Baruch Records LLC

Music Database Group Report

CIS 9340 - Database Management Systems - PMWA
12/18/2019

Edwin Echevarria
Jinyoung Chung
Seungrok Lee
Brian Skutch

Introduction

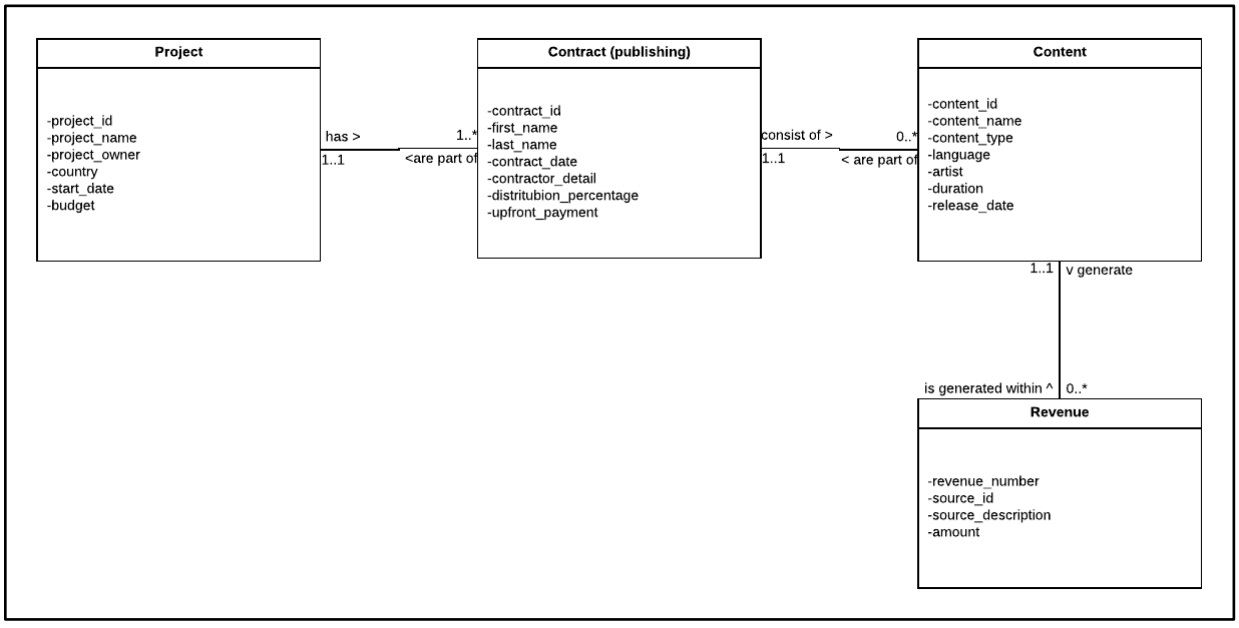
Founded in 2019, Baruch Records is a music label catering to independent artists globally. In the upcoming year we plan to release thirty-six singles (projects). In order to keep track of project information and cost recoupment, we have decided to invest in a music database management system. Below is a report outlining our steps we took to transform this business plan from conceptual to physical.

Entity Relationship Diagram

We first gathered the user and system requirements during the system analysis period. Once the collection stage of the database system development lifecycle was complete, we began the database design stage. The Entity Relationship (ER) is a model for communication that ensures that we have a precise understanding of the nature of the data and how it is to be used by the enterprise.

Entity Relationship Diagram - Based on the user requirements, ER modeling is a top-down approach to database design that identifies the Entities, Relationships, Attributes, and Identifiers.

ER Diagram



Relational Model

A great strength of the Relational Model is its simple logical structure. With the ER Diagram completed we had an accurate model of the user's view. The next step was to convert this conceptual representation into a model which can be implemented directly into a database. We converted the key parts (Entities, Relationships, Attributes, and Identifiers) into the Relational Model.

