GMU Database Ph.D. Qualifying Exam Preparation Notes

Eric O. Scott

January 6, 2014

Abstract: The definitions, concepts and exercises in this study guide should be most of what you need to know to prepare for the Ph.D. qualifying exam in Databases at George Mason University. This was my personal attempt to assemble the important concepts from INFS 614 in one place to A) guide my reading of the textbook, and B) repeat and recall information I wanted to memorize. A lot of detail is left out. Some of the exercises have solutions – these were written by me and may contain errors.

Contents

1 Database Models 1
  1.1 Entity-Relationship Model ................................................................. 1
  1.2 Relational Model .............................................................................. 2
  1.3 Problems ......................................................................................... 3

2 Query Languages 5
  2.1 Schemas Used in Examples ............................................................... 5
  2.2 Relational Algebra ........................................................................... 5
  2.3 SQL ................................................................................................ 6
  2.4 Problems ......................................................................................... 7

3 Normalization 18
  3.1 Functional Dependency .................................................................. 18
  3.2 Decomposition ................................................................................. 19
  3.3 Normal Forms ................................................................................ 19
  3.4 Problems ......................................................................................... 21

1 Database Models

1.1 Entity-Relationship Model

Definition 1. A key constraint, represented by an arrow connecting an entity to a relationship, implies that the entity appears in at most one instance of the relationship. If there is a key constraint on one side of a binary relationship, the relationship is one-to-many (ex. one department to many employees). If there are key constraints on both sides, it is one-to-one.

Definition 2. If all the entities in an entity set appear in an instance of a relation, then we say the entity’s participation in that relation is total – represented in the ER diagram as a bold line connecting the entity to the relationship. Otherwise, its participation is partial – a light line.
Definition 3. A weak entity can be defined only by considering some of its attributes (a partial key) in conjunction with the primary key of another entity, its identifying owner. A weak entity must have total participation in a one-two-many identifying relationship with the owner.

1.2 Relational Model

Definition 4. A relational schema $S$ is a set of attribute names $A_1, \ldots, A_n$ and their corresponding domains $D_1, \ldots, D_n$.

Definition 5. A relational tuple over a schema $S$ is a mapping $t : \{A_1, \ldots, A_n\} \rightarrow \bigcup_{i=1}^{n} D_i$ such that $\forall i, t(A_i) \in D_i$.

Definition 6. A relational instance (a.k.a. relation) over a schema $S$ is a set of relational tuples over $S$.

Definition 7. A relational database schema is a set of relational schemas $\{S_1, \ldots, S_n\}$.

Definition 8. A relational database over the schemas $\{S_1, \ldots, S_n\}$ is a set of relational instances $\{r_1, \ldots, r_n\}$ over the schemas $\{S_1, \ldots, S_n\}$.

Definition 9. A superkey of an entity set is a (sub)set of the attributes such that no two entities in the set is allowed to have the same values on all these (key) attributes.

Definition 10. A candidate key is a superkey that does not have a “redundant” attribute, i.e., if any attribute is removed, the set is not a superkey anymore.

Definition 11. A particular candidate key is designated by the designer to be the primary key.

Definition 12. When defining a relation in SQL, the CREATE TABLE syntax allows the specification of four integrity constraints:

- Domain constraints such as sid CHAR(20)
- Primary key constraints such as PRIMARY KEY(sid)
- Uniqueness constraints such as UNIQUE(fname, lname)
- Foreign key constraints such as FOREIGN KEY (studid) REFERENCES Students

Several options are available for how to enforce foreign key constraints. They are specified as part of the FOREIGN KEY (studid) REFERENCES Students syntax, and are applied when a row in Students is updated or deleted.

- **ON DELETE CASCADE** delete all rows that reference the Student
- **ON DELETE SET DEFAULT** replaces the value of studid with its default value. You can also SET NULL
- **ON UPDATE NO ACTION** leaves dangling references when a Student’s primary key is modified
- **ON UPDATE CASCADE** also modifies the references
Example 1. A relation definition showing the important features of CREATE TABLE:

```sql
CREATE TABLE Students ( sid CHAR(20),
  major INTEGER,
  name CHAR(30),
  login CHAR(20),
  age INTEGER,
  gpa REAL,
  UNIQUE (name, age),
  SET CONSTRAINT pkey PRIMARY KEY (sid),
  SET CONSTRAINT fkey FOREIGN KEY (major) REFERENCES Majors
  ON UPDATE CASCADE
  ON DELETE SET DEFAULT
```

The syntax SET CONSTRAINT ConstraintFoo DEFERRED or SET CONSTRAINT ConstraintFoo IMMEDIATE can be used to toggle whether every statement in a transaction has its constraints checked immediately, or whether all constraints are checked at the end of the transaction (which is necessary for insertions of multiple entities which reference each other).

1.3 Problems

Problem 1. Design a small database for used textbooks, as might be used by an on-line trading site like Amazon. The database will store textbooks for sale. Each textbook has an ISBN, a category (e.g., Math) it belongs to, a name. In addition, status descriptions about the books are maintained. Each book is required to have one such description so that the system can rate it fairly. Each textbook is published by a company in the publishing year. Each publisher has to have published at least some textbooks. Each textbook has one or more sellers, which may be either companies (corporation sellers) or individuals (individual sellers). For each company, the database maintains a name of the company, its address, its phone numbers (could be more than one phone number, each with a number and a description), and its contact person (who is an individual with all the related information for individuals, see next sentence). For each individual, the database keeps a name, a phone number and an email address. A contact person whose company sells a book cannot be selling the same book as an “individual seller” at the same time (he/she may sell other books as an individual seller). (Source: Given by Prof. A. Brodsky in INFS 614)
Problem 2. Professors and GTAs are assigned to teach the sections of each class being offered in a semester. At the end of the semester, they get a “team rating” (professors and GTAs together get one rating per section, rating is not done on individual). To support the assignment of professors to sections, a record is kept of which class each professor can teach. Classes can have one or more prerequisite classes. Students can take several sections each semester, and receive a grade for taking each section. Students may end up waiting for some sections, and receive a “rank” (determining the order they will be admitted if other students drop). However, no more than 10 students can wait on a class at the same time. Note that GTAs are students, however they differ in that they have a salary. All people (e.g. students, professors) are uniquely identified by their social security number. All classes are identified by department name (e.g. “INFS”) and course number (e.g. “614”). Sections of classes are distinguished by their section number (e.g. “02”).

