

# VERİ TABANI SİSTEM GERÇEKLEME DERS NOTLARI -1

## 1.DERS : GİRİŞ, TANIMLAR..

### Kaynaklar (türkçe, ing.):

- *Prof. Dr. Ünal Yarımağan*, Veri Tabanı Sistemleri
- *Elmasri, Navathe*, Fund. of Database Systems, 5th ed., Addison Wesley
- “Edward Sciore, Database Design and Implementation, 2009, John Wiley”

# Giris Konu Başlıkları

- VT nedir?
- VTYS nedir?
- Veri modeli nedir?
- Örnek bir veri tabanı: **UNIVERSİTE**
- VT kullanıcıları
- VT programla dilleri
- VTYS olanakları ve dosya-işleme sisteminden farkları
- VTYS ana işlevsel modülleri
- VTYS 3-şema mimarisi ve veri bağımsızlığı
- VT kullanım/erişim mimarileri (2-katlı, 3 –katlı)
- VT türleri
- RM
- RA-SQL

# Veri Tabanı Nedir?

- Birden çok uygulama tarafından kullanılan
  - Gereksiz yinelenmelerden arınmış
  - Düzenli bir şekilde saklanan
  - Birbiriyle ilişkili (*uyumlu olarak*)
  - Sürekli , fakat statik olmayan
  - Belirli bir amaç için bir araya getirilmiş
- VERİ TOPLULUĞU (*küçük bir dünya*)'dur.
- Örnek: şirket, bakanlık, üniversite, market stok takip....

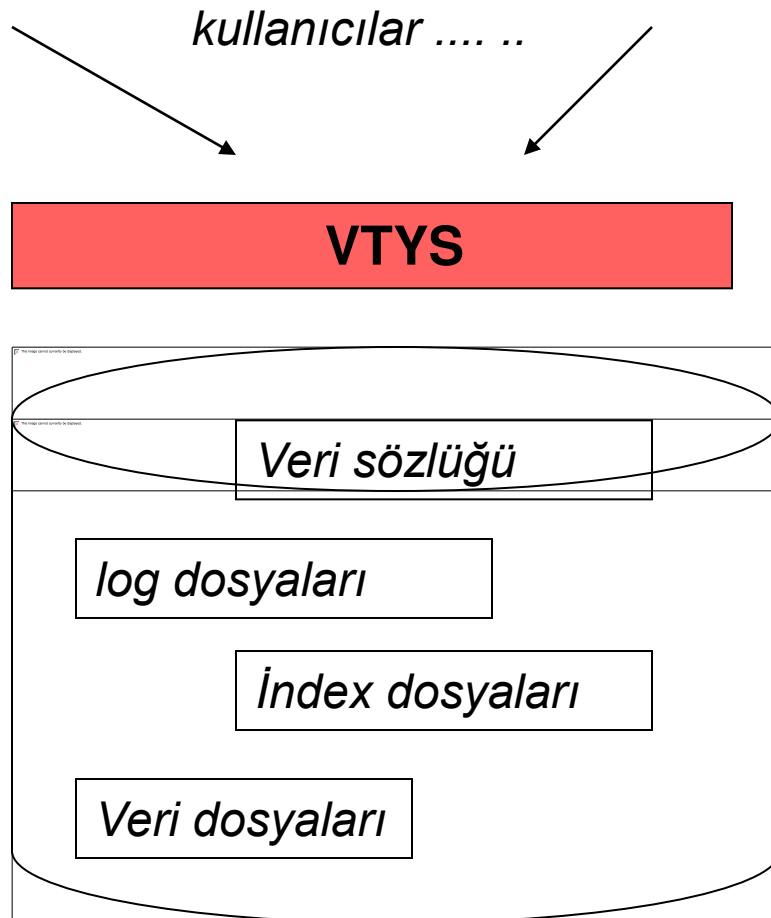
# VT nedir?

- VT'na yeni bilgi, en kısa zamanda yansitilmali
- VT büyüklüğü için bir kısıtlama yok
- Örnek : Amazon.com
  - 2 teraB ( $10^{12}$  B)
  - 20 milyon kitap
  - 200 sunucu bilgisayar üzerinde kayıtlı
  - Günlük 15 milyon kullanıcı
  - Yaklaşık 100 admin

# VTYS Nedir?

- Veri tabanı sistemi ile ilgili her türlü işletimsel gereksinimleri karşılamak için kullanılan **sistem seviyesinde, karmaşık, merkezi** yazılım sistemine VTYS denir. **VTYS genel olarak şu olanakları sağlar:**
  - VT tanımlanması, gerçekleşmesi (oluşturulması), kullanımı, paylaşımı
  - Kontrollü veri tekrarı
  - Sorgu işlemede verimli erişim metodlarını kullanır.
  - Çoklu kullanıcıli hizmet, veri kurtarma ve yedekleme imkanı sağlar.
  - Farklı kullanıcı arayüzlerine imkan sağlar.
  - Üst seviyeli karmaşık iş kısıtlamalarının tanımlanması, gerçekleşmesi ve sağlanmasına olanak sağlar.
  - Güvenlik tanımlamaları ve sağlanmasını kontrol eder.
- VT sistemine, **gerek işletim sistemi gerek diğer kullanıcılar (uygulama programları gibi...) doğrudan erişemez**; ancak VTYS üzerinden erişebilir.

# Genel VT / VTYS yapısı



- Veri sözlüğü, veri tabanı tanımlarının (*metada*) saklandığı dosyalardır.
- İndex dosyaları, fiziksel erişim dosyalarıdır.
- Log dosyaları güvenlik amaçlı dosyalardır.

# Konular

- VT nedir?
- VTYS nedir?
- **Veri modeli nedir?**
- Örnek bir veri tabanı: **UNİVERSİTE**
- VT kullanıcıları
- VT programla dilleri
- VTYS olanakları ve dosya-işleme sisteminden farkları
- VTYS ana işlevsel modülleri
- VTYS 3-şema mimarisi ve veri bağımsızlığı
- VT kullanım/erişim mimarileri (2-katlı, 3 –katlı)
- VT türleri
- RM
- RA-SQL

# Veri Modeli Nedir?

- Gerçek dünya verilerini kavramsal ve mantıksal seviyede düzenlemek için kullanılan *yapı ve kavramlar bütünü* olarak tanımlanır.
- Örnek; E-R modeli, ilişkisel model (RM)
- Genel VT tasarımı:
  - kavramsal tasarım
  - mantıksal düzenleme, varlık ve bağıntıların belirlenmesi
  - veri tipleri, değer aralığı, uzunluk belirlenmesi
  - veri bütünlüğü kısıtlamalarının belirlenmesi
  - fiziksel tasarım tercihleri
  - kullanıcıların belirlenmesi ve güvenlik ayarları



# Categories of Data Models

## ■ Conceptual (high-level, semantic) data models:

- Provide concepts that are close to the way many users perceive data. (Also called entity-based or *object-based* data models.)

## ■ Implementation (representational) data models:

- used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).

## ■ Physical (low-level, internal) data models:

- Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals

# Schemas versus Instances

- Database Schema:
  - The ***description*** of a database.
  - Includes descriptions of the database structure, data types, and the constraints on the database.
- Schema Diagram:
  - An ***illustrative*** display of (most aspects of) a database schema.
- Schema Construct:
  - A ***component*** of the schema or an object within the schema, e.g., STUDENT, COURSE.
- Database State:
  - The actual data stored in a database at a ***particular moment in time***. This includes the collection of all the data in the database.
  - Also called database instance (or occurrence or snapshot).
    - The term *instance* is also applied to individual database components, e.g. *record instance*, *table instance*, *entity instance*

# Example of a Database Schema vs. State

## STUDENT

Name	Student_number	Class	Major
------	----------------	-------	-------

## COURSE

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

## PREREQUISITE

Course_number	Prerequisite_number
---------------	---------------------

## SECTION

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------

## GRADE\_REPORT

Student_number	Section_identifier	Grade
----------------	--------------------	-------

**Figure 2.1**

Schema diagram for the database in Figure 1.2.

## COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

## SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

## GRADE\_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

## PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

**Figure 1.2**

A database that stores student and course information.

# Konular

- VT nedir?
- VTYS nedir?
- Veri modeli nedir?
- Örnek bir veri tabanı: UNIVERSİTE
- **VT kullanıcıları**
- **VT programla dilleri**
- **VTYS olanakları ve dosya-işleme sisteminden farkları**
- **VTYS ana işlevsel modülleri**
- **VTYS 3-şema mimarisi ve veri bağımsızlığı**
- **VT kullanım/erişim mimarileri (2-katlı, 3 –katlı)**
- **VT türleri**
- **RM**
- **RA-SQL**

# VT kullanıcıları

## ■ Yönetici (**Admin**)

- VT erişimi ve kullanımı kontrol eder
- Sistem için gerekli s/w,h/w desteğini belirler
- Güvenlik açığını, verimsiz kaynak kullanımını belirler

## ■ Tasarımcı (**Designer**)

- Verinin her aşamada modellenmesi ile ilgilenir. Gerçekleme öncesi aşamalardan sorumludur. VT kullanıcıları ile haberleşir gereksinim analizi yapar. Genel olarak bütün kullanıcılar ile yakın temas vardır.

## ■ Son kullanıcılar (**End user**)

## ■ VT sistem yazılımcısı

# DBMS Languages

- Data Definition Language (DDL)
  - Used by the DBA and database designers to specify the conceptual schema of a database.
  - In many DBMSs, the DDL is also used to define internal and external schemas (views).
- Data Manipulation Language (DML)
  - High-Level or Non-procedural Languages: These include the relational language SQL
    - May be used in a standalone way or may be embedded in a programming language
  - Low Level or Procedural Languages:
    - These must be embedded in a programming language

# DBMS Programming Language Interfaces

- **Programmer interfaces** for embedding DML in a programming languages:
  - **Embedded Approach:** e.g. embedded SQL (for C, C++, etc.), SQLJ (for Java)
  - **Procedure Call Approach:** e.g. JDBC for Java, ODBC for other programming languages
  - **Database Programming Language Approach:** e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components
- **User-friendly interfaces**
  - Menu-based, popular for browsing on the web, Forms-based, Graphics-based (*Point and Click, Drag and Drop, etc.*)
  - Natural language: requests in written English

# Database System Utilities

- To perform certain functions such as:
  - Loading data stored in files into a database. Includes data conversion tools.
  - Backing up the database periodically on tape.
  - Reorganizing database file structures.
  - Report generation utilities.
  - Performance monitoring utilities.
  - Other functions, such as sorting, user monitoring, data compression, etc.
- Data dictionary / repository:
  - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
- Application Development Environments and CASE (computer-aided software engineering) tools : *PowerBuilder (Sybase)*, *JBuilder (Borland)*, *JDeveloper 10G (Oracle)*



# Niye VTYS? *dosya-işleme niye yetersiz?*

```
public static List<String> getStudents1997() {  
    List<String> result = new ArrayList<String>();  
    FileReader rdr = new FileReader("students.txt");  
    BufferedReader br = new BufferedReader(rdr);  
    String line = br.readLine();  
    while (line != null) {  
        String[] vals = line.split("\t");  
        String gradyear = vals[2];  
        if (gradyear.equals("1997"))  
            result.add(vals[1]);  
        line = br.readLine();  
    }  
    return result;  
}
```

