CS331-008 – Group 11

Tyler Ambrozy – Ta389@njit.edu

Mark Youssef – May23@njit.edu

Brian Doherty – Btd22@njit.edu

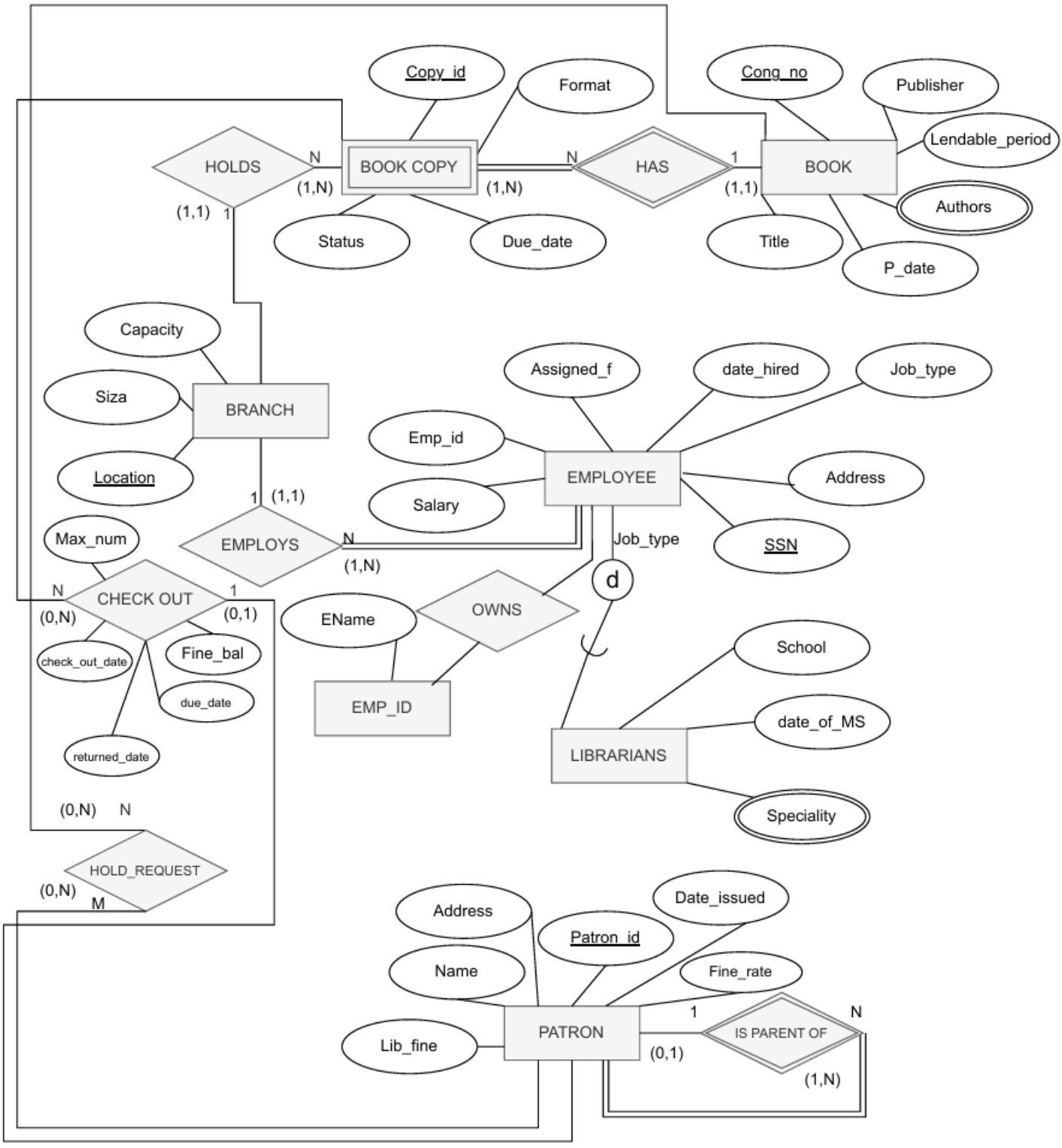
Newark Public Library Database

System Requirements:

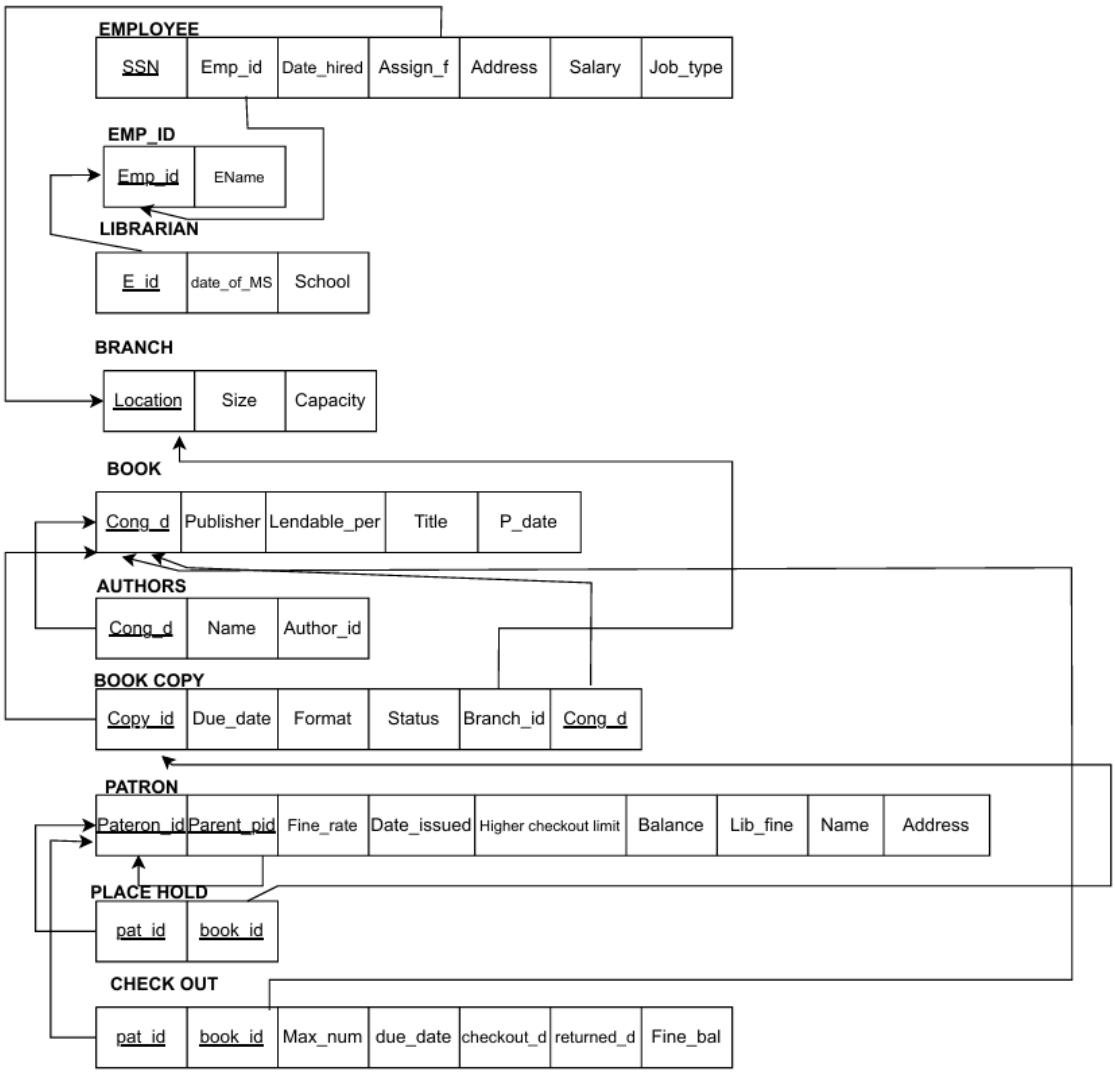
1. Besides the main library NPL maintains numerous branch facilities throughout the city of Newark. Management needs to track the branch location, size (in sq. ft.), capacity (linear feet of shelf space), and employees.
2. NPL employs librarians, desk clerks, maintenance, security, and administrative staff. For all its employees, NPL keeps a record of employee id, SSN, name, address, assigned facility, salary (monthly), and date hired. For librarians, NPL tracks the school and date of their MS in Library Science degrees. In addition, librarians may have one or more specialty fields (e.g. law, various branches of science, on-line research, etc.)
3. Each patron has a library card and NPL wants to record a patron id, name, address, date issued and library fine balance (if any). In addition, NPL identifies child patrons (under age 13). Child library cards identify the responsible parent (who must have a library card), have a lower overdue fine rate and have a higher number of books that can be checked out at any one time.
4. NPL wants to maintain a central record of all its books. Each book is identified by a Library of Congress number and NPL needs to track title, authors, publisher, publication date, and lendable period (how long the book can be check out for). NPL further needs to track every copy of each book: format (hardcover, paperback), which branch it is assigned to, its status (on shelf, checked out, needs maintenance, etc.) and its due date (if checked out). NOTE: only lendable books are considered in-scope for this database. It will not include other library assets such as audio books, periodicals, reference works, etc.
5. Patrons can check out and return any available book if they have not exceeded the maximum number allowable and if they do not have an overdue fine balance. Patrons can also place a hold request on any book that is not available for lending at a point in time.

The primary goal was to make sure the database functions perfectly given the system requirements above. We have achieved this and considered possibly expanding the database with additional features. Introducing new requirements, however, came with conflicts that affected the existing system design. That is why we focused on an implementation that was simple and directly aligned with the expectations of this mini world.

the entity-relationship design



the (relational) logical database design.



- After creating an EMPLOYEE entity I had to create a job_type attribute since we have to identify a specialty of the employee.

- Librarian had to be a separate entity.

- We had to include a foreign key in BOOK_COPY since it has the relationship HAS with BOOK.

- Patronp_id is a foreign key in a recursive relationship in PATRON, which is pointing at Patron_id.

- We had to draw some lines doubled because of the (1:N) type of constraints. Like Patronp_id and Patron_id in PATRON entity, and EMPLOYEE with BRANCH.

PATRON (NO NORMALIZING NEEDED)

Primary Key: PATRONID

Foreign Key: PARENTID → PATRON(PATRONID) (self-referencing

FD: PATRONID → NAME, ADDRESS, DATEISSUED, FINEBALANCE, PARENTID

- All attributes depend on the PK.
- No transitive dependencies.

AUTHOR (NO NORMALIZING NEEDED)

Primary Key: AUTHOR_ID

Foreign Key: CONG_D → BOOK(CONG_D)

FD: AUTHOR_ID → NAME, CONG_D

- All attributes depend on the PK.
- No transitive dependencies.

BOOK (NO NORMALIZING NEEDED)

Primary Key: CONG_D

FD: CONG_D → TITLE, AUTHOR, PUBLISHER, PUBDATE, LENDABLEPERIOD

- All attributes depend on the PK.
- No transitive dependencies.

BOOK_COPY (NO NORMALIZING NEEDED)

Primary Key: Composite (CONG_D, COPYID)

Foreign Keys:

CONG_D → BOOK(CONG_D)

BLOCATION → BRANCH(BLOCATION)

FD: (CONG_D, COPYID) → FORMAT, BLOCATION, STATUS, DUEDATE

- All attributes depend on the PK.
- No transitive dependencies.

