/\* SUBQUERIES Learn how subqueries can be used to filter data, generate values, and construct temporary data sets. A subquery is a query contained within another SQL statement When the containing statement has finished executing, the data returned by any subqueries is discarded, making a subquery act like a temporary table with statement scope (meaning that the server frees up any memory allocated to the subquery results after the SQL statement has finished execution).\*/

SELECT customer\_id, first\_name, last\_name

FROM customer

WHERE customer\_id = (SELECT max(customer\_id) FROM customer);

SELECT max(customer\_id)

FROM customer;

SELECT customer\_id, first\_name, last\_name

FROM customer

WHERE customer\_id = 599;

/\*Subquery Types some subqueries are completely self-contained (called noncorrelated subqueries), while others reference columns from the containing statement (called correlated subqueries).\*/

/\*Non Related Subqueries The example from earlier is a noncorrelated subquery; it may be executed alone and does not reference anything from the containing statement. Most subqueries that you encounter will be of this type unless you are writing update or delete statements, which frequently make use of correlated subqueries (more on this later). Along with being noncorrelated, the example from earlier in the chapter also returns a result set containing a single row and column. This type of subquery is known as a scalar subquery and can appear on either side of a condition using the usual operators (=, <>, <, >, <=, >=).\*/

SELECT city\_id, city

FROM city

WHERE country\_id <> (SELECT country\_id FROM country WHERE country = 'India');

SELECT country\_id

FROM country

WHERE country = 'India';

SELECT country\_id, city\_id, city

FROM city

WHERE country\_id = (SELECT country\_id FROM country WHERE country = 'India');

SELECT city\_id, city

FROM city

WHERE country\_id <> (SELECT country\_id FROM country WHERE country <> 'India');

/\*The containing query fails because an expression (country\_id) cannot be equated to a set of expressions (country\_ids 1, 2, 3, ..., 109). In other words, a single thing cannot be equated to a set of things.\*/

/\*The in and not in operators While you can’t equate a single value to a set of values, you can check to see whether a single value can be found within a set of values.\*/

SELECT country\_id

FROM country

WHERE country IN ('Canada', 'Mexico');

/\*The following query uses the in operator with a subquery on the righthand side of the filter condition to return all cities that are in Canada or Mexico:\*/

desc city;

SELECT city\_id, city

FROM city

WHERE country\_id IN (SELECT country\_id FROM country WHERE country IN ('Canada', 'Mexico'));

SELECT city\_id, city

FROM city

WHERE country\_id NOT IN (SELECT country\_id FROM country WHERE country IN ('Canada', 'Mexico'));

/\*The all operator While the in operator is used to see whether an expression can be found within a set of expressions, the all operator allows you to make comparisons between a single value and every value in a set.\*/

SELECT first\_name, last\_name

FROM customer

WHERE customer\_id <> ALL (SELECT customer\_id FROM payment WHERE amount = 0);

/\*Without using the all operator\*/

SELECT first\_name, last\_name

FROM customer

WHERE customer\_id NOT IN (SELECT customer\_id FROM payment WHERE amount = 0);

/\*All operator with having clause\*/

SELECT customer\_id, COUNT(\*)

FROM rental

GROUP BY customer\_id

HAVING COUNT(\*) > ALL (SELECT COUNT(\*) FROM rental AS r INNER JOIN customer AS c ON r.customer\_id = c.customer\_id INNER JOIN address AS a ON c.address\_id = a.address\_id INNER JOIN city AS ct ON a.city\_id = ct.city\_id INNER JOIN country AS co ON ct.country\_id = co.country\_id WHERE co.country IN ('United States', 'Mexico', 'Canada') GROUP BY r.customer\_id);

SELECT c.customer\_id, COUNT(\*)

FROM rental AS r

INNER JOIN customer AS c ON r.customer\_id = c.customer\_id

INNER JOIN address AS a ON c.address\_id = a.address\_id

INNER JOIN city AS ct ON a.city\_id = ct.city\_id

INNER JOIN country AS co ON ct.country\_id = co.country\_id

WHERE co.country IN ('United States', 'Mexico', 'Canada')

GROUP BY r.customer\_id;

/\*The any operator Like the all operator, the any operator allows a value to be compared to the members of a set of values; unlike all, however, a condition using the any operator evaluates to true as soon as a single comparison is favorable. This Query will find all customers whose total film rental payments exceed the total payments for all customers in Bolivia, Paraguay, or Chile:\*/