(a) Using a file system model

```
select SName from STUDENT where GradYear = 1997
```

(b) Using the relational model

## **Figure 1-3**

Two ways to retrieve the name of students graduating in 1997

- Veri sorgulama kolaylığı

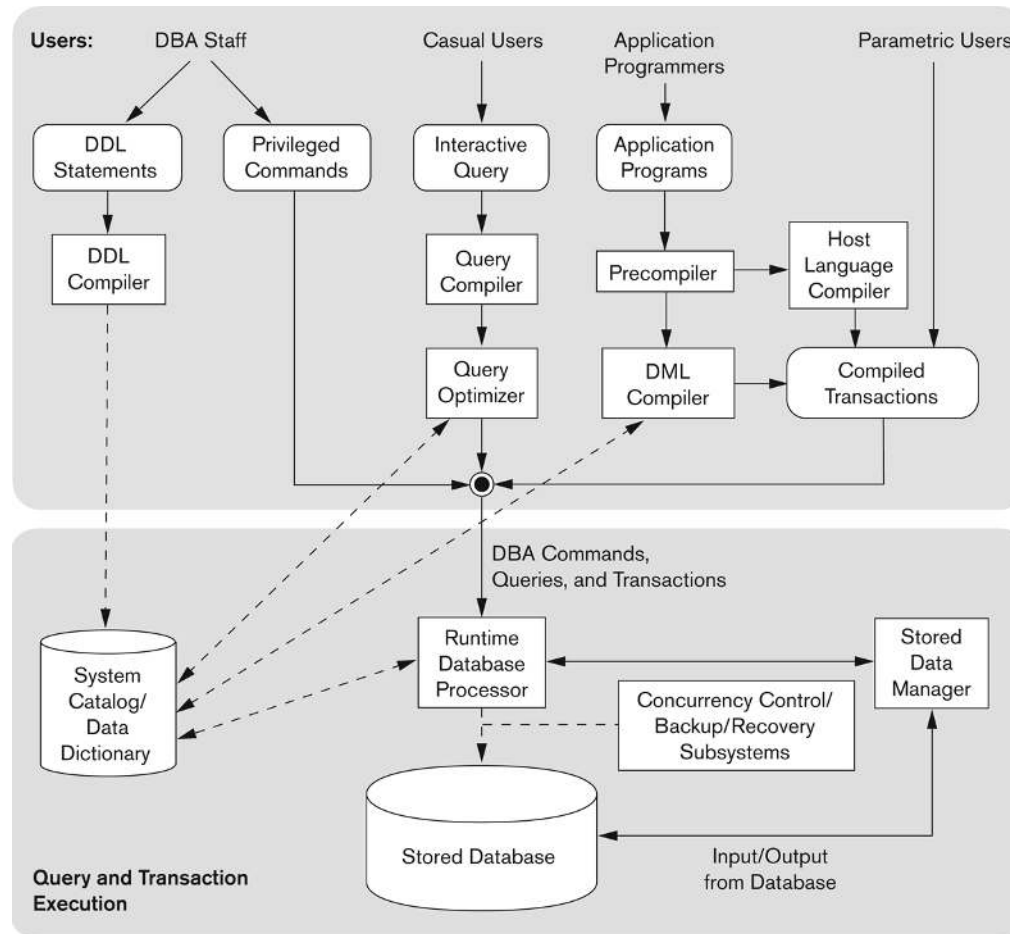
# Niye VTYS? *dosya-işleme niye yetersiz?*

- Üstveri(*metadata*):Veri tabanında saklanan verinin tanımları (meta data) ve diğer kısıtlamalar da saklanıyor. Farklı küçük dünyalar aynı veri tabanında saklanabiliyor. (Buna *Program-veri bağımsızlığı* denir.)
  - *Dosya işleme uygulaması sadece o uygulama için yazılmış.*
- Kayıt saklama ve erişimde
  - *“güçlü” veri yapıları ihtiyacı*
  - *Tampon kullanımı*
- *program-operasyon bağımsızlığı* ihtiyacı
- Verinin farklı görünümü (*multiple views*)
- Paylaşım (*sharing*) ve çoklu hareket işleme (*transaction processing*) imkanı
  - *Eşzamanlılık*
  - *Veri kurtarma ve geri sarma*
  - *Güvenlik ve yetkilendirme*
- **SONUÇ OLARAK;** yavaş saklama unitesinde saklanan, çok kişi tarafından erişilen, büyük veri yığınlarında, veri kurtarma desteği<sup>18</sup> ile gerçek zamanda hizmet vermek için VTYS’na ihtiyacımız var!<sup>18</sup>

# Konular

- VT nedir?
- VTYS nedir?
- Veri modeli nedir?
- Örnek bir veri tabanı: UNIVERSİTE
- VT kullanıcıları
- VT programlama dilleri
- VTYS olanakları ve dosya-işleme sisteminden farkları
- **VTYS ana işlevsel modülleri**
- **VTYS 3-şema mimarisi ve veri bağımsızlığı**
- **VT kullanım/erişim mimarileri (2-katlı, 3 –katlı)**
- **VT türleri**
- **RM**
- **RA-SQL**

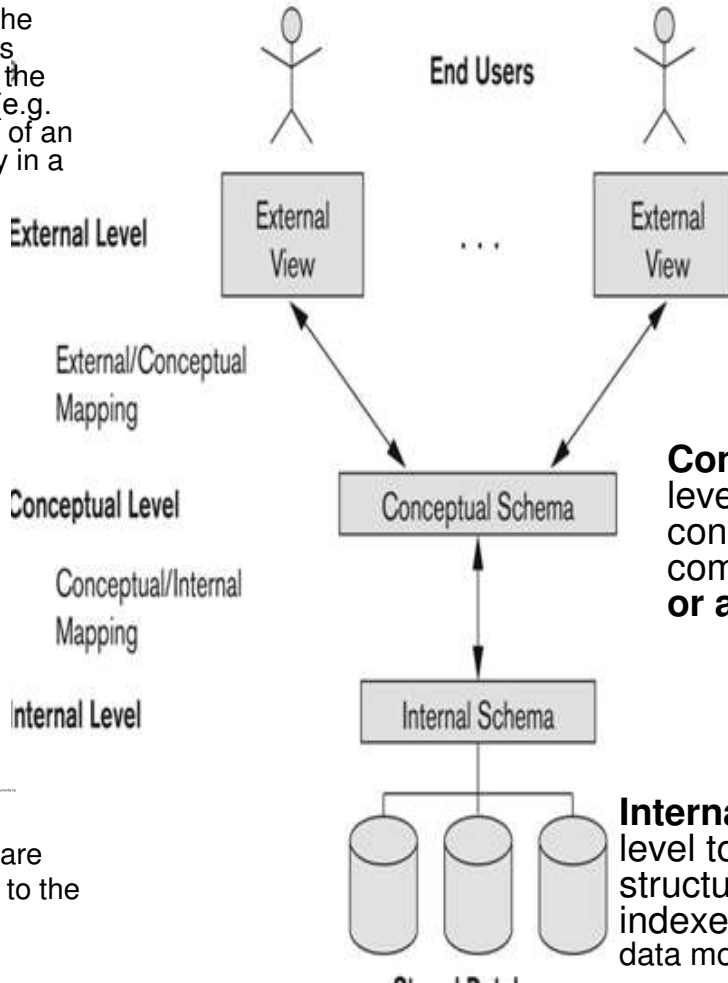
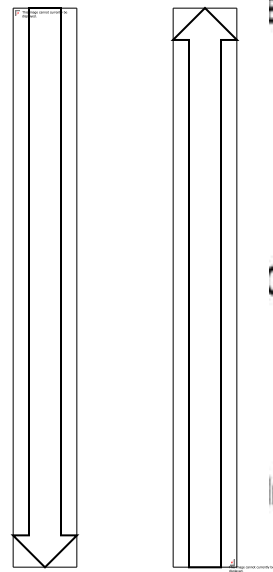
# Typical DBMS Component Modules



**Figure 2.3**  
Component modules of a DBMS and their interactions.

# The three-schema architecture

Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)



**External schemas** at the external level to describe the various user views. Usually uses the same data model as the conceptual schema.

How data is used?

**Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users. Uses a **conceptual** or an **implementation** data model.

What data is inside?

**Internal schema** at the internal level to describe physical storage structures and access paths (e.g. indexes). Typically uses a **physical** data model.

How data is stored?

- Proposed to support DBMS characteristics of:

- **Program-data independence.**
- Support of **multiple views** of the data.

- Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization

# Data Independence

- **Physical Data Independence:**

- The capacity to change the internal schema without having to change the conceptual schema. *(For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance)*

- **Logical Data Independence:**

- The capacity to change the conceptual schema without having to change the external schemas and their associated application programs.

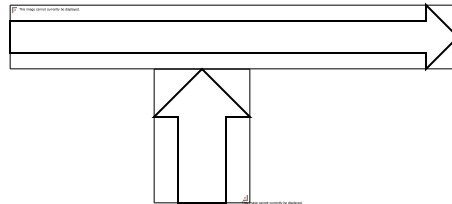
- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.

- The higher-level schemas themselves are **unchanged**.

- Hence, the application programs need not be changed since they refer to the external schemas.

# Physical Data Independence

Select Sname  
from STUDENT  
where GradYear=1997

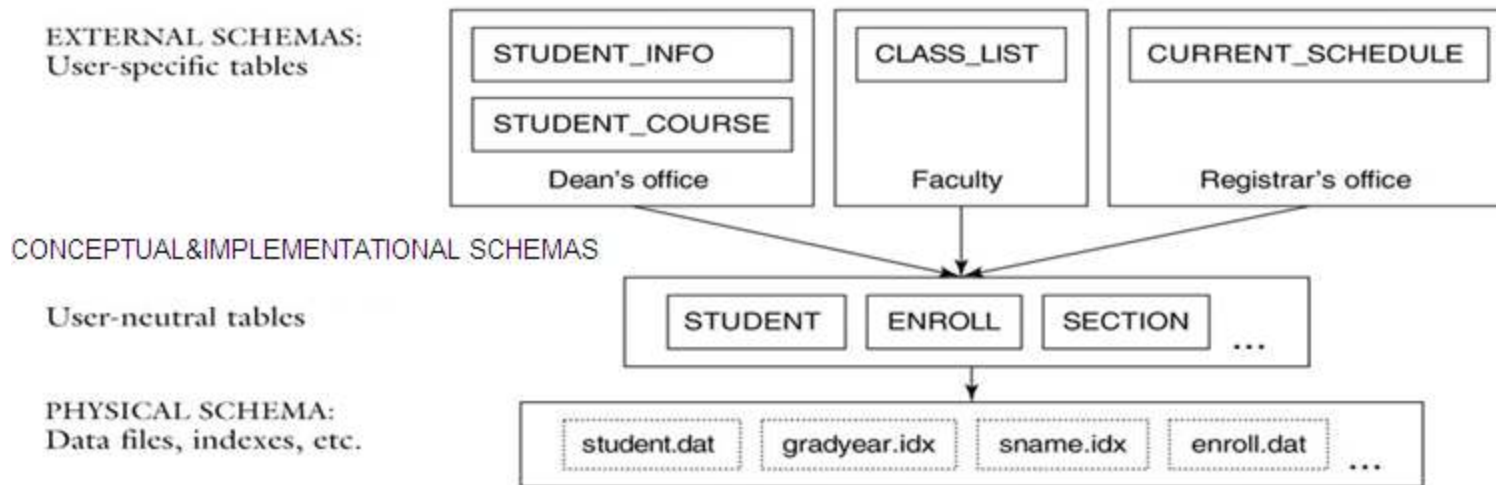


Database catalog  
(conceptual and  
physical schemas)

```
public static List<String> getStudents1997() {  
    List<String> result = new ArrayList<String>();  
    FileReader rdr = new FileReader("students.txt");  
    BufferedReader br = new BufferedReader(rdr);  
    String line = br.readLine();  
    while (line != null) {  
        String[] vals = line.split("\\t");  
        String gradyear = vals[2];  
        if (gradyear.equals("1997"))  
            result.add(vals[1]);  
        line = br.readLine();  
    }  
    return result;  
}
```

