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

COURSE	CId	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

Table/File

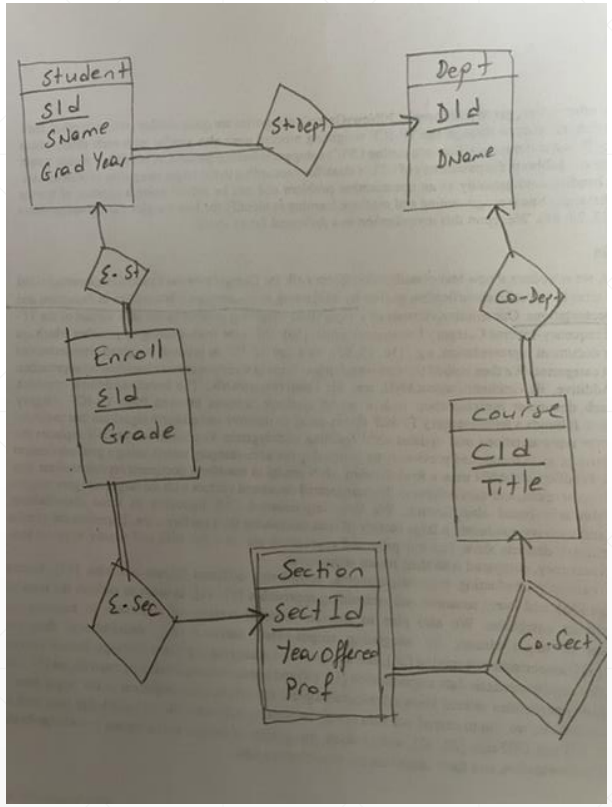
Record/Row

Column/Attribute

Key (Key constraint)

Foreign Key (Referential Integrity)

University Database ERD & Relational Schema



Entities become tables
Student (Sid, Sname, GradYear,)
Dept (Did, Dname)
Section (CourseId, SectId, YearOffered, Prof)
Course (Cid, Title)
Enroll (Eid, Grade)

Relationships become tables
Co-Dept (Did, CId)
St-Dept (Did, Sid)
E-St (Eid, Sid)
E-Sec (SectId,Eid)

Co-Sec does not become a table CourseId is added to PK of Section since Section is weak entity

One Cardinality Reductions

Some tables receive foreign keys
Student (Sid, Sname, GradYear, MajorId) 2
Dept (Did, Dname)
Section (CourseId, SectId, YearOffered, Prof)
Course (Cid, Title, DeptId) 1
Enroll (Eid, Grade, StudentId, SectionId) 3 &4

Some tables are removed
~~Co-Dept (Did, CId)~~ 1
~~St-Dept (Did, Sid)~~ 2
~~E-St (Eid, Sid)~~ 3
~~E-Sec (SectId,Eid)~~ 4

Queries

- **Relational Algebra**
  - The foundation upon which real query languages like SQL based
- **SQL**
  - The industry standard query language
- Every SQL query has a corresponding relational algebra query.

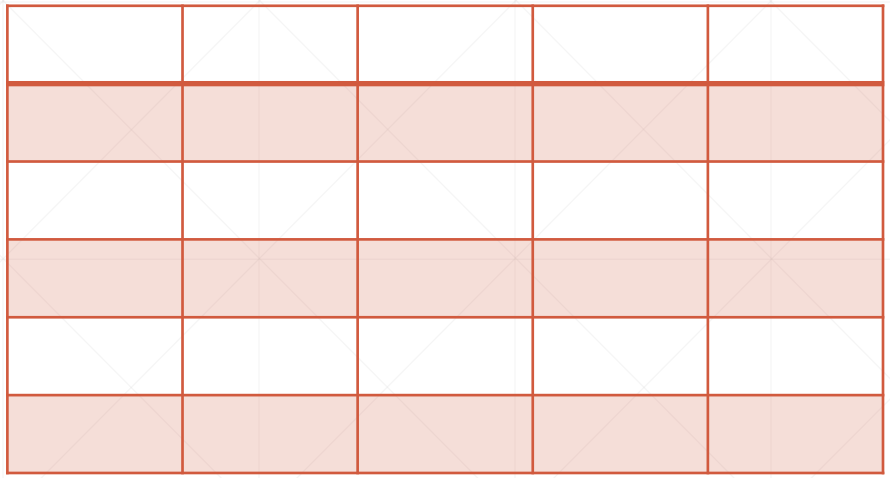
Relational Algebra

- A relational algebra query is composed of ***operators***.
- Each operator performs one specialized task.
  - Taking one or more tables as input
  - Producing one output table
- The composition of operators in a query can be written as a query tree.

