CS2102

AY21/22 SEM 1

github/jovyntls

DBMS

Transactions

- **transaction, T** $\rightarrow$ a finite sequence of database operations
- 4 properties of a transaction: **ACID** properties

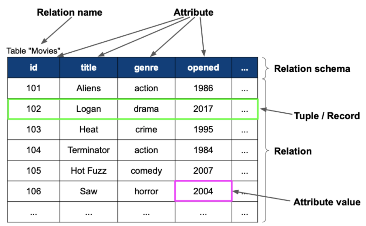
ACID properties

1. **Atomicity** $\rightarrow$ either all effects of T are reflected in the database, or none
2. **Consistency** $\rightarrow$ the execution of T guarantees to yield a *correct state* of the DB
3. **Isolation** $\rightarrow$ execution of T is *isolated* from the effects of concurrent transactions
4. **Durability** $\rightarrow$ after the commit of T, its effects are *permanent* in case of failures

Serializability

- Requirement for Concurrent Execution: **serializable transaction execution**
  - concurrency: to optimise performance
  - serializability: to ensure integrity of data
- (concurrent execution of a set of transactions is) **serializable** $\rightarrow$ execution is equivalent to some serial execution of the same set of transactions
  - **equivalent** $\rightarrow$ they have the same *effect* on the data

RELATIONAL MODEL



- **relation schema** $\rightarrow$ defines a relation
  - specifies **attributes** and data constraints
  - $R(A_1, A_2, \ldots, A_n)$: relation schema with name R and n attributes $A_1, A_2, \ldots, A_n$
- **relational database schema** $\rightarrow$ set of relation schemas + data constraints
  - TableName(col_1, col_2, col_3) with dom(col_1) = {x, y, z}
- **domain** $\rightarrow$ a set of *atomic* values
  - $dom(A_i)$ = set of possible values for A_i
    - e.g. dom(course) = {cs2102, cs2030, cs2040}
  - for all value v of attribute $A_i, v \in \{dom(A_i) \cup \{null\}\}$
- **relation** $\rightarrow$ a set of *tuples*
  - each instance of schema R is a relation which is a subset of $\{(a_1, a_2, \ldots, a_n) \mid a_i \in dom(A_i) \cup \{null\}\}$
- **integrity constraint** $\rightarrow$ condition that restricts what constitutes valid data
- **structural** $\rightarrow$ (integrity constraint) inherent to the data model

Key Constraints

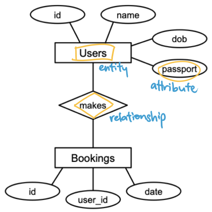
- **superkey** $\rightarrow$ subset of attributes that *uniquely* identifies a tuple in a relation
- **key** $\rightarrow$ superkey that is also **minimal** - no proper subset of the key is a superkey
- **candidate keys** $\rightarrow$ set of all keys for a relation
- **primary key** $\rightarrow$ selected candidate key; cannot be `null`
  - **prime attributes** $\rightarrow$ attributes of the primary key

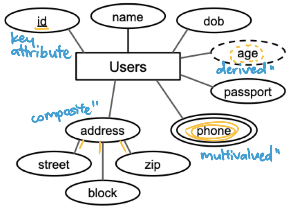
CONSTRAINTS

- **Not-Null Constraints** violation: $\exists t \in$ Employees where `t.id IS NOT NULL` evaluates to **false**
- **Unique Constraints** violation: For any 2 tuples $t_i, t_k \in R$, $(t_i \cdot A <> t_k \cdot A)$ or $(t_i \cdot B <> t_k \cdot B)$ evaluates to **false**
  - ! `null` rows will NOT violate unique key constraints
- **Primary Key Constraints**: prime attributes cannot be null
  - (entity integrity constraint)
- **Foreign Key Constraints**: each FK in the referencing relation *must*:
  - appear as a **PK** in the referenced relation, OR
  - be a `null` value
- `R.sid` $\rightarrow$ `S.id`: `R.sid` is a FK referencing PK `id` in S