- Ease of use
- Query optimization
- Isolation from changes to physical schema

# Logical Data Independence



- In a database, if users have their own external schemas, then this db support logical data independence.
- Adv. of logical data dependence:
  - Customized external schema
  - Isolation from changes to conceptual schema
    - Suppose a change in conceptual schema: split `STUDENT` table into 2 tables: `CURR_STUDENT` and `ALUMNI`. What happens to external schema?
  - Better security



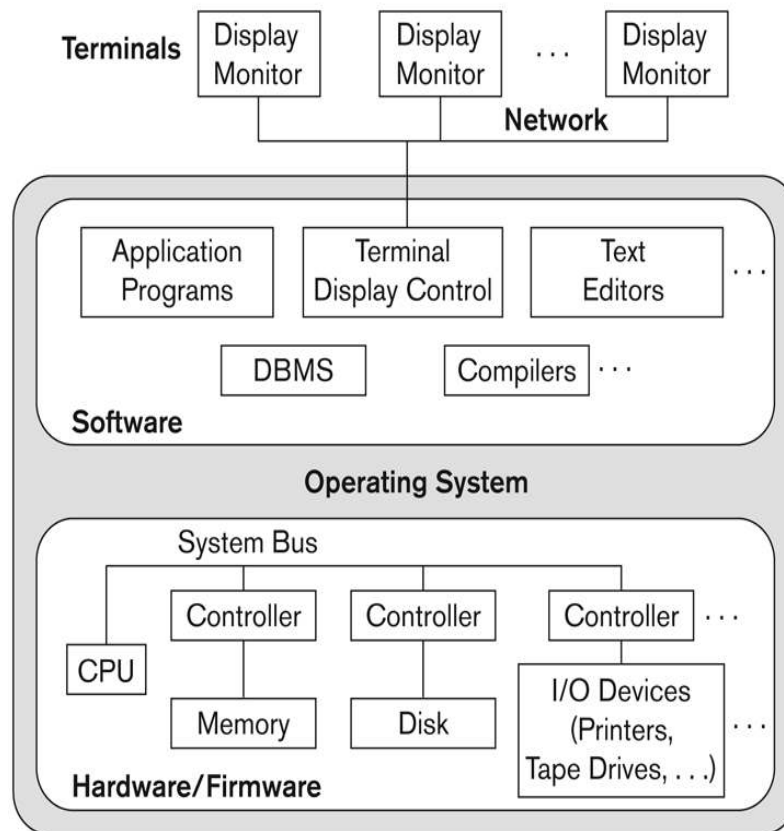
# Konular

- VT nedir?
- VTYS nedir?
- Veri modeli nedir?
- Örnek bir veri tabanı: UNIVERSİTE
- VT kullanıcıları
- VT programla dilleri
- VTYS olanakları ve dosya-işleme sisteminden farkları
- VTYS ana işlevsel modülleri
- VTYS 3-şema mimarisi ve veri bağımsızlığı
- **VT kullanım/erişim mimarileri (2-katlı, 3 –katlı)**
- **VT türleri**
- **RM**
- **RA-SQL**

# Centralized and Client-Server DBMS Architectures

## ■ Centralized DBMS:

- Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.
- User can still connect through a remote terminal – however, all processing is done at centralized site.

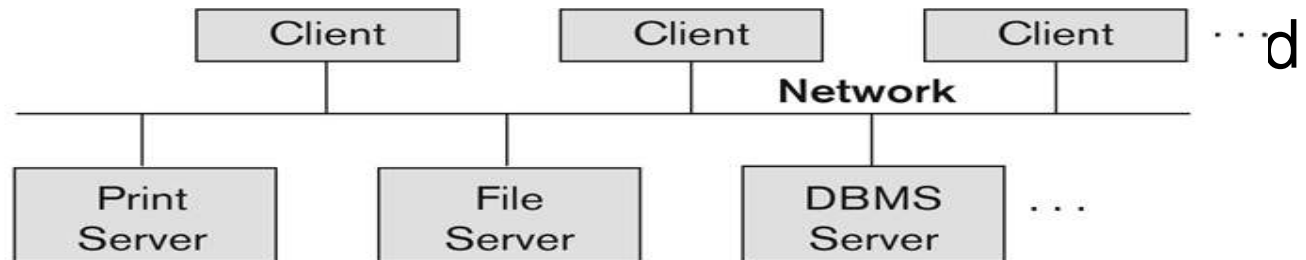


**Figure 2.4**  
A physical centralized architecture.

# Basic 2-tier Client-Server Architectures

- Specialized Servers with Specialized functions: *Print server, File server, DBMS server, Web server, Email server...*

**Figure 2.5**  
Logical two-tier  
client/server  
architecture.



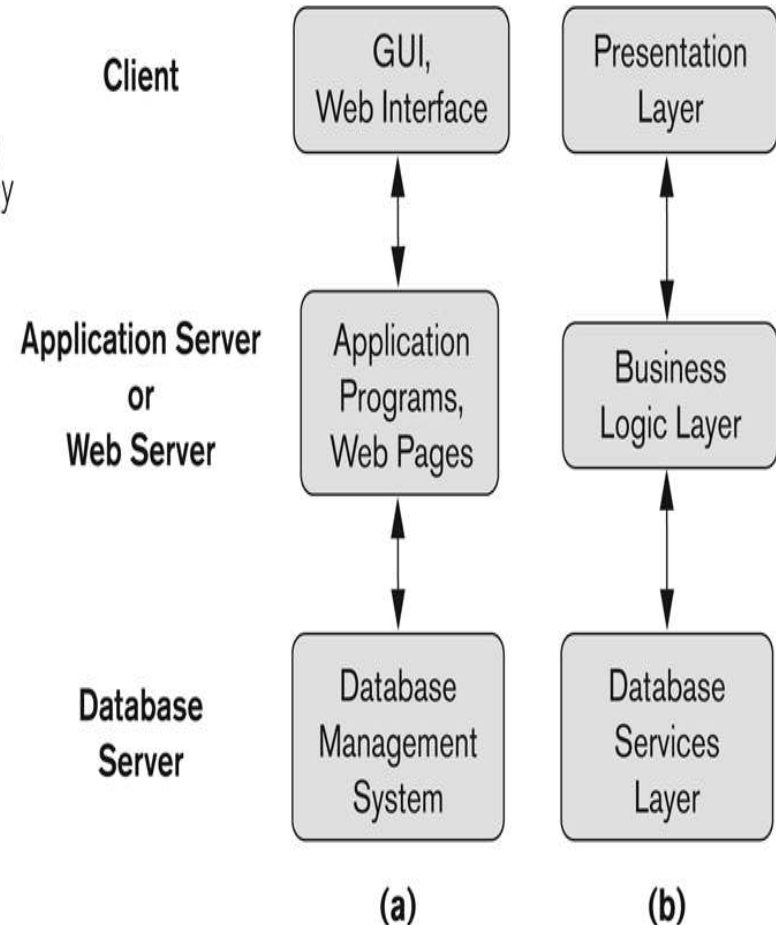
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
  - ODBC: Open Database Connectivity standard
  - JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC
- Variations of clients are possible (tightly vs loosely coupled): e.g., in some object DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

# Three Tier Client-Server Architecture

- Common for Web applications
- Three-tier Architecture Can Enhance Security:
  - Database server only accessible via middle tier (Application Server or Web Server)
  - Clients cannot directly access database server

**Figure 2.7**

Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.



# Classification of DBMSs

- Based on the data model used
  - Traditional: Relational, Network, Hierarchical.
  - Emerging: Object-oriented, Object-relational. XML
- Other classifications
  - Single-user (*typically used with personal computers or embedded dbms*)  
vs. multi-user (*most DBMSs*).
  - Centralized (*uses a single computer with one database*)  
vs. distributed (*uses multiple computers, multiple databases*)
- Homogeneous DDBMS
- Heterogeneous DDBMS
- Federated or Multidatabase Systems
- Distributed Database Systems have now come to be known as client-server based database systems because:
  - They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

# Extension to DB capabilities, New Applications

- DB for Scientific applications
- Image/videos DB
- Data mining
- Spatial/temporal DB
- Information retrieval (*deals with text in general, books, manuscripts, library-based articles*)
- For more generic information about DBMS look at  
[\*http://en.wikipedia.org/wiki/Database\\_management\\_system\*](http://en.wikipedia.org/wiki/Database_management_system)



# Relational Model

- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations and Dealing with Constraint Violations

# General view:

- the relational model consisted of
  - (1) data independence from hardware and storage implementation and
  - (2) automatic navigation, or a high-level, nonprocedural language for accessing data. Instead of processing one record at a time, a programmer could use the language to specify single operations that would be performed across the **entire data set**.
- 2 historical famous relational databases: System R by IBM and Ingres by UC-Berkeley.
- Tables are the basic building blocks.

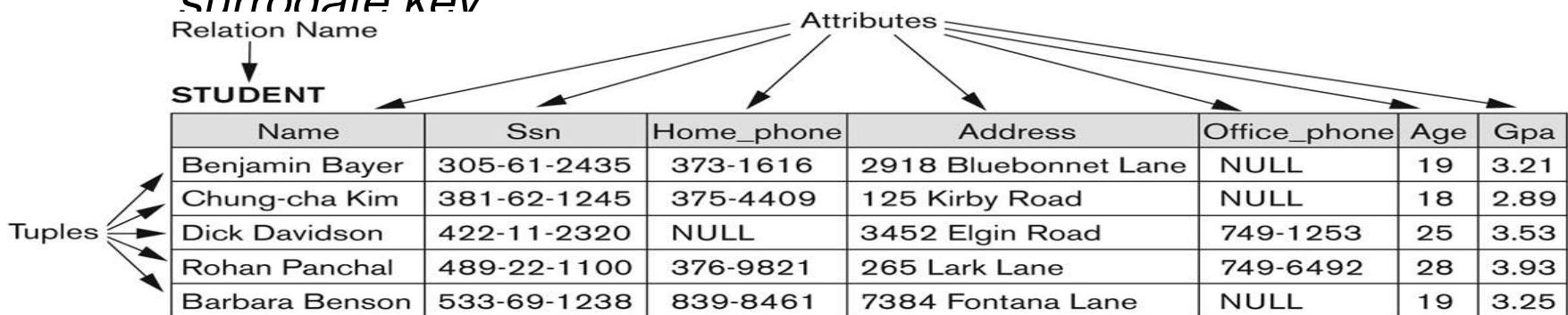


# Relational Model Concepts

- The relational Model of Data is based on the concept of a *Relation*
  - The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations
- We review *the essentials of the formal relational model* .
- In *practice*, there is a *standard model* based on SQL.
- Note: *There are several important differences between the formal model and the practical model.*
- A Relation is a mathematical concept based on the ideas of **sets**
- The model was first proposed by Dr. E.F. Codd of IBM Research in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970
- The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted ACM Turing Award

# Definitions-1

- Informally, a **relation** looks like a **table** of values.
- A relation typically contains a **set of rows**. (*not considered to be ordered*)
- The data elements in each **row** represent certain facts that correspond to a real-world **entity** or **relationship**
  - *In the formal model, rows are called tuples*
- Each **column** has a column header that gives an indication of the meaning of the data items in that column
  - *In the formal model, the column header is called an attribute name (or just attribute)*
- **Key** of a Relation:
  - The key uniquely identifies that row in the table. In the STUDENT table, SSN is the key.
  - Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table. Called *artificial key* or *surrogate key*



**Figure 5.1**

The attributes and tuples of a relation STUDENT.