- Entities converted to Relations,
- Relationships became Foreign Keys
- Identifiers became the Keys of the Relations

Relational Model - The key parts are the relations, tuples, attributes, keys, and foreign keys. All data is logically structured within these relations.

Relational Model

PROJECT (project_id, project_name, project_owner, country, start_date, budget)

CONTRACT (contract_id, first_name, last_name, contract_date, contractor_detail, distribution_percentage, upfront_payment, project_id(fk))

CONTENT (content_id, content_name, content_type, language, artist, duration, release_date, contract_id(fk))

REVENUE (revenue_number, source_id, source_description, amount, content_id(fk))

Functional Dependencies

Normalization is a technique for producing a set of relations with desirable properties according to the data requirements. With the Relational Model done we began the process of normalization by identifying the functional dependencies. This provided us with a set of integrity constraints, the most important being candidate keys and primary keys.

Functional Dependencies - Outlines the relationships between the attributes within a single relation.

Functional Dependencies

PROJECT

FD1: project_id -> project_name, project_owner, country, start_date, budget

CONTRACT

FD1: contract_id -> first_name, last_name, contract_date, contractor_detail, distribution_percentage, upfront_payment

CONTENT

FD1: content_id -> content_name, content_type, language, artist, duration, release_date

REVENUE

FD1: revenue_number -> source_id, source_description, amount

FD2: source_id -> source_description

Normalization

With both Keys and Functional Dependencies outlined we ran a series of tests (described as normal forms) to help identify the optimal grouping for these attributes. When a relation failed to meet a normal form definition we changed it by splitting the tables. Once we reached the third normal form we finally had a set of suitable relations to support the data requirements of the enterprise.

Normalization - The process of splitting up relations in order to assure they are free from a certain set of modification anomalies.

First Normal Form	Second Normal Form	Third Normal Form
● Atomic values only ● No repeating groups	● In 1st normal form ● No partial key dependencies	● In 2nd normal form ● No transitive dependencies

Normalization

PROJECT (project_id, project_name, project_owner, country, start_date, budget)

CONTRACT (contract_id, first_name, last_name, contract_date, contractor_detail, distribution_percentage, upfront_payment, project_id(fk))

CONTENT (content_id, content_name, content_type_number, language, artist, duration, release_date, contract_id(fk))

CONTENT_TYPE (content_type_number, content_type)

REVENUE (revenue_number, source_id, amount, content_id(fk))

REVENUE_SOURCE (source_id, source_description)

SQL Data Definition Language

With Normalization completed we then were ready to define the schema of the database objects using a data definition language (SQL). In addition to the building of the schema, SQL allows us to perform data management tasks and perform both simple and complex queries.

```
CREATE TABLE project (
        project_id          VARCHAR(30) NOT NULL,
        project_name  VARCHAR(30) NOT NULL,
        project_owner VARCHAR(55) NOT NULL,
        start_date          DATE,
        country             VARCHAR(30) NOT NULL,
        Budget              NUMBER,
        CONSTRAINT pk_project_id PRIMARY KEY (project_id)
);
 CREATE TABLE contract (
        contract_id         VARCHAR(30) NOT NULL,
        project_id          VARCHAR(30) NOT NULL,
        first_name          VARCHAR(100) NOT NULL,
        last_name           VARCHAR(100) NOT NULL,
        contract_date  DATE,
        contractor_detail VARCHAR(30) NOT NULL,
        Distribution_percentatge NUMBER,
        upfront_payment NUMBER,
        CONSTRAINT pk_contract_id PRIMARY KEY (contract_id),
        FOREIGN KEY (project_id) REFERENCES project (project_id)
);
```

