, ...);
```

Example: CREATE TABLE in PostgreSQL

psql shell:

```
postgres=# CREATE TABLE Students (
postgres(# sid INTEGER,
postgres(# name TEXT,
postgres(# login VARCHAR(32));
CREATE TABLE
postgres=# \d Students
          Table "public.Students"
 Column |          Type          | Modifiers
--------+------------------------+-----------
 sid    | integer                |
 name   | text                   |
 login  | character varying(32)  |

postgres=#
```

Destroying and Altering Relations

Destroy the Students relation (table and data dropped):

```
postgres=# DROP TABLE Students;
DROP TABLE
postgres=#
```

Altering the Students relation:

```
postgres=# ALTER TABLE Students ADD ssn VARCHAR(10);
ALTER TABLE
postgres=# ALTER TABLE Students DROP ssn;
ALTER TABLE
postgres=#
```

Adding and Deleting Tuples (rows)

Insert a single tuple:

```
postgres=# INSERT INTO Students (sid,name,login)
postgres-# VALUES (12000, 'Hallgrimur H. Gunnarsson', 'hhg@cs');
INSERT 0 1
postgres=#
```

Delete all tuples satisfying some condition:

```
postgres=# DELETE FROM Students WHERE login='hhg@cs';
DELETE 1
postgres=#
```

More generally:

```
DELETE FROM <name> WHERE <condition>
```

Integrity Constraints (ICs)

IC: condition that must be satisfied for *any* instance of the database

1. ICs are specified when schema is defined
2. ICs are checked when relations are modified

A *legal* instance of a relation is one that satisfies all specified ICs

If the DBMS checks ICs, stored data is more faithful to "real-world" meaning

Also avoids inconsistencies, data entry errors, etc.

Keys

Keys are a way to associate tuples in different relations

Key associations can be enforced by integrity constraints in schema definitions

Students:

sid	name	login
50000	Dave	dave@cs
53666	Jones	jones@cs
53688	Smith	smith@eecs
53650	Smith	smith@math

Enrolled:

sid	cid	grade
50000	History 101	A
50000	Math 101	A
53666	History 101	B
53650	Databases	A

The field Students.sid is primary key

The field Enrolled.sid is a foreign key that references Students.sid

Primary Keys

A set of fields is a *key* for a relation if:

1. No two distinct tuples can have the same values in all key fields, and
2. This is not true for any subset of the key

If part one is true but part two is false then we have a *superkey*

A *key* is minimal in the sense that no proper subset of a key can also be a key

Of all the *candidate keys* for a relation, one is chosen to be the *primary key*

Example: $\{ sid \}$ is a key for Students, $\{ sid, name \}$ is a superkey for Students

Primary and Candidate Keys in SQL

Specify UNIQUE constraint for each *candidate key*

Specify PRIMARY KEY for the *primary key*

Example:

```
CREATE TABLE Students
(
    sid INTEGER,
    name TEXT,
    login VARCHAR(32),
    UNIQUE (sid, name),
    PRIMARY KEY (sid)
);
```

Primary and Candidate Keys in SQL (2)

Another example:

```
CREATE TABLE Enrolled
(
    sid INTEGER,
    cid VARCHAR(32),
    grade CHAR(1)
    PRIMARY KEY (sid, cid)
);
```

Q: What restrictions follow from using $\{ sid, cid \}$ as the primary key?

Foreign Keys, Referential Integrity

Foreign Key: a "logical pointer"

1. Set of fields in one relation that is used to "refer" to a tuple in another relation
2. Reference to *primary key* of the second relation

References are based on *primary keys*, since they uniquely identify tuples in relations

Foreign key associations can be enforced by integrity constraints, thereby ensuring *referential integrity*, i.e. no dangling references

Example: Enrolled.sid is a foreign key that refers to Students

Foreign Keys in SQL

Example:

```
CREATE TABLE Enrolled
(
    sid INTEGER,
    cid VARCHAR(32),
    grade CHAR(1)
    PRIMARY KEY (sid, cid),
    FOREIGN KEY (sid) REFERENCES Students(sid)
);
```

Enforcing Referential Integrity

Assume: Enrolled.sid is a foreign key referencing Students

Two ways for dangling tuples to arise:

1. Insertion or update to Enrolled introduces key not in Students
2. Deletion or update to Students leaves dangling tuples in Enrolled

Default action is to reject change.

(The only option for Enrolled is to reject the change)

Other options for changes to Students: CASCADE, SET NULL, SET DEFAULT

CASCADE vs. SET NULL vs. SET DEFAULT

SQL-92 and above support all 4 options on deletes and updates:

1. Default is NO ACTION (delete/update is rejected)
2. CASCADE: propagate action to affected tuples
3. SET NULL: sets foreign key of referencing tuple to NULL
4. SET DEFAULT: sets foreign key of referencing tuple to default value

Example: Delete a student, what happens to his enrollments?

1. CASCADE: Delete the enrollments as well
2. SET NULL: Set Enrolled.sid of affected enrollments as NULL
3. SET DEFAULT: Set Enrolled.sid of affected enrollments to default value

Referential Integrity in SQL

Example:

```
CREATE TABLE Enrolled
(
    sid INTEGER,
    cid VARCHAR(32),
    grade CHAR(1)
    PRIMARY KEY (sid, cid),
    FOREIGN KEY (sid) REFERENCES Students(sid)
        ON DELETE CASCADE
);
```

Views

A *view* is just a relation, but we store a *definition* rather than a set of tuples

Computed on the fly (or pre-computed if it is a *materialized view*)

Example:

```
CREATE VIEW Course_Info (cid,enrollment) AS
   SELECT cid, COUNT(*) as enrollment FROM Enrolled GROUP BY cid;
```

Describe View in PostgreSQL

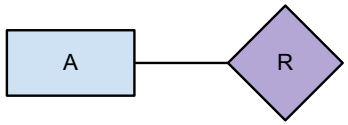
Example:

```
postgres=# \d Course_Info
           View "public.Course_Info"
   Column   |         Type          | Modifiers
------------+-----------------------+-----------
 cid        | character varying(32) |
 enrollment | bigint                |
View definition:
 SELECT Enrolled.cid, count(*) AS enrollment
   FROM Enrolled
  GROUP BY Enrolled.cid;
```

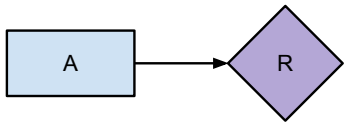
Part II: ER to Relational

Review of ER Constraints

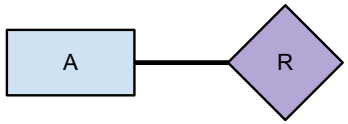
Entity may appear in any number of relations in R, including none:



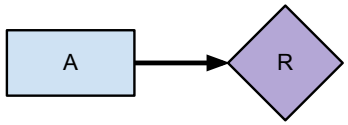
An entity in A may appear in at most one relation in R (key constraint):



Every entity in A must be a member of at least one relation in R (total constraint):



Every entity in A must be a member of exactly one relation in R (both):

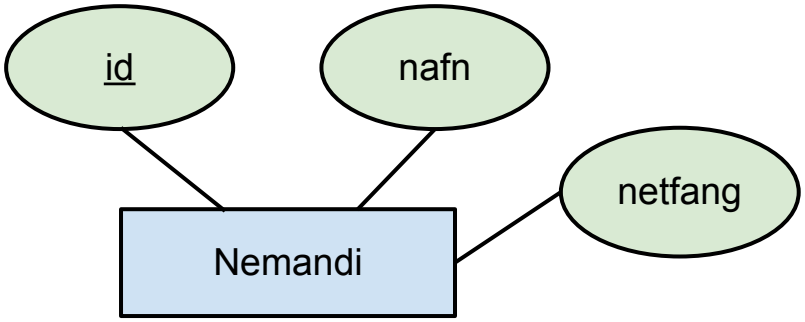


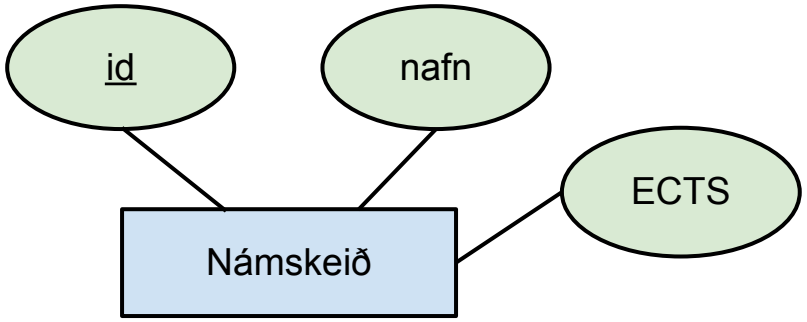
Convert ER diagram into tables

Rough algorithm:

1. Put all entities in their own table
2. Put reference on "many" in many-one relationship to the "one"
3. Put all many-many relationships into their own table

Step 1: Entity Sets to Tables

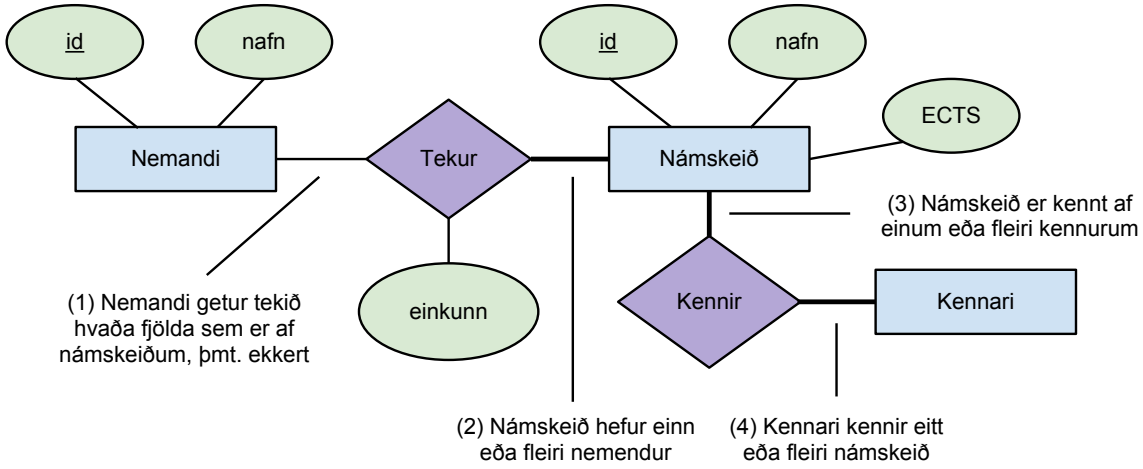