# Definitions -2

- The **Schema** (or description) of a Relation:
  - Denoted by  $R(A_1, A_2, \dots, A_n)$
  - $R$  is the **name** of the relation, The **attributes** of the relation are  $A_1, A_2, \dots, A_n$
  - Example:  
CUSTOMER (Cust-id, Cust-name, Address, Phone#)  
CUSTOMER is the relation name. Defined over the four attributes: Cust-id, Cust-name, Address, Phone#. Each attribute has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.
- A **relation** is a **set** of such tuples (rows). A **tuple** is an ordered set of values (enclosed in angled brackets ' $< \dots >$ '). Each value is derived from an appropriate *domain*.  
For example: A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - $<632895, \text{"John Smith"}, \text{"101 Main St. Atlanta, GA 30332"}, \text{"(404) 894-2000"}>$
  - This is called a 4-tuple as it has 4 values
  - A tuple (row) in the CUSTOMER relation.

# Definitions -3

- A **domain** has a logical definition:
  - Example: “USA\_phone\_numbers” are the set of 10 digit phone numbers valid in the U.S.
  - A domain has a **data-type** or a **format** defined for it.
    - The USA\_phone\_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
    - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.
  - The attribute name designates the **role** played by a domain in a relation:
    - Used to interpret the meaning of the data elements corresponding to that attribute
    - Example: The domain Date may be used to define two attributes named “Invoice-date” and “Payment-date” with different meanings
- Example: attribute Cust-name is defined over the domain of character strings of maximum length 25
  - $\text{dom}(\text{Cust-name})$  is `varchar(25)`
  - The role these strings play in the CUSTOMER relation is that of the *name of a customer*.
- The **relation state,  $r(R)$**  is a **subset** of the Cartesian product of the domains of its attributes
  - each domain contains the set of all possible values the attribute can take.

## Definitions – *example*: Relational Database Schema

- A set S of relation schemas that belong to the same database.
  - S is the name of the whole **database schema**
  - $S = \{R_1, R_2, \dots, R_n\}$
  - $R_1, R_2, \dots, R_n$  are the names of the individual **relation schemas** within the database S
- Here is the COMPANY database schema with 6 relation schemas:

### EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

### DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

### DEPT\_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

### PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

### WORKS\_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

### DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

**Figure 5.5**  
Schema diagram for  
the COMPANY  
relational database  
schema.

# Definition Summary

<u>Informal Terms</u>		<u>Formal Terms</u>
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

# Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three main types of constraints in the relational model:
  - **Key** constraints (*anahtar kısıtı*)
  - **Entity integrity** constraints (*varlık bütünlük kısıtı*)
  - **Referential integrity** constraints (*ima bütünlük kısıtı*)
- Another implicit constraint is the **domain** constraint
  - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)
- Yet another constraint is Semantic constraints..

# 1-Key Constraints

## ■ Superkey of R:

- Is a set of attributes SK of R with the following condition:
  - No two tuples in any valid relation state  $r(R)$  will have the same value for SK
  - That is, for any distinct tuples  $t1$  and  $t2$  in  $r(R)$ ,  $t1[SK] \neq t2[SK]$
  - This condition must hold in *any valid state*  $r(R)$

## ■ Key of R:

- The only way to access only 1 record.
- A "minimal" superkey
- That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

## ■ Example: Consider the CAR relation schema:

- CAR(State, Reg#, SerialNo, Make, Model, Year)
- CAR has two **keys**:
  - Key1 = {State, Reg#}
  - Key2 = {SerialNo}
- Both are also **superkeys** of CAR
- {SerialNo, Make} is a superkey but *not* a key.



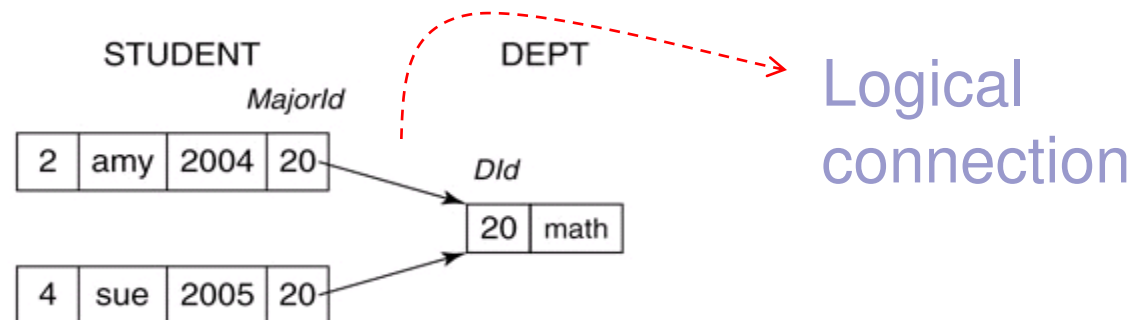
# 2-Entity Integrity

## ■ Entity Integrity:

- The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of  $r(R)$ .
  - This is because primary key values are used to *identify* the individual tuples.
  - $t[PK] \neq \text{null}$  for any tuple  $t$  in  $r(R)$
  - If PK has several attributes, null is not allowed in any of these attributes
- Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

# 3-Referential Integrity

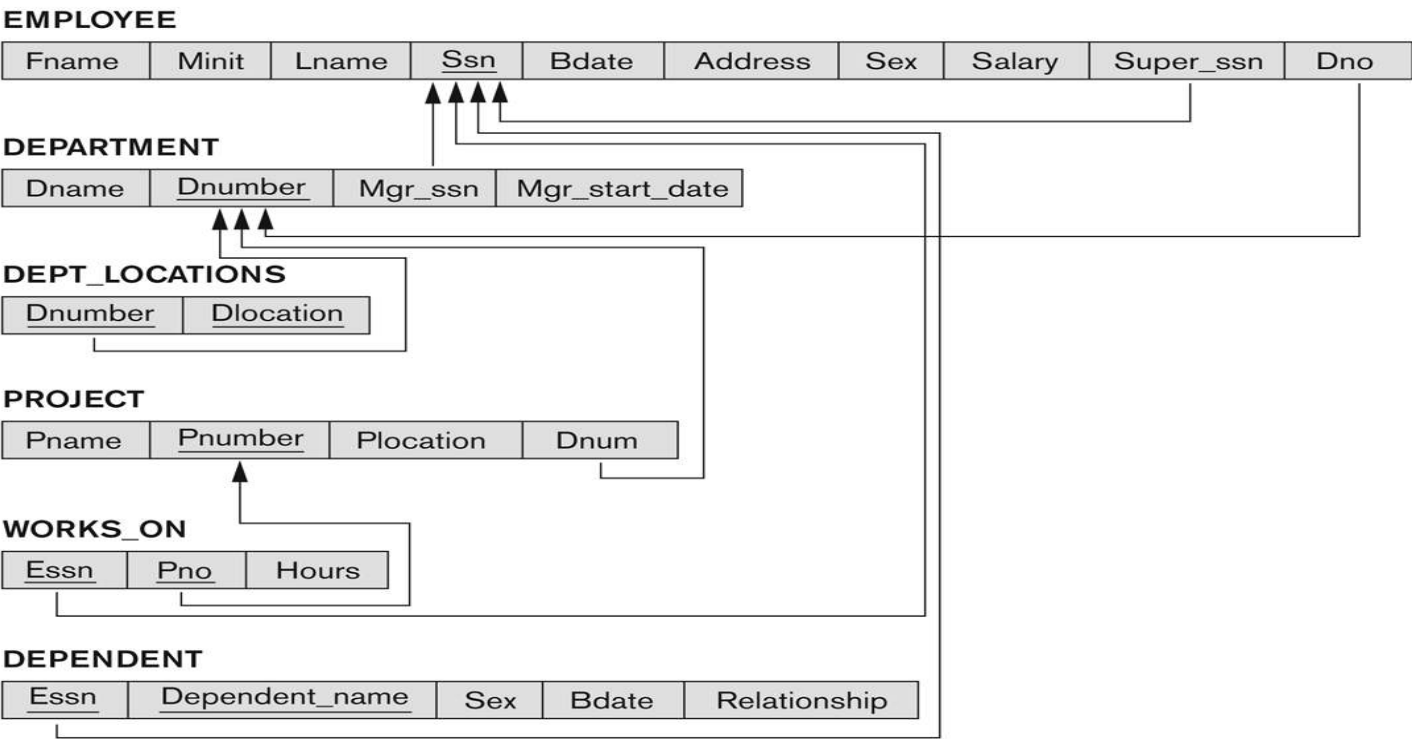
- A constraint involving **two** relations
  - The previous constraints involve a single relation.
- Used to specify a **relationship** among tuples in two relations: The **referencing relation** and the **referenced relation**.
- Tuples in the **referencing relation** R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the **referenced relation** R2.
  - A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if  $t1[FK] = t2[PK]$ .
- **Statement of the constraint**
  - The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be **either**:
    - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, or
    - (2) a **null**.
  - In case (2), the FK in R1 should **not** be a part of its own primary key.



# Displaying a relational database schema and its constraints

- Each relation schema can be displayed as a row of attribute names
- The name of the relation is written above the attribute names
- The primary key attribute (or attributes) will be underlined
- A foreign key (referential integrity) constraints is displayed as a directed arc (arrow) from the foreign key attributes to the referenced table
  - Can also point the the primary key of the referenced relation for clarity
- **COMPANY relational schema diagram:**

**Figure 5.7**  
Referential integrity constraints displayed on the COMPANY relational database schema.



# Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics and cannot be expressed by the model per se
  - Example: “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
- A **constraint specification** language may have to be used to express these. SQL-99 allows **triggers** and **ASSERTIONS** to express for some of these

# Populated database state

- Each *relation* will have many tuples in its current relation state
- The *relational database state* is a union of all the individual relation states
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple
- Next slide shows an example state for the COMPANY database

# Update Operations on Relations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine
- Specifying constraints **in SQL**
  - NOT NULL may be specified on an attribute
  - Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases
  - referential integrity constraints (foreign keys).