Relational Algebra (Cont.)

- The single-table operators are:
  - **Select**
  - **Project**
  - **Sort**
  - **Rename**
  - **Extend**
  - **Groupby**

Select



- select (input_table, predicate)
- The output table has the same columns as its input table, but with some rows removed.
- Q1 = select(STUDENT, GradYear=2004)
- 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)

Select

- Q3 = select(select(STUDENT, GradYear=2004), MajorId=10 or MajorId=20)

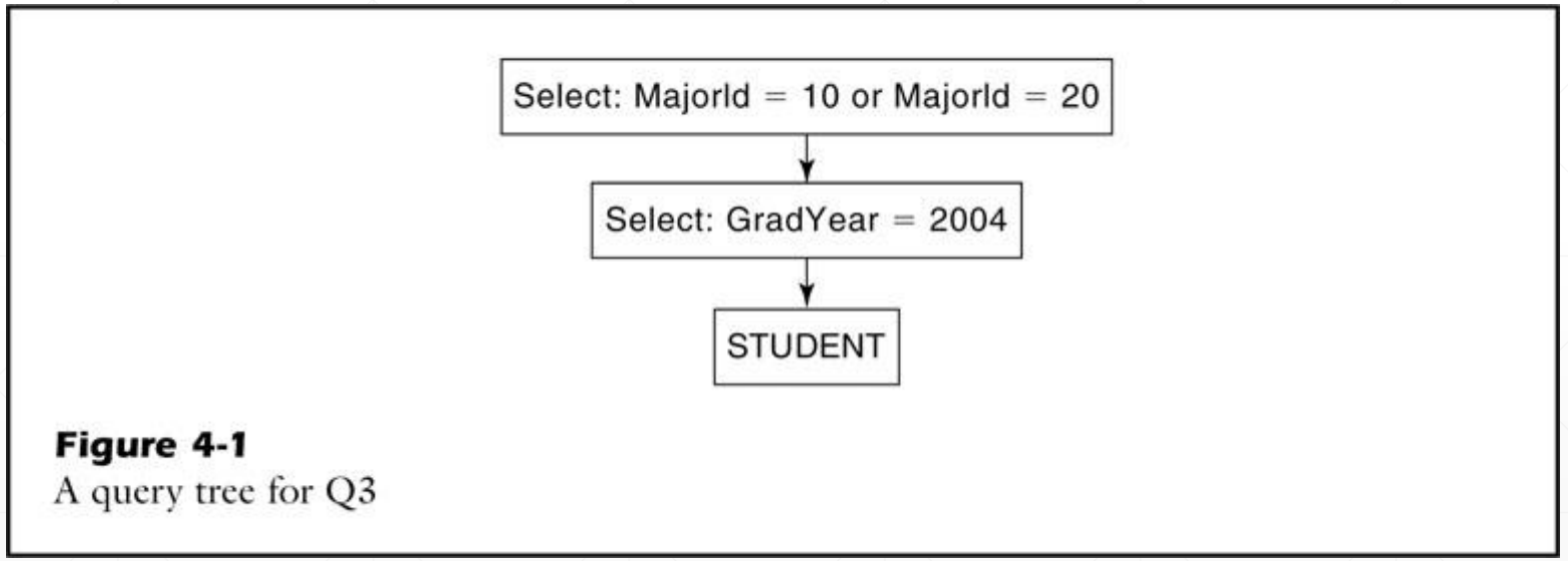
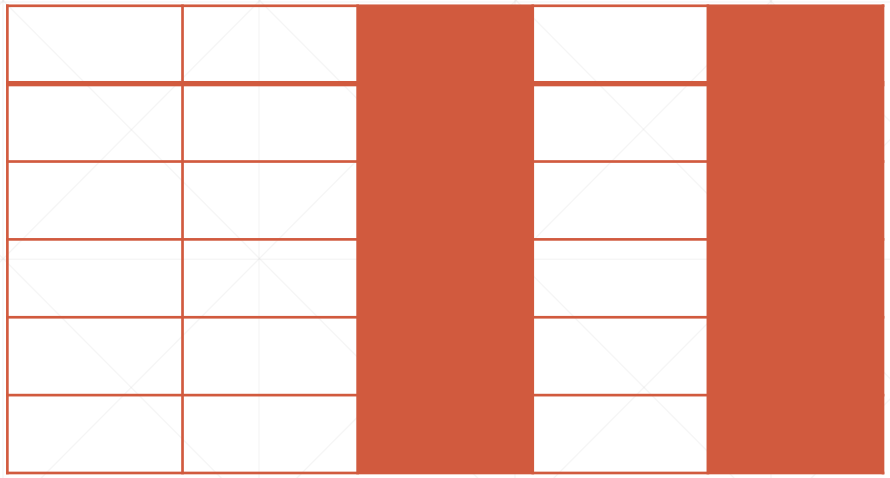


