CS2102 Cheatsheet

DBMS Definitions

A *data model* is a collection of concepts for describing data.

A *schema* is a description of the structure of a database using a data model.

A *schema instance* is the content of a database at a particular time.

Relational Data Model

Data is modelled by *relations*, and each relation has a definition called a *relation schema*.

- Schema specifies attributes and data constraints
- Data constraints include domain constraints

A relation can be seen as a *table with rows and columns*

- Number of columns = *Degree/Arity*
- Number of rows = *Cardinality*

Each row is called a *tuple/record*. It has a *component* for each attribute of the relation.

A relation is thus a set of tuples and an instance of the *relation schema*, i.e. of a single table.

A *domain* is a set of atomic values, e.g. integers. All values for an attribute is either in this domain or *null*.

null is a special value that represents not applicable or unknown.

A *relational database schema* consists of a set of relation schemas and their data constraints, i.e. of multiple tables.

A *relational database* is an instance of the schema and is a collection of tables.

Integrity Constraints

Condition that restricts the data that can be stored in a database instance.

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

- *Domain constraints* restrict the attribute values of relations, e.g. only integers allowed
- A *key* is a *superkey* which is minimal, i.e. no proper subset of itself is a superkey
  - A *superkey* is a subset of attributes in a relation that unique identifies its tuples
  - Key attribute values cannot be null (*key constraints*)
  - A relation can have multiple keys, called *candidate keys*. One of these keys is selected as the *primary key*.
- A *foreign key* refers to the primary key of a second relation (which can be itself)
  - Each foreign key value must be the primary key value in the referenced relation or be null (*foreign key constraint*)
  - Also known as *referential integrity constraints*

Relational Algebra

A formal language for asking *queries* on relations.

A *query* is composed of a collection of operators called *relational operators*.

Relations are *closed* under relational operators.

Selection: $\boldsymbol{\sigma_c}$

$\sigma_c(R)$ selects tuples from relation R that satisfy the selection condition c.

Example: $\sigma_{price<20}(Sells)$

The *selection condition* is a boolean combination of *terms*.

A *term* is one of the following forms:

attribute **op** constant; attribute$_1$ **op** attribute$_2$; term$_1$ **and** term$_2$; term$_1$ **or** term$_2$; **not** term$_1$; (term$_1$)

- op $\in \{=, <>, <, \leq, >, \geq\}$.
- Operator precedence: (), op, not, and, or.

Result of a *comparison operation* involving a null value is *unknown*.

Result of an *arithmetic operation* involving a null value is *null*.

A tuple is only selected if the condition evaluates to *true* on it.

x	y	x AND y	x OR y	NOT x
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	TRUE	FALSE	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	TRUE	
TRUE	FALSE	FALSE	TRUE	
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE	

Table: Three-valued logic system

Projection: $\boldsymbol{\pi_l}$

$\pi_l(R)$ projects attributes given by a list l of attributes from relation R.

Example: $\sigma_{rname,pizza}(Sells)$

Duplicate records are removed in the output relation.

Renaming: $\boldsymbol{\rho_l}$

l is a list of attribute renamings of the form $a_1: b_1, \cdots, a_n: b_n$.

The order of the attribute renamings in l does not matter.

Set Operators

Union: $R \cup S$ returns a relation containing all tuples that occur in R or S

Intersection: $R \cap S$ returns a relation containing all tuples that occur in both R and S

Set-difference: $R - S$ returns a relation containing all tuples in R but not in S

These operators require input relations to be *union compatible*

- Same number of attributes, and
- Corresponding attributes have the same domains
- No need to use the same attribute names

The resultant schema will follow the schema of R above.

Cross-Product: $\times$

Given $R(A,B,C)$ and $S(X,Y)$, $R \times S$ gives us (A,B,C,X,Y)

$$R \times S = \{(a,b,c,x,y) | (a,b,c) \in R, (x,y) \in S\}$$

Also known as *cartesian product*

- When finding pairs of the same type, e.g. pairs of customers, we can add an additional selection of e.g. $C_1 < C_2$ to filter out duplicates.

Inner Join: $\boldsymbol{R \bowtie_c S}$

$$R \bowtie_c S = \sigma_c(R \times S)$$

Natural Join: $\boldsymbol{R \bowtie S}$

$$R \bowtie S = \pi_l(R \bowtie_c \rho_{a_1:b_1,\cdots,a_n:b_n}(S))$$

where

- $A = \{a_1, a_2, \cdots, a_n\}$ is the set of common attributes between R and S
- $c = (a_1 = b_1)$ and $\cdots$ and $(a_n = b_n)$
- l includes, in this order
  - List of attributes in R that are also in A
  - List of attributes in R that are not in A
  - List of attributes in S that are not in A

Dangling Tuples: $\boldsymbol{dangle(R \bowtie_c S)}$

Let $attr(R)$ be the list of attributes in the schema of R.

