Relational Data Modeling & performance issues

Vu Tuyet Trinh
trinhvt@soict.hust.edu.vn

Department of Information Systems
SoICT-HUST

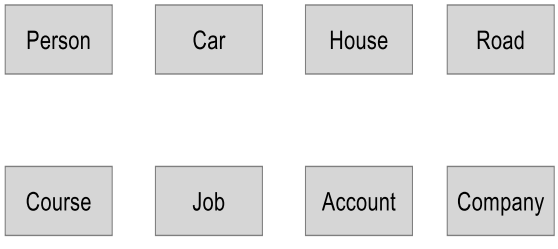
Outline

- Design
  - Extended Entity Relationship
    - Top Down
    - Conceptual/Abstract View
  - Functional Dependencies
    - Bottom Up
    - Synthesise relations
- Schema tuning
  - What
  - Why
  - How

Entity

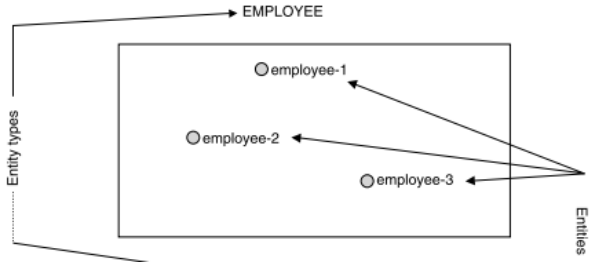
- Definitions of Entity
  - A thing that can be distinctively identified
  - An object with a physical existence or may be an object with conceptual existence
  - A thing or object in the real world that is distinguishable from all other objects
  - Some item in the real world that we wish to track

Entity - Example



Entity types

- Entities are an individual things
  - A car , a particular person
- Entity Types : Group of entities of the same properties



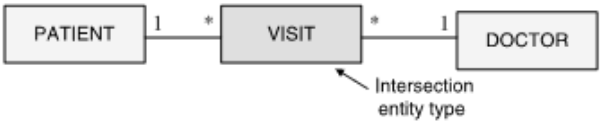
Weak/ Strong Entity Type

- Strong Entity Type
  - Entity occurrence that can exist in the database on their own form
- Weak Entity Type
  - Entity that cannot exist by themselves in the database

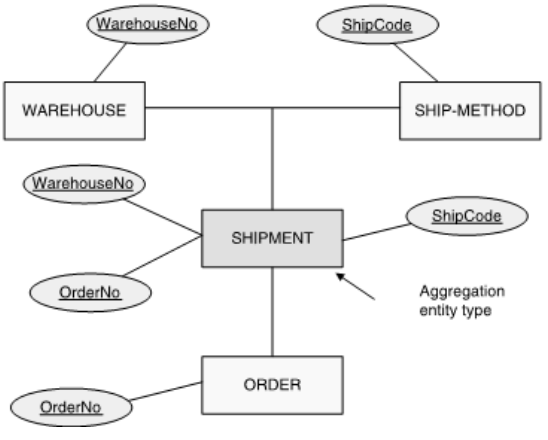
Weak Entity Type	Related Strong Entity Type
ORDER-DETAIL	ORDER
INVOICE-DETAIL	INVOICE
STATEMENT-ITEM	VENDOR-STATEMENT
ORDER	CUSTOMER
EMPLOYEE-DEPENDENT	EMPLOYEE

