|  |  |  |  |
| --- | --- | --- | --- |
| RA Operator | Name | SQL | Description |
| τ  τA1, A2…(R) | Sort | ORDER BY A1, A2 (DESC|ASC) | Tuples sorted by attributes, in order of listing |
| δ  δ(R) | Eliminate Duplicates | SELECT DISTINCT | Duplication elimination is expensive, because tuples must be sorted or partitioned first |
| Π  ΠC(R) | Extended Projection | SELECT A + B (etc) FROM… | arbitrary expressions involving attributes |
| γ  γl(R) | Count | COUNT (L) FROM… | arbitrary expressions involving attributes |
| outjoint  R outjoint S | Outerjoin | R OUTERJOIN S | a theta-join that resolves dangling tuples by adding NULL for empty values |

**Lecture 6**

**Extended Operators on bags**

**Transactions**

Why?

* Concurrent database access
* Resilience to system failures

Transaction: A sequence of one or more SQL statements treated as one unit.

May begin implicitly by executing a statement, or implicitly with BEGIN TRANSACTION, finishing with a COMMIT or ROLLBACK statement.

**ACID**

**A: Atomic →** Either the entire transaction is executed or nothing is.

Each transaction is “all-or-nothing,” never half-executed.

**C: Consistent →** Database constraints preserved.

Each client, for each transaction can assume all constraints hold when transaction begins and *must* guarantee all constraints hold when the transaction ends.

**I: Isolated** **→** It appears to the user as if only one process executes at a time.

Operations may be interleaved, but execution must be equivalent to some sequential order of transactions.

**D: Durable** **→** Effects of a process survive a system crash.

If the system crashes after commits, all effects of transaction remain in database.

Commit **→** Previous modifications are now permanent in the database.

Rollback **→** Causes the transaction to end, by aborting the process. Results in no effects on the database.

Failures like division by 0 or constraint violations can cause rollback without programmer intervention.

Interleaving Statements

max before min, delete before insert

Transactions solve the issue of interleaving two separate groups of statements from different users.

Isolation Levels (choices about what interactions are allowed by transactions that execute at about the same time)

1. Serializable **→** You see the database before or after other operations are completed.
2. Repeatable Read → Read committed, and everything seen the first time is seen the second time (and possible more).
3. Read Committed **→** You only see transactions that have been committed.
4. Read Uncommitted → Transactions see data even if the modifying transactions were not committed, and never are.

**Views :** relation defined in terms of stored tables (base tables) and other views.

1. Virtual => not stored in database, just a query for constructing the relation
2. Materialized => actually constructed and stored

Views can be queried like tables, but cannot be updated unless it satisfies certain conditions.

CREATE [MATERIALIZED | VIRTUAL] VIEW <name> AS <query> // default is virtual

Ex: CREATE VIEW test AS

SELECT T1.StudentPID, T1.Number, T2.Number

FROM Take AS T1, Take AS T2

WHERE (T1.StudentPID = T2.StudentPID)

AND (T1.DeptName = ’CS’)

AND (T2.DeptName = ’Math’);

CanDrink(drinker, beer)

CREATE VIEW CanDrink AS

SELECT drinker, beer

FROM Frequents, Sells

WHERE Frequents.bar = Sells.bar;

Querying views (and limited modification):

SELECT beer FROM CanDrink

WHERE drinker = ‘Sally’;

**Indexes**:

index = data structure used to speed access to tuples of a relation, given values of one or more attributes

In a DBMS, it is *always* stored as a balanced search tree with giant nodes called a *B-tree*.

There is no standard for declaring indexes!

CREATE INDEX IndName ON Table(attribute);

CREATE INDEX IndName ON Table(a1, a2);

Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.

An index speeds up any query it is applicable to.

It slows down modifications on its relation as the index must also be modified.

**Lecture 7** : Terminology

Data model – describes high-level conceptual structuring of data

Schema – describes how data is to be structured and stored in a database

Data – actual ‘instance’ of database

Properties of a good data model:

1. It is easy to write correct and easy to understand queries.
2. Minor changes in the problem domain do not change the schema.
3. Major changes in the problem domain can be handled without too much difficulty.
4. Can support efficient database access.