Figure 4-1
A query tree for Q3

Project



- project (input_table, {field_name(s)})
- The output table has the same rows as its input table, but with some columns removed.
- Q5 = project(STUDENT, {SName, GradYear})
- Q6 = project(select(STUDENT, MajorId=10), {SName})
  - The name of students having major 10.

Project

- Q6 = project(select(STUDENT, MajorId=10), {SName})

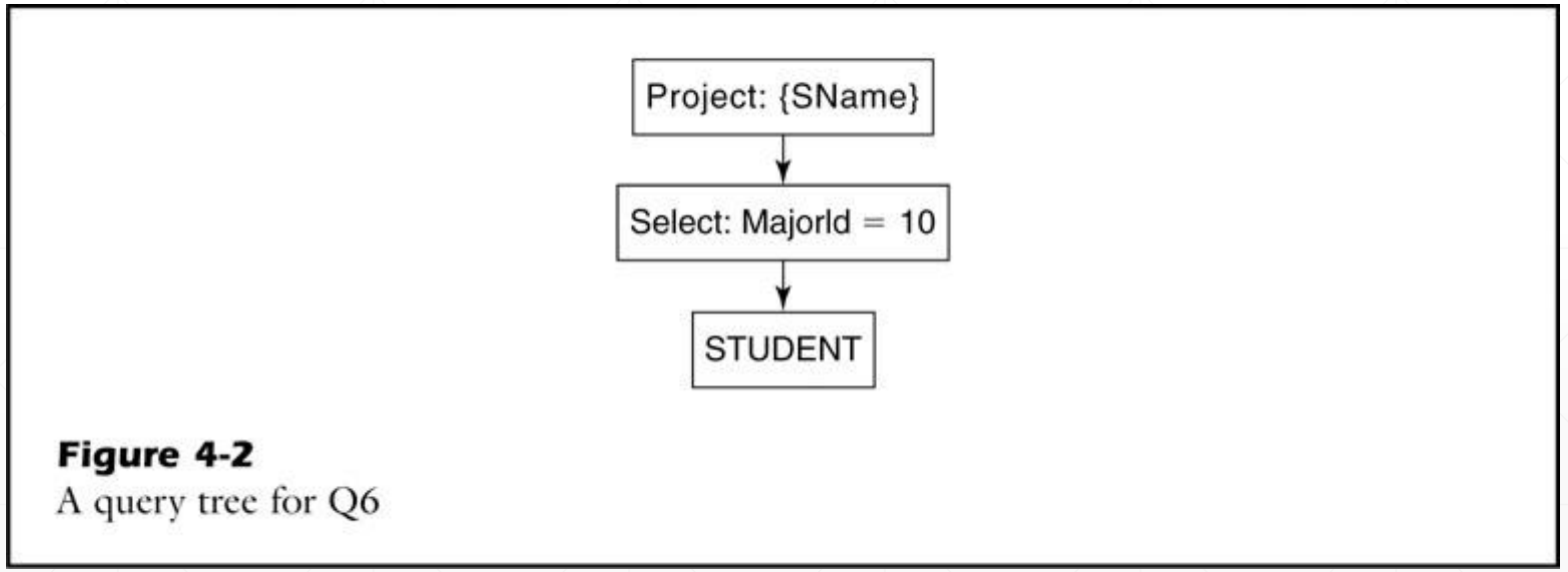


Figure 4-2
A query tree for Q6

Sort

- sort (input_table, [field_name(s)])
- The output table is the same as the input table, except that the rows are in a different order.
- Q8 = sort(STUDENT, [GradYear, SName])
  - Student records sorted by graduation year, then name.

Rename

- rename (input_table, field_name, new_field_name)
- The output table is the same as the input table, except that one column has a different name.
- Q9 = rename(Q6, SName, CSMajors)
  - Records from Q6, with field SName renamed to be CSMajors.

Extend

- extend (input_table, expression, new_field_name)
- The output table is the same as the input table, except that there is an additional column containing a computed value.
- Q10 = extend(STUDENT, GradYear-1863, GradClass)
  - Extending STUDENT to have the calculated field GradClass.
- Q11 = extend(STUDENT, 'BC', College)
  - Extending STUDENT to have the calculated field College.

GroupBy

- groupby (input_table, {grouping_field_name(s)}, {aggregation_expression(s)})
- The output table has one row for every group of input records.
- Q12 = groupby(STUDENT, {MajorId}, {Min(GradYear), Max(GradYear)})
  - The minimum and maximum graduation year, per major.

