(1) the SQL code you have used to create the schema of your database (only create table and alter table statements (if any), not statements for inserting values)

**Data Definition Language (DDL)**

**CARDS**

```
CREATE TABLE cards(

idx int,

asciiName varchar(150),

convertedManaCost mediumint,

edhrecRank smallint,

faceConvertedManaCost mediumint,

faceManaValue mediumint,

faceName varchar(150),

isReserved bool,

layout varchar(20) NOT NULL,

loyalty varchar(5),

manaCost varchar(50),

manaValue mediumint NOT NULL,

name varchar(150) PRIMARY KEY,

power varchar(5),

text text,

toughness varchar(5)

);

ALTER TABLE cards DROP COLUMN idx;
```

**COLORS**

```
CREATE TABLE colors(

idx int,

card varchar(150),

color char(1),

PRIMARY KEY (card, color),

FOREIGN KEY (card) REFERENCES cards(name)

);

ALTER TABLE colors DROP COLUMN idx;
```

**KEYWORDS**

```
CREATE TABLE keywords(

idx int,

card varchar(150),

keyword varchar(20),

PRIMARY KEY (card, keyword),

FOREIGN KEY (card) REFERENCES cards(name)

);

ALTER TABLE keywords DROP COLUMN idx;
```

**LEGALITIES**

```
CREATE TABLE legalities(

idx int,

card varchar(150) PRIMARY KEY,

commander enum(“Legal”,”Not Legal”,”Banned”,”Restricted”) NOT NULL,

historic enum(“Legal”,”Not Legal”,”Banned”,”Restricted”) NOT NULL,
legacy enum(“Legal”,”Not Legal”,”Banned”,”Restricted”) NOT NULL,
modern enum(“Legal”,”Not Legal”,”Banned”,”Restricted”) NOT NULL,
pauper enum(“Legal”,”Not Legal”,”Banned”,”Restricted”) NOT NULL,
standard enum(“Legal”,”Not Legal”,”Banned”,”Restricted”) NOT NULL,
vintage enum(“Legal”,”Not Legal”,”Banned”,”Restricted”) NOT NULL,
FOREIGN KEY (card) REFERENCES cards(name)
);
ALTER TABLE legalities DROP COLUMN idx;
```

**RULINGS**

```
CREATE TABLE rulings(
idx int,
card varchar(150),
date date,
ruling text NOT NULL,
dailycount tinyint,
PRIMARY KEY (card, date, dailycount),
FOREIGN KEY (card) REFERENCES cards(name)
);
ALTER TABLE rulings DROP COLUMN idx;
```

**SUBTYPES**

```
CREATE TABLE subtypes(
idx int,
card varchar(150),
subtype varchar(30),
PRIMARY KEY (card, subtype),
FOREIGN KEY (card) REFERENCES cards(name)
);
ALTER TABLE subtypes DROP COLUMN idx;
```

**SUPERTYPES**

```
CREATE TABLE supertypes(
idx int,
card varchar(150),
supertype varchar(30),
PRIMARY KEY (card, supertype),
FOREIGN KEY (card) REFERENCES cards(name)
);
ALTER TABLE supertypes DROP COLUMN idx;
```

**TYPES**

```
CREATE TABLE types(
idx int,
card varchar(150),
type varchar(30),
PRIMARY KEY (card, type),
FOREIGN KEY (card) REFERENCES cards(name)
);
ALTER TABLE types DROP COLUMN idx;
```

**SETS**

```
CREATE TABLE sets(

idx int,

baseSetSize smallint NOT NULL,

block varchar(30),

code varchar(10),

isFoilOnly bool NOT NULL,

isOnlineOnly bool NOT NULL,

name varchar(50) UNIQUE,

releaseDate date NOT NULL,

totalSetSize smallint NOT NULL,

type varchar(30) NOT NULL,

PRIMARY KEY (code)

);

ALTER TABLE sets DROP COLUMN idx;

ALTER TABLE sets DROP COLUMN baseSetSize;
```