Given the above description, do the following:
1. Draw an ER-diagram for the database, identifying the following:
   (a) all the entity sets;
   (b) all the relationship sets and their multiplicity;
   (c) the primary key for each entity set (and weak entity set, if any) and each relationship set.
   Invent more attribute(s) for the entity sets if you like.
2. Indicate (what and why) feature(s)/property(ies) (if any) in the above description that are NOT captured by your ER-diagram;
3. Give 2 examples of the types of reports that can be obtained from the database, and state the involved entity sets and/or relationship sets. Each report example must involve at least two entity/relationship sets; (For example, an report can be “List all the GTAs who have NOT taken all the prerequisite classes for the classes that are assigned to teach”.)
4. Convert the entity-relationship design (in (a)) to a scheme for a relational database. List all relation schemes. For each relation scheme, state
   (a) the name of the relation,
   (b) the names of its attributes,
   (c) the domain (or data type) of each attribute,
   (d) the primary key,
   (e) the foreign key(s);
5. Use Oracle on ITE LAB machine to create the tables from 4 above and insert at least two tuples to each table. Implement all these in one script (text) file

(Source: Given by Prof. A. Brodsky in INFS 614)

Problem 3. Consider the following description of your local community library. Create the corresponding ER diagram.

1. The library has books, CDs, tapes and other items, which are lent to library patrons.
2. Library patrons have accounts, and addresses.
3. If a loaned item is overdue, it accumulates penalty
4. Some patrons are minors, so they must have sponsoring patrons who are responsible for paying penalties (or replacing a library item in case of loss).

(Source: Given by Prof. A. Brodsky on an INFS 614 midterm)
2 Query Languages

2.1 Schemas Used in Examples

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/11/96</td>
</tr>
<tr>
<td>58</td>
<td>102</td>
<td>11/12/96</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>09/24/12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Red</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Red</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Bob</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>22</td>
<td>Gary</td>
<td>5</td>
<td>27</td>
</tr>
</tbody>
</table>

2.2 Relational Algebra

**Definition 13.** Basic relational operators:

\[ \rho \quad \text{Rename} \quad \neg \quad \text{Minus} \]
\[ \sigma \quad \text{Select} \quad \times \quad \text{Cross product} \]
\[ \pi \quad \text{Project} \quad \bowtie_c \quad \text{Conditional Join} \]
\[ \cup \quad \text{Union} \quad \bowtie_{y=b} \quad \text{Equijoin} \]
\[ \cap \quad \text{Intersection} \quad \bowtie \quad \text{Natural join (Equijoin on all fields having the same name)} \]

**Definition 14.** Let \( A \) be a relation instance with exactly two fields \( x \) and \( y \) and \( B \) be a relation instance with just one field, \( y \), with the same domain as in \( A \). The division \( A/B \) is defined as the set of all \( x \) values such that for every \( y \) value in \( B \), there is a tuple \( \langle x, y \rangle \) in \( A \).

The division operator is so named because it is much like an inverse of the cross product. For integers, \( A/B \) is the largest integer \( Q \) such that \( QB \leq A \). Likewise, for relation instance, \( A/B \) is the largest relation instance \( Q \) such that \( Q \times B \subseteq A \).
2.3 SQL

Definition 15. In addition to logical connectives (AND, OR and NOT) and comparators, SQL provides the following features:

- **SELECT** Specify the select-list
- **FROM** Specify the from-list
- **WHERE** Specify qualification conditions
- **DISTINCT** Convert the output from a bag to a set
- **UNION** Multiset union
- **INTERSECT** Multiset intersection
- **EXCEPT** Multiset subtraction
- **IN** Check if an element is in a given multiset
- **op ANY** Check if op is true for any element in a multiset
- **op ALL** Check if op is true for all elements in a multiset
- **EXISTS** Check if a multi set is empty
- **NOT** Can also be used as a prefix to IN and EXISTS
- **COUNT([DISTINCT] A)** Count [unique] elements in column A
- **SUM([DISTINCT] A)** Sum [unique] elements in column A
- **AVG([DISTINCT] A)** Average [unique] elements in column A
- **MAX(A)** Maximum value in column A
- **MIN(A)** Minimum value in column A
- **GROUP BY** Specify a grouping-list for aggregate operators
- **HAVING** Specify group qualifications
- **EVERY(A op B)** See if a condition holds for all tuples in the group
- **ANY(A op B)** See if a condition holds for some tuple in the group

Comparison operators can be applied to the output of nested queries that return a single value. Ex. the following is legal as a qualification condition: S.age > (SELECT MAX(S2.age) FROM Sailors S2).

When **GROUP BY** is used, only aggregates (ex. **COUNT(*) AS reservationCount**) or column names appearing in the grouping-list may be used in the select list.

The group qualifications given in **HAVING** may only refer to column names appearing in the grouping-list (except inside aggregates, EVERY or ANY, which can take any column).

Definition 16. Due to the potential presence of null values in relational databases, comparisons may return unknown. To handle expressions such as rating = 8 AND age < 40, SQL's logical connectives use three-valued logic in the following manner:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>NOT A</th>
<th>B</th>
<th>A OR B</th>
<th>A</th>
<th>B</th>
<th>A AND B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>unknown</td>
<td>unknown</td>
<td>T</td>
<td>unknown</td>
<td>unknown</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td></td>
</tr>
</tbody>
</table>

Division Find sailors who’ve reserved all boats:

- **SELECT** S.sid
- **FROM** Sailors S
WHERE (SELECT R.bid
FROM Reserve R
WHERE R.sid = S.sid
CONTAINS (SELECT B.bid
FROM Boats B))

We want to know if $A \subseteq B$. Not all DBMS’s support CONTAINS. Since $A \subseteq B$ iff $(B - A) = \emptyset$, we can check this with the other set operators:

SELECT S.sid
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
FROM Boat B
EXCEPT (SELECT R.bid
WHERE R.sid = S.sid))

2.4 Problems

(Source: Ramakrishnan & Gehrke, problem 4.2)

Problem 4. Given two relations $R_1$ and $R_2$, where $R_1$ contains $N_1$ tuples, $R_2$ contains $N_2$ tuples, and $N_2 > N_1 > 0$, give the minimum and maximum possible sizes (in tuples) for the resulting relation produced by each of the following relational algebra expressions. In each case, state any assumptions about the schemas $R_1$ and $R_2$ needed to make the expression meaningful.

1. $R_1 \cup R_2$
   - Minimum: $N_1$
   - Maximum: $N_1 + N_2$

2. $R_2 \cap R_2$
   - Minimum: 0
   - Maximum: $N_1$

3. $R_1 - R_2$
   - Minimum: 0
   - Maximum: $N_1$

4. $R_1 \times R_2$
   - Minimum: $N_1 N_2$
   - Maximum: $N_1 N_2$

5. $\sigma_{a=5}(R_1)$
   - Minimum: 0
   - Maximum: $N_1$

6. $\pi_a(R_1)$
   - Minimum: 1
   - Maximum: $N_1$

7. $R_1/R_2$
   - Minimum: 0
   - Maximum: $N_1/N_2$
Problem 5. (Source: In-class examples given by Prof. A. Brodsky in INFS 614) Given the relation Works (pname, cname, city, straddr), answer the following query in Tuple Relational Calculus, Relational Algebra, and SQL:

Find people who work for the company ‘FBC.’