GroupBy Examples

- Q12 = groupby(STUDENT, {MajorId}, {Min(GradYear), Max(GradYear)})

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

Figure 4-3
The output of query Q12

GroupBy Examples (Cont.)

- Q13 = groupby(STUDENT, {MajorId,GradYear}, {Count(SId)})
  - The number of students in each major, per graduation year.

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

Figure 4-4
The output of query Q13

GroupBy Examples (Cont.)

- Q14 = groupby(STUDENT, {}, {Min(GradYear)})
  - The earliest graduation year of any student.
  - This query returns a table containing one row and one column; its value is the earliest graduation year of any student.
- Q15 = groupby(STUDENT, {MajorId}, {})
  - A list of the MajorId values of all students, with duplicates removed.

GroupBy Examples (Cont.)

- Q16 = groupby(STUDENT, {}, {Count(MajorId)})
  - The number of students having a major.
- Q17 = groupby(STUDENT, {}, {CountDistinct(MajorId)})
  - The number of different majors.

GroupBy Examples (Cont.)

- Q18 = select(ENROLL, Grade='A')
- Q19 = groupby(Q18, {SectionId}, {Count(EId)})
- Q20 = groupby(Q19, {}, {Max(CountOfEId)})

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

Figure 4-5
The output of queries Q18–Q20

The most number of A's given in any section.

GroupBy Examples (Cont.)

- Q21 = groupby(groupby(select(ENROLL, Grade='A'), {SectionId}, {Count(EId)}), {}, {Max(CountOfEId)})

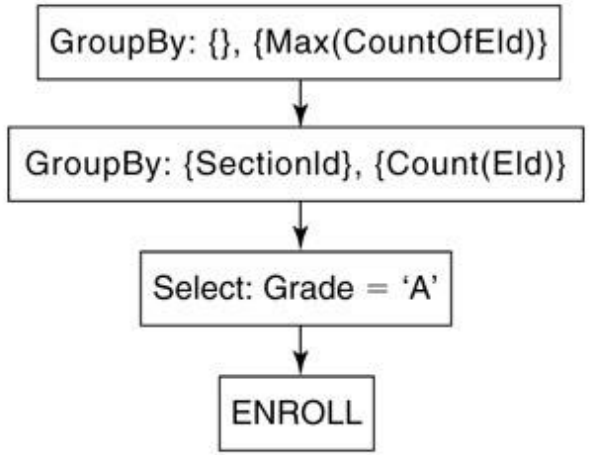


Figure 4-6
The query tree for Q21

The most number of A's given in any section (alternative version).

