

DBMS

Transactions

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

ACID properties

- Atomicity** \rightarrow either all effects of T are reflected in the database, or none
- Consistency** \rightarrow the execution of T guarantees to yield a *correct state* of the DB
- Isolation** \rightarrow execution of T is *isolated* from the effects of concurrent transactions
- 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 name: Movies

id	title	genre	opened	...
101	Aliens	action	1986	...
102	Logan	drama	2017	...
103	Heat	crime	1995	...
104	Terminator	action	1984	...
105	Hot Fuzz	comedy	2007	...
106	Saw	horror	2004	...
...

Relation schema: Table "Movies"

Tuple / Record: (101, Aliens, action, 1986, ...)

Relation: Set of tuples

Attribute value: 1986

- relation schema** \rightarrow defines a relation
 - specifies **attributes** and data constraints
- $R(A_1, A_2, \dots, A_n)$: relation schema with name R and n attributes A_1, A_2, \dots, A_n
- relational database schema** \rightarrow set of relation schemas + data constraints
 - TableName(col_1, col_2, col_3) with $\text{dom}(\text{col}_1) = \{x, y, z\}$
- domain** \rightarrow a set of *atomic* values
 - $\text{dom}(A_i) =$ set of possible values for A_i
 - e.g. $\text{dom}(\text{course}) = \{\text{cs2102}, \text{cs2030}, \text{cs2040}\}$
 - for all value v of attribute $A_i, v \in \{\text{dom}(A_i) \cup \{\text{null}\}\}$
- relation** \rightarrow a set of *tuples*
 - each instance of schema R is a relation which is a subset of $\{(a_1, a_2, \dots, a_n) \mid a_i \in \text{dom}(A_i) \cup \{\text{null}\}\}$
- 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

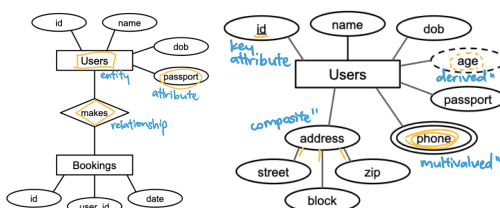
- key** \rightarrow superkey that is also **minimal** - no proper subset of the key is a superkey
- candidate keys** \rightarrow set of all keys for a relation
- primary key** \rightarrow selected candidate key; cannot be **null**
 - prime attributes** \rightarrow attributes of the primary key

CONSTRAINTS

- Not-Null Constraints** violation: $\exists t \in \text{Employees}$ where $t.\text{id}$ 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
- $R.\text{sid} \rightarrow S.\text{id}$: $R.\text{sid}$ 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

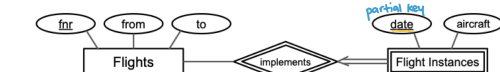


Relationship Sets

- 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

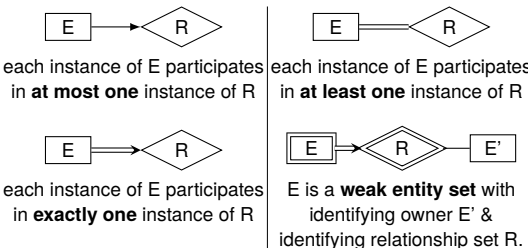
- weak entity sets** \rightarrow entity set that does not have its own key
 - can only be uniquely identified through the primary key of its **owner entity**
- partial key** \rightarrow set of attributes that uniquely identifies a weak entity for a given owner entity (identifies the exact instance of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



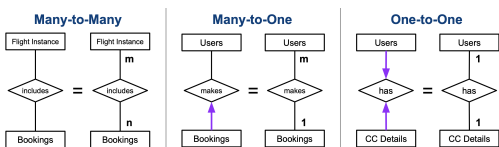
- requirements
 - many-to-one relationship (identifying relationship) from weak entity set to owner entity set
 - weak entity set must have **total participation** in identifying relationship

Participation Constraints

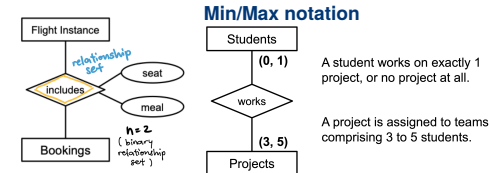
- partial participation constraint** \rightarrow participation (of an entity in a relationship) is not mandatory (0 or more)
- total participation constraint** \rightarrow participation is mandatory (1 or more)



Cardinality Constraints

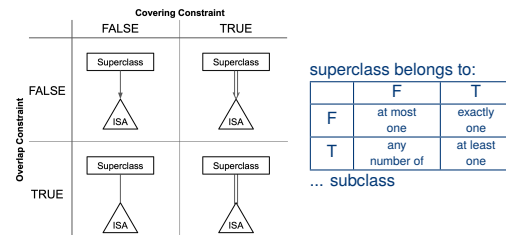


Alternative Representations



ISA Hierarchy Constraints

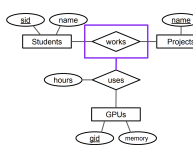
- overlap constraint** \rightarrow a superclass entity can belong to multiple subclasses
- covering constraint** \rightarrow a superclass entity **has** to belong to a subclass