Proof. Algebra:

\[ \pi_{\text{pname}} (\sigma_{\text{cname} = 'FBC'}(W)) \]

Calculus:

\[ \{ t.\text{pname} \mid t \in W \land t.\text{cname} = 'FBC' \} \]

SQL:

```
SELECT w.pname
FROM Works w
WHERE w.cname = 'FBC'
```
Problem 7. (Source: In-class examples given by Prof. A. Brodsky in INFS 614) Given the relation Works (pname, cname, city, straddr), answer the following query in Tuple Relational Calculus, Relational Algebra, and SQL:

Find all companies such that all employees live in Washington, D.C. Try the SQL version of this two ways, with a nested SELECT and again with an EXCEPT.

Proof. Algebra:

\[ \pi_{\text{cname}}(\text{Works}) - \pi_{\text{cname}}(\sigma_{\text{city}<>'WashingtonD.C.'}(\text{Works})) \]

Calculus:

\[ \{ t.\text{cname} \mid t \in \text{Works} \land \forall s \in \text{Works}, \ (s.\text{cname} = t.\text{cname}) \Rightarrow (s.\text{city} = 'Washington, D.C.') \} \]

SQL (EXCEPT version):

\[
\text{(SELECT w1.cname FROM Works w1) EXCEPT (SELECT w2.cname FROM Works w2 WHERE w2.city != 'Washington, D.C.')}
\]

SQL (Nested SELECT version):

\[
\text{SELECT DISTINCT w1.cname FROM Works w1 WHERE NOT IN (SELECT w2.cname FROM Works w2 WHERE w2.city <> 'Washington, D.C.'))}
\]
Problem 8. (Source: In-class examples given by Prof. A. Brodsky in INFS 614) Given the relation \texttt{Works} (\texttt{pname}, \texttt{cname}, \texttt{city}, \texttt{straddr}), answer the following query in Tuple Relational Calculus, Relational Algebra, and SQL:

Find all companies such that all employees live in the same city.

Proof. Algebra:

\[
\rho(W_1, \text{Works}) \\
\rho(W_2, \text{Works}) \\
\pi_{\text{cname}}(\text{Works}) - \pi_{\text{cname}}(\sigma_{W_1.\text{city} <> W_2.\text{city}}(W_1 \bowtie_{W_1.\text{cname}=W_2.\text{cname}} W_2))
\]

Calculus:

\[
\{t.\text{cname} \mid t \in \text{Works} \land \\
(\forall s \in \text{Works}, \\
\quad (t.\text{cname} = s.\text{cname}) \Rightarrow (t.\text{city} = s.\text{city}))\}
\]

SQL (This kind of nested select, which references the parent query, is called a correlated query):

\[
\text{SELECT DISTINCT } w1.\text{cname} \\
\text{FROM } \text{Works } w1 \\
\text{WHERE NOT IN } (\text{SELECT } w2.\text{cname} \\
\quad \text{FROM } \text{Works } w2 \\
\quad \text{WHERE } w2.\text{cname} = w1.\text{cname} \land \\
\quad w2.\text{city} <> w1.\text{city})
\]

Problem 9. (Source: In-class examples given by Prof. A. Brodsky in INFS 614) Given the relation \texttt{Works} (\texttt{pname}, \texttt{cname}, \texttt{city}, \texttt{straddr}), answer the following query in Tuple Relational Calculus, Relational Algebra, and SQL:

Find a company whose employees live in all cities.

Proof. Algebra:

\[
\pi_{\text{cname},\text{city}}(\text{Works})/\pi_{\text{city}}(\text{Works})
\]

Calculus:

\[
\{t.\text{cname} \mid t \in \text{Works} \land \\
(\forall s \in \text{Works}, \\
\quad (\exists r \in \text{Works}, r.\text{city} = s.\text{city} \land r.\text{cname} = t.\text{cname}))\}
\]

SQL (Employee cities \subseteq all cities if and only if all cities \text{−} employee cities = 0):

\[
\text{SELECT } w1.\text{cname} \\
\text{FROM } \text{Works } w1 \\
\text{WHERE NOT EXISTS } (\text{SELECT } w2.\text{city} \\
\quad \text{FROM } \text{Works } w2 \\
\quad \text{EXCEPT } (\text{SELECT } w3.\text{city} \\
\quad \text{FROM } \text{Works } w3 \\
\quad \text{WHERE } w3.\text{cname} = w1.\text{cname}))
\]
These relation schemas are used in the next seven problems. Primary keys are underlined.

• Musicians(ssn, name, annualIncome)
• Instruments(instrID, iname, key)
• Plays(ssn, instrID)
• SongsAppear(songID, authorSSN, title, albumIdentifier)
• Lives(ssn, address, phone)
• Place(address, otherInfo)
• Perform(songID, ssn)
• AlbumProducer(ssn, albumIdentifier, copyrightDate, speed, title)

Problem 10. (Source: Given on homework by Prof. A. Brodsky in INFS 614)
Use (i) Relational Algebra and (ii) Tuple Relational Calculus to express the following query:

Find the instruments (InstID) played by musician named ‘John’.

Proof. Algebra:
\[ \rho(IPM, Instruments ⊥ depth{Instruments.instrID}=Plays.instrID \, Plays ⊥ depth{Plays.ssnn}=Musicians.ssn \, Musicians) \]
\[ \pi_{instrID}(\sigma_{Musicians.name='John'}(IPM)) \]

Calculus:
\[ \{ i.instrID \mid i \in Instruments \land \\
(\exists p \in Plays, \\
(\exists m \in Musicians, \\
i.instrID = p.instrID \land \\
p.ssn = m.ssn \land \\
m.name = 'John' ) \} \]
**Problem 11.** *(Source: Given on homework by Prof. A. Brodsky in INFS 614)*

Use (i) Relational Algebra and (ii) Tuple Relational Calculus to express the following query:

Find the titles of the albums produced by musicians who play guitar or piano (iname='guitar' or 'piano').

**Proof.** Algebra:

\[
\rho(MPI, Musicians \bowtie_{Musicians.ssn=Plays.ssn} Plays \bowtie_{Plays.instrID=Instruments.instrID} Instruments) \\
\rho(GP, \pi_{Musicians.ssn}(\sigma_{iname='Guitar' \lor iname='Piano'}(MPI))) \\
\pi_{title}(AlbumProducer \bowtie GP)
\]

Calculus:

\[
\{a.title | a \in AlbumProducer \land \\
(\exists m \in Musicians, \\
(\exists p \in Plays, \\
(\exists i \in Instruments, \\
a.ssn = m.ssn \land \\
a.ssn = p.ssn \land \\
p.instrID = i.instrID \land \\
(i.iname = 'Guitar' \lor i.iname = 'Piano')))}
\]

**Problem 12.** *(Source: Given on homework by Prof. A. Brodsky in INFS 614)*

Use (i) Relational Algebra and (ii) Tuple Relational Calculus to express the following query:

Find the musicians who played both song1 (songID='song1') and song2 (songID='song2').

