Contents

Chapter 1

Introduction

Database is set of stored data that is persistent, integrated, interconnected with as little as possible redundency, based on a data model, under control of a central system, used by multiple user, shared and concurrent.

Indexes are used to improve searching. Explain the following

1. B-tree and B^+-tree
2. R-tree (for two dimension trees)
3. ISO and image
4. HDFS

Chapter 2

Entity-Relationship Model

In database design, database designers first need to know the data they are modeling. This includes knowing the *data requirements* and *functional requirements*. Functional requirements are the operations on the data. Once the requirements have been analyzied, the next step is designing a *conceptual schema/model* for the database. Note that this step is not dependent on the DBSM system and is done using high-level conceptual modeling. Then, the database designers must be this conceptual model to a DBSM system, this step is called *logical design* or *data mapping design*. The last step is *physical design* in which the goal is to design the underlying hardware, that is the storage structures, file organization, indicies, access paths, and specifiying physical parameters for the design.

2.1 ER model

Entity-Relationship model is a conceptual model of data.

2.2 Entity

An *entity* is an objecte or a thing that exists independently. Each entity has *attributes* that describe it. For example STUDENT is an entity that is has a name and a student number. Each attribute has some value, base on which we can categorize the attributes into

Definition:

Simple/Atomic Attributes that are not divisible into other attributes. For example "First Name".

Composite Attributes that can be divided into other attributes. For example "Name" can be divided into "First Name", "Middle Name", and "Last Name"

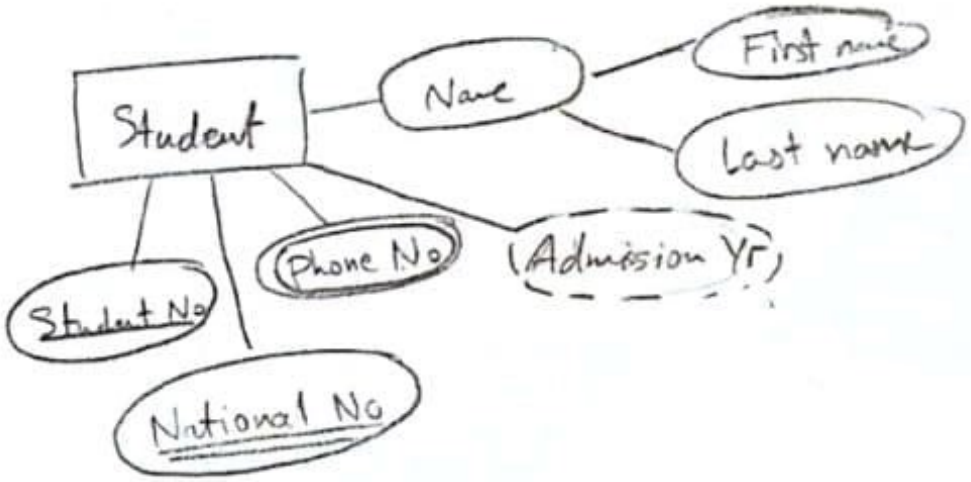
Singlevalued Attributes that can only have one value. For example "Social Security Number".

Multivalued Attributes that can have many values. For example "Phone Number" might have multiple values.

Derived Attributes that can be determined from another attributes which is called the *stored attribute*. For example, "Area" can be determined from "Radius" in case of a CIRCLE entity.

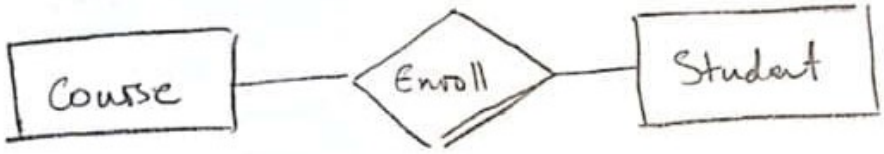
Attributes can have *null* value. We can consider a domain of values _ a set_ for each attribute. To allow nullable and multiplevalued attribute we can consider an attribute to be a subset of this domain.