We say that $t \in R$ is a dangling tuple in R wrt $R \bowtie_c S$ if $t \notin \pi_{attr(R)}(R \bowtie_c S)$

Left Outer Join: $\boldsymbol{R \rightarrow_c S}$

Let $null(R)$ denote a tuple of null values of same arity as R

$$R \rightarrow_c S = (R \bowtie_c S) \cup (dangle(R \bowtie_c S) \times \{null(S)\})$$

Right Outer Join: $\boldsymbol{R \leftarrow_c S}$

$$R \leftarrow_c S = (R \bowtie_c S) \cup (\{null(R)\} \times dangle(S \bowtie_c R))$$

Full Outer Join: $\boldsymbol{R \leftrightarrow_c S}$

$$R \leftrightarrow_c S = (R \rightarrow_c S) \cup (\{null(R)\} \times dangle(S \bowtie_c R))$$

Natural Left Outer Join: $\boldsymbol{R \rightarrow S}$

Same schema as $R \bowtie S$

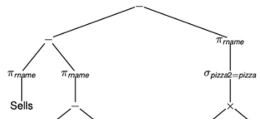
Natural Right Outer Join: $\boldsymbol{R \leftarrow S}$

Same schema as $R \bowtie S$

Natural Full Outer Join: $\boldsymbol{R \leftrightarrow S}$

Same schema as $R \bowtie S$

Operator Trees



Structured Query Language (SQL)

Domain-specific language designed for computations on relations. It is a declarative language. Contains two parts: *Data Definition Language* and *Data Manipulation Language*

Data Types

- boolean: true/false (null = unknown)
- integer: signed four byte integer
- float8: double-precision floating point number (8 bytes)
- numeric: arbitrary precision floating point number
- numeric(p, s): maximum total of p digits with maximum of s digits in fractional part
- char(n): fixed-length string consisting of n characters
- varchar(n): variable-length string up to n characters
- text: variable-length character string
- date: calendar date (year, month, day)
- timestamp: date and time

Create/Drop Table

```
create table Students (
    studentId    integer,
    name         varchar(100),
    birthDate    date
    dept         varchar(20) default 'CS'

);
drop table Students;
drop table if exists Students;
// deletes objects that depend on this table
drop table if exists Students cascade;
```

Is Null Predicate

```
x IS NULL
```

It evaluates to true for null values, else false

```
x IS NOT NULL = NOT (x IS NULL)
```

Is Distinct From Predicate

`x IS DISTINCT FROM y`

Equivalent to `x <> y` if both `x` and `y` are non-null, else is false if both are null, else true if only one is null

`x IS NOT DISTINCT FROM y = NOT(x is DISTINCT FROM y)`

not null Constraint

```
name         varchar(100) not null,
```

unique Constraint

```
studentId    integer unique,
```

or

```
unique (city, state) -- at bottom
```

primary key Constraint

```
studentId    integer primary key,
```

or

```
studentId    integer unique not null,
```

or

```
primary key (sid, cid) -- at bottom
```

foreign key Constraint

```
studentId    integer references Student (id),
```

or

```
foreign key (a, b) references Other (a, b) -- at bottom
```

If any of the referencing columns is null, a referencing row can escape satisfying the foreign key constraint. If we add `match full` to the end of the constraint, then a referencing row only escapes if all of the referencing columns are null.

check Constraints

```
check (day in (1,2,3,4,5)), -- besides day
check ((hour >= 8) and (hour <= 17)), -- at bottom
```

Constraint Names

```
bDate    date constraint bdate check (bdate is not null)
```

or

```
constraint obj_pri_key primary key (name,day,hour)
```

Insert

```
insert into Students
values (12345, 'Alice', '1999-12-25', 'Maths');
insert into Students (name, studentId)
values ('Bob', 67890), ('Carol', 11122);
```

Delete

```
delete from Students; -- deletes all
delete from Students where dept = 'Maths';
```

Update

```
update Accounts set balance = balance * 1.02;
update Accounts set  balance = balance + 500, name = 'Alice'
where accountId = 12345;
```

Foreign Key Constraints Violations

No Action: Rejects action if it violates constraint (default)
Restrict: Same as No Action but constraint checking is not deferred
Cascade: Propagates delete/update to referencing tuples
Set Null: Updates foreign keys to null
Set Default: Updates foreign keys to some default value. We will need to specify this value at the referencing column, and it must meet the foreign key constraints, else action will fail

```
foreign key (a, b) references Other (a, b)
            on delete cascade
```

Transactions

Starts with `begin;` and ends with `commit;` or `rollback;`. Can contain multiple SQL statements.

