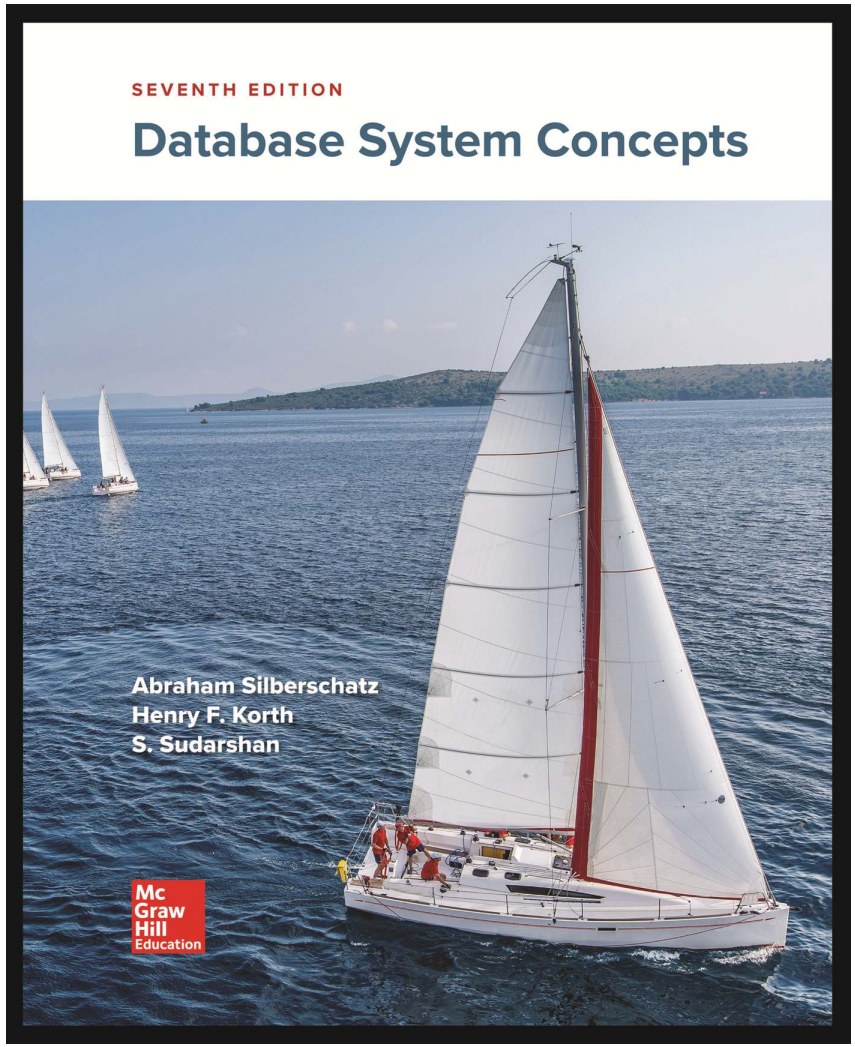
Database System Concepts, 7^{th} Edition
Chapter 4: Intermediate SQL

Silberschatz, Korth and Sudarshan

March 2, 2025

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit `https://db-book.com/`.

Plan

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Index Definition in SQL
- Authorization

Joined Relations

- Join operations take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join.
- The join operations are typically used as subquery expressions in the from clause.
- Three types of joins:
  - Natural join
  - Inner join
  - Outer join

Natural Join in SQL

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column.
- List the names of instructors along with the course ID of the courses that they taught.

```
1        SELECT
2            name, course_id
3        FROM
4            students, takes
5        WHERE
6            student.ID = takes.ID;
```

Natural Join in SQL

- Same query in SQL with "natural join" construct.

```
1  SELECT
2      name, course_id
3  FROM
4      student NATURAL JOIN takes;
```

Natural Join in SQL (Cont.)