```
CREATE TABLE Nemandi
(
    id INTEGER,
    nafn TEXT,
    netfang VARCHAR(32),
    PRIMARY KEY (id)
);
```

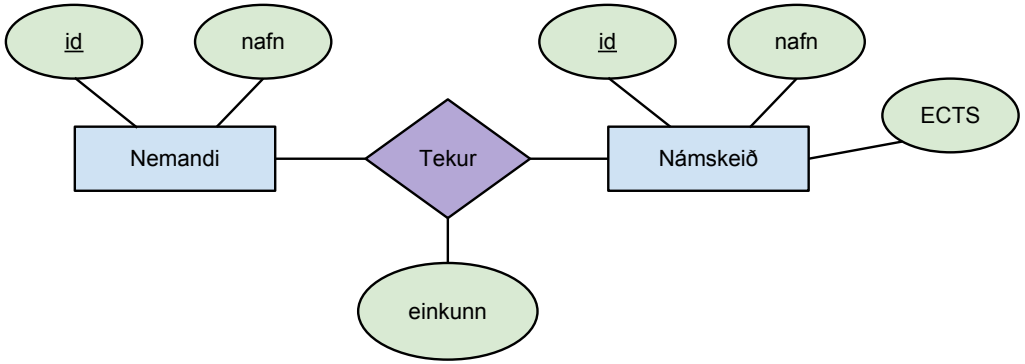
```
CREATE TABLE Namskeid
(
    id INTEGER,
    nafn TEXT,
    ECTS INTEGER,
    PRIMARY KEY (id)
);
```

Step 2: Many-One Relationships




```
CREATE TABLE Namskeid
(
    id INTEGER,
    nafn TEXT,
    ECTS INTEGER,
    kennari INTEGER,
    PRIMARY KEY (id),
    FOREIGN KEY (kennari) REFERENCES Kennari (id)
);
```

Step 3: Many-to-Many Relationships




```
CREATE TABLE Tekur
(
    nemandi INTEGER,
    namskeid INTEGER,
    einkunn FLOAT,
    PRIMARY KEY (nemandi, namskeid),
    FOREIGN KEY (nemandi) REFERENCES Nemandi (id),
    FOREIGN KEY (namskeid) REFERENCES Namskeid (id)
);
```