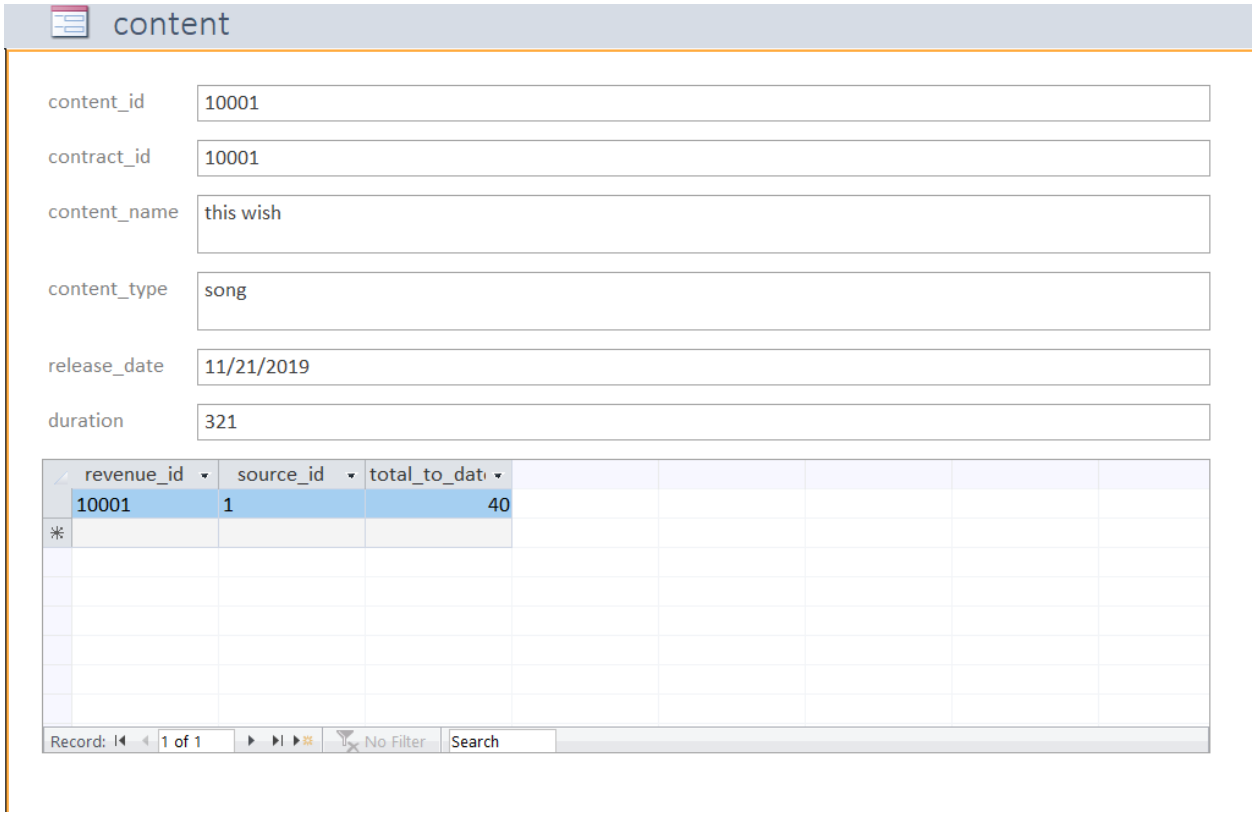
```
CREATE TABLE content (
        content_id        VARCHAR(30) NOT NULL,
        contract_id       VARCHAR(30) NOT NULL,
        content_name VARCHAR(100) NOT NULL,
        content_type   VARCHAR(100) NOT NULL,
        release_date   DATE,
        duration NUMBER,
        CONSTRAINT pk_content_id PRIMARY KEY (content_id),
        FOREIGN KEY (contract_id) REFERENCES contract (contract_id)
);

CREATE TABLE revenue (
        revenue_id        VARCHAR(30) NOT NULL,
        countent_id       VARCHAR(30) NOT NULL,
        source_id          VARCHAR(30) NOT NULL,
        total_to_date   NUMBER,
        CONSTRAINT pk_revenue_id PRIMARY KEY (revenue_id),
        FOREIGN KEY (content_id) REFERENCES content (content_id),
        FOREIGN KEY (source_id) REFERENCES source (source_id);
);
CREATE TABLE source (
        source_id         VARCHAR(30) NOT NULL,
        source_type      VARCHAR(30) NOT NULL,
        source_description         VARCHAR(100) NOT NULL,
        CONSTRAINT pk_source_id PRIMARY KEY (source_id);
);
```