Relational Algebra (Cont.)

- The two-table operators are:
  - **Product**
  - **Join**
  - **Semijoin**
  - **Antijoin**
  - **Union**
  - **Outer join**

Product

- product (input_table_1, input_table_2)
- The output table consists of all possible combinations of records from its input tables.
- Q22 = product(STUDENT, DEPT)
  - All combinations of records from STUDENT and DEPT.

Product (Cont.)

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

Figure 4-7
The output of query Q22

Product (Cont.)

- Q23 = select(product(STUDENT, DEPT), MajorId=DId)
  - All students and their major departments.

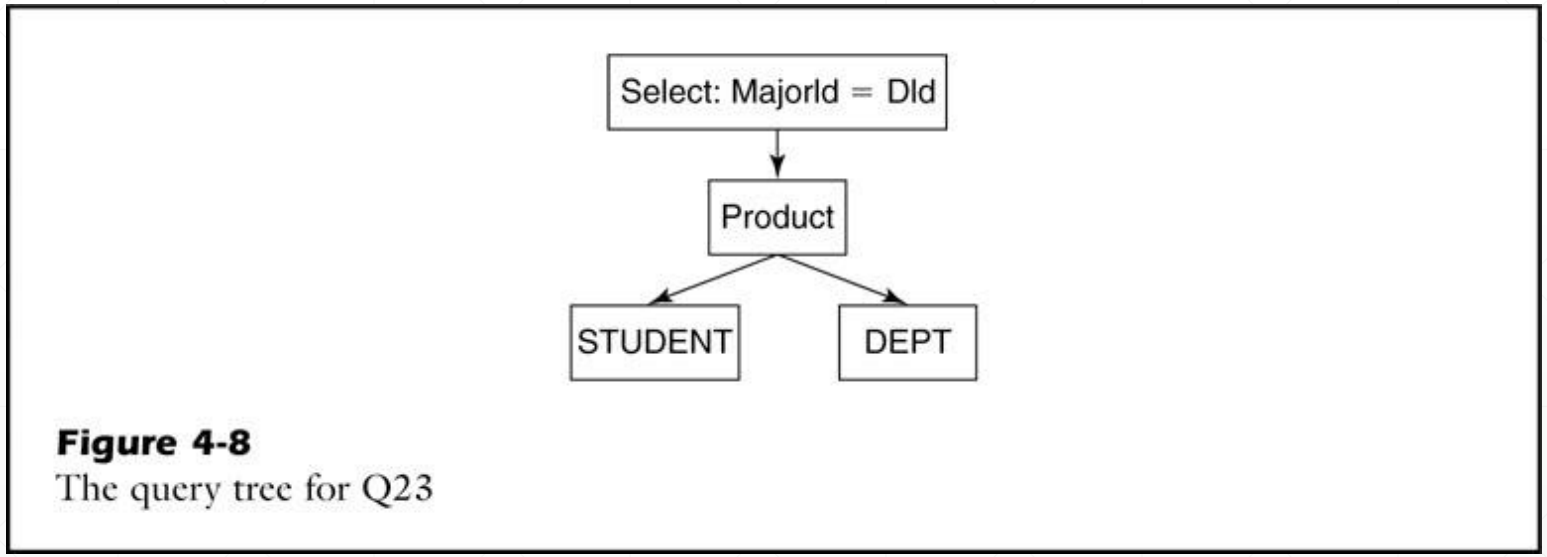


Figure 4-8
The query tree for Q23

Join

- join(T1, T2, P) = select(product(T1, T2), P)
- It returns only those combinations of records from two input tables satisfying the predicate.
- Query Q24 is equivalent to Q23.
  - Q23 = select(product(STUDENT, DEPT), MajorId=DId)
  - Q24 = join(STUDENT, DEPT, MajorId=DId)
  - All students and their major departments.