**Proof.** Algebra:

\[
\rho(P_1, Perform) \\
\rho(P_2, Perform) \\
\pi_{ssn}(\sigma_{P_1.songID='song1' \land P_2.songID='song2'}(P_1 \times P_2))
\]

Calculus:

\[
\{p.ssn | p \in Perform \land \\
(\exists q \in Perform, \\
p.ssn = q.ssn \land \\
p.songID = 'song1' \land \\
q.songID = 'song2')\}
\]
Problem 13. *(Source: Given on homework by Prof. A. Brodsky in INFS 614)*

Use (i) Relational Algebra and (ii) Tuple Relational Calculus to express the following query:

Find the musician(s) with the highest annual income.

**Proof.**

**Algebra:**

\[
\rho(M_1, \text{Musicians}) \\
\rho(M_2, \text{Musicians}) \\
\pi_{\text{ssn}}(\text{Musicians}) - \pi_{M_1.\text{ssn}}(M_1 \bowtie_{M_1.\text{annualIncome} < M_2.\text{annualIncome}} M_2)
\]

**Calculus:**

\[
\{ m.\text{ssn} \mid m \in \text{Musicians} \land \\
(\forall n \in \text{Musicians}, \\
\quad m.\text{annualIncome} \geq n.\text{annualIncome}) \}
\]

Problem 14. *(Source: Given on homework by Prof. A. Brodsky in INFS 614)*

Use (i) Relational Algebra and (ii) Tuple Relational Calculus to express the following query:

Find all the pairs of musicians (give names) who share the same addresses.

**Proof.**

**Algebra:**

\[
\rho(ML_1, \text{Musicians} \bowtie_{\text{Musicians.\text{ssn}=Lives.\text{ssn}}} \text{Lives}) \\
\rho(ML_2, \text{Musicians} \bowtie_{\text{Musicians.\text{ssn}=Lives.\text{ssn}}} \text{Lives}) \\
\pi_{ML_1.\text{name}, ML_2.\text{name}}(ML_1 \bowtie_{ML_1.\text{address}=ML_2.\text{address}} ML_2)
\]

**Calculus:**

\[
\{(m.\text{name}, n.\text{name}) \mid m \in \text{Musicians} \land n \in \text{Musicians} \land \\
(\exists m \in \text{Lives}, \\
\quad (\exists n \in \text{Lives}, \\
\quad \quad l_m.\text{ssn} = m.\text{ssn} \land \\
\quad \quad l_n.\text{ssn} = n.\text{ssn} \land \\
\quad \quad l_m.\text{address} = l_n.\text{address})\}
\]
Problem 15. (Source: Given on homework by Prof. A. Brodsky in INFS 614)
Use (i) Relational Algebra and (ii) Tuple Relational Calculus to express the following query:

Find the musicians (names) who played all the songs written by ‘John’ (musician author name = ‘John’).

Proof. Algebra:
\[
\rho(\text{JohnSongs}, \pi_{\text{songID}}(\sigma_{\text{name}}=\text{John}(\text{SongsAppear} \Join_{\text{authorSSN}}=\text{Musicians.ssn} \text{ Musicians})))
\]
\[
\rho(\text{SSNS}, \pi_{\text{ssn}, \text{songID}}(\text{Performs})/\text{JohnSongs})
\]
\[
\pi_{\text{name}}(\text{SSNS} \Join \text{Musicians})
\]

Calculus:
\[
\{m.\text{name} | m \in \text{Musicians} \land \\
\quad (\forall s \in \text{SongsAppear}, \\
\quad (\exists j \in \text{Musicians}, j.\text{name} = \text{John} \land s.\text{authorSSN} = j.\text{ssn}) \\
\quad \Rightarrow (\exists p \in \text{Performs}, p.\text{songID} = s.\text{songID} \land p.\text{ssn} = m.\text{ssn})\}
\]

The following schema is used for the rest of the problems in this section:
- **Department** (D-code, D-Name, Chair-SSn)
- **Course** (D-code, C-no, Title, Units)
- **Prereq** (D-code, C-no, P-code, P-no)
- **Class** (Class-no, D-code, C-no, Instructor-SSn)
- **Faculty** (Ssn, F-Name, D-Code, Rank)
- **Student** (Ssn, S-Name, Major, Status)
- **Enrollment** (Class-no, Student-Ssn)
- **Transcript** (Student-Ssn, D-code, C-no, Grade)
Problem 16. (Source: Given on homework by Prof. A. Brodsky in INFS 614)
Specify the following query in (i) the Relational Algebra, (ii) Relational Calculus and (iii) implement them in SQL.
List the courses (D-code and C-no), along with the names of the students who are currently taking them.

Proof. Algebra:

\[
\rho(C, \text{Course}) \\
\rho(E, \text{Enrollment}) \\
\rho(S, \text{Student}) \\
\pi \text{D-code, C-no, S-Name} (C \bowtie C.\text{C-no} = E.\text{Class-no} E \bowtie E.\text{Student-Ssn} = S.\text{Ssn} S)
\]

Calculus:

\[
\{ (\text{c.D-Code, c.C-no, s.S-Name}) \mid c \in \text{Course} \land s \in \text{Student} \land \exists e \in \text{Enrollment}, \\
\quad \text{c.C-no} = e.\text{Class-no} \land \\
\quad s.\text{Ssn} = e.\text{Student-Ssn}\}
\]

SQL:

\[
\text{SELECT c.D_Code, c.C_no, s.S_Name} \\
\text{FROM Course c, Student s, Enrollment e} \\
\text{WHERE c.C_no = e.Class_no AND s.Ssn = e.Student_Ssn}
\]

Problem 17. (Source: Given on homework by Prof. A. Brodsky in INFS 614)
Specify the following query in (i) the Relational Algebra, (ii) Relational Calculus and (iii) implement them in SQL.
List all the courses (D-code and C-no) that John (i.e., S-Name=“John”) got ‘A’ grade.

Proof. Algebra:

\[
\pi \text{D-code, C-no} (\sigma \text{S-Name = ‘John’} \land \text{Grade = ‘A’} (\text{Transcript} \bowtie \text{Student-Ssn} = \text{Ssn} \text{ Student}))
\]

Calculus:

\[
\{ (t.D-code, t.C-no) \mid t \in \text{Transcript} \land \\
\quad (\exists s \in \text{Student}, \\
\quad \quad t.\text{Student-ssn} = s.\text{Ssn} \land \\
\quad \quad s.\text{S-name} = ‘John’ \land \\
\quad \quad t.\text{Grade} = ‘A’)\}
\]

SQL:

\[
\text{SELECT t.D_code, t.C_no} \\
\text{FROM Transcript t, Student s} \\
\text{WHERE t.Student_Ssn = s.Ssn AND} \\
\quad s.S_name = ‘John’ \land \\
\quad t.Grade = ‘A’
\]
Problem 18. (Source: Given on homework by Prof. A. Brodsky in INFS 614)
Specify the following query in (i) the Relational Algebra, (ii) Relational Calculus and (iii) implement them in SQL.
List the courses (D-Code and C-No) that do not require any pre-requisites.