CHECKOUT (NO NORMALIZING NEEDED)

Primary Key: Composite (COPYID, PATRONID, CHECKOUTDATE)

Foreign Keys:

(CONG_D, COPYID) → BOOK_COPY

PATRONID → PATRON(PATRONID)

FD: (COPYID, PATRONID, CHECKOUTDATE) → DUEDATE, RETURNDATE, CONG_D

- All attributes depend on the PK.
- No transitive dependencies.
- Composite PK.

EMPLOYEE

Primary Key: SSN

Foreign Keys:

EMP_ID → EMPLOYEE_ID(EMP_ID)

ASSIGNED_F → BRANCH(BLOCATION)

FD: SSN → DATE_HIRED, ADDRESS, EMP_ID, SALARY, ASSIGNED_F

- EMP_ID → ENAME is handled by decomposition to EMPLOYEE_ID.
- No transitive dependencies.

EMPLOYEE_ID (NO NORMALIZING NEEDED)

Primary Key: EMP_ID

FD: EMP_ID → ENAME

HOLD_REQUEST (NO NORMALIZING NEEDED)

Primary Key: REQUESTID

Foreign Key: PATRONID → PATRON(PATRONID)

FD: REQUESTID → COPYID, PATRONID, REQUESTDATE

- All attributes depend on the PK.
- No transitive dependencies.

LIBRARIAN

Primary Key: EMP_ID

Foreign Key: EMP_ID → EMPLOYEE

FD: EMP_ID → SCHOOL, DEGREE_DATE

- All attributes depend on the PK.
- No transitive dependencies.

LIBRARIAN_SPECIALITY

Primary Key: (EMP_ID, SPECIALTY)

Foreign Key: EMP_ID → LIBRARIAN(EMP_ID)

BRANCH

Primary Key: BLOCATION

- All attributes depend on the PK.
- No transitive dependencies.

The queries needed to normalize the database (Only used to split EMPLOYEE):

~ Creating the EMPLOYEE_ID table

```
CREATE TABLE EMPLOYEE_ID (
  Emp_id CHAR(10) PRIMARY KEY,
  EName VARCHAR(15)
);
```

~Transferring all the data of EName and Emp_id from EMPLOYEE to EMPLOYEE_ID

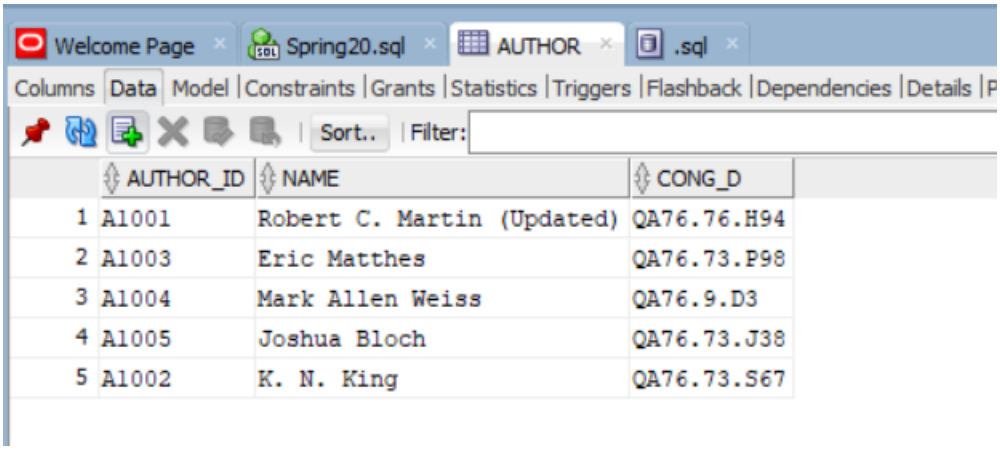
```
INSERT INTO EMPLOYEE_ID (Emp_id, EName)
SELECT DISTINCT Emp_id, EName
FROM EMPLOYEE
WHERE Emp_id IS NOT NULL;
```

~Dropping the column

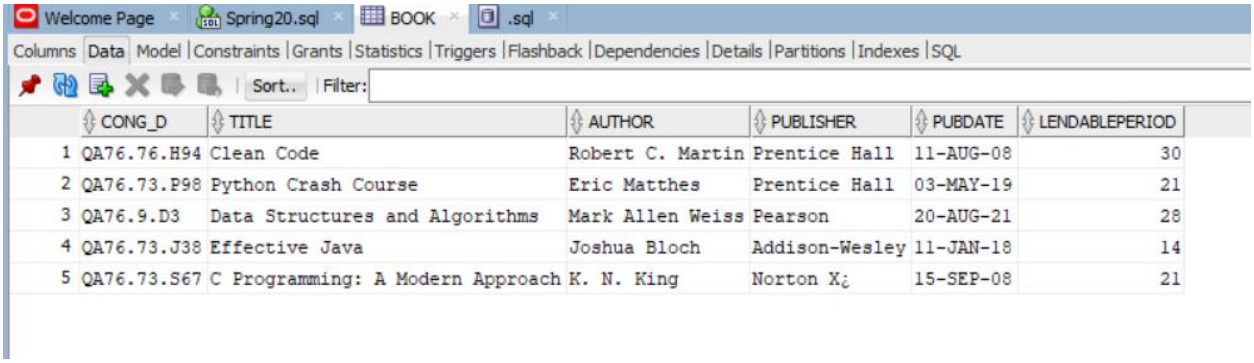
```
ALTER TABLE EMPLOYEE DROP COLUMN EName;
```