**PRINTINGS**

```
CREATE TABLE printings(

idx int,

artist varchar(50),

frameVersion varchar(10) NOT NULL,

isFullArt bool,

isPromo bool,
```

```sql
isTextless bool,
name varchar(150) NOT NULL,
rarity varchar(10) NOT NULL,
setCode varchar(10) NOT NULL,
uuid char(36),
PRIMARY KEY (uuid),
FOREIGN KEY (name) REFERENCES cards(name),
FOREIGN KEY (setCode) REFERENCES sets(code)
);
ALTER TABLE printings DROP COLUMN idx;
```

**PRICES**

```sql
CREATE TABLE prices(
idx int,
uuid char(36),
normal float,
foil float,
PRIMARY KEY (uuid)
);
ALTER TABLE prices DROP COLUMN idx;
ALTER TABLE prices ADD FOREIGN KEY (uuid) REFERENCES printings(uuid);

ALTER TABLE cards DROP COLUMN asciiName;
ALTER TABLE cards DROP COLUMN faceConvertedManaCost;
ALTER TABLE cards DROP COLUMN faceManaValue;
ALTER TABLE cards DROP COLUMN faceName;

ALTER TABLE cards DROP COLUMN manaValue;
```

(2) the SQL code of the queries (possibly with an explanation)

(3) the SQL code used for query optimization for HW2. For each query, indicate the un-optimized version and the optimized one. In case the optimization has been realized through indexes, insert the SQL code for the index creation; in case you have modified the schema (e.g. changed the domain of a field, or constructed a new materialized table, etc.), insert the code you have used for this modification.

**Queries (Slow + optimized versions)**

1) Return the **cards** appearing in the **top 100** of the **EDHRec.com Ranking**, having at least one **printing** whose **normal price** is **less than one tenth** of the **average normal price** (order the result by **price**, in ascending order)

```
SELECT ptg.name AS name, min(p.normal) AS minPrice

FROM prices p NATURAL JOIN printings ptg

WHERE p.normal < 1/10* (

    SELECT avg(p.normal)

   FROM prices p)

GROUP BY ptg.name

HAVING ptg.name in (

    SELECT c.name

   FROM cards c

   WHERE edhrecRank <= 100 AND edhrecRank IS NOT NULL)

ORDER BY minPrice
```

2) Return the **average text length** of the cards in each **block** of sets (order the result by **average text length**, in descending order)

```
SELECT s.block AS Block, avg(length(c.text)) AS AvgTextLength

FROM cards c, printings ptg, sets s

WHERE c.name = ptg.name AND s.code = ptg.setCode

GROUP BY s.block

HAVING s.block IS NOT NULL

ORDER BY AvgTextLength DESC
```

**3)** Return the **artists** that realized the **maximum number of printings** for each card **type** of cards having **common** or **uncommon** rarity (return **all the artists** in case of ties)

```
WITH temp AS (SELECT t.type, ptg.artist, count(*) AS count

    FROM printings ptg JOIN types t ON ptg.name = t.card

    WHERE ptg.rarity="common" OR ptg.rarity="uncommon"

    GROUP BY t.type, ptg.artist)


SELECT temp_0.*

FROM temp AS temp_0 LEFT JOIN temp as temp_1 ON temp_0.type=temp_1.type
    AND temp_0.count < temp_1.count

WHERE temp_1.count IS NULL

ORDER BY temp_0.count DESC;
```

**4)** Return the number of **new cards** for each **set**

**SLOW VERSION [14 secs]**

```
SELECT s.name AS setName, count(*) AS count

FROM sets AS s, (SELECT ptg1.name AS name, min(s1.releaseDate) AS
firstReleaseDate

FROM printings ptg1, sets s1

WHERE ptg1.setCode = s1.code

GROUP BY ptg1.name) temp0

JOIN

(SELECT ptg2.name AS name, s2.code AS code, s2.releaseDate AS releaseDate

FROM printings ptg2, sets s2

WHERE ptg2.setCode = s2.code) temp1

ON temp0.name = temp1.name AND temp0.firstReleaseDate = temp1.releaseDate

WHERE s.code = temp1.code

GROUP BY temp1.code

ORDER BY count DESC
```