Database Application Components

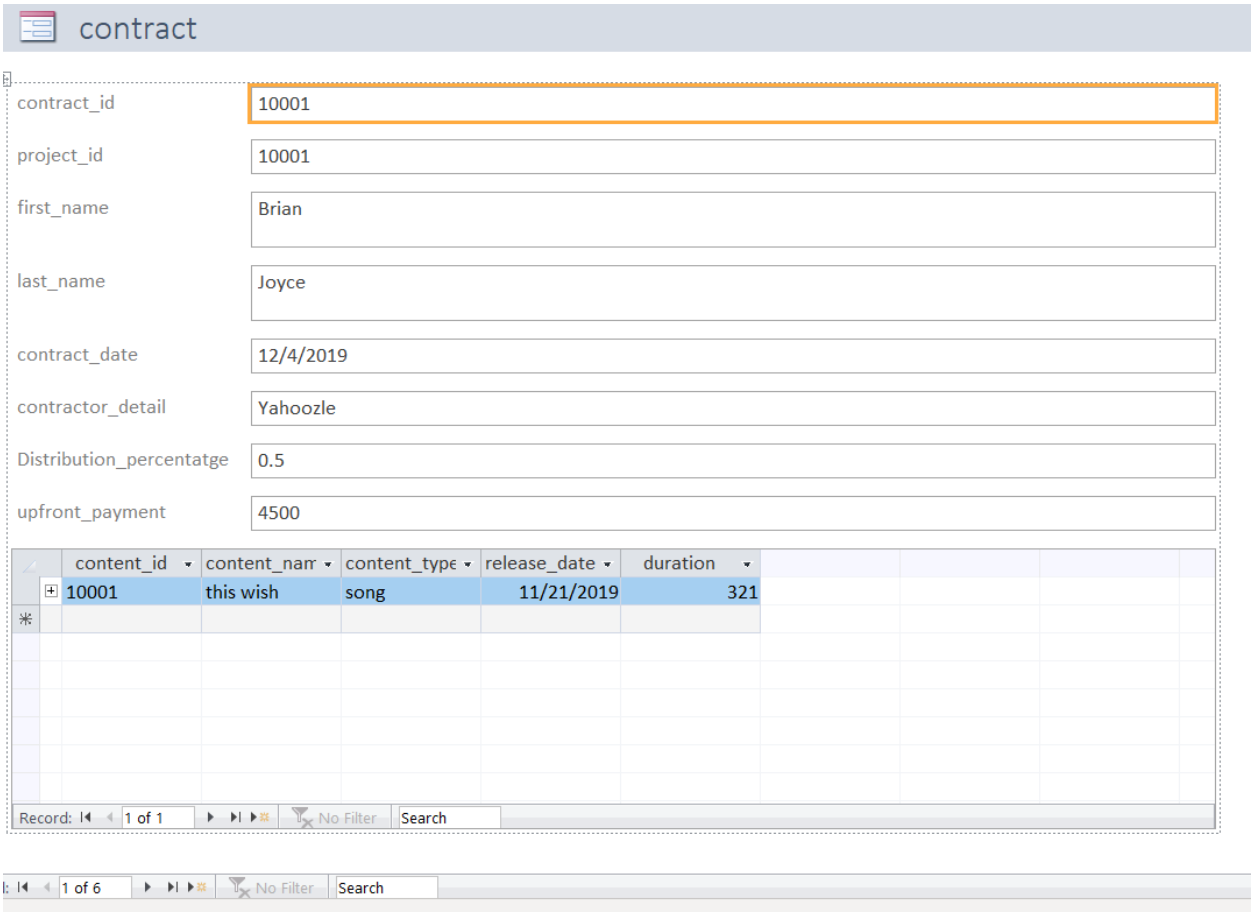
With the schema defined, we turned our attention to the applications which will be used to access the database.