~Creating a foreign key

```
ALTER TABLE EMPLOYEE
ADD FOREIGN KEY (Emp_id) REFERENCES EMPLOYEE_ID(Emp_id);
```



	AUTHOR_ID	NAME	CONG_D
1	A1001	Robert C. Martin (Updated)	QA76.76.H94
2	A1003	Eric Matthes	QA76.73.P98
3	A1004	Mark Allen Weiss	QA76.9.D3
4	A1005	Joshua Bloch	QA76.73.J38
5	A1002	K. N. King	QA76.73.S67



	CONG_D	TITLE	AUTHOR	PUBLISHER	PUBDATE	LENDABLEPERIOD
1	QA76.76.H94	Clean Code	Robert C. Martin	Prentice Hall	11-AUG-08	30
2	QA76.73.P98	Python Crash Course	Eric Matthes	Prentice Hall	03-MAY-19	21
3	QA76.9.D3	Data Structures and Algorithms	Mark Allen Weiss	Pearson	20-AUG-21	28
4	QA76.73.J38	Effective Java	Joshua Bloch	Addison-Wesley	11-JAN-18	14
5	QA76.73.S67	C Programming: A Modern Approach	K. N. King	Norton X¿	15-SEP-08	21

Welcome Page | Spring20.sql | BOOK_COPY | .sql

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	COPYID	CONG_D	FORMAT	BLOCATION	STATUS	DUEDATE
1	2001	QA76.73.P98	Paperback	Main Branch	On Shelf	(null)
2	2002	QA76.9.D3	Hardcover	Main Branch	Checked Out	22-APR-25
3	2005	QA76.76.H94	Hardcover	Main Branch	Checked Out	25-APR-25
4	2003	QA76.73.J38	Hardcover	Science and Technology Branch	Checked Out	(null)
5	2004	QA76.73.S67	Paperback	Main Branch	Needs Maintenance	(null)

Welcome Page | Spring20 | Grants | BRANCH | .sql

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | D

Sort.. | Filter:

	BLOCATION	BSIZE	BCAPACITY
1	Main Branch	11000	50000
2	North Newark Branch	5000	20000
3	South Ward Branch	4000	15000
4	West Side Branch	4500	18000
5	Science and Technology Branch	6000	25000

Welcome Page | Spring20.sql | CHECKOUT | .sql

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SC

Sort.. | Filter:

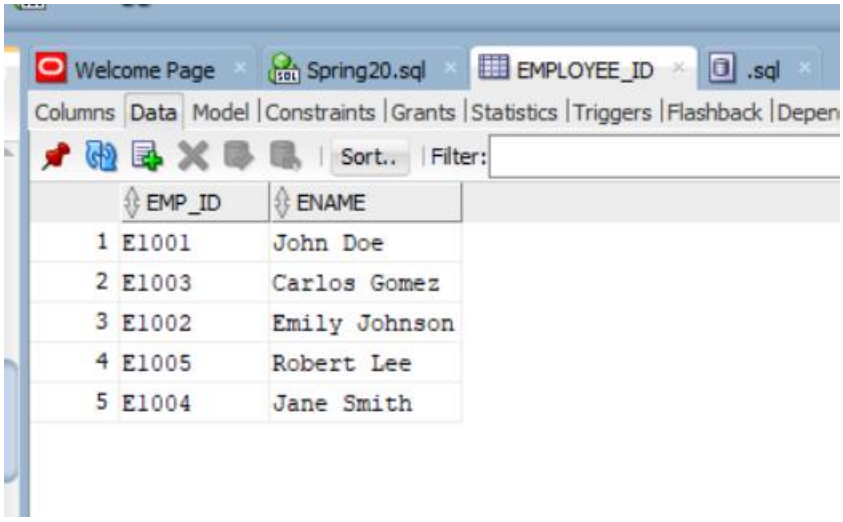
	COPYID	PATRONID	CHECKOUTDATE	DUEDATE	RETURNDATE	CONG_D
1	2004	4	18-FEB-25	04-MAR-25	02-MAR-25	(null)
2	2002	2	02-APR-25	16-APR-25	10-APR-25	(null)
3	2003	3	22-MAR-25	05-APR-25	30-MAR-25	(null)
4	2001	1	18-FEB-25	04-MAR-25	02-MAR-25	(null)
5	2005	5	29-MAR-25	12-APR-25	(null)	(null)

Welcome Page | Spring20.sql | EMPLOYEE | .sql

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	SSN	DATE_HIRED	ADDRESS	EMP_ID	SALARY	ASSIGNED_F
1	123456789	10-MAR-15	123 Main St, Newark, NJ	E1001	60000	Main Branch
2	234567890	15-JUN-18	456 North Ave, Newark, NJ	E1002	40000	North Newark Branch
3	345678901	05-JAN-20	789 South Rd, Newark, NJ	E1003	35000	South Ward Branch
4	456789012	15-JUN-18	456 North Ave, Newark, NJ	E1004	40000	North Newark Branch
5	567890123	10-APR-22	202 West St, Newark, NJ	E1005	32000	Main Branch