SemiJoin

- semijoin(input_table_1, input_table_2, predicate)
- The output table consists of the records from the first input table that match some record in the second input table.
- Q35 = semijoin(DEPT, STUDENT, DId=MajorId)
  - The departments that have at least one student in their major.
- Q36 = join(DEPT, STUDENT, DId=MajorId)
- Q37 = groupby(Q36, {DId, DName}, {})
  - The departments that have at least one student in their major (alternative version)

SemiJoin (Cont.)

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

Figure 2-1
The schema of the university database

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

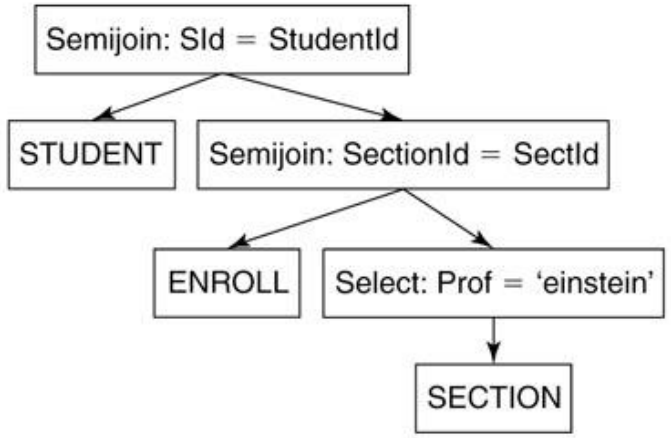


Figure 4-11
The query tree for Q38–Q40

AntiJoin

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

Figure 2-1
The schema of the university database

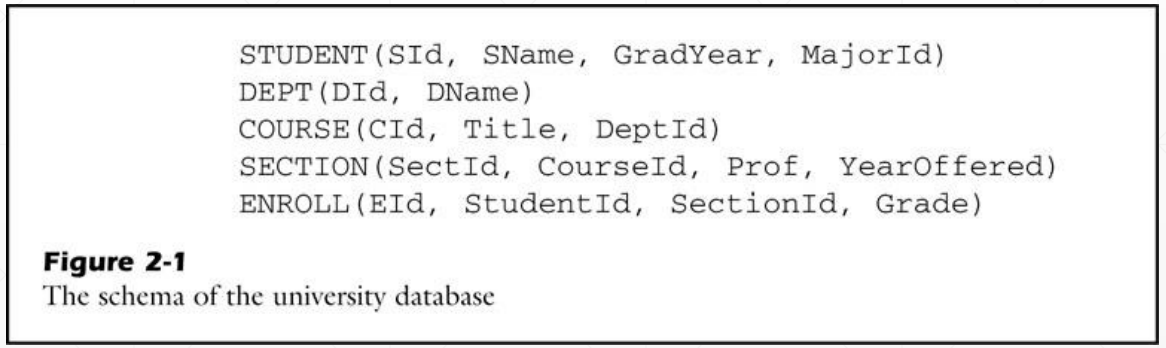
- antijoin(input_table_1, input_table_2, predicate)
- The output table consists of the records from the first input table that DO NOT match records in the second input table.
- Q41 = antijoin(DEPT, STUDENT, DId=MajorId)
  - The departments that have no student majors.

Sections where nobody received an F.

- Q42 = select(ENROLL, Grade='F')
- Q43 = antijoin(SECTION, Q42, SectionId=SectId)

Semi-Join & AntiJoin Example

- Professors who have never given a grade of F.
  - Q44 = select(ENROLL, Grade='F')
  - Q45 = semijoin(SECTION, Q44, SectionId=SectId)
  - Q46 = rename(Q45, Prof, BadProf)
  - Q47 = antijoin(SECTION, Q46, Prof=BadProf)
  - Q48 = groupby(Q47, {Prof}, {})



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

Figure 2-1
The schema of the university database

Professors who gave an F in every section they taught.