Intersection Entity Type

Many-to-many relationship

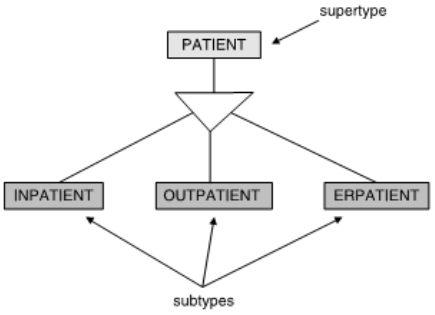


Aggregation Entity Type

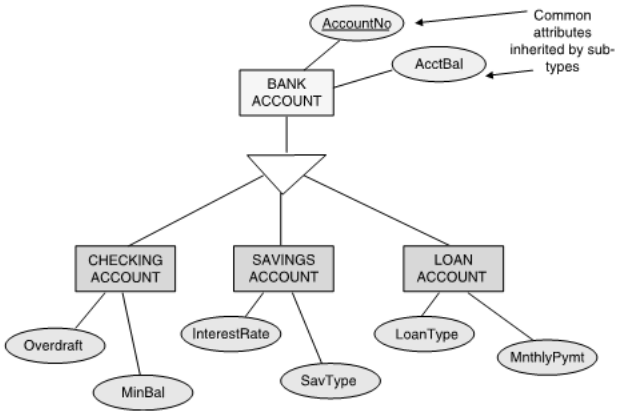


Generalization - Specialization

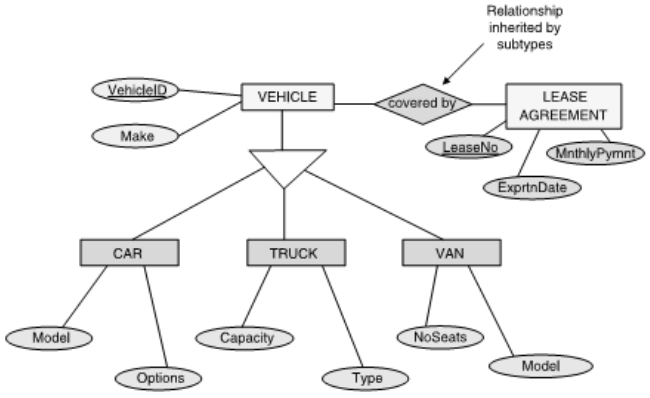
- Subtype entity inherits all properties of the super-type entity but also might have their own properties



Inheritance of attributes

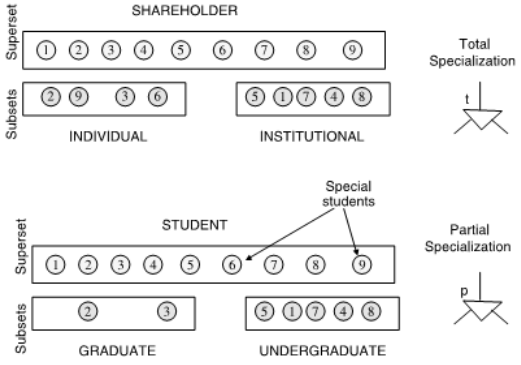


Inheritance of Relationships



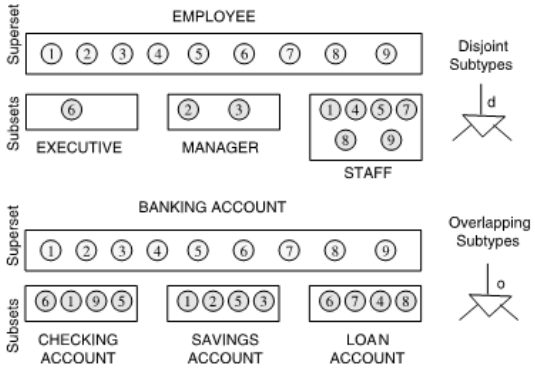
Constraints of Generalization/Specialization

- Total / Partial
  - Total Specialization: All subtypes are defined
  - Partial Specialization: Not all subtypes are known



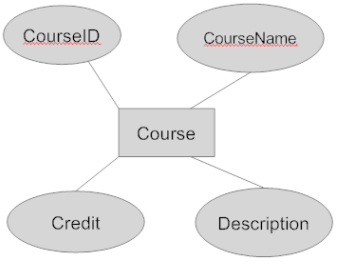
Constraints of Generalization/Specialization

- Disjoint/Overlapping



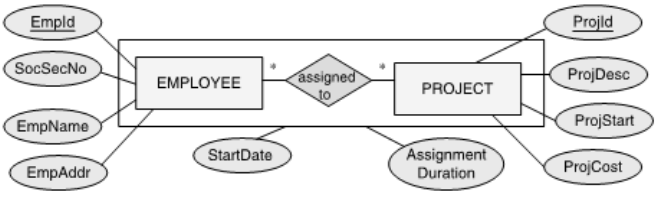
Attribute

- An attribute is a distinct and specific characteristic of an entity type that is of interest to the organization
- Entities of the same type share the same attributes
- Attribute must ty type or relationship