Proof. Algebra:
\[ \pi_{c.\text{D-code},c.\text{C-no}}(\text{Course}) - \pi_{c.\text{D-code},c.\text{C-no}}(\text{Prereq}) \]

Calculus:
\[
\{ (t.\text{D-code}, t.\text{C-no}) \mid t \in \text{Course} \land \\
\quad \neg(\exists p \in \text{Prereq}, \\
\quad \quad p.\text{D-code} = t.\text{D-code} \land \\
\quad \quad p.\text{C-no} = t.\text{C-no}) \}
\]

SQL:
\[
(\text{SELECT } c.\text{D-code}, c.\text{C-no} \\
\text{FROM Course } c \\
\text{WHERE } c.\text{D-code}, c.\text{C-no}) \\
\text{EXCEPT} \\
(\text{SELECT } p.\text{D-code}, p.\text{C-no} \\
\text{FROM Prereq } p)
\]

Problem 19. (Source: Given on homework by Prof. A. Brodsky in INFS 614)
Specify the following query in (i) the Relational Algebra, (ii) Relational Calculus and (iii) implement them in SQL.
Give the students (Ssn) who are enrolled in INFS614 (i.e., D-code=“INFS” and C-no=“614”) and have satisfied all its prerequisites.
Problem 20. For the schemas given in (3), implement the following queries in SQL (Oracle):

1. List the names of students who are enrolled in 4 or less classes. List these students by their major.
2. List the students (SSN and Name) along with the number of courses they have taken (in Transcripts). If a student has not taken any course, that student should also be listed (with 0 as the number of courses). The list should be ordered by the number of courses (in an ascending order), and in case of a tie, by the SSN of the students.
3. List the classes that require the least number of prerequisite courses among those which do require some prerequisite courses (i.e., if a class does not require any prerequisite, don’t count it).
4. List the department chairs (SSN) whose department has a name that starts with ‘I’.
5. Find the number of students who do have a major (major is not ‘NULL’).
6. List only the students (SSN and name) who satisfy the condition that the number of units they are currently taking is greater than 9.
7. List the GPA (along with SSN and Name) of all students. Assume we only have grades A, B, C, and F, and use A=4.0, B=3.0, C=2.0, and F=0.0 when GPA are calculated. Ignore the students who have not finished any courses. Hint: use decode.
8. For each course (in the transcript table), give the percentage of the students who have not got an ‘A’.
9. For each course (d-code, c-num) in the INFS program give the number of students who got ‘A’, the number of students who got ‘B’, the number of students who got ‘C’ and the number of students who got ‘F’. The 4 numbers must be shown in the same row as the d-code and c-num. If a course does not show up in the transcript table, then give 4 zeros. You must use outer-join for this query. Hint: use decode.
10. List the courses (d-code and c-num) that have exactly 3 prerequisite courses.

(Source: Given by Prof. A. Brodsky in INFS 614)

Problem 21. (Source: Given by Prof. A. Brodsky on an INFS 614 midterm)

Answer the following query in Tuple Relational Calculus, Relational Algebra, and SQL:

Find students (SSN and Name) who are currently taking classes (in Enrollment) taught by faculty from the Math or Computer Science department. Note that a faculty teaching a class offered by a department (D-code in Class) may belong to a different department (D-code in Faculty).

Problem 22. (Source: Given by Prof. A. Brodsky on an INFS 614 midterm)

Answer the following query in Tuple Relational Calculus, Relational Algebra, and SQL:

Find students (SSN and Name) who are currently taking ONLY classes (in Enrollment) taught by faculty from the Math or Computer Science department, i.e., are not taking classes taught by instructors who are not from the Math or Computer Science department. Note that a faculty teaching a class offered by a department (D-code in Class) may belong to a different department (D-code in Faculty).

Problem 23. (Source: Given by Prof. A. Brodsky on an INFS 614 midterm)

Answer the following query in SQL:

For every student (SSN and S-name) find the total number of classes this student has taken.
Problem 24. (Source: Given by Prof. A. Brodsky on an INFS 614 midterm)
Answer the following query in SQL:

For every student (SSN and S-name) find the total number of classes this student has taken (in Transcript) from a faculty in the Math or Computer Science department.

Problem 25. (Source: Given by Prof. A. Brodsky on an INFS 614 midterm)
Answer the following query in SQL:

For every student (SSN and S-name), compute this student’s GPA for courses he/she has taken (in Transcript) from a faculty in the Math or Computer Science department.

3 Normalization

3.1 Functional Dependency

Definition 17. Let $R$ be a relational schema and let $X$ and $Y$ be nonempty sets of attributes in $R$. We say that an instance $r$ of $R$ satisfies the functional dependency (FD) $X \rightarrow Y$ if the following holds for every pair of tuples $t_1$ and $t_2$ in $r$:

$$\text{If } t_1.X = t_2.X, \text{ then } t_1.Y = t_2.Y.$$  

Equivalently, $\pi_{X,Y}(R)$ is a function, i.e. $Y$ is a function of $X$.

Note that the empty table satisfies every FD.

Definition 18. The set of all functional dependencies implied by a given set $F$ of FDs is called the **closure** of $F$, denoted as $F^+$.

Definition 19. Armstrong’s Axioms for inferring FD entailment are given as follows, where $X, Y$ and $Z$ denote sets of attributes over a relational schema $R$:

- **Reflexivity**: If $X \supseteq Y$, then $X \rightarrow Y$.
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$.
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Theorem 1. Armstrong’s axioms are **sound**, in that they generate only FDs in $F^+$ when applied to a set $F$ of FDs. They are also **complete**, in that repeated application of these rules will generate all FDs in the closure $F^+$.

Definition 20. These additional rules are sound, and they are provable via Armstrong’s axioms:

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

Definition 21. The attribute closure $X^+$ with respect to a set $F$ of FDs is the set of attributes $A$ such that $X \rightarrow A$ can be inferred using the Armstrong Axioms.

Theorem 2. The following algorithm computes the attribute closure $X^+$ of the attribute set $X$ with respect to the set of FDs $F$.

```
closure ← X
repeat
  if there is an FD $U \rightarrow V$ in $F$ such that $U \subseteq closure$ then
    closure ← closure $\cup$ $V$
until there is no change.
```
This is essentially a forward-chaining algorithm for inference.

**Theorem 3.** The following expression provides an alternative to Armstrong’s Axioms which is sound and complete:

\[ F \models A \rightarrow C \iff C \in A^+ \]

### 3.2 Decomposition

**Definition 22.** A decomposition of a relation schema \( R \) consists of replacing the relation schema by two (or more) relation schemas that each contain a subset of the attributes of \( R \) and together include the attributes in \( R \).