- The `FROM` clause in can have multiple relations combined using natural join:

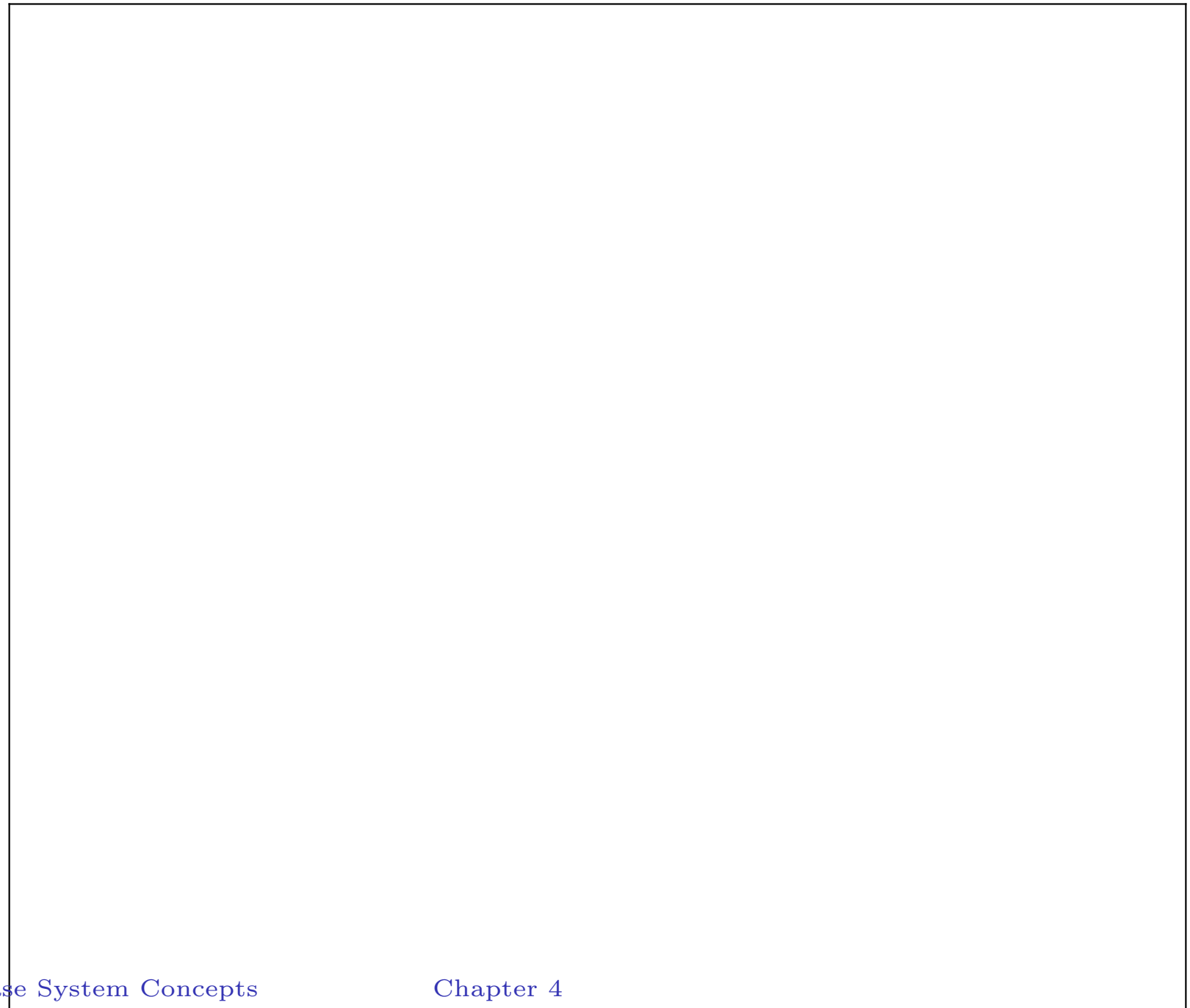
SELECT $A_1, A_2, \ldots, A_n$
FROM r_1
    NATURAL JOIN r_2
    NATURAL JOIN $\ldots$
    NATURAL JOIN r_n
WHERE P;