# Specifying constraints in SQL

```
CREATE TABLE DEPT (  
    DNAME    VARCHAR(10)    NOT NULL,  
    DNUMBER  INTEGER        NOT NULL,  
    MGRSSN   CHAR(9)        DEFAULT '000' ,  
    MGRSTARTDATE    CHAR(9) ,  
    PRIMARY KEY (DNUMBER) ,  
    UNIQUE (DNAME) ,  
    FOREIGN KEY (MGRSSN) REFERENCES EMP  
        ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

```
CREATE TABLE EMP (  
    ENAME     VARCHAR(30) NOT NULL ,  
    ESSN      CHAR(9) ,  
    BDATE     DATE ,  
    DNO       INTEGER    DEFAULT 1 ,  
    SUPERSSN  CHAR(9) ,  
    PRIMARY KEY (ESSN) ,  
    FOREIGN KEY (DNO) REFERENCES    DEPT ON DELETE SET  
    DEFAULT ON UPDATE    CASCADE ,  
    FOREIGN KEY (SUPERSSN) REFERENCES EMP ON DELETE SET  
        NULL ON UPDATE CASCADE) ;
```

# Possible violations for INSERT

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple



# Possible violations for DELETE

- **DELETE may violate only referential integrity:**
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 8 for more details)
      - RESTRICT option: reject the deletion
      - CASCADE option: remove all referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each foreign key constraint

# Possible violations for UPDATE

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints

**Figure 5.6**

One possible database state for the COMPANY relational database schema.

**EMPLOYEE**

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

**DEPARTMENT**

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

**DEPT\_LOCATIONS**

<u>Dnumber</u>	<u>Dlocation</u>
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

**WORKS\_ON**

<u>Essn</u>	<u>Pno</u>	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

**PROJECT**

Pname	<u>Pnumber</u>	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

**DEPENDENT**

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

# Exercise-2

- Apply following operations to COMPANY database state at the previous slide. Determine the violated integrity constraints.
  - Insert < 'ProductA', 4, 'Bellaire', 2 > into PROJECT. (violates referential integrity)
  - Insert < 'Production', 4, '943775543', '01-OCT-88' > into DEPARTMENT. (Violates both the key constraint and referential integrity)
  - Delete the WORKS\_ON tuples with ESSN= '333445555' (no violation)
  - Delete the EMPLOYEE tuple with SSN= '987654321'. (Violates referential integrity )
  - Modify the SUPERSSN attribute of the EMPLOYEE tuple with SSN= '999887777' to '943775543'. (violates referential integrity)

# RA:relational algebra

- has a similar power with SQL.
- RA has NO formal syntax while SQL is industry standard language.
- RA is task oriented query language while SQL is result-oriented.
- DBMS translate SQL to RA in order to execute it.
- RA query can be expressed as a *query tree* that contains nodes for tables and operators within the query. *query tree* shows execution order and is used in execution&optimization.

# operators

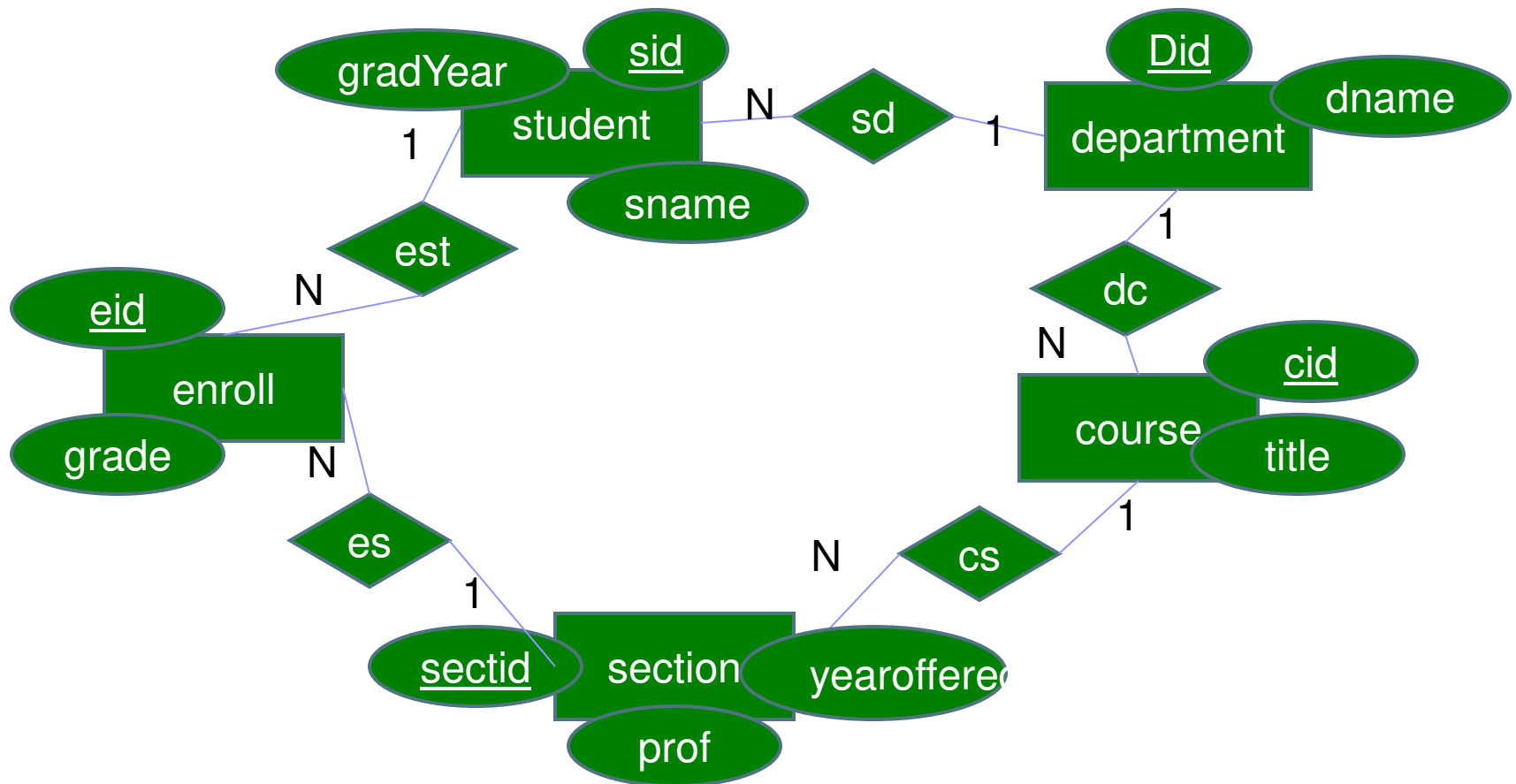
## ■ Single-table operators:

- ☐ Select
- ☐ Project
- ☐ Sort
- ☐ Rename
- ☐ Extend
- ☐ Groupby

## ■ 2-table operators

- ☐ product
- ☐ join
- ☐ semijoin
- ☐ antijoin
- ☐ union
- ☐ outerjoin

# A student record database:



```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

# Student record db state

STUDENT	SIId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

DEPT	DId	DName
	10	compsci
	20	math
	30	drama

COURSE	CIId	Title	DeptId
	12	db systems	10
	22	compilers	10
	32	calculus	20
	42	algebra	20
	52	acting	30
	62	elocution	30

SECTION	SectId	CourseId	Prof	YearOffered
	13	12	turing	2004
	23	12	turing	2005
	33	32	newton	2000
	43	32	einstein	2001
	53	62	brando	2001

ENROLL	EId	StudentId	SectionId	Grade
	14	1	13	A
	24	1	43	C
	34	2	43	B+
	44	4	33	B
	54	4	53	A
	64	6	53	A

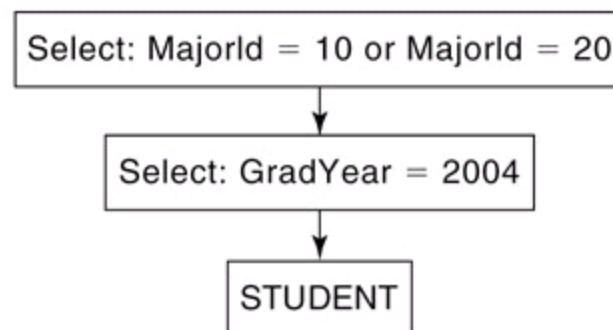
**Figure 1-1**  
Some records for a university database



# select

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

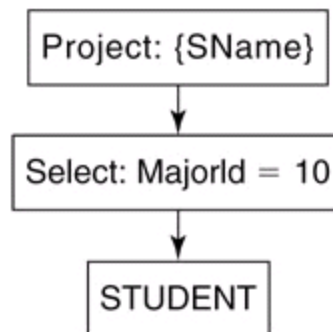
- list students who graduated in 2004:
  - Q1: `select(STUDENT, GradYear=2004)`
- list students who graduated in 2004 with a major of 10 or 20:
  - Q2: `select (STUDENT, GradYear=2004 and (MajorId=10 or MajorId=20))`
  - Q3: `select (select(STUDENT, GradYear=2004), MajorId=10 or MajorId=20)`
  - Q4: `select (Q1, MajorId=10 or MajorId=20)`
- query tree for Q3:



# project

STUDENT(SId, SName, GradYear, MajorId)  
DEPT(DId, DName)  
COURSE(CId, Title, DeptId)  
SECTION(SectId, CourseId, Prof, YearOffered)  
ENROLL(EId, StudentId, SectionId, Grade)

- List the name and graduation year of all students:
  - Q5: project ( STUDENT, {SName,GradYear})
- List the name of all students having major 10
  - Q6: project ( select(STUDENT, MajorId=10), {SName})
- What about the following query?
  - Q7: select ( project (STUDENT, {SName}), MajorId=10)
- Query tree for Q6:



# sort, rename, extend

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Sort the students by graduation year and then name
  - Q8: sort( STUDENT, [GradYear, SName])
    - *Sort operator is typically the outmost operator.*
- Rename the records from Q6 with field *SName* as *CSMajors*
  - Q9: rename(Q6, SName, CSMajors)
    - *rename operator is used in multi-table operators (s.a. Product, join) in order to alter schema of one table to match the other's.*
- Extend STUDENT table to have the calculated field GradClass as the the number years since 1863 (first grad year)
  - Q10: extend (STUDENT, GradYear-1863, GradClass)
- Extend STUDENT table to have the calculated field College as "YTU"
  - Q11: extend (STUDENT, 'YTU', College)

# Group by

STUDENT(SId, SName, GradYear, MajorId)  
DEPT(DId, DName)  
COURSE(CId, Title, DeptId)  
SECTION(SectId, CourseId, Prof, YearOffered)  
ENROLL(EId, StudentId, SectionId, Grade)

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

Q12	MajorId	MinOfGradYear	MaxOfGradYear
	10	2004	2005
	20	2001	2005
	30	2003	2004

- Show max and min graduation year per major
  - Q12: groupby ( STUDENT, {MajorId}, {Min(GradYear), Max(GradYear)})
- Show the number of students in each major per graduation year.
  - Q13: groupby ( STUDENT, {MajorId, GradYear}, {Count(SId)})