A set of entities that have similar attribute like STUDENT is called an *entity type*. An important attributes of the entities in an entity type is their *key/identifier attribute* which uniquely determines the specific entity. Some entity types might have more than on key attribute, For example a STUDENT can be determined via its "Student Number" or "Social Security Number".



2.3 Relationship

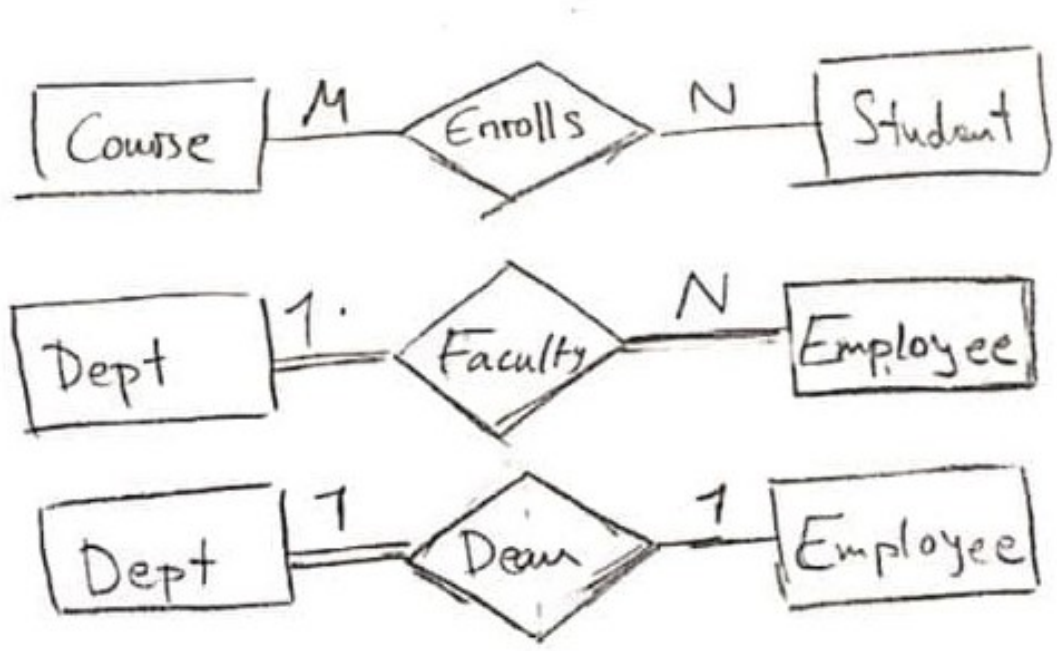
The interaction between two or more entities is a *relationship*. Mathematically, a relationship is a Cartesian product n entity types. $E_1, \ldots, E_n$ said to *participate* in a relationship type $R = E_1 \times \cdots \times E_n$ and individual entities $e_1, \ldots, e_n$ are said to *paraticipate* in a relationshiple instance $r = (e_1, \ldots, e_n)$. The degree of a relationship is the number of entity types that participate in it. Binary relationship are relationship with two entity types. Binary relationship can have different *cardinality ratios*



Many-to-Many denoted as $M : N$. For example each STUDENT can enroll in multiple COURSEs and each COURSE has many STUDENT enrolled in it.

One-to-Many denoted as $N : 1$. Each DEPARTMENT has many FACULTY but each FACULTY has only one DEPARTMENT.

One-to-One denoted as $1 : 1$. Each DEPARTMENT has one DEAN and each DEAN belongs to one DEPARTMENT.



[H] *Total participation* or *existence dependecy* is when each entity in the first entity type must be in a relationship with another entity in the second entity type. For example, each FACULTY must have a DEPARTMENT. In *partial participation* some of the entities in the first entity type are in a relationship with the entities in the second entity type. For example, not every FACULTY is a DEAN.