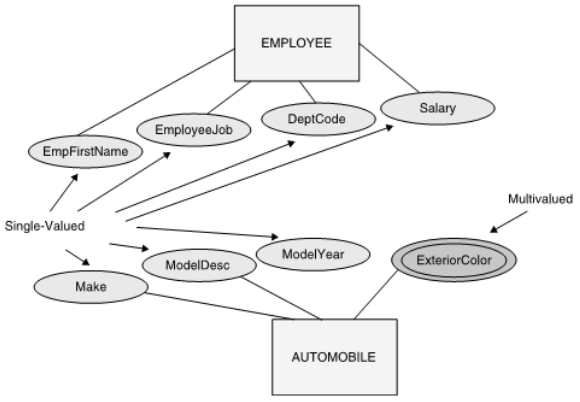
Attribute of relationship

- Relationship can have attributes



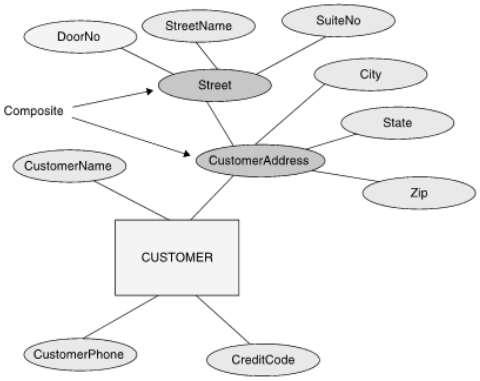
Types of attribute

- Single valued and Multi valued attribute



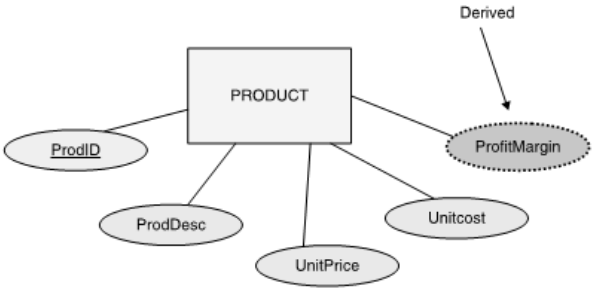
Types of attribute

- Simple and Composite Attributes
  - Composite attribute: Attribute that can be divided further into smaller unit



Type of attributes

- Derived values attribute: attribute of which the value can be derived from values of other existing attributes



Constraints on attribute

- Constraints are rules or restrictions imposed on attribute
  - Value set

Attribute	Domain
EmployeeGender	"Male"/"Female"
ApplicanceColor	"White"/"Black"/"Silver"/"Beige"
EmploymentStatus	"Full-time"/"Part-time"
StudentZip	Valid Zip Codes
CreditCardNumber	Valid 10-digit card number

Constraints on attribute

- Range constraint

Attribute	Domain
HourlyWage	From 5.00 to 50.00
EmployeeDOB	Greater than 1/1/1930
WeeklyWorkHours	0.00 > and < = 80.00
ShipmentCharge	4.99 to 24.99
ExamScore	0 to 100

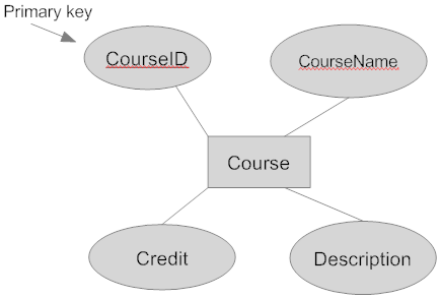
- Type

Attribute	Domain
YearlySalary	Numeric, long integer
DaysWorked	Numeric
CustomerName	Text up to 60 characters
ProductDesc	Text up to 45 characters
CityName	Text up to 65 characters