Q13	MajorId	GradYear	CountOfSId
	10	2004	2
	10	2005	1
	20	2001	2
	20	2004	1
	20	2005	1
	30	2003	1
	30	2004	1

# Group by

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

- **Show the min graduation year of any student**
  - Q14: `groupby ( STUDENT, {}, {Min(GradYear)})`
- **Show the list of major ids of all students with duplicates removed.**
  - Q15: `groupby ( STUDENT, {MajorId}, {})`
- **Show the number of students having a major.**
  - Q16: `groupby ( STUDENT, {}, {Count(MajorId)})`
    - *Count function count all records including duplicates*
- **Show the number of different majors**
  - Q17: `groupby ( STUDENT, {}, {CountDistinct(MajorId)})`
    - Count and CountDistinct (and other aggregate funcs.) ignore NULL

# Group by

STUDENT(SId, SName, GradYear, MajorId)  
 DEPT(DId, DName)  
 COURSE(CId, Title, DeptId)  
 SECTION(SectId, CourseId, Prof, YearOffered)  
 ENROLL(EId, StudentId, SectionId, Grade)

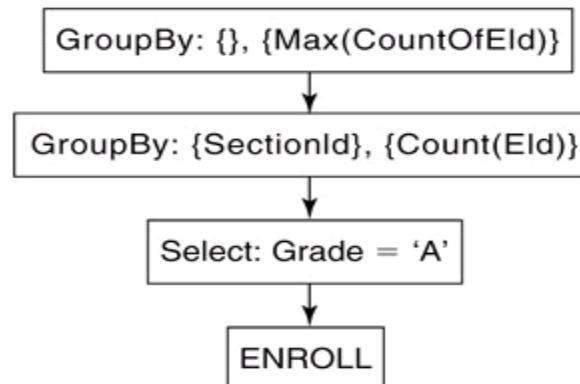
ENROLL	EId	StudentId	SectionId	Grade
	14	1	13	A
	24	1	43	C
	34	2	43	B+
	44	4	33	B
	54	4	53	A
	64	6	53	A

Q18	EId	StudentId	SectionId	Grade
	14	1	13	A
	54	4	53	A
	64	6	53	A

Q19	SectionId	CountOfEId
	13	1
	53	2

Q20	MaxOfCountOfEId
	2

- Show the most number of A's given in any section
  - 3 steps:
    - Q18: select ( ENROLL, Grade='A')
    - Q19: groupby (Q18, {SectionId},{Count(EId)})
    - Q20: groupby (Q19,{},{Max(CountOfEId)})
  - 1 step:
    - Q21: groupby (groupby (select ( ENROLL, Grade='A') , {SectionId},{Count(EId)}), {},{Max(CountOfEId)})
  - Query tree for Q21:



# product

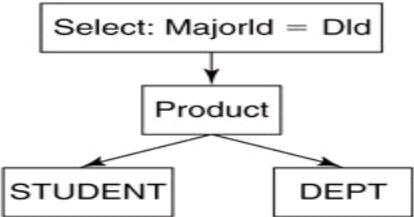
STUDENT(SId, SName, GradYear, MajorId)  
DEPT(DId, DName)  
COURSE(CId, Title, DeptId)  
SECTION(SectId, CourseId, Prof, YearOffered)  
ENROLL(EId, StudentId, SectionId, Grade)

- All combinations of records from STUDENT and DEPT
  - Q22: product (STUDENT,DEPT)

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

DEPT	DId	DName
	10	compsci
	20	math
	30	drama

- All students and their major department's name
  - Q23: select( product (STUDENT,DEPT), MajorId=DId)



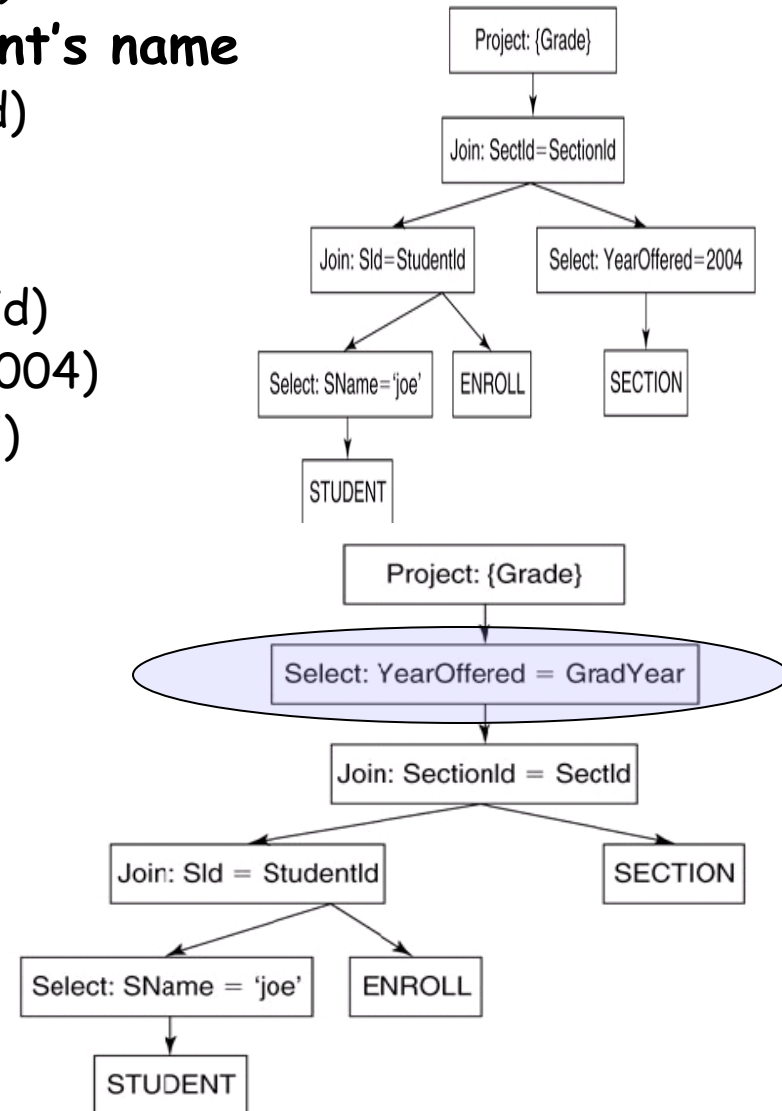
Q22	SId	SName	MajorId	GradYear	DId	DName
	1	joe	10	2004	10	compsci
	2	amy	20	2004	10	compsci
	3	max	10	2005	10	compsci
	4	sue	20	2005	10	compsci
	5	bob	30	2003	10	compsci
	6	kim	20	2001	10	compsci
	7	art	30	2004	10	compsci
	8	pat	20	2001	10	compsci
	9	lee	10	2004	10	compsci
	1	joe	10	2004	20	math
	2	amy	20	2004	20	math
	3	max	10	2005	20	math
	4	sue	20	2005	20	math
	5	bob	30	2003	20	math
	6	kim	20	2001	20	math
	7	art	30	2004	20	math
	8	pat	20	2001	20	math
	9	lee	10	2004	20	math
	1	joe	10	2004	30	drama
	2	amy	20	2004	30	drama
	3	max	10	2005	30	drama
	4	sue	20	2005	30	drama
	5	bob	30	2003	30	drama
	6	kim	20	2001	30	drama
	7	art	30	2004	30	drama
	8	pat	20	2001	30	drama
	9	lee	10	2004	30	drama




# join

STUDENT(SId, SName, GradYear, MajorId)  
DEPT(DId, DName)  
COURSE(CId, Title, DeptId)  
SECTION(SectId, CourseId, Prof, YearOffered)  
ENROLL(EId, StudentId, SectionId, Grade)

- $\text{join}(T1, T2, P) \equiv \text{select}(\text{product}(T1, T2), P)$
- All students and their major department's name
  - Q24:  $\text{join}(\text{STUDENT}, \text{DEPT}, \text{MajorId}=\text{DId})$
- The grades Joe received during 2004
  - Q25:  $\text{select}(\text{STUDENT}, \text{SName}='Joe')$
  - Q26:  $\text{join}(\text{Q25}, \text{ENROLL}, \text{SId}=\text{StudentId})$
  - Q27:  $\text{select}(\text{SECTION}, \text{YearOffered}=2004)$
  - Q28:  $\text{join}(\text{Q26}, \text{Q27}, \text{SectId}=\text{SectionId})$
  - Q29:  $\text{project}(\text{Q27}, \text{Grade})$
  - Is there a more efficient way?
- Find the grades Joe received during his graduation year. (remember Q29)
  - $\text{select YearOffered}=\text{GradYear}$ , only after joining with SECTION
  - There are 3 predicates to join 3 tables, but only 2 is enough.
  - Named as circular query







```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

# Join

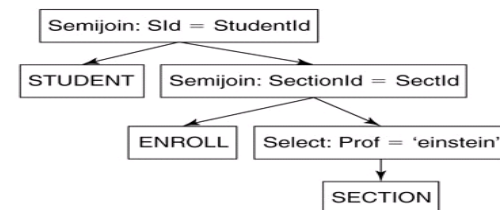
- **The sections that gave out the most A's. (remember Q20)**
  - Q30: join (Q20, Q19, MaxCountOfEId= CountOfEId)
  - Q31: join (Q30,SECTION,SectionId=SectId)
- **List students that have the same major as Joe.**
  - Q32: project ( select ( STUDENT, SName='Joe'), MajorId)
  - Q33: rename( Q32, MajorId, JoesMajor)
  - Q34: join (Q33, STUDENT, MajorId = JoesMajorId)
  - This type of join is named as self-join (or recursive).
  - We have to rename in order to write predicate correctly.

# semijoin

STUDENT(SId, SName, GradYear, MajorId)  
 DEPT(DId, DName)  
 COURSE(CId, Title, DeptId)  
 SECTION(SectId, CourseId, Prof, YearOffered)  
 ENROLL(EId, StudentId, SectionId, Grade)

- Q24 lists pairs of matching records. Sometimes, we only need it there is match.
- List departments that have at least 1 student in their major.
  - Q35: semijoin (DEPT,STUDENT,DId=MajorId)
    - *Q35 consists of those records from the first table that match some record in the second table.*
  - Equivalent join:
    - Q36: join (DEPT,STUDENT,DId=MajorId)
    - Q37: groupby (Q36, {DId,DName},{})
      - Groupby operator projects and then removes duplicates..
- The students who took a course from Profesor Einstein.

- Q38: select (SECTION, Prof='Einstein')
- Q39: semijoin ( ENROLL, Q38, SectionId = SectId)
- Q40: semijoin ( STUDENT, Q39, StudentId = SId)



SECTION	SectId	CourseId	Prof	YearOffered
	13	12	turing	2004
	23	12	turing	2005
	33	32	newton	2000
	43	32	einstein	2001
	53	62	brando	2001

ENROLL	EId	StudentId	SectionId	Grade
	14	1	13	A
	24	1	43	C
	34	2	43	B+
	44	4	33	B
	54	4	53	A
	64	6	53	A

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	arr	2004	30
	8	pat	2001	20
	9	lee	2004	10

# antijoin

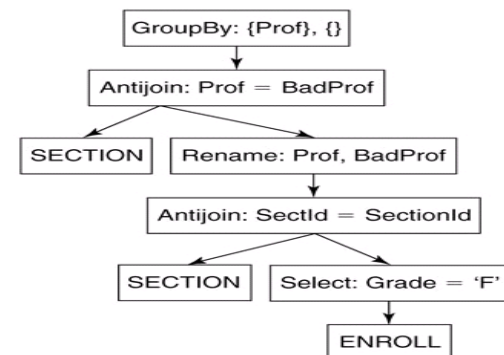
```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Opposite of semijoin; thus returns exactly those records that the semijoin omits.
- Antijoin cannot be simulated by a join.
- **List departments that have no students majors.**
  - Q41: antijoin ( DEPT, STUDENT, DId=MajorId)
    - *Note that union of Q35 and Q41 is DEPT and Q35 and Q41 has no common record.*
- **Find the sections where nobody received a grade of 'F'.**
  - Q42: select (ENROLL, Grade = 'F')
  - Q43: antijoin (SECTION,Q42, SectId = SectionId)
  - What about the following solution?
    - Join ( Select (ENROLL, Grade <> 'F'), SECTION, SectId=SectionId)
- Antijoin is used in the following type of queries:
  - "NOT EXIST x" type
  - "FOR ALL x" = "NOT EXIST NOT x"
- **Find those sections in which everyone got an 'A'.** (*sections in which there does not exist someone who did not get an 'A'*)
  - select (ENROLL, Grade <> 'A')
  - antijoin (SECTION,Q42, SectId = SectionId)