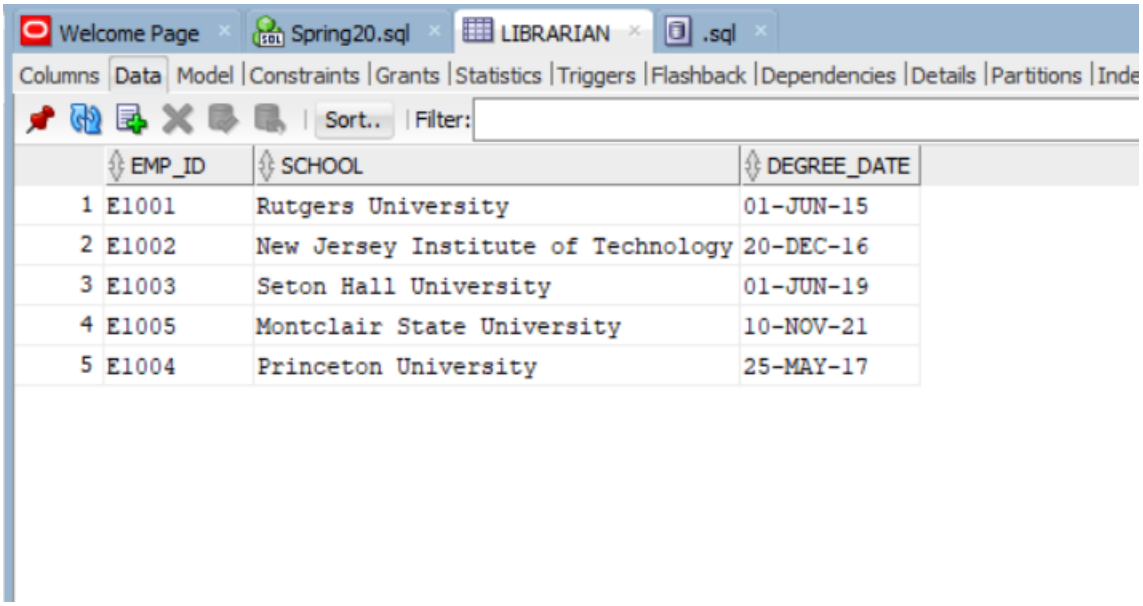


Welcome Page | Spring20.sql | HOLD_REQUEST | .sql

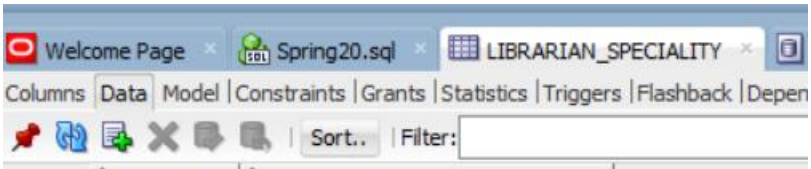
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Depende

Sort.. | Filter:

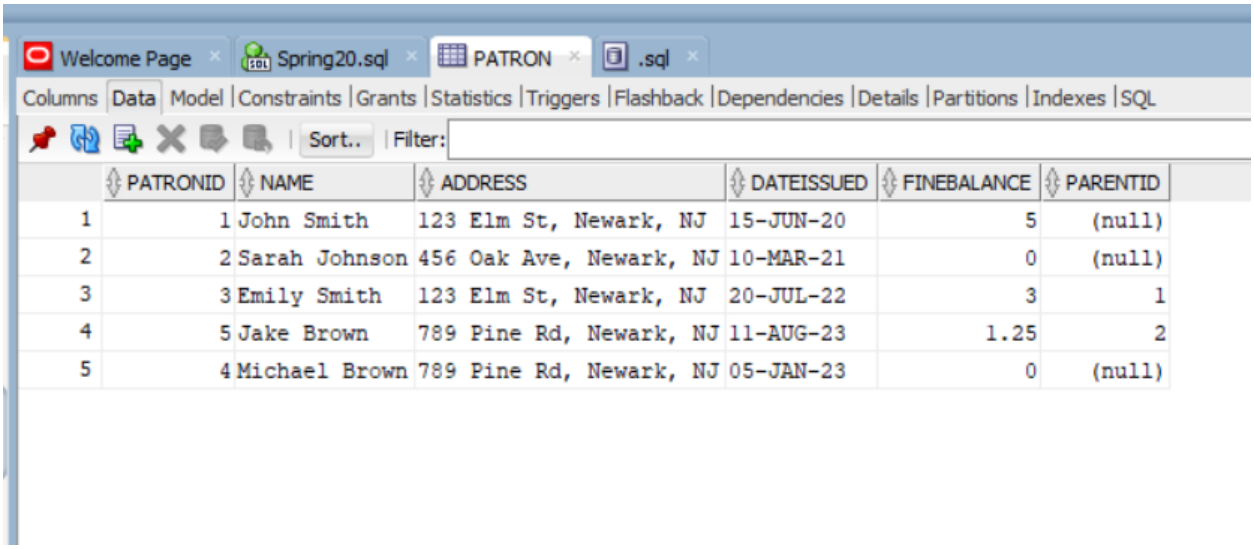
	REQUESTID	CO...	PATRONID	REQUESTDATE
1	101	2001	1	05-APR-25
2	103	2002	2	10-APR-25
3	102	2003	3	15-JAN-25
4	105	2004	4	20-MAR-25
5	104	2005	5	12-APR-25



	EMP_ID	SCHOOL	DEGREE_DATE
1	E1001	Rutgers University	01-JUN-15
2	E1002	New Jersey Institute of Technology	20-DEC-16
3	E1003	Seton Hall University	01-JUN-19
4	E1005	Montclair State University	10-NOV-21
5	E1004	Princeton University	25-MAY-17



