SQL Worksheet: Receipts

1. Introduction

This is a database about receipts. A receipt represents a sale, and belongs to a store. A receipt has many receipt_item's, where each item is for a product, and has a quantity. Each product belongs to exactly one category. The relational schema diagram is as in figure 1 (again, this is NOT necessarily how it would be done in real life).

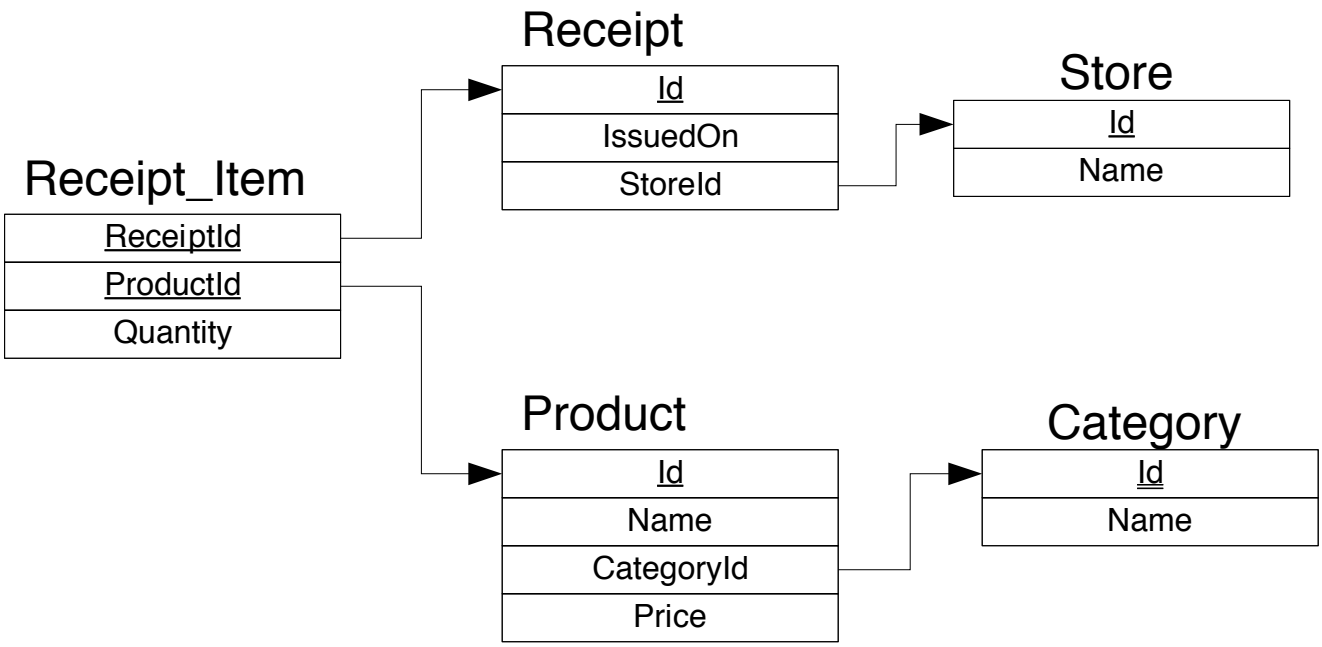

Figure 1: Relational Schema diagram for Receipts database

Keep in mind that in the diagram the arrows represent foreign keys; the arrows are directional (they point from FK to PK) but the *associations* are bidirectional; we can traverse those arrows in either direction, since we just do a join.

The SQL for the tables is as follows (available in the Examples tab, as receipts_worksheet_db.sql):

```
CREATE TABLE Category (
  Id       INTEGER PRIMARY KEY,
  Name VARCHAR(20) UNIQUE NOT NULL
);

CREATE TABLE Product (
  Id        INTEGER PRIMARY KEY,
  Name   VARCHAR(20) NOT NULL,
  CategoryId    INTEGER NOT NULL
       REFERENCES Category(Id),
  Price  Numeric(7,2) CHECK (Price>0)
);

CREATE TABLE Store (
  Id       INTEGER PRIMARY KEY,
  Name VARCHAR(20) UNIQUE NOT NULL
);
```

```
CREATE TABLE Receipt (
  Id      INTEGER PRIMARY KEY,
  IssuedOn      DATE,
  StoreId INTEGER REFERENCES Store(Id)
);

CREATE TABLE Receipt_Item (
  ReceiptId      INTEGER REFERENCES Receipt(id),
  ProductId      INTEGER REFERENCES Product(id),
  Quantity       INTEGER NOT NULL DEFAULT 1,
  CONSTRAINT RI_PK PRIMARY KEY
                    (ReceiptId,ProductId)
);
```

2. Examples and Exercises

2.1. Display the Id and Name of all the products that belong to the category with Id 1

This is a very simple SELECT statement, with a simple condition.

```
SELECT Id, Name
FROM Product
WHERE CategoryId=1
```

Example 1: Products belonging to category 1

2.2. Complete the query to Display the Id and Name of all the products that belong to the category with Id 2

```
SELECT Id, Name
FROM Product
WHERE
```

Completion 1: Products belonging to category 2

2.3. Display the total number of different products and their average price

Here we use aggregates, but no groupings. We can use COUNT to get us the number of products, and AVG to get us their price.

```
SELECT COUNT(*), AVG(Price)
FROM Product
```