- Null Value

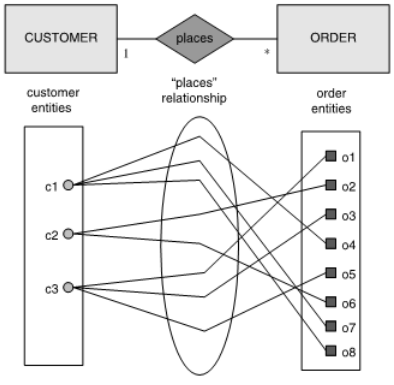
Key attribute

- Key attribute:
  - Attributes whose values enables us to uniquely identify individual occurrences of an entity type
  - Primary key: One of the key which is chosen to serve as identifier of an entity type.



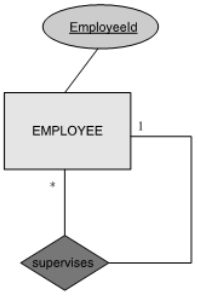
Relationship

- A relationship is an association among several entities



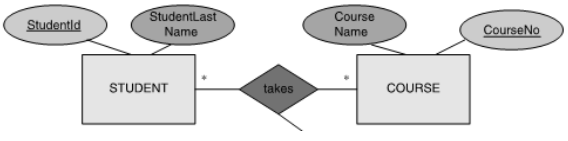
Degrees of Relationship

- Degree of relationship refers to the number of entity types that participate in the relationship
  - Unary relationship

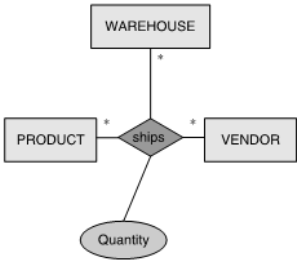


Degree of Relationship

- Binary relationship



- Ternary relationshi

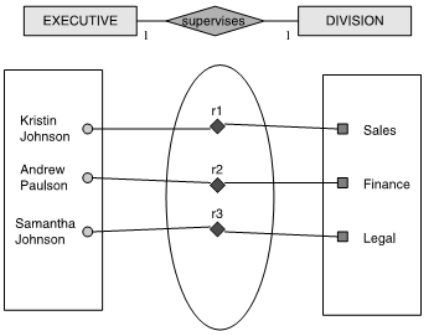


Constraint of Relationship

- Cardinality Constraint:
  - Indicate the number of instance of a specific entity types participate in a relationship
  - One to One
    - An entity in entity set A

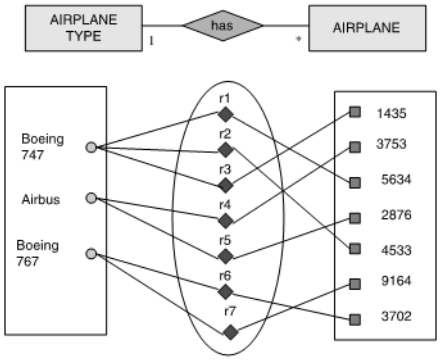
    Related to at most one entity

    In entity set B and vice versa



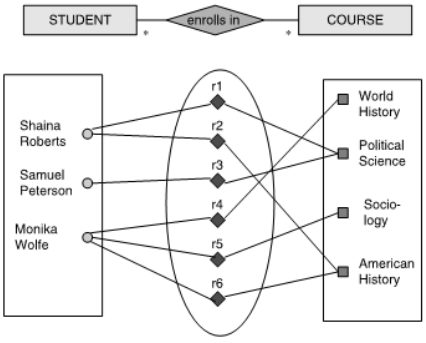
Cardinality Constraint

- One to Many / Many to One
  - If there is a one to many relationship between two entity sets A and B then
    - An entity in set A might related to many entities in B but an entity in B can only related to at most one entity in A



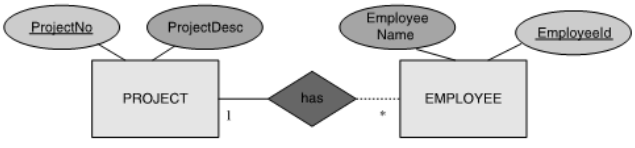
Cardinality Constraint

- Many to Many
  - An entity in entity set A might related to many entities in set B and vice versa