A relationship might have attributes for itself. For instance, a DEAN *MANAGES* a DEPARTMENT from a starting time to some other time.

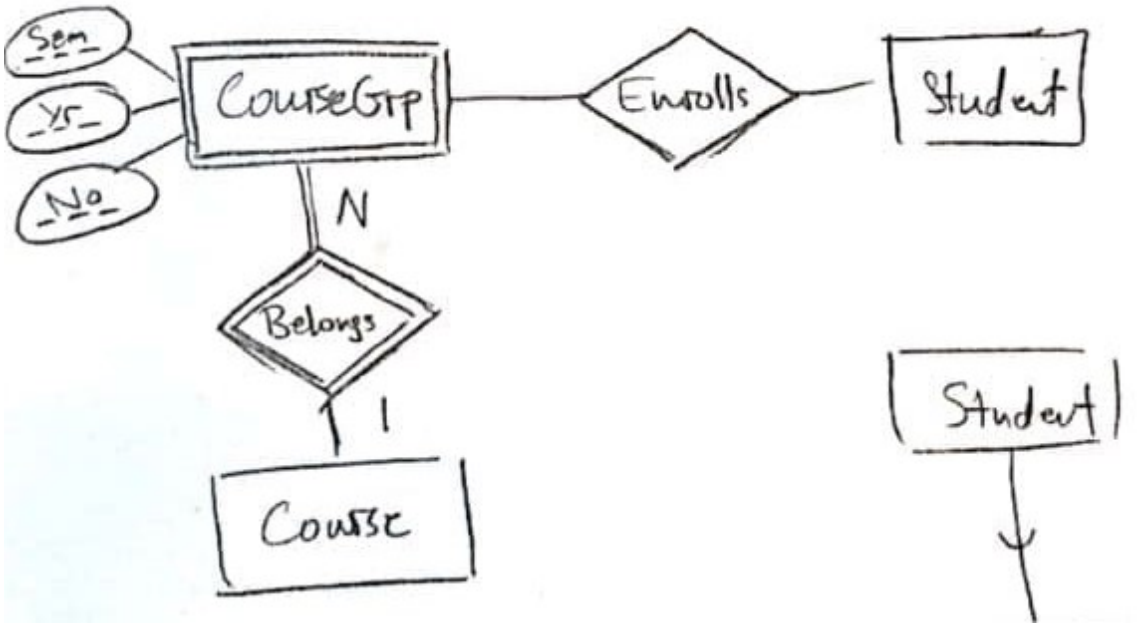
2.4 Weak entity

An entity type that does not have a key attribute is called a *weak entity*. Those entities that have a key attribute are called *strong entity*. In real world, a weak entity is an entity that is depedent on the existence of another entity, called *identifying/owner entity type*. In ER model we can model this relationship by a total participation relationship which is called *identifying relationship*. A weak entity type normal has one or more *partial key* attributes that distinguish it from the other related weak entities to the same owner entity. For example, a COURSEGROUP entity has the "Semester", "Year", and the "Group Number" attributes of a COURSE entity. Every COURSEGROUP entity must be related to a COURSE entity and here, each of its attributes are partial keys.

By adding weak entities we can reduce the order of a relationship to two and even remove attributes of the relationship and give it to the weak entity.

2.5 ER notation and diagrams

Insert the corresponding diagrams :)



2.6 Enhanced/Extended ER model

2.6.1 Inheritance

For inheritance we need a *supertpye* and *subtype* which inherits attributes of the supertype. We show the concept of inheritance in EER model through *specialization* and *generalization*. Specialization is the process of defining a set of subclasses of an entity type. Generalization is the process of finding common attributes and creating a supertype. A specialization has the following characteristics