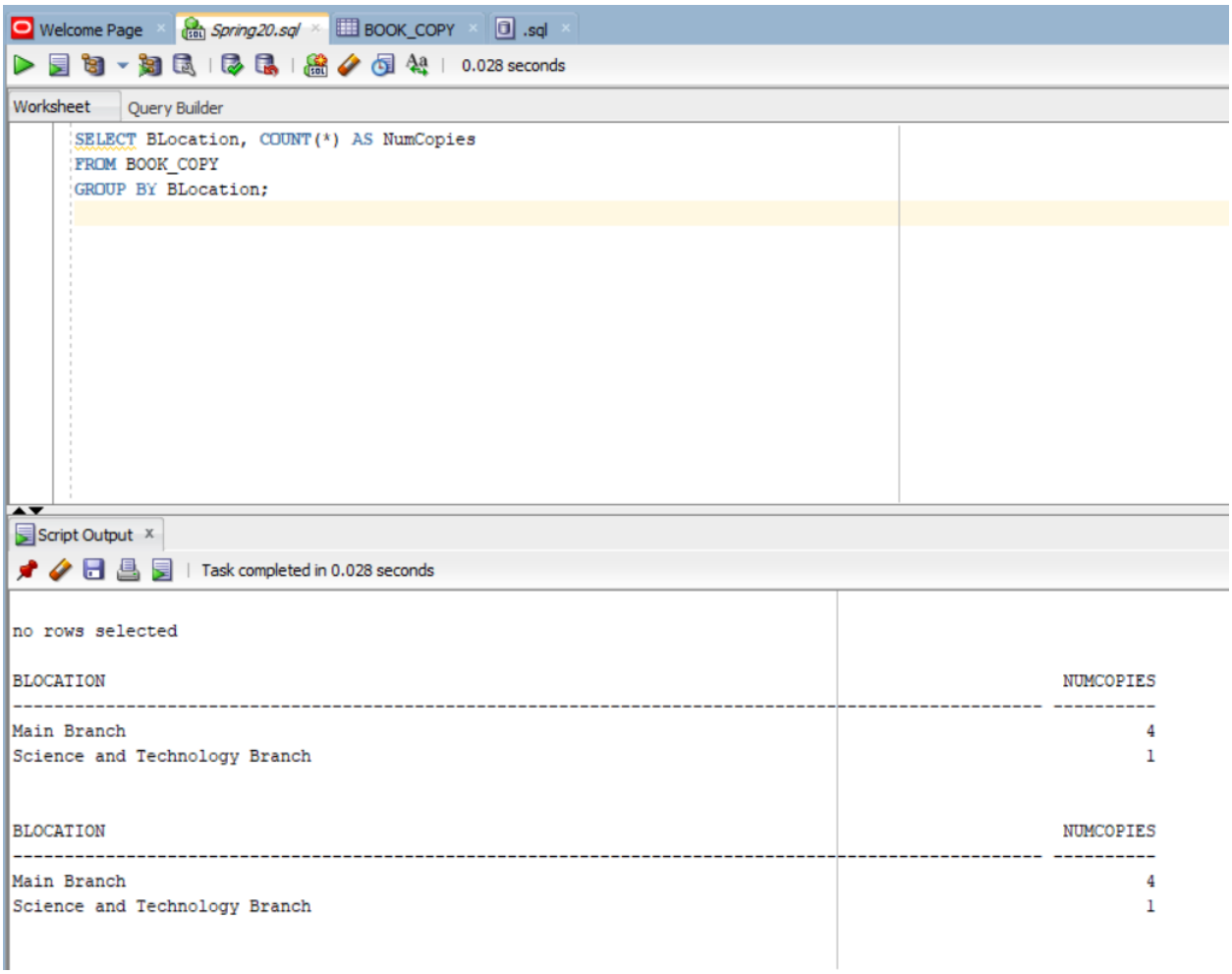
	EMP_ID	SPECIALTY
1	E1001	Children¿s Literature
2	E1002	Community Outreach
3	E1003	Advanced Programming
4	E1005	Library Technology Systems
5	E1004	STEM Educational Programs



	PATRONID	NAME	ADDRESS	DATEISSUED	FINEBALANCE	PARENTID
1	1	John Smith	123 Elm St, Newark, NJ	15-JUN-20	5	(null)
2	2	Sarah Johnson	456 Oak Ave, Newark, NJ	10-MAR-21	0	(null)
3	3	Emily Smith	123 Elm St, Newark, NJ	20-JUL-22	3	1
4	5	Jake Brown	789 Pine Rd, Newark, NJ	11-AUG-23	1.25	2
5	4	Michael Brown	789 Pine Rd, Newark, NJ	05-JAN-23	0	(null)

1st Query: Count how many book copies exist at each branch.

```
SELECT BLocation, COUNT(*) AS NumCopies
FROM BOOK_COPY
GROUP BY BLocation;
```