- **Data Entry Forms** - The primary means to edit and enter data into the database
- **Report** - Used to convey large portions of data
- **Queries** - Common queries to the database
- **Navigation Form** - Provides a way for users to access forms, reports, and queries

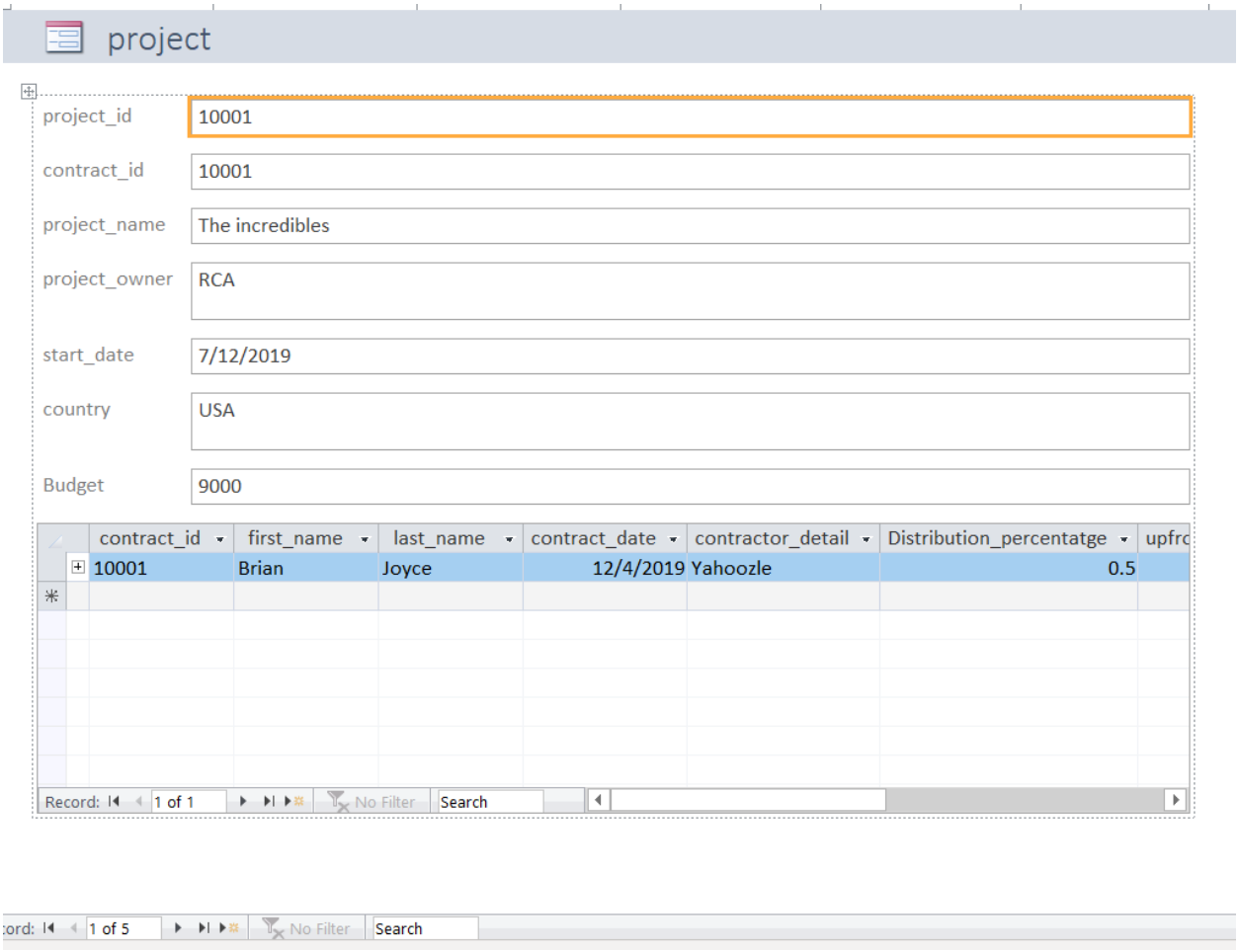
Content Form



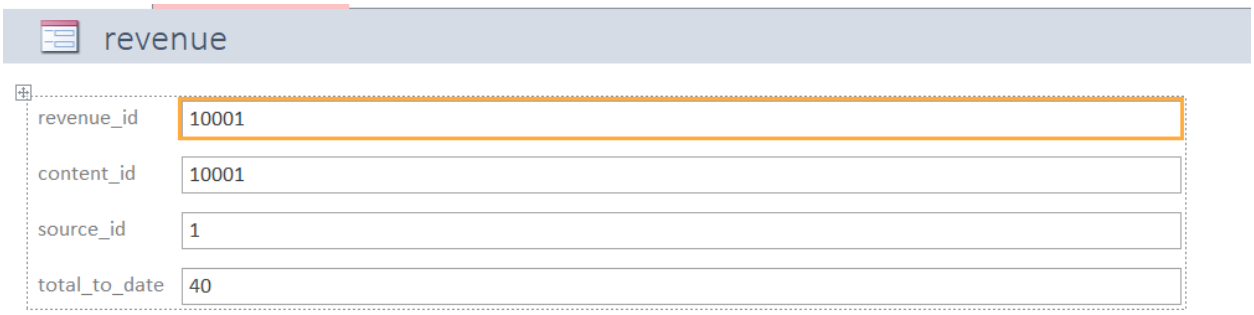
Contract Form



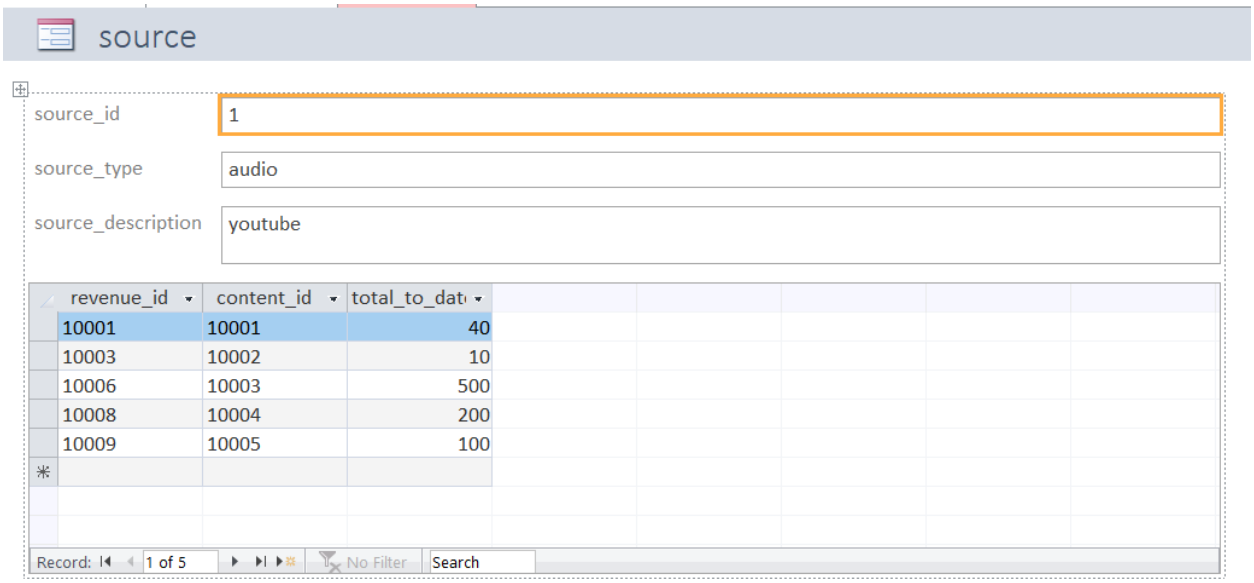
Project Form



Revenue Form



Source Form



Report Example

Revenue Totals

Wednesday, December 18, 2019
6:14:12 PM

content_id	content_name	revenue_id	source_id	total_to_date	Distribut	upfront_payment
10001	this wish	10001	1	40	0.5	4500
10002	next summer	10002	2	500	0.5	5000
10002	next summer	10003	1	10	0.5	5000
10003	true feelings	10004	3	100	0.5	7500
10003	true feelings	10005	2	1000	0.5	7500
10003	true feelings	10006	1	500	0.5	7500
10003	true feelings	10007	4	20	0.5	7500
10004	can you believe	10008	1	200	0.5	10000