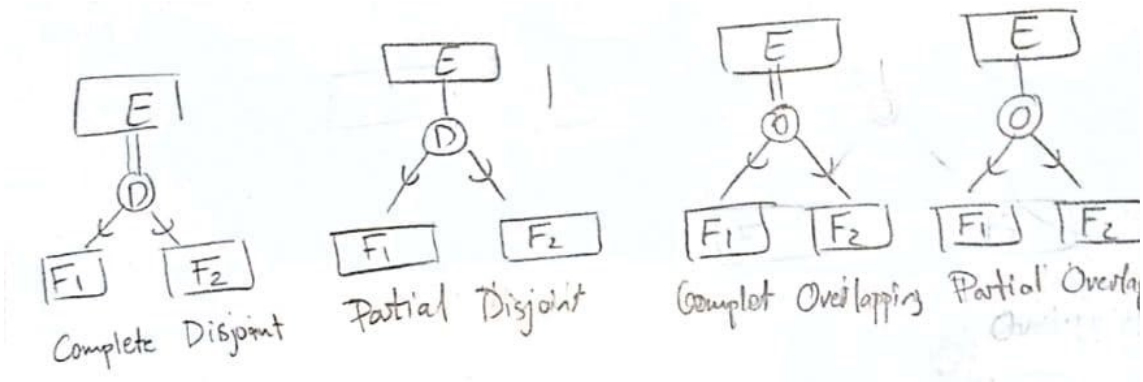
Total every entity of the supertype must be an instance of at least one one the subtypes.

Partial opposite of total.

Disjoint every entity of the supertype belongs to at most one of the subtypes.

Overlapping opposite of disjoint.

Totalness and disjointness are indepedent of each other and hence we can have four combinations.



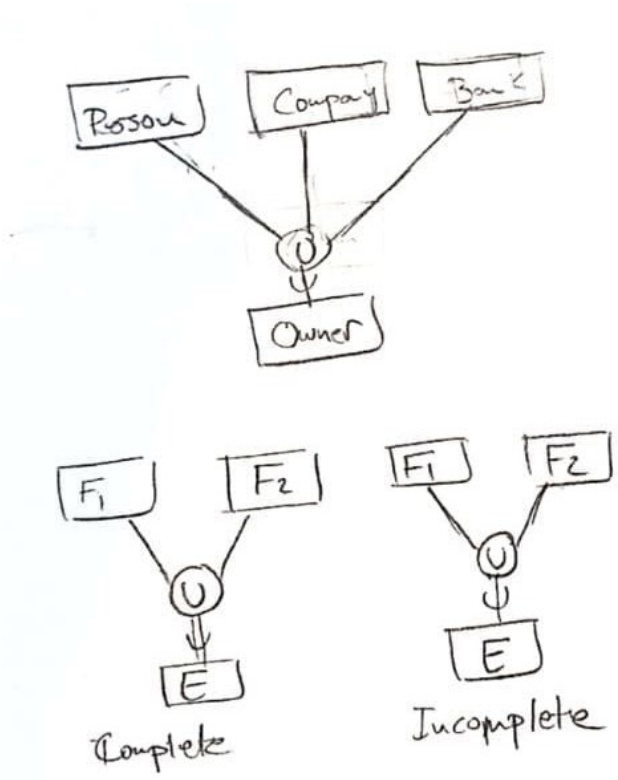
Multiple inheritance can occure when a subtype inherites attributes of one or more supertypes. For example, a TEACHING-ASSISTANT is both a STUDENT and an EMPLOYEE.

2.6.2 Union-Type (Category)

When we want to combine different entities, we use Union types. For example, a VEHICLE can be a CAR or a TRUCK.

Total every entity of the subtype is one the supertype.

Partial opposite of total.



If the keys of the supertpyes are from the same domain, then we can have the same key for the subtype. But, it is better to have different key. Also note that, total union type is equivalent to disjoint total inheritance.

We represent both inheritance and union-type with a IS-A relationship.

2.7 Aggregation and association

We need to represent membershipe/being a component/containment relationships as well. IS-A-PART-OF relationships denote the following meanings – insert diagram

- F is a part of E.
- E contains F.
- E has F.