Student Relation

Takes Relation

pages/nj2.pdf

Student Natural Join Takes



Dangerous in Natural Join

Beware of unrelated attributes with same name which get equated incorrectly

Example:

List the names of students instructors along with the titles of courses that they have taken.

```
1    SELECT
2        name, title
3    FROM
4        student
5    natural join
6        takes
7    natural join
8        course;
```

Dangerous in Natural Join

Beware of unrelated attributes with same name which get equated incorrectly

Example:

List the names of students instructors along with the titles of courses that they have taken.

```
1    SELECT
2        name, title
3    FROM
4        student
5    natural join
6        takes
7    natural join
8        course;
```

Incorrect!

Dangerous in Natural Join

Example:

List the names of students instructors along with the titles of courses that they have taken.

```
1    SELECT
2        name, title
3    FROM
4        student
5    NATURAL JOIN
6        takes
7    NATURAL JOIN
8        course;
```

- This query omits all (student name, course title) pairs where the student takes a course in a department other than the student's own department.

Dangerous in Natural Join

Example:

List the names of students instructors along with the titles of courses that they have taken.

```
1    SELECT
2        name, title
3    FROM
4        student
5    NATURAL JOIN
6        takes, course
7    WHERE
8        takes.course_id = course.course_id;
```

- The correct version (above), correctly outputs such pairs.

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses `null` values.
- Three forms of outer join:
  - left outer join
  - right outer join
  - full outer join

Outer Join Examples

- Relation *course*:

course_id	**title**	**dept_name**	**credits**
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*:

course_id	**prereq_id**
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that
  - course information is missing for CS-437
  - prereq information is missing for CS-315

Left Outer Join

- course `NATURAL LEFT OUTER JOIN` prereq

course_id	**title**	**dept_name**	**credits**	**prereq_id**
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	`null`

- In relational algebra: course $\ltimes$ prereq

Right Outer Join

- course `NATURAL RIGHT OUTER JOIN` prereq

course_id	**title**	**dept_name**	**credits**	**prereq_id**
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	`null`	`null`	`null`	CS-101

- In relational algebra: course $\bowtie$ prereq

Full Outer Join

- course `NATURAL FULL OUTER JOIN` prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	`null`
CS-347	`null`	`null`	`null`	CS-101

- In relational algebra: course $\ltimes$ prereq

Joined Types and Conditions

- **Join operations** – take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the `FROM` clause
- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join conditions
natural
on <predicate>
using $(A_1, A_2, \ldots, A_n)$

Joined Relations – Examples

- course `NATURAL RIGHT OUTER JOIN` prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	`null`	`null`	`null`	CS-101

- course `FULL OUTER JOIN` prereq `USING` (course_id)

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	`null`
CS-347	`null`	`null`	`null`	CS-101

Joined Relations – Examples

- course `INNER JOIN` prereq `ON` course.course_id = prereq.course_id

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

- What is the difference between the above, and a natural join?

- course `LEFT OUTER JOIN` prereq `ON` course.course_id = prereq.course_id

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	`null`	`null`

Joined Relations – Examples

- course `NATURAL RIGHT OUTER JOIN` prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	`null`	`null`	`null`	CS-101

- course `FULL OUTER JOIN` prereq `USING` (course_id)

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	`null`
CS-347	`null`	`null`	`null`	CS-101

Plan

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Index Definition in SQL
- Authorization

Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
1    SELECT
2        ID, name, dept_name
3    FROM
4        instructor;
```

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

View Definition

- A view is defined using the create view statement which has the form:

  CREATE VIEW v **AS** $< query\_expression >$;

  where $< query\_expression >$ is any legal `SQL` expression. The view name is represented by v.
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

View Definition and Use

- A view of instructors without their salary.

```
1    CREATE VIEW faculty AS
2    SELECT ID, name, dept_name
3    FROM instructor
```

View Definition and Use

- A view of instructors without their salary.