ENTITY RELATIONSHIP MODEL

- **entity set** $\rightarrow$ collection of entities of the same type
- **attribute** $\rightarrow$ specific information describing an entity
  - **key attribute** $\rightarrow$ uniquely identifies each entity
  - **composite attribute** $\rightarrow$ composed of multiple other attributes
  - **multivalued attribute** $\rightarrow$ may comprise more than one value for a given entity
  - **derived attribute** $\rightarrow$ derived from other attributes



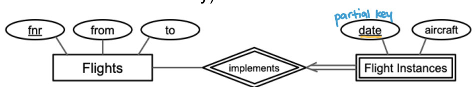


Relationship Sets

- **degree** $\rightarrow$ no. of entity roles participating in a relationship
  - an n-ary relationship set involves n entity roles (where n is the degree of the relationship set)

Dependency Constraints

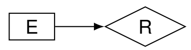
- **weak entity sets** $\rightarrow$ entity set that does not have its own key
  - can only be uniquely identified through the primary key of its **owner entity**
  - **partial key** $\rightarrow$ set of attributes that uniquely identifies a weak entity for a given owner entity (identifies the exact instance of a weak entity)



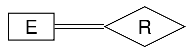
- requirements
  1. many-to-one relationship (identifying relationship) from weak entity set to owner entity set
  2. weak entity set must have **total participation** in identifying relationship

Participation Constraints

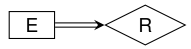
- **partial participation constraint** $\rightarrow$ participation (of an entity in a relationship) is not mandatory (0 or more)
- **total participation constraint** $\rightarrow$ participation is mandatory (1 or more)



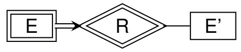
each instance of E participates in **at most one** instance of R



each instance of E participates in **at least one** instance of R

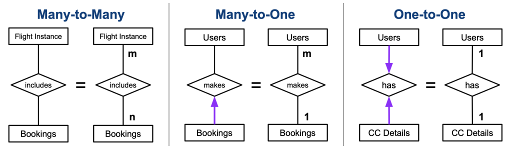


each instance of E participates in **exactly one** instance of R

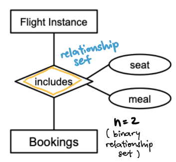


E is a **weak entity set** with identifying owner E' & identifying relationship set R.

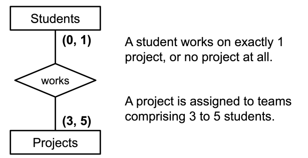
Cardinality Constraints



Alternative Representations

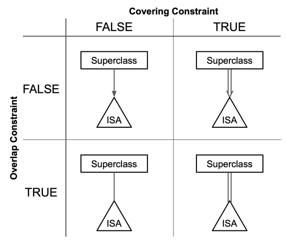


Min/Max notation



ISA Hierarchy Constraints

- **overlap contraint** $\rightarrow$ a superclass entity can belong to **multiple** subclasses
- **covering constraint** $\rightarrow$ a superclass entity **has to** belong to a subclass



superclass belongs to:

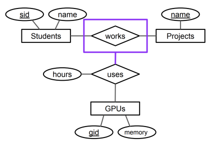
	F	T
F	at most one	exactly one
T	any number of	at least one

... subclass

Aggregation

- abstraction that treats **relationships** as higher-level entities
  - e.g. treating 2 entities + 1 relationship as an entity set

```
CREATE TABLE Uses (
  gid INTEGER,
  sid CHAR(20),
  pname VARCHAR(50),
  hours NUMERIC,
  PRIMARY KEY (gid, sid, pname),
  FOREIGN KEY (gid) REFERENCES GPUs (gid),
  FOREIGN KEY (sid, pname) REFERENCES
        works (sid, pname)
);
```