- F is a member of E.

We can also view relationships as an entity (i literally have no fucking clue why they dont make a science out of it and standardize it with better capabilities).

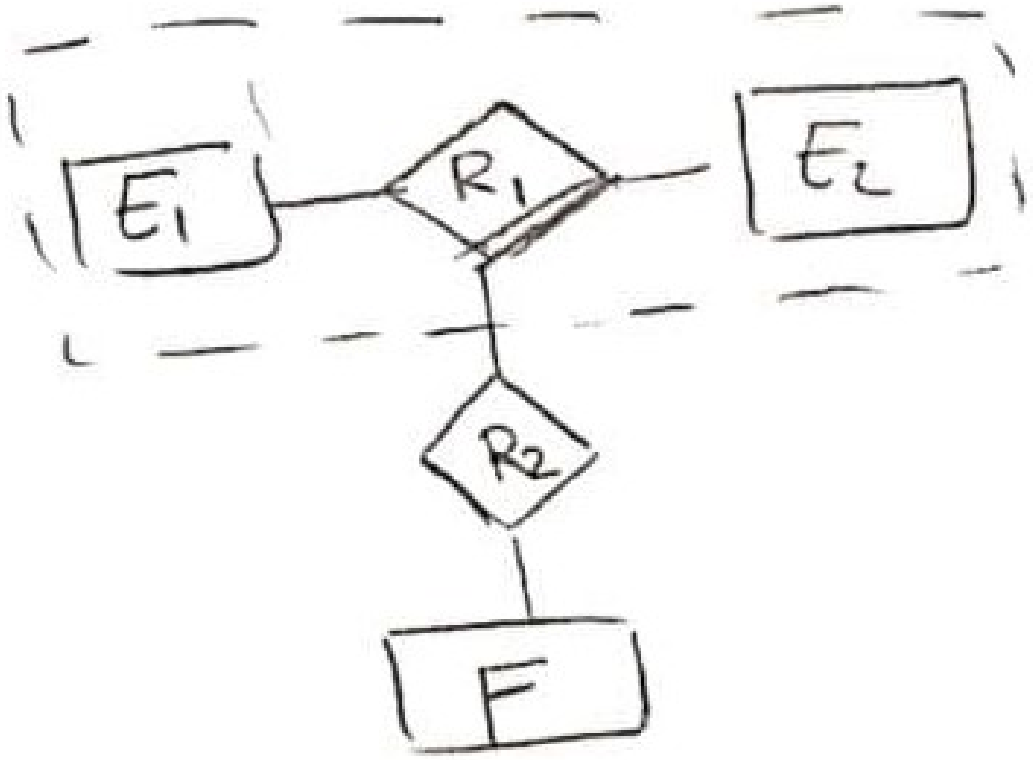
2.8 Traps

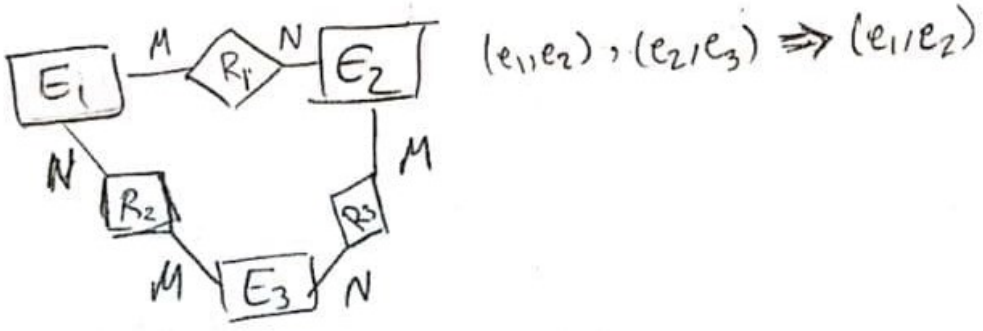
There are two traps

Definition:

Loop trap thinking that $(i, s), (s, p), (p, i) \implies (i, s, p)$ is wrong.