```
CREATE VIEW faculty AS
SELECT ID, name, dept_name
FROM instructor
```

- Find all instructors in the Biology department

```
SELECT name
FROM faculty
WHERE dept_name = 'Biology'
```

View Definition and Use

- A view of instructors without their salary.

```
CREATE VIEW faculty AS
SELECT ID, name, dept_name
FROM instructor
```

- Find all instructors in the Biology department

```
SELECT name
FROM faculty
WHERE dept_name = 'Biology'
```

- Create a view of department salary totals

```
CREATE VIEW departments_total_salary(dept_name, total_salary) AS
SELECT dept_name, SUM(salary)
FROM instructor
GROUP BY dept_name;
```

Views Defined Using Other Views

- One view may be used in the expression defining another view.
- A view relation v_1 is said to depend directly on a view relation v_2 if v_2 is used in the expression defining v_1.
- A view relation v_1 is said to depend on view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2.
- A view relation v is said to be recursive if it depends on itself.

Views Defined Using Other Views

```
1   CREATE VIEW
2       physics_fall_2017 AS
3   SELECT
4       course.course_id, sec_id, building, room_number
5   FROM
6       course, section
7   WHERE
8       course.course_id = section.course_id AND
9       course.dept_name = 'Physics' AND
10      section.semester = 'Fall' AND
11      section.year = '2017';
```

Views Defined Using Other Views

```
1    CREATE VIEW
2        physics_fall_2017_watson AS
3    SELECT
4        course_id, room_number
5    FROM
6        physics_fall_2017
7    WHERE
8        building= 'Watson';
```

View Expansion

- Expand the view:

```
1  CREATE VIEW physics_fall_2017_watson AS
2      SELECT course_id, room_number
3      FROM physics_fall_2017
4      WHERE building= 'Watson';
```

- To:

```
1  CREATE VIEW physics_fall_2017_watson AS
2      SELECT course_id, room_number
3      FROM ( SELECT course.course_id, building, room_number
4               FROM course, section
5               WHERE course.course_id = section.course_id
6               AND course.dept_name = 'Physics'
7               AND section.semester = 'Fall'
8               AND section.year = '2017' )
9      WHERE building= 'Watson';
```

View Expansion (Cont.)

- A way to define the meaning of views defined in terms of other views.
- Let view v1 be defined by an expression e1 that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:

```
REPEAT
        Find any view relation v_i in e_i
        Replace the view relation v_i
            by the expression defining v_i
UNTIL no more view relations are present in e_i
```

- As long as the view definitions are not recursive, this loop will terminate.

Materialized Views

- Certain database systems allow view relations to be physically stored.
  - Physical copy created when the view is defined.
  - Such views are called **Materialized view**.
- If relations used in the query are updated, the materialized view result becomes out of date.
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.

Update of a View

- Add a new tuple to faculty view which we defined earlier.

```
INSERT INTO faculty
VALUES ('30765', 'Green', 'Music');
```

- This insertion must be represented by the insertion into the instructor relation.
  - Must have a value for salary.
- Two approaches:
  - Reject the insert.
  - Insert the tuple:
    (‘30765’, ‘Green’, ‘Music’, null)
    into the *instructor* relation.

Some Updates Cannot be Translated Uniquely

```
CREATE VIEW instructor_info AS
    SELECT
        ID, name, building
    FROM
        instructor, department
    WHERE
        instructor.dept_name= department.dept_name;
```

```
INSERT INTO instructor_info
VALUES ('69987', 'White', 'Taylor');
```

- Issues:
  - Which department, if multiple departments in Taylor?
  - What if no department is in Taylor?

And Some Not at All