**NOT FASTER VERSION [14 secs] (ATTEMPT TO USE 2 VIEWS, ACTUALLY NOT SPEEDING THINGS UP!)**

```
CREATE VIEW view0 AS(

        SELECT ptg1.name AS name, min(s1.releaseDate) AS firstReleaseDate

        FROM printings ptg1, sets s1

        WHERE ptg1.setCode = s1.code

        GROUP BY ptg1.name);


CREATE VIEW view1 AS(

        SELECT ptg2.name AS name, s2.code AS code, s2.releaseDate AS
releaseDate
```

```
    FROM printings ptg2, sets s2

    WHERE ptg2.setCode = s2.code);


SELECT s.name AS setName, count(*) AS count

FROM sets AS s, view0 AS temp0 JOIN view1 AS temp1 ON temp0.name = temp1.name AND temp0.firstReleaseDate = temp1.releaseDate

WHERE s.code = temp1.code

GROUP BY temp1.code

ORDER BY count DESC
```

**FAST VERSION [Creating 2 materialized views: (0.85+0.55) secs; Query execution: 1.5 secs] (ADDING 2 MATERIALIZED VIEWS). It is about 9 times FASTER (5 times considering materialized views building for the first execution)**

```
CREATE TABLE mat_view0 AS(

    SELECT ptg1.name AS name, min(s1.releaseDate) AS firstReleaseDate

    FROM printings ptg1, sets s1

    WHERE ptg1.setCode = s1.code

    GROUP BY ptg1.name);


CREATE TABLE mat_view1 AS(

    SELECT ptg2.name AS name, s2.code AS code, s2.releaseDate AS releaseDate

    FROM printings ptg2, sets s2

    WHERE ptg2.setCode = s2.code);


SELECT s.name AS setName, count(*) AS count
```

```
FROM sets AS s, mat_view0 AS temp0 JOIN mat_view1 AS temp1 ON temp0.name
= temp1.name AND temp0.firstReleaseDate = temp1.releaseDate

WHERE s.code = temp1.code

GROUP BY temp1.code

ORDER BY count DESC
```

**5)** Return the **rulings** of the **cards** with at least a **supertype**, containing one or more of the cards **keywords**

```
SELECT supc.card AS card, supc.keyword AS keyword, r.ruling AS ruling

FROM rulings r, (SELECT *

    FROM keywords k

   WHERE EXISTS (SELECT *

        FROM supertypes AS supt

      WHERE k.card = supt.card)) supc

WHERE r.card = supc.card AND r.ruling LIKE CONCAT("%",supc.keyword,"%")
```

**6)** For each **color**, return the number of **cards** of type “**creature**” having both **power** and **toughness** larger or equal to **10** that are legal in “**Legacy**”

```
SELECT cols.color AS color, count(*) AS count

FROM cards c JOIN colors cols ON c.name = cols.card JOIN legalities l ON c.name
    = l.card JOIN types t on c.name = t.card

WHERE ((CAST(c.power AS SIGNED) > 9) AND (CAST(c.toughness AS SIGNED) >
    9)) AND t.type = "Creature" AND l.legacy = "Legal"

GROUP BY cols.color
```

**7)** For each **non-foil-only set**, return **name**, **size**, the **sum** of both the **normal** and **foil prices** (when **NOT NULL**) of each of its printings and the "**Foil Markup**" (**ratio** between **total foil** and **total normal price**)

```
SELECT s.name AS name, s.totalSetSize AS setSize, round(sum(p.normal),2) AS
    totalNormalPrice, round(sum(p.foil),2) AS totalFoilPrice,
    round((sum(p.foil)/sum(p.normal)),2) AS FoilMarkup

FROM sets s, printings ptg, prices p

WHERE s.code = ptg.setCode AND ptg.uuid = p.uuid AND s.isFoilOnly = "0"

GROUP BY ptg.setCode

HAVING FoilMarkup IS NOT NULL

ORDER BY FoilMarkup DESC
```

**8)** Return the **uuid** and the **normal price** of all the **printings** released in the 21st Century, whose **price** is less than or equal to **10€**, of **cards** that have **any color symbol** in their **mana cost** but are **colorless** (order the result by **normal price**, in descending order)