# antijoin

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Find professors who have never given an 'F'.
  - Find Sections that have an 'F'.  
Exclude those sections' Profs from all professors.
  - Q44: select (ENROLL, Grade = 'F')  
Q45: semijoin (SECTION, Q44, SectId = SectionId)  
Q46: rename (Q45, Prof, BadProf) // find sections where an 'F' was given.  
Q47: antijoin (SECTION, Q46, Prof = BadProf )  
Q48: groupby (Q47, {Prof}, {}) //removes duplicates
- Find professors who gave at least one 'F' in every section they taught.
  - Find sections where nobody received a grade of 'F'. Q43 did this.  
Find those professors who did not give any section within Q43.
  - Q49: rename (Q43, Prof, GoodProf)  
Q50: antijoin (SECTION, Q49, Prof = GoodProf)  
Q51: groupby (Q50, {Prof}, {})



```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Requires both table has the same schema
- Find the combined names of students and professors
  - Q52: rename( `project(STUDENT, {Sname})`, SName, Person)
  - Q53: rename( `project(SECTION, {Prof})`, Prof, Person)
  - Q54: union (Q52,Q53)

# outerjoin


STUDENT(SId, SName, GradYear, MajorId)  
 DEPT(DId, DName)  
 COURSE(CId, Title, DeptId)  
 SECTION(SectId, CourseId, Prof, YearOffered)  
 ENROLL(EId, StudentId, SectionId, Grade)

- Same arguments as the join
- Result table has the records of the join unioned with non-matching records from each of the tables.
- The outerjoin of STUDENT and ENROLL
  - Q55: outerjoin (STUDENT, ENROLL, SId=StudentId)

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

ENROLL	EId	StudentId	SectionId	Grade
	14	1	13	A
	24	1	43	C
	34	2	43	B+
	44	4	33	B
	54	4	53	A
	64	6	53	A

Q55	SId	SName	MajorId	GradYear	EId	StudentId	SectionId	Grade
	1	joe	10	2004	14	1	13	A
	1	joe	10	2004	24	1	43	C
	2	amy	20	2004	34	2	43	B+
	4	sue	20	2005	44	4	33	B
	4	sue	20	2005	54	4	53	A
	6	kim	20	2001	64	6	53	A
	3	max	10	2005	null	null	null	null
	5	bob	30	2003	null	null	null	null
	7	art	30	2004	null	null	null	null
	8	pat	20	2001	null	null	null	null
	9	lee	10	2004	null	null	null	null



```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

# outerjoin

- Show the number of enrollments for each student that has taken at least one course
  - Q56: `groupby (ENROLL, {StudentId}, {count(EId)})`
- Show the number of enrollments for all students
  - Q57: `outerjoin (STUDENT, ENROLL, SId=StudentId)`
  - Q58: `groupby ( Q57, {SId}, {count(EId)})` // *remember: count ignores NULL*



# SQL (*Structured Query Language*)

- History:
  - SQL1, @1989
  - SQL2=SQL92 (*create schema, referential integrity options and additional data types (s.a. DATE, TIME, and TIMESTAMP) are added..*)
  - SQL3=SQL99 (*object-relational database concepts, call level interfaces, and integrity management*)
- *SQL is based on*
  - *Relational algebra:*
    - *introduced by E. F. Codd in 1972,*
    - *Provide the basic concepts behind computing SQL syntax.*
    - *It is a procedural way to construct data-driven queries*
    - *it addresses the **how** logic of a structured query.*
  - *The tuple relational calculus ( TRC )*
    - *had a large effect on the underlying appearance of SQL.*
    - *uses declarative expressions,*
    - *address the **what** logic of a structured query.*
- Additionally SQL provides
  - insertion, modification and deletion.
  - Arithmetic operators
  - Display of data
  - Assignment
  - Aggregate functions



# SQL: Data Definition, Constraints, and Schema Changes

- Used to CREATE, DROP, and ALTER the descriptions of the tables (relations) of a database
- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- Specifying constraints
  - NOT NULL may be specified on an attribute
  - Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases
  - referential integrity constraints (foreign keys).

```
CREATE TABLE DEPT (  
  DNAME    VARCHAR(10)  NOT NULL,  
  DNUMBER  INTEGER      NOT NULL,  
  MGRSSN   CHAR(9)     DEFAULT '000',  
  MGRSTARTDATE CHAR(9),  
  PRIMARY KEY (DNUMBER),  
  UNIQUE (DNAME),  
  FOREIGN KEY (MGRSSN) REFERENCES EMP  
    ON DELETE SET DEFAULT  
    ON UPDATE CASCADE);
```

```
CREATE TABLE EMP(  
  ENAME      VARCHAR(30) NOT NULL,  
  ESSN       CHAR(9),  
  BDATE      DATE,  
  DNO        INTEGER  DEFAULT 1,  
  SUPERSSN   CHAR(9),  
  PRIMARY KEY (ESSN),  
  FOREIGN KEY (DNO) REFERENCES DEPT ON  
    DELETE SET DEFAULT ON UPDATE CASCADE,  
  FOREIGN KEY (SUPERSSN) REFERENCES EMP  
    ON DELETE SET NULL ON UPDATE CASCADE);
```

# DROP TABLE / ALTER TABLE

## ■ DROP TABLE

- Used to remove a relation (base table) and its definition
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example: **DROP TABLE DEPENDENT;**

## ■ ALTER TABLE

- Used to add an attribute to one of the base relations
  - The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is not allowed for such an attribute
- Example:  
**ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);**
- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple.
  - This can be done using the UPDATE command.

# Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the **SELECT** statement
  - This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model:
  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
  - Hence, an SQL relation (table) is a **multi-set** (sometimes called a **bag**) of tuples; it is *not* a set of tuples. Example: {A, B, C, A} is a bag. {A, B, C} is also a bag that also is a set. Bags also resemble lists, but the order is irrelevant in a bag.
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

# SQL Types: numeric, string, date

Type Name	Flavor	Precision	Scale	Sample Constant
NUMERIC(5,3)	exact	5 digits	3	31.416
INTEGER or INT	exact	9 digits	0	314159265
SMALLINT	exact	4 digits	0	3142
FLOAT(3)	approximate	3 bits	--	3.5
FLOAT	approximate	24 bits	--	3.1415926
DOUBLE PRECISION	approximate	53 bits	--	3.141592653589793

**Figure 4-14**  
Numeric types in SQL

VARCHAR(n), CHAR(n)

DATE '2008-07-04' = 4 July, 2008

INTERVAL '5' DAY

Function Name	Meaning	Example Usage	Result
<i>current_date</i>	Return the current date.	<code>current_date</code>	Today's date
<i>extract</i>	Extract the year, month, or day from a date.	<code>extract(month, date '2008-07-04')</code>	7
+	Add an interval to a date.	<code>date '2008-07-04' + interval '7' month</code>	date '2009-02-04'
-	Subtract an interval from a date or subtract two dates.	<code>date '2008-07-04' - date '2008-06-30'</code>	interval '5' day

**Figure 4-16**  
Some common SQL date/interval functions

Function Name	Meaning	Example Usage	Result
<i>lower</i> (also, <i>upper</i> )	Turn the characters of the string into lower case (or upper case).	<code>lower('Einstein')</code>	'einstein'
<i>trim</i>	Remove leading and trailing spaces.	<code>trim(' Einstein ')</code>	'Einstein'
<i>char_length</i>	Return the number of characters in the string.	<code>char_length('Einstein')</code>	8
<i>substring</i>	Extract a specified substring.	<code>substring('Einstein' from 2 for 3)</code>	'ins'
<i>current_user</i>	Return the name of the current user.	<code>current_user</code>	'einstein' (assuming that he is currently logged in)
	Catenate two strings.	<code>'A. '    'Einstein'</code>	'A. Einstein'
<i>like</i>	Match a string against a pattern.	<code>'Einstein' like '_i%i_%'</code>	true

**Figure 4-15**  
Some common SQL string functions

# select

- Calculate a new Id and the graduation decade for each student

- Q68: select 'Student #' || s.SId AS NewSId, s.SName,  
cast(s.GradYear/10 as int) \* 10 AS GradDecade  
from STUDENT s

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

Q68	NewSId	SName	GradDecade
	Student #1	joe	2000
	Student #2	amy	2000
	Student #3	max	2000
	Student #4	sue	2000
	Student #5	bob	2000
	Student #6	kim	2000
	Student #7	art	2000
	Student #8	pat	2000
	Student #9	lee	2000

- Calculate the number of years since graduation for each student
  - Q69: select s.\*, extract(YEAR,current\_date)-s.GradYear AS AlumYears  
from STUDENT s;
- Determine if a student has graduated
  - Q71: select q.\*, if(q.AlumYrs>0,'alum', 'in school') AS GradStats  
from Q69 q;

STUDENT(SId, SName, GradYear, MajorId)  
 DEPT(DId, DName)  
 COURSE(CId, Title, DeptId)  
 SECTION(SectId, CourseId, Prof, YearOffered)  
 ENROLL(EId, StudentId, SectionId, Grade)

from

- All combinations of STUDENT and DEPT records..