10005	hey now	10009	1	100	0.5	7500
10006	this is it	10010	2	200	0.5	10000
10006	this is it	10011	3	3000	0.5	10000

11

```
SELECT project.project_id, project.project_name, contract.contract_id,
contract.Distribution_percentatge, contract.upfront_payment, content.content_id,
content.content_name, revenue.total_to_date, revenue.revenue_id

FROM ((project INNER JOIN contract ON project.[project_id] = contract.[project_id])
INNER JOIN content ON contract.[contract_id] = content.[contract_id]) INNER JOIN
revenue ON content.[content_id] = revenue.[content_id];
```

Current Contracts

Wednesday, December 18, 2019
6:33:07 PM

contract_id	last_name	content_id	content_name	project_id	project_name	project_owner	revenue_id	source_id
10001	Joyce	10001	this wish	10001	The incredibles	RCA	10001	1
10002	Smith	10002	next summer	10002	Punkhead Trio	MGA	10002	2
10002	Smith	10002	next summer	10002	Punkhead Trio	MGA	10003	1
10003	Twines	10003	true feelings	10003	Silverback Swimmers	Sony	10004	3
10003	Twines	10003	true feelings	10003	Silverback Swimmers	Sony	10005	2
10003	Twines	10003	true feelings	10003	Silverback Swimmers	Sony	10006	1
10003	Twines	10003	true feelings	10003	Silverback Swimmers	Sony	10007	4
10006	Twines	10006	this is it	10003	Silverback Swimmers	Sony	10010	2
10006	Twines	10006	this is it	10003	Silverback Swimmers	Sony	10011	3
10004	Beans	10004	can you believe	10004	Janky Jangles	RCA	10008	1
10005	Small	10005	hey now	10005	Paperback Writers	Sona	10009	1

11

SELECT contract.contract_id, contract.last_name, content.content_id, content.content_name, project.project_id, project.project_name, project.project_owner, revenue.revenue_id, revenue.source_id

FROM ((project INNER JOIN contract ON project.[project_id] = contract.[project_id]) INNER JOIN content ON contract.[contract_id] = content.[contract_id]) INNER JOIN revenue ON content.[content_id] = revenue.[content_id];