**SLOW VERSION [14 secs]**

```
SELECT p.uuid, p.normal

FROM prices p JOIN printings ptg ON p.uuid=ptg.uuid JOIN sets s ON ptg.setCode
    = s.code JOIN cards c ON ptg.name = c.name LEFT JOIN colors col ON c.name
    = col.card

WHERE p.normal <= 10 AND s.releaseDate >= "2000-01-01" AND c.manaCost
    RLIKE "[WUBRG]" AND col.card IS NULL

ORDER BY p.normal
```

**FAST VERSION [1.5 secs] (Rewriting the SQL query). It is about 9 times FASTER: AVOID MULTIPLE JOINS + ORDER BY has to deal with fewer columns (shorter rows, performance improvement since we are dealing with a row oriented DB)**

```
SELECT p.uuid, p.normal

FROM prices p

WHERE p.normal <= 10 AND p.uuid IN (

        SELECT ptg.uuid

        FROM printings ptg

   WHERE ptg.setCode IN (

                SELECT s.code

      FROM sets s

      WHERE s.releaseDate >= "2000-01-01") AND ptg.name IN (

                        SELECT c.name

         FROM cards c

         WHERE c.manaCost RLIKE "[WUBRG]" AND c.name NOT IN (

                                SELECT col.card

                                FROM colors col)))

ORDER BY p.normal
```

9) Return the **uuid** and the **normal price** of all the **printings** released in the 21st Century, whose **price** is equal to 0.5€ of **cards** that have **any color symbol** in their **mana cost** but are **colorless** (order the result by **normal price**, in descending order)

**SLOW VERSION [62K rows scan]**

```
EXPLAIN SELECT p.uuid, p.normal

FROM prices p

WHERE p.normal = 0.5 AND p.uuid IN (

        SELECT ptg.uuid
        FROM printings ptg
   WHERE ptg.setCode IN (
                SELECT s.code
      FROM sets s
      WHERE s.releaseDate >= "2000-01-01") AND ptg.name IN (
                        SELECT c.name
         FROM cards c
         WHERE c.manaCost RLIKE "[WUBRG]" AND c.name NOT IN (
                                SELECT col.card
                                FROM colors col)))
ORDER BY p.normal
```

**FAST VERSION [56 rows scan] (Adding index to speed up EQUALITY CONDITION CHECKING)**

```
CREATE INDEX pnormal ON prices(normal);


EXPLAIN SELECT p.uuid, p.normal
FROM prices p
WHERE p.normal = 0.5 AND p.uuid IN (
        SELECT ptg.uuid
        FROM printings ptg
   WHERE ptg.setCode IN (
                SELECT s.code
      FROM sets s
      WHERE s.releaseDate >= "2000-01-01") AND ptg.name IN (
                        SELECT c.name

        FROM cards c

        WHERE c.manaCost RLIKE "[WUBRG]" AND c.name NOT IN(

                                SELECT col.card

                                FROM colors col)))

ORDER BY p.normal
```

**10)** For each set, return the names of the **previous** and the **next** set by **release date** (break ties by **alphabetical** order)

**SLOW VERSION [0.7 secs]**

```
WITH temp_prev AS (SELECT s1.name AS currSet, s2.name AS prevSet,
    s2.releaseDate AS prevReleaseDate

    FROM sets s1 LEFT JOIN sets s2 ON (s1.releaseDate > s2.releaseDate OR
    (s1.releaseDate = s2.releaseDate AND s1.name > s2.name))),

   temp_next AS (SELECT s1.name AS currSet, s2.name AS nextSet,
    s2.releaseDate AS nextReleaseDate

    FROM sets s1 LEFT JOIN sets s2 ON (s1.releaseDate < s2.releaseDate OR
    (s1.releaseDate = s2.releaseDate AND s1.name < s2.name)))



SELECT prev_table.currSet, prev_table.prevSet, next_table.nextSet

FROM (SELECT temp_prev1.currSet, max(temp_prev1.prevSet) AS prevSet,
    temp_prev1.prevReleaseDate AS prevReleaseDate

FROM temp_prev AS temp_prev1

WHERE (temp_prev1.currSet, temp_prev1.prevReleaseDate) in (SELECT
    temp_prev0.currSet, max(temp_prev0.prevReleaseDate)

    FROM temp_prev AS temp_prev0

   GROUP BY temp_prev0.currSet) OR temp_prev1.prevReleaseDate IS NULL

GROUP BY temp_prev1.currSet) prev_table
```

```
NATURAL JOIN

(SELECT temp_next1.currSet, min(temp_next1.nextSet) AS nextSet,
    temp_next1.nextReleaseDate AS nextReleaseDate

FROM temp_next AS temp_next1

WHERE (temp_next1.currSet, temp_next1.nextReleaseDate) in (SELECT
    temp_next0.currSet, min(temp_next0.nextReleaseDate)

    FROM temp_next AS temp_next0

   GROUP BY temp_next0.currSet) OR temp_next1.nextReleaseDate IS NULL

GROUP BY temp_next1.currSet) next_table

ORDER BY currSet
```