**Definition 23.** Let \( R \) be a relational schema and let \( F \) be a set of FDs over \( R \). A decomposition of \( R \) into two schemas with attribute sets \( X \) and \( Y \) is said to be a lossless-join decomposition with respect to \( F \) if, for every instance \( r \) of \( R \) that satisfies the dependencies in \( F \), we have

\[ \pi_X(r) \bowtie \pi_Y(r) = r. \]

In other words, we can recover the original relation from the decomposed relations.

**Theorem 4.** Let \( R \) be a relation and \( F \) be a set of FDs that hold over \( R \). The decomposition of \( R \) into relations with attribute sets \( R_1 \) and \( R_2 \) is lossless if and only if \( F^+ \) contains either the FD \( R_1 \cap R_2 \rightarrow R_1 \) or the FD \( R_1 \cap R_2 \rightarrow R_2 \).

In other words, a decomposition of \( R \) into \( R_1 \) and \( R_2 \) is lossy if \( R_1 \cap R_2 \) is not a superkey for either \( R_1 \) or \( R_2 \). This means that if we compute \( R_1 \bowtie R_2 \), we get extra tuples that weren’t in the original relation. We have lost information about how the cross product should be constrained to recover the original relation.

**Definition 24.** Let \( R \) be a relational schema that is decomposed into two schemas with attribute sets \( X \) and \( Y \), and let \( F \) be a set of FDs over \( R \). The projection of \( F \) on \( X \) is the set of FDs in the closure \( F^+ \) that involve only the attributes in \( X \). We denote the projection of \( F \) on attributes \( X \) as \( F_X \). Note that a dependency \( U \rightarrow V \) in \( F^+ \) is in \( F_X \) only if all the attributes in \( U \) and \( V \) are in \( X \).

**Definition 25.** The decomposition of relational schema \( R \) with FDs \( F \) into schemas with attribute sets \( X \) and \( Y \) is dependency-preserving if \( (F_X \cup F_Y)^+ = F^+ \). That is, if we take the dependencies in \( F_X \) and \( F_Y \) and compute the closure of their union, we get back all dependencies in the closure of \( F \).

In other words, if we can deduce all the same functional dependencies by looking only at \( X \) and \( Y \) separately, then the decomposition is dependency-preserving.

### 3.3 Normal Forms

**Definition 26.** A relation is in first normal form (1NF) if every field contains only atomic values, that is, no lists or sets. This is implicit in our definition of the relational model.

1NF also eliminates multiple occurrences of the same attribute in the same tuple.

**Definition 27.** Let \( R \) be a relation schema, \( F \) be the set of FDs given to hold over \( R \), \( X \) be a subset of the attributes of \( R \), and \( A \) be an attribute of \( R \). \( R \) is in **Boyce-Codd normal form** if, for every FD \( X \rightarrow A \) in \( F^{+'} \), one of the following statements is true:

- \( A \in X \); that is, it is a trivial FD, or
- \( X \) is a superkey.
**Definition 28.** Let $R$ be a relation schema, $F$ be the set of FDs given to hold over $R$, $X$ be a subset of the attributes of $R$, and $A$ be an attribute of $R$. $R$ is in **third normal form** (3NF) if, for every FD $X \rightarrow A$ in $F$, one of the following statements is true:

- $A \in X$: that is, it is a trivial FD, or
- $X$ is a superkey, or
- $A$ is part of some candidate key for $R$.

Since these definitions only require any condition (not all) to be true, BCNF is more specific than 3NF. Any BCNF relation schema is also in 3NF.

BCNF guarantees a lossless join decomposition. In general, however, there may not be a dependency-preserving decomposition into BCNF. This means joins must be executed to test constraints—a costly computation that must be executed every time there is a change. Not desirable!

3NF relaxes the constraints of BCNF just enough to guarantee both a lossless-join and dependency preservation. The cost is that 3NF allows some kinds of redundancy.

**Example of redundancy in 3NF:** let $R$ have attributes $SBDC$ and FD’s $S \rightarrow C$ and $C \rightarrow S$. $S$ is not a key, and $C$ is part of the key $BDC$, so $R$ is in 3NF. But $(S,C)$ pairs are stored redundantly if they appear in multiple tuples.

The dependencies that 3NF **disallows** are separated into two cases:

**Definition 29.** If an FD $X \rightarrow A$ violates 3NF, and $X$ is a proper subset of some key $K$, then we call $X \rightarrow A$ a **partial dependency**.

2NF is a normal form that eliminates partial dependency.

**Definition 30.** If an FD $X \rightarrow A$ violates 3NF, and $X$ is not a proper subset of some key $K$, then we call $X \rightarrow A$ a **transitive dependency**.

The elimination of transitive dependencies is the difference between 2NF and 3NF.

**Theorem 5** (Ramakrishnan & Gehrke, 2003, p. 623). The following algorithm will decompose the relational schema $R$ into Boyce-Codd Normal Form:

1. Suppose that $R$ is not in BCNF and $F$ is the set of FDs given to hold over $R$. Let $X \subset R$, $A$ be a single attribute in $R$, and $X \rightarrow A$ be an FD in $F$ that causes a violation of BCNF. Decompose $R$ into $R - A$ and $XA$.
2. If either $R - A$ or $XA$ is not in BCNF under the projections $F_{R-A}$ and $F_{XA}$, respectively, decompose them further by a recursive application of this algorithm.

In general there is not a unique BCNF decomposition. “The theory of dependencies can tell us when there is redundancy and give us clues about possible decompositions to address the problem, but it cannot discriminate among decomposition alternatives” [Ramakrishnan & Gehrke, 2003, p. 624].

**Definition 31.** The **cover** $(F)$ of $F$ is given by:

$$
\{ X \rightarrow A \mid A \text{ is a single attribute} \\
\wedge F \models X \rightarrow A \\
\wedge X \rightarrow A \text{ is not trivial} \\
\wedge X \text{ is minimal} \}
$$
Definition 32. A minimal cover for a set $F$ of FDs is a set $G$ of FDs such that:

1. Every dependency in $G$ is of the form $X \rightarrow A$, where $A$ is a single attribute.
2. The closure $F^+$ is equal to the closure $G^+$.
3. If we obtain a set $H$ of dependencies from $G$ by deleting one or more dependencies or by deleting attributes from a dependency in $G$, then $F^+ \neq H^+$, i.e. the closure changes.

We care about the minimal cover because it gives us a way to define what relation schemas we need to add to a BCNF decomposition to make it a dependency-preserving 3NF decomposition.

"Intuitively, a minimal cover for a set $F$ of FDs is an equivalent set of dependencies that is minimal in two respects: (1) Every dependency is as small as possible; that is, each attribute on the left side is necessary and the right side is a single attribute. (2) Every dependency in it is required for the closure to be equal to $F^+$" [Ramakrishnan & Gehrke, 2003, p. 625].

Given a set $F$ of FDs, a minimal cover can be obtained by the following algorithm:

1. Create a set $G$ of FDs equivalent to $F$ with a single attribute on the RHS via the decomposition axiom.
2. For each FD in $G$, check each attribute on the LHS to see if it can be deleted without changing what can be inferred. If so, remove the attribute.
3. Check each remaining FD to see if it can be derived from the others without changing the equivalence to $F^+$. If so, remove it.