**Entity Sets**

Entity – thing/object

Entity set – collection of similar entities

Attribute – property of entity set

Entity 2

relationship

Entity

Entity 2 subclass

isa

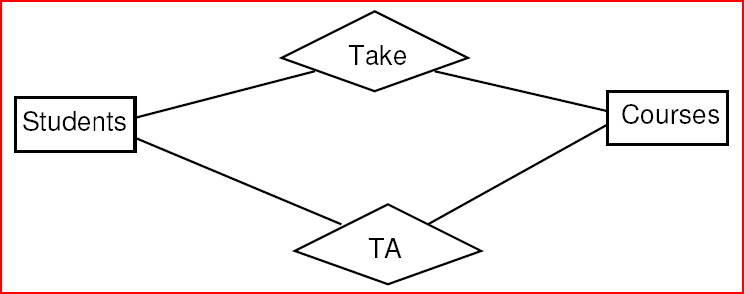
Value of a relationship is a set of lists of currently related entities, one from each of the related entity sets.

Show a many-one relationship by an arrow entering the ‘one’ side.

Show a one-one relationship by an arrow entering both entity sets.

Rounded arrow – exactly one.

Parallel relationships:



**Lecture 8** : Constraints

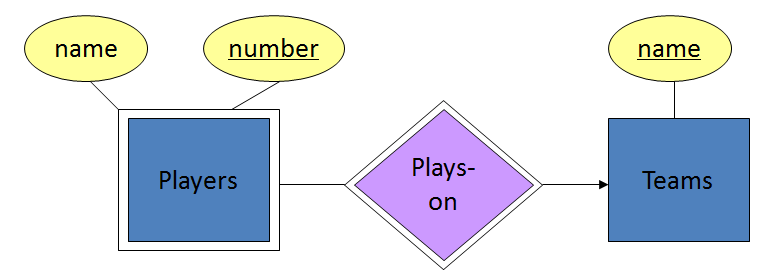
Keys are attributes or sets of attributes uniquely identifying an entity within the entity set.

Referential integrity constraints require a value referred to actually exists in the database.

A key for an entity set E is a set K of one or more attributes such that given any two entities e1 and e2 in E, e1 and e2 cannot have identical values for all the attributes in K.

Key attributes in E/R diagrams should be underlined.

Entity set *E* is said to be *weak* if in order to identify entities of *E* uniquely, we need to follow one or more many-one relationships from *E* and include the key of the related entities from the connected entity sets.



Two players may have the same name, players on 2 teams could have the same number. Together with Team, number should be unique.

The key for a weak entity set is its own underlined attributes and the keys for the supporting entity sets.

**Design**:

Be faithful to the specification of the application.

1. No meaningless/unnecessary attributes.
2. Define multiplicity appropriately.

Avoid redundancy.

1. Saying the same thing in 2 different ways.
2. Wastes space and encourages inconsistency. (change one of the repeated attributes but not the other)

Keep the entities and relationship simple. (don’t use an entity set with an attribute will do)

1. An entity should have at least one non-key attribute, and/or is the many in a many-one or many-many relationship.

Select the right relationships.

1. Do not add unnecessary relationships, i.e. relationships that may be deduced from another.

Select the right type of element.

1. If, for an entity set E, all relationships involving E have arrows entering E, attributes of E collectively identify an entity (no dependent attributes), and no relationship involves E more than once, then:
   1. replace E as follows: if there is a many-one relationship R from an entity set F to E, remove R and make the attributes of E be attributes of F, OR if there is a multi-way relationship R with an arrow to E, make the attributes of E be new attributes of R and remove the arrow from R to E.

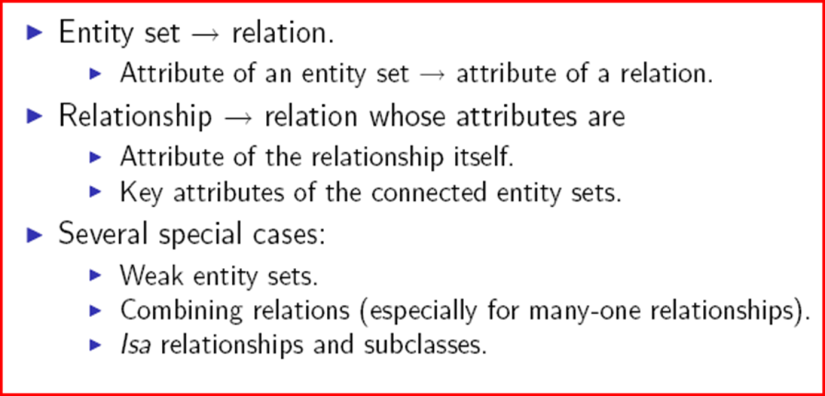
Limit the use of weak entity sets.