Example 2: Number of products and their average price

2.4. Display the total number of different products belonging to category 1, and their average price

Here, we still not need to do groupings, we can just use a WHERE clause to limit the products to those belonging to category 1.

```
SELECT COUNT(*), AVG(Price)
FROM Product
WHERE CategoryId=1
```

Example 3: Number of products and their average price

2.5. Display the Id of all categories which have products, with the number of products belonging to that category.

Here, although we mention categories in the statement, we can get all the information from the product table, since that's where we need to count the products, and the table has the category id. Here we can use the COUNT aggregate, to get the number of products. Since we want to get the counts of products within the same category, we need to include the category id in the GROUP BY clause.

```
SELECT CategoryId, COUNT(*)
FROM Product
GROUP BY CategoryId
```

Example 4: Number of products per category

If you run this query with our sample data, you will notice that not all categories appear; category 3, has no products.

2.6. Complete the query to Display the Id of each store, with the number of receipts for that store.

This is a similar query to the one above, except that now we look at the receipt table:

```
SELECT
FROM Receipt
GROUP BY
```

Completion 2: Id of each store, with the number of receipts for that store.

2.7. Display the id and name of each product, with the *name* of the category it belongs to.

Here we need to join the Product table with the category table, since we need information from both tables. The clearest way is to use a JOIN operator within the FROM clause. To save typing, we alias Product as P and Category as C

```
SELECT P.Id, P.Name, C.Name
FROM Product P JOIN Category C ON (P.CategoryId=C.Id)
```

Example 5: Products with the name of their category

2.8. Complete the query to Display the Id and Date of each receipt, with the *name* of the store the receipt belongs to

This is very similar to 2.7, with different tables. Complete the Join conditon.

```
SELECT R.Id, R.Date, S.Name
FROM Receipt R JOIN Store S ON (                                    )
```

Completion 3: Id and date of each receipt, with the name of the store it belongs to

2.9. Display the Id and name of each category, with the number of products that belong to the category

Here we need to join both tables, and then do the count and a grouping;

```
SELECT C.Id, C.Name, COUNT(*)
FROM Product P JOIN Category C ON (P.CategoryId=C.Id)
GROUP BY C.Id, C.Name
```

Example 6: Categories with their number of products

Here we have one problem; if a category has no products, it will not appear in the result set, since the join condition will not be true for the category and any row in products; it is not absolutely clear from the query whether categories with no products should appear, although it can be considered an implicit requirement (... name of *each* category ...). If we need to include categories with no products, we need to use an OUTER JOIN. The next query makes the requirement explicit:

2.10. Display the Id and name of each category, with the number of products that belong to the category. Make sure ALL categories appear, even those with no products.

Since we want ALL categories to appear, we need to use an outer join. When we do an outer join, all the rows in the specified table will appear in the result; if the row doesn't actually match any row on the other table, the fields from the other table in the result set for that extra row are assigned a value of NULL.

So, we modify the query in 2.9 to make it an outer join; we also need to modify the argument to the COUNT aggregate. If we put *, then SQL will count all *rows*, so we would get a value of 1 for categories with no products; however, if we put a field or expression, SQL only counts those that are *not null*, so we just need to put some field from the *other* table. Notice that since we want all *categories* to appear, and the category table appears to the *right* of the JOIN operator, we use a RIGHT OUTER JOIN.

```
SELECT C.Id, C.Name, COUNT(P.Id)
FROM Product P RIGHT OUTER JOIN Category C ON (P.CategoryId=C.Id)
GROUP BY C.Id, C.Name
```

Example 7: Categories with their number of products

2.11. Display the id of any store which has sold the product with id 1

Here, the Receipt_Item table has the product id, and from there we can get to the Receipt table, which has the Store id; We could try:

```
SELECT R.StoreId
FROM Receipt R JOIN Receipt_Item RI ON (R.Id=RI.ReceiptId)
WHERE RI.ProductId=1
```

Example 8: Stores who have sold product 1 (not quite right)

The problem with this query is that it would produce some stores several times, since the same product may have been sold many times in a store; we can use the DISTINCT modifier to make sure that doesn't happen:

```
SELECT DISTINCT R.StoreId
FROM Receipt R JOIN Receipt_Item RI ON (R.Id=RI.ReceiptId)
WHERE RI.ProductId=1
```

Example 9: Stores who have sold product 1 (no duplicates)

2.12. Display the id of any store which has sold the product named 'Toy Car'

This is similar to 2.11, but we need to also join with the product table, so we can check for the name of the product rather than its id. When joining several tables, the full JOIN clause is syntactically equivalent to a single table name.

```
SELECT DISTINCT R.StoreId
FROM Receipt R JOIN Receipt_Item RI ON (R.Id=RI.ReceiptId)
     JOIN Product P ON (RI.ProductId=P.Id)
WHERE P.Name='Toy Car'
```

Example 10: Stores who have sold product 1 (no duplicates)

3. Further Exercises

3.1. Display the name of all stores which have sold any products belonging to the category with id 1

3.2. Display the name of all stores which have sold any products belonging to the category named 'Toys'

3.3. Display the id and date of each receipt, with the total number products that were sold in that receipt (you need to add the quantities for all the receipt items corresponding to that receipt)

3.4. Display the id of each receipt, with the total amount of each receipt. The amount is calculated by multiplying the quantity mentioned on each Item and multiplying by the corresponding product's price, and adding this number for all items corresponding to that receipt.