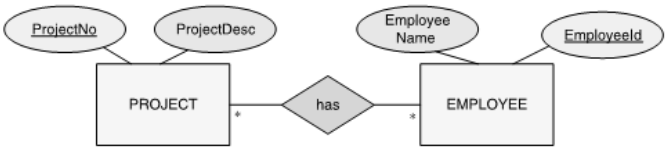
Participation Constraint

- This constraint deal with whether participation of individual entities in a relationship is mandatory or optional
  - Partial Participation
    - Business rule: In the company, one project has at least one employee assigned to it. Some employees may not be working for any project at all



Participation Constraint

- Total Participation
  - Business rule: In the company, one project has at least one employee assigned to it. Every employee works on one or more projects



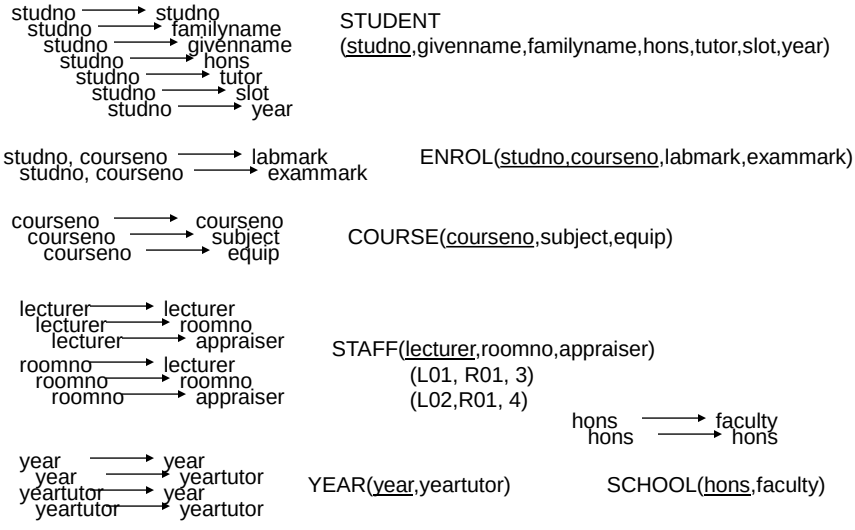
Design Issues

- Relationship or Entity Type ?
- N-ary Relationship or Breaking it up into Binary relationships?
- Multiple relationships between entity types?

Bottom-up approach: normalization

- Student(Id, name, suburb, courseno, coursename, dept)

Using functional dependencies to…Synthesise relations



studno	name	tutor	roomno	courseno	labmark	subject
s1	jones	bush	2.26	cs250	65	prog
s1	jones	bush	2.26	cs260	80	graphics
s1	jones	wibby	2.26	cs270	47	elecs
s2	brown	kahn	IT206	cs250	67	prog
s2	brown	kahn	IT206	cs270	65	elecs
s3	smith	goble	2.82	cs270	49	comms
s4	blogg	goble	2.82	cs280	50	design
s5	jones	zobel	2.34	cs250	0	prog
s6	peters	kahn	A17	cs250	2	prog
null	*null*	capon	A14	*null*	*null*	*null*
null	*null*	*null*	*null*	cs290	*null*	specs

F
studno → name, tutor
tutor → roomno
roomno → tutor
courseno → subject
studno, courseno → labmark

F+
studno, courseno → name partial

studno → roomno transitive

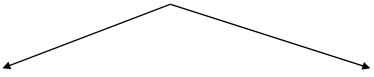
Process of Normalization

- Represent all user views (e.g forms, reports etc) as a collection of relations
- Normalize these relations
- Combine relations that have exactly the same primary key/s.