Aggregation

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

```
CREATE TABLE Users (
  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:
 - A_i appears in the GROUP BY clause
 - A_i appears as input of an aggregation function in the SELECT clause
 - the primary key of R appears in the GROUP BY clause
- NULLIF(val1, val2)** returns $\text{val1} == \text{val2} ? \text{null} : \text{val1}$

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	Query	Result
SELECT MIN(A) FROM R;	null	SELECT MIN(A) FROM S;	null
SELECT MAX(A) FROM R;	null	SELECT MAX(A) FROM S;	null
SELECT AVG(A) FROM R;	null	SELECT AVG(A) FROM S;	null
SELECT SUM(A) FROM R;	null	SELECT SUM(A) FROM S;	null
SELECT COUNT(A) FROM R;	0	SELECT COUNT(A) FROM S;	0
SELECT COUNT(*) FROM R;	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 - attribute₁ **op** attribute₂
 - expr₁ **AND** expr₂; expr₁ **OR** expr₂; \neg expr; (expr)
 - with **op** $\in \{=, <>, <, \leq, \geq, >\}$
 - operator precedence**: (), **op**, \neg , **AND**, **OR**

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 $R(A_1, A_2, \dots, A_n)$)
- 2 possible formats for ℓ
 - ℓ is the new *schema* in terms of the new attribute names
 - $\ell = (B_1, B_2, \dots, B_n)$
 - $B_i = A_i$ if attribute A_i does not get renamed
 - ℓ is a list of attribute renamings of the form:
 - $B_i \leftarrow A_i, \dots, B_k \leftarrow A_k$
 - 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 $R \bowtie S = \pi_\ell(R \bowtie_c \rho_{b_i \leftarrow a_i, \dots, b_k \leftarrow a_k}(S))$
 - $A = \{a_i, \dots, a_k\}$ = the set of attributes that R and S have in common
 - $c = ((a_i = b_i) \wedge \dots \wedge (a_k = b_k))$
 - ℓ = 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** $\rightarrow R \bowtie_{\Leftarrow} S$
 $= R \bowtie_\theta S \cup (dangle(R \bowtie_\theta S) \times \{null(S)\})$
- right outer join** $\rightarrow R \bowtie_{\rightarrow} S$
 $= R \bowtie_\theta S \cup (\{null(R)\} \times dangle(S \bowtie_\theta R))$
- full outer join** $\rightarrow R \bowtie_{\Leftarrow \rightarrow} S$
 $= R \bowtie_\theta S \cup (dangle(R \bowtie_\theta S) \times \{null(S)\}) \cup (\{null(R)\} \times dangle(S \bowtie_\theta R))$

Natural Outer Joins

- natural left/right/full outer join: $R \bowtie_{\Leftarrow} S / R \bowtie_{\rightarrow} S / R \bowtie_{\Leftarrow \rightarrow} S$
- 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 $A_1, A_2, \dots, A_m, B_1, B_2, \dots, B_n$ be some attributes.
- uniquely identifies** $\rightarrow \{A_1 A_2 \dots A_m\} \rightarrow \{B_1 B_2 \dots B_n\}$ whenever 2 tuples have the same values on $A_1 A_2 \dots A_m$, they always have the same values on $B_1 B_2 \dots B_n$
 - $\{A\} \rightarrow \{B\}$: 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** $\rightarrow F_1$ is equivalent to F_2 ($F_1 \equiv F_2$) \Leftrightarrow
 - $F_2 \vdash F_1$: every FD in F_1 can be derived from F_2
 - $F_1 \vdash F_2$: every FD in F_2 can be derived from F_1

Armstrong's Axioms

- axiom of **reflexivity**: set \rightarrow a subset of attributes ($\{A, B\} \rightarrow \{A\}$)
- axiom of **augmentation**:
if $\{A\} \rightarrow \{B\}$, then $\forall C, \{AC\} \rightarrow \{BC\}$
- 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: $\{A\} \rightarrow \{B\} \rightarrow \{BC\} \Leftrightarrow \{A\} \rightarrow \{B\} \wedge \{A\} \rightarrow \{C\}$

Closures

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

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!
- \checkmark no update/deletion/insertion anomalies
- \checkmark small redundancies
- \checkmark **lossless join** - original table can always be reconstructed from decomposed tables (natural join).
 $\Rightarrow R = R_1 \bowtie R_2 = \pi_{R_1}(R) \bowtie \pi_{R_2}(R)$
- \times 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

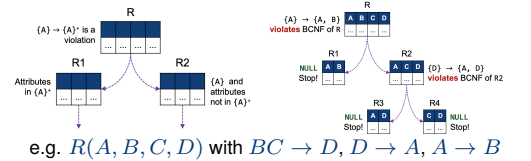
Non-Trivial and Decomposed FD

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

BCNF Normalisation

- for a BCNF-violating FD $\{A\} \rightarrow \{A\}^+$, create tables
 - $R_1(\{A\}^+)$ containing the superkey, and
 - $R_2(\{A\} \cup (R - \{A\}^+))$
- if table does not contain all attributes:
 - compute closure of each subset of the table's attributes
 - 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)



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)
- \checkmark 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,
- derive *minimal basis* F_b of F
 - from the minimal basis, combine (union) the FDs for which the LHS is the same (**canonical cover**)
 - create a table for each FDs remained in the minimal basis after union
 - 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
 - equivalence: $F \equiv F_b$ (every FD in F_b can be derived from F and vice versa)
 - every FD in F_b is non-trivial and decomposed
 - \forall FD in F_b , none of the LHS attributes are redundant
 - 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

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

COMMON ER MODELS