- *Atomicity*: Either all effects are reflected or none.
- *Consistency*: Executed in isolation, preserves the DB consistency
- *Isolation*: From the effects of other concurrent transactions
- *Durability*: Effects persist even if system failures occur

Deferrable Constraints

unique, primary key and foreign key constraints can be deferred using `deferrable initially deferred` or `deferrable initially immediate`

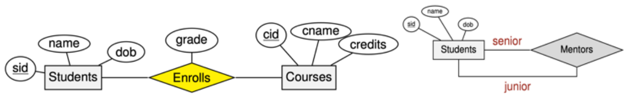
We can change this using `set constraints`, e.g. `set constraints fkey deferred;` or `set constraints fkey immediate;` (retroactive)

Modifying Schema

```
alter table Students alter column dept drop default;
alter table Students drop column dept;
alter table Students add column faculty varchar(20);
alter table Students add constraint fk_grade foreign key
(grade) references Grades;
```

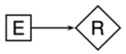
Entity-Relationship (ER) Model

Entity: Real-world object distinguishable from other objects
Attribute: Specific information describing an entity, represented by ovals
Entity set: Collection of similar entities, represented by rectangles
Key: Represented as underlined attributes
Relationship: Association among two or more entities
Relationship set: Collection of similar relationships, represented by diamonds. Attributes are used to describe information about relationships.

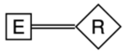


By default, these relationships are *many-to-many*.
Relationship role: Shown explicitly when one entity set appears two or more times in a relationship set
Degree: An n-ary relationship set involves n entity roles; degree $= n$.
$n = 2 \rightarrow$ binary, $n = 3 \rightarrow$ ternary.
Relationship keys: Each relationship set instance will have the primary keys of the entities as well as its own attributes. The primary key of the relationship set will contain those primary keys as well as a subset of its own attributes, which will be underlined.

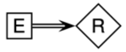
Relationship Constraints



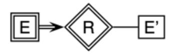
Key Constraint: Each instance of E can participate in *at most one* instance of R. Represented by an arrow. Allows for *one-to-many* if one entity has an constraint but the other doesn't, or *one-to-one* if both entities have the constraint.



Total Participation Constraint: Each instance of E must participate in *at least one* instance of R. Represented by a double line.
A single line is a *partial participation constraint*, i.e. 0 or more.



Key & Total Participation Constraint: Each instance of E participates in exactly one instance of R. Represented by double line arrow.

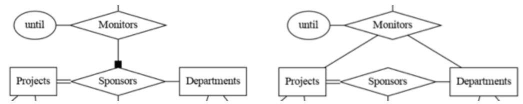


Weak Entity Set: E is a weak entity set with identifying owner E' & identifying relationship set R. E does not have its own key and requires the primary key of its *owner entity* to be uniquely identified. It must have a many-to-one relationship with E' **and** total participation in R.
Partial Key: Set of attributes of a weak entity set that uniquely identifies a weak entity for a given owner entity.

Database Design

We can represent primary key. Cannot represent unique, not null.
Without constraints, we generally represent relationship sets as tables with foreign keys that form part of its primary key, i.e. association class.
With constraints, we can choose to represent them as a separate table or as an attribute in an entity set's table (the one that's many in many-to-one, or any in an one-to-one).
Note that using a separate table for key & total participation constraint will not enforce that constraint by schema, while the latter method does.
If the same entity set participates twice in a relationship (i.e. relationship roles), we can have two self-referencing foreign keys, `firstId` and `secondId`, in the table and a `check (firstId <> secondId)`.
Weak entity set & its identifying relationship set can be represented as a single relation, which has a foreign key that deletes on cascade and forms part of its primary key.

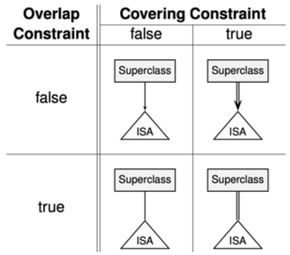
Aggregation



When a relation between two entities is treated as a single entity. The above two diagrams are not the same, since the right does not enforce a relationship between `Projects` and `Departments`.

ISA Hierarchies

We can classify an entity set into subclasses. Every entity in a subclass entity set is an entity in its superclass entity set



- *Overlap Constraint*: Can an entity belong to multiple subclasses?
- *Covering Constraint*: Does an entity in a superclass have to belong to some subclass?

We can create a relation per subclass/superclass, with the subclass' tables having a foreign key referencing the superclass' table, along with additional attributes.