- Q73: select s.\*,d.\*  
 from STUDENT s,DEPT d

DEPT	DId	DName
	10	comp sci
	20	math
	30	drama

- All pairs of STUDENT names

- Q74: select s1.SNAME as name1, s2.SName as name2  
 from STUDENT s1, STUDENT s2

- The join of STUDENT and DEPT

- Q75: select s.SName, d.DName  
 from STUDENT s join DEPT d ON s.MajorId = d.DId


STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

- Outer join lists all records even it has no match.

select s.SName, d.DName  
 from STUDENT s full join DEPT d ON s.MajorId=d.DId

- The names of students and their professors

- Q76: select s.SName, k.Prof  
 from (STUDENT s join ENROLL e ON s.SId=e.StudentId) join  
 SECTION k on e.SectionId = k.SectId

 <h1>where</h1>	STUDENT(SId, SName, GradYear, MajorId) DEPT(DId, DName) COURSE(CId, Title, DeptId) SECTION(SectId, CourseId, Prof, YearOffered) ENROLL(EId, StudentId, SectionId, Grade)

- **The names of students graduating in 2005 or 2006.**
  - Q77: Select s.SName  
From STUDENT s  
Where (s.GradYear=2005) or (s.GradYear=2006)
- **The names of students and their professors (*compare with Q76*)**
  - Q78: select s.SName, k.Prof  
from STUDENT s, ENROLL e, SECTION k  
where s.SId=e.StudentId and e.SectionId = k.SectId
- **Find the grades Joe received during his graduation year. (*remember Q29*)**
  - Q79: select e.Grade  
from STUDENT s, ENROLL e, SECTION k  
where s.SId=e.StudentId and e.SectionId = k.SectId and  
k.YearOffered=s.GradYear and s.SName='Joe';
- **The students whose names begin with 'j' and graduate this year**
  - Q80: select s.\*  
from STUDENT s  
where s.SName like 'j%' and s.GradYear=extract(YEAR,current\_date)
- **Grades given by the current professor-user**
  - Q79: select s.SName, e.Grade  
from STUDENT s, ENROLL e, SECTION k  
where s.SId=e.StudentId and e.SectionId = k.SectId and  
k.Prof=current\_user;



# groupby

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

1. Grouping fields are in group by clause. Aggregate functions are in select clause
  2. The order of execution:  
FROM → WHERE → GROUP BY → SELECT
  3. The only non-computed fields allowed in the select are grouping fields.
  4. Thus; empty group by clause AND select clause with aggregate function can only show the result of aggregate function.
- **The minimum and maximum graduation year per major (remember Q12)**
    - Q82: select s.MajorId, min (GradYear), max (GradYear)  
from STUDENT s  
group by s.MajorId
  - **Find the section that gave out the most 'A' grades. (remember Q18-20)**

Q83: select e.SectionId, count (EId) AS numAs  
from ENROLL e  
where e.Grade='A'  
group by e.SectionId

**The maximum number of A's given in any section:**

Q84: select max (q.numAs) AS maxAs  
from Q83 q

**Find the section that gave out the most 'A' grades.**

Q85: select Q83.SectionId  
from Q83, Q84  
where Q83.numAs=Q84.maxAs

**illegal**

**Find the section that gave out the most 'A' grades**

Q84a: select q.SectionId, max (q.numAs) as maxAs  
from Q83 q

**Find the section that gave out the most 'A' grades.**

Q84b: select q.SectionId  
from Q83 q  
where q.numAs=max (q.numAs) **80**



# group by

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Show the name of each student and the number of courses they took.
  - Q86: select e.StudentId, count (EId) AS HowMany  
from ENROLL e  
group by e.StudentId
  - Q87: select s.SName,q.HowMany  
from Q86 q, STUDENT s  
where q.StudentId=s.StudentId
  - Q87 is technically correct. But does not show students that have not yet taken any course.  
Thus, we need outerjoin:  
Q87a: select s.SName,q.HowMany  
from STUDENT s **full join** Q86 q ON q.StudentId=s.StudentId
  - Q87a is technically correct. But does not show 0 number of course for students that have not yet taken any course. It shows NULL. Thus, we do outerjoin before counting.  
Q88: select e.SName, count (EId) AS HowMany  
from STUDENT s **full join** ENROLL e ON s.SId=e.StudentId  
group by s.SName
  - Q88 is technically correct. But it groups records by student name in order to show it in the output. In case of any student having the same name, the result will be erroneous.  
Q89: select e.StudentId, count (EId) AS HowMany  
from STUDENT s **full join** ENROLL e ON s.SId=e.StudentId  
group by s.SId, s.SName // has no difference with group by s.SId

STUDENT(SId, SName, GradYear, MajorId)  
DEPT(DId, DName)  
COURSE(CId, Title, DeptId)  
SECTION(SectId, CourseId, Prof, YearOffered)  
ENROLL(EId, StudentId, SectionId, Grade)

## group by

- List of all major department names of the grad year of 2004, with no duplicates.

- Q90: select d.DName  
from STUDENT s, DEPT d  
where s.MajorId = d.DId and s.GradYear=2008  
group by d.DName *// used to remove duplicates!*

- *//alternate version of Q90*

- Q91: select distinct d.DName  
from STUDENT s, DEPT d  
where s.MajorId = d.DId and s.GradYear=2008

- Show the number of different majors that students have taken.

- Q92: select count (distinct s.MajorId)  
from STUDENT s

# having

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- List professors who taught more than 4 sections in 2008.

- Q93: select k.Prof, count(k.SectId) as howMany  
from SECTION k  
where k.YearOffered=2008 and howMany>4  
group by k.Prof

illegal

- Q94: select k.Prof, count(k.SectId) as howMany  
from SECTION k  
where k.YearOffered=2008  
group by k.Prof

Q95: select q.Prof, q.howMany  
from Q94 q  
where q.howMany>4

- Q96: select k.Prof, count(k.SectId) as howMany  
from SECTION k  
where k.YearOffered=2008  
group by k.Prof  
having count(k.SectId)>4

- The order of execution:

FROM → WHERE → GROUP BY → HAVING → SELECT

# Nested queries

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Nested queries is used to express semijoin and antijoin.
- $\text{semijoin}(T1, T2, A=B) \equiv \text{select } * \text{ from } T1 \text{ where } A \text{ IN } (\text{select } B \text{ from } T2)$
- $\text{antijoin}(T1, T2, A=B) \equiv \text{select } * \text{ from } T1 \text{ where } A \text{ NOT IN } (\text{select } B \text{ from } T2)$
- Determine the departments having at least one major (*remember Q35*)
  - Q97: 

```
select d.*
from DEPT d
where d.DId IN (select s.MajorId
                from STUDENT s )
```
- Determine those students who took a course with Prof. Einstein. (*remember Q38-40*)
  - Q98: 

```
select s.*
from STUDENT s
where s.SId IN
      (select e.StudentId
       from ENROLL e
       where e.SectionId IN
            (select k.SectId
             from SECTION k
             where k.Prof='einstein'))
```

# Nested queries:

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- List the sections where nobody received an 'F'.  
(remember Q42-43)

- Q99: select  
from SECTION k  
where k.SectId NOT IN  
(select e.SectionId  
from ENROLL e  
where e.Grade ='F')

## union:

- Combined names of students and professors  
(remember Q54)

- Q100: select s.SName as Person from STUDENT s  
union  
select k.Prof as Person from SECTION k

# order by

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- The order of execution:

FROM → WHERE → GROUP BY → HAVING → SELECT → UNION → ORDER BY

- The number of math majors per graduation year sorted by count descending and then year.

- Q101: select s.GradYear, count(SId) as howMany  
from STUDENT s  
group by s.GradYear  
order by howMany DESC, s.GradYear

- The combined names of students and professors in sorted order  
(Sorted version of Q100)

- Q100: select s.SName as Person from STUDENT s  
union  
select k.Prof as Person from SECTION k  
order by Person

# SQL update

1. insert 1 new record by giving its values
  2. insert multiple new record by specifying a query
  3. Delete records
  4. Modify the contents of records
- insert into STUDENT (SId,SName,MajorId,GradYear)  
values (10, 'ron', 30, 2009)
  - create table ALUMNI(SId int,SName varchar(10),Major varchar(8),GradYear int)  
insert into ALUMNI (SId , SName, Major , GradYear )  
select s.SId , s.SName, d.DName , s.GradYear  
from STUDENT s, DEPT d  
where s.MajorId = d.DId and s.GradYear<extract (YEAR, current\_date)
  - insert into STUDENT (SId , SName, Major , GradYear )  
select 22 AS SId , 'jon' AS SName, s.MajorId , s.GradYear  
from STUDENT s  
where s.SId=1
  - delete from SECTION where SectID NOT IN (select e.SectionId from ENROLL e)
  - delete from SECTION
  - update STUDENT set MajorId=10, GradYear=GradYear+1 where MajorId=20

# Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

SELECT	<attribute list>
FROM	<table list>
[WHERE	<condition>]
[GROUP BY	<grouping attribute(s)>]
[HAVING	<group condition>]
[ORDER BY	<attribute list>]

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- The order of execution:  
FROM → WHERE → GROUP BY → HAVING → SELECT → UNION → ORDER BY



# SQL regular views

```

STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)

```

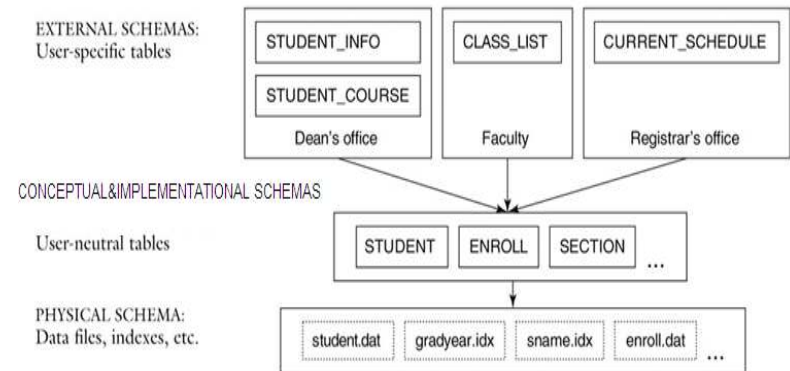
- Regular View:
  - named query
  - Store only definition, real data is not stored...
  - Used to hold subqueries

- create view Q65 as  
select s.SName, s.GradYear  
from STUDENT s

- select q.SName  
from Q65 q  
where q.GradYear=2004

- select q.SName  
from (select s.SName,  
s.GradYear from STUDENT  
s) q  
where q.GradYear=2004

- Views are used for logical data independence.
- External schema consists of views.



- **STUDENT\_INFO** (SId, SName, GPA, NumCoursesPassed, NumCoursesFailed)
- **STUDENT\_COURSES** (SId, YearOffered, Title, Prof, Grade)
- create view **STUDENT\_COURSES** as  
select e.StudentId as SId, k.YearOffered, c.Title,  
k.Prof, e.Grade  
from ENROLL e, SECTION k, COURSE c  
where e.SectionId=k.SectId and k.CourseId=c.CId

# Regular View Updates

- The records in the view do not really exist, they are computed from other records in db.
- If every record  $r$  has a unique corresponding record  $r'$  in some underlying db table, then **view is updatable**.
- In general;
  - Single table views except those containing grouping are updatable.
  - Multi-table views are NOT updatable.
- create view StudentMajor as

```
select s.SName, d.DName
from STUDENT s, DEPT d
where s.MajorId=d.DId
```

  - Delete record {'sue', 'math'}
    - Delete 'sue' record from STUDENT
    - Delete 'math' record from DEPT
    - Update 'MajorId' of 'sue' record in STUDENT
  - Insert record {'joe', 'drama'}
    - Update 'joe's MajorId to 30
    - Insert a new 'joe' record into STUDENT with having a MajorId=10 (*this is valid because SName is not key*)
- Since db update via views is awkward, STORED PROCEDURES are more prevalent.

DEPT	DId	DName
	10	compsci
	20	math
	30	drama

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10