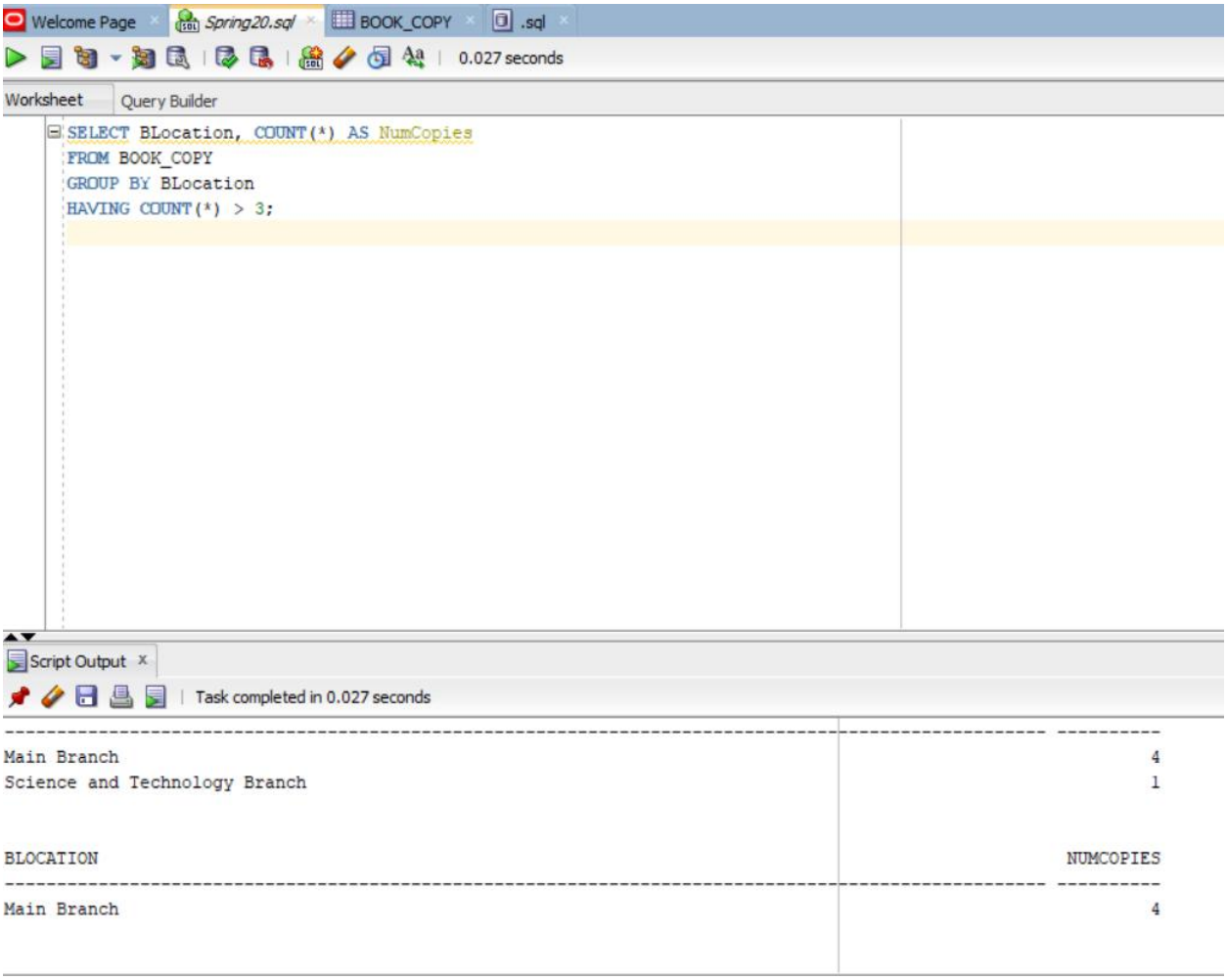
2nd Query: Show all branches that have more than 3 book copies.

SELECT BLocation, COUNT(*) AS NumCopies

FROM BOOK_COPY

GROUP BY BLocation

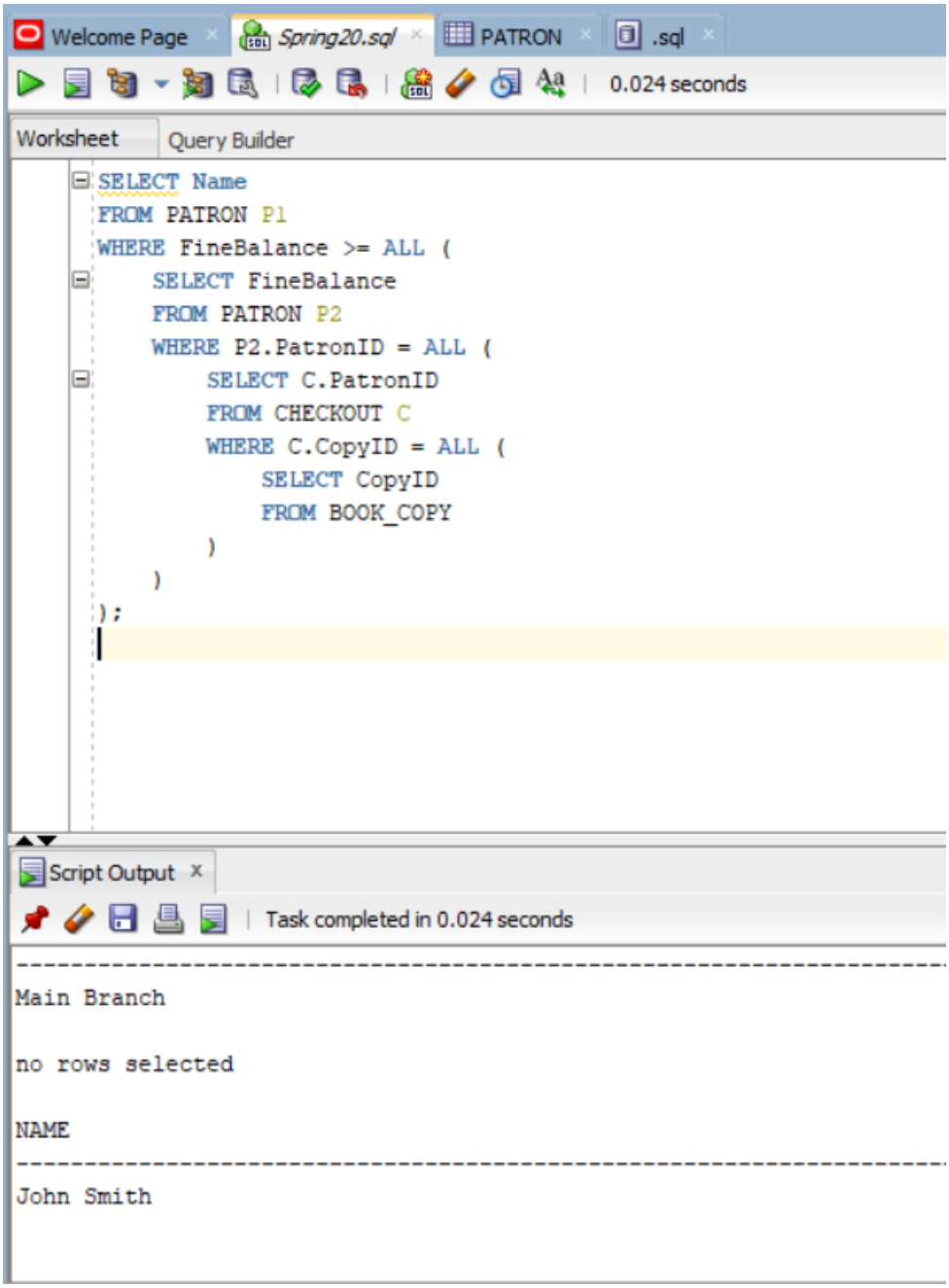
HAVING COUNT(*) > 3;