SQL

- FROM $\rightarrow$ WHERE $\rightarrow$ GROUP BY $\rightarrow$ HAVING $\rightarrow$ SELECT $\rightarrow$ ORDER BY $\rightarrow$ LIMIT/OFFSET
- `_` - any single character; `%` - any sequence of characters (*)
- `UNION` / `INTERSECT` / `EXCEPT` : removes duplicate tuples
- `UNION ALL` / `... ALL` : keeps duplicates
- if column A_i or table R appears in the `SELECT` / `HAVING` clause, one of the following conditions must hold:
  1. A_i appears in the GROUP BY clause
  2. A_i appears as input of an aggregation function in the SELECT clause
  3. the primary key of R appears in the GROUP BY clause
- `NULLIF(val1, val2)` returns val1 == val2 ? `null` : `val1`

Handling NULLs

- **comparison** operation with `null` $\Rightarrow$ *unknown*
- **arithmetic** operation with `null` $\Rightarrow$ `null`
- `null = null` $\Rightarrow$ *unknown* (null, neither true nor false)
- `null = some_value` $\Rightarrow$ neither true nor false

x	y	x<>y	x IS DISTINCT FROM y	x IS NULL
1	1	FALSE	FALSE	FALSE
1	2	TRUE	TRUE	FALSE
null	1	*null*	TRUE	TRUE
null	*null*	*null*	FALSE	TRUE

Query	Interpretation
SELECT MIN(A) **FROM** R;	Minimum non-null value in A
SELECT MAX(A) **FROM** R;	Maximum non-null value in A
SELECT AVG(A) **FROM** R;	Average of non-null values in A
SELECT SUM(A) **FROM** R;	Sum of non-null values in A
SELECT COUNT(A) **FROM** R;	Count of non-null values in A
SELECT COUNT(*) **FROM** R;	Count of rows in R
SELECT AVG(**DISTINCT** A) **FROM** R;	Average of distinct non-null values in A
SELECT SUM(**DISTINCT** A) **FROM** R;	Sum of distinct non-null values in A
SELECT COUNT(**DISTINCT** A) **FROM** R;	Count of distinct non-null values in A

Let R be an empty relation; let S be a non-empty relation with n tuples but ONLY null values for A.

Query	Result
SELECT MIN(A) **FROM** R;	null
SELECT MAX(A) **FROM** R;	null
SELECT AVG(A) **FROM** R;	null
SELECT SUM(A) **FROM** R;	null
SELECT COUNT(A) **FROM** R;	0
SELECT COUNT(*) **FROM** R;	0

Query	Result
SELECT MIN(A) **FROM** S;	null
SELECT MAX(A) **FROM** S;	null
SELECT AVG(A) **FROM** S;	null
SELECT SUM(A) **FROM** S;	null
SELECT COUNT(A) **FROM** S;	0
SELECT COUNT(*) **FROM** S;	***n***

RELATIONAL ALGEBRA

UNARY OPERATORS

Selection, σ_c

- $\sigma_c(R)$ $\rightarrow$ select all tuples from R satisfying condition c.
  - $\forall$ tuple $t \in R,\ t \in \sigma_c(R) \iff c$ evaluates true on t
  - input and output relation have the same schema
- **selection condition** $\rightarrow$
  - a *boolean expression* of one of the following forms:
    - constant selection - attribute **op** constant
    - attribute selection - attribute$_1$ **op** attribute$_2$
    - expr$_1 \wedge$ expr$_2$; expr$_1 \vee$ expr$_2$; $\neg$ expr ; (expr)
  - with **op** $\in \{=, <>, <, \leq, \geq, >\}$
    - **operator precedence**: (), **op**, $\neg$, $\wedge$, $\vee$

Projection, π_ℓ

- $\pi_\ell(R)$ $\rightarrow$ projects all attributes of a given **relation** specified in list ℓ (duplicates removed from output relation)
  - **order** of attributes matters!