**FAST VERSION [Modifying DB schema: 0.7 secs; Query execution: <0.001 secs] (MODIFYING DATABASE SCHEMA – TRADEOFF BETWEEN FUTURE PROJECTIONS/JOINS COST AND QUERY EXECUTION)**

```
ALTER TABLE sets ADD prevSet varchar(50);

ALTER TABLE sets ADD nextSet varchar(50);



WITH temp_prev AS (SELECT s1.name AS currSet, s2.name AS prevSet,
    s2.releaseDate AS prevReleaseDate

    FROM sets s1 LEFT JOIN sets s2 ON (s1.releaseDate > s2.releaseDate OR
    (s1.releaseDate = s2.releaseDate AND s1.name > s2.name))),

   temp_next AS (SELECT s1.name AS currSet, s2.name AS nextSet,
    s2.releaseDate AS nextReleaseDate

    FROM sets s1 LEFT JOIN sets s2 ON (s1.releaseDate < s2.releaseDate OR
    (s1.releaseDate = s2.releaseDate AND s1.name < s2.name)))



UPDATE sets s

SET s.prevSet = (SELECT prev_table.prevSet
    FROM (SELECT temp_prev1.currSet, max(temp_prev1.prevSet) AS prevSet,
    temp_prev1.prevReleaseDate AS prevReleaseDate

        FROM temp_prev AS temp_prev1

        WHERE (temp_prev1.currSet, temp_prev1.prevReleaseDate) in (SELECT
    temp_prev0.currSet, max(temp_prev0.prevReleaseDate)

                FROM temp_prev AS temp_prev0

                GROUP BY temp_prev0.currSet) OR temp_prev1.prevReleaseDate IS
    NULL

        GROUP BY temp_prev1.currSet) AS prev_table

   WHERE s.name = prev_table.currSet

);


WITH temp_prev AS (SELECT s1.name AS currSet, s2.name AS prevSet,
    s2.releaseDate AS prevReleaseDate

    FROM sets s1 LEFT JOIN sets s2 ON (s1.releaseDate > s2.releaseDate OR
    (s1.releaseDate = s2.releaseDate AND s1.name > s2.name))),

   temp_next AS (SELECT s1.name AS currSet, s2.name AS nextSet,
    s2.releaseDate AS nextReleaseDate

    FROM sets s1 LEFT JOIN sets s2 ON (s1.releaseDate < s2.releaseDate OR
    (s1.releaseDate = s2.releaseDate AND s1.name < s2.name)))


UPDATE sets s

SET s.nextSet = (SELECT next_table.nextSet

    FROM (SELECT temp_next1.currSet, min(temp_next1.nextSet) AS nextSet,
    temp_next1.nextReleaseDate AS nextReleaseDate

        FROM temp_next AS temp_next1

        WHERE (temp_next1.currSet, temp_next1.nextReleaseDate) in (SELECT
    temp_next0.currSet, min(temp_next0.nextReleaseDate)

                FROM temp_next AS temp_next0
```

```
                    GROUP BY temp_next0.currSet) OR temp_next1.nextReleaseDate IS
     NULL

          GROUP BY temp_next1.currSet) AS next_table

    WHERE s.name = next_table.currSet

);


SELECT s.name, s.prevSet, s.nextSet

FROM sets s
```