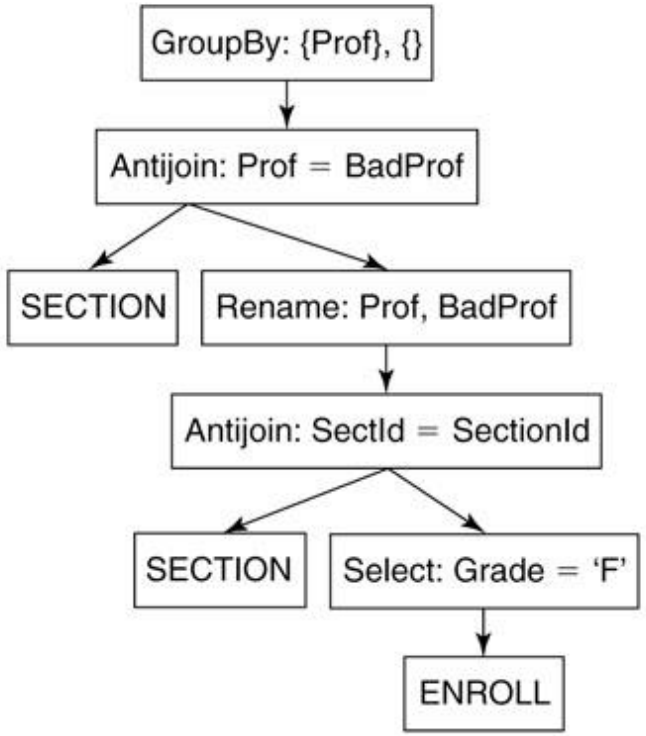


Figure 4-12
The query tree for Q49–Q51

Book's solution for Q49 (sections without F) and Q50 (profs of the sections with F) is corrected as seen on the left text box. Figure is not changed!!!!

Semi-Join & AntiJoin Example

- Q43 = antijoin(SECTION, Q42, SectionId=SectId)

- Q49 = rename(Q43, Prof, GoodProf)
- Q50 = antijoin(SECTION, Q49, Prof = GoodProf)
- Q51 = groupby(Q50, {Prof}, {})

Union

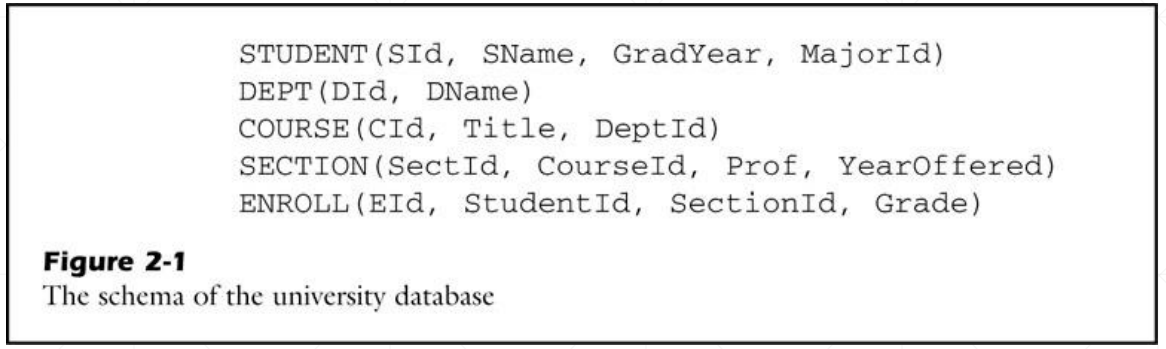


Figure 2-1
The schema of the university database

- union(input_table_1, input_table_2)
- The output table consists of the records from each of its two input tables.
- Requires that both underlying tables have the same schema.
- Q52 = rename(project(STUDENT, {SName}), SName, Person)
- Q53 = rename(project(SECTION, {Prof}), Prof, Person)
- Q54 = union(Q52, Q53)
  - The combined names of students and professors.

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

Figure 4-13
The output of query Q55

Outer Join

- outerjoin(input_table_1, input_table_2, predicate)
- The output table contains all the records of the join, together with the non-matching records padded with nulls.
- Q55 = outerjoin(STUDENT, ENROLL, SId=StudentId)

SQL (Structured Query Language)

- SQL commands broadly fit into five categories:
  - 1) DDL (Data Definition Language)
  - 2) DML (Data Manipulation Language)
  - 3) DCL (Data Control Language)
  - 4) DQL (Data Query Language)
  - 5) TCL (Transactional Control Language)

SQL (1-Data Definition Language)

- The DDL commands in SQL are used to create database schema and to define the type and structure of the data that will be stored in a database.
  - CREATE (database, table)
  - ALTER (table)
  - DROP (database, table)
  - TRUNCATE (table)

SQL (2-Data Manipulation Language)

- There are four basic ways to update a database in SQL:
  - insert one new record by giving its values
  - insert multiple new records by specifying a query
  - delete records
  - modify the contents of records

SQL (3-Data Control Language)

Additional functions for authorization needs.