Renaming, ρ_ℓ

- $\rho_\ell(R) \rightarrow$ renames the attributes of a relation R (with schema $R(A_1, A_2, \ldots, A_n)$)
- 2 possible formats for ℓ
  - ℓ is the new *schema* in terms of the new attribute names
    - $\ell = (B_1, B_2, \ldots, B_n)$
    - $B_i = A_i$ if attribute A_i does not get renamed
  - ℓ is a list of attribute renamings of the form: $B_i \leftarrow A_i, \ldots, B_k \leftarrow A_k$
    - each $B_j \leftarrow A_j$ renames attribute A_j to attribute B_j
    - order of renaming doesn't matter

SET OPERATORS

- **union** $\rightarrow R \cup S$ returns a relation w/ all tuples in both R **or** S
- **intersection** $\rightarrow R \cap S$ ... all tuples in both R **and** S
- **set difference** $\rightarrow R - S$ ... all the tuples in R **but not in** S

! for all set operators: R and S must be **union-compatible**

Union Compatibility

- two relations R and S are **union-compatible** $\rightarrow$ if
  - R and S have the same number of attributes; and
  - corresponding attributes have the *same or compatible domains*
- note: R and S do not have to use the same attribute names

CROSS PRODUCT

- **cross product** $\rightarrow$ given two relations $R(A, B, C)$ and $S(X, Y)$, $R \times S$ returns a relation with schema (A, B, C, X, Y) defined as $R \times S = \{(a, b, c, x, y) \mid (a, b, c) \in R, (x, y) \in S\}$
- **size** of cross product $= |R| * |S|$

JOIN OPERATORS

Inner Joins

- eliminate all tuples that do not satisfy a matching criteria (i.e. **attribute selection**)
- θ**-join** $\rightarrow$ (of two relations R and S) $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- **Equi Join** $\bowtie$ $\rightarrow$ special case of θ-join defined over the **equality** operator ($=$) only
- **Natural Join** $\bowtie$ $\rightarrow$ performed over all attributes R and S have in common
  - the natural join (of two relations R and S) is defined as $R \bowtie S = \pi_\ell(R \bowtie_c \rho_{b_i \leftarrow a_i, \ldots, b_k \leftarrow a_k}(S))$
    - $A = \{a_i, \ldots, a_k\} =$ the set of attributes that R and S have in common
    - $c = ((a_i = b_i) \wedge \cdots \wedge (a_k = b_k))$
    - $\ell =$ list of all attributes of R + list of all attributes in S that are **not in** A
  - output relation contains the common attributes of R and S only *once*

Outer Joins

- **dangling tuples** $\rightarrow$ tuples in R or S that do not match with tuples in the other relation
  - ***dangle***$(R \bowtie_\theta S) \rightarrow$ set of dangling tuples in R wrt to $R \bowtie_\theta S$ (missing attribute values are padded with `null`)
    - *dangle*$(R \bowtie_\theta S) \subseteq R$
  - always removed by inner joins, kept by outer joins

- $null(R) \rightarrow n$-component **tuple** of `null` values where n is the number of attributes of R

Definitions

- **left outer join** $\rightarrow R ⟕_\theta S$ $= R \bowtie_\theta S \cup (\textit{dangle}(R \bowtie_\theta S) \times \{\textit{null}(S)\})$
- **right outer join** $\rightarrow R ⟖_\theta S$ $= R \bowtie_\theta S \cup (\{\textit{null}(R)\} \times \textit{dangle}(S \bowtie_\theta R))$
- **full outer join** $\rightarrow R ⟗_\theta S$ $= R \bowtie_\theta S \cup (\textit{dangle}(R \bowtie_\theta S) \times \{\textit{null}(S)\})$ $\cup (\{\textit{null}(R)\} \times \textit{dangle}(S \bowtie_\theta R))$

Natural Outer Joins

- natural left/right/full outer join: $R ⟕ S$ / $R ⟖ S$ / $R ⟗ S$
- output relation contains the common attributes only once

FUNCTIONAL DEPENDENCIES

- **normal form** $\rightarrow$ a definition of minimum requirements in terms of **redundancy**
- **prime attribute** $\rightarrow$ appears in at least one key
- an attribute not in any RHS of any FD **must** be in every key

Functional Dependencies

Let $A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n$ be some attributes.