```
CREATE VIEW history_instructors AS
    SELECT *
    FROM instructor
    WHERE dept_name= 'History';
```

- What happens if we insert:
  (‘25566’, ‘Brown’, ‘Biology’, 100000)
  into *history_instructors*?

View Updates in SQL

- Most `SQL` implementations allow updates only on simple views:
  - The `FROM` clause has only one database relation.
  - The `SELECT` clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
  - Any attribute not listed in the `SELECT` clause can be set to `null`.
  - The query does not have a `GROUP BY` or `HAVING` clause.

Plan

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Index Definition in SQL
- Authorization

Plan

Join Expressions

Views

Transactions

Integrity Constraints

SQL Data Types and Schemas

Index Definition in SQL

Authorization

Plan

Join Expressions

Views

Transactions

Integrity Constraints

SQL Data Types and Schemas

Index Definition in SQL

Authorization

Plan

Join Expressions

Views

Transactions

Integrity Constraints

SQL Data Types and Schemas

Index Definition in SQL

Authorization

Plan

Join Expressions

Views

Transactions

Integrity Constraints

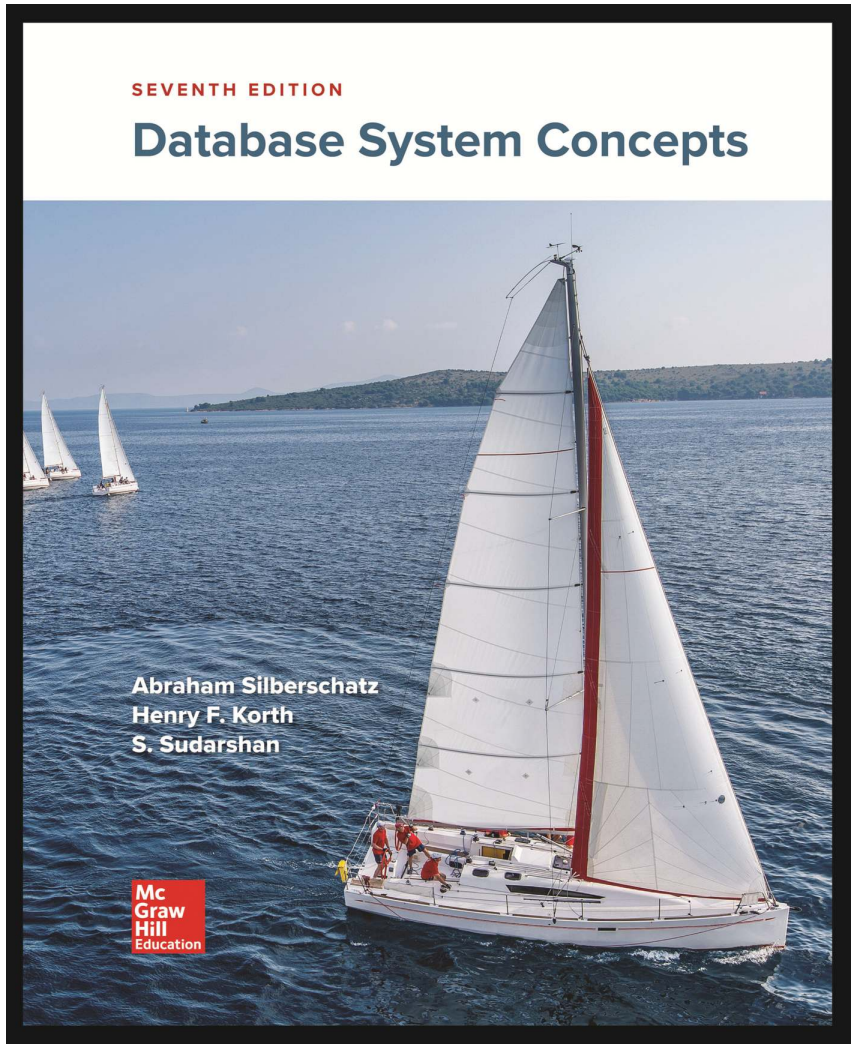
SQL Data Types and Schemas

Index Definition in SQL

Authorization

1

Database System Concepts



Content has been extracted from *Database System Concepts*, Seventh Edition, by Silberschatz, Korth and Sudarshan. Mc Graw Hill Education. 2019.
Visit `https://db-book.com/`.