Chasm trap failing to model correctly that in a chain of 1:N relationship, the last entity is "connected" to the first entity, because the participation is not total. (If it is ambiguous, let me tell that i have seen a million examples and still dont fucking understand, it is so fucking dumb _ or maybe i am :\).


E1
R1
E2
R2
F


E1
M
R1
N
E2
N
R2
M
R3
M
E3
N
(e1,e2), (e2,e3) ⇒ (e1,e2)

Chapter 3

Logical Design

Student

Stid	Stname	StLev	stmajor
777	St7	BS	CE

Courses

Coid	Cotitle	CoCredit
489	650	3

M:N

1) Enroll

Stid	Coid	Grade
777	489	20

1:1

1) Dean

Dept	Prof
10	100

2) Dept

Deid	Detitle	Dephone	Dedean
10	Math	09823	100

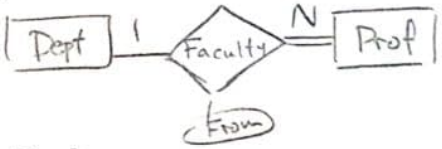
Dept

Deid	Detitle	DePhone
710	Math	09823

Prof

Prid	Prname	Rank
100	xxx	Ass

1:N



1)

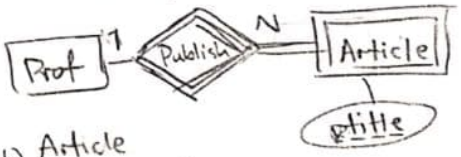
Faculty Prid	Deid	From
100	10	Mont2

2) Prof

Poid	Prname	Rank	Dept
100	xxx	Ass	10

If no total then pk must be nullable

Weak Entity



1) Article

Prid	ptitle
100	"your mama gay"

Multivalued Attribute

1) New table

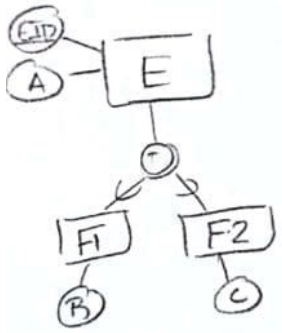
Prid	Phone
100	098 513
100	098 514

2) Nullable Columns (If max is known)

Prid	Prname	Rank	Phone1	Phone2	Phone3
100	xxx	Ass	098513	098514	null

3) Tuple (If supported)

Prid	Prname	Rank	Phones
100	xxx	Ass	(098513, 098514)



Scheme 1

E(Eid, A)
F1(Eid, B)
F2(Eid, C)

No way to know the type from Eid

Type / Scheme #	Complete Disjoint	Partial Disjoint	Complete Overlap	Partial Overlap
1	Yes	Yes	Yes	Yes
2	Yes	Yes	No	No
3	Yes	Yes	Yes	Yes
4	Yes	No	Yes	No

Scheme 3

E(Eid, A, B, C, T1, T2)

bit

Scheme 2

E(Eid, A, B, C, Type)

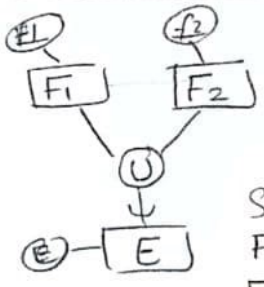
Too many null column

Scheme 4

F1(Eid, A, B)
F2(Eid, A, C)

Have repition if overlapping

Problem if new attribute for supertype



Scheme 1

F1(f1, e)	F1(f1)	F1(f1, c)
F2(f2, c)	F2(f2)	F2(f2, e)
E(e)	E(e, f1, f2)	E(e, type)