- Examples of DCL commands include:
- GRANT to allow specified users to perform specified tasks.
- REVOKE to remove the user accessibility to database object.

SQL (4-Data Query Language)

- An SQL query has multiple sections, called ***clauses***, in which the various kinds of operations appear.
- There are six common clauses:
  - select
  - from
  - where
  - group by
  - having
  - order by

SQL Queries (4)

- The ***select*** clause lists the columns of the output table.
  - An output column can be a column from an input table, or it can be the results of a calculation.
- The ***from*** clause lists the input tables.
  - An input table can be the name of a stored table or view, or it can be an inline subquery that defines an anonymous view.
- The ***where*** clause contains the selection predicate.
- The ***group by*** clause specifies the fields used for aggregation.
  - The output table will consist of one row for every combination of field values that satisfies the selection predicate.

SQL Queries (4)

- The ***having*** works in conjunction with the group by clause.
  - It contains a predicate that gets evaluated after grouping occurs.
  - Consequently, the predicate can call aggregation functions and test their values.
- The ***order by*** clause specifies the sort order for the output records.
  - Sorting occurs after all other query processing is performed.

References

- Edward Sciore, Database Design and Implementation, Wiley, 2009.

References

- Edward Sciore, Database Design and Implementation, Wiley, 2009.

Flights Database

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

Example Relational Algebra Queries

a) Get the complete details of all flights to New Delhi.

b) Get the details about all flights from Chennai to New Delhi.

c) Find only the flight numbers for passenger with pid 123 for flights to Chennai before 06/11/2020.

d) Find the passenger names for passengers who have bookings on at least one flight.

e) Find the passenger names for those who do not have any bookings in any flights.

f) Find the agency names for agencies that are located in the same city as passenger with passenger id 123.

h) Get the details of flights that are scheduled on either of the dates 01/12/2020 or 02/12/2020 or both at 16:00 hours

j) Find the details of all male passengers who are associated with Jet agency.

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

a) Get the complete details of all flights to New Delhi.

select (flight, dest = “New Delhi”)

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

b) Get the details about all flights from Chennai to New Delhi.

select (flight, src = “Chennai” AND dest = “New Delhi”)

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

c) Find only the flight numbers for passenger with pid 123 for flights to Chennai before 06/11/2020.

T1= select (flight, dest = “Chennai” AND fdate < 06/11/2020)
T2= join (T1, booking, t1.fid = booking. fid)
T3= select (T2, pid=123)
T4 = project(T3, {fid})

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

d) Find the passenger names for passengers who have bookings on at least one flight.

T1= semijoin(passenger, booking, passenger.pid=booking.pid)
T2= project (T1, {pname}]

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

e) Find the passenger names for those who do not have any bookings in any flights.

T1= antijoinjoin(passenger, booking, passenger.pid=booking.pid)
T2= project (T1, {pname}]

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

f) Find the agency names for agencies that located in the same city as passenger with passenger id 123.

T1= select (passenger, pid=123)
T2= join (T1, agency, pcity=acity)
T3=project(T2,{aname})

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

h) Get the details of flights that are scheduled on either of the dates 01/12/2020 or 02/12/2020 or both at 16:00 hours.

select (flight, fdate = 01/12/2020 AND time = 16:00) ∪ select(flight, fdate = 02/12/2020 AND time = 16:00)

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

j) Find the details of all male passengers who are associated with Jet agency.

T1= join(passenger, booking, passenger.pid=booking.pid)
T2= join(T1, agency, agency.aid=booking.aid)
T3=project(select(T2, pgender="Male" AND aname = "Jet agency")), {pid, pname})