3rd Query: Find the names of patrons whose fine balance is greater than or equal to the fine balances of all patrons who have checked out every single book copy in the library.

```
SELECT Name
FROM PATRON P1
WHERE FineBalance >= ALL (
  SELECT FineBalance
  FROM PATRON P2
  WHERE P2.PatronID = ALL (
    SELECT C.PatronID
```

```
      FROM CHECKOUT C
      WHERE C.CopyID = ALL (
        SELECT CopyID
        FROM BOOK_COPY
      )
  )
);
```



4th Query: Find the names of patrons who have requested book copies located at branches that have more than 3 total copies.

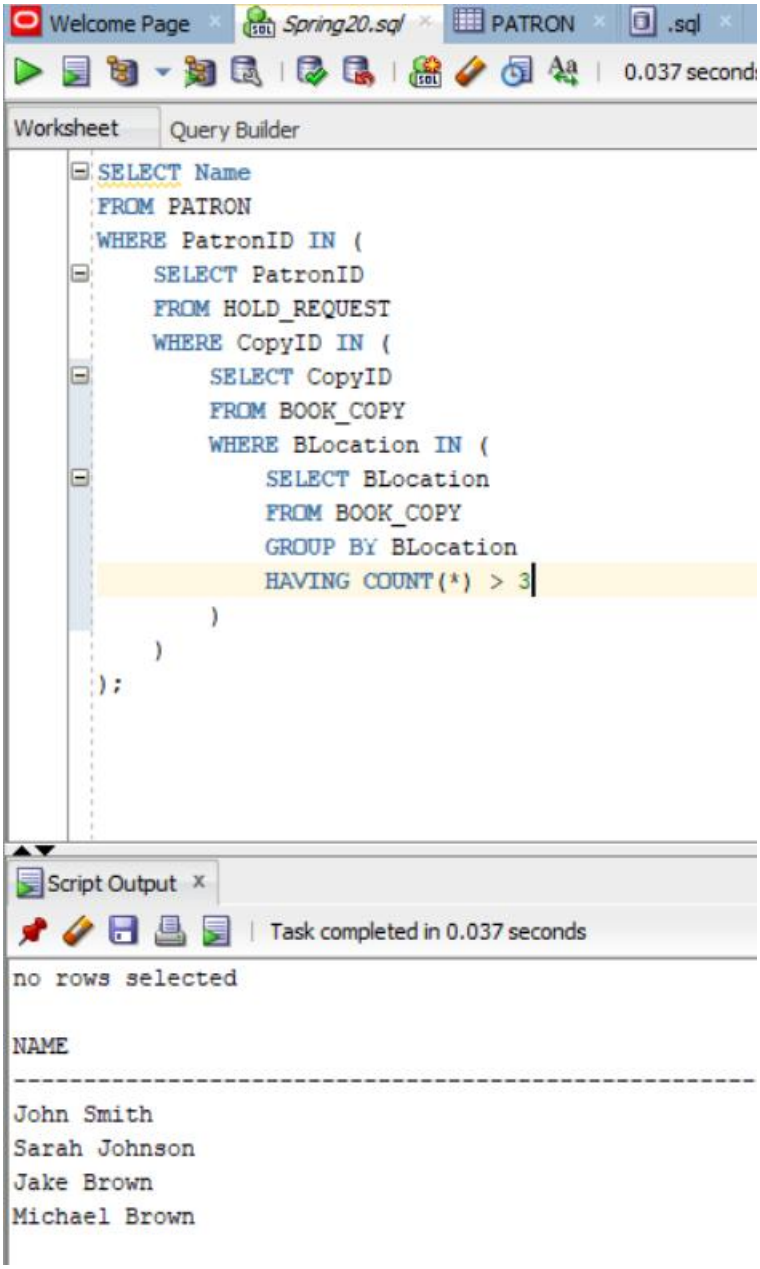
SELECT Name

FROM PATRON

WHERE PatronID IN (

SELECT PatronID

```
  FROM HOLD_REQUEST
  WHERE CopyID IN (
    SELECT CopyID
    FROM BOOK_COPY
    WHERE BLocation IN (
      SELECT BLocation
      FROM BOOK_COPY
      GROUP BY BLocation
      HAVING COUNT(*) > 3
    )
  )
);
```



Narrative Conclusion:

Our experience while working on this project involved lots of unexpected outcomes. In particular, the most difficult parts of our procedure can be described as backtracking every advancement that we made in updating our database to make sure our original models/diagrams matched. An ideal example of this procedure is the errors we ran into when updating the primary keys before creating foreign keys for other entities. Some of the

positive learning experiences however included being able to more responsibly manage, update, and delete data with less instances of it being mistakenly altered, which is good etiquette for scenarios of handling important data. If we were to go back, we would have told our past selves this methodology of handling data, as it actually ended up being more efficient than what we were originally doing.

Even though we found it very fun and unpredictable, we came out with a fear of databases in their broad complexity.