Armed with a minimal cover, we can now give an algorithm for a dependency-preserving 3NF decomposition:

**Theorem 6.** Let $R$ be a relation with a set $F$ of FDs that is a minimal cover, and let $R_1, R_2, \ldots, R_n$ be a lossless-join decomposition of $R$ s.t. each $R_i$ is in 3NF. Then the following algorithm produces a dependency-preserving 3NF decomposition of $R$:

1. Identify the set $N$ of dependencies in $F$ that is not preserved, that is, is not included in the closure of the union of $F_i$’s (where $F_i$ denotes the projection of $F$ onto the attributes in $R_i$).
2. For each FD $X \rightarrow A \in N$, create a relation schema $XA$ and add it to the decomposition of $R$.

3.4 Problems

**Problem 26.** Consider a relation schema $R(A,B,C)$ and its relation instance as follows:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Which of the following functional dependencies are satisfied by the above relation instance. If the dependency is not satisfied, explain why by specifying the tuples (i.e., the counterexample) that cause the violation:

1. $AB \rightarrow C$
2. $A \rightarrow B$
3. $C \rightarrow A$
4. $BC \rightarrow A$
5. $ABC \rightarrow A$
6. $AB \rightarrow AC$

(Source: Given by Prof. A. Brodsky in INFS 614)
Problem 27. Consider relation schema \( R(A, B, C) \) and the set of functional dependencies: \( F = \{B \rightarrow A, A \rightarrow C\} \). Do the following:

1. List all the non-trivial FDs that are implied by \( F \).
2. Compute the cover of \( F \), i.e., the set of all non-trivial FDs that are implied by \( F \), with a single attribute on the right and a minimal left hand side.
3. Find a non-empty instance of \( R \) (i.e., give a number of rows) that satisfies every FD in \( F \).
4. Can you find an instance that satisfies every FD in \( F \), but does not satisfy the FD \( AB \rightarrow C \)? If yes, give the instance. If not, explain why.

(Source: Given by Prof. A. Brodsky in INFS 614)

Problem 28. Consider the two following set of functional dependencies:

\[
F = \{B \rightarrow CE, E \rightarrow D, E \rightarrow CD, B \rightarrow CE, B \rightarrow A\}
\]

and

\[
G = \{E \rightarrow CD, B \rightarrow AE\}
\]

Are they equivalent? Explain your answer.

(Source: Given by Prof. A. Brodsky in INFS 614)

Problem 29. Consider the following relation schema \( R(A, B, C, D, E, F, G, H, I, J) \) and the set of functional dependencies \( F = \{A \rightarrow DE, IJ \rightarrow H, I \rightarrow A, J \rightarrow FG, G \rightarrow BC\} \).

1. Is \( R \) in BCNF? Justify your answer.

   Proof. No. If \( R \) is in BCNF, then either \( A \rightarrow DE \) must be trivial (which it is not) or \( A \) must be a superkey. \( A \) cannot be a superkey, because it does not appear on the LHS of any other FD and thus determines only \( D \) and \( E \), which determine nothing else. Thus \( R \) is not in BCNF.

2. Is \( R \) in 3NF? Justify your answer.

   Proof. No. If \( R \) is in 3NF, then each FD in \( F \) is either trivial, or the LHS is a superkey, or the RHS is part of some key. We already showed that the dependency \( A \rightarrow DE \) does not satisfy either of the first two options. Furthermore, \( DE \) is not part of some key, because the only key is \( IJ \). Thus \( R \) is not in 3NF.
Problem 30. (Source: Given by Prof. A. Brodsky in INFS 614) Consider a relation schema R(A, B, C, D, E) with the FD’s C → E, D → BC, E → D, B → A and A → D. This relation is in BCNF.

1. Explain why it is in BCNF.
   
   Proof. By inspection we can see that e is a superkey. Also, C is a superkey since we have C → E, D is a superkey since by D → BC we have D → C, and A is a superkey since we have A → D, and thus B is a superkey since we have B → A. Thus the LHS of every functional dependency is a superkey. Thus R is in BCNF. □

2. Now, suppose you decompose R into relations S(C, D, E) and T(A, B, D). Is this a lossless join decomposition?
   
   Proof. S and T are a lossless-join decomposition of R iff CDE ∩ ABD = D determines all the attributes in either S or T for every instance of R. By the FDs D → BC and C → E we infer that D → CDE. Therefore D is a superkey of S, and the decomposition is not lossy. □

3. Give all the non-trivial FD’s for relation S (i.e., non-trivial FDs involving only C, D, E).
   
   Proof. By drawing a graph of all the FD’s, it’s easy to see that every attribute determines every other attribute – i.e. all attributes in R are candidate keys. The projection on S is thus
   
   \[ F^{+}_{CDE} = \{ C \rightarrow D, C \rightarrow E, D \rightarrow C, D \rightarrow E, E \rightarrow C, E \rightarrow D, \]
   
   \[ C \rightarrow CE, D \rightarrow CE, E \rightarrow CD \]
   
   \[ CE \rightarrow D, CDE \rightarrow E, DCE \rightarrow C \}, \]
   
   where the second row is generated by the union axiom, and the third from augmentation and reflexivity. □

4. Give all the non-trivial FD’s for relation T (i.e., non-trivial FDs involving only A, B, D).
   
   Proof. Proceeding identically as we did for S,
   
   \[ F^{+}_{ABD} = \{ A \rightarrow B, A \rightarrow D, B \rightarrow A, B \rightarrow D, D \rightarrow A, D \rightarrow B, \]
   
   \[ A \rightarrow BD, B \rightarrow AD, D \rightarrow AB, \]
   
   \[ AD \rightarrow B, AB \rightarrow D, BD \rightarrow A \}. \]
   
   □

5. Does this decomposition preserve dependencies?
   
   Proof. Yes. The decomposition of R into S and T is dependency-preserving iff \( (F_{CDE} \cup F_{ABD})^{+} = F^{+} \). Observe that the FD’s in F imply that all attributes determine all attributes, i.e. that \( F^{+} \) contains all possible FD’s over ABCDE. Now, from only the FD’s given in (3) and (4), which are derivable from \( F_{CDE} \cup F_{ABD} \), we see that all attributes determine D and D determines all attributes. By transitivity, all attributes determine all attributes, and thus \( (F_{CDE} \cup F_{ABD})^{+} = F^{+} \). □
Problem 31. (Source: Given by Prof. A. Brodsky in INFS 614) Consider the relational schema
\( R(A, B, C, D, E, F) \) with the following functional dependencies: \( AC \rightarrow F, B \rightarrow D, AB \rightarrow CEF, ACE \rightarrow B, \) and \( AEF \rightarrow BC \). Do the following:
1. Give all the candidate keys for relation schema \( R(A, B, C, D, E, F) \) (under the set semantics).