- **uniquely identifies** $\rightarrow$ $\{A_1 A_2 \ldots A_m\} \rightarrow \{B_1 B_2 \ldots B_n\}$ whenever 2 tuples have the same values on $A_1 A_2 \ldots A_m$, they always have the same values on $B_1 B_2 \ldots B_n$
  - $\{A\} \rightarrow \{B\}$: functional dependency - A determines B
- two **attributes** are **functionally equivalent** $\rightarrow$ if either one can determine the other
- **dependency preserving** $\rightarrow$ can derive all FDs from table closures
- **equivalence** $\rightarrow$ F1 is equivalent to F2 ($F1 \equiv F2$) $\Leftrightarrow$
  - $F2 \vdash F1$: every FD in F1 can be derived from F2
  - $F1 \dashv F2$: every FD in F2 can be derived from F1

Armstrong's Axioms

1. axiom of **reflexivity**: set $\rightarrow$ a subset of attributes $(\{A, B\} \rightarrow \{A\})$
2. axiom of **augmentation**: if $\{A\} \rightarrow \{B\}$, then $\forall C$, $\{AC\} \rightarrow \{BC\}$
3. axiom of **transitivity**: if $\{A\} \rightarrow \{B\}$ and $\{B\} \rightarrow \{C\}$, then $\{A\} \rightarrow \{C\}$

Extended Armstrong's Axioms

- rule of **decomposition**: if $\{A\} \rightarrow \{BC\}$ then $\{A\} \rightarrow \{B\} \wedge \{A\} \rightarrow \{C\}$
- rule of **union**: if $\{A\} \rightarrow \{B\} \wedge \{A\} \rightarrow \{C\}$, then $\{A\} \rightarrow \{BC\}$
- combined: $\{A\} \rightarrow \{BC\} \Leftrightarrow \{A\} \rightarrow \{B\} \wedge \{A\} \rightarrow \{C\}$

Closures

- $\{A_1 A_2 \ldots A_m\}^+$ is the closure of $A_1 A_2 \ldots A_m$

BOYCE-CODD NF (BCNF)

BCNF

- **BCNF** $\rightarrow$ every *non-trivial & decomposed FD* has a *superkey* as its LHS
  - stronger than 3NF - has fewer redundancies
- ! a table with exactly one or two attributes is always in BCNF!
- ✓ no update/deletion/insertion anomalies
- ✓ small redundancies
- ✓ **lossless join** - original table can always be reconstructed from decomposed tables (natural join). $\Rightarrow \quad R = R1 \bowtie R2 = \pi_{R1}(R) \bowtie \pi_{R2}(R)$
- × may **not** preserve all FDs - decomposed table may have no non-trivial & decomposed FDs
  - exists FD that cannot be derived from FDs on R1 and R2
- **lossless decomposition** $\rightarrow \{R_1, R_2\}$ is lossless if $R_1 \cap R_2$ is a superkey of R_1 or R_2
  - $R_1 \cap R_2$ uniquely identifies all the attributes in R_1 or R_2
  - closure of $R_1 \cap R_2 = R_1$ or R_2

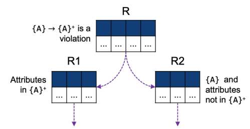
Non-Trivial and Decomposed FD

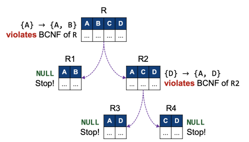
- **non-trivial** $\rightarrow \{A\} \rightarrow \{B\}$ where $\{A\} \not\subseteq \{B\}$
- **decomposed** $\rightarrow \{A\} \rightarrow \{B\}$ where B is a single attribute

BCNF Normalisation

- for a BCNF-violating FD $\{A\} \rightarrow \{A\}^+$, create tables
  - $R1(\{A\}^+)$ containing the superkey, and
  - $R2\left(\{A\} \cup (R - \{A\}^+)\right)$
- if table does not contain all attributes:
  1. compute closure of each subset of the table's attributes
  2. remove RHS attributes not in the table

! *implicit* functional dependencies should be checked too! (because explicit FDs may not apply to R2 when R2 is missing attributes)





e.g. $R(A, B, C, D)$ with $BC \rightarrow D$, $D \rightarrow A$, $A \rightarrow B$

THIRD NORMAL FORM (3NF)