First Normal Form (1NF)

- Definition
  - there are no repeating groups.
  - a unique key has been identified for each relation
  - all attributes are functionally dependent on all or part of the key.
- Example

STUDENT_DETAILS
(studno, name, tutor, roomno, {courseno, labmark, subject})
studno → name, tutor
tutor → roomno, roomno → tutor
courseno → subject
studno, courseno → labmark



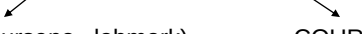
STUDENT
(studno, name, tutor, roomno)
studno → name, tutor
tutor → roomno,
roomno → tutor

ENROL (studno, courseno, subject, labmark)
courseno → subject
studno, courseno → labmark

Second Normal Form (2NF)

- Definition
  - the relation is in 1 NF
  - all non-key attributes are fully functionally dependent on the entire key
    - *partial dependency has been removed*
- Example

ENROL (studno, courseno, subject, labmark)
courseno → subject
studno, courseno → labmark

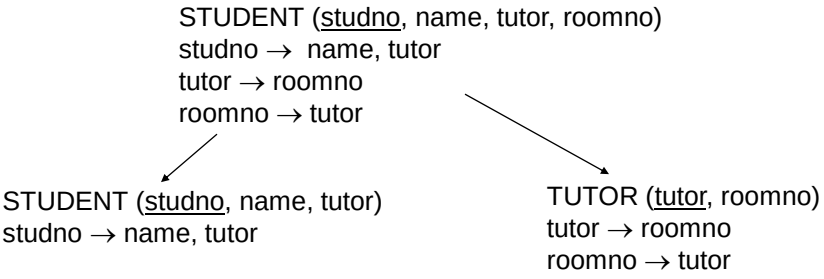


ENROL' (studno, courseno, labmark)
studno, courseno → labmark

COURSE (courseno, subject)
courseno → subject

Third Normal Form (3NF)

- Definition
  - the relation is in 2NF
  - all transitive dependencies have been removed.
  - Transitive dependency: non-key attribute dependent on another non-key attribute.



Example

- STUDENT (studno, name, tutor)
  studno → name, tutor
- TUTOR (tutor, roomno)
  tutor → roomno
  roomno → tutor
- ENROL
  (studno, courseno, labmark)
  studno, courseno → labmark
- COURSE (courseno, subject)
  courseno → subject

Boye-Codd Normal Form (BCNF)

- Definition
  - the relation is in 3NF
  - any remaining anomalies that result form functional dependencies have been removed.

More Normal Forms

- Fourth Normal Form (4NF)
  - the relation is in BCNF
  - any multivalued dependencies have been removed.
- Fifth Normal Form (5NF)
  - the relation is in 4NF
  - any remaining anomalies that result form join dependencies have been removed.

- Remarks
  - only in rare situations that a relation in 3NF is not in 4NF or 5NF.
  - most relations that are in 3NF are also in BCNF.

Lossless or Non-additive Join

Given R ~ a relational scheme, F ~ a set of functional dependencies on R.

Decomposition R = (R1, R2)

- The decomposition of R is non-additive if at least one of the following functional dependencies are in F+

$$R1 \cap R2 \rightarrow R1$$
$$R1 \cap R2 \rightarrow R2$$

- The decomposition of R is non-additive if for every state r of R that satisfies F

$$\pi_{<R1>}(r) * \ldots * \pi_{<Rm>}(r) = r$$

Lossless or Non-additive Join

ENROL (<u>studno, courseno</u>, subject, labmark)
courseno → subject
studno, courseno → labmark

ENROL' (<u>studno, courseno</u>, labmark)
studno, courseno → labmark

COURSE (<u>courseno</u>, subject)
courseno → subject

ENROL' ∩ COURSE = courseno

courseno → subject

(<u>courseno</u>, subject) = COURSE

Lossless or Non-additive Join

STUDENT

studno	name	tutor	roomno
s1	jones	bush	2.26
s2	brown	kahn	IT206
s3	smith	goble	2.82
s4	bloggs	goble	2.82
s5	jones	zobel	2.34
s6	peters	kahn	IT206

studno → name
studno → tutor
tutor → roomno
roomno → tutor

STUDENT1

studno	name	tutor
s1	jones	bush
s2	brown	kahn
s3	smith	goble
s4	bloggs	goble
s5	jones	zobel
s6	peters	kahn