SELECT customer\_id, SUM(amount)

FROM payment

GROUP BY customer\_id

HAVING SUM(amount) > ANY (SELECT SUM(p.amount) FROM payment AS p INNER JOIN customer AS c ON p.customer\_id = c.customer\_id INNER JOIN address AS a ON c.address\_id = a.address\_id INNER JOIN city AS ct ON a.city\_id = ct.city\_id INNER JOIN country AS co ON ct.country\_id = co.country\_id WHERE co.country IN ('Bolivia', 'Paraguay', 'Chile') GROUP BY co.country);

SELECT SUM(p.amount)

FROM payment AS p

INNER JOIN customer AS c ON p.customer\_id = c.customer\_id

INNER JOIN address AS a ON c.address\_id = a.address\_id

INNER JOIN city AS ct ON a.city\_id = ct.city\_id

INNER JOIN country AS co ON ct.country\_id = co.country\_id

WHERE co.country IN ('Bolivia', 'Paraguay', 'Chile')

GROUP BY co.country;

/\*Multicolumn Subqueries\*/

SELECT fa.actor\_id, fa.film\_id

FROM film\_actor fa

WHERE fa.actor\_id IN (SELECT actor\_id FROM actor WHERE last\_name = 'MONROE')

AND fa.film\_id IN (SELECT film\_id FROM film WHERE rating = 'PG');

SELECT actor\_id, first\_name, last\_name

FROM actor

WHERE last\_name = 'monroe';

/\*However, you could merge the two single-column subqueries into one multicolumn subquery and compare the results to two columns in the film\_actor table.\*/

SELECT actor\_id, film\_id

FROM film\_actor

WHERE (actor\_id, film\_id) IN (SELECT a.actor\_id, f.film\_id FROM actor a CROSS JOIN film f WHERE a.last\_name = 'MONROE' AND f.rating = 'PG');

/\*Correlated Subqueries All of the subqueries shown thus far have been independent of their containing statements, meaning that you can execute them by themselves and inspect the results. A correlated subquery, on the other hand, is dependent on its containing statement from which it references one or more columns. Unlike a noncorrelated subquery, a correlated subquery is not executed once prior to execution of the containing statement; instead, the correlated subquery is executed once for each candidate row (rows that might be included in the final results).\*/

SELECT c.first\_name, c.last\_name

FROM customer AS c

WHERE 20 = (SELECT COUNT(\*) FROM rental r WHERE r.customer\_id = c.customer\_id);

SELECT c.first\_name, COUNT(\*)

FROM rental AS r

INNER JOIN customer AS c USING (customer\_id)

GROUP BY first\_name

HAVING COUNT(\*) = 20;

SELECT c.first\_name, c.last\_name

FROM customer AS c

WHERE (SELECT SUM(p.amount) FROM payment AS p WHERE p.customer\_id = c.customer\_id) BETWEEN 180 AND 240;

/\*The exists Operator the most common operator used to build conditions that utilize correlated subqueries is the exists operator. You use the exists operator when you want to identify that a relationship exists without regard for the quantity\*/

SELECT c.first\_name, c.last\_name

FROM customer c

WHERE EXISTS( SELECT 100 FROM rental r WHERE r.customer\_id = c.customer\_id AND DATE(r.rental\_date) < '2005-05-25');

/\*Using the exists operator, your subquery can return zero, one, or many rows, and the condition simply checks whether the subquery returned one or more rows. If you look at the select clause of the subquery, you will see that it consists of a single literal (1); since the condition in the containing query only needs to know how many rows have been returned, the actual data the subquery returned is irrelevant. Your subquery can return whatever strikes your fancy\*/

SELECT c.first\_name, c.last\_name

FROM customer c

WHERE EXISTS( SELECT r.rental\_date, r.customer\_id, 'ABCD' str, 2 \* 3 / 7 nmbr FROM rental r WHERE r.customer\_id = c.customer\_id AND DATE(r.rental\_date) < '2005-05-25');

/\*However, the convention is to specify either select 1 or select \* when using exists.\*/

SELECT a.first\_name, a.last\_name

FROM actor a

WHERE NOT EXISTS( SELECT 1 FROM film\_actor fa INNER JOIN film f ON f.film\_id = fa.film\_id WHERE fa.actor\_id = a.actor\_id AND f.rating = 'R');