3NF

- a table is in **3NF** $\rightarrow$ if every *non-trivial* & *decomposed* FD:
  - its LHS is a **superkey**, OR
  - its RHS is a **prime attribute** (any attribute in any key)
- ✓ will preserve all FDs
- relaxed form of BCNF
  - satisfies BCNF $\Rightarrow$ satisfies 3NF
  - violates 3NF $\Rightarrow$ violates BCNF
- if all attributes are prime attributes, the table is in 3NF

3NF Synthesis

for table R and a set of FDs F,

1. derive *minimal basis* F_b of F
2. from the minimal basis, combine (union) the FDs for which the LHS is the same (**canonical cover**)
3. create a table for each FDs remained in the minimal basis after union
4. if **none** of the tables contain a key for the original table R, create a table containing a key of R

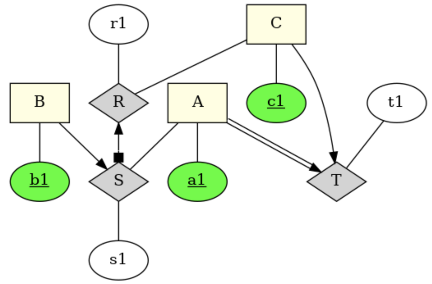
Minimal Basis / Minimal Cover

- minimal basis, F_b: **simplified** $\rightarrow$ 4 conditions
  1. equivalence: $F \equiv F_b$ (every FD in F_b can be derived from F and vice versa)
  2. every FD in F_b is non-trivial and decomposed
  3. $\forall$ FD in F_b, none of the LHS attributes are redundant
  4. no FD in F_b is redundant
- **redundant** $\rightarrow$ can be removed without affecting the original FD (i.e. $F \equiv F_{b*}$ where F_{b*} is formed by removing the attribute)

to obtain a minimal basis

1. ensure equivalence
2. transform FDs to non-trivial and decomposed
3. remove redundant attributes
   - 3.1. for an FD {A} $\rightarrow$ B for a **set** of attributes A, for an attribute C in A,
   - 3.2. compute $\{A - C\}^+$ using F
   - 3.3. if $B \in \{A - C\}^+$, then we can remove C
4. remove redundant FDs
   - 4.1. try removing and check for equivalence

COMMON ER MODELS



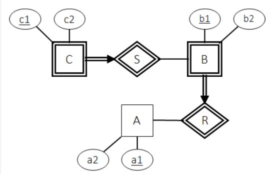

```sql
-- merge A and T
CREATE TABLE B (
  b1     integer,
  PRIMARY KEY (b1)
);

CREATE TABLE C (
  c1     integer,
  PRIMARY KEY (c1)
);

CREATE TABLE AT (
  a1     integer,
  c1     integer NOT NULL,
  t1     integer NOT NULL,
  PRIMARY KEY (a1),
  UNIQUE (c1),
  FOREIGN KEY (c1) REFERENCES C
);

CREATE TABLE S (
  a1     integer NOT NULL,
  b1     integer,
  s1     integer NOT NULL,
  PRIMARY KEY (b1),
  FOREIGN KEY (a1) REFERENCES AT,
  FOREIGN KEY (b1) REFERENCES B
);

CREATE TABLE R (
  b1     integer,
  c1     integer NOT NULL,
  r1     integer NOT NULL,
  PRIMARY KEY (b1),
  FOREIGN KEY (c1) REFERENCES C,
  FOREIGN KEY (b1) REFERENCES S
);
```




```sql
DROP TABLE IF EXISTS A, B, C CASCADE;
CREATE TABLE A (
  a1 INTEGER PRIMARY KEY,
  a2 INTEGER
);

CREATE TABLE B (
  a1 INTEGER REFERENCES A ON DELETE
        CASCADE,
  b1 INTEGER,
  b2 INTEGER,
  PRIMARY KEY (a1, b1)
);

CREATE TABLE C (
  a1 INTEGER,
  b1 INTEGER,
  c1 INTEGER,
  c2 INTEGER,
  PRIMARY KEY (a1, b1, c1),
  FOREIGN KEY (a1, b1) REFERENCES B ON
        DELETE CASCADE
);
```