studno → name
studno → tutor

TUTOR

tutor	roomno
kahn	IT206
bush	2.26
goble	2.82
zobel	2.34

tutor → roomno
roomno → tutor

- STUDENT1 ⋈ TUTORS = STUDENT

Dependency Preservation

- The union of dependencies that hold on the individual relations in decomposition D must be equivalent to F.
- Given F on R, $\pi_F(R_i)$ where $R_i \subseteq R$ is the set of dependencies X→Y in F^+ such that the attributes in $X \cup Y$ are all contained in R_i
- Decomposition $D = \{R_1, R_2, ..., R_m\}$ of R is dependency preserving w.r.t. F if

$$(\pi_F(R_1)) \cup .... \cup \pi_F(R_m)))^+ = F^+$$

- Given the restriction of functional dependencies to a relation is the fds that involve attributes of that relation F_i for R_i

$\bigcup_{i=1}^{n} Fi \neq F$ possible, but... $(\bigcup_{i=1}^{n} F_i)^+ = F^+$

Dependency Preservation

studno → name
studno → tutor

tutor → roomno
tutor → appraiser
roomno → tutor
roomno → appraiser
studno → appraiser
studno → roomno

studno → appraiser
studno → roomno

STUDENT

studno	name	tutor	roomno	appraiser
s1	jones	bush	2.26	capon
s2	brown	kahn	IT206	watson
s3	smith	goble	2.82	capon
s4	bloggs	goble	2.82	capon
s5	jones	zobel	2.34	watson
s6	peters	kahn	IT206	watson

STUDENT'

studno	name	tutor
s1	jones	bush
s2	brown	kahn
s3	smith	goble
s4	bloggs	goble
s5	jones	zobel
s6	peters	kahn

TUTOR

studno	roomno	appraiser
s1	2.26	capon
s2	IT206	watson
s3	2.82	capon
s4	2.82	capon
s5	2.34	watson
s6	IT206	watson

studno → name
studno → tutor

STUDENT' * TUTOR = STUDENT

Designing a relational schema

- Build a relational database
  - without redundancy
    - normalisation
  - without loss of information or gain of data
    - lossless join decomposition
  - without losing dependency integrity
    - dependency preservation

What is Database Tuning?

Activity of making a database application run faster:

– Faster means higher throughput (or response time)

– Avoiding transactions that create bottlenecks or avoiding queries that run for hours unnecessarily is a must.

– A 5% improvement is significant.

Why Database Tuning?

- Troubleshooting:
  - Make managers and users happy given an application and a DBMS
- Capacity Sizing:
  - Buy the right DBMS given application requirements
- Application Programming:
  - Coding your application for performance

Some Schema are better than others

- Schema1:

  OnOrder1(supplier_id, part_id, quantity, supplier_address)
- Schema 2:

  OnOrder2(supplier_id, part_id, quantity);

  Supplier(supplier_id, supplier_address);

- Space
  - Schema 2 saves space
- Information preservation
  - Some supplier addresses might get lost with schema 1.
- Performance trade-off
  - Frequent access to address of supplier given an ordered part, then schema 1 is good.
  - Many new orders, schema 1 is not good.

Vertical Partitioning

- Three attributes: account_ID, balance, address.
- Functional dependencies:
  - account_ID –> balance
  - account_ID –> address
- Two normalized schema design:
  - (account_ID, balance, address)

  or
  - **(**account_ID, balance**)**
  - (account_ID, address)
- Which design is better?

Vertical Partitioning

- Which design is better depends on the query pattern:
  - The application that sends a monthly statement is the principal user of the address of the owner of an account
  - The balance is updated or examined several times a day.
- The second schema might be better because the relation (account_ID, balance) can be made smaller:
  - More account_ID, balance pairs fit in memory, thus increasing the hit ratio
  - A scan performs better because there are fewer pages.

Vertical Antipartitioning

- Brokers base their bond-buying decisions on the price trends of those bonds. The database holds the closing price for the last 3000 trading days, however the 10 most recent trading days are especially important.
  - (bond_id, issue_date, maturity, …)
    (bond_id, date, price)

  Vs.
  - (bond_id, issue_date, maturity, today_price, …10dayago_price)
    (bond_id, date, price)