Proof.
We can construct all candidate keys via a sequential search of the power set of \( R \)'s attributes by reasoning as follows:

- If we add \( B \), then we have \( AB \rightarrow CEF \) and \( B \rightarrow D \). Thus \( AB \) is a candidate key.
- If we instead add \( C \) to \( A \), the we have \( AC \rightarrow F \). We cannot add \( B \) because \( AB \) is a candidate key (making \( C \) superfluous). To infer more than \( A \), \( C \) and \( F \), we must add \( E \) to \( AC \). With \( ACE \) we infer \( B \), and by \( B \rightarrow D \) we have \( D \). So \( ACE \) is a candidate key (neither \( AE \) nor \( AC \) are superkeys, thus \( A \), \( C \) and \( E \) are each essential).
- If we instead add \( E \) to \( A \), then we can infer nothing else from \( AE \) alone until we add one or more of \( B \), \( C \) or \( F \). We cannot add \( B \) because \( AB \) is a candidate key (making \( E \) superfluous).
  - If we add \( C \), we have \( ACE \), which we have already shown is a candidate key.
  - If we add \( F \), we have \( AEF \), which determines \( BC \) and thus all attributes. \( AEF \) is a candidate key, because neither \( AE \) nor \( AF \) are superkeys (thus \( A \), \( E \) and \( F \) are each essential).
- If we instead add \( F \) to \( A \), then we can infer nothing else from \( AF \) alone until we add one or more of \( B \), \( C \) or \( E \). We cannot add \( C \), because \( AC \rightarrow F \), making \( F \) superfluous. So we must add \( E \), but we have already shown that \( AEF \) is a candidate key.

Thus the only candidate keys for \( R \) are \( AB, ACE, AEF \).

2. Compute the cover of \( F \), i.e., the set of all non-trivial FDs that are implied by \( F \), with a single attribute on the right and a minimal left hand side.

Proof. No. Please stop asking me to manually solve combinatorially complex problems. It’s cruel.

3. Is relation \( R \) in the 3NF? If not, give an example of FD that violates the 3NF condition and explain why.

Proof. No. The FD \( B \rightarrow D \) holds over \( R \), but it is not trivial, \( B \) is not a superkey, and \( D \) is not part of any of the candidate keys \( (AB, ACE \) or \( AEF) \).

4. Is relation \( R \) in BCNF? If not, give an example FD that violates the BCNF condition and explain why.

Proof. No. The FD \( B \rightarrow D \) holds over \( R \), but it is not trivial, and \( B \) is not a superkey.
Problem 32. (Source: Given by Prof. A. Brodsky in INFS 614) Consider the relational schema \( R(A, B, C, D, E, F) \) with the following functional dependencies: \( AC \rightarrow F, B \rightarrow D, AB \rightarrow CEF, ACE \rightarrow B, \) and \( AEF \rightarrow BC. \) Do the following:

1. Decompose the relational schema into schemas that satisfy BCNF.

   Proof. Of the five FD’s given to hold over \( R, \) the following three are in violation of BCNF because their LHS is not a superkey: \( AC \rightarrow F, B \rightarrow D, AB \rightarrow CEF. \)

   First let’s deal with \( B \rightarrow D. \) Decompose \( R \) into \( ABCEF \) and \( BD. \) Since any relation schema over two attributes is in BCNF, we’re done with \( BD. \) Observe that \( AB \) is a key for \( ABCEF. \) Thus, in the new relation \( ABCEF, \) the FD \( AB \rightarrow CEF \) is no longer in violation of BCNF. The FD \( AC \rightarrow F \) is still in violation, however.

   Now decompose \( ABCEF \) into \( ABCE \) and \( ACF. \) Only the FD \( ACE \rightarrow B \) applies to \( ABCE, \) and it is in BCNF. Only the FD \( AC \rightarrow F \) applies to \( ACF, \) and it is in BCNF since \( AC \) is a key.

   Thus the decomposition of \( R \) into the three schemas \( ABCE, ACF \) and \( BD \) satisfies BCNF. \( \square \)

2. Does the result of your decomposition preserve dependencies?

   Proof. Call \( F \) the original five functional dependencies given to hold over \( R. \) Now \( F_{ABCE} = \{ AB \rightarrow CEF, ACE \rightarrow B \}, F_{ACF} = \{ AC \rightarrow F \}, \) and \( F_{BD} = \{ B \rightarrow D \}. \) Note that, through the projections, we have discarded the functional dependency \( AEF \rightarrow BC. \)

   In general, the decomposition preserves dependencies iff \( (F_{ABCE} \cup F_{ACF} \cup F_{BD})^+ = F^+. \) However, since \( F \subseteq F^+, \) if any FD in \( F \) is not contained in \( (F_{ABCE} \cup F_{ACF} \cup F_{BD})^+, \) then the closure of the union of projections cannot equal \( F^+. \) In particular, if we cannot derive \( AEF \rightarrow BC \) from the four FD’s in the projections, then the decomposition is not dependency-preserving.

   Consider the four FD’s in the projections, \( F_P = \{ AC \rightarrow F, B \rightarrow D, AB \rightarrow CEF, ACE \rightarrow B \}. \) Now we use an exhaustive backward-chaining argument to show that the FD \( AEF \rightarrow BC \) cannot be inferred from the FDs in \( F_P, \) i.e. that \( F_P \nvdash AEF \rightarrow BC. \)

   • \( AEF \rightarrow BC \) can be decomposed into \( AEF \rightarrow B \) and \( AEF \rightarrow C. \)

   • We can only derive an FD with \( B \) on the RHS by apply Armstrong’s axioms to \( ACE \rightarrow B, \) because this is the only FD in \( F_P \) with \( B \) appearing on the RHS.

   • Stated differently, \( F_P \vDash AEF \rightarrow B \) only if we have some means to show that \( F_P \vDash AEF \rightarrow ACE, \) i.e. \( F_P \vDash AEF \rightarrow C. \)

   • We can only derive an FD with \( C \) on the RHS by applying Armstrong’s axioms to \( AB \rightarrow CEF, \) because this is the only FD in \( F_P \) with \( C \) appearing on the RHS.

   • Therefore \( F_P \vDash AEF \rightarrow C \) only if we have some means to show that \( F_P \vDash AEF \rightarrow AB, \) i.e. \( F_P \nvdash AEF \rightarrow B. \)

   • We have exhausted all the means available to try and prove that \( F_P \vDash AEF \rightarrow B \) or \( F_P \vDash AEF \rightarrow C, \) but all we have been able to conclude is that one entailment is true if the other is true and vice versa. Therefore there is way to prove that either of them are true given the FD’s in \( F_P. \)

   • Therefore \( F_P \nvdash AEF \rightarrow BC. \)

   Thus \( F_P^+ \) does not contain \( AEF \rightarrow BC, \) and the closure of the union of the projections of \( F \) cannot be equal to \( F^+. \) Thus the decomposition is not dependency-preserving. \( \square \)