1. We need weak entity sets usually when there is no global authority capable of creating unique IDs.

Ex: it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world.

Some relationships that appear to be non-binary may be better represented using binary relationships. (ex: parents, relating child to parent is best replaced by 2 binary relationships father and mother.

**E/R Diagrams to Relations**:



For each entity set, create a relation with the same name and with the same set of attributes.

For each weak entity set W, create a relation with the same name whose attributes are

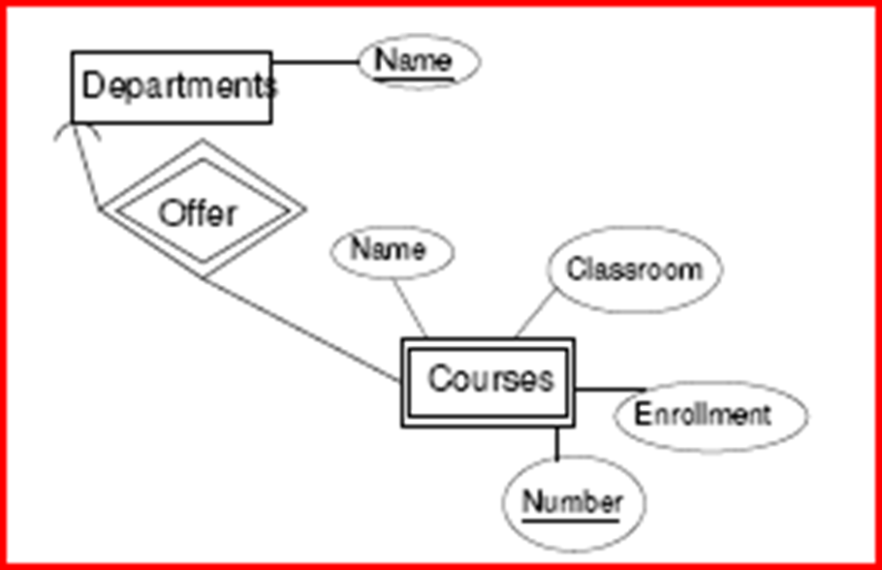
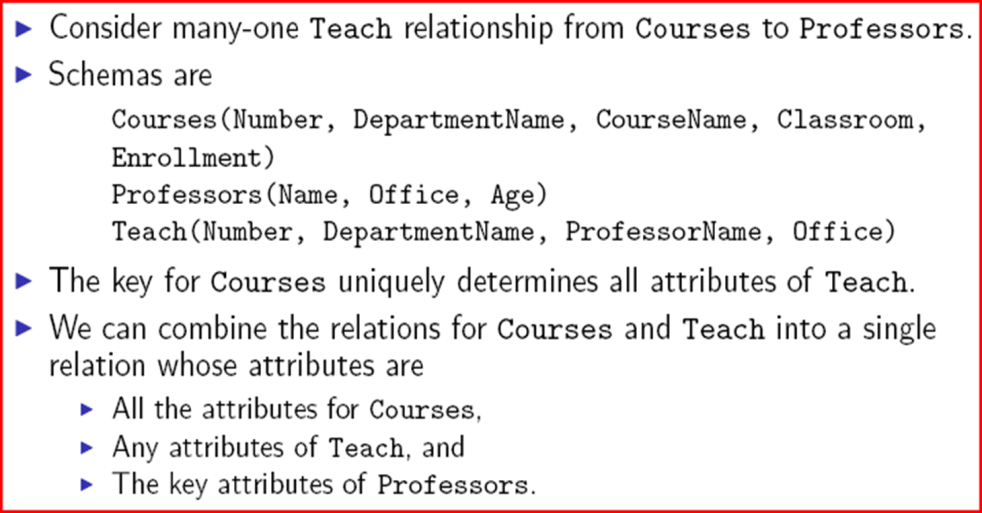
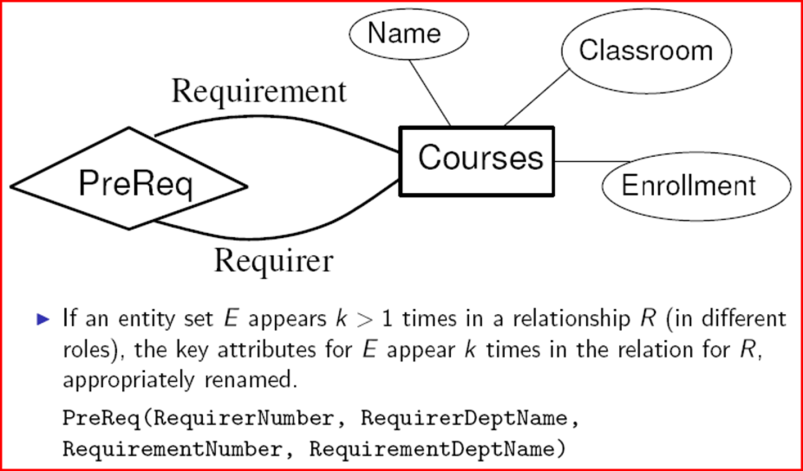
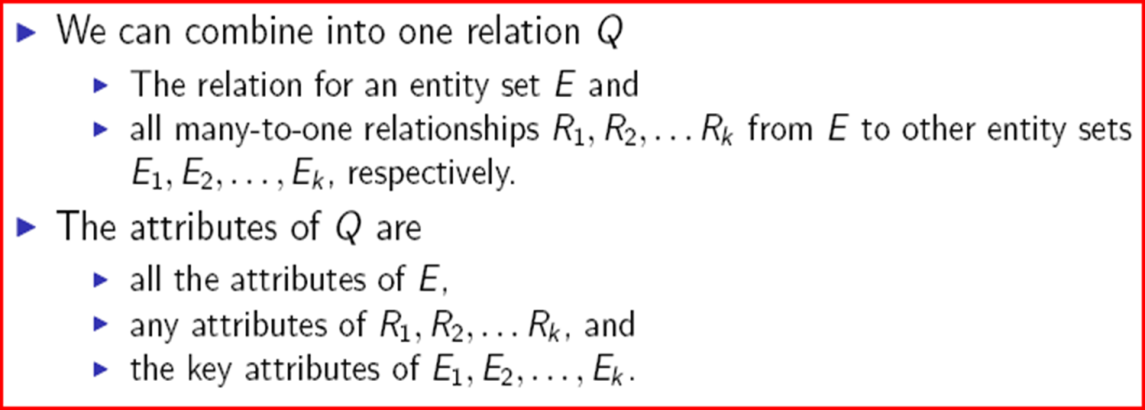
Attributes of W and

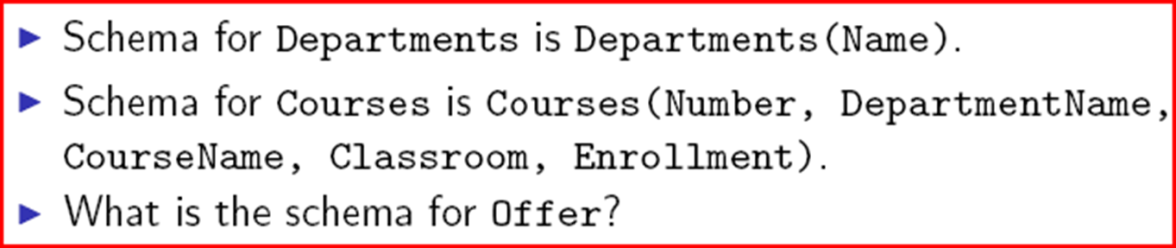
Key attributes of the other entity sets that help form the key for W.

For each non-supporting relationship, create a relation with the same name whose attributes are

Attributes of the relationship itself.

Key attributes of the connected entity sets (even if they are weak).





The schema for Offer is a subset of the schema for the weak entity set, so we can dispense with the relation for Offer.