Common Query Example

Check current list of projects

```
SELECT project.*

FROM project;
```

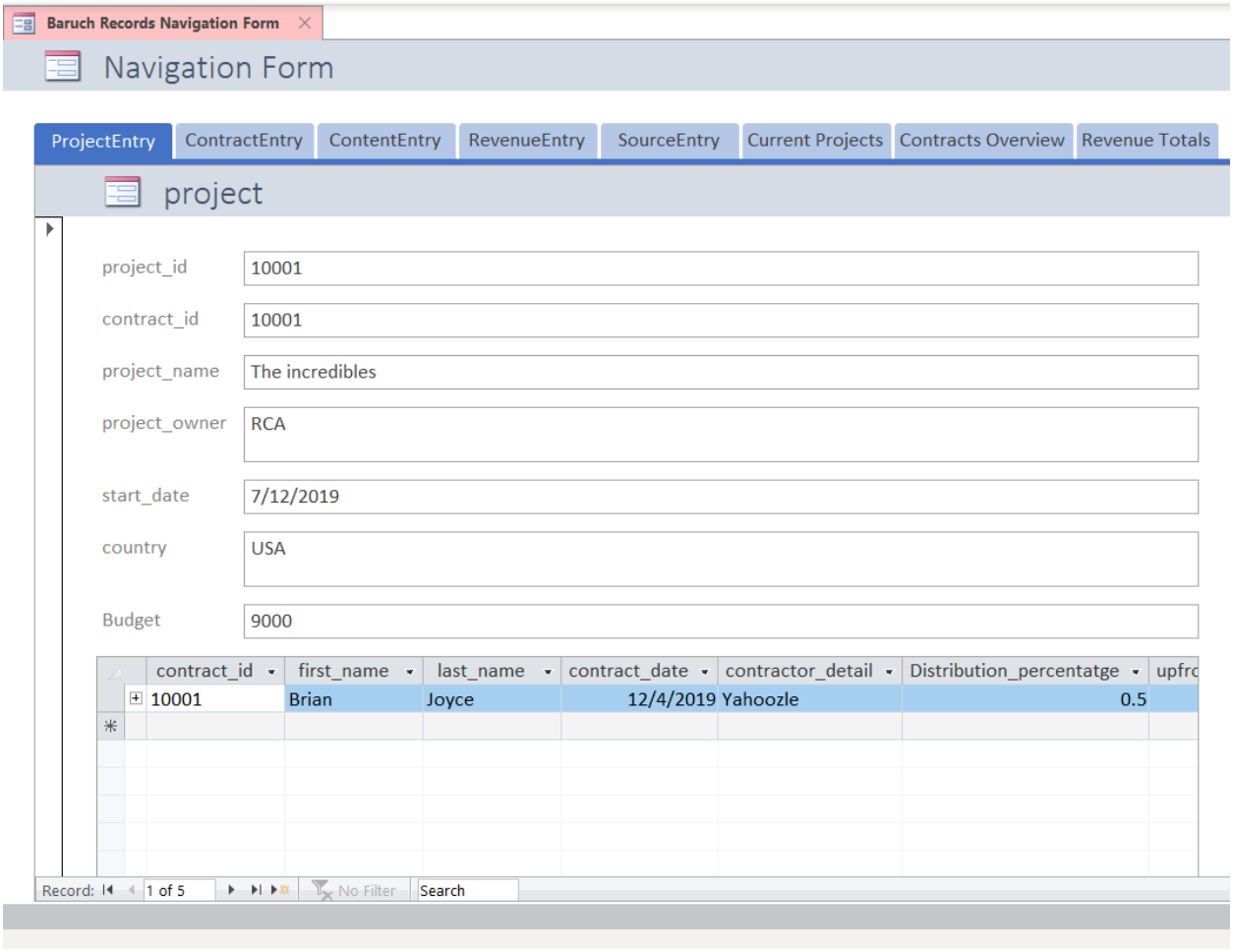
Check current list of contracts

```
SELECT contract.*

FROM contract;
```

Check Contract Id with associated Content Name, Upfront Payment, Distribution Percentage and Revenue Total to Date

```
SELECT contract.contract_id, content.content_name, contract.upfront_payment,
contract.Distribution_percentatge, revenue.total_to_date

FROM (contract INNER JOIN content ON contract.contract_id = content.contract_id)
INNER JOIN revenue ON content.content_id = revenue.content_id;
```

Navigation Form



Conclusion

The most challenging aspects for our team was formulating the Entity-Relationship Diagram and Relational Model. While the process seems simple on paper, with so many points of view, it was difficult to reach a consensus. As business leaders we now understand why, “Cost of Overhead” is so important to consider before choosing to build a database. This process requires a team of experienced professionals and sufficient time. Even for the smallest of companies, the cost of these types of resources would add up very quickly with no end in sight.