Chapter 4

SQL

4.1 Creating table

```
CREATE TABLE stt(
    stid    CHAR(8) PRIMARY KEY,
    stname  VARCHAR(20) NOT NULL,
    stlev   CHAR(2) DEFAULT 'BS',
    stssn   CHAR(10) UNIQUE NOT NULL,
    CHECK (stlev IN ('BS', 'MS', 'PD'))
)
```

```
CREATE TABLE stcot(
    stid    CHAR(8),
    coid  VARCHAR(20) NOT NULL,
    qtr   SMALLINT,
    yr   CHAR(5),
    grade DECIMAL(4,2),
    PRIMARY KEY (stid, coid),
    FOREIGN KEY (stid) REFERENCES stt(stid),
    FOREIGN KEY (coid) REFERENCES cot(coid)
)
```

4.2 Remove table

```
CREATE TABLE stt [CASCADE | RESTRICT]
```

4.3 Update table

```
ALTER TABLE stt
    ADD COLUMN stdepid INT;

ALTER TABLE stt
    DROP COLUMN stssn [CASCADE | RESTRICT];

ALTER TABLE stt
    ALTER COLUMN stlev SET DEFAULT 'BS';

ALTER TABLE stt
    ADD CONSTRAINT con1 CHECK(stdepid  BETWEEN 11 AND 45);

ALTER TABLE stt
    DROP CONSTRAINT con1;
```

```
INSERT INTO stt (stid,stname,stlev,stdepid)
    VALUES ('999','ali','BS',4);

INSERT INTO stt
    VALUES ('888','mohsel','BS',5);

DELETE FROM stt WHERE
    stid = '999' AND stname = 'ali';

DELETE FROM stt WHERE TRUE;
-- is the same as
TRUNCATE TABLE stt;
```

In case we may want to delete a foreign key or what have you, we can determine the action in defintion

```
CREATE TABLE stcot(
    stid    CHAR(8),
    coid  VARCHAR(20) NOT NULL,
    qtr   SMALLINT,
    yr   CHAR(5),
    grade DECIMAL(4,2),
    PRIMARY KEY (stid, coid),
    FOREIGN KEY (stid) REFERENCES stt(stid) ON [DELETE| UPDATE]
   [RESTRICT | CASCADE | SET NULL | SET DEFAULT | NO ACTION],
    FOREIGN KEY (coid) REFERENCES cot(coid)
    -- on default is RESTRICT
)
```

4.4 Retrieving information

```
SELECT * FROM stt;
SELECT stid FROM stt;
SELECT stid,stlev FROM stt WHERE stdepid = 11;
SELECT stid FROM stt WHERE stdepid IS [NOT] NULL;
SELECT stid FROM stt
    WHERE stname [NOT] LIKE ['%N','M^%', '--A--'];
-- '%N' : ends with N
-- '^M%': does not start with M
-- '--[A-D]--' : exactly 5 characters and the third is A
SELECT * FROM stt ORDER BY stid [ASCENDING | DESCENDING];

-- We also have INTERSECT, UNION, UNION ALL, EXCEPT between two
   tables

SELECT stid FROM stt as T1 WHERE T1.stid = '98999';
-- Other keywords ANY, IN, NOT IN, ALL , EXISTS

SELECT [COUNT(*), MAX(grade), MIN(grade), SUM(grade * 2 ), SUM(
   grade)] FROM stcot
        WHERE qtr = 1
    GROUP BY (stid)
    HAVING COUNT(DISTINCT(COID)) > 10;

-- First WHERE is executed the GROUP By
-- When using GROUP BY, only the grouped entities and
   accumulative function are allowed to be queried

SELECT * FROM stt,stcot
    WHERE stt.stid = stcot = stid;

SELECT * FROM stt [INNER | [LEFT | RIGHT | FULL] OUTER] JOIN
   stcot [ON stt.stid = stcot.stid] [USING(stid)]

-- LEFT OUTER if there is no row in stcot corresponding to the
   row in stt
-- RIGHT OUTER if there is no row in stt for some row in
   setcounter

WITH RECURSIVE rec_table(
    (SELECT ...)
    UNION [ALL]